ICML/COLT 1997

LEARNING in PLANNING

Manuela M. Veloso

Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213-3891
phone: (412) 268-8464
fax: (412) 268-5576
email: veloso@cs.cmu.edu
http://www.cs.cmu.edu/~mmv/

Short Tutorial Description: Learning in Planning

Planning is a decision making process which involves the generation of sequences of actions that achieve a set of goals from a given state. The main issues involved in the computational planning task are: How to acquire and represent the planning action model? How to generate plans in a computationally tractable way? How to create plans of good quality? and finally How to scale up to real-world problems?

Machine learning approaches can be applied to planning to automate the process of interpreting the planning experience into reusable task knowledge in order to improve the overall planner's performance. Learning in planning goes beyond inductive data classification. It addresses the acquisition of knowledge to efficiently guide a decision making process to reach solutions of good quality based on its input data. Learning in planning provides ground for both applied and theoretical research.

Comments on this Document

- Thanks to Daniel Borrajo, Universidad Carlos III, Madrid, for his help organizing this tutorial.
- A list of references (certainly not exhaustive) is included at the end of the document.
- The author of the tutorial is available for further explanations and contacts after the tutorial. Feel free to contact **veloso@cs.cmu.edu**.

Outline

- Motivation: Planning **and** Learning
- Planning
- Learning Applied to Planning
- Conclusion

Planning involves:

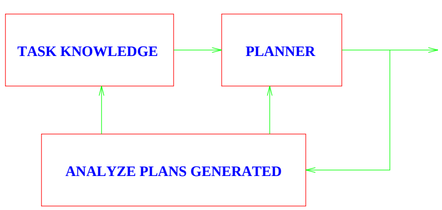
- Given knowledge about a task domain
- Given a problem specified as:
  - ▷ an initial configuration of the state of the "world"
  - ▷ a set of goals to be achieved
- Find a **solution** to the problem, i.e., a ***way*** to transform the initial configuration into a new state of the world where the goal statement is true.

Many issues to resolve...

A few are:

- What knowledge defines the task domain?
- How to represent the planning action model?
- What is the (sufficient) initial state of the world?
- What are the (prioritized) goals?
- How to acquire domain knowledge efficiently from expert human planners?
- Which algorithm to use to generate the solution plan?
- How to generate plans in a computationally tractable way?
- How to create plans of ***good*** quality?
- How to scale up to real-world problems?

Reaching Planning Expertise



- Knowledge engineering approaches:
  - ▷ Handcode and refine domain knowledge.
  - ▷ Specify control strategies.
  - ▷ Define knowledge to produce quality plans.
- Machine learning approaches:
  - ▷ **Automate** the interpretation of the planning experience into reusable task knowledge: domain, control, and quality.
  - ▷ Most recent trend: Combine with user's input.

Tutorial Goals

- Overview of planning algorithms
- Overview of learning approaches combined with planning

Accumulate and transfer problem solving experience

Outline

- Motivation: Planning **and** Learning
  - ▷ Knowledge engineering bottleneck
  - ▷ Learning: automated improvement with experience
  - ▷ Many learning opportunities in planning
- Planning
  - ▷ Introduction
  - ▷ Planning Algorithms
- Learning Applied to Planning
- Conclusion

Example: Process Planning Task

[**Pérez 95**, Gil, Hayes]

The essence of AI planning: forming plans to achieve goals.

How can we achieve a desired final configuration given some initial given one?

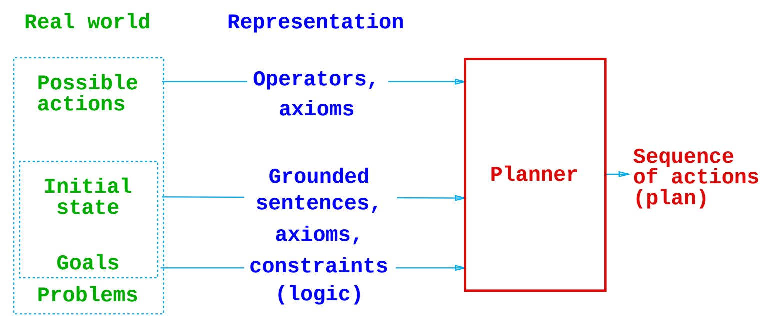
- Different processes (actions).
- Different machines.
- Different tools.
- Parts have orientations.
- Interaction among processes.
- Efficiency, quality, accuracy.
- ...

Planning Domains

Many AI planning domains with different degrees of *realism*:

- Process planning
- Image processing
- Logistics transportation
- Crisis management
- *Generating collection procedures*
- *Bank risk management*
- *Credit card fraud detection*
- Robot navigation
- Machine shop scheduling
- Blocks world
- Puzzles
- Matrix algebra
- Artificial domains
- ...

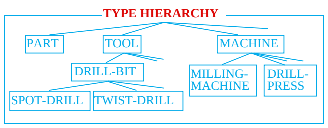
Example: Task Definition



- State-based representation of the world:
  - ▷ Operators: $F : states \rightarrow states$ (generalized)
  - ▷ Goal: set of sentences which must be true in the final state.

Example: Problem Representation

Objects
;;machines
(mm2 milling-machine)
(drill7 drill-press) ...
;;tools
(spot-drill3 spot-drill)
(twist-drill5 twist-drill)
(end-mill6 end-mill)
(soluble-oil soluble-oil) ...
;;holding devices
(vise12 vise) ...
;;parts
(part17 part)...



State
(diameter-of-drill-bit twist-drill5 9/64)
(material-of part17 aluminum)
(size-of part17 length 5)
(size-of part17 height 3)
(size-of part17 width 3)...

Goal ((<part> part))
(and (size-of <part> height 2)
(has-spot <part> hole1 side1 1.375 0.25)))

Plan
1. **put-tool-drill** drill7 spot-drill3
2. **put-holding-device-drill** drill7 vise12
3. **clean** part17
4. **put-on-machine-table** drill7 part17
5. **hold** drill7 vise12 part17 side1 side2-side5
6. **drill-in-drill-press** drill7 spot-drill3 part17

...

Example: Action Representation

drill-in-drill-press
<mach>: type drill-press
<drill-bit>: type spot-drill
<device>: type (or vise chuck)
<part>: type part
<hole>: type hole
<side>: type side
Pre: (holding-tool <mach> <drill-bit>)
(holding <mach> <device> <part>)
Add: (has-spot <part> <hole> <side>)
Del: (is-clean <part>)

put-tool-drill
<mach>: type drill-press
<tool>: type drill-bit
Pre: (avail-tool-holder <mach>)
(avail-tool <tool>)
Add: (holding-tool <mach> <tool>)
Del: (avail-tool-holder <mach>)
(avail-tool <tool>)

Many other actions (In Prodigy: more than 100):

- face-mill, remove-tool-from-drill, hold-with-vise,...

Domain Representation

- Operators – rules – with:
  - ▷ Precondition expression – must be satisfied before the operator is applied.
  - ▷ Set of effects – describe how the application of the operator changes the state.
- Precondition expression: propositional, typed first-order predicate logic, negation, conjunction, disjunction, existential and universal quantification, and functions.
- Effects: *add* and *delete* lists.
- Universally quantified effects.
- Conditional effects – dependent on conditions on the state.

Generating a Solution Plan

Several planning **algorithms**:

- Linear planning – Planning with a **stack** of goals.
- Nonlinear planning – Interleaving of goals
  - ▷ State-space search
  - ▷ Plan-space search
- Hierarchical planning
  - ▷ Emphasis on action decomposition/refinement
  - ▷ Very little search

A complex process:

- **Alternative operators to achieve a goal.**
- **Multiple goals that interact.**
- **Efficiency, quality, and accuracy – hard.**

Search Strategies

Using a set of operators:

- Forward chaining – progression
  - ▷ From the initial state,
  - ▷ Apply operators with preconditions true in the state,
  - ▷ Get new states.
- Backward-chaining – regression
  - ▷ From the goal state,
  - ▷ Find operators that can add goal,
  - ▷ Set its preconditions as new goals.
- Partial order – network of constraints among plan steps – no direct reasoning about an explicit state.
- Total order – plan steps are ordered during search – use of a uniquely specified state.

