

IJCAI-2016 Tutorial

A Concise Introduction to Planning Models and Methods

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela



Outline

- **Introduction to Planning and Problem Solving**
- **Classical Planning:** Deterministic actions and complete information
- **Beyond Classical Planning:** Transformations
- **Probabilistic Models:** Markov Decision Processes (MDPs), and Partially Observable MDPs (POMDPs)
- **Challenges. Summary**
 - **Reference:** *A concise introduction to models and methods for automated planning*, H. Geffner and B. Bonet, Morgan & Claypool, 6/2013.
 - **Other references:** *Automated planning: theory and practice*, M. Ghallab, D. Nau, P. Traverso. Morgan Kaufmann, 2004, and *Artificial intelligence: A modern approach. 3rd Edition*, S. Russell and P. Norvig, Prentice Hall, 2009.
 - **Relevant biblio:** listed at the end

What is planning?

- Thinking before acting
- Thinking what to do next
- Thinking how best to achieve given goal
- Use predictive model for action selection or control . . .







We'll make this all precise and address:

- What is planning
- Why planning is hard
- How can (automated) planning be done effectively
- Different types of plans and settings

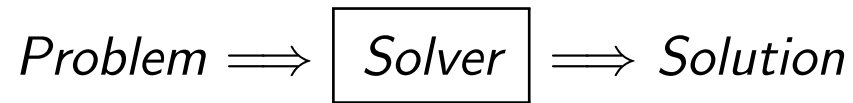
Planning and Autonomous Behavior

The **control problem**: what to do next. Three approaches:

- **Programming-based**: Specify control by hand
- **Learning-based**: Learn control from experience
- **Model-based (Planning)**: **Derive control from model**

Stench		Breeze	
	Glitter Breeze Stench 		Breeze
Stench		Breeze	
	Breeze		Breeze

Planners, Models and Solvers



- It's also useful to see planners in relation to other AI models and solvers:
 - ▷ **Constraint Satisfaction/SAT**: find state that satisfies constraints
 - ▷ **Bayesian Networks**: find probability over variable given observations
 - ▷ **Planning**: find action sequence or policy that produces desired state
 - ▷ **Answer Set Programming**: find answer set of logic program
 - ▷ **General Game Playing**: find best strategy in presence of n -actors, ...
- Solvers for these models are **general**; not tailored to specific instances
- Models are all **intractable**, and some extremely powerful (POMDPs)
- Solvers all have a clear and crisp scope; **they are not architectures**
- Challenge is mainly **computational: how to scale up**
- Methodology is **empirical**: benchmarks and competitions
- Significant **progress** . . .

Familiar Model and Solver: Linear Equations

$$Problem \implies \boxed{Solver} \implies Solution$$

- **Problem:** The age of John is 3 times the age of Peter. In 10 years, it will be only 2 times. How old are John and Peter?
- **Expressed as:** $J = 3P$; $J + 10 = 2(P + 10)$
- **Solver:** Gauss-Jordan (Variable Elimination)
- **Solution:** $P = 10$; $J = 30$

Solver is **general** as deals with any problem expressed as an instance of **model**

Linear Equations Model, however, is **tractable**; AI models are not

For AI solvers to scale up, structure of problems needs to be exploited

SAT

- **SAT** is the problem of determining whether there is a **truth assignment** that satisfies a set of clauses

$$x \vee \neg y \vee z \vee \neg w \vee \dots$$

- Problem is NP-Complete, which in practice means worst-case behavior of SAT algorithms is **exponential** in number of variables ($2^{100} = 10^{30}$)
- Yet current SAT solvers manage to solve problems with **thousands of variables and clauses**, and used widely (circuit design, verification, planning, etc)

How SAT solvers manage to do it?

Two types of **efficient (poly-time) inference** in every node of the search tree:

- **Unit Resolution:**

- ▷ *Derive clause C from $C \vee L$ and unit clause $\sim L$*

- **Conflict-based Learning and Backtracking:**

- ▷ *When empty clause \square derived, find 'causes' S of \square , add $\neg S$ to theory, and backtrack til S disabled*

Other ideas are **logically possible** but **do not work** (do not scale up):

- Generate and test each one of the possible assignments (**pure search**)
- Apply resolution without the unit restriction (**pure inference**)

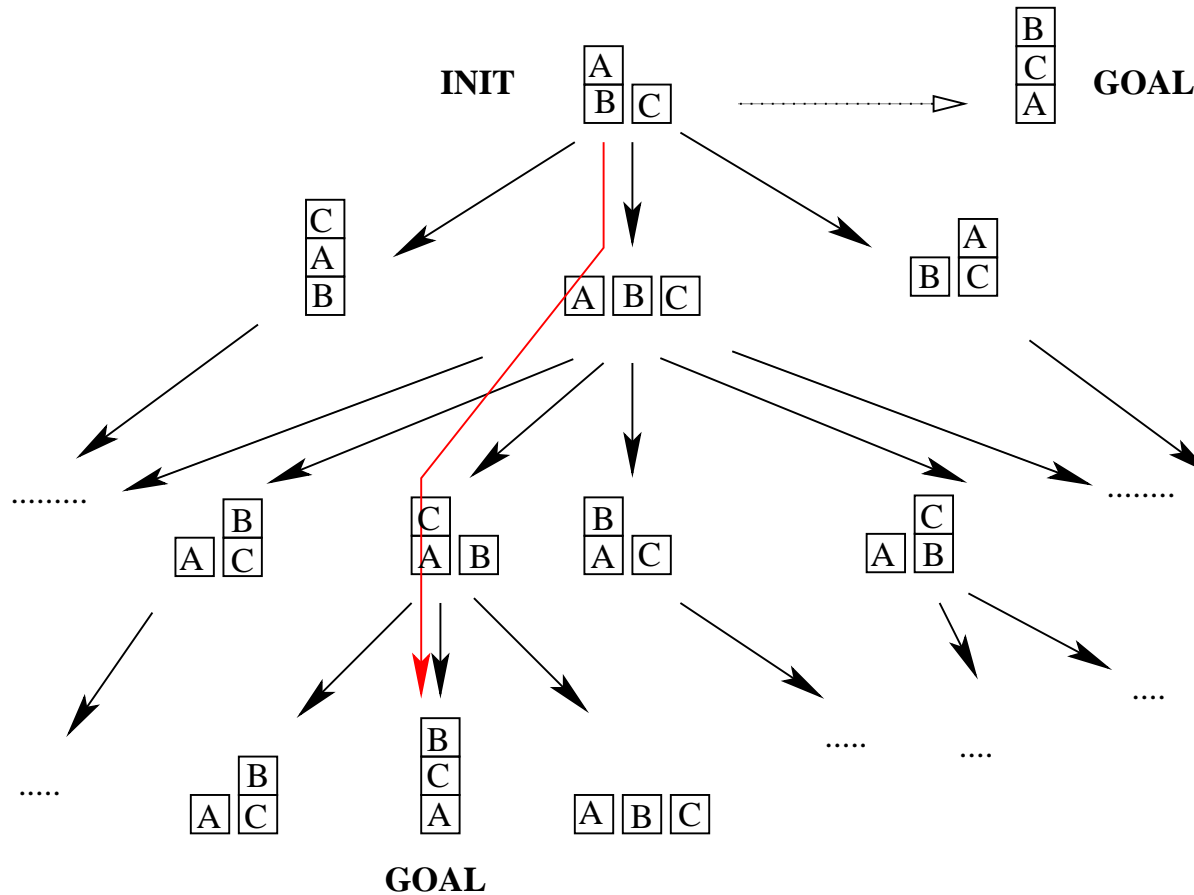
Basic (Classical) Planning Model and Task

