Learning & Planning

Lecture 1

Subbarao Kambhampati

Arizona State University

http://rakaposhi.eas.asu.edu/ml-summer.html

Lectures at Machine Learning Summer School, Canberra, 2006

Acknowledgements

- Suresh Katukam, Yong Qu, Laurie Ihrig—whose graduate theses got me into learning & planning
- Terry Lyle Zimmerman, who kept my interest in learning & planning alive through the AI Magazine Article
  - And who is not a little miffed that he is not here in Sunny DownUnder giving these lectures
- The GILA group for resurrecting my interest in Learning & Planning
- Current group members, including Will Cushing, for comments on these slides
- Manuela Veloso & Daniel Borrajo for access to their Learning & Planning tutorial
- Alan Fern, Sunwook Yoon and Robert Givan for clarifications/discussions
- National Science Foundation, NASA, DARPA, AFOSR, ONR and IBM for support over the years

Aims of the Lectures

- To give you a feel for the exciting work that has been and is being done in the intersection of planning & learning over last 20 years
- This will not be a comprehensive survey but a (possibly biased) selection of topics
- For comprehensive survey see:
  - Zimmerman & Kambhampati, AI Magazine 2003
  - Also see http://rakaposhi.eas.asu.edu/ml-summer.html for additional resources
    - Including final updated slides + voice recording

Two ways to view these lectures

- As an application area for Machine Learning: learning techniques in planning
  - To improve planning performance
  - To learn domain dynamics
  - To learn strategies
- To do that, you will need to know a bit about what is planning and its current state of the art
  - I will focus more on deterministic planning (since that is where most work to-date has been done)

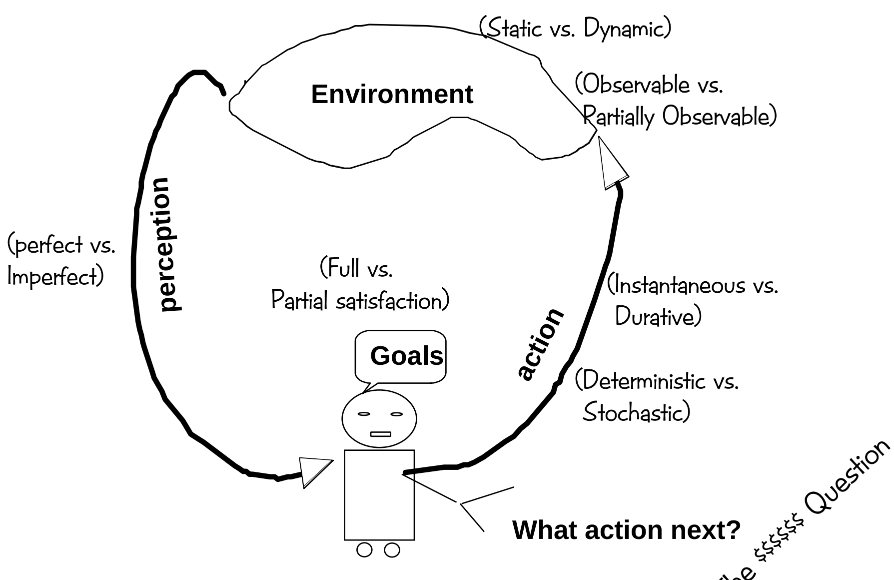
- As an opportunity to motivate/learn about interesting learning techniques
- Specifically, most learning techniques used in planning are:
  - Unabashedly "Knowledge-based"
    - In contrast most techniques you heard in MLSS start *tabula rasa*
    - ..or smuggle background knowledge through the *kernel* backdoors ☺
  - Often relational
    - In contrast to several of the MLSS lectures that talk about attribute-oriented learning

My Background

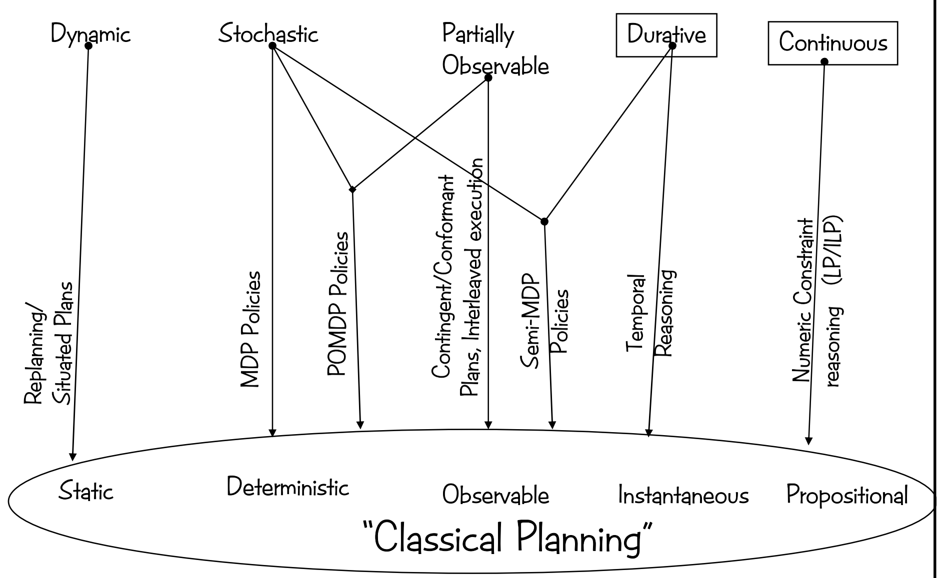
Primary interest: Automated Planning

- I do a lot of work on plan synthesis..
  - Most recently in scaling up plan synthesis algorithms using reachability heuristics
- Have—in the past—done a lot of work on learning in the context of planning
  - When that was the best method to scale-up planner performance
    - Search control rule learning; case-based planning
- Have co-authored a survey of the work in planning and learning (AI Mag, 2003)
- Am currently freshly interested in learning to do planning in domains with partially complete domain knowledge
  - Especially, when examples of successful plans are available

What's all this Learning in aid of?