Planning Issues

- Representation of plans
  - ▷ Plans as sequence of state changes
  - ▷ Plans as total orders of steps
  - ▷ Plans as successive levels of refinement
  - ▷ Plans as partial orders of steps
- Conditional actions/effects
- Arbitrary functions computing side-effects
- Temporal reasoning – actions take time – explicit representation of time
- Interleave of planning and execution
- Nondeterministic outcome of actions
- Probabilistic occurrence of external events

▷ Planning as search, i.e., a decision-making process – learn search heuristics

▷ Planning representation – learn efficient, complete, correct domain specifications

Means-ends Analysis

[**Newell and Simon 60s**] [**Ernst and Newell 69**]

GPS Algorithm (*initial-state, goals*)

- If *goals* $\subseteq$ *initial-state*, then return *True*
- Choose a difference d between *initial-state* and *goals*
- Choose an operator o to reduce the difference d
- If no more operators, then return *False*
- *State*=**GPS**(*initial-state*, preconditions(o))
- If *State*, then return **GPS**(apply(o,*State*), *goals*)

Linear Planning

STRIPS reduced Algorithm (*initial-state, goals*)
[**Fikes and Nilsson 71**]
Stack= goals
State=initial-state
Repeat until *Stack=empty*
    Case top of *Stack* of
        operator:
            Unmet-preconditions=set of preconditions of o not true in *State*
            If *Unmet-preconditions= empty*,
            Then *State*=apply(o,*State*)
            Else Introduce *Unmet-preconditions* into *Stack*
        set of goals:
            If *goals*$\subseteq$*State*, Then remove *goals* from *Stack*
            (*) Introduce goal $g \in$*goals*| $g \notin$*initial-state* into *Stack*
        single goal:
            If *goal*$\subseteq$*State*, Then remove *goal* from *Stack*
            Else If goal loop, Then backtrack
                Else (*) Select operator o | $g \in$*effects*(o)
                    Introduce o in *Stack*

Linear Planning: Discussion

Advantages:

- Linear planning assumes that goals are independent.
- Reduced search space, because goals are solved one at a time.
- Clearly an advantage if goals are independent.

Disadvantages:

- Linear planning may produce *unoptimal* solutions.
- Linear planning is **incomplete.**

Strict Completeness: A planning algorithm is *strictly complete* if all the solutions to a given problem are included in its search space.

Completeness: A planning algorithm is *complete* if at least one solution to a given problem, when one exists, is included in its search space.

Example: Irreversible Actions


```
(OPERATOR LOAD-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<obj> OBJECT)
   (<loc> LOCATION))
  (and (at <obj> <loc>)
       (at <roc> <loc>)))
 (effects ()
  (add (inside <obj> <roc>))
  (del (at <obj> <loc>))))
```

```
(OPERATOR UNLOAD-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<obj> OBJECT)
   (<loc> LOCATION)))
  (and (inside <obj> <roc>)
       (at <roc> <loc>)))
 (effects ()
  (add (at <obj> <loc>))
  (del (inside <obj> <roc>))))
```

```
(OPERATOR MOVE-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<from-l> LOCATION)
   (<to-l> LOCATION))
  (and (at <roc> <from-l>)
       (has-fuel <roc>)))
 (effects ()
  (add (at <roc> <to-l>))
  (del (at <roc> <from-l>))
  (del (has-fuel <roc>))))
```


Incompleteness of Linear Planning

Initial state:
(at obj1 locA)
(at obj2 locA)
(at ROCKET locA)
(has-fuel ROCKET)

Goal statement:
(and
(at obj1 locB)
(at obj2 locB))

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	*failure*

Goal	*Plan*
(at obj2 locB)	(LOAD-ROCKET obj2 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj2 locB)
(at obj1 locB)	*failure*

State-Space Nonlinear Planning

Extend linear planning:

- From **stack** to **set** of goals.
- Include in the search space all possible interleaving of goals.

State-space nonlinear planning is **complete**.

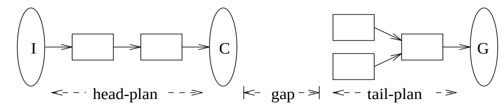
Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA)
(at obj2 locB)	(LOAD-ROCKET obj2 locA)
 (at obj1 locB)	(MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	(UNLOAD-ROCKET obj2 locB)

PRODIGY4.0 Planning Algorithm

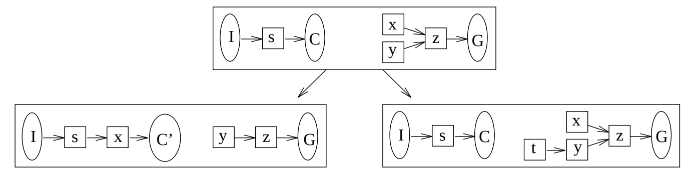
1. Terminate if the goal statement is satisfied in the current state.
2. Compute the **SET** of *pending goals* $\mathcal{G}$, and the **set** of *applicable operators* $\mathcal{A}$.
   - ▷ A goal is pending if it is a precondition, not satisfied in the current state, of an operator selected to be in the plan to achieve a particular goal.
   - ▷ An operator is applicable when all its preconditions are satisfied in the state.
3. Choose a goal G from $\mathcal{G}$ or select an operator A from $\mathcal{A}$.
4. If G has been chosen, then
   - ▷ *Expand goal* G, i.e., get the set $\mathcal{O}$ of *relevant instantiated operators* that could achieve the goal G,
   - ▷ Choose an operator O from $\mathcal{O}$,
   - ▷ Go to step 1.
5. If an operator A has been selected as directly applicable, then
   - ▷ *Apply* A,
   - ▷ Go to step 1.

Prodigy4.0 Search Representation

Representation of an incomplete plan during search:



Modifying the current plan – children of a search node:



Applying an operator (moving it to the head)

Adding an operator to the tail-plan

Hierarchical Planning

- General-purpose search heuristics do not solve reasonably complex representations of domains
- A well chosen simplification of the representation can improve the performance
- Need to simplify search and representation
- Key idea: Identify levels of abstraction, details.

Example: ABSTRIPS [Sacerdoti, 74]

- Each precondition has a *criticality value*
- Planning algorithm: *incremental refinement*
  - ▷ For *cv* from maximum-criticality-value down to minimum
    - Plan using only preconditions of criticality+*cv* refining previous abstract plan

Other examples:

- NOAH [Sacerdoti, 75] – Nets of action hierarchies
- O-PLAN [Tate 80] – elaborated abstract levels, no search

Representation in ABSTRIPS

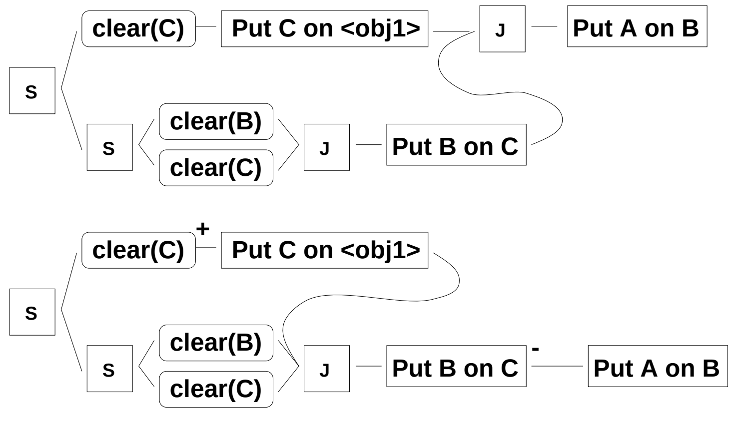
PUSH-THRU-DOOR

Preconditions:
{6}pushable(box1)∧{6}type(door1,DOOR)∧ {6}type(room1,ROOM)∧
{2}status(door1,OPEN)∧{1}next-to(box1,door1)∧ {1}next-to(ROBOT,box)∧
∃ room2 [{5}in-room(box,room2)∧{5}in-room(ROBOT,room2)∧
{6}connects(door1,room1,room2)]

Deletions:
at(ROBOT,$1,$2)∧next-to(ROBOT,$1)∧at(box1,$1,$2)∧
next-to(box1,$1)∧next-to($1,box1)∧in-room(ROBOT,$1)∧in-room(box1,$1)

Additions:
in-room(box1,room2)∧in-room(ROBOT,room2)∧next-to(ROBOT,box1)

Example of Planning in NOAH — Hierarchical



Several Other Planning Algorithms — State/Plan-space

- TWEAK [Chapman 87], SNLP [McAllester & Rosenblitt 91] , UCPOP [Penberthy and Weld 92]
  - ▷ Emphasis on plan-space *search*
- NONLIN [Tate 76], O-PLAN [Tate], SIPE [Wilkins 88]
  - ▷ Emphasis on plan decomposition
- UNPOP, Planning and acting [McDermott 78]
- Reactive planning [Georgeff & Lansky 87], [Firby 87], [Hendler & Sanborn 87]
- Action and time [Allen 84] [Dean & McDermott 87]
- Walksat [Selman et al. 92, Kautz & Selman 92, 96]
- Flecs [Veloso & Stone 95]
- Graphplan [Blum & Furst 95]

Plan-Space Partial-Order Nonlinear Planning — Plan-space

SNLP Planning Algorithm [McAllester & Rosenblitt 91]

1. Terminate if the goal set is empty.
2. Select a goal g from the goal set and identify the plan step that needs it, S_{need}.
3. Let S_{add} be a step (operator) that adds g, either a new step or a step that is already in the plan. Add the causal link $S_{add} \xrightarrow{g} S_{need}$, constrain S_{add} to come before S_{need}, and enforce bindings that make S_{add} add g.
4. Update the goal set with **all** the preconditions of the step S_{add}, and delete g.
5. Identify *threats* and resolve the conflicts by adding ordering or bindings constraints.

- A step S_k threatens a causal link $S_i \xrightarrow{g} S_j$ when it occurs between S_i and S_j, and it adds or deletes p.
- Resolve threats by using *promotion*, *demotion*, or *separation*.

