IJCAI-11 Tutorial:

Advanced Introduction to Planning: Models and Methods

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
`http://www.dtic.upf.edu/~hgeffner`

References at the end . . .

Contents: General Idea

Planning is the **model-based approach** to autonomous behavior

Tutorial focuses on most common **planning models** and **algorithms**

- Classical Model; Classical Planning: *complete info, deterministic actions*
- Non-Classical Models ; Non-Classical Planning: *incomplete info, sensing, . . .*
  - ▷ Bottom-up Approaches: *Transformations into classical planning*
  - ▷ Top-down Approaches: *Native solvers for more expressive models*

More Precise Outline

1. Introduction to AI Planning
2. Classical Planning as Heuristic Search
3. Beyond Classical Planning: Transformations
   - *Soft goals, Incomplete Information, Plan Recognition*
4. Planning with Uncertainty: Markov Decision Processes (MDPs)
5. Planning with Incomplete Information: Partial Observable MDPs (POMDPs)
6. Open Problems and Challenges

Planning: Motivation

How to develop systems or 'agents'
that can make decisions on their own?

Example: Acting in Wumpus World (Russell and Norvig)

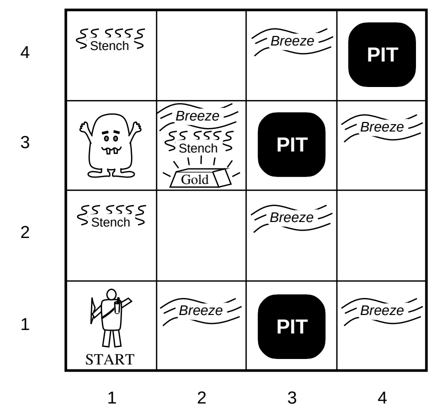
Wumpus World PEAS description

Performance measure
gold +1000, death -1000
-1 per step, -10 for using the arrow

Environment
Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Shooting kills wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up gold if in same square
Releasing drops the gold in same square

Actuators Left turn, Right turn,
Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell



Chapter 7 5

Autonomous Behavior in AI

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

Planning is the **model-based approach to autonomous behavior** where agent controller derived from model of the actions, sensors, and goals.

Different **models** yield different types of **controllers** . . .

Basic State Model: 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 s_0 into S_G, and it is **optimal** if it minimizes **sum of action costs** (e.g., # of steps)

Resulting controller is **open-loop**

Different **models** and **controllers** obtained by relaxing assumptions in **bold** . . .

Uncertainty but No Feedback: 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)

POMDPs are **partially observable, probabilistic** state models:

- states $s \in S$
- a set $G \subseteq S$ of goal states
- 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
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$

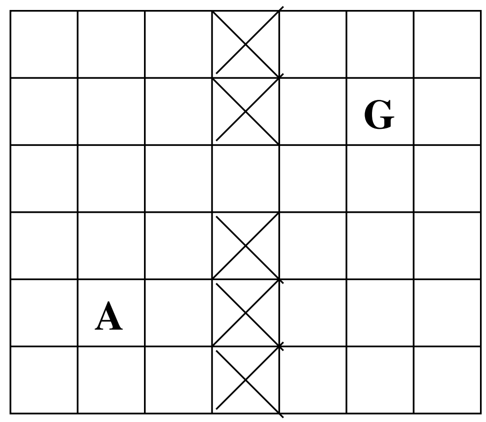
– **Belief states** are probability distributions over S

– **Solutions** are policies that map belief states into actions

– **Optimal** policies minimize **expected** cost to go from b_0 to G

Example

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

$$\textit{Model} \Longrightarrow \boxed{\textit{Planner}} \Longrightarrow \textit{Controller}$$

- 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

- 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 $Pre(o) \subseteq F$

From Language to Models

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
- 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

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

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

Example: Blocks in Strips (PDDL Syntax)

```
(define (domain BLOCKS)
  (:requirements :strips) ...
  (:action pick_up
          :parameters (?x)
          :precondition (and (clear ?x) (ontable ?x) (handempty))
          :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) ...)
  (:action put_down
           :parameters (?x)
           :precondition (holding ?x)
           :effect  (and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x)))
  (:action stack
          :parameters (?x ?y)
          :precondition (and (holding ?x) (clear ?y))
          :effect  (and (not (holding ?x)) (not (clear ?y)) (clear ?x)(handempty) ..

(define (problem BLOCKS_6_1)
   (:domain BLOCKS)
   (:objects F D C E B A)
   (:init (CLEAR A) (CLEAR B) ...  (ONTABLE B) ... (HANDEMPTY))
   (:goal (AND (ON E F) (ON F C) (ON C B) (ON B A) (ON A D))))
```

Example: Logistics in Strips PDDL

```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location  truck airplane - vehicle  vehicle packet - thing  ..)
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - location) ...)
  (:action load
    :parameters (?x - packet ?y - vehicle)
    :vars (?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location)
    :vars (?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at ?x ?z))
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  airplane1 - ...)
  (:init (at packet1 office1) (at packet2 office3) ...)
  (:goal (and (at packet1 office2) (at packet2 office2))))
```

Example: 15-Puzzle in PDDL

```
(define (domain tile)
 (:requirements :strips :typing :equality)
 (:types tile position)
 (:constants blank - tile)
 (:predicates (at ?t - tile ?x - position ?y - position)
       (inc ?p - position ?pp - position)
       (dec ?p - position ?pp - position))
 (:action move-up
   :parameters (?t - tile ?px - position ?py - position  ?bx - position ?by - ...)
   :precondition (and (= ?px ?bx) (dec ?by ?py) (not (= ?t blank)) ...)
   :effect (and (not (at blank ?bx ?by)) (not (at ?t ?px ?py)) (at blank ?px ?py) ..
   ...
(define (domain eight_tile) ..
  (:constants t1 t2 t3 t4 t5 t6 t7 t8 - tile    p1 p2 p3 - position)
  (:timeless (inc p1 p2) (inc p2 p3) (dec p3 p2) (dec p2 p1)))

(define (situation eight_standard)
  (:domain eight_tile)
  (:init (at blank p1 p1) (at t1 p2 p1) (at t2 p3 p1) (at t3 p1 p2) ..)
  (:goal (and (at t8 p1 p1) (at t7 p2 p1) (at t6 p3 p1) ..)
```

Next

- Solving classical planning problems
- Using classical planners for non-classical tasks
- Other models and solvers . . .

Computational Approaches to Classical Planning

- **General Problem Solver (GPS) and Strips** (50's-70's): mean-ends analysis, decomposition, regression, . . .
- **Partial Order (POCL) Planning** (80's): work on any open subgoal, resolve threats; UCPOP 1992
- **Graphplan** (1995 – 2000): build graph containing all possible **parallel** plans up to certain length; then extract plan by searching the graph backward from Goal
- **SATPlan** (1996 – . . .): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- **Heuristic Search Planning** (1996 – . . .): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P
- **Model Checking Planning** (1998 – . . .): search state space $\mathcal{S}(P)$ with 'symbolic' Breadth first search where sets of states represented by formulas implemented by BDDs . . .