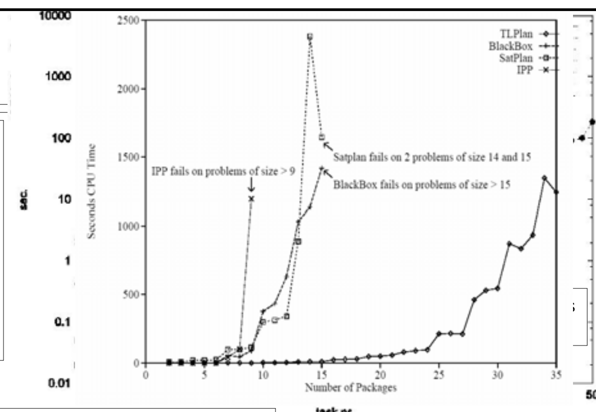
Subbarao Kambhampati



Subbarao Kambhampati

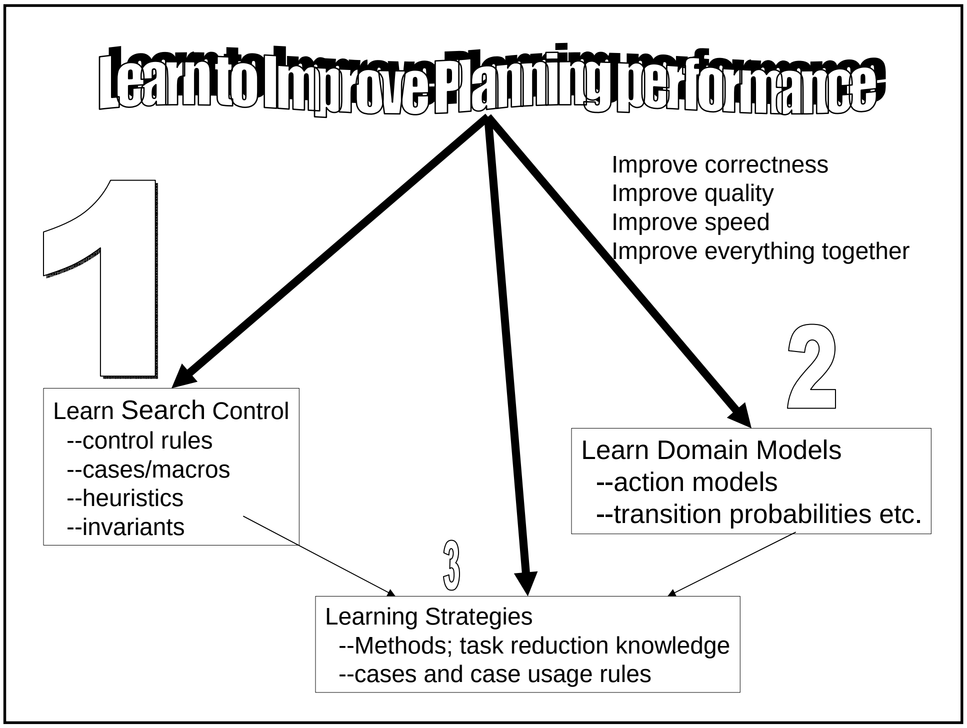
History of Learning in Planning

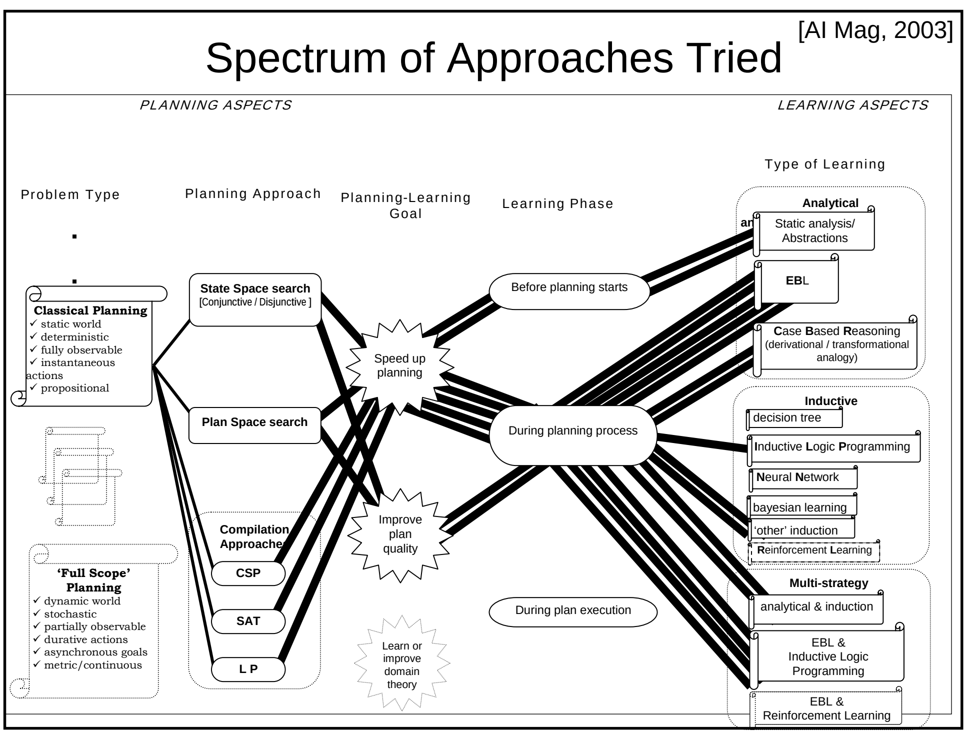
- Pre-1995 planning algorithms could synthesize about 6 – 10 action plans in minutes
- ➔ Massive dependence on speedup learning techniques
  - ➔ Golden age for Speedup Learning in Planning ☺



- Significant scale-up in the last 6-7 years mostly through powerful reachability heuristics
  - Now, we can synthesize 100 action plans in seconds.
  - Reduced interest in learning as a crutch

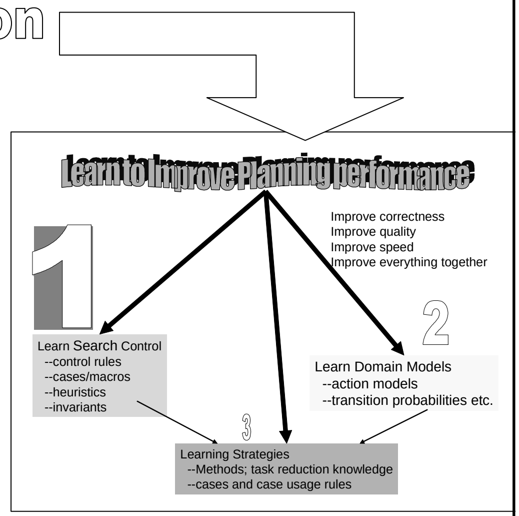
But KBPlanners (customized by humans) did even better
opening up renewed interest in learning the kinds of knowledge humans are are able to put in
..as well as interest in learning domain models and strategies
Now that we have fast planners, the domain modeling comes to the foreground


Learn to Improve Planning performance
Improve correctness
Improve quality
Improve speed
Improve everything together
1
2
3
Learn Search Control
--control rules
--cases/macros
--heuristics
--invariants
Learn Domain Models
--action models
--transition probabilities etc.
Learning Strategies
--Methods; task reduction knowledge
--cases and case usage rules


[AI Mag, 2003]
Spectrum of Approaches Tried
PLANNING ASPECTS
LEARNING ASPECTS
Type of Learning
Problem Type
Planning Approach
Planning-Learning Goal
Learning Phase
Classical Planning
✓ static world
✓ deterministic
✓ fully observable
✓ instantaneous actions
✓ propositional
'Full Scope' Planning
✓ dynamic world
✓ stochastic
✓ partially observable
✓ durative actions
✓ asynchronous goals
✓ metric/continuous
State Space search
[Conjunctive / Disjunctive]
Plan Space search
Compilation Approaches
CSP
SAT
L P
Speed up planning
Improve plan quality
Learn or improve domain theory
Before planning starts
During planning process
During plan execution
Analytical
Static analysis/ Abstractions
EBL
Case Based Reasoning
(derivational / transformational analogy)
Inductive
decision tree
Inductive Logic Programming
Neural Network
bayesian learning
'other' induction
Reinforcement Learning
Multi-strategy
analytical & induction
EBL & Inductive Logic Programming
EBL & Reinforcement Learning

Overview

Brief Introduction to Planning

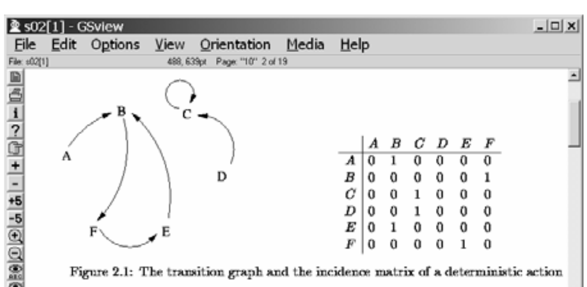


Transition System Models

A transition system is a two tuple <S, A>
Where
  S is a set of "states"
  A is a set of "transitions"
    each transition a is a subset of SXS
    --If a is a (partial) function then deterministic transition
    --otherwise, it is a "non-deterministic" transition
      --It is a stochastic transition
        If there are probabilities associated with each state a takes s to
    --Finding plans becomes is equivalent to finding "paths" in the transition system

Each action in this model can be Represented by incidence matrices (e.g. below)

The set of all possible transitions Will then simply be the SUM of the Individual incidence matrices
Transitions entailed by a sequence of actions will be given by the (matrix) multiplication of the incidence matrices



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	0	0	0	1
C	0	0	1	0	0	0
D	0	0	1	0	0	0
E	0	1	0	0	0	0
F	0	0	0	0	1	0

Figure 2.1: The transition graph and the incidence matrix of a deterministic action

Transition system models are called "Explicit state-space" models
  In general, we would like to represent the transition systems more compactly
    e.g. State variable representation of states.
  These latter are called "Factored" models

These were discussed orally but were not shown in the class

State Variable (Factored) Models

- Planning systems tend to use factored models (rather than direct transition models)
  - World is made up of states which are defined in terms of state variables
    - Can be boolean (or multi-ary or continuous)
  - States are *complete assignments over state variables*
    - So, k boolean state variables can represent how many states?
  - Actions change the values of the state variables
    - Applicability conditions of actions are also specified in terms of partial assignments over state variables

Blocks world

State variables:
Ontable(x) On(x,y) Clear(x) hand-empty holding(x)

Init:
Ontable(A),Ontable(B),
Clear(A), Clear(B), hand-empty

Goal:
~clear(B), hand-empty

Initial state:
Complete specification of T/F values to state variables
--By convention, variables with F values are omitted

Goal state:
A partial specification of the desired state variable/value combinations
--desired values can be both positive and negative

Pickup(x)
Prec: hand-empty,clear(x),ontable(x)
eff: holding(x),~ontable(x),~hand-empty,~Clear(x)

Putdown(x)
Prec: holding(x)
eff: Ontable(x), hand-empty,clear(x),~holding(x)

Stack(x,y)
Prec: holding(x), clear(y)
eff: on(x,y), ~cl(y), ~holding(x), hand-empty

Unstack(x,y)
Prec: on(x,y),hand-empty,cl(x)
eff: holding(x),~clear(x),clear(y),~hand-empty

All the actions here have only positive preconditions; but this is not necessary

PDDL Standard

```
(:action pick-up
        :parameters (?ob1)
        :precondition (and (clear ?ob1)
                           (on-table ?ob1)
                           (arm-empty)
                           (block ?ob1))
        :effect
        (and (not (on-table ?ob1))
             (not (clear ?ob1))
             (not (arm-empty))
             (holding ?ob1)))
```

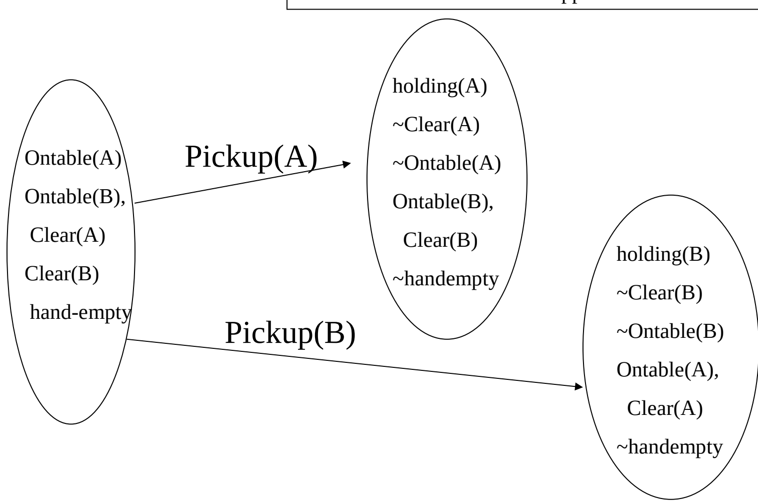
PDDL standard under continual extension
(02) Support for time/durative actions
(04) Support for stochastic outcomes
(06) Support for soft constraints /preferences

```
(define (problem sussman-anomaly)
  (:domain blocks-world)
  (:objects A B C)
  (:init (block a) (block b) (block c)
   (on-table a) (on-table b) (on c a)
   (clear b) (clear c) (arm-empty))
  (:goal (and (on a b) (on b c))))
```

Subbarao Kambhampati

Progression:

An action A can be applied to state S iff the preconditions are satisfied in the current state
The resulting state S' is computed as follows:
--every variable that occurs in the actions effects gets the value that the action said it should have
--every other variable gets the value it had in the state S where the action is applied

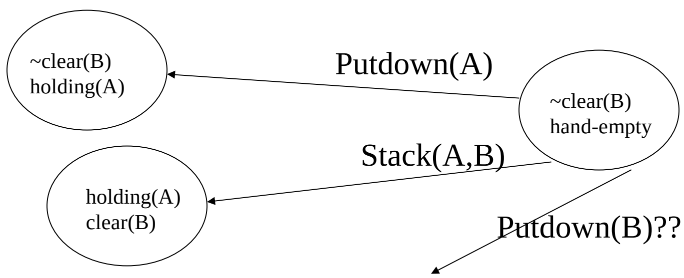


Regression:

A state S can be regressed over an action A
(or A is applied in the backward direction to S)
Iff:
--There is no variable v such that v is given different values by the effects of A and the state S
--There is at least one variable v' such that v' is given the same value by the effects of A as well as state S
The resulting state S' is computed as follows:
-- every variable that occurs in S, *and does not occur* in the effects of A will be copied over to S' with its value as in S
-- every variable that occurs in the precondition list of A will be copied over to S' with the value it has in in the precondition list

Termination test:
Stop when the state s' is *entailed by* the initial state s_I

**Same entailment dir as before..*

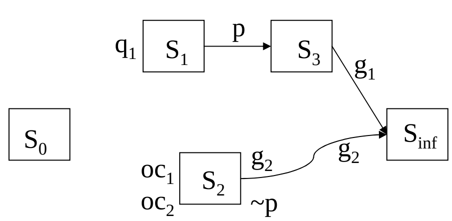


POP Algorithm

1. *Plan Selection: Select a plan P from the search queue*
2. *Flaw Selection*: Choose a flaw f (open cond or unsafe link)
3. *Flaw resolution*:
   If f is an open condition,
   choose an action *S* that achieves *f*
   If f is an unsafe link,
   choose promotion or demotion
   Update P
   Return NULL if no resolution exist
4. *If* there is no flaw left, *return* P

1. Initial plan:

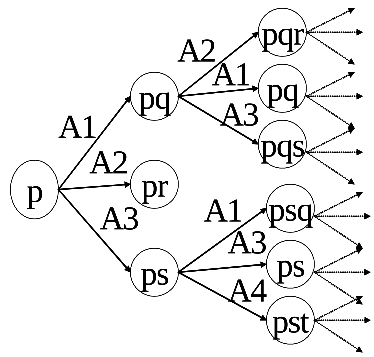
2. Plan refinement (flaw selection and resolution):



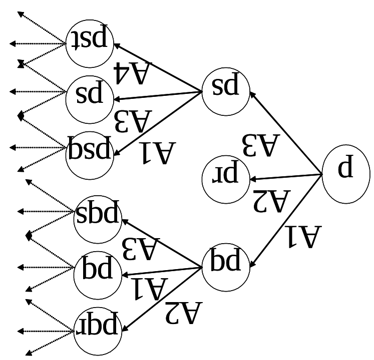
Choice points
- Flaw selection *(open condition? unsafe link? Non-backtrack choice)*
- Flaw resolution/Plan Selection *(how to select (rank) partial plan?)*

Search & Control

Progression Search

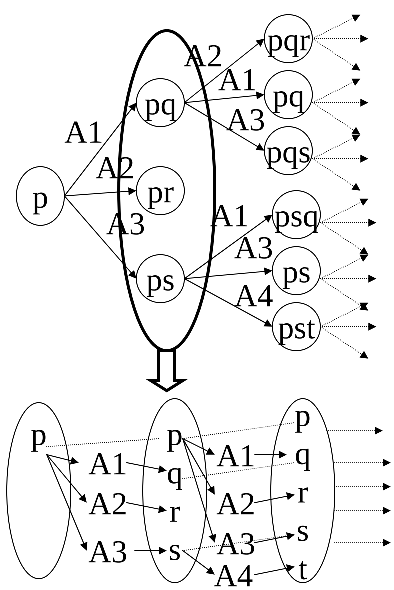


Regression Search



Which branch should we expand?
..depends on which branch is
leading (closer) to the goal

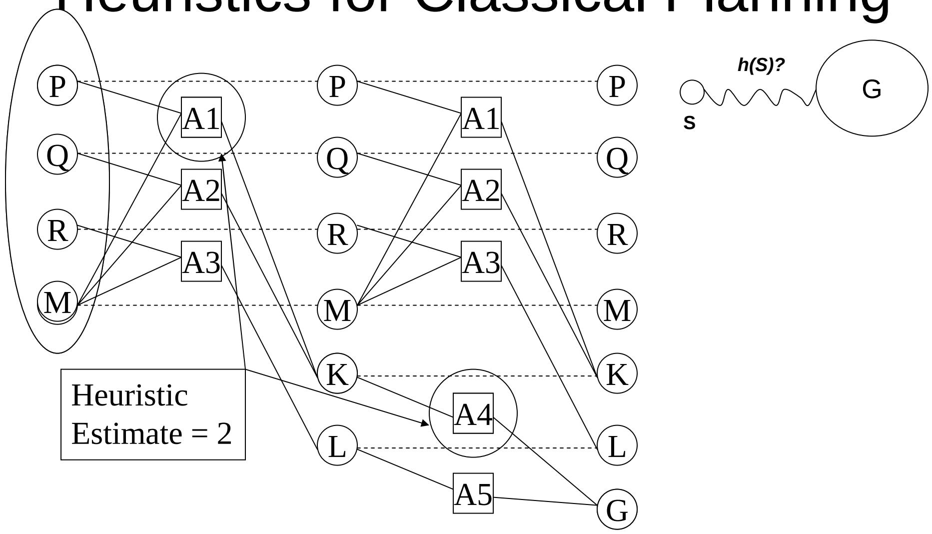
Planning Graph Basics



- Envelope of Progression Tree (Relaxed Progression)
  - Linear vs. Exponential Growth
- Reachable states correspond to subsets of proposition lists
  - BUT not all subsets are states
- Can be used for estimating non-reachability
  - If a state S is not a subset of k^{th} level prop list, then it is definitely not reachable in k steps

[ECP, 1997]

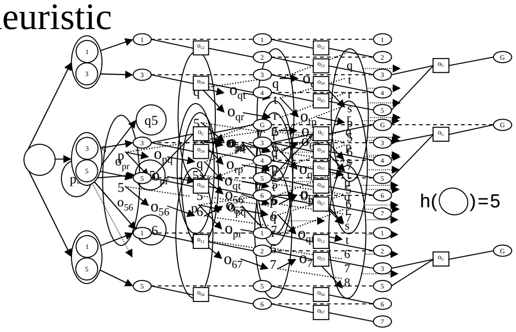
Heuristics for Classical Planning



Relaxed plans are solutions for a relaxed problem

Planning Graphs for heuristics

- Construct planning graph(s) at each search node
  - Extract relaxed plan to achieve goal for heuristic



Learn to Improve Planning performance

Improve correctness
Improve quality
Improve speed
Improve everything together

1

Learn Search Control
--control rules
--cases/macros
--heuristics
--invariants

2

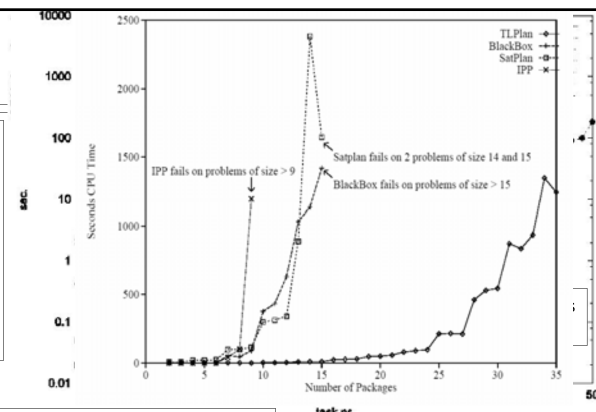
Learn Domain Models
--action models
--transition probabilities etc.

3

Learning Strategies
--Methods; task reduction knowledge
--cases and case usage rules

History of Learning in Planning

- Pre-1995 planning algorithms could synthesize about 6 – 10 action plans in minutes
- ➔ Massive dependence on speedup learning techniques
  - ➔ Golden age for Speedup Learning in Planning ☺



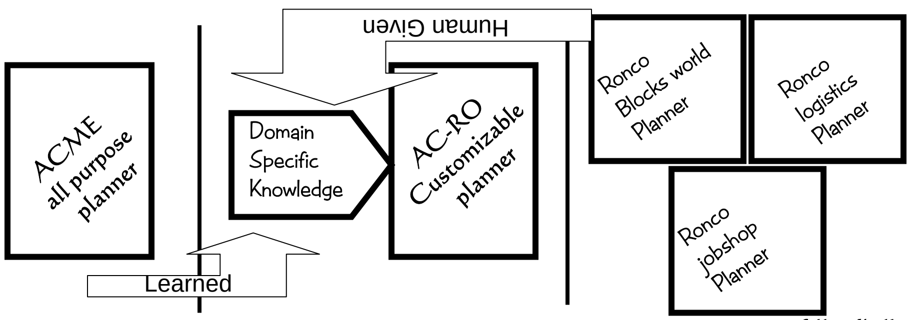
- Significant scale-up in the last 6-7 years mostly through powerful reachability heuristics
  - Now, we can synthesize 100 action plans in seconds.
  - Reduced interest in learning as a crutch

But KBPlanners (customized by humans) did even better
opening up renewed interest in learning the kinds of knowledge humans are are able to put in
..as well as interest in learning domain models and strategies
Now that we have fast planners, the domain modeling comes to the foreground

Planner Customization (using domain-specific Knowledge)

- Domain independent planners tend to miss the regularities in the domain
- Domain specific planners have to be built from scratch for every domain

An *"Any-Expertise"* Solution: Try adding domain specific control knowledge to the domain-independent planners



How is the Customization Done?

- **Given by humans (often, they are quite willing!)[IPC KBPlanning Track]**
  - **As declarative rules (HTN Schemas, Tlplan rules)**
    - » **Don't need to know how the planner works..**
    - » **Tend to be hard rules rather than soft preferences…**
    - » Whether or not a specific form of knowledge can be exploited by a planner depends on the type of knowledge and the type of planner
  - **As procedures (SHOP)**
    - » **Direct the planner's search alternative by alternative..**

- **Through Machine Learning**
  - **Learning Search Control rules**
    - **UCPOP+EBL,**
    - **PRODIGY+EBL, (Graphplan+EBL)**
  - **Case-based planning (plan reuse)**
    - **DerSNLP, Prodigy/Analogy**
  - **Learning/Adjusting heuristics**
  - **Domain pre-processing**
    - » **Invariant detection; Relevance detection;**
      Choice elimination, Type analysis
      - **STAN/TIM, DISCOPLAN etc.**
      - **RIFO; ONLP**
  - **Abstraction**
    - **ALPINE; ABSTRIPS, STAN/TIM etc.**

We will start with KB-Planning track to get a feel for what control knowledge has been found to be most useful; and see how to get it..