- A system that can be in one of many **states**
- States assign **values** to a set of **variables**
- **Actions** change the values of certain variables
- **Basic task:** find **action sequence** to drive **initial state** into **goal state**

$$Model \implies \boxed{Box} \implies Action\ sequence$$

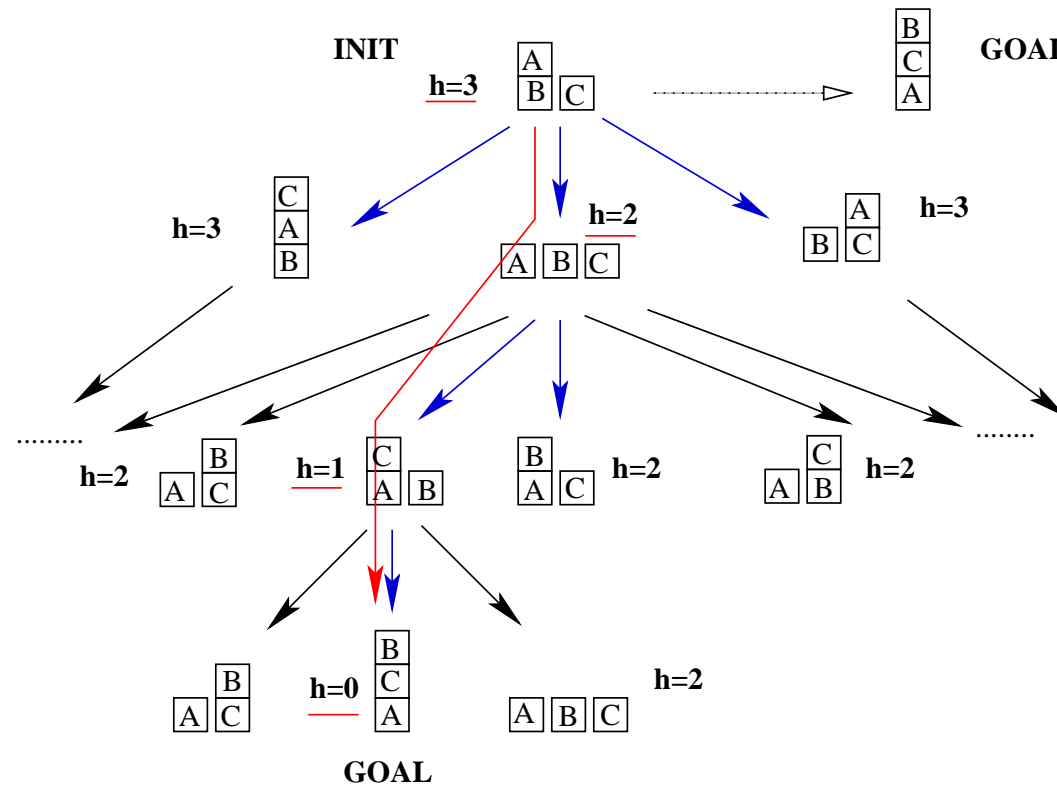
- **Complexity:** NP-hard; i.e., exponential in number of vars **in worst case**
- **Box** is generic; it should work on any domain no matter what variables are about

Concrete Planning Example



- Given the **actions** that move a 'clear' block to the table or onto another 'clear' block, **find a plan** to achieve the goal
- How to find path in the graph whose size is **exponential** in number of blocks?

How Problem Solved? Heuristics Derived Automatically



- **Heuristic evaluations** $h(s)$ provide 'sense-of-direction'
- Derived **efficiently** in a **domain-independent** fashion from **relaxations** where effects made **monotonic** (delete relaxation).

Models, Solvers, and Inference: A bit of Cognitive Science

- We have learned a lot about **effective inference mechanisms** in last 20–30 years from work on **scalable domain-independent solvers**
- The problem of AI in the 80s with **knowledge-based approach** was not just lack of (commonsense) **knowledge**, but lack of **effective inference mechanisms**
- Commonsense based not only on massive amounts of knowledge, but also **massive amounts of fast and effective but unconscious inference**
- This is evident in **Vision** and **NLP** but no less true in **Everyday Reasoning**
- The **unconscious**, not necessarily Freudian, getting renewed attention:
 - ▷ *Strangers to Ourselves: the Adaptive Unconscious* by T. Wilson (2004)
 - ▷ *The New Unconscious*, by Ran R. Hassin et al. (Editors) (2004)
 - ▷ *Blink: The Power Of Thinking Without Thinking* by M. Gladwell (2005)
 - ▷ *Gut Feelings: The Intelligence of the Unconscious* by Gerd Gigerenzer (2007)
 - ▷ . . .
 - ▷ *Thinking, Fast and Slow*. D. Kahneman (2011)

Planning Models: Classical AI 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** or **plan** is a sequence of applicable actions $\pi = a_0, \dots, a_n$ that maps s_0 into S_G ; i.e., there are states s_0, \dots, s_{n+1} s.t. $s_{i+1} = f(a_i, s_i)$ and $a_i \in A(s_i)$ for $i = 0, \dots, n$, and $s_{n+1} \in S_G$.

The plan is **optimal** if it minimizes the **sum of action costs** $\sum_{i=0, n} c(a_i, s_i)$. If costs are all 1, plan cost is plan **length**

Different **models** 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

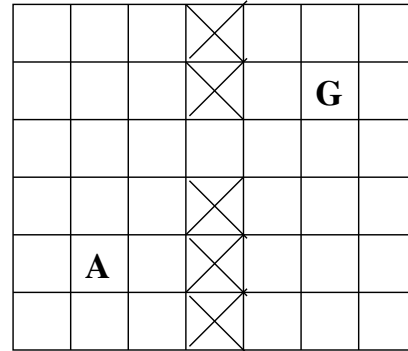
Partially Observable MDPs (POMDPs)

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

- states $s \in S$
 - actions $A(s) \subseteq A$
 - transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
 - observable **goal states** S_G
 - 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 goal

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 non-deterministic and location observable, it's an **MDP** or **FOND**
- If actions non-deterministic and location partially obs, **POMDP** or **Contingent**

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

Planner is generic solver for instances of a particular model

Classical planners, MDP Planners, POMDP planners, . . .