State-space and Plan-space — Comparison

- Planning is NP-hard.
- Two different planning approaches: state-space and plan-space planning

	State-space	*Plan-space*
Commitments in plan step orderings	Yes	No
Therefore, suffer with goal orderings	Yes	No
Therefore, handle goal interactions	Poorly	Efficiently

Step Ordering Commitments

WHY?

Use of a uniquely specified STATE of the world while planning

In PRODIGY4.0 advantages include:

- Means-ends analysis - plan for goals that reduce the differences between current and goal states.
- Informed selection of operators - select operators that need less planning work than others.
- State useful for learning, generation and match of conditions supporting informed decisions.
- Helpful for generating anytime planning - provide *valid*, executable, plans at any time.
- Probabilistic planning - may be useful to reason about states, events that affect them, and eventual transitions.

Parallel between Commitments - Example

Operator Polish	Operator Drill-Hole
preconds: ()	preconds: ()
adds: polished	adds: has-hole
deletes: ()	deletes: polished

Goal: *polished* and *has-hole* Initial state: empty PRODIGY4.0	Goal: *polished* and *has-hole* Initial state: *polished* SNLP
- plan for goal *polished*	- plan for goal *polished*
- select *Polish*	- select *Initial* state
• order *Polish* as first step	• link *Initial* to *polished*
- plan for goal *has-hole*	- plan for goal *has-hole*
- select *Drill-Hole*	- select *Drill-Hole*
• order *Drill-Hole* $\succ$ *Polish*	• link *Drill-Hole* to *has-hole*
• *polished* deleted, backtrack	• threat - relink *polished*
- *Polish* $\succ$ *Drill-Hole*	- select *Polish*
	- link *Polish* to *polished*
	- *Polish* $\succ$ *Drill-Hole*

Serializability and Linkability

- A set of subgoals is *serializable* [Korf]:
  - If there exists some ordering whereby they can be solved sequentially,
  - without ever violating a previously solved subgoal.
- Easily serializable, laboriously serializable [Barrett and Weld].

- A set of subgoals is *easily linkable*:
  - If, independently of the order by which the planner links these subgoals to operators,
  - it never has to undo those links.
  - Otherwise it is *laboriously linkable*.

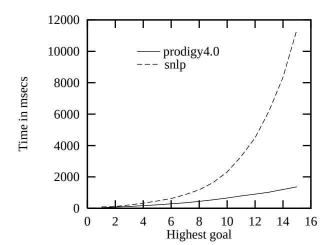
Laboriously Linkable Goals

operator A_i		operator A_*	
preconds	g_*, g_{i-1}	preconds	()
adds	g_i	adds	g_*
deletes	g_*	deletes	()

`Initial state`: g_*

`Goal statement`: g_*, g_5

`Plan`: $A_1, A_*, A_2, A_*, A_3, A_*, A_4, A_*, A_5$



The Importance of the Commitment Choice

- PRODIGY4.0's algorithm simplifies to the regular expression (**Subgoal** **Apply***)*.

Two main choices:
- Plan for a goal OR
- Change the state, i.e. simulate execution.

- PRODIGY4.0 uses **state** to determine ...
  - ▷ if the goal state has been reached.
  - ▷ which goals still need to be achieved.
  - ▷ which operators are applicable.
  - ▷ which operators to try first while planning.

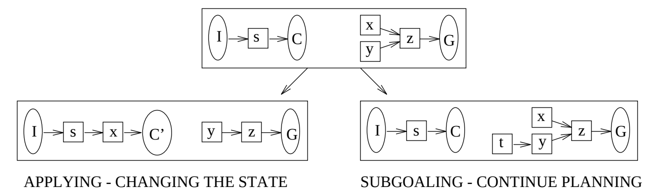
Two Heuristics: SAVTA, SABA

SAVTA**:** Eager application = **Eager** state changes

$\boxed{\text{Subgoal After eVery Try to Apply}}$

SABA**:** Eager subgoaling = **Delayed** state changes

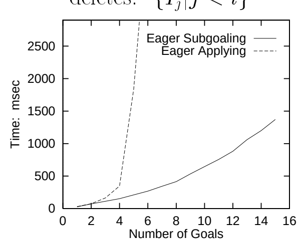
$\boxed{\text{Subgoal Always Before Applying}}$



Eagerly Subgoaling Can Be Better

Operator:	**paint-white** <obj>	**paint-yellow** <obj>	⋯	**paint-black** <obj>
preconds:	(usable white)	(usable yellow)	⋯	(usable black)
adds:	(white <obj>)	(yellow <obj>)	⋯	(black <obj>)
deletes:		(usable white)	⋯	(usable white)
			⋯	(usable yellow)
			⋮	⋮
				(usable brown)

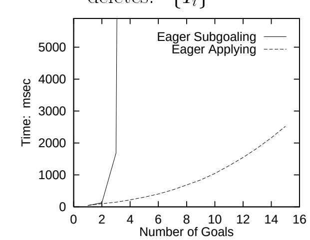
Operator: A_i
preconds: $\{I_i\}$
adds: $\{G_i\}$
deletes: $\{I_j | j < i\}$



Eagerly Applying Can Be Better

Operator:	**paint-with-brush1** <parts> <color>	...	**paint-with-brush8** <parts> <color>
preconds:	(unused brush1)	⋯	(unused brush8)
adds:	(painted <parts> <color>)	⋯	(painted <parts> <color>)
deletes:	(unused brush1)	⋯	(unused brush8)

Operator: A_i
preconds: $\{I_i\}$
adds: $\{< g >\}$
deletes: $\{I_i\}$



FLECS – Flexible Commitment Strategy

Comparison

Operator:	designate-roller <wall> <roller> <color>	fill-roller <roller> <color>	paint-wall <wall> <roller> <color>
preconds:	(clean <roller>) (needs-painting <wall>)	(clean <roller>) (chosen <roller> <color>)	(ready <wall> <roller> <color>) (filled-with-paint <roller> <color>)
adds:	(ready <wall> <roller> <color>) (chosen <roller> <color>)	(filled-with-paint <roller> <color>)	(painted <wall> <color>)
deletes:		(clean <roller>)	(ready <wall> <roller> <color>) (needs-painting <wall>)

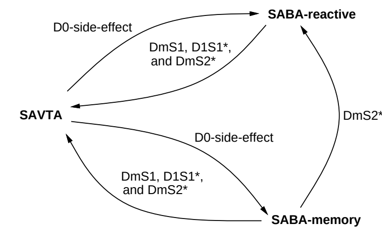
Initial State	Goal Statement	An Optimal Solution
(needs-painting wallA)	(painted wallA red)	<Designate-Roller wallA roller1 red>
(needs-painting wallB)	(painted wallB red)	<Designate-Roller wallB roller1 red>
(needs-painting wallC)	(painted wallC red)	<Designate-Roller wallC roller1 red>
(needs-painting wallD)	(painted wallD green)	<Fill-Roller roller1 red>
(needs-painting wallE)	(painted wallE green)	<Paint-Wall wallA roller1 red>
(clean roller1)		<Paint-Wall wallB roller1 red>
(clean roller2)		<Paint-Wall wallC roller1 red>
		<Designate-Roller wallD roller2 green>
		<Designate-Roller wallE roller2 green>
		<Fill-Roller roller2 green>
		<Paint-Wall wallD roller2 green>
		<Paint-Wall wallE roller2 green>

	time(sec)	solution
eager applying	500	no
eager subgoaling	500	no
variable strategy	4	yes

Summary – Planning

Comparison

- Several different planning algorithms.
- **There is not a planning strategy that is universally better than the others.**
- Even for a particular planning algorithm: **There is no single domain-independent search heuristic that performs more efficiently than others for all problems or in all domains.**



Learning is appropriate for **ANY** planner.

Outline

So far and next

- Motivation
- Case-Based Reasoning
- Rule-Based, Operator-Based Planning
  - ▷ Planning Algorithms
  - ▷ State-space planning; linear and nonlinear
  - ▷ Hierarchical planning
  - ▷ Plan-space planning
  - ▷ Comparison: Prodigy4.0 and SNLP; Different search heuristics in Prodigy4.0.
  - ▷ No universally optimal planning search algorithm or representation.
  - ▷ Learning from experience may improve planning performance.
- **Learning Applied to Planning**
- Planning by Analogical/Case-based Reasoning
- Conclusion

Macro-Operators

Macro-Ops

- First idea to apply learning to planning/problem solving
- Learning started being applied to state-space planning (STRIPS [Fikes & Nilsson, 72])
- Originally conceived for two-fold purpose:
  - ▷ Learning sequences of actions
  - ▷ Monitoring execution of plans
- Key idea: create new operators by joining the descriptions of the individual operators that form a plan
- Creation of macro-operators through triangle tables
- Examples: Rubik's cube [Korf, 83], ACT* [Anderson, 83], MORRIS [Minton, 85], ...
- Iterative macro-operators [Cheng & Carbonell 86] , [Shell & Carbonell 89]
- Flexible reuse of macro-operators [Yang & Fisher 92]

Triangle Tables

∗ on-table(A) ∗ clear(A) ∗ arm-empty	**pick-up(A)**		
∗ clear(B)	∗ holding(A)	**stack(A,B)**	
∗ clear(C) ∗ on(C,D)		clear(A) on(A,B) ∗ arm-empty	**unstack(C,D)**
		clear(A) on(A,B)	holding(C) clear(D)

Discussion: Macro-Operators

- Advantages:
  - ▷ Reuse of past experience
  - ▷ Replanning from failures
  - ▷ Less search depth
  - ▷ Less matching time
  - ▷ Side-effect: learning operators subsequences
- Disadvantages:
  - ▷ Considered in addition to simple operators
  - ▷ Increased branching factor
- Need to consider utility

Analytical Learning

Machine learning:

- Inductive methods
  - ▷ Data-intensive
  - ▷ Extract a general description of a *concept* from many examples
- Deductive methods
  - ▷ Knowledge-intensive
  - ▷ Explain and analyze single example of instance of concept
  - ▷ Explanation identifies the relevant features of the example = sufficient conditions for describing the concept.
  - ▷ Generalize instantiated explanation to apply to other instances of the concept.