Types of Guidance

- Declarative knowledge about desirable or undesirable solutions and partial solutions (SATPLAN+DOM; Cutting Planes)
- Declarative knowledge about desirable/undesirable search paths (TLPlan & TALPlan)
- A declarative grammar of desirable solutions (HTN)

(largely) independent of the details of the specific planner
[affinities do exist between specific types of guidance and planners)

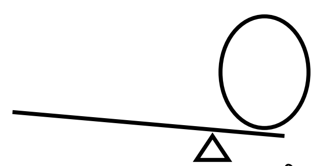
Planner specific. Expert needs to understand the specific details of the planner's search space

- Procedural knowledge about how the search for the solution should be organized (SHOP)
- Search control rules for guiding choice points in the planner's search (NASA RAX; UCPOP+EBL; PRODIGY)
- Cases and rules about their applicability

Subbarao Kambhampati

With right domain knowledge any level of performance can be achieved...

- HTN-SAT, SATPLAN+DOM beat SATPLAN…
  - Expect reduction schemas, declarative knowledge about inoptimal plans
- TLPLAN beats SATPLAN, GRAPHPLAN
  - But expects quite detailed domain knowledge on expected state sequences
- SHOP beats TLPLAN…(but not TALPlan)
  - Expects user to write a "program" for the domain in its language
    - » Explicit instructions on the order in which schemas are considered and concatenated



Who gets the credit?
-The provider of domain knowledge?
-The planner that is capable of using it?

Subbarao Kambhampati

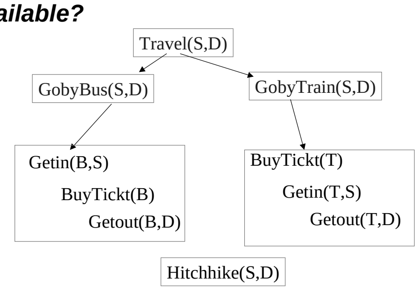
Ways of using the Domain Knowledge

- As search control
  - HTN schemas, TLPlan rules, SHOP procedures
  - *Issues of Efficient Matching*
- To prune unpromising partial solutions
  - HTN schemas, TLPlan rules, SHOP procedures
  - *Issues of maintaining multiple parses*
- As declartative axioms that are used along with other knowledge
  - SATPlan+Domain specific knowledge
  - Cutting Planes (for ILP encodings)
  - *Issues of domain-knowledge driven simplification*
- Folded into the domain-independent algorithm to generate a new domain-customized planner
  - CLAY
  - *Issues of Program synthesis*

Subbarao Kambhampati

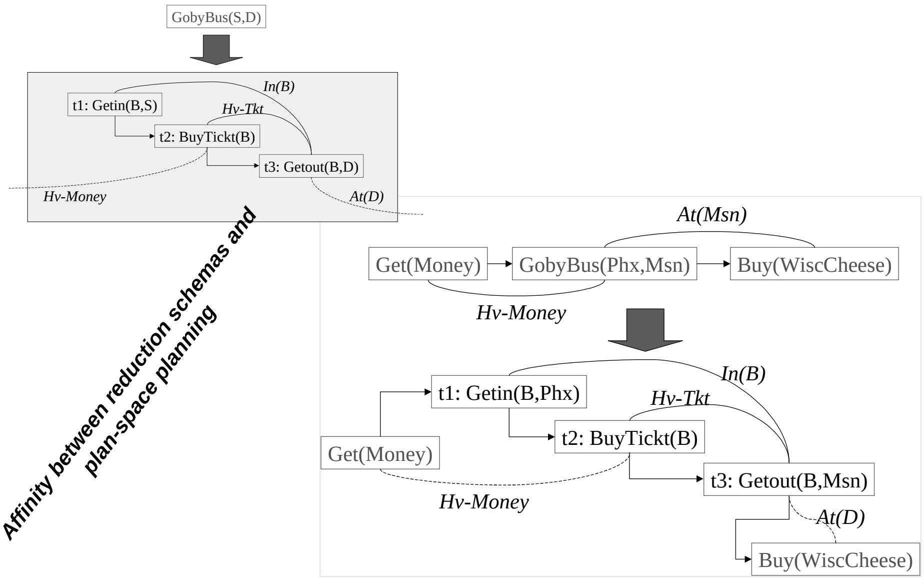
Task Decomposition (HTN) Planning

- The OLDEST approach for providing domain-specific knowledge
  - Most of the fielded applications use HTN planning
- Domain model contains non-primitive actions, and schemas for reducing them
- Reduction schemas are given by the designer
  - Can be seen as encoding user-intent
    - » *Popularity of HTN approaches a testament of ease with which these schemas are available?*
- Two notions of completeness:
  - Schema completeness
    - » (Partial Hierarchicalization)
  - Planner completeness



Subbarao Kambhampati

Modeling Action Reduction



Subbarao Kambhampati

Dual views of HTN planning

- Capturing hierarchical structure of the domain
  - Motivates top-down planning
    - » Start with abstract plans, and reduce them
- Many technical headaches
  - Respecting user-intent, maintaining systematicity and minimality
    [Kambhampati et. al. AAAI-98]
    - » Phantomization, filters, promiscuity, downward-unlinearizability..

- Capturing expert advice about desirable solutions
  - Motivates bottom-up planning
    - » Ensure that each partial plan being considered is "legal" with respect to the reduction schemas
    - » Directly usable with disjunctive planning approaches
    [Mali & Kambhampati, 98]
- Connection to efficiency is not obvious

Relative advantages are still unclear...

[Barrett, 97]

Subbarao Kambhampati

[Nau et. al., 99]

Full procedural control: The SHOP way

Shop provides a "high-level" programming language in which the user can code his/her domain specific planner

-- Similarities to HTN planning
-- Not declarative (?)

The SHOP engine can be seen as an interpreter for this language

```
(:method (travel-to ?y)
   (:first (at ?x)
    (at-taxi-stand ?t ?x)
    (distance ?x ?y ?d)
    (have-taxi-fare ?d))
  `((!hail ?t ?x) (!ride ?t ?x ?y)
    (pay-driver ,(+ 1.50 ?d)))
   ((at ?x) (bus-route ?bus ?x ?y))
  `((!wait-for ?bus ?x)
    (pay-driver 1.00)
    (!ride ?bus ?x ?y)))
```

Travel by bus only if going by taxi doesn't work out

Blurs the domain-specific/domain-independent divide
How often does one have this level of knowledge about a domain?

Subbarao Kambhampati

[Kautz & Selman, AIPS-98]

Non-HTN Declarative Guidance

Invariants: *A truck is at only one location*
at(truck, loc1, I) & loc1 != loc2 => ~at(truck, loc2, I)

Optimality: *Do not return a package to the same location*
at(pkg, loc, I) & ~at(pkg,loc,I+1) & I<J => ~at(pkg,loc,j)

Simplifying: *Once a truck is loaded, it should immediately move*
~in(pkg,truck,I) & in(pkg,truck,I+1) & at(truck, loc, I+1) => ~at(truck, loc, I+2)

Once again, additional clauses first increase the encoding size but make them easier to solve after simplification (unit-propagation etc).

Subbarao Kambhampati

Rules on desirable State Sequences: TLPlan approach

TLPlan [Bacchus & Kabanza, 95/98] controls a forward state-space planner

Rules are written on state sequences using the linear temporal logic (LTL)

LTL is an extension of prop logic with temporal modalities

U	**until**	**[]**	**always**
O	**next**	**<>**	**eventually**

Example:

If you achieve on(B,A), then preserve it until On(C,B) is achieved:

[] (on(B,A) => on(B,A) U on(C,B))

Subbarao Kambhampati

TLPLAN Rules can get quite baroque

Good towers are those that do not violate any goal conditions

$$
\begin{aligned}
&goodtower(x) \triangleq clear(x) \wedge goodtowerbelow(x)\\
&goodtowerbelow(x) \triangleq (ontable(x) \wedge \neg\mathrm{GOAL}(\exists[y{:}on(x,y)] \vee holding(x)))\\
&\quad \vee \exists[y{:}on(x,y)]\ \neg\mathrm{GOAL}(ontable(x) \vee holding(x)) \wedge \neg\mathrm{GOAL}(clear(y))\\
&\qquad \wedge \forall[z{:}\mathrm{GOAL}(on(x,z))]\ z = y \wedge \forall[z{:}\mathrm{GOAL}(on(z,y))]\ z = x\\
&\qquad \wedge goodtowerbelow(y)
\end{aligned}
$$

Keep growing "good" towers, and avoid "bad" towers

$$
\begin{aligned}
&\Box\Big(\forall[x{:}clear(x)]\ goodtower(x) \Rightarrow \bigcirc goodtowerabove(x)\\
&\qquad \wedge badtower(x) \Rightarrow \bigcirc(\neg\exists[y{:}on(y,x)]\ )\\
&\qquad \wedge (ontable(x) \wedge \exists[y{:}\mathrm{GOAL}(on(x,y))]\ \neg goodtower(y))\\
&\qquad\qquad \Rightarrow \bigcirc(\neg holding(x))\Big)
\end{aligned}
$$

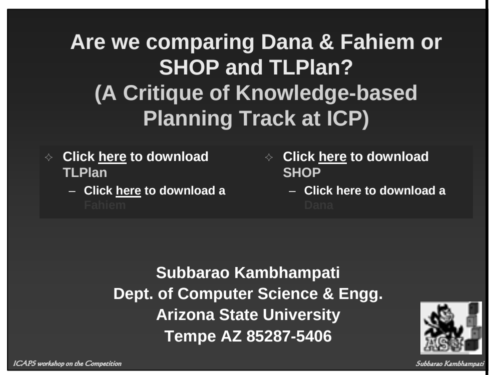
How "Obvious" are these rules? Can these be learned?

