3 - Introduction to Planning

知识点 & 题目

The key problem is to select the action to do next. This is the so-called control problem. Three approaches to this problem:

- Programming-based: Specify control by hand
- Learning-based: Learn control from experience
- Model-based: Specify problem by hand, derive control automatically

Approaches not orthogonal; successes and limitations in each

Different models yield different types of controllers

Programming-Based Approach

Control specified by programmer

- Advantage: domain-knowledge easy to express
- Disadvantage: cannot deal with situations not anticipated by programmer

Learning-Based Approach

Learns a controller from experience or through simulation

- Unsupervised (Reinforcement Learning):
    - penalize Mario time that 'dies'
    - reward agent each time opponent 'dies' and level is finished
- Supervised (Classification)
    - learn to classify actions into good or bad from info provided by teacher
- Evolutionary
    - from pool of possible controllers: try them out, select the ones that do best, and mutate and recombine for a number of iterations, keeping best
- Advantage: does not require much knowledge in principle
- Disadvantage: in practice, hard to know which features to learn, and is slow

Model-Based Approach / General Problem Solving L3 P10

Specify model for problem: actions, initial situation, goals, and sensors

Let a solver compute controller automatically

Advantage:

- Powerful: In some applications generality is absolutely necessary

- Quick: Rapid prototyping. 10s lines of problem description vs. 1000s lines of C++ code. (Language generation)
- Flexible & Clear: Adapt/maintain the description
- Intelligent & domain-independent: Determines automatically how to solve complex problem effectively!

Disadvantage:

- Efficiency loss: Without any domain-specific knowledge about Chess, you don't beat Kasparov

Trade-off between 'automatic and general' vs. 'manual work but effective'

Model-based approach to intelligent behavior called Planning in AI

Different planning models

Classical Planning:

- finite and discrete state space S
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic transition function $s' = f(a, s)$ for $a \in A(s)$
- positive action costs $c(a, s)$

- A solution is a sequence of applicable actions that maps s0 into SG, and it is optimal if it minimizes sum of action costs (e.g., # of steps)
- Different models and controllers obtained by relaxing assumptions in blue

Conformant Planning:

- finite and discrete state space S
- a set of possible initial state $S_0 \in S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a non-deterministic transition function $F(a, s) \subseteq S$ for $a \in A(s)$
- uniform action costs $c(a, s)$

- A solution is still an action sequence but must achieve the goal **for any possible initial state and transition**.
- More complex than classical planning, verifying that a plan is conformant intractable in the worst case; but special case of planning with partial observability.

Planning with Markov Decision Processes: MDPs are fully observable, probabilistic state models

- a state space S
- initial state $s_0 \in S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$

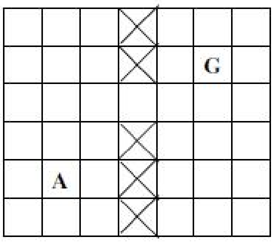
- Solutions are functions (policies) mapping states into actions
- Optimal solutions minimize expected cost to goal

Partially Observable MDPs: POMDPs are partially observable, probabilistic state models

- states $s \in S$
- actions $A(s) \subseteq A$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial belief state b_0
- final belief state b_f
- sensor model given by probabilities $P_a(o|s)$, $o \in Obs$

- Belief states are probability distributions over S
- Solutions are policies that map belief states into actions
- Optimal policies minimize expected cost to go from b0 to G

Agent **A** must reach **G**, moving one cell at a time in **known** map



- If actions deterministic and initial location known, planning problem is **classical**
- If actions stochastic and location observable, problem is an **MDP**
- If actions stochastic and location partially observable, problem is a **POMDP**

Different combinations of uncertainty and feedback: three problems, three models

Models, Languages, and Solvers

- A planner is a solver over a class of models; it takes a model description, and computes the corresponding controller
- Many models, many solution forms: uncertainty, feedback, costs, . . .
- Models described in suitable planning languages (Strips, PDDL, PPDDL, . . .) where states represent interpretations over the language.