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 compact form by means of **planning languages** (Strips, PDDL, PPDDL, . . .) where **states** represent interpretations over the language.

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 $s \in \mathcal{S}$ are **collections of atoms** from F (valuations over F)
 - the initial state s_0 is I (initially an atom in F is true iff it's in I)
 - the goal states s are such that $G \subseteq s$ (. . .)
 - the actions a in $A(s)$ are ops in O s.t. $Pre(a) \subseteq s$
 - the next state is $s' = s \setminus Del(a) \cup Add(a)$
 - action costs $c(a, s)$ are all 1
- (Optimal) **Solution** of P is (optimal) **solution** of $\mathcal{S}(P)$
- Slight language extensions often convenient (e.g., **negation** and **conditional effects**); some required for describing richer models (costs, probabilities, ...).

Example: Simple Problem in Strips

Problem $P = \langle F, I, O, G \rangle$ where:

- $F = \{p, q, r\}$
- $I = \{p\}$
- $G = \{q, r\}$
- O has two actions a and b such that:
 - ▷ $Prec(a) = \{p\}$, $Add(a) = \{q\}$, $Del(a) = \{\}$
 - ▷ $Prec(b) = \{q\}$, $Add(b) = \{r\}$, $Del(b) = \{q\}$

More meaningful example: Carrying packages

Carrying packages between rooms; $P = \langle F, I, O, G \rangle$ where $p2ATrA$ is pkg A at room A, etc.

- $F = \{p1ATrA, p1ATrB, p2ATrA, p2ATrB, robotATrA, robotATrB, p1held, p2held\}$
- $I = \{p1ATrA, p2ATrA, robotATrB\}$
- $G = \{p1ATrB, p2ATrB\}$
- O contains 10 actions:
 - ▷ Pick-p1rA: $Prec = \{p1ATrA, robotATA\}$, $Add = \{p1held\}$, $Del = \{p1ATrA\}$
 - ▷ Pick-p1rB: $Prec = \{p1ATrB, robotATB\}$, $Add = \{p1held\}$, $Del = \{p1ATrB\}$
 - ▷ Drop-p1rA: $Prec = \{p1held, robotATA\}$, $Add = \{p1ATrA\}$, $Del = \{p1held\}$
 - ▷ Drop-p1rB: $Prec = \{p1held, robotATB\}$, $Add = \{p1ATrB\}$, $Del = \{p1held\}$
 - ▷ Move-A-to-B: $Prec = \{robotATA\}$, $Add = \{robotATB\}$, $Del = \{robotATA\}$
 - ▷ Move-B-to-A: $Prec = \{robotATB\}$, $Add = \{robotATA\}$, $Del = \{robotATB\}$
 - ▷ Pick-p2rA: $Prec = \{p2ATrA, robotATA\}$, $Add = \{p2held\}$, $Del = \{p2ATrA\}$
 - ▷ Pick-p2rB: $Prec = \{p2ATrB, robotATB\}$, $Add = \{p2held\}$, $Del = \{p2ATrB\}$
 - ▷ Drop-p2rA: $Prec = \{p2held, robotATA\}$, $Add = \{p2ATrA\}$, $Del = \{p2held\}$
 - ▷ Drop-p2rB: $Prec = \{p2held, robotATB\}$, $Add = \{p2ATrB\}$, $Del = \{p2held\}$

Too much repetition above; better to use **action schemas** . . .

Previous Example Encoded Using Schemas

New encoding uses **variables** to avoid repetition: variables replaced by **constants** of given **types**.

Types are K , for set of **packages**, and R , for **rooms**: $K = \{p1, p2\}$, $R = \{rA, rB\}$

Problem $P = \langle F, I, O, G \rangle$ can be expressed as:

- $F = \{atp(?p, ?r), atr(?r), holding(?p) \mid ?p \in K, ?r \in R\}$
- $I = \{atp(p1, rA), atp(p2, rA), atr(rA)\}$
- $G = \{atp(p1, rB), atp(p2, rB)\}$,
- O contains 3 **action schemas**; use abbrev. $P = Prec$, $A = Add$, $D = Del$
 - ▷ $Pick(?p \in K, ?r \in R)$: $P : \{atp(?p, ?r), atr(?r)\}$, $A : \{holding(?p)\}$, $D : \{atp(?p, ?r)\}$
 - ▷ $Drop(?p \in K, ?r \in R)$: $P : \{holding(?p), atr(?r)\}$, $A : \{atp(?p, ?r)\}$, $D : \{holding(?p)\}$
 - ▷ $Move(?r1 \in R, ?r2 \in R)$: $P : \{atr(?r1)\}$, $A : \{atr(?r2)\}$, $D : \{atr(?r1)\}$
- **Grounded actions** obtained by replacing variables by constants of same type
- Symbols like atp , atr and $holding$ called **predicates**
- Atoms like $atp(p1, rA)$ obtained by replacing variables in $atp(?p, ?r)$ by $p1$ and rA

PDDL: A Standard Syntax for Classical Planning Problems

- PDDL stands for **Planning Domain Description Language**
- Developed for **International Planning Competition (IPC)**; evolving since 1998.
- In IPC, planners are evaluated over unseen problems encoded in **PDDL**

$$\textit{Problem in PDDL} \implies \boxed{\textit{Planner}} \implies \textit{Plan}$$

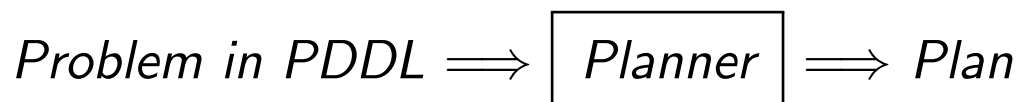
- PDDL specifies syntax for problems $P = \langle F, I, O, G \rangle$ supporting **STRIPS**, **variables**, **types**, and much more
- Problems in PDDL specified in two parts:
 - ▷ **Domain:** contains action and atom **schemas** along with **argument types**.
 - ▷ **Instance:** contains **initial situation**, **goal**, and **constants** (objects) of each type

Example: 2-Gripper Problem in PDDL Syntax

```
(define (domain gripper)
  (:requirements :typing)
  (:types room ball gripper)
  (:constants left right - gripper)
  (:predicates (at-robot ?r - room)(at ?b - ball ?r - room)(free ?g - gripper)
    (carry ?o - ball ?g - gripper))
  (:action move
    :parameters (?from ?to - room)
    :precondition (at-robot ?from)
    :effect (and (at-robot ?to) (not (at-robot ?from))))
  (:action pick
    :parameters (?obj - ball ?room - room ?gripper - gripper)
    :precondition (and (at ?obj ?room) (at-robot ?room) (free ?gripper))
    :effect (and (carry ?obj ?gripper) (not (at ?obj ?room)) (not (free ?gripper))))
  (:action drop
    :parameters (?obj - ball ?room - room ?gripper - gripper)
    :precondition (and (carry ?obj ?gripper) (at-robot ?room))
    :effect (and (at ?obj ?room) (free ?gripper) (not (carry ?obj ?gripper)))))

(define (problem gripper2)
  (:domain gripper)
  (:objects roomA roomB - room Ball1 Ball2 - ball)
  (:init (at-robot roomA) (free left) (free right) (at Ball1 roomA)(at Ball2 roomA))
  (:goal (and (at Ball1 roomB) (at Ball2 roomB))))
```

Computation: How to Solve Classical Planning Problems?