The heart of TLPlan is the ability to *incrementally* and *effectively* evaluate the truth of LTL formulas.

Subbarao Kambhampati

What are the lessons of KB Track?

- **If TLPlan did better than SHOP in ICP, then how are we supposed to interpret it?**
  - **That TLPlan is a superior planning technology over SHOP?**
  - **That the naturally available domain knowledge in the competition domains is easier to encode as linear temporal logic statements on state sequences than as procedures in the SHOP language?**
  - **That Fahiem Bacchus and Jonas Kvarnstrom are way better at coming up with domain knowledge for blocks world (and other competition domains) than Dana Nau?**



May be we should "learn" this guidance

How is the Customization Done?

- **Given by humans (often, they are quite willing!)[IPC KBPlanning Track]**
  - **As declarative rules (HTN Schemas, Tlplan rules)**
    - » **Don't need to know how the planner works..**
    - » **Tend to be hard rules rather than soft preferences…**
    - » Whether or not a specific form of knowledge can be exploited by a planner depends on the type of knowledge and the type of planner
  - **As procedures (SHOP)**
    - » **Direct the planner's search alternative by alternative..**

- **Through Machine Learning**
  - **Learning Search Control rules**
    - **UCPOP+EBL,**
    - **PRODIGY+EBL, (Graphplan+EBL)**
  - **Case-based planning (plan reuse)**
    - **DerSNLP, Prodigy/Analogy**
  - **Learning/Adjusting heuristics**
  - **Domain pre-processing**
    - » **Invariant detection; Relevance detection;**
      Choice elimination, Type analysis
      - **STAN/TIM, DISCOPLAN etc.**
      - **RIFO; ONLP**
  - **Abstraction**
    - **ALPINE; ABSTRIPS, STAN/TIM etc.**

We will start with KB-Planning track to get a feel for what control knowledge has been found to be most useful; and see how to get it..

Approaches for Learning Search Control

"speedup learning"

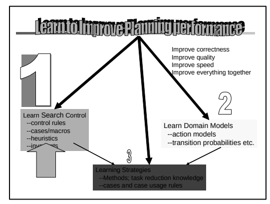
Improve an existing planner

- Learn rules to guide choice points
- Learn plans to reuse
  - --Macros
  - --Annotated cases
- Learn adjustments to heuristics

Learn "from scratch" how to plan

--Learn "reactive policies"
State x Goal→action

[Work by Khadron, 99;
Givan, Fern, Yoon, 2003→]



No "from-scratch" learner ever placed well in the Intl. Planning Competition.
Macro-FF, an extension of a successful planner called FF, placed 1st in 3 domains in IPC-2004 (..but there was no Knowledge-based Planning track in 2004)

Inductive Learning of Search Control

- Convert to "classification" learning
  - +ve examples: Search nodes on the success path
  - -ve examples: Search nodes one step away from the success path
  - Learn a classifier
    - Classifier may depend on the features of the problem (Init, Goal), as well as the current state.
- Several systems:
  - Grasshopper (Leckie & Zuckerman; 1998)
  - Inductive Logic Programming; (Estlin & Mooney; 1993)

Subbarao Kambhampati

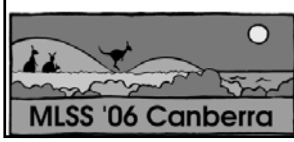
Learning & Planning

Lecture 2

Subbarao Kambhampati

Arizona State University

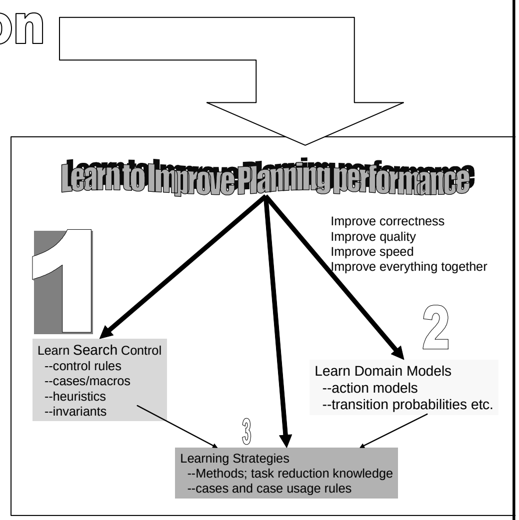
http://rakaposhi.eas.asu.edu/ml-summer.html

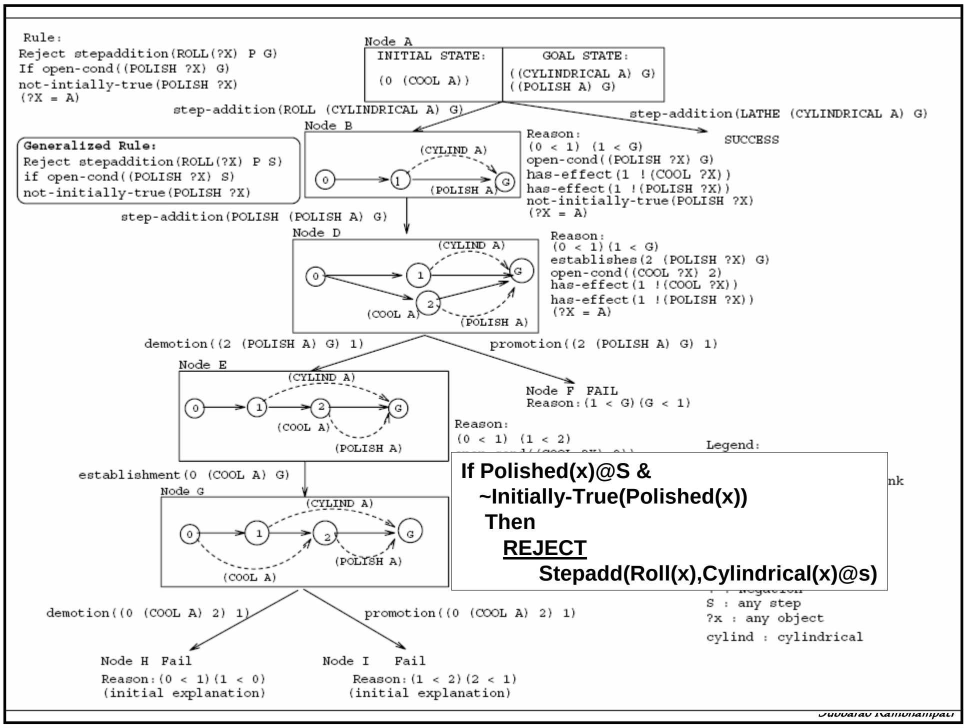


Lectures at Machine Learning Summer School, Canberra, 2006

Overview

Brief Introduction to Planning





Explanation-based Learning

- Start with a labelled example, and some background domain theory
- *Explain*, using the background theory, why the example deserves the label
  - Think of explanation as a way of picking class-relevant features with the help of the background knowledge
- Use the explanation to generalize the example (so you have a general rule to predict the label)
- Used extensively in planning
  - *Given a correct plan for an initial and goal state pair, learn a general plan*
  - *Given a search tree with failing subtrees, learn rules that can predict failures*
  - *Given a stored plan and the situations where it could not be extended, learn rules to predict applicability of the plan*

Learning Search Control Rules with EBL

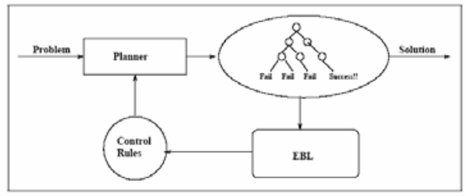
Explain leaf level failures

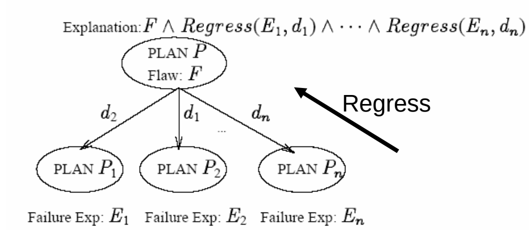
Regress the explanations to compute interior node failure explanations

Use failure explanations to set up control rules

Problems:
- -- Most branches end in depth-limits
  - >No analytical explanation
  - >Use preference rules?
- -- The utility problem
  - >Learn general rules
  - >Keep usage statistics & prune useless rules

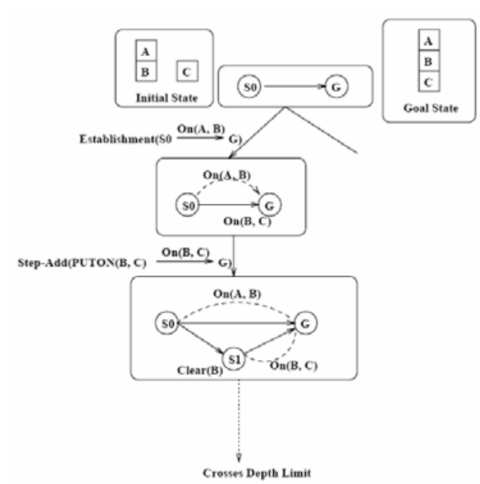
(Kambhampati, Katukam, Qu, 95)



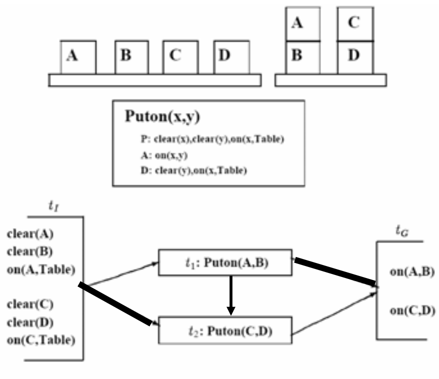


Issues in EBL for Search Control Rules

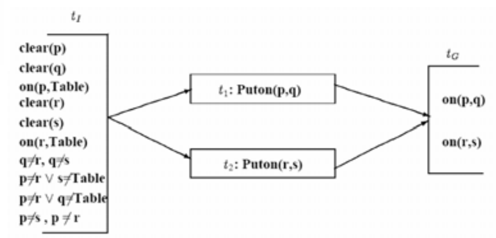
- Effectiveness of learning depends on the explanation
  - Primitive explanations of failure may involve constraints that are directly inconsistent
  - But it would be better if we can unearth hidden inconsistencies
- ..an open issue is to learn with probably incorrect explanations
  - UCPOP+CFEBL



We can also explain (& generalize) Success



Success explanations tend to involve more components of the plan than failure explanations



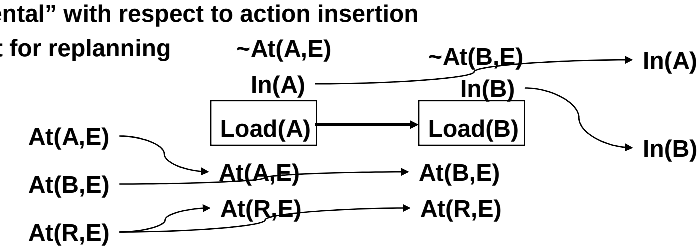
Checking correctness of a plan: The Causal Approach

Contd..

