CSE 473

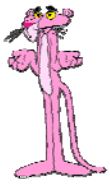
Automated Planning

Dan Weld

(With slides by UW AI faculty & Dana Nau

I have a plan - a plan that cannot possibly fail.

- Inspector Clousseau



Popular Application

© Mausam

Overview

- Introduction & Agents
- Search, Heuristics & CSPs
- Adversarial Search
- **Logical Knowledge Representation**
- **Planning** & MDPs
- Reinforcement Learning
- Uncertainty & Bayesian Networks
- Machine Learning
- NLP & Special Topics

Planning & Logic

- Actions specified using first-order logic
- Planning implemented using SAT solver
  - E.g., DPLL or WalkSAT
  - Also an example of solving FOL using propositional SAT

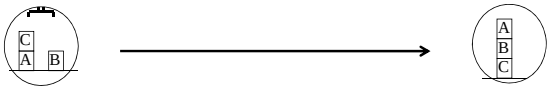
Logistics

- PS2 due today
- HW1 due in one week
  - Parts due in between:
    - Friday written problem
    - Monday feedback on another person's answer
    - Wed revise your answer

Planning

- Given
  - a logical description of the **initial situation**,
  - a logical description of the **goal conditions**, and
  - a logical description of a set of **possible actions**,
- Find
  - a **sequence of actions** (a **plan of actions**) that brings us from the initial situation to a situation in which the goal conditions hold.

Example: BlocksWorld



Planning Input: State Variables/Propositions

- Types: block --- a, b, c
- (on-table a) (on-table b) (on-table c)
- (clear a) (clear b) (clear c)
- (arm-empty)
- (holding a) (holding b) (holding c)
- (on a b) (on a c) (on b a) (on b c) (on c a) (on c b)

on-table(a)

No. of state variables =16
No. of states = 2^{16}
No. of reachable states = ?

- (on-table ?b); clear (?b)
- (arm-empty); holding (?b)
- (on ?b1 ?b2)

Planning Input: Actions

- pickup a b, pickup a c, …
- place a b, place a c, …
- pickup-table a, pickup-table b, …
- place-table a, place-table b, …
- pickup ?b1 ?b2
- place ?b1 ?b2
- pickup-table ?b
- place-table ?b

Total: 6 + 6 + 3 + 3 = 18 "ground" actions
Total: 4 action schemata

Planning Input: Actions (contd)

- :action pickup ?b1 ?b2
  :precondition
  (on ?b1 ?b2)
  (clear ?b1)
  (arm-empty)
  :effect
  (holding ?b1)
  (not (on ?b1 ?b2))
  (clear ?b2)
  (not (arm-empty))
- :action pickup-table ?b
  :precondition
  (on-table ?b)
  (clear ?b)
  (arm-empty)
  :effect
  (holding ?b)
  (not (on-table ?b))
  (not (arm-empty))

Planning Input: Initial State

- (on-table a) (on-table b)
- (arm-empty)
- (clear c) (clear b)
- (on c a)

- All other propositions false
  - not mentioned → assumed false
  - "Closed world assumption"

Planning Input: Goal

- (on-table c) AND (on b c) AND (on a b)
- Is this a state?
- In planning a goal is a *set of states*
  - Like the goal test in problem solving search
  - But specified declaratively (in logic) rather than with code

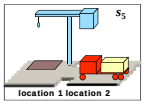
Planning *vs.* Problem-Solving ?

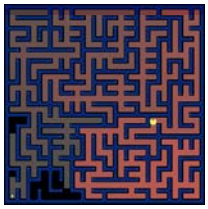
Basic difference: **Explicit, logic-based representation**

- States/Situations: descriptions of the world by logical formulae
  → agent can explicitly reason about the world.
- Goal conditions as logical formulae vs. goal test (black box)
  → agent can reflect on its goals.
- Operators/Actions: Transformations on logical formulae
  → agent can reason about the effects of actions
  by inspecting the definition of its operators.