Explanation-Based Generalization – EBG

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.

EBL: A Deductive Learning Method

Why are examples needed?

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

Actual purpose of EBL:

▷ <u>not</u> to "learn" more about target concept,

▷ <u>but</u> to "re-express" target concept in a more operational manner (=efficiency).

EBL Applied to Problem Solving/Planning

Inputs:

- Target concept definition – decision to be made
- Training example:
  ▷ The search episode with its successes and failures
- Domain theory:
  ▷ Operators used in the search
  ▷ Objects and possibly relationships in the world which may be used to build the explanation
- Operationality criterion:
  ▷ Describe concept using terms that are interpretable (efficiently) by the problem solver
  ▷ Several possible criteria

EBL Applied to Problem Solving/Planning

Output:

Generalization of the training example, that is

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

1. **Explain** (prove) why example is instance of target concept.
   - uses domain theory
   - prunes away unimportant aspects of example
   - final explanation is operational
2. **Generalize** explanation

EBL in PRODIGY

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

- knowledge-intensive approach
- analyzes trace of solving a problem
- explains "why" the choices made during problem solving were, or were not, appropriate
- acquires control knowledge – *better* search heuristics

Control rule:

- Applies 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.

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.
- If EBL system is eager to learn provably correct knowledge, the explanation effort is really large and the EBL system requires a *complete* domain theory for generalization.

Utility problem: The more rules learned, the slower the deliberation

- Possible solutions:
  - ▷ Perform utility analysis and discard low-utility rules
  - ▷ Heuristics to learn only effective knowledge
  - ▷ Incremental refinement of learned rules
- Factors influencing utility of control knowledge
  - ▷ Matching cost (cost of utilization)
  - ▷ Frequency of application
  - ▷ Savings every time it is applied

Learning in Nonlinear Plan-Space

- Application of known methods for State-space planners in Plan-Space planners
- Explanations in previous work compute the set of weakest preconditions
- These methods cannot be applied to partially ordered plans, because they not capture all interactions among plan operators of a partially ordered plan
- In Plan-Space planners, explanations are based on the Modal Truth Criterion
- [Kambhampati & Kedar 91], [Kambhampati & Chen 93]

Learning Control Rules in PO planning

- Differences with State-Based Planning
  - ▷ Different algorithm for regressing and generalizing explanations
  - ▷ Different types of failures
- Examples: SNLP+EBL (Katukam and Kambhampati, 94) and UCPOP+EBL (Qu and Kambhampati, 95)
- Types of failures
  - ▷ Analytical
  - ▷ Cross of depth limits (need of domain axioms)

SNLP Decision Points

1. Selection of open condition
2. Establishment of open conditions
   - ▷ Existing step (*which one?*)
   - ▷ Initial state (particular case of above)
   - ▷ New step (*which one?*)
3. Selection of a threat (*which one?*)
4. Resolution of a threat
   - ▷ Promotion (*where?*)
   - ▷ Demotion (*where?*)
   - ▷ Separation (addition of non-codesignation constraints) (SNLP)
   - ▷ Confrontation (conditional effects) (UCPOP)

Learning Process

- Backtracking applied to situations 2 and 4
- Intra-trial learning vs. after-trial learning
- Learning of selection and rejection search control rules
  - ▷ Construction of initial explanation
  - ▷ Regression of explanation over the decisions
  - ▷ Propagation of explanation up the failure branch
  - ▷ Generation of control rules
  - ▷ Simple utility analysis (do not learn when level of failure falls below constant l)
  - ▷ Rules storage (bounded isomorphism checks are done)

Analytical Failures

- Can be explained in terms of:
  - ▷ Inconsistencies in the ordering constraints (e.g. $(s_1 \prec s_2) \wedge (s_2 \prec s_1)$)
  - ▷ Inconsistencies in the binding constraints (e.g. $x \approx y \wedge x \not\approx y$)
  - ▷ Unestablishable open conditions (e.g. goal: $p(x)$ and $\nexists s \in S \mid p(x) \in \mathit{effects}(s)$)
- Generalization
  - ▷ Standard EBL: constants for variables
  - ▷ Bindings forced by initial and goal states are removed
  - ▷ Only binding constraints from the initial explanation are kept
  - ▷ Step names are also generalized (except for the *start* step)
- Discussion
  - ▷ Good results on some synthetic domains
  - ▷ Ineffective in recursive domains

Depth Limit Failures

- No domain independent explanation can be given to these failures
- Possible to use strong consistency checks based on domain axioms
- Restricted representation of domain axioms [Drummond & Curry, 88]

Operation:

- Necessarily preservable conditions (*np-conditions*) of a step s': $\mathit{np\text{-}conditions}(s') = \{c \mid s_1 \xrightarrow{c} s_2 \in \mathcal{L} \wedge s_1 \prec s' \prec s_2\}$
- $\mathit{preconds}(s') \cup \mathit{np\text{-}conditions}(s')$ must be consistent with respect to domain axioms

Discussion

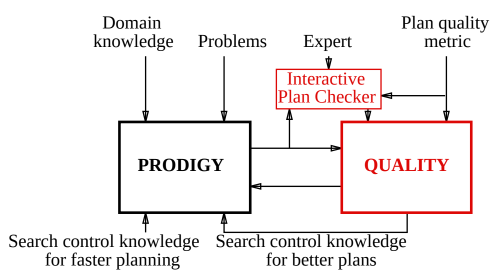
- Experimental data confirms the utility of learning search control rules for partial-order nonlinear planners (SNLP and UCPOP)
- The regression and propagation phases can be used as a form of dependency-directed backtracking
- In SNLP experiments, no control rule was generated from analytical failures
- In UCPOP experiments, the richer the representation, the easier to learn from analytical failures
- Depth Limit Failures require domain axioms
- Generated control rules are difficult to understand by an expert

Learning To Achieve Plan Quality

- Beyond learning to improve problem solving *efficiency*.
- Real-world applications begin to require *good quality* solutions.
- Interactions among goals and scenarios affect the quality of solutions
  - ▷ Explicit goal interactions – efficiency
  - ▷ Quality goal interactions (harder to learn)
- Plan length might not be the only cost measure
- Two approaches:
  - ▷ QUALITY learns from the difference between a *good* solution and a *worse* solution [Pérez 95]
  - ▷ HAMLET learns to select alternatives that lead to optimal solutions [Borrajo & Veloso 94, 96]

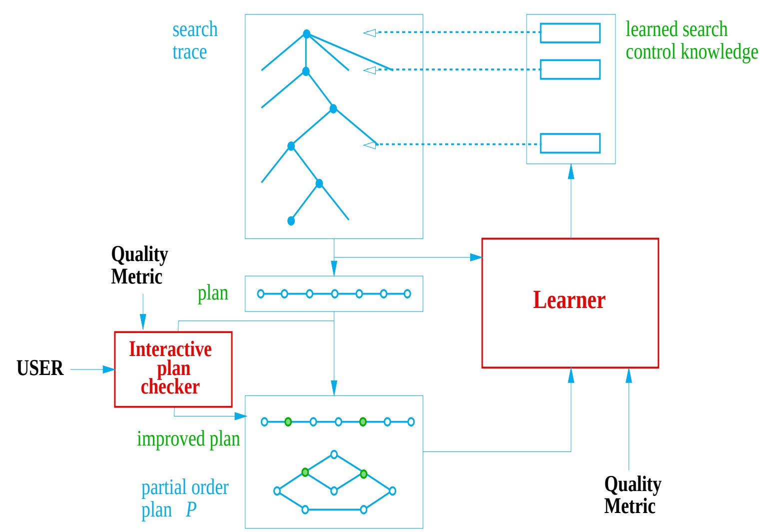
Learning to Generate Quality Plans

- Learn **control knowledge** to guide future search towards better plans (instead of "post-facto" plan modification)



- **Learning = change of representation:**
  From quality metric **into** search control knowledge available at problem solving decision time, since plan and search tree are only partially available.