- Causal Proof: Check if each of the goals and preconditions of the action are
  - » "established" : There is a preceding step that gives it
  - » "unclobbered": No possibly intervening step deletes it
    - Or for every preceding step that deletes it, there exists another step that precedes the conditions and follows the deleter adds it back.

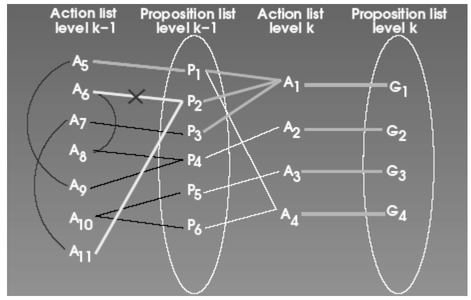
Causal proof is

- "local" (checks correctness one condition at a time)
- "state-less" (does not need to know the states preceding actions)
  - » Easy to extend to *durative* actions
- "incremental" with respect to action insertion
  - » Great for replanning



Status of EBL learning in Planning

- Explanation-based learning from failures has been ported to modern planners
  - GP-EBL [Kambhampati, 2000] ports EBL to Graphplan
    - » "Mutual exclusion relations" are learned
      - (exploits the connection between EBL and "nogood" learning in CSP)
    - » Impressive speed improvements
      - EBL is considered standard part of Graphplan implementation now..
    - » …but much of the learning was ***intra problem***



Subbarao Kambhampati

Some misconceptions about EBL

- Misconception 1: EBL *needs* complete and correct background knowledge
    - » (Confounds "Inductive vs. Analytical" with "Knowledge rich vs. Knowledge poor")
  - *If you have complete and correct knowledge then the learned knowledge will be in the* <u>*deductive closure*</u> of the original knowledge;
  - If not, then the learned knowledge will be tentative (just as in inductive learning)
- Misconception 2: EBL is *competing* with inductive learning
  - In cases where we have weak domain theories, EBL can be seen as a "feature selection" phase for the inductive learner
- Misconception 3: *Utility* problem is endemic to EBL
  - Search control learning of any sort can suffer from utility problem
    - » E.g. Using inductive learning techniques to learn search control

Useful Directions for EBL

- Often we may have background knowledge that is "complete" and "correct" in certain local regions, but not across the board
  - E.g. My knowledge of AI is incomplete/incorrect; but my knowledge of planning is complete and possibly correct, while my knowledge of Machine Learning is incomplete but largely correct
  - Characterizing the limitations of the background knowledge will help in estimating the expected accuracy of the learned knowledge
- Handling partial explanations of failure in search control rule learning
  - UCPOP+CFEBL

Subbarao Kambhampati

Approaches for Learning Search Control

"speedup learning"

Improve an existing planner

Learn "from scratch" how to plan

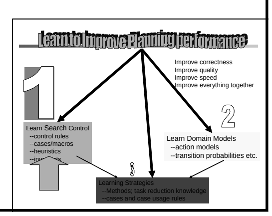
--Learn "reactive policies"
State x Goal→action

[Work by Khadron, 99;
Givan, Fern, Yoon, 2003→]

Learn rules
to guide choice points

Learn plans
to reuse
--Macros
--Annotated cases

Learn adjustments to
heuristics



Case-based Planning
Macrops, Reuse, Replay

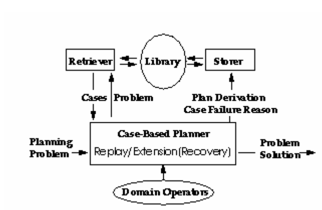
- **Structures being reused**
  - **Opaque vs. Modifiable**
  - **Solution vs. Solving process (derivation/search trace)**
- **Acquisition of structures to be reused**
  - **Human given vs. Automatically acquired**
    - » **Adapting human-given cases is a way of handling partial domain models**
- **Mechanics of reuse**
  - **Phased vs. simultaneous**
- **Costs**
  - **Storage & Retrieval costs; Solution quality**

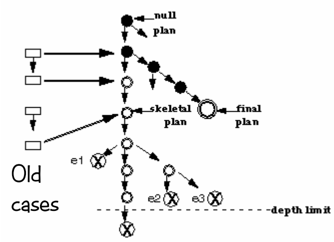
Subbarao Kambhampati

(Ihrig & Kambhampati, JAIR 97)

Case-study: DerSNLP

- **Modifiable derivational traces are reused**
- **Traces are automatically acquired during problem solving**
  - **Analyze the interactions among the parts of a plan, and store plans for *non-interacting* subgoals separately** EBL
    - » **Reduces retrieval cost**
  - **Use of EBL failure analysis to detect interactions**
- **All relevant trace fragments are retrieved and replayed before the control is given to from-scratch planner**
  - **Extension failures are traced to individual replayed traces, and their storage indices are modified appropriately**
    - » **Improves retrieval accuracy**





Subbarao Kambhampati

Reuse/Macrops Current Status

- Since ~1996 there has been little work on reuse and macrop based improvement of base-planners
  - People sort of assumed that the planners are already so fast, they can't probably be improved further
- Macro-FF, a system that learns 2-step macros in the context of FF, posted a respectable performance at IPC 2004 (but NOT in the KB-track)
  - » Uses a sophisticated method assessing utility of the learned macrops (& also benefits from the FF enforced hill-climbing search)
    - Macrops are retained only if they improve performance significantly on a suite of problems
  - Given that there are several theoretical advantages to reuse and replay compared to Macrops, it would certainly be worth seeing how they fare at IPC [Open]

Subbarao Kambhampati

Approaches for Learning Search Control

"speedup learning"

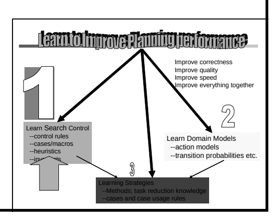
Improve an existing planner

- Learn rules to guide choice points
- Learn plans to reuse
  - --Macros
  - --Annotated cases
- Learn adjustments to heuristics

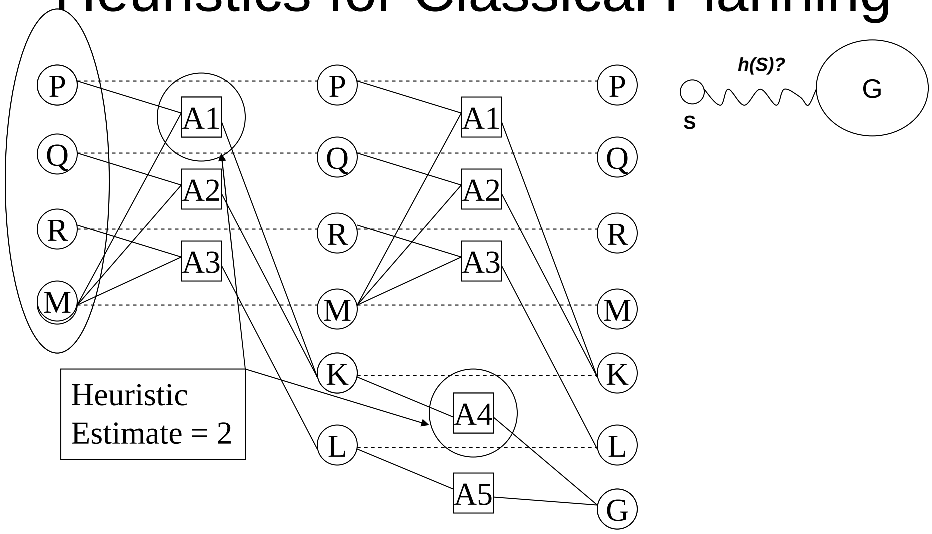
Learn "from scratch" how to plan

--Learn "reactive policies"
State x Goal→action

[Work by Khadron, 99;
Givan, Fern, Yoon, 2003→]

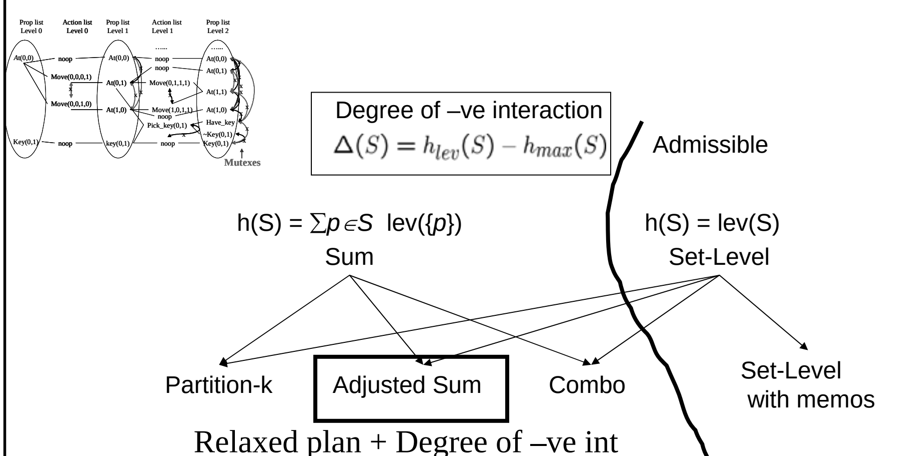


Heuristics for Classical Planning



Relaxed plans are solutions for a relaxed problem

Cost of a Set of Literals



- lev(p) : index of the first level at which p comes into the planning graph
- lev(S): index of the first level where all props in S appear non-mutexed.