One Planner Solves Many Domains

"no code required"





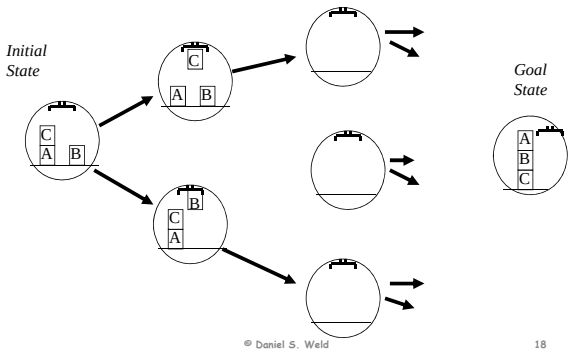
Specifying a Planning Problem

- Description of initial state of world
  – Set of propositions
- Description of goal:
  – E.g., Logical conjunction
  – Any world satisfying conjunction is a goal
- Description of available actions

Classical Planning

- Simplifying assumptions
  – Atomic time
  – Agent is omniscient (no sensing necessary).
  – Agent is sole cause of change
  – Actions have deterministic effects
- STRIPS representation
  – World = set of true propositions (conjunction)
  – Actions:
    - Precondition: (conjunction of *positive* literals, no functions)
    - Effects (conjunction of literals, no functions)
  – Goal = conjunction of *positive* literals
    on(A,B) ∧ on(B, C)

Forward World-Space Search



Forward State-Space Search

- Initial state: set of positive ground literals
  – CWA: literals not appearing are false
- Actions:
  – applicable if preconditions satisfied
  – add positive effect literals
  – remove negative effect literals
- Goal test: does state logically satisfy goal?
- Step cost: typically 1

Heuristics for State-Space Search

- Count number of false goal propositions in current state
  - Admissible?
  - NO
- Subgoal independence assumption:
  - Cost of solving conjunction is sum of cost of solving each subgoal independently
  - Optimistic: ignores negative interactions
  - Pessimistic: ignores redundancy
  - Admissible? No
  - Can you make this admissible?

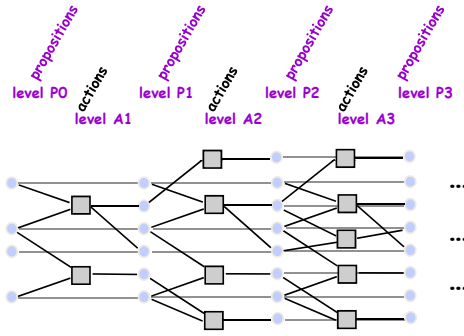
Heuristics for State Space Search (contd)

- Delete all preconditions from actions, solve easy relaxed problem, use length
  - Admissible?
  - YES
- **:action pickup-table ?b**
  :precondition (and (on-table ?b)
  (clear ?b)
  (arm-empty))
  :effect (and (holding ?b)
  ~~**(not (on-table ?b))**~~
  ~~**(not (arm-empty)))**~~

Planning Graph: Basic idea

- Construct a planning graph: encodes constraints on possible plans
- Use this planning graph to compute an informative heuristic (Forward A*)
- Planning graph can be built for each problem in polynomial time

The Planning Graph



Note: a few noops missing for clarity

Regression search

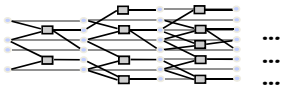
- States
- Operators
- Initial State
- Goal

Planning Graphs

- Planning graphs consists of a seq of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions that represent what *might be* possible at that step in the plan
  - *Might be* is the key to efficiency
  - Records only a restricted subset of possible negative interactions among actions.

Planning Graphs

- Alternate levels
  - *Literals* = all those that *could* be true at that time step, depending upon the actions executed at preceding time steps.
  - *Actions* = all those actions that *could* have their preconditions satisfied at that time step, depending on which of the literals actually hold.



PG Example

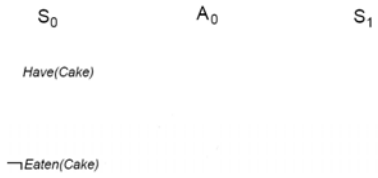
Init(Have(Cake))

Goal(Have(Cake) ∧ Eaten(Cake))

Action(Eat(Cake),
PRECOND: Have(Cake)
EFFECT: ¬Have(Cake) ∧ Eaten(Cake))

Action(Bake(Cake),
PRECOND: ¬ Have(Cake)
EFFECT: Have(Cake))

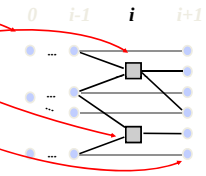
PG Example



Create level 0 from initial problem state.