- Planning is one of the oldest areas in AI; many ideas have been tried
- We focus on the two ideas that appear to work best **computationally**
 - ▷ Planning as **Heuristic Search**
 - ▷ Planning as **SAT**
- These methods able to solve problems over huge state spaces
- Of course, some problems are inherently hard, and for them, **general, domain-independent planners** unlikely to approach the performance of **specialized methods**

Solving P by solving $\mathcal{S}(P)$: Path-finding in graphs

Search algorithms for planning exploit the **correspondence** between **(classical) states model** 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

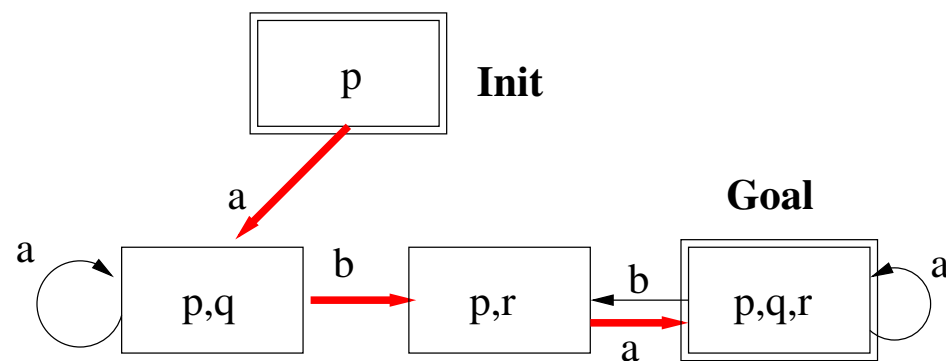
In the **planning as heuristic search** formulation, the problem P is solved by **path-finding** algorithms over the **graph** associated with model $\mathcal{S}(P)$

Example: Simple Problem P and Reachable Graph $\mathcal{S}(P)$

Problem $P = \langle F, I, O, G \rangle$ above where:

- $F = \{p, q, r\}$, $I = \{p\}$, $G = \{q, r\}$
- O with two actions a and b such that:
 - ▷ $Prec(a) = \{p\}$, $Add(a) = \{q\}$, $Del(a) = \{\}$
 - ▷ $Prec(b) = \{q\}$, $Add(b) = \{r\}$, $Del(b) = \{q\}$

Graph associated with reachable fragment of model $\mathcal{S}(P)$; plan in red



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 (BFS), LRTA*, ...*

Heuristic search algorithms can find paths in very large graphs; with more than 10^{20} states as in Rubik's cube

Search Algorithms: General Scheme

Idea: pick *node* from search boundary *Nodes* containing initially root node. If *node* is goal, return solution. Else add children to boundary and repeat. Fail if boundary is empty.

Algorithms like DFS, BrFS, and BFS instances of this general schema

- **DFS:** boundary is a **stack**, pick top, add on top
- **BrFS:** boundary is a **queue**, pick first, add last
- **BFS:** boundary is **priority queue**, pick node that min **evaluation function** f
 - ▷ **A*:** $f(s) = g(s) + h(s)$; $g(s)$ is cost paid up to s , $h(s)$ is estimated cost to goal
 - ▷ **WA*:** $f(s) = g(s) + Wh(s)$; $W > 1$
 - ▷ **Greedy BFS:** $f(s) = h(s)$

Key dimensions: completeness, optimality, and complexity in time and space

On-Line Search: Learning Real Time A* (LRTA*):

- LRTA* is a very interesting **real-time** search algorithm
- It's like **hill-climbing** where **best child selected** and others **discarded**, except that **heuristic** V , initially $V = h$, **updated dynamically**

1. **Evaluate** each action a in s as: $Q(a, s) = c(a, s) + V(s')$
2. **Apply** action a that minimizes $Q(a, s)$
3. **Update** $V(s)$ to $Q(a, s)$
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!**
- Also, it generalizes well to **stochastic actions** (MDPs)

Heuristics: where they come from?

- General idea: heuristic functions obtained as **optimal cost functions** of **relaxed (simplified) problems**
- Examples:
 - *Sum of Manhattan distances in N-puzzle*
 - *Number of misplaced blocks in Blocks-World*
 - *Euclidean Distance in Routing Finding*
 - *Spanning Tree in Traveling Salesman Problem*
- Yet
 - how to get and solve suitable **relaxations**?
 - how to get heuristics **automatically** for planning?

Heuristics for Classical Planning

- Key development in planning in the 90s: automatic extraction of **heuristic functions** to guide search for plans in $S(P)$
- Heuristics derived from **relaxation** where **delete-lists** of actions **dropped**
- This is called the **delete-relaxation**
- $P(s)$ is P but with s as the initial state, P^+ is **delete-relaxation** of P , and $h_P^*(s)$ is **optimal cost** to solve P from s . Then **heuristic** $h(s)$ could be set to:

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

- Yet, this doesn't work **computationally**: solving $P^+(s)$ **optimally** as difficult as solving $P(s)$ **optimally** (NP-hard)
- On the other hand, while solving relaxation $P^+(s)$ **optimally** is **hard**, just **finding one solution** not necessarily optimal, is **easy**

Basic Heuristics and Decomposition of Delete-Free Problems

If plan π_1 achieves G_1 and plan π_2 achieves G_2 , π_1, π_2 achieves G_1 and G_2 in P^+

Iterative procedure to compute plans for all atoms in $P^+(s)$ based on this observation:

- Atom p **reachable** in 0 steps with **empty plan** $\pi(p)$ if $p \in s$
- Atom p **reachable** $i + 1$ steps with **plan** $\pi(p_1), \dots, \pi(p_n), a_p$ if
 - ▷ p not reachable in i steps or less, and
 - ▷ \exists action a_p that adds p with **preconds** p_1, \dots, p_n reachable in $\leq i$ steps

Properties: relaxed plans

- Procedure terminates in number of steps bounded by number of atoms
- If atom p reachable, $\pi(p)$ is a **relaxed plan** for p ; i.e. plan in $P^+(s)$
- If atom p not reachable, there is no plan for p in $P(s)$

Basic Heuristics: h_{max} , h_{FF}

- $h(s) = i$ iff goal g reachable in i steps: is **admissible heuristic** called h_{max}
- $h(s) =$ number of **different** actions in $\pi(g)$: is FF heuristic: h_{FF}