QUALITY: The Architecture [Pérez 95]



Discussion

- Learn control rules to prefer operators, bindings, and goals in **domain-independent** fashion.
- Learning is **driven by failure**, when current control strategy must be overridden.
- If the **quality metric changes**, the learned knowledge is invalidated and re-learned.
- **Limited class** of quality metrics.
  - Tradeoffs in the quality factors lead to conflicts between rules.
  - Non-local tradeoffs are hard to capture with local control rules.

Solution: algorithms to learn and use **control-knowledge trees**.

Learning in Planning Search Trees

Labeling procedure:

- Find failure and successes to learn from
- Traverse trace (in post-order) labeling each node (failure, success, unknown).

Generation of control rules:

- Identify relevant features by goal regression
- Generalize instances in rules
- Left hand side (antecedent): conjunction of relevant features
- Right hand side (consequent): the decision learned

Outcome:

- Learned rules may be overspecific, i.e. may have a superset of the real relevant features.
- Learned rules may be overgeneral, i.e. may have a subset of the real relevant features (when applied to nonlinear planning)

Example - The Logistics Domain

- Packages are moved between cities. Trucks carry packages between locations within a city and airplanes carry packages across cities.
- There is no knowledge about
  - ▷ not moving carriers if they need to be loaded
  - ▷ unload a truck if an object is in the same city
  - ▷ load two objects "at the same time" if they need to go to the same place, and they are in the same place
- Changing representation is an open research option that we are also exploring

```
(operator FLY-AIRPLANE
    (preconds ((<plane> AIRPLANE)
               (<loc-from> AIRPORT)
               (<loc-to> AIRPORT))
     (at-airplane <plane> <loc-from>))
    (effects ((add (at-airplane <plane> <loc-to>))
              (del (at-airplane <plane> <loc-from>)))))
```


Other Logistics Domain Operators

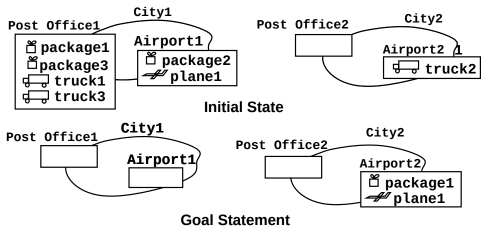

```
(OPERATOR UNLOAD-AIRPLANE
  (params <obj> <airplane> <loc>)
  (preconds ((<obj> object) (<airplane> airplane) (<loc> airport))
    (and (at-airplane <airplane> <loc>)
         (inside-airplane <obj> <airplane>)))
  (effects ((del (inside-airplane <obj> <airplane>))
            (add (at-object <obj> <loc>)))))

(OPERATOR LOAD-TRUCK
  (params <obj> <truck> <loc>)
  (preconds ((<obj> object) (<truck> truck) (<loc> location))
    (and (at-truck <truck> <loc>)
         (at-object <obj> <loc>)))
  (effects ((del (at-object <obj> <loc>))
            (add (inside-truck <obj> <truck>)))))

(OPERATOR DRIVE-TRUCK
  (params <truck> <loc-from> <loc-to>)
  (preconds ((<truck> truck) (<loc-from> location) (<loc-to> location))
    (and (same-city <loc-from> <loc-to>)
         (at-truck <truck> <loc-from>)))
  (effects ((del (at-truck <truck> <loc-from>))
            (add (at-truck <truck> <loc-to>)))))
```

Basic EBL is Over-General

Problem:



If only caring for efficiency, EBL learns the following rule:

```
(control-rule select-bind-fly-airplane-2
  (if (current-operator fly-airplane)
      (current-goal (at-airplane <plane1> <airport3>))
      (true-in-state (at-airplane <plane1> <airport2>)))
  (then select bindings ((<plane> . <plane1>)
                         (<loc-from> . <airport2>)
                         (<loc-to> . <airport3>))))
```

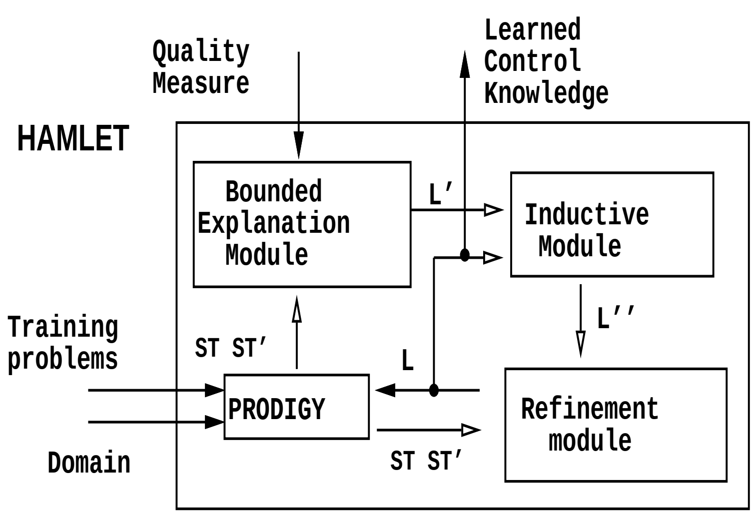

HAMLET: Deduction and Induction

- 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

Rule learned by HAMLET (previous example):

```
(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>))))
```


HAMLET's Architecture



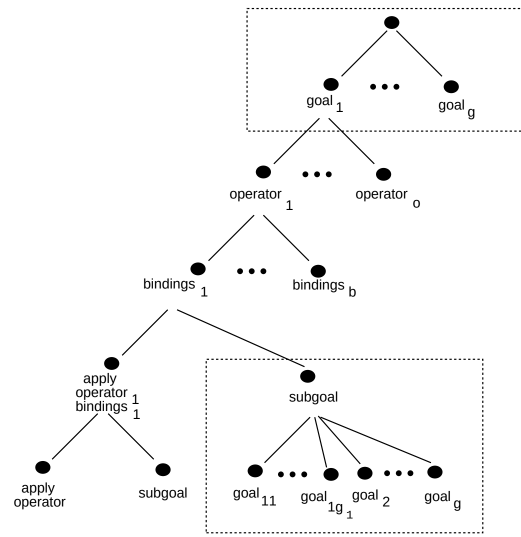
HAMLET's Algorithm

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")

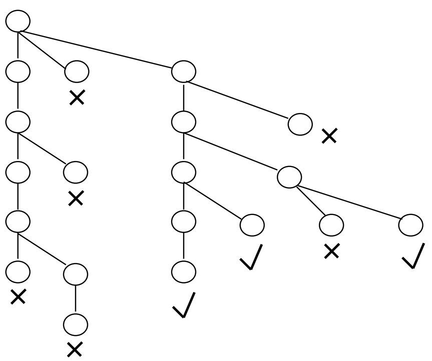
Bounded Explanation Module

- HAMLET's characteristics
  - ▷ no need for extra domain knowledge
  - ▷ reduced explanation effort
  - ▷ convergence towards correctness
- Bounded explanation steps
  - ▷ Labeling the decision tree. *Eagerness*
  - ▷ Credit Assignment. *Optimal learning*
  - ▷ Generation of control rule. *Goal Regression*
  - ▷ Parametrization. *Variable differentiation*

Generalized decision tree - Prodigy



A Typical Search Tree



What are the learning opportunities?

Induction Module

- Why induction?
  - ▷ Bounded explanation generates possibly over-specific rules
- HAMLET does induction over
  - ▷ State
  - ▷ Subgoaling structure
  - ▷ Interacting goals
  - ▷ Type hierarchy
- 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*

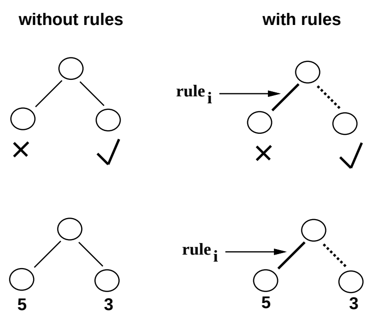
Inducing Over Two Rules

- 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
- A negative example for HAMLET is
  - ▷ Situation in which a control rule was applied, and
  - ▷ the resulting decision led to a *failure*, or
  - ▷ the resulting decision led to a *worse solution* than the best one for that decision

Negative Cases



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> <airplane4>)))
  (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 - Quality and Incremental

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 – Analytical Learning

- 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.

Why Analogical Reasoning

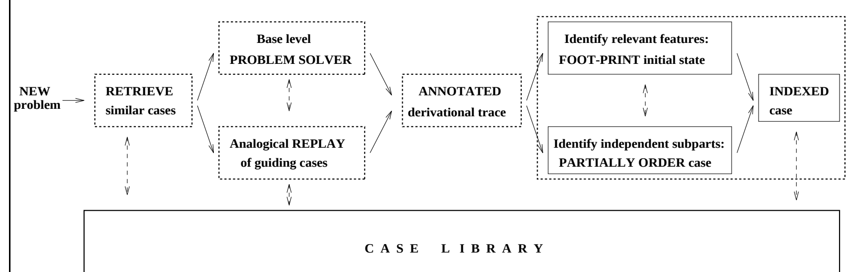
Derivational analogy/case-based reasoning in planning:

- Learns from local and **global** decisions chains – accumulates successful plans with justified local choices.
- Reuses **partially** matched learned experience – past and new problems need only to be *similar* for reuse.
- Performs **lazy** generalization, as learned episodes are not *explained* for correctness. (Therefore it does not require a complete domain theory.)