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**
  1. Planning as **Heuristic Search**
  2. Planning as **SAT**
  3. **Other:** Local Search (LPG), Monte-Carlo Search (Arvand), . . .

I'll focus on **1** (see IJCAI-11 Tutorial on Classical Planning by Jussi Rintanen for more complete overview)

Recall: Problem P into State Model $S(P)$

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
- 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

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

– Thus P can be solved by solving $\mathcal{S}(P)$

Solving P by solving $\mathcal{S}(P)$

Search algorithms for planning exploit the **correspondence** between **(classical) states model** $\mathcal{S}(P)$ and **directed graphs**:

- The **nodes** of the graph represent the **states** s in the model
- The edges (s, s') capture corresponding transition in the model with same cost

In the **planning as heuristic search** formulation, the problem P is solved by **path-finding** algorithms over the **graph** associated with model $\mathcal{S}(P)$

Search Algorithms for Path Finding in Directed Graphs

- **Blind search/Brute force algorithms**
  - ▷ Goal plays **passive** role in the search
    e.g., Depth First Search (DFS), Breadth-first search (BrFS), Uniform Cost (Dijkstra), Iterative Deepening (ID)

- **Informed/Heuristic Search Algorithms**
  - ▷ Goals plays **active** role in the search through **heuristic function** $h(s)$ that estimates cost from s to the goal
    e.g., A, IDA*, Hill Climbing, Best First, WA*, DFS B&B, LRTA*, . . .*

Properties of Search Algorithms

- **Completeness**: whether guaranteed to find solution
- **Optimality**: whether solution guaranteed to be optimal
- **Time Complexity**: how time increases with size
- **Space Complexity:** how space increases with size

	DFS	BrFS	ID	A*	HC	IDA*	B&B
Complete	No	Yes	Yes	Yes	No	Yes	Yes
Optimal	No	Yes*	Yes	Yes	No	Yes	Yes
Time	∞	b^d	b^d	b^d	∞	b^d	b^D
Space	$b \cdot d$	b^d	$b \cdot d$	b^d	b	$b \cdot d$	$b \cdot d$

– Parameters: d is solution depth; b is branching factor

– Breadth First Search (BrFS) optimal when costs are uniform

– A*/IDA* optimal when h is **admissible**; $h \leq h^*$

Learning Real Time A* (LRTA*)

- LRTA* is a very interesting **real-time** search algorithm
- It's like a **hill-climb** or **greedy** search that **updates** the heuristic V as it moves along, starting with $V = h$.

1. **Evaluate** each action a in s as: $Q(a, s) = c(a, s) + V(s')$
2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Exit** if s' is goal, else go to 1 with $s := s'$

- Two remarkable **properties**
  - ▷ **Each trial** of LRTA* gets eventually to the goal if space connected
  - ▷ **Repeated trials** eventually get to the goal **optimally**, if h **admissible**!
- In addition, simple change in line 1 yields **RTDP** that solves **MDPs**!

Heuristic Search Planning

- Explicitly **searches** graph associated with model $S(P)$ with **heuristic** $h(s)$ that estimates cost from s to goal
- **Key idea:** Heuristic h extracted **automatically** from problem P

This is the mainstream approach in classical planning (and other forms of planning as well), enabling the solution of problems over **very large spaces**

Heuristics: where they come from?

- General idea: heuristic functions obtained as **optimal cost functions** of **relaxed problems**

- Examples:

  – *Manhattan distance in N-puzzle*
  – *Euclidean Distance in Routing Finding*
  – *Spanning Tree in Traveling Salesman Problem*
  – *Shortest Path in Job Shop Scheduling*

- Yet

  – how to get and solve suitable relaxations?
  – how to get heuristics automatically?

Heuristics for Classical Planning

- Key development in planning in the 90's is automatic extraction of informative heuristic functions **from the problem P itself**

- Most common relaxation in planning, P^+, obtained by dropping **delete-lists** from ops in P. If $c^*(P)$ is optimal cost of P, then heuristic set to

$$h^+(P) \stackrel{\text{def}}{=} c^*(P^+)$$

- Heuristic h^+ **intractable** but easy to **approximate**; i.e.

  ▷ *computing* **optimal** *plan for P^+ is* **intractable**, but
  ▷ *computing a non-optimal plan for P^+ (***relaxed plan***) easy*

- While this relaxation is 10-15 years old by now, it still provides heuristics for state-of-the-art planners such as LAMA-2011 winner of 2011 IPC

Additive Heuristic

- For all **atoms** p, if $O(p)$ denotes actions that add p:

$$h(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s\text{, else} \\ \min_{a \in O(p)} \left[cost(a) + h(Pre(a); s) \right] \end{cases}$$

- For **sets** of atoms C, assume **independence**:

$$h(C; s) \stackrel{\text{def}}{=} \sum_{r \in C} h(r; s)$$

- Resulting **heuristic function** $h_{add}(s)$:

$$h_{add}(s) \stackrel{\text{def}}{=} h(Goals; s)$$

Heuristic not admissible but informative and fast

Max Heuristic

- For all **atoms** p,

$$h(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s\text{, else} \\ \min_{a \in O(p)} \left[cost(a) + h(Pre(a); s) \right] \end{cases}$$

- For **sets** of atoms C, replace **sum** by **max**

$$h(C; s) \stackrel{\text{def}}{=} \max_{r \in C} h(r; s)$$

- Resulting **heuristic function** $h_{max}(s)$:

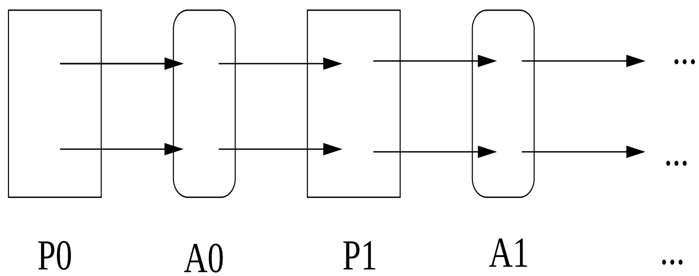
$$h_{max}(s) \stackrel{\text{def}}{=} h(Goals; s)$$

Heuristic admissible but not very informative . . .