Graph Expansion

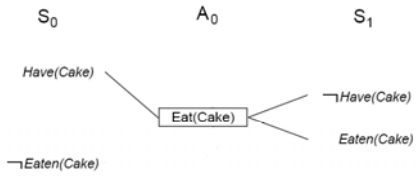
Proposition level 0
initial conditions
Action level i
no-op for each proposition at level i-1
action for each operator instance whose preconditions exist at level i-1
Proposition level i
effects of each no-op and action at level i



No-op-action(P),
PRECOND: P
EFFECT: P
Have a no-op action for each ground fact

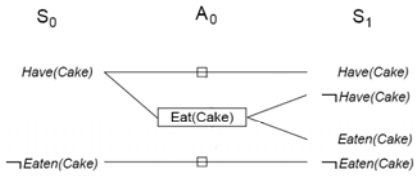
 33

PG Example



Add all applicable actions.
Add all effects to the next state.

PG Example

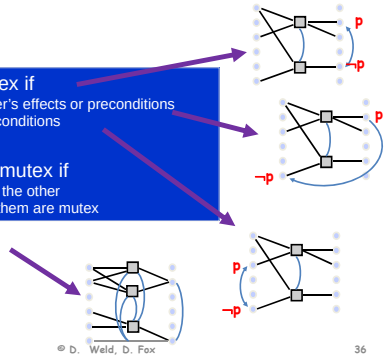


Add *persistence actions* (aka no-ops) to map all literals in state S_i to state S_{i+1}.

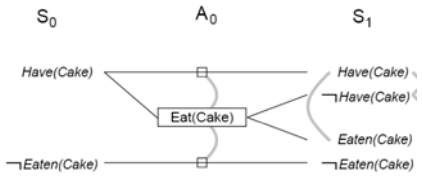
Mutual Exclusion

Two actions are mutex if
- one clobbers the other's effects or preconditions
- they have mutex preconditions

Two proposition are mutex if
- one is the negation of the other
- all ways of achieving them are mutex

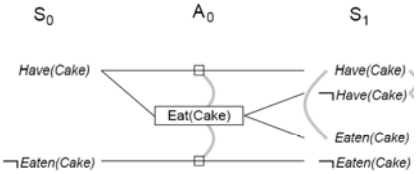


PG Example



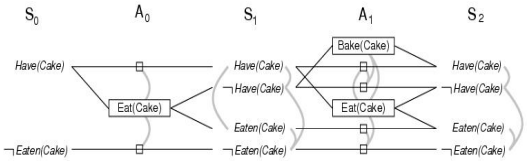
Identify *mutual exclusions* between actions and literals based on potential conflicts.

Cake example

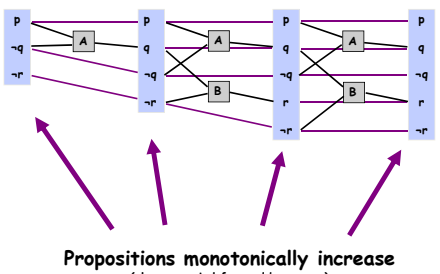


- Level S_1 contains all literals that might result from picking any subset of actions in A_0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states and the mutex links are the constraints that define this set of states.

Cake example

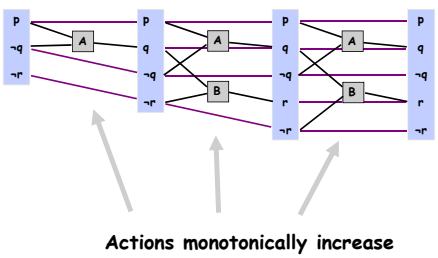


Observation 1



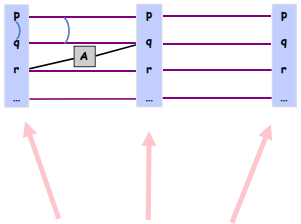
Propositions monotonically increase
(always carried forward by no-ops)

Observation 2



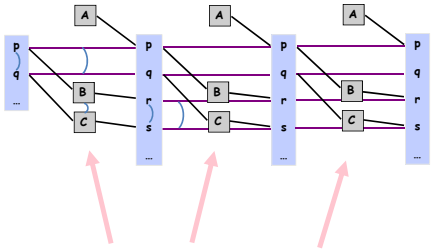
Actions monotonically increase

Observation 3



Proposition mutex relationships monotonically decreas

Observation 4



Action mutex relationships monotonically decrease

Observation 5

Planning Graph 'levels off'.

- After some time k all levels are identical
- Because it's a finite space, the set of literals never decreases and mutexes don't reappear.