Example

Problem $P = \langle F, I, O, G \rangle$ where

- $F = \{p_i, q_i \mid i = 0, \dots, n\}$
- $I = \{p_0, q_0\}$
- $G = \{p_n, q_n\}$
- O contains actions a_i and b_i , $i = 0, \dots, n - 1$:
 - ▷ $Prec(a_i) = \{p_i\}$, $Add(a_i) = \{p_{i+1}\}$, $Del(a_i) = \{p_i\}$
 - ▷ $Prec(b_i) = \{q_i\}$, $Add(b_i) = \{q_{i+1}\}$, $Del(b_i) = \{q_i\}$

Heuristics for state $s = I$ where p_0 and q_0 are true:

- ▷ $h_{max}(s) = n$; $h_{FF}(s) = 2n$; $h^*(s) = 2n$ (optimal)

Yet if any atom p_i or q_i with $i \neq n$ **added to G** ,

- ▷ $h_{max}(s) = n$; $h_{FF}(s) = 2n$; $h^*(s) = \infty$

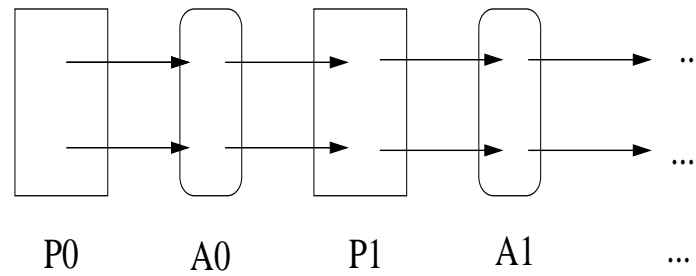
Max Heuristic and Layered Graphs

- Iterative reachability procedure above builds layers P_i of atoms from s :

$$P_0 = \{p \in s\}$$

$$A_i = \{a \in O \mid Pre(a) \subseteq P_k, k \leq i\}$$

$$P_{i+1} = \{p \in Add(a) \mid a \in A_i, p \notin P_k, k < i + 1\}$$



The max heuristic is implicitly **represented** in layered graph:

$$h_{max}(s) = \min i \text{ such each } p \in G \text{ is in layer } P_k, k \leq i$$

Alternative, Mathematical Definition of Max Heuristic

- For all **atoms** p :

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

- For **sets** of atoms C , set:

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

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

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

From Max to Sum: The Additive Heuristic

- For all **atoms** p :

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

- For **sets** of atoms C , replace **Max** by **Sum**:

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

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

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

Heuristic h_{add} is not admissible like h_{max} but informative like h_{FF}

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

First generation of **heuristic search planners**, searched the graph defined by state model $\mathcal{S}(P)$ using standard search algorithms like **Greedy Best-First** or WA^* , and **heuristics** like h_{add}

More recent planners like **FF**, **FD**, and **LAMA** go beyond this in two ways:

- They exploit the **structure of the heuristic** and/or problem further:
 - ▷ **Helpful Actions:** actions most relevant in relaxation
 - ▷ **Landmarks:** implicit problem subgoals
- They use **novel search algorithms**
 - ▷ **Enforced Hill Climbing (EHC)**
 - ▷ **Multi-Queue Best First Search**

The result is that they can often solve **huge problems, very fast**

Enforced Hill Climbing Search in the FF Planner

FF planner works in two phases

- The **second phase** is a **Greedy Best-First** search guided by h_{FF} . This is **complete** but **slow**
- First phase, called **EHC (Enforced Hill Climbing)** is **incomplete** but **fast**
 - ▷ Starting with $s = s_0$, **EHC** does a **breadth-first search** from s using only the **helpful actions** until a state s' is found such that $h_{\text{FF}}(s') < h_{\text{FF}}(s)$.
 - ▷ If such a state s' is found, the process is **repeated** starting with $s = s'$. Else, the EHC **fails**, and the second phase is triggered.
- An action is **helpful** in s if it is applicable in s and adds an atom p not in s such that p is a goal or the precondition of an action in the relaxed plan from s

Landmarks and Helpful Actions in LAMA's Multi-queue BFS

- Standard **best-first algorithms** work with a single queue that is ordered according to the evaluation function f ($f = h$ in GBFS, $f = g + h$ in A^* , $f = g + W * h$ in WA^* , etc).
- If there are two or more **evaluation functions** f , it is also possible to have **several queues** each one ordered by a **different evaluation function**
- **Multi-queue Best-First** search picks **best node** in one queue, then best node in another queue, and so on, **alternating**
- LAMA uses **four queues** and **two heuristics**:
 - ▷ two queues are ordered by h_{FF} , and two by **number of unachieved landmarks**
 - ▷ one queue in each pair is restricted to the nodes obtained by “helpful” actions

Landmarks and Multi-queue Best-First Search in LAMA (cont)

- **Landmarks** are implicit subgoals of the problem; formally atoms p that must be true in **all plans**
- For example, $clear(A)$ is a **landmark** in any Blocks problem where block A is above a **misplaced** block.
- While finding **all landmarks** is computationally hard, **some landmarks** are easy to identify with methods similar to those used for computing heuristics
- Indeed, atom p is a landmark in $P^+(s)$, and hence of $P(s)$, iff heuristics like $h_{max}(s)$ becomes **infinite** once the actions that add p are excluded from the problem.
- Thus, **delete-relaxation landmarks** can be computed in polynomial time; more efficient methods than this, however, available (and used in LAMA).

Something Different: SAT Approach to Planning

- SAT is the problem of determining whether a set of **clauses** is satisfiable
- A clause is a disjunction of **literals** where a literal is an atom p or its negation $\neg p$

$$x \vee \neg y \vee z \vee \neg w$$

- Many problems can be mapped into SAT
- SAT is intractable (exponential in the worst case unless $P=NP$) yet very large SAT problems can be solved in practice

Planning as SAT

Maps **problem** $P = \langle F, O, I, G \rangle$ and **horizon** n into “clauses” $C(P, n)$:

- **Init:** p_0 for $p \in I$, $\neg q_0$ for $q \in F$ and $q \notin I$
 - **Goal:** p_n for $p \in G$
 - **Actions:** For $i = 0, 1, \dots, n - 1$, and each action $a \in O$
 - $a_i \supset p_i$ for $p \in \text{Prec}(a)$
 - $a_i \supset p_{i+1}$ for each $p \in \text{Add}(a)$
 - $a_i \supset \neg p_{i+1}$ for each $p \in \text{Del}(a)$
 - **Persistence:** For $i = 0, \dots, n - 1$, and each atom $p \in F$, where $O(p^+)$ and $O(p^-)$ stand for the actions that add and delete p resp.
 - $p_i \wedge \bigwedge_{a \in O(p^-)} \neg a_i \supset p_{i+1}$
 - $\neg p_i \wedge \bigwedge_{a \in O(p^+)} \neg a_i \supset \neg p_{i+1}$
 - **Sequence:** For each $i = 0, \dots, n - 1$, if $a \neq a'$, $\neg(a_i \wedge a'_i)$
- $C(P, n)$ satisfiable **iff** there is a plan with length bounded by n
 - **Plan** can be read from **truth valuation** that satisfies $C(P, n)$.
 - Encoding simple but not best computationally; for that: parallelism, NO-OPs, lower bounds
 - Best current SAT planners are very good (Rintanen); potentially better than heuristic search planners on highly constrained problems