Max Heuristic and (Relaxed) Planning Graph

- Build reachability graph P_0, A_0, P_1, A_1, ...

$$
\begin{aligned}
P_0 &= \{p \in s\} \\
A_i &= \{a \in O \mid Pre(a) \subseteq P_i\} \\
P_{i+1} &= P_i \cup \{p \in Add(a) \mid a \in A_i\}
\end{aligned}
$$



– Graph implicitly **represents** max heuristic when $cost(a) = 1$:

$$h_{max}(s) = \text{min } i \text{ such that } G \subseteq P_i$$

Heuristics, Relaxed Plans, and FF

- (Relaxed) Plans for P^+ can be obtained from **additive** or **max** heuristics by recursively collecting **best supports** backwards from goal, where a_p is **best support** for p in s if $p \notin s$ and

$$a_p \;=\; \text{argmin}_{a \in O(p)}[cost(a) + h(Pre(a))]$$

- A plan $\pi(p; s)$ for p in delete-relaxation can then be computed backwards as

$$\pi(p; s) = \begin{cases} \emptyset & \text{if } p \in s \\ \{a_p\} \cup \cup_{q \in Pre(a_p)} \pi(q; s) & \text{otherwise} \end{cases}$$

- In FF, the **relaxed plan** obtained using $h = h_{max}$ as

$$\pi(s) = \cup_{p \in Goals} \pi(p; s)$$

- Heuristic then used in FF is not h_{max} but more informed

$$h_{\text{FF}}(s) = |\pi(s)|$$

State-of-the-art Planners: EHC Search, Helpful Actions, Landmarks

- In original formulation of **planning as heuristic search**, the states s and the heuristics $h(s)$ are **black boxes** used in **standard search algorithms**

- More recent planners like **FF** and **LAMA** go beyond this, exploiting the structure of the heuristic and/or problem further:

  ▷ **Helpful Actions (HA)**: *critical for large branching factors*
  ▷ **Landmarks**: *provide subgoaling and serialization when goals 'in conflict'*

- They also use novel search algorithms

  ▷ **Enforced Hill Climbing (EHC)**: *incomplete but effective search, uses HA*
  ▷ **Multi-Queue Best First Search**: *alternative way to use HA*

- As a result, they can often solve **huge problems**, **very fast**; much better than **just plugging a heuristic into standard search algorithm**

Classical Planning: Status

- The good news: **classical planning works reasonably well**
  - ▷ *Large problems solved very fast (non-optimally)*
- **Model simple but useful**
  - ▷ *Operators not primitive; can be policies themselves*
  - ▷ *Fast closed-loop replanning able to cope with uncertainty sometimes*
- Not so good; **limitations:**
  - ▷ *Does not model* **Uncertainty** *(no probabilities)*
  - ▷ *Does not deal with* **Incomplete Information** *(no sensing)*
  - ▷ *Does not accommodate* **Preferences** *(simple cost structure)*
  - ▷ . . .

Beyond Classical Planning: Two Strategies

- **Top-down:** Develop solver for **more general class of models;** e.g., Markov Decision Processes (MDPs), Partial Observable MDPs (POMDPs), . . .

  $+$: generality
  $-$: complexity

- **Bottom-up:** Extend the scope of **current 'classical' solvers**

  $+$: efficiency
  $-$: generality

- We'll do both, starting with **transformations** for

  ▷ compiling **soft goals** away (planning with preferences)
  ▷ compiling **uncertainty** away (incomplete information)
  ▷ doing **plan recognition** (as opposed to plan generation)

Compilation of Soft Goals

- Planning with **soft goals** aimed at plans π that maximize **utility**

$$u(\pi) = \sum_{p \in do(\pi, s_0)} u(p) \quad - \quad \sum_{a \in \pi} c(a)$$

- Actions have **cost** $c(a)$, and soft goals **utility** $u(p)$

- Best plans achieve best **tradeoff** between **action costs** and **utilities**

- Model used in recent planning competitions; **net-benefit track** 2008 IPC

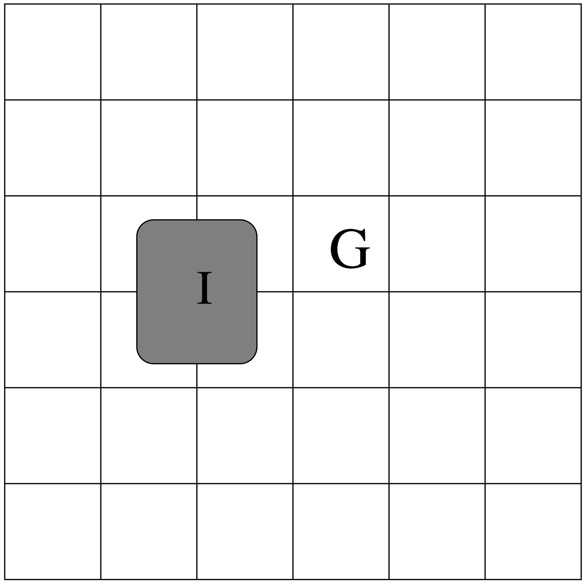
- Yet it turns that soft goals **do not** add expressive power, and can be **compiled away**

Compilation of Soft Goals (cont'd)

- For each soft goal p, create **new hard goal** p' initially false, and **two new actions**:
  - ▷ $collect(p)$ with precondition p, effect p' and **cost** 0, and
  - ▷ $forgo(p)$ with an empty precondition, effect p' and **cost** $u(p)$
- Plans π maximize $u(\pi)$ iff minimize $c(\pi) = \sum_{a \in \pi} c(a)$ in resulting problem
- Compilation yields better results that native soft goal planners in 2008 IPC

	IPC6 Net-Benefit Track			Compiled Problems			
Domain	Gamer	HSP^*_P	Mips-XXL	Gamer	HSP^*_F	HSP^*_0	Mips-XXL
crewplanning(30)	4	16	8	-	8	**21**	8
elevators (30)	11	5	4	**18**	8	8	3
openstacks (30)	**7**	5	2	6	4	6	1
pegsol (30)	24	0	23	22	**26**	14	22
transport (30)	12	12	9	-	**15**	**15**	9
woodworking (30)	13	11	9	-	**23**	22	7
total	71	49	55		84	**86**	50

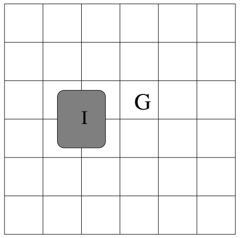
Incomplete Information: Conformant Planning



Problem: A robot must move from an **uncertain** I into G with **certainty**, one cell at a time, in a grid nxn

- Problem very much like a classical planning problem except for **uncertain** I
- Plans, however, quite different: best **conformant plan must move the robot to a corner first (localization)**

Conformant Planning: Belief State Formulation



- call a **set** of possible states, a **belief state**
- actions then map a belief state b into a bel state $b_a = \{s' \mid s' \in F(a,s) \;\&\; s \in b\}$
- **conformant problem** becomes a path-finding problem in **belief space**

Problem: number of belief state is **doubly exponential** in number of variables.

– **effective representation** of belief states b

– **effective heuristic** $h(b)$ for estimating cost in belief space

Recent alternative: translate into classical planning . . .