Tradeoffs EBL – Analogical reasoning:

- Hard to beat if provably correct learned knowledge.
- Learning at local decisions may increase the transfer of learned knowledge (but increases also the matching cost).
- Need to define a *similarity* metric between planning situations.

Prodigy/Analogy



Challenges of Analogy – CBR in Planning

- How to accumulate episodic problem solving experience?

What to preserve from the search tree?

- How to organize a large case library?

What are the appropriate indices?

- How to retrieve past experience efficiently?

What are similar problem situations?

- How to reuse a set of previously solved analogous problems?

What to transfer from partial matches?

Retaining Episodic Experience

What to preserve from a planning search experience?

- What is **needed** at replay time: **guidance for choices.**
- What is **naturally** known at search time.

- Identify decision points in the search procedure.
- Create language to capture justifications at search time and associate meaning for replay time.

Automatic Case Generation

A plan to be stored, i.e., a **case**, corresponds to:

- the compacted search tree
- a sequence of annotated decision nodes
- captures planning rationale

Annotations are the justifications
for the decisions taken:

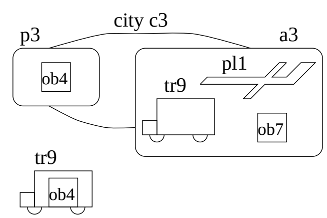
- Dependencies between goals and plan steps
- Record of failed explored alternative steps
- Pointers to eventual control guidance

Goal Node	*Applied Op Node*	*Chosen Op Node*
:step	*:step*	*:step*
:sibling-goals	*:sibling-goals*	*:sibling-relevant-ops*
:sibling-applicable-ops	*:sibling-applicable-ops*	*:why-this-operator*
:why-subgoal	*:why-apply*	*:relevant-to*
:why-this-goal	*:why-this-operator*	
:precond-of		

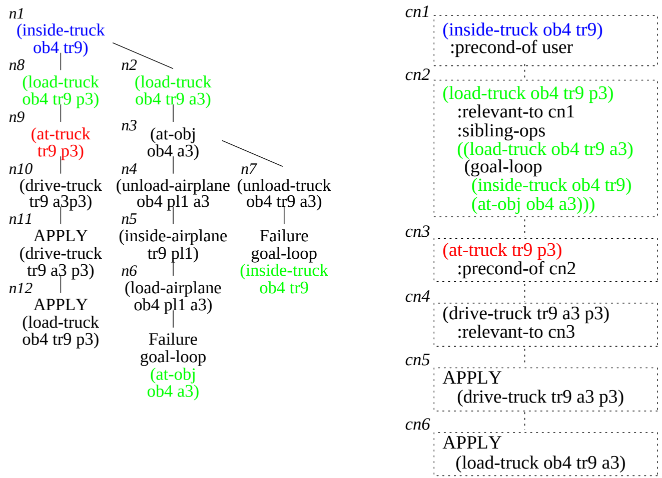
Example – Learning a Planning Case


```
(state (and
  (at-obj ob4 p3)
  (at-obj ob7 a3)
  (at-truck tr9 a3)
  (at-airplane pl1 a3)
  (same-city a3 p3))

(goal
  (inside-truck ob4 tr9))
```



Example (cont.)



Storage – Indexing a Case

What are the appropriate indices?

- The goal statement and the initial state define a problem.
- Parameterize the instantiated situation.

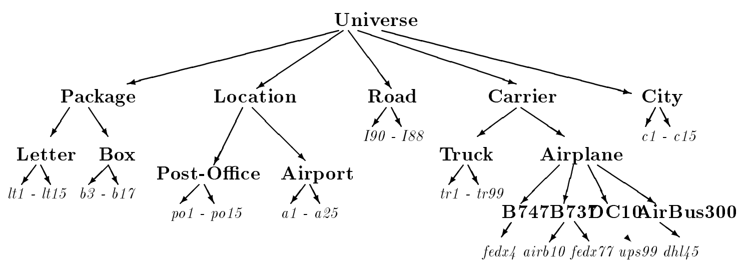
Index through:

the **relevant** initial state,

the set of **interacting** goals

Class Hierarchy

- Instances are defined through a class hierarchy.

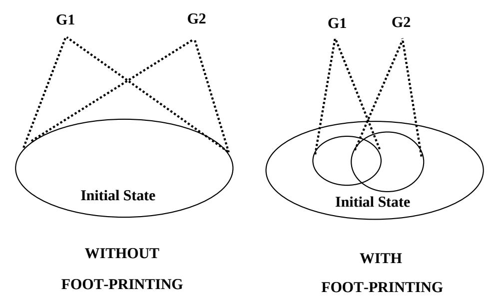


- Conservative reliable approach:

  ▷ **Parameterize to the immediate parent**

Foot-Printing the Initial State

- The derivational trace identifies for each goal the set of **weakest preconditions** necessary to achieve that goal (*goal regression*).



Indexing Parts of a Case

Partially ordered solution identifies **independent** subparts of a problem solving episode.

Goals in each subpart interact.

Retrieval Strategy

- Get guidance for possible interacting goals

Retrieve past cases where the problem solver experienced equivalent goal interactions.

- Goal interactions are responsible for the majority of the search.

Search Savings

Let

- b – average branching factor of the search tree,
- l – solution length,
- S – search effort without analogy,
- δ_{t_r} – match value between the case retrieved and the new problem, as a function of the retrieval time t_r.

Then the complexity of S is $S = \mathcal{O}(b^{\mathcal{O}(l)})$.

Effect of analogical reasoning:
decrease of the average branching factor
(directly related to the match value of the guiding case)

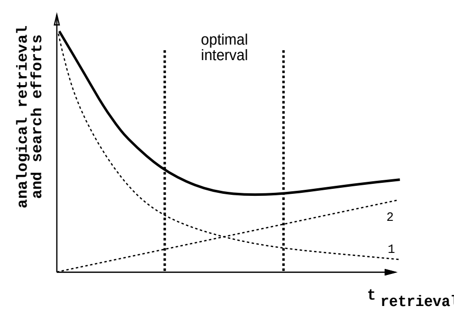
$$S_{analogy} = ((1 - \delta_{t_r})b)^l$$

Desired integration PS-CBR inequality:

$$\boxed{t_r + ((1 - \delta_{t_r})b)^l \ll b^l}$$

Optimal Retrieval Interval

$$t_r + (1 - m(1 - dC^{-\alpha t_r}))^l b^l \ll b^l$$



There is an optimal retrieval time interval which is a function of the match rate increase α.

Efficient Resource-Bounded Retrieval

- Indexing hash tables reduce the set of candidate analogs in constant time.
- Matching algorithm is incremental to allow stopping retrieval if some "reasonable" partial match is found.
- No effort to retrieve the *best* set of candidate analogs in the case library.

Example – Retrieval of Similar Problems

New problem

(goal (inside-airplane ob3 pl5)
(inside-truck ob8 tr2))

(initial-state (inside-truck ob3 tr2)
(at-truck tr2 p4)
(at-airplane pl5 a12)
(at-obj ob8 p4))

Past cases

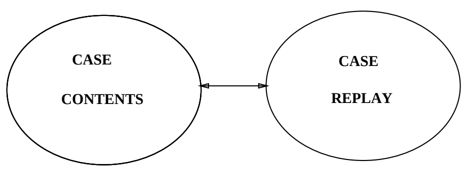
(goal (inside-airplane <ob12> <pl18>))
(relevant-state (at-obj <ob12> <ap29>)
(at-airplane <pl18> <ap37>))

(goal (inside-truck <ob32> <tr57>))
(relevant-state (at-obj <ob32> <po19>)
(at-truck <tr57> <ap7>))

(goal (inside-airplane ob3 pl5))
(relevant-state (at-obj ob3 <ap29>)
(at-airplane pl5 a12))

(goal (inside-truck ob8 tr2))
(relevant-state (at-obj ob8 p4)
(at-truck tr2 <ap7>))

Generation and Replay



Generation and replay
share representational **language**

Generation creates case language.
The replay procedure interprets it.

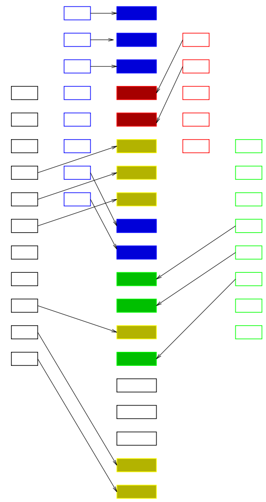
Replay involves:

- a complete reinterpretation of the justification structures in the new context
- the development of appropriate actions to be taken when transformed justifications are no longer valid.