A Basic Language for Classical Planning: STRIPS T4

- A **problem** in STRIPS is a tuple $P = \langle F, O, I, G \rangle$:
  - F stands for set of all atoms (boolean vars)
  - O stands for set of all operators (actions)
  - $I \subseteq F$ stands for initial situation
  - $G \subseteq F$ stands for goal situation
- Operators $o \in O$ **represented** by
  - the Add list $Add(o) \subseteq F$
  - the Delete list $Del(o) \subseteq F$
  - the Precondition list $Pre(o) \subseteq F$

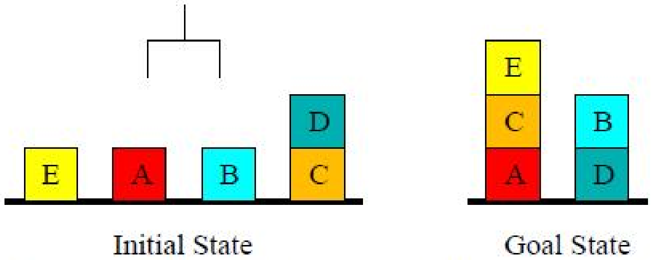
A STRIPS problem $P = \langle F, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are collections of atoms from F. $S = 2^F$
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

$\rightarrow$ (Optimal) **Solution** of P is (optimal) **solution** of $\mathcal{S}(P)$

$\rightarrow$ Slight language extensions often convenient: **negation**, **conditional effects**, **non-boolean variables**; some required for describing richer models (costs, probabilities, ...).

The Blocks world:



- Propositions: $on(x,y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \ldots, onTable(C), on(D,C), clear(D), armEmpty()\}$.
- Goal: $\{on(E,C), on(C,A), on(B,D)\}$.
- Actions: $stack(x,y)$, $unstack(x,y)$, $putdown(x)$, $pickup(x)$.
- $stack(x,y)$? $pre : \{holding(x), clear(y)\}$
  $add : \{on(x,y), armEmpty(), clear(x)\}$
  $del : \{holding(x), clear(y)\}$.

PDDL T4 A2

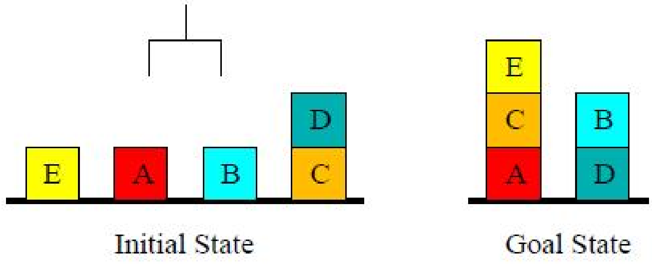
PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

A PDDL planning task comes in two pieces:

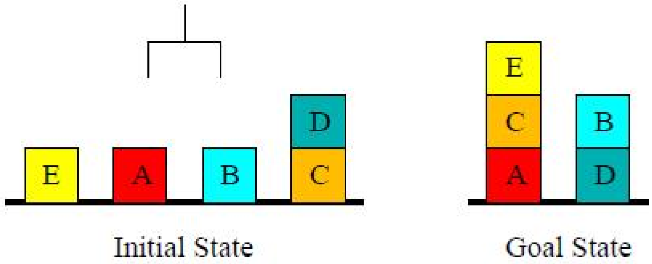
- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each benchmark domain has one domain file.