Basic Translation: Move to the 'Knowledge Level'

Given **conformant problem** $P = \langle F, O, I, G \rangle$

- F stands for the fluents in P
- O for the operators with effects $C \rightarrow L$
- I for the initial situation (**clauses** over F-literals)
- G for the goal situation (set of F-literals)

Define **classical problem** $K_0(P) = \langle F', O', I', G' \rangle$ as

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL \mid$ clause $L \in I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ but preconds L replaced by KL, and effects $C \rightarrow L$ replaced by $KC \rightarrow KL$ (**supports**) and $\neg K \neg C \rightarrow \neg K \neg L$ (**cancellation**)

$K_0(P)$ is **sound** but **incomplete**: every classical plan that solves $K_0(P)$ is a conformant plan for P, but not vice versa.

Key elements in Complete Translation $K_{T,M}(P)$

- A set T of **tags** t: consistent sets of **assumptions** (literals) about the **initial situation** I

$$I \not\models \neg t$$

- A set M of **merges** m: **valid subsets of tags** ($=$ DNF)

$$I \models \bigvee_{t \in m} t$$

- **New (tagged) literals** KL/t meaning that L **is true if** t **true initially**

A More General Translation $K_{T,M}(P)$

Given **conformant problem** $P = \langle F, O, I, G \rangle$

- F stands for the fluents in P
- O for the operators with effects $C \rightarrow L$
- I for the initial situation (**clauses** over F-literals)
- G for the goal situation (set of F-literals)

define **classical problem** $K_{T,M}(P) = \langle F', O', I', G' \rangle$ as

- $F' = \{KL/t\ ,\ K\neg L/t \mid L \in F$ and $t \in T\}$
- $I' = \{KL/t \mid$ if $I \models t \supset L\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ but preconds L replaced by KL, and effects $C \rightarrow L$ replaced by $KC/t \rightarrow KL/t$ (**supports**) and $\neg K\neg C/t \rightarrow \neg K\neg L/t$ (**cancellation**), and **new merge actions**

$$\bigwedge_{t \in m, m \in M} KL/t \ \rightarrow \ KL$$

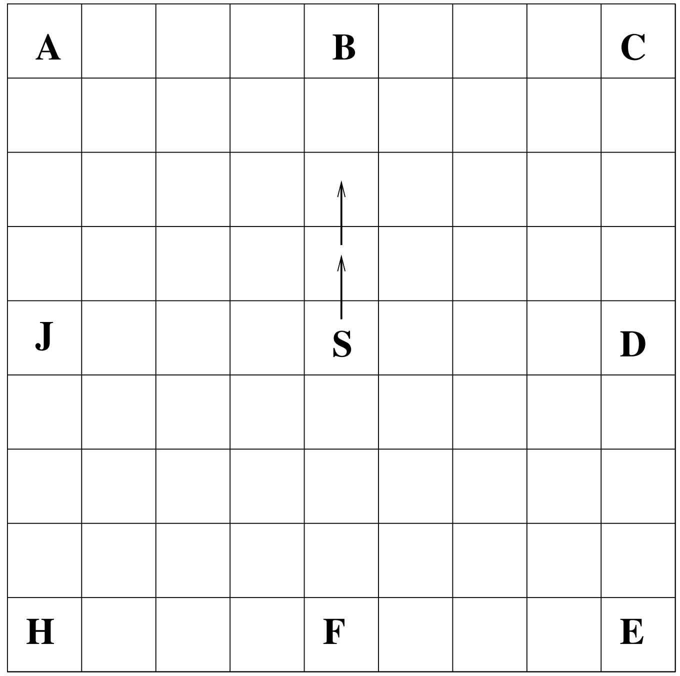
The two **parameters** T and M are the set of **tags** (assumptions) and the set of **merges** (valid sets of assumptions) . . .

Compiling Uncertainty Away: Properties

- General translation scheme $K_{T,M}(P)$ is always **sound**, and for suitable choice of the sets of **tags** and **merges**, it is **complete**.
- $K_{S0}(P)$ is **complete instance** of $K_{T,M}(P)$ obtained by setting T to the set of **possible initial states** of P
- $K_i(P)$ is a **polynomial instance** of $K_{T,M}(P)$ that is **complete** for problems with **width** bounded by i.
  - ▷ *Merges for each L in $K_i(P)$ chosen to* **satisfy** i *clauses in I relevant to L*
- The **width** of many benchmarks **bounded** and equal 1!
- Such problems can be solved with a **classical planner** after a **low poly** translation

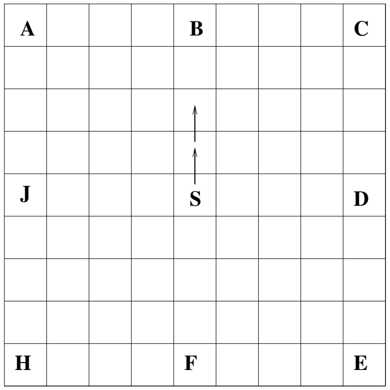
Translation extended to planning with partial observability and provides basis for state-of-the-art approaches . . .

Plan Recognition



- Agent can **move** one unit in the four directions
- Possible **targets** are A, B, C, . . .
- Starting in S, he is **observed** to move up twice
- **Where** is he going? Why?

Example (cont'd)



- From Bayes, **goal posterior** is $P(G|O) = \alpha\, P(O|G)\, P(G)$, $G \in \mathcal{G}$
- If **priors** $P(G)$ given for each goal in $\mathcal{G}$, the question is what is $P(O|G)$
- $P(O|G)$ measures **how well goal** G **predicts observed actions** O
- In **classical** setting,
  - ▷ G predicts O **best** when need to get off the way **not** to comply with O
  - ▷ G predicts O **worst** when need to get off the way **to comply with** O

Posterior Probabilities from Plan Costs

- From Bayes, **goal posterior** is $P(G|O) = \alpha\, P(O|G)\, P(G)$,
- If **priors** $P(G)$ given, set $P(O|G)$ to

$$\mathbf{function}(c(G+\overline{O})\,-\,c(G+O))$$

  - ▷ $c(G+O)$: cost of achieving G **while complying with** O
  - ▷ $c(G+\overline{O})$: cost of achieving G **while not complying with** O

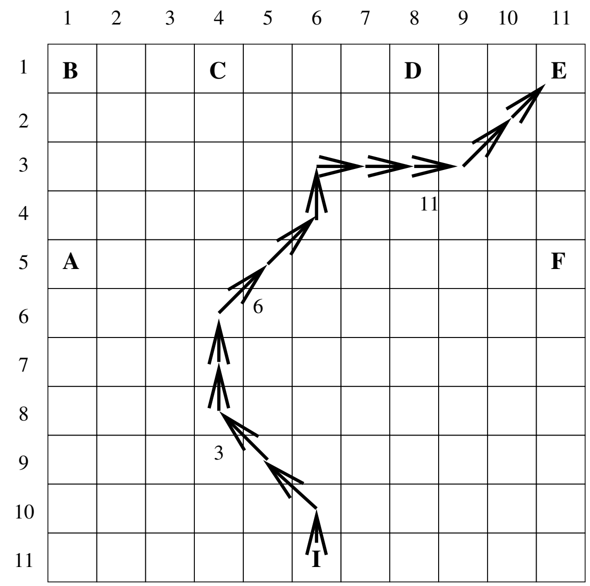
– Costs $c(G+O)$ and $c(G+\overline{O})$ **computed** by **classical planner**