Analogical Replay Of Multiple Plan Cases

1. Terminate if the goal is satisfied in the state.
2. Choose a guiding case. If a failure, then backtrack and reset pointers to guiding cases.
3. If a goal is chosen, then
   - 3.1. Validate the goal justifications. If not validated, go to step 2.
   - 3.2. Create a new goal node; link it to the case node. Advance the case.
   - 3.3. Select the operator chosen in the case.
   - 3.4. Validate the operator and bindings choices. If not validated, base-level plan for the goal. Go to step 2.
   - 3.5. Link the new operator node to the case node. Advance the case. Go to step 2.
4. If an applicable operator is chosen, then
   - 4.1. Check if it can be applied in the current state. If it cannot, do extra planning for the new goals. Go to step 2.
   - 4.2. Link the new applied operator node to the case node. Advance the case. Apply the operator. Go to step 1.

Sketch – Replaying Multiple Cases



Reuse of Annotated Experience

Extend case when extra planning is needed
Reduce case when past planning is not needed

Advantages of replay:

- Proposal and validation of choices versus generation and search of alternatives
- Reduction of the branching factor
  - ▷ past failed alternatives are pruned by validating the record of past failures;
  - ▷ if needed, PRODIGY/ANALOGY backtracks also in the guiding cases and uses information on failure to make more informed backtracking decisions.
- Subgoaling links identify the subparts of the case to replay – the steps that are not part of the active goals are skipped.

Replay of Multiple Planning Cases

Planning Cases	*New Planning Episode*
Chosen step	Proposed step
Goal dependencies	Search direction
Operator choices	Operator selections
Record of failures	Pruning of alternatives
Sibling alternatives	Proposed sibling steps
Additional reasons	Additional control

Extend cases when extra planning is needed.
Reduce cases when past planning is not needed.

Planning cases are merged to maintain global rationale

Global rationale includes:

- Interdependency between plan steps choices.
- Justification-based selection of alternatives.
- Avoidance of failures encountered.
- Additional information gathered.

An intelligent incremental learning process

Experiments

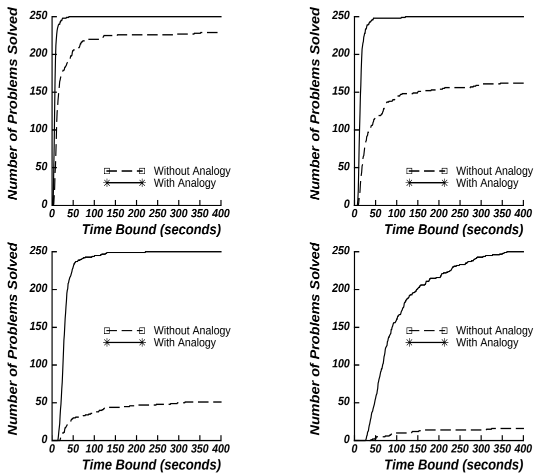
Several different domains, including logistics transportation

- Solvability horizon of generative planner is greatly increased due to the integrated replay of planning cases.

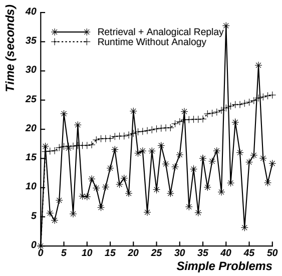
Example application domain: Route planning

- Routes are accumulated in a case library.
- Routes are abstracted and indexed according to situational parameters, such as: time of the day, day of the week, and driver.
- Geometric features are used by the similarity metric used at retrieval time [Haigh,Shewchuk].
- Multiple routes are merged at planning time.
- Planning cases are integrated with generative planning.
- Relevant parts of the cases are validated, pursued and merged.
- Generative planner does any extra planning work needed to merge the planning cases.

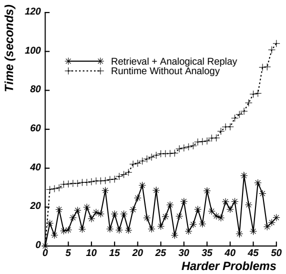
Solvability Horizon and Complexity



Retrieval plus Replay Time



Simple problems



Harder problems

Resource-Bounded Retrieval

Solution Length

Base-level planner better — *even* — *Analogical planner better*

-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	2	7	28	39	168	72	36	37	26	16	9	7	3	2	2	0	0	1
79						**168**	**211**												
17.25%						**36.68%**	**46.07%**												

7.9%	**16.8%**	**21.1%**	**54.2%**

7.9%	**92.1%**

Route Planning by Analogy

Discussion – PRODIGY/ANALOGY

- Integration of analogical reasoning into general problem solving as a method of learning at the strategy level.
- Characteristics of **learning by analogical reasoning** in PRODIGY/ANALOGY:
  - The strategy-level learning process is cast as the automation of the complete cycle of
    - constructing,
    - storing,
    - retrieving,
    - and replaying problem solving episodes.
  - No substantial effort invested in deriving general rules of behavior to apply to individual decisions.
  - Learned knowledge is flexibly applied to new situations, i.e., even if only a partial match exists among past and new problems.

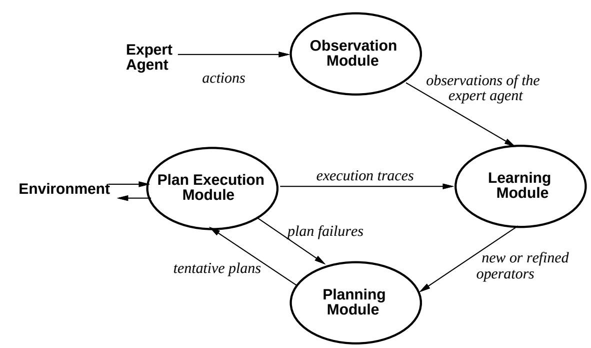
Learning Domain Knowledge

- **Gil 92** – EXPO
  - ▷ Automated refinement of planning operators
  - ▷ Refinement through controlled experimentation
- **Chen 92** – LIFE
  - ▷ Automated discovery of problem solving operators
- **Wang 95** – OBSERVE
  - ▷ Automated learning of planning operators
  - ▷ Observation of planning agent
  - ▷ Refinement through own practice

OBSERVE: Approach

- Motivation: Acquiring planning knowledge from experts is hard.
- Learn planning knowledge by observation and practice.
- Observe changes in the state:
  - Learn preconditions and effects of planning operators.
  - Infer subgoaling structure from observed plan.
- Generate plans from possibly over-specific planning knowledge.
- Repair plans and task knowledge from practice.

Learning Planning Knowledge [Wang 95]



$\boxed{\text{OBSERVE converges to correct planning domain description.}}$

Conclusion

- Motivation: Planning **and** Learning
  - ▷ Knowledge engineering bottleneck
  - ▷ Learning: automated improvement with experience
  - ▷ Many learning opportunities in planning
- Planning
  - ▷ Introduction
  - ▷ Planning Algorithms
  - ▷ State-space planning; linear and nonlinear
  - ▷ Plan-space planning; partial-order and hierarchical
  - ▷ Comparison: Prodigy4.0 and SNLP; Different search heuristics in Prodigy4.0.
  - ▷ No universally optimal planning search algorithm or representation.
  - ▷ Learning from experience may improve planning performance.

Conclusion (cont.)

- Learning
  - ▷ Learning opportunities
  - ▷ Learning control knowledge to improve efficiency
    - - Macro operators
    - - Explanation-based learning
    - - Analogical/case-based planning
  - ▷ Learning control knowledge to improve plan quality
    - - Incorporate user's evaluation metric
    - - Incremental inductive refinement
  - ▷ Learning planning domain operators
    - - Observation and practice
- Conclusion: This summary

References

[1] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, jan 1991.

[2] James F. Allen, James Hendler, and Austin Tate (eds.). *Readings in Planning*. Morgan Kaufmann, 1990.

[3] John Allen and Pat Langley. Integrating memory and search in planning. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 301–312, San Diego, CA, November 1990. Morgan Kaufmann.

[4] John R. Anderson. *The Architecture of Cognition*. Harvard University Press, Cambridge, Mass, 1983.

[5] Anthony Barrett and Daniel S. Weld. Characterizing subgoal interactions for planning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1388–1393, 1993.

[6] Anthony Barrett and Daniel S. Weld. Partial-order planning:Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1), 1994.

[7] Neeraj Bhatnagar. *On-line learning from search failures*. PhD thesis, Rutgers University, 1992.

[8] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995. Extended version to appear in *Artificial Intelligence*, 1997.

[9] Daniel Borrajo and Manuela Veloso. Incremental learning of quality-oriented control knowledge for planning. In *Working notes of the AAAI Fall Series Symposium 1994 on Planning and Learning*, New Orleans, LO, November 1994.

[10] Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal. Special Issue on Lazy Learning*, 10:1–34, 1996.

[11] Jaime G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*, pages 137–162, Palo Alto, CA, 1983. Tioga Press.

[12] Jaime G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume II*, pages 371–392. Morgan Kaufman, 1986.

[13] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, SCS, Carnegie Mellon University, June 1992.