Learning to Improve Heuristics

- Most modern planners use reachability heuristics
  - These approximate the reachability by ignoring some types of interactions (usually, negative interactions between subgoals)
  - While effective in general, ignoring such negative interactions can worsen the heuristic guidance and lead the planners astray
    - » A way out is to "adjust" the reachability information with the information about interactions that were ignored
  1. (Static) Adjusted Sum heuristics as popularized in AltAlt
     - Increases the heuristic cost (as we need to propagate negative interactions)
     - Could be bad for progression planners which grow the planning graph once for each node..
  2. (Learn dynamically) Learn to predict the difference between the heuristic estimate by the relaxed plan and the "true" distance
     - Soongwook et. Al. show that this is feasible—and manage to improve the performance of FF

[Soongwook et a, ICAPS 2006]

Subbarao Kambhampati

Learning Adjustments to Heuristics

- Start with a set of training examples [Problem, Plan]
  - Use a standard planner, such as FF to generate these
- For each example [(I,G), Plan]
  - For each state S on the plan
    - » Compute the relaxed plan heuristic S^R
    - » Measure the actual distance of S from goal S* (easy since we have the current plan—assuming it is optimal)
  - Inductive learning problem
    - » Training examples: Features of the relaxed plan of S
      - Yoon et al use a taxonomic feature representation Class labels: S^*-S^R (adjustment)
    - » Learn the classifier
      - Finite linear combination of features of the relaxed plan

[Yoon et al, ICAPS 2006]

Subbarao Kambhampati

What types of control knowledge used by KBplanners can be learned currently?

- **Learnable (now)**
  - **Invariants**
  - **Simple search control rules**
  - **Adjustment rules for heuristics**
- **Open problems**
  - **Complex state sequence rules used by TLPlan**
    - » **Invariants, a special case, is learnable**
  - **HTN schemas (or SHOP programs)**
    - » **Most current work still *starts with* human-given methods, and learns conditions under which they can be applied**

Subbarao Kambhampati

Approaches for Learning Search Control

"speedup learning"

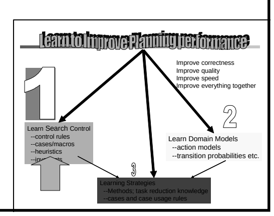
Improve an existing planner

- Learn rules to guide choice points
- Learn plans to reuse
  - --Macros
  - --Annotated cases
- Learn adjustments to heuristics

Learn "from scratch" how to plan

--Learn "reactive policies"
State x Goal→action

[Work by Khadron, 99;
Winner & Veloso, 2002;
Givan, Fern, Yoon, 2003→
Gretton & Thiebaux, 2004]

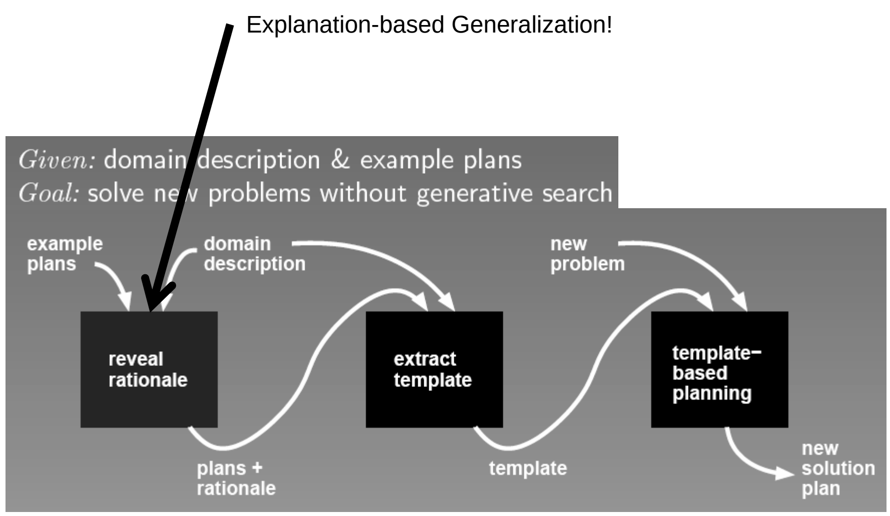


Learning From Scratch: Learning Reactive Policies

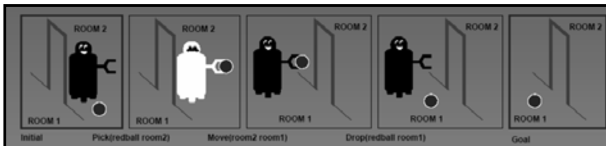
- **We are not interested in speeding up an existing planner, but rather directly learn how to find plans for a large distribution of problems**
- **Specifically, we want to learn a policy of the form [state, goals]→action**
  - **Notice that this is a policy predicated both on states and goals (so it is supposed to work for all problems)**
- **Examples:**
  - Winner & Veloso, 2002 (EBL/Induction)
  - Khadron, 99 (Also, Martin/Geffner; 2002) (Induction)
  - Givan, Fern, Yoon, 2003 (also Gretton & Thiebaux, 2004) (Induction; Reinforcement Learning)

Subbarao Kambhampati

Learning Domain Specific Planners



Subbarao Kambhampati



if (*inGoalState* (at(?3:ball ?1:room)) *and*
inCurrentState (at(?3:ball ?2:room)) *and*
inCurrentState (at-robby(?2:room)) *and*
inCurrentState (free-arm)) *then*
Pick(?3 ?2)
Move(?2 ?1)
Drop(?3 ?1)

if (*inGoalState* (at(?3:ball ?1:room)) *and*
inCurrentState (at(?3:ball ?2:room)) *and*
inCurrentState (at-robby(?1:room))) *then*
Move(?1 ?2)
if (*inGoalState* (at(?3:ball ?1:room)) *and*
inCurrentState (at(?3:ball ?2:room)) *and*
inCurrentState (at-robby(?2:room)) *and*
inCurrentState (free-arm)) *then*
Pick(?3 ?2)
Move(?2 ?1)
Drop(?3 ?1)

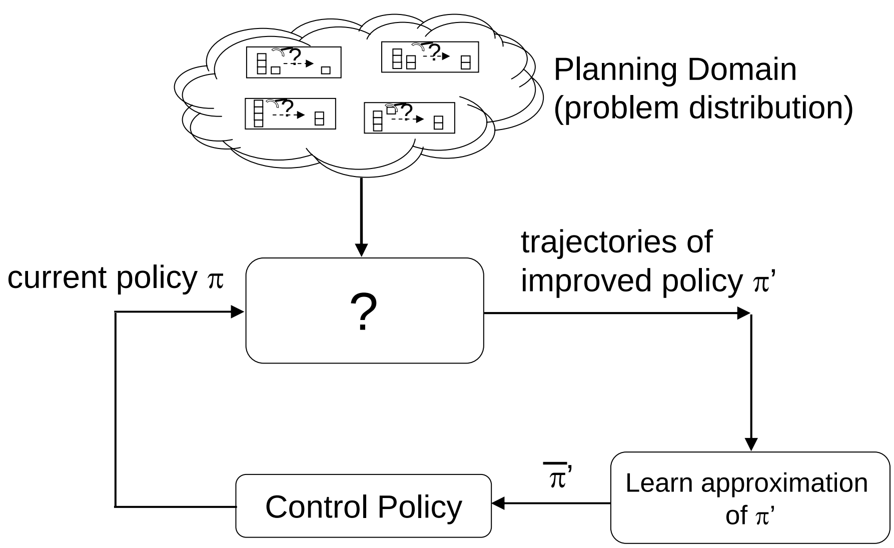
if (*inGoalState* (at(?3:ball ?1:room)) *and*
inCurrentState (at(?3:ball ?2:room)) *and*
inCurrentState (at-robby(?1:room)) *and*
inCurrentState (free-arm)) *then*
Move(?1 ?2)
Pick(?3 ?2)
Move(?2 ?1)
Drop(?3 ?1)

Learning Policies

A policy can be seen as a classifier, we can learn it…

- **If we have access to <u>*n*</u> traces (trajectories) of the <u>optimal policy</u>, we can use standard inductive learning techniques to learn the full policy**
  - **Khadron, 99 learns decision lists**
- **Even if we don't have access to traces from an optimal policy, we can use reinforcement learning techniques to <u>*improve*</u> an existing policy iteratively [Fern, Yoon, Givan]**
  - **Start with an initial policy**
  - **[Policy Iteration:] Use policy rollout to compute n trajectories of the improved policy**
  - **[Policy Learning:] Learn the representation of the new policy using the trajectories**
  - **Repeat**
    - » **A planner using this idea came 3nd in the IPC probabilistic planning competion (it was beaten by FF-replan…)**