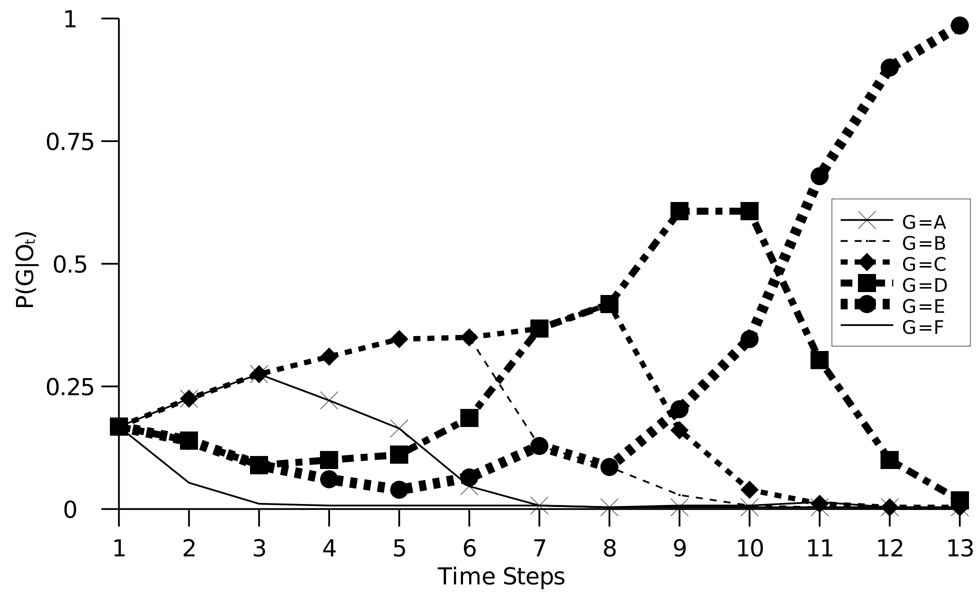
– Goals of **complying** and **not complying** with O translated into normal goals

– **Function** of cost difference set to **sigmoid**; follows from assuming $P(O|G)$ and $P(\overline{O}|G)$ are Boltzmann distributions $P(O|G) = \alpha'\, exp\{-\beta\, c(G,O)\}$, . . .

– Result is that posterior probabilities $P(G|O)$ **computed in** $2|\mathcal{G}|$ **classical planner calls**, where $\mathcal{G}$ is the set of possible goals

Illustration: Noisy Walk





Graph on left shows 'noisy walk' and possible targets; curves on right show resulting **posterior probabilities** $P(G|O)$ of each possible target G as a function of time

Approach to plan recognition can be generalized to other models (MDPs, POMDPs); the idea is that if you have a **planner** for a model, then you also have a **plan recognizer** for that model given a **pool of possible goals.**

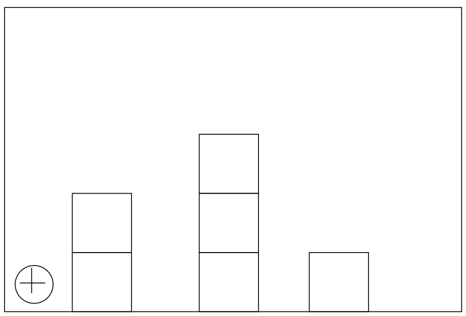
Summary: Transformations into Classical Planning

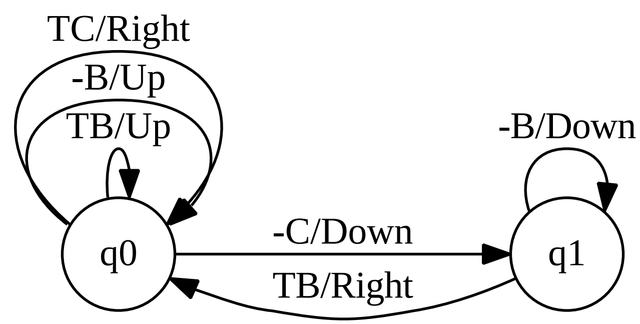
- **Classical Planning** solved as **path-finding** in state space
  - ▷ Most used techniques are **heuristic search** and **SAT**
- **Beyond classical planning:** two approaches
  - ▷ **Top-down:** solvers for richer models like MDPs and POMDPs (Next)
  - ▷ **Bottom-up:** compile non-classical features away
- We have followed second approach with **transformations** to
  - ▷ eliminate **soft goals** when planning with preferences
  - ▷ eliminate **uncertainty** in conformant planning
  - ▷ do **plan recognition** rather than plan generation
- Other transformations used for compiling away **sensing**, **LTL plan constraints**, **control knowledge**, **HTN hierarchies**, etc.

Transformations; Further illustration: Finite State Controllers

- **Problem** P: find **green block** using visual-marker (circle) that can move around one cell at a time (à la Chapman and Ballard)

- **Observables:** Whether cell marked contains a green block (G), non-green block (B), or neither (C); and whether on table (T) or not (–)





- Controller on the right **solves** the problem, and not only that, it's **compact** and **general**: it applies to **any number of blocks** and **any configuration**!

Controller obtained by running a **classical planner** over **transformed problem**

Last Part: MDP and POMDP Planning

- A **planner** is a **solver over a class of models;** it takes a model description, and computes the corresponding controller

$$\textit{Model} \Longrightarrow \boxed{\textit{Planner}} \Longrightarrow \textit{Controller}$$

- We focus next on models that yield **closed-loop controllers**, where next action depends on previous **observations**

Planning with Markov Decision Processes: Goal MDPs

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** from s_0 to goal

Discounted Reward Markov Decision Processes

A more common formulation of MDPs . . .

- a state space S
- initial state $s_0 \in S$
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- **rewards** $r(a, s)$ positive or negative
- a **discount factor** $0 < \gamma < 1$; **there is no goal**

– **Solutions** are **functions (policies)** mapping states into actions

– **Optimal** solutions max **expected discounted accumulated reward** from s_0

Partially Observable MDPs: Goal POMDPs

POMDPs are **partially observable, probabilistic** state models:

- states $s \in S$
- set of goal states $G \subseteq S$
- 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
- action costs $c(a, s) > 0$
- **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 b_0 to G

Discounted Reward POMDPs

Alternative common formulation of POMDPs:

- 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
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$
- **rewards** $r(a,s)$ positive or negative
- **discount factor** $0 < \gamma < 1$; **there is no goal**

– **Solutions** are **policies** mapping states into actions

– **Optimal** solutions max **expected discounted accumulated reward** from b_0

Expected Cost/Reward of Policy (MDPs)

- In Goal MDPs, **expected cost of policy π starting in s**, denoted as $V^\pi(s)$, is

$$V^\pi(s) = E_\pi[\sum_i c(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i)\,]$$

  where expectation is **weighted sum** of **cost** of possible state trajectories **times** their **probability** given π

- In Discounted Reward MDPs, **expected discounted reward from s** is

$$V^\pi(s) = E_\pi[\sum_i \gamma^i\, r(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i)]$$

Goal states assumed absorbing, cost-free, and observable . . .