[14] Jaime G. Carbonell and Yolanda Gil. Learning by experimentation: The operator refinement method. In R. S. Michalski and Y. Kodratoff, editors, *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 191–213. Morgan Kaufmann, Palo Alto, CA, 1990.

[15] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[16] Pat W. Cheng and Jaime G. Carbonell. The FERMI system: Inducing iterative rules from experience. In *Proceedings of AAAI-86*, pages 490–495, Philadelphia, PA, 1986.

[17] Ken Currie and Austin Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 1990.

[18] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[19] Kenneth DeJong. Learning with genetic algorithms: An overview. *Machine Learning*, 3(2/3):121–138, October 1988.

[20] Robert B. Doorenbos and Manuela M. Veloso. Knowledge organization and the utility problem. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 28–34, Amherst, MA, June 1993.

[21] Mark Drummond and Ken Currie. Goal ordering in partially ordered plans. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 960–965, Detroit, MI, 1989.

[22] George W. Ernst and Allen Newell. *GPS: A Case Study in Generality and Problem Solving*. ACM Monograph Series. Academic Press, New York, NY, 1969.

[23] Tara A. Estlin and Raymond Mooney. Hybrid learning of search control for partial order planning. In *New Directions in AI Planning*. IOS Press, 1996. Proceedings of the Third European Workshop on Planning, 1995.

[24] Oren Etzioni. *A Structural Theory of Explanation-Based Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as technical report CMU-CS-90-185.

[25] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–301, 1993.

[26] Richard E. Fikes, P. E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[27] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[28] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[29] Karen Z. Haigh, Jonathan Shewchuk, and Manuela M. Veloso. Exploring geometry in analogical route planning. *To appear in Journal of Experimental and Theoretical Artificial Intelligence*, 1997.

[30] Kristian J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, 1986.

[31] Steve Hanks and Daniel Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.

[32] Laurie Ihrig and Subbarao Kambhampati. Derivational replay for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 992–997, 1994.

[33] Robert L. Joseph. Graphical knowledge acquisition. In *Proceedings of the 4^{th} Knowledge Acquisition For Knowledge-Based Systems Workshop*, Banff, Canada, 1989.

[34] Subbarao Kambhampati. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. PhD thesis, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD, 1989.

[35] Subbarao Kambhampati and Jengchin Chen. Relative utility of EBG based plan reuse in partial ordering vs. total ordering planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 514–519, 1993.

[36] Subbarao Kambhampati and James A. Hendler. A validation based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, 1992.

[37] Subbarao Kambhampati and Smadar Kedar. Explanation based generalization of partially ordered plans. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 679–685, 1991.

[38] Suresh Katukam and Subbarao Kambhampati. Learning explanation-based search control rules for partial order planning. In *Proceedings of the AAAI-94*. AAAI, 1994.

[39] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of ECAI-92, European Conference on Artificial Intelligence*, Vienna, Austria, 1992.

[40] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[41] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68, 1994.

[42] Richard E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[43] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[44] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[45] Pat Langley. Learning effective search heuristics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 419–421, 1983.

[46] C. Leckie and I. Zukerman. Learning search control rules for planning: An inductive approach. In *Proceedings of Machine Learning Workshop*, pages 422–426, 1991.

[47] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, 1991.

[48] Drew V. McDermott. Planning and acting. *Cognitive Science*, 2-2:71–109, 1978.

[49] R. S. Michalski, J. G. Carbonell, and T. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume I. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1983.

[50] R. S. Michalski, J. G. Carbonell, and T. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[51] R. S. Michalski and Y. Kodratoff, editors. *Machine Learning, An Artificial Intelligence Approach*, volume III. Morgan Kaufmann, Palo Alto, CA, 1990.

[52] R. S. Michalski and G. Tecucci, editors. *Machine Learning, A Multistrategy Approach*, volume IV. Morgan Kaufmann, Palo Alto, CA, 1994.

[53] Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 1983.

[54] Steven Minton. Selectively generalizing plans for problem solving. In *Proceedings of AAAI-85*, pages 596–599, 1985.

[55] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.

[56] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Dan R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: Optimizing problem solving performance through experience. *Artificial Intelligence*, 1989.

[57] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[58] Tom M. Mitchell and Sebastian B. Thrun. Explanation based learning: A comparison of symbolic and neural network approaches. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 197–204, University of Massachusetts, Amherts, MA, USA, 1993. Morgan Kaufmann.

[59] Tom M. Mitchell, Paul E. Utgoff, and R. B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine Learning, An Artificial Intelligence Approach*, volume I, pages 163–190. Tioga Press, Palo Alto, CA, 1983.

[60] Jack Mostow. Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, and T. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume I*, volume I, pages 367–403. Morgan Kaufman, Los Altos, CA, 1983.

[61] Héctor Muñoz-Avila, Juergen Paulokat, and Stefan Wess. Controlling a nonlinear hierarchical planner using case-based reasoning. In *Proceedings of the 1994 European Workshop on Case-Based Reasoning*, November 1994.

[62] Allen Newell, J. C. Shaw, and Herbert A. Simon. Empirical explorations with the logic theory machine: A case study in heuristics. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, NY, 1963.

[63] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[64] D. Ourston and R.J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66, 1994.

[65] J. S. Penberthy and D. S. Weld. UCPOP:A sound, complete, partial order planner for ADL. In *Proceedings of KR-92*, pages 103–114, 1992.

[66] M. Alicia Pérez. *Learning Search Control Knowledge to Improve Plan Quality*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1995. Available as technical report CMU-CS-95-175.

[67] M. Alicia Pérez and Jaime G. Carbonell. Control knowledge to improve plan quality. In *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, IL, 1994.

[68] M. Alicia Pérez and Oren Etzioni. DYNAMIC: A new role for training problems in EBL. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Conference on Machine Learning*, pages 367–372. Morgan Kaufmann, San Mateo, CA, 1992.

[69] Yong Qu and Subbarao Kambhampati. Learning search control rules for plan-space planners: Factors affecting the performance. Technical report, Arizona State University, February 1995.

[70] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, August 1990.

[71] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, Inc., 1991. Second edition.

[72] Paul S. Rosenbloom, Allen Newell, and John E. Laird. Towards the knowledge level in SOAR: The role of the architecture in the use of knowledge. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990.

[73] David Ruby and Dennis Kibler. Learning episodes for optimization. In *Proceedings of the Machine Learning Conference 1992*, pages 379–384, San Mateo, CA, 1992. Morgan Kaufmann.

[74] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[75] Arthur Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, NY, 1963.

[76] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[77] Jude W. Shavlik and Geoffrey G. Towell. Refining symbolic knowledge using neural networks. In Ryszard Michalski and Gheorghe Tecuci, editors, *Machine Learning. A Multistrategy Approach.*, volume IV, pages 405–429. Morgan Kaufmann, 1994.

[78] Peter Shell and Jaime G. Carbonell. Towards a general framework for composing disjunctive and iterative macro-operators. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[79] W. M. Shen. Functional transformations in AI discovery systems. *Artificial Intelligence*, 41:257–272, 1990.

[80] Mark Stefik. Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16:141–169, 1981.

[81] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–140, 1981.

[82] R.E. Step and R.S. Michalski. Conceptual clustering: inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume II*. Morgan Kaufman, 1986.

[83] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 164–169, June 1994.

[84] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 694–700, San Mateo, CA, 1989. Morgan Kaufmann.

[85] M. Tambe, A. Newell, and P. S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5(3):299–348, 1990.

[86] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–900, 1977.

[87] Manuela Veloso and Jim Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 170–175, June 1994.

[88] Manuela Veloso and Daniel Borrajo. Learning strategy knowledge incrementally. In *Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence*, New Orleans, LO, November 1994.

[89] Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, pages 81–120, 1995.

[90] Manuela M. Veloso. *Planning and Learning by Analogy*. Springer Verlag, 1994.

[91] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278, 1993.

[92] Manuela M. Veloso and Jaime G. Carbonell. Towards scaling up machine learning: A case study with derivational analogy in PRODIGY. In S. Minton, editor, *Machine Learning Methods for Planning*, pages 233–272. Morgan Kaufmann, 1993.

[93] Manuela M. Veloso and Jaime G. Carbonell. Case-based reasoning in PRODIGY. In R. S. Michalski and G. Teccuci, editors, *Machine Learning: A Multistrategy Approach, Volume IV*, pages 523–548. Morgan Kaufmann, 1994.

[94] Manuela M. Veloso and Peter Stone. FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, 3:25–52, 1995.

[95] R. Waldinger. Achieving several goals simultaneously. In N. J. Nilsson and B. Webber, editors, *Readings in Artificial Intelligence*, pages 250–271. Morgan Kaufman, Los Altos, CA, 1981.

[96] Xuemei Wang and Manuela Veloso. Learning planning knowledge by observation and practice. In *Proceedings of the ARPA Planning Workshop*, Tucson, AZ, February 1994.

[97] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[98] Hua Yang and Douglas Fisher. Similarity-based retrieval and partial reuse of macro-operators. Technical Report CS-92-13, Department of Computer Science, Vanderbilt University, 1992.

[99] J. Zelle and R. Mooney. Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1993.