Optimal Planning

State of the art in optimal planning is **forward search** on state space, either:

- Standard A* combined with **admissible heuristics**
- Search with data structures to efficiently store state subsets (open/closed lists):
 - Search can be blind using breadth-first search
 - Informed using symbolic A* with **admissible heuristics**

In either approach, algorithm is **fixed** what changes is the heuristic

Most effective heuristics to date:

- Use landmark information to obtain admissible estimates
- Integrate different information **automatically extracted** from representation (such as landmarks, abstractions, “constraints” on addition/deletion of atoms along plans) into LP whose solution is guaranteed to provide admissible estimates

A Last Twist: A Stupid but Powerful Blind-Search Algorithm?

Assign each state s generated in the breadth-first search, a number, $novelty(s)$:

- $novelty(s) = 1$ if some **atom** p true in s and false in all previous states
- $novelty(s) = 2$ if some **atom pair** $p \& q$ true in s and false in previous states . . .
- . . .

Iterative Width (IW):

- $IW(i)$ is a **breadth-first** search that **prunes** newly generated states s with $novelty(s) > i$.
- $IW(i)$ runs is **exponential in i** , **not** in number of variables as **normal BrFS**
- IW is **sequence of calls** $IW(i)$ for $i = 1, 2, \dots$ over problem P until problem solved or i exceeds number of variables in problem

How well does IW do? Planning with atomic goals

#	Domain	I	IW(1)	IW(2)	Neither
1.	8puzzle	400	55%	45%	0%
2.	Barman	232	9%	0%	91%
3.	Blocks World	598	26%	74%	0%
4.	Cybersecure	86	65%	0%	35%
...
22.	Pegsol	964	92%	8%	0%
23.	Pipes-NonTan	259	44%	56%	0%
24.	Pipes-Tan	369	59%	37%	3%
25.	PSRsmall	316	92%	0%	8%
26.	Rovers	488	47%	53%	0%
27.	Satellite	308	11%	89%	0%
28.	Scanalyzer	624	100%	0%	0%
...
33.	Transport	330	0%	100%	0%
34.	Trucks	345	0%	100%	0%
35.	Visitall	21859	100%	0%	0%
36.	Woodworking	1659	100%	0%	0%
37.	Zeno	219	21%	79%	0%
Total/Avg		37921	37.0%	51.3%	11.7%
<hr/>					
# Instances	<i>IW</i>	<i>ID</i>	<i>BrFS</i>	<i>GBFS + h_{add}</i>	
37921	34627	9010	8762	34849	

Top: Instances solved by IW(1) and IW(2). **Bottom:** Comparison with ID, BrFS, and GBFS with h_{add}

Sequential IW: Using IW Sequentially to Solve Joint Goals

SIW runs *IW* sequentially for achieving **one (more) goal at a time** (hill-climbing)

Domain	I	Serialized <i>IW</i> (<i>SIW</i>)				GBFS + h_{add}		
		S	Q	T	M/ <i>Awe</i>	S	Q	T
8puzzle	50	50	42.34	0.64	4/1.75	50	55.94	0.07
Blocks World	50	50	48.32	5.05	3/1.22	50	122.96	3.50
Depots	22	21	34.55	22.32	3/1.74	11	104.55	121.24
Driver	20	16	28.21	2.76	3/1.31	14	26.86	0.30
Elevators	30	27	55.00	13.90	2/2.00	16	101.50	210.50
Freecell	20	19	47.50	7.53	2/1.62	17	62.88	68.25
Grid	5	5	36.00	22.66	3/2.12	3	195.67	320.65
OpenStacksIPC6	30	26	29.43	108.27	4/1.48	30	32.14	23.86
ParcPrinter	30	9	16.00	0.06	3/1.28	30	15.67	0.01
Parking	20	17	39.50	38.84	2/1.14	2	68.00	686.72
Pegsol	30	6	16.00	1.71	4/1.09	30	16.17	0.06
Pipes-NonTan	50	45	26.36	3.23	3/1.62	25	113.84	68.42
Rovers	40	27	38.47	108.59	2/1.39	20	67.63	148.34
Sokoban	30	3	80.67	7.83	3/2.58	23	166.67	14.30
Storage	30	25	12.62	0.06	2/1.48	16	29.56	8.52
Tidybot	20	7	42.00	532.27	3/1.81	16	70.29	184.77
Transport	30	21	54.53	94.61	2/2.00	17	70.82	70.05
Visitall	20	19	199.00	0.91	1/1.00	3	2485.00	174.87
Woodworking	30	30	21.50	6.26	2/1.07	12	42.50	81.02
...								
Summary	1150	819	44.4	55.01	2.5/1.6	789	137.0	91.05

Why IW does so well? A Width Notion

Consider a **chain** $t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_n$ where each t_i is a **set of atoms** from P

- A chain is **valid** if t_0 is true in Init and **all optimal plans** for t_i can be **extended into optimal plans** for t_{i+1} by adding a **single** action
- The **size** of the chain is the **size of largest** t_i in the chain
- **Width** of P is **size of smallest** chain $t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_n$ such that that the optimal plans for t_n are optimal plans for P .

Theorem 1: Domains like Blocks, Logistics, Gripper, \dots have all **bounded and small width**, independent of problem **size** provided that goals are **single atoms**

Theorem 2: *IW* runs in time exponential in width of P

IW is **blind search/exploration**. No PDDL or goals used, and can be used with a **simulator**