MDPs/POMDPs: Themes and Variations

- Goal MDPs and POMDPs **more expressive** than Discounted MDPs and POMDPs, in spite of restriction on costs and goals
- **Probabilities** not that critical though; **qualitative** MDPs and POMDPs where **probabilities** replaced by **sets**, and **expected cost** by **cost in worst case** also useful and challenging
- **Contingent Planning** or **Planning with Partial Observability** refer to Qualitative POMDPs
- We focus on **full solutions** to these problems, or what's called **off-line planning**
- Full solutions, however, not strictly required in **on-line planning** where **action selection mechanism** often suffices . . .

Computation: Solving MDPs

Conditions that ensure **existence** of optimal policies and **termination** of some of the methods we'll see:

- For **discounted MDPs**, $0 < \gamma < 1$, none needed as everything is bounded; e.g. discounted cumulative reward no greater than $C/1 - \gamma$, if $r(a, s) \leq C$ for all a, s

- For **goal MDPs**, absence of **dead-ends** assumed so that $V^*(s) \neq \infty$ for all s

Basic Dynamic Programming Methods: Value Iteration (1)

- **Greedy policy** π_V for $V = V^*$ is **optimal**:

$$\pi_V(s) = \text{arg min}_{a \in A(s)}[c(s,a) + \sum_{s' \in S} P_a(s'|s)V(s')]$$

- Optimal V^* is unique solution to **Bellman's optimality equation** for MDPs

$$V(s) = \min_{a \in A(s)}[c(s,a) + \sum_{s' \in S} P_a(s'|s)V(s')]$$

where $V(s) = 0$ for goal states s

- For **discounted reward MDPs**, Bellman equation is

$$V(s) = \max_{a \in A(s)}[r(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)V(s')]$$

Basic DP Methods: Value Iteration (2)

- **Value Iteration** finds V^* solving Bellman eq. by **iterative procedure:**
  - ▷ Set V_0 to arbitrary value function; e.g., $V_0(s) = 0$ for all s
  - ▷ Set V_{i+1} to result of Bellman's **right hand side** using V_i in place of V:

$$V_{i+1}(s) := \min_{a \in A(s)} [c(s,a) + \sum_{s' \in S} P_a(s'|s) V_i(s')]$$

- $V_i \mapsto V^*$ as $i \mapsto \infty$
- $V_0(s)$ must be initialized to 0 for all goal states s

(Parallel) Value Iteration and Asynchronous Value Iteration

- Value Iteration (VI) converges to **optimal value function** V^* asympotically
- Bellman eq. for **discounted reward** MDPs similar, but with **max** instead of **min**, and sum multiplied by γ
- In practice, VI stopped when **residual** $R = \max_s |V_{i+1}(s) - V_i(s)|$ is small enough
- Resulting greedy policy π_V has **loss** bounded by $2\gamma R/1-\gamma$
- **Asynchronous Value Iteration** is **asynchronous** version of VI, where states **updated in any order**
- Asynchronous VI also converges to V^* when **all states updated infinitely often**; it can be **implemented** with single V vector

Policy Evaluation

- **Expected cost** of policy π from s to goal, $V^{\pi}(s)$, is weighted avg of **cost** of **state trajectories** $\tau : s_0, s_1, \ldots,$ times their **probability** given π

  trajectory cost is $\sum_{i=0,\infty} cost(\pi(s_i), s_i)$ and *probability* $\prod_{i=0,\infty} P_{\pi(s_i)}(s_{i+1}|s_i)$

- Expected costs $V^{\pi}(s)$ can also be characterized as solution to Bellman equation

$$V^{\pi}(s) \;=\; c(a,s) \,+\, \sum_{s' \in S} P_a(s'|s) V^{\pi}(s')$$

  where $a = \pi(s)$, and $V^{\pi}(s) = 0$ for goal states

- This set of **linear equations** can be solved analytically, or by VI-like procedure

- **Optimal expected cost** $V^*(s)$ is $\min_{\pi} V^{\pi}(s)$ and **optimal policy** is the $\arg\min$

- For **discounted reward** MDPs, all similar but with $r(s,a)$ instead of $c(a,s)$, max instead of min, and sum discounted by γ

Policy Iteration

- Let $Q^\pi(a,s)$ be **expected cost** from s when doing a first and then π

$$Q^\pi(a,s) \;=\; c(a,s) \;+\; \sum_{s' \in S} P_a(s'|s) V^\pi(s')$$

- When $Q^\pi(a,s) < Q^\pi(\pi(s),s)$, π **strictly improved** by changing $\pi(s)$ to a

- **Policy Iteration (PI)** computes π^* by seq. of **evaluations** and **improvements**

  1. Starting with arbitrary policy π
  2. Compute $V^\pi(s)$ for all s (**evaluation**)
  3. Improve π by setting $\pi(s)$ to $a = \arg\min_{a \in A(s)} Q^\pi(a,s)$ (**improvement**)
  4. If π changed in 3, go back to 2, else **finish**

- PI finishes with π^* after **finite** number of iterations, as # of policies is **finite**

Dynamic Programming: The Curse of Dimensionality

- **VI** and **PI** need to deal with value vectors V of size $|S|$

- **Linear programming** can also be used to get V^* but $O(|A||S|)$ constraints:

$$\max_V \sum_s V(s) \text{ subject to } V(s) \leq c(a,s) + \sum_{s'} P_a(s'|s)V(s') \text{ for all } a, s$$

  with $V(s) = 0$ for goal states

- MDP problem is thus **polynomial** in S but **exponential** in $\#$ vars

- Moreover, **this is not worst case**; vectors of size $|S|$ needed **to get started!**

Question: Can we do better?