Properties of Planning Graph

- If goal is absent from last level?
  - Then goal cannot be achieved!
- If there exists a plan to achieve goal?
  - Then goal is present in the last level &
  - No mutexes between conjuncts
- If goal is present in last level (w/ no mutexes) ?
  - There still may not exist any viable plan

Heuristics based on Planning Graph

- Construct planning graph starting from s
- h(s) = level at which goal appears non-mutex
  - Admissible?
  - YES
- Relaxed Planning Graph Heuristic
  - Remove negative preconditions build plan. graph
  - Use heuristic as above
  - Admissible? YES
  - More informative? NO
  - Speed: FASTER

FF

- Topmost classical planner until 2009
- State space local search
  - Guided by relaxed planning graph
  - Full best-first seach to escape plateaus
  - A few other bells and whistles…

Planning Summary

- Problem solving algorithms that operate on explicit propositional representations of states and actions.
- Make use of domain-independent heuristics.
- STRIPS: restrictive propositional language
- Heuristic search
  - forward (progression)
  - backward (regression) search [didn't cover]
- Local search FF [didn't cover]

Generative Planning

Input

Description of (initial state of) world *(in some KR)*
Description of goal *(in some KR)*
Description of available actions *(in some KR)*

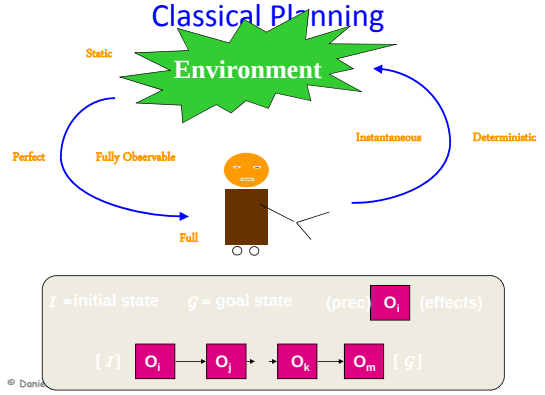
Output

Controller
E.g. Sequence of actions
E.g. Plan with loops and conditionals
E.g. Policy = f: states -> actions

Input Representation

- *Description of initial state of world*
  - E.g., Set of propositions:
  - **((block a) (block b) (block c) (on-table a) (on-table b) (clear a) (clear b) (clear c) (arm-empty))**
- *Description of goal: i.e. set of worlds or* ***??***
  - E.g., Logical conjunction
  - Any world satisfying conjunction is a goal
  - **(and (on a b) (on b c)))**
- *Description of available actions*

Classical Planning



Compilation to SAT

- Init state
- Actions
- Goal

→ ?

The Idea

- Suppose a plan of length n exists
- Encode this hypothesis in SAT
  - Init state true at t_0
  - Goal true at T_n
  - Actions imply effects, etc
- Look for **satisfying** assignment
- Decode into plan

RISC: The Revolutionary Excitement

Axioms

Axiom	Description / Example
Init	The initial state holds at t=0
Goal	The goal holds at t=2n
A ⇒ P, E	Paint(A,Red,t) ⇒ Block(A, t-1) Paint(A,Red,t) ⇒ Color(A, Red, t+1)
Frame	
At-least-one	
Exclude	

Space of Encodings

- Action Representations
  - Regular
  - Simply-Split
  - Overloaded-Split
  - Bitwise
- Frame Axioms
  - Classical
  - Explanitory

Frame Axioms

- Classical
  - ∀P, A, t if P@t-1 ∧
  - A@t ∧
  - A doesn't affect P
  - then P@t+1
- Explanatory

Action Representation

Representation	One Propositional Variable per	Example
Regular	fully-instantiated action	Paint-A-Red, Paint-A-Blue, Move-A-Table
Simply-split	fully-instantiated action's argument	Paint-Arg1-A ∧ Paint-Arg2-Red
Overloaded-split	fully-instantiated argument	Act-Paint ∧ Arg1-A ∧ Arg2-Red
Bitwise	Binary encodings of actions	Bit1 ∧ ~Bit2 ∧ Bit3

more vars

Paint-A-Red = 5

more clses

Optimization 1: Factored Splitting

- use partially-instantiated actions

HasColor-A-Blue-(t-1) ^ Paint-Arg1-B-t ^
Paint-Arg2-Red-t ⇒ HasColor-A-Blue-(t+1)

Explanatory Frames

factored / unfactored	Simple	Overloaded
Variables	.46	.69
Clauses	.30	.50
Literals	.20	.38

Optimization 2: Types

A *type* is a fluent which no actions affects.
- type interference
- prune impossible operator instantiations
- type elimination

Type opts
No type opts

Literals	Regular	Simple	Overloaded	Bitwise
Classical	.27	.39	.34	.30
Explanatory	.10	.97	.67	.74