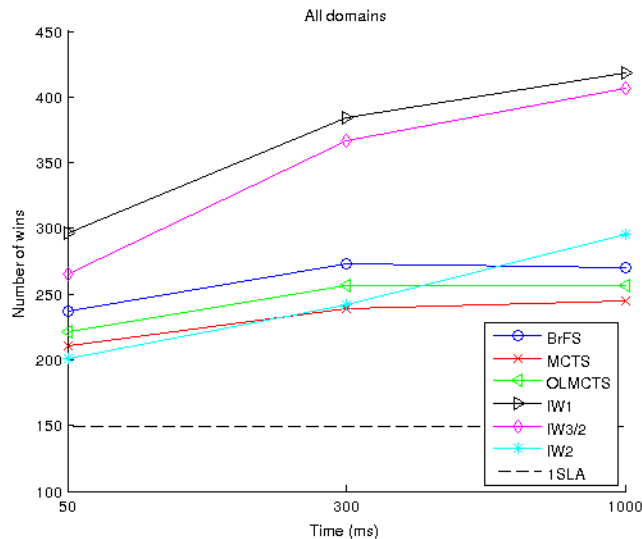
IW on the Atari Video Games

Game	IW(1)		BFS		BrFS	UCT
	Score	Time	Score	Time	Score	Score
ALIEN	25634	81	12252	81	784	7785
AMIDAR	1377	28	1090	37	5	180
ASSAULT	953	18	827	25	414	1512
ASTERIX	153400	24	77200	27	2136	290700
ASTEROIDS	51338	66	22168	65	3127	4661
ATLANTIS	159420	13	154180	71	30460	193858
BANK HEIST	717	39	362	64	22	498
BATTLE ZONE	11600	86	330800	87	6313	70333
BEAM RIDER	9108	23	9298	29	694	6625
...
ROBOT TANK	68	34	52	34	2	50
SEAQUEST	14272	25	6138	33	288	5132
SPACE INVADERS	2877	21	3974	34	112	2718
STAR GUNNER	1540	19	4660	18	1345	1207
TENNIS	24	21	24	36	-24	3
TIME PILOT	35000	9	36180	29	4064	63855
TUTANKHAM	172	15	204	34	64	226
UP AND DOWN	110036	12	54820	14	746	74474
VENTURE	1200	22	980	35	0	0
VIDEO PINBALL	388712	43	62075	43	55567	254748
WIZARD OF WOR	121060	25	81500	27	3309	105500
ZAXXON	29240	34	15680	31	0	22610
# Times Best (54 games)	26		13		1	19

Avg Score collected by IW(1) vs. UCT and other when used in on-line mode (lookahead) in 54 Games. **Atoms** = values of each of the 128 **bytes** in 1024-bit state

IW on the General-Video Games (GVG-AI)

Time	50ms				300ms				1-Look	RND
	BrFS	MC	OLMC	IW(1)	BrFS	MC	OLMC	IW(1)		
Camel Race	2	1	1	0	1	3	0	24	0	1
Digdug	0	0	0	0	0	0	0	0	0	0
Firestorms	12	6	2	13	14	7	6	25	10	0
Infection	20	21	19	22	21	19	22	21	19	22
Firecaster	0	0	0	0	0	0	1	0	0	0
Overload	9	6	8	20	17	3	5	23	0	0
Pacman	1	0	0	2	1	1	4	14	0	0
Seaquest	13	13	15	9	11	17	22	9	12	0
Whackamole	20	18	25	23	22	23	25	21	21	5
Eggomania	0	0	1	21	0	0	2	22	0	0
Total	77	65	71	110	87	73	87	159	62	28



Top: # wins per game out of 25

Left: # wins as function of time for diff algorithms

AI Planning: Status

- The good news: **classical planning works reasonably well**
 - ▷ *Large problems can be solved fast (non-optimally)*
- **Model simple but useful**
 - ▷ *Operators not primitive; can be policies themselves*
 - ▷ *Fast closed-loop replanning able to cope with uncertainty sometimes*
- **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 (conformant planning)
 - ▷ deriving **finite state controllers** (usually set by hand)
 - ▷ doing **plan recognition** (as opposed to plan generation)
 - ▷ . . .

Planning with Soft Goals (Terminal Rewards)

- **Soft goals** as opposed to **hard goals** are to be achieved if worth the costs
- **Utility of plan π is utility of soft goals p achieved minus plan cost:**

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

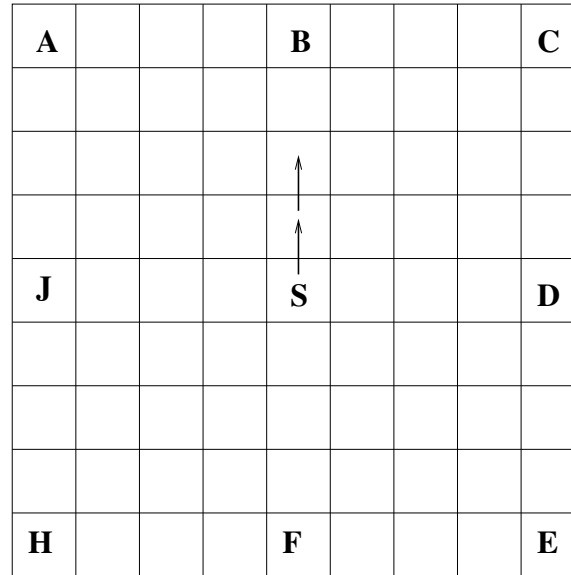
- **Best plan** achieves the hard goals while **maximizing utility**
- 2008 Int. Planning Competition featured **soft goal planning** track (net-benefit)
- Problem looks different than “classical” minimization of **(positive) action costs**
- **Two choices** to make: **which soft goals to achieve** and **how**

Soft Goal Planning with a Classical Planner: Transformation

Yet soft goals can be easily **compiled away**

- 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 translation**
- Classical planners over **translation** outperform **native net-benefit planners**
- This **transformation** is simple and polynomial; others are neither but still **useful**

Goal Recognition with a Planner



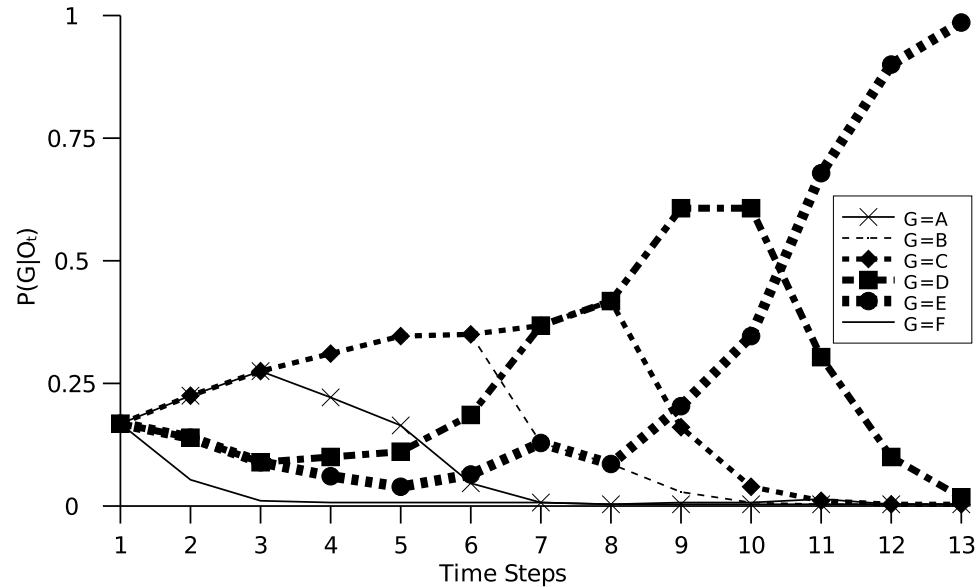
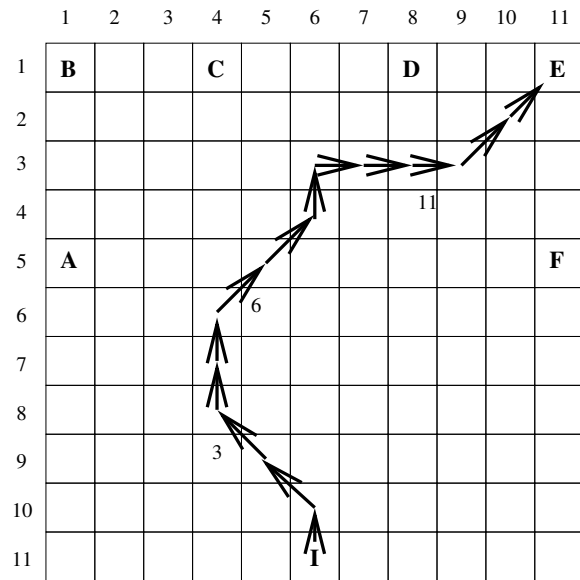
- Agent can **move** one unit in the four directions
- Possible **targets** are A, B, C,
- Starting in S, he is **observed** to move up twice
- **Where** is he going? Why?