Dynamic Programming and Heuristic Search

- **Heuristic search** algorithms like A* and IDA* manage to solve **optimally** problems with more than 10^{20} states, like Rubik's Cube and the 15-puzzle
- For this, **admissible heuristics** (lower bounds) used to **focus/prune** search
- Can admissible heuristics be used for **focusing updates** in DP methods?
- Often states **reachable** with **optimal policy** from s_0 much smaller than S
- Then convergence to V^* **over all** s not needed for **optimality** from given s_0

Theorem 1. *If V is an* **admissible** *value function s.t. the* **residuals** *over the states reachable with π_V from s_0 are all zero, then π_V is an* **optimal policy** *from s_0 (i.e. it minimizes $V^\pi(s_0)$)*

Learning Real Time A* (LRTA*) Revisited

1. **Evaluate** each action a in s as: $Q(a, s) = c(a, s) + V(s')$
2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Exit** if s' is goal, else go to 1 with $s := s'$

- LRTA* can be seen as **asynchronous value iteration** algorithm for **deterministic** actions that takes advantage of theorem above (i.e. updates $=$ DP updates)

- **Convergence** of LRTA* to V implies residuals along π_V reachable states from s_0 are all zero

- Then 1) $V = V^*$ along such states, 2) $\pi_V = \pi^*$ from s_0, but 3) $V \neq V^*$ and $\pi_V \neq \pi^*$ over other states; yet this is irrelevant given s_0

Real Time Dynamic Programming (RTDP) for MDPs

RTDP is a generalization of LRTA* to MDPs that adapts Bellman equation used in the **Eval** step

1. **Evaluate** each action a applicable in s as

$$Q(a, s) = c(a, s) + \sum_{s' \in S} P_a(s'|s)V(s')$$

2. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$
3. **Update** $V(s)$ to $Q(\mathbf{a}, s)$
4. **Observe** resulting state s'
5. **Exit** if s' is goal, else go to 1 with $s := s'$

Same properties as LRTA* but over MDPs: **after repeated trials**, greedy policy eventually becomes **optimal** if $V(s)$ initialized to admissible $h(s)$

Find-and-Revise: A General DP + HS Scheme

- Let $Res_V(s)$ be **residual** for s given **admissible** value function V

- **Optimal** π for MDPs from s_0 can be obtained for sufficiently small $\epsilon > 0$:

  1. **Start** with admissible V; i.e. $V \leq V^*$
  2. **Repeat:** find s reachable from π_V & s_0 with $Res_V(s) > \epsilon$, and **Update** it
  3. **Until** no such states left

- V remains **admissible** (**lower bound**) after updates

- **Number of iterations** until convergence bounded by $\sum_{s \in S}[V^*(s) - V(s)]/\epsilon$

- Like in **heuristic search**, convergence achieved **without visiting or updating** many of the states in S; LRTDP, LAO*, ILAO*, HDP, LDFS, etc. are algorithms of this type

POMDPs are MDPs over Belief Space

- Beliefs b are **probability distributions** over S
- An action $a \in A(b)$ maps b into b_a

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s')$$

- The probability of observing o then is:

$$b_a(o) = \sum_{s \in S} P_a(o|s)b_a(s)$$

- . . . and the new belief is

$$b_a^o(s) = P_a(o|s)b_a(s)/b_a(o)$$

RTDP for POMDPs

Since POMDPs are MDPs over belief space algorithm for POMDPs becomes

1. **Evaluate** each action a applicable in b as

$$Q(a,b) = c(a,b) + \sum_{o \in O} b_a(o) V(b_a^o)$$

2. **Apply** action $\mathbf{a}$ that minimizes $Q(a,b)$
3. **Update** $V(b)$ to $Q(\mathbf{a},b)$
4. **Observe** o
5. **Compute** new belief state b_a^o
6. **Exit** if b_a^o is a final belief state, else set b to b_a^o and go to 1

- Resulting algorithm, called **RTDP-Bel**, discretizes beliefs b for writing to and reading from hash table
- **Point-based POMDP methods** do not discretize beliefs, using instead a **finite representation** of value function over dense set of all beliefs
- Both class of methods shown to solve POMDPs with tens of thousands of states

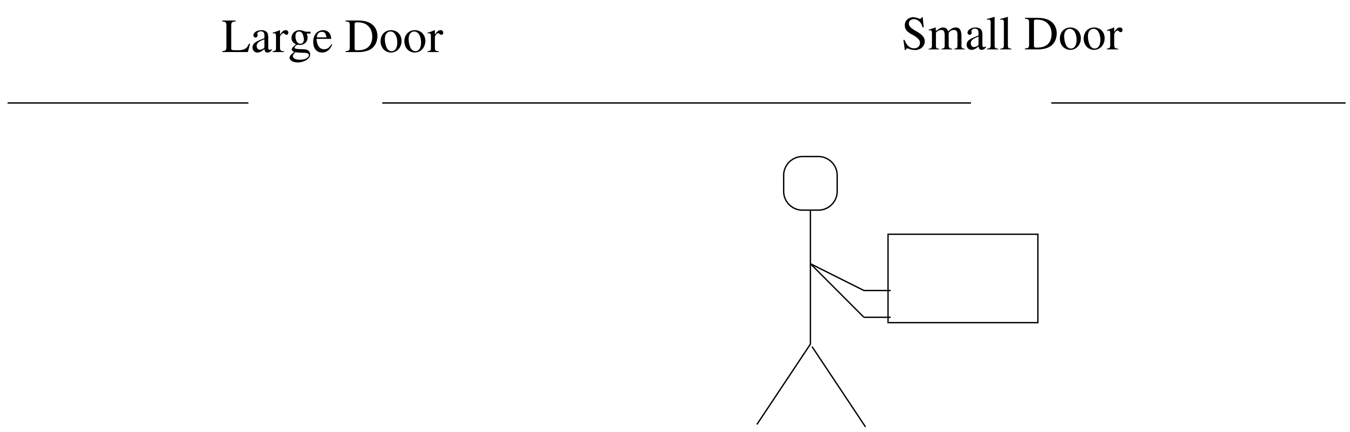
Summary: MDP and POMDP Planning

- Two approaches for dealing with **uncertainty** and **sensing**:
  - ▷ **Bottom-up:** compile uncertainty/sensing away and use classical planners
  - ▷ **Top-down:** develop native solvers for more expressive models; e.g. MDPs and POMDPs
- Methods for MDP and POMDP planning include
  - ▷ Standard **dynamic programming** methods like **value** and **policy iteration**
  - ▷ **Heuristic search** DP methods like **RTDP**, **LAO***, and **HSVI**
  - ▷ Adaptive Sparse **Lookahead Methods** such as **UCT** . . .