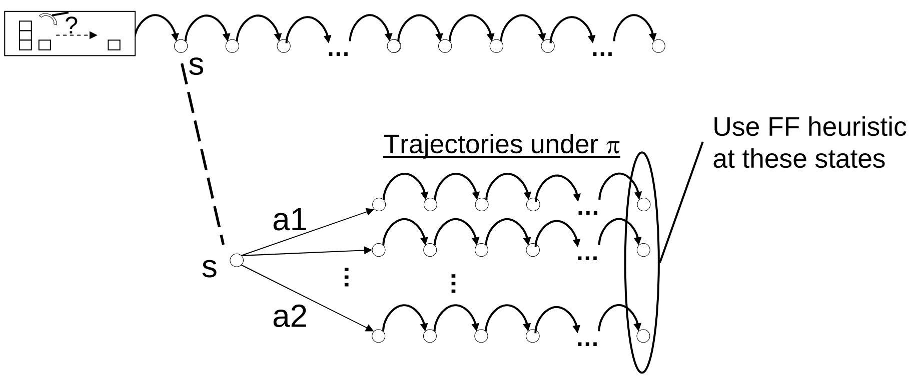
Approximate Policy Iteration



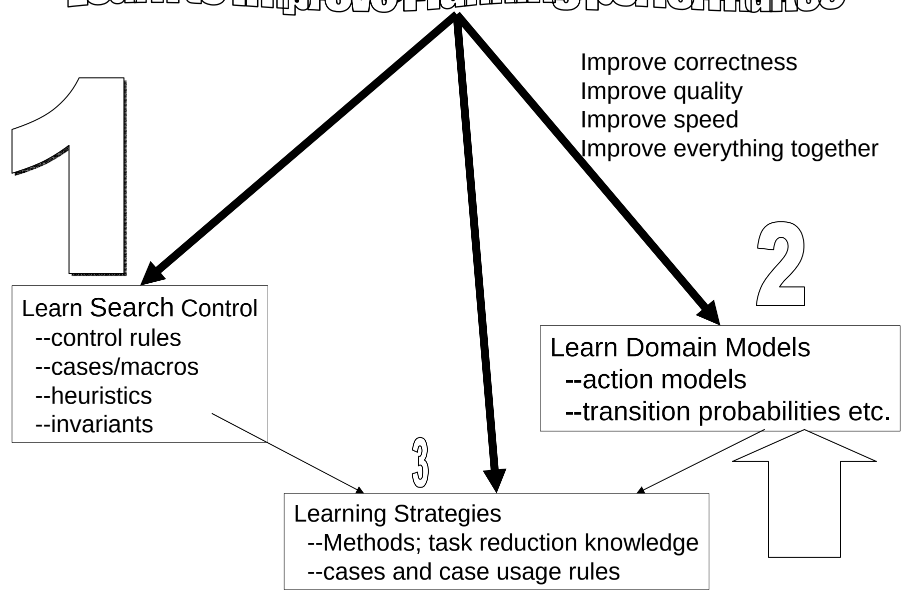
Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'



Learn to Improve Planning performance



Learning Domain Knowledge

Learning from scratch

→Operator Learning

Operationalizing existing knowledge

→EBL-based operationalization [Levine/DeJong; 2006]

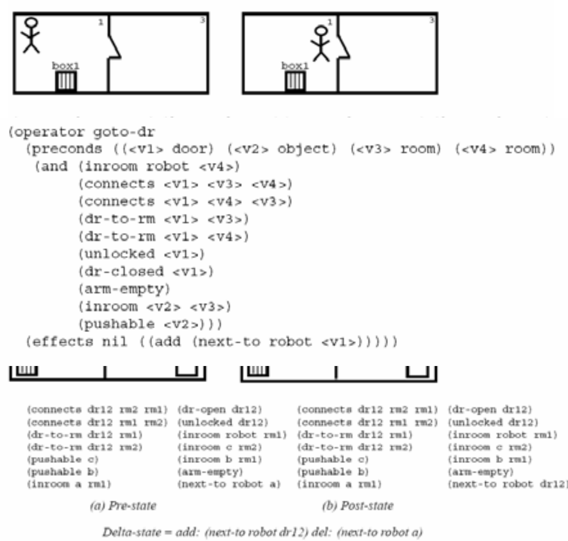
→RL for focusing on "interesting parts" of the model
…lots of people including [Aberdeen et. Al. 06]

Learning Domain Knowledge (From observation)

- Learning Operators (Action Models)
  - Given a set of [Problem; Plan: (operator sequence)] examples; and the space of domain predicates (fluents)
  - Induce operator descriptions
    - Operators will have more parameters in expressive domains
      - Durations and time points; probabilities of outcomes etc.
- Dimensions of variation
  - Availability of intermediate states (Complete or Partial)
    - Makes the problem easy—since we can learn each action separately. Unrealistic (especially "complete" states)
  - Availability of partial action models
    - Makes the problem easier by biasing the hypotheses (we can partially explain the correctness of the plans). Reasonably realistic.
  - Interactive learning in the presence of humans
    - Makes it easy for the human in the loop to quickly steer the system from patently wrong models

OBSERVE: Assumes full knowledge of intermediate states

- When we have full knowledge of intermediate states, the action effects are easy to learn!
  - Preconditions need to be learned by inducing over a sample of states where the action is applicable
  - Zettlemeyer et al dc this for stochastic case [2005]



[Wang, 1994]

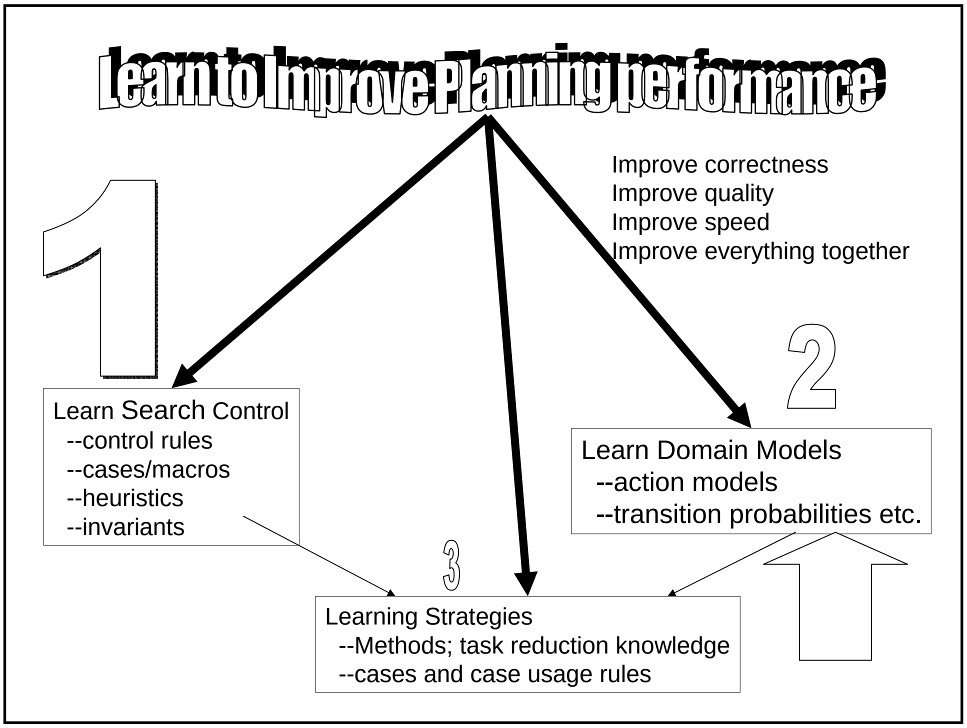
ARMS

(Doesn't assume intermediate states; but requires action parameters)

- Idea: See the example plans as "constraining" the hypothesis space of action models
  - The constraints can be modeled as SAT constraints (with variable weights)
  - Best hypotheses can be generated by solving the MAX-SAT instances
- Performance judged in terms of whether the learned action model can *explain* the correctness of the observed plans (in the test set)
  - Notice that if my theory is incomplete I might be able to explain both correct and incorrect plans to be correct…

- Constraints
  - Actions' preconditions and effects must share action parameters
  - Actions must have non-empty preconditions and effects; Actions cannot add back what they require; Actions cannot delete what they didn't ask for
  - For every pair of frequently co-occurring actions ai-aj, there must be some causal reason
    - E.g. ai must be giving something to aj OR ai is deleting something that aj gives

[Yang et. al. 2005]

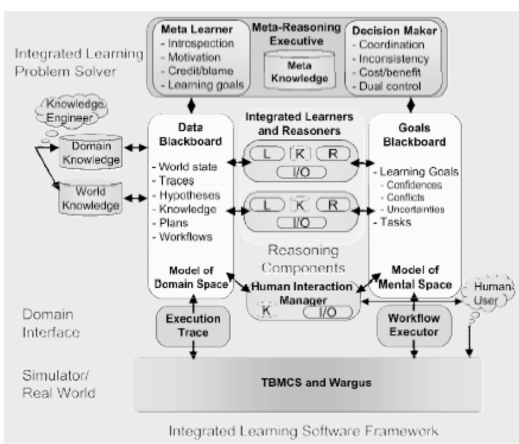


HTN Learning (Open)

- Learning hierarchical knowledge (e.g. HTN reductions) given plan examples is significantly harder
  - Often only the ground actions can be observed; the non-primitive actions cannot be
- Very little work on learning hierarchical knowledge
  - CAMEL system learns *applicability* conditions for specific methods *assuming that non-primitive actions are given*

Integrated Strategy Learning (Open)

- Learning strategic information in domains with partial domain knowledge
  - Work until now focused on either learning search control assuming full domain model OR learning domain model without much regard to search
  - A lot of important problems fall in the middle
    - Consider the problem of learning how to assemble a book-shelf by seeing it being made once
      - You clearly have at least partial knowledge about the domain (even if you, like me, are mechanically challenged)
        - » Even though you are unlikely to know all the tricky details
      - What you want to learn is general strategic knowledge about how to put together similar entities



- **Goal-driven, explanation-based learning approach of GILA gleans and exploits knowledge in multiple natural formats**
  - Single example analyzed under the lenses of multiple “ILRs” to learn/improve
    - Planning operators
    - Task networks
    - Planning cases
    - Domain uncertainty

Challenges and Solutions

- Domain Knowledge in multiple modalities
  → Use multiple ILRs customized to different types of knowledge
- Learning in multiple time-scales
  → Combine eager (e.g. EBL-style) and lazy (e.g. CBR-style) learning techniques
- Handling partially correct domain models and explanation
  → Use local closed world assumptions
- Avoiding balkanization of learned knowledge
  → Use structured explanations as a unifying “glue”
- Meeting explicit learning goals
  → Use Goal-driven meta-learning techniques

Summary

- Learning methods have been used in planning for both improving search and for learning domain physics
  - Most early work concentrated on search
  - Most recent work is concentrating on learning domain physics
    - Largely because we seem to have a very good handle on search
- Most effective learning methods for planning seem to be:
  - Knowledge based
    - Variants of Explanation-based learning have been very popular
  - Relational
- Many neat open problems...