The Blocks World:



- Domain:

```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
...
```



- Problem:

```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

Satisficing vs. Optimal

- Satisficing planning
  - Input: A planning task P.
  - Output: A **plan** for P, or 'unsolvable' if no plan for P exists.
  - By **PlanEx**, we denote the problem of deciding, given a planning task P, whether or not there exists a plan for P.
- Optimal planning
  - Input: A planning task P.
  - Output: An **optimal plan** for P, or 'unsolvable' if no plan for P exist.
  - By **PlanLen**, we denote the problem of deciding, given a planning task P and an integer B, whether or not there exists a plan for P of length at most B.
- The techniques successful for either one of these are almost disjoint!
- Satisficing planning is much more effective in practice
- Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

- PlanEx and PlanLen are PSPACE-complete.
    - At least as hard as any other problem contained in PSPACE.
- In general, PlanEx and PlanLen have the same complexity.
- Within particular applications, bounded length plan existence is often harder than plan existence.
- This happens in many IPC benchmark domains: PlanLen is NP-complete while PlanEx is in P.
    - For example: Blocksworld and Logistics.
- In practice, optimal planning is (almost) never 'easy'

NP & PSPACE:

Def Turing machine: Works on a tape consisting of tape cells, across which its R/W head moves. The machine has internal states. There are transition rules specifying, given the current cell content and internal state, what the subsequent internal state will be, and whether the R/W head moves left or right or remains where it is. Some internal states are accepting ('yes'; else 'no').

Def NP: Decision problems for which there exists a non-deterministic Turing machine that runs in time polynomial in the size of its input. Accepts if at least one of the possible runs accepts.

Def PSPACE: Decision problems for which there exists a deterministic Turing machine that runs in space polynomial in the size of its input.

Relation: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE = NPSPACE**, and hence (trivially) **NP** *subset* **PSPACE**.

Computation

Key issue: exploit two roles of language

- specification: concise model description
- computation: reveal useful heuristic information (structure)

Two traditional approaches: search vs. decomposition

- explicit search of the state model S(P) direct but not effective til recently
- near decomposition of the planning problem thought a better idea

State of the Art in Classical Planning

- significant progress since Graphplan
- empirical methodology
    - standard PDDL language
    - planners and benchmarks available; competitions
    - focus on performance and scalability
- large problems solved (non-optimally)
- different formulations and ideas
    - Planning as Heuristic Search
    - Planning as SAT (Satisfiability)

   - Other: Local Search (LPG), Monte-Carlo Search, . . .

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems.
   - Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.
- Classical search problems require to find a path of actions leading from an initial state to a goal state.
   - They assume a single-agent, fully-observable, deterministic, static environment. Despite this, they are ubiquitous in practice.
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- Plan existence (bounded or not) is PSPACE-complete to decide for STRIPS.
- PDDL is the de-facto standard language for describing planning problems.

题目

Quiz

Question

If planners x, y both compete in IPC'YY, and x wins, is x 'better than' y?

(A): Yes. (B): No.

$\rightarrow$ Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a 'winner'! On other domains and/or according to other criteria, you may well be better off with the 'looser'.

$\rightarrow$ It's complicated, over-simplification is dangerous. (But, of course, nevertheless is being done all the time).

Question 4

1 / 1 pts

Conformant Planning has:

- A set of possible initial states and a probabilistic transition function

Correct!

- A set of possible initial states and a non-deterministic transition function
- A probability distribution over the initial states and a probabilistic transition function
- A probability distribution over the initial states and a non-deterministic transition function

Question 5

1 / 1 pts

POMDPs have:

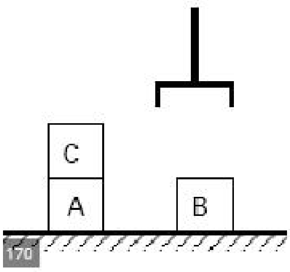
Correct!

- A sensor model given by probabilities drawn from observations about the environment
- A set of possible initial states
- A deterministic transition function
- A complexity that is identical to classical planning

Question 6

0 / 1 pts

Given the following initial state in a Blocks World problem, how many propositions are required to specify the initial state



You Answered

3

Correct Answers

6 (with margin: 0)

Correct Answers 0 (with margin: 0)

The following are required: onTable(A), on(C, A), onTable(B), clear(C), clear(B), armEmpty