Challenges and Open Problems

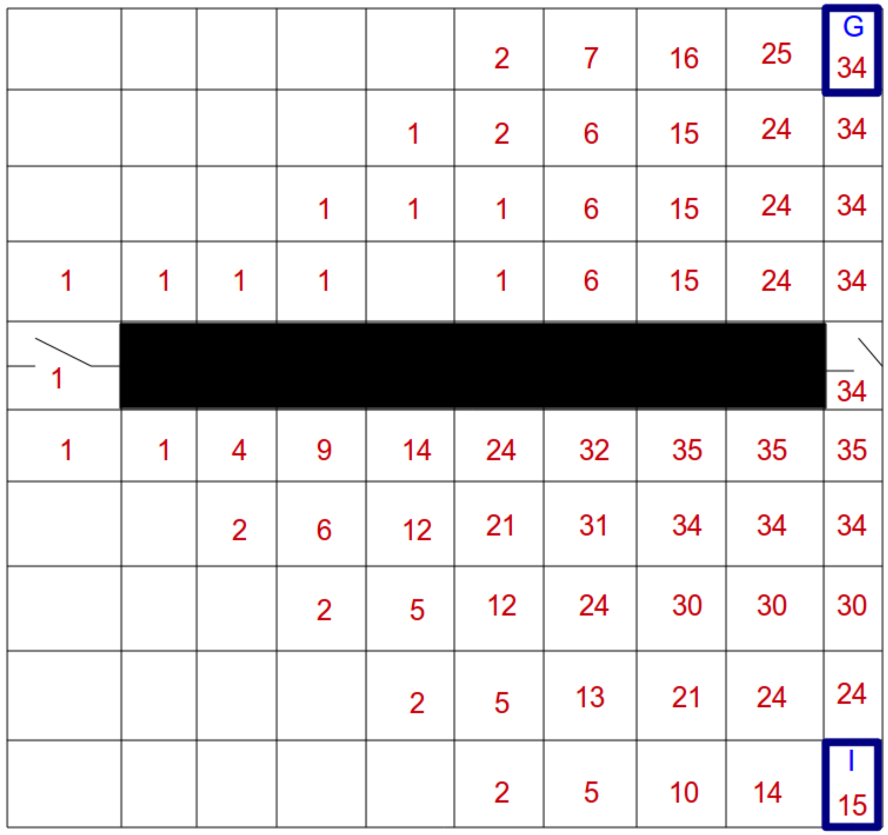
- **Classical Planning**
  - ▷ states & heuristics $h(s)$ not **black boxes**; how to exploit **structure** further?
- **Probabilistic MDP & POMDP Planning**
  - ▷ for scalability, inference can't be at level of **states** or **beliefs** but at level of **variables**
- **Multi-agent Planning**
  - ▷ for scalability, it should build on **single-agent planning** and **plan recognition**; game theory seldom needed
- **Hierarchical Planning**
  - ▷ how to **infer** and **use** hierarchies; what can be **abstracted away** and when?

Example: Best first search can be pretty blind



- Problem involves agent that has to get large package through one of two doors
- The package doesn't fit through the nearest door

Best first search can be pretty blind: Doors Problem



- Numbers in cells show **number of states expanded** where agent at that cell
- Algorithm is **greedy best first search** with **additive heuristic**
- Number of state expansions is close to 998; FF expands 1143 states, LAMA more!
- **34 different states expanded** with agent at **target**, only last one with pkg!

Another Challenge: Scale up in Wumpus World

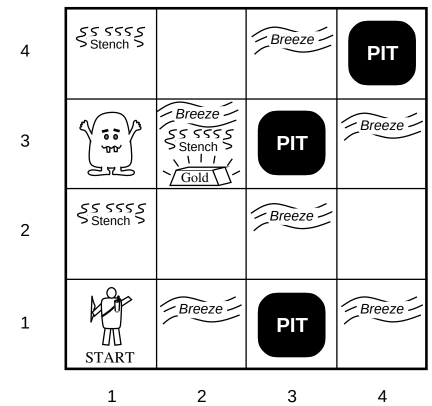
Wumpus World PEAS description

Performance measure
- gold +1000, death -1000
- -1 per step, -10 for using the arrow

Environment
- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

Actuators Left turn, Right turn,
Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell



Options: Compilation + classical planners; UCT methods for POMDPs

Summary

- Planning is the **model-based** approach to autonomous behavior
- Many models and dimensions; all **intractable**
- Challenge is computational, **how to scale up**
- Lots of room for **ideas** whose value must be shown **empirically**
- Key technique in **classical planning** is automatic derivation and use of **heuristics**
- Power of classical planners used for other tasks via **transformations**
- Heuristics also used in the context of more expressive **MDP** and **POMDP** solvers
- **Challenges**: learn while searching; solve Wumpus, multi-agent, . . .
- **Promise:** a solid methodology for **autonomous agent design**

References

[1] A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *Proc. IJCAI-09*.

[2] J. Baier and S. McIlraith. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. ICAPS-07*.

[3] J. Baier and S. McIlraith. Planning with temporally extended goals using heuristic search. In *Proc. ICAPS-06*.

[4] C. L. Baker, R. Saxe, and J. B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 2009.

[5] R.K. Balla and A. Fern. Uct for tactical assault planning in real-time strategy games. In *Proc. IJCAI-09*.

[6] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Art. Int.*, 1995.

[7] D. Bertsekas. *Dynamic Programming and Optimal Control, Vols 1 and 2.* Athena Scientific, 1995.

[8] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[9] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proc. IJCAI-95*.

[10] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. IJCAI-03*.

[11] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS-2000*.

[12] B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *Proc. IJCAI-09*.

[13] B. Bonet and H. Geffner. Solving large POMDPs using real time dynamic programming. In *Proc. AAAI Fall Symp. on POMDPs*, 1998.

[14] B. Bonet and H. Geffner. Planning as heuristic search. *Art. Intel.*, 2001.

[15] B. Bonet and E. Hansen. Heuristic search for planning under uncertainty. In J. Halpern R. Dechter, H. Geffner, editor, *Heuristics, Probability, and Causality: A Tribute to Judea Pearl*. College Publications, 2010.

[16] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*.

[17] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS-09*.

[18] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proc. AAAI-93*.

[19] B. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. ECP-97*.

[20] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in lpg. *JAIR*, 2003.

[21] E. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Art. Int.*, 2001.

[22] Malte Helmert. The Fast Downward planning system. *JAIR*, 2006.

[23] J. Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR*, 20, 2003.

[24] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 2001.

[25] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *JAIR*, 2004.

[26] L. Kaelbling, M. Littman, and T. Cassandra. Planning and acting in partially observable stochastic domains. *Art. Int.*, 1998.

[27] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. ECAI-92*.

[28] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In *Proc. IJCAI-99*.

[29] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI*, 1996.

[30] E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proc. ECAI-08*, 2008.

[31] E. Keyder and H. Geffner. Soft goals can be compiled away. *JAIR*, 2009.

[32] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, 2006.

[33] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 1990.

[34] M. Lekavỳ and P. Návrat. Expressivity of strips-like and htn-like planning. *Agent and Multi-Agent Systems: Technologies and Applications*, pages 121–130, 2007.

[35] D. McDermott. Using regression-match graphs to control search in planning. *Art Int.*, 1999.

[36] H. Nakhost and M. Muller. Monte-Carlo exploration for deterministic planning. In *Proc. IJCAI-09*.

[37] A. Newell and H. Simon. GPS: a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw Hill, 1963.

[38] H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *JAIR*, 2009.

[39] J. Penberthy and D. Weld. Ucpop: A sound, complete, partiall order planner for adl. In *Proc. KR-92*.

[40] J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large pomdps. *JAIR*, 2006.

[41] M. Ramirez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proc. AAAI-2010*.

[42] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 2010.

[43] J. Rintanen. Heuristics for planning with sat. *Proc. CP-2010*.

[44] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010. 3nd Edition.

[45] D. Silver and J. Veness. Monte-carlo planning in large pomdps. *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[46] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proc. UAI-04*.

[47] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[48] S. Yoon, A. Fern, and R. Givan. FF-replan: A baseline for probabilistic planning. In *Proc. ICAPS-07*.