Goal Recognition with a Planner: Formulation

A			B			C
			↑			
			↑			
J			S			D
H			F			E

- From Bayes, **goal posterior** is $P(G|O) = \alpha P(O|G) P(G)$, $G \in \mathcal{G}$
- $P(O|G)$ measures **how well goal G predicts observations O** , defined as monotonic function of **difference between two costs**:
 - ▷ $c(G + O)$: cost of achieving G **while complying with obs O**
 - ▷ $c(G + \bar{O})$: cost of achieving G **while not complying with obs O**
- These costs can be computed by classical planner after transformation; **goal posterior $P(G|O)$ results from $|\mathcal{G}|$ calls to classical planner** (assuming priors $P(G)$ given).

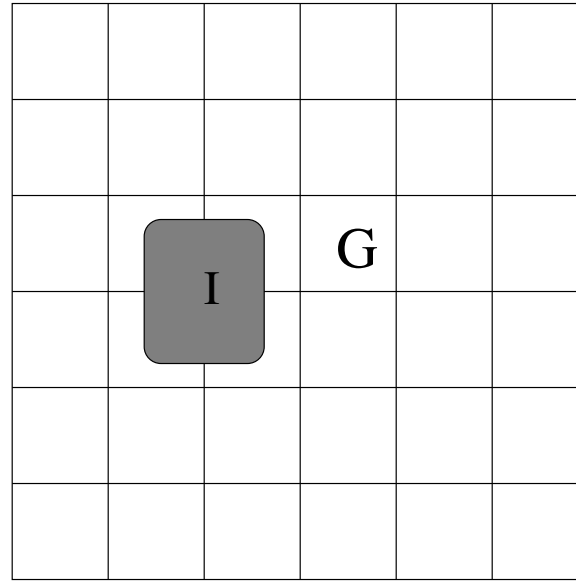
Goal Recognition Example



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

Posterior probabilities $P(G|O)$ obtained from Bayes' rule and costs computed by **classical planner**

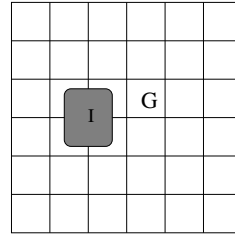
Incomplete Information: Conformant Planning



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

- 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 belief 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

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL \mid \text{clause } L \in I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ 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

- $F' = \{KL/t, K\neg L/t \mid L \in F \text{ and } t \in T\}$
- $I' = \{KL/t \mid \text{if } I \models t \supset L\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ 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

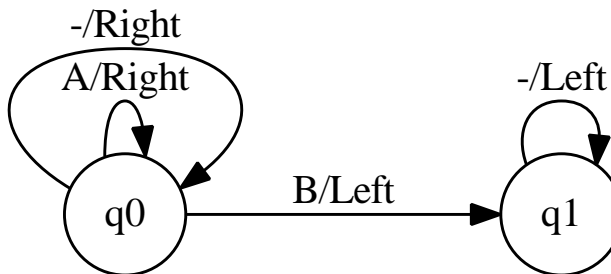
- General translation scheme $K_{T,M}(P)$ is always **sound**, and for suitable choice of the sets of **tags** and **merges**, it is **complete**.
- $K_{S0}(P)$ is **complete instance** of $K_{T,M}(P)$ obtained by setting T to the set of **possible initial states** of P
- $K_i(P)$ is a **polynomial instance**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

Derivation of Finite State Controllers Using Planners

- Starting in A , move to B and back to A ; marks A and B **observable**.



- This **finite-state controller** solves the problem



- FSC is **compact** and **general**: can add noise, vary distance, etc. and still works
- Heavily **used in practice**, e.g. video-games and robotics, but **written by hand**

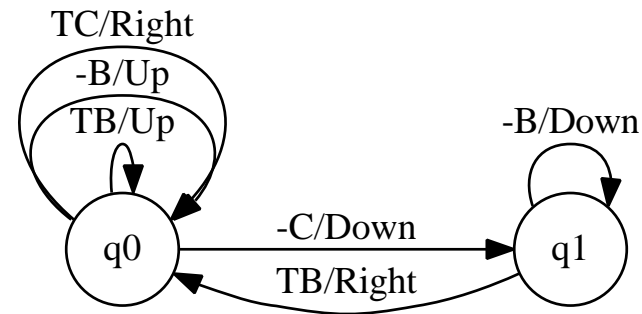
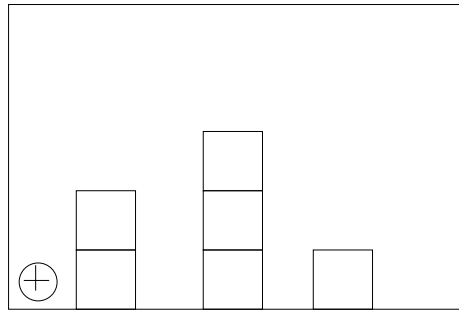
Derivation of Finite State Controllers Using Planners: Idea

- FSC maps controller state, observation pair into action, controller state pair

$$(q, o) \mapsto (a, q')$$

- For deriving FSC using planner, introduce “actions” (q, o, a, q') for reducing original problem P with **sensing** into **conformant** problem P'
- Action (q, o, a, q') behaves like action a but conditional on q and o being true, making q' true as well
- Actions (q, o, a, q') in the plan for transformed problem P' encode **finite-state controller** that solves P
- Plan for **conformant** P' can be obtained by running **classical planner** on further transformed problem $K(P')$

Finite State Controller: Learning from a Single Example



- **Example:** move 'eye' (circle) one cell at a time til **green block** found
- **Observables:** Whether cell 'seen' contains a green block (G), non-green block (B), or neither (C); and whether on table (T) or not (–)
- **Controller** shown derived using a **classical planner** after **transformations**
- Derived controller is general and applies not only to instance shown but to **any other problem instance**; i.e., **any number of blocks** and **any configuration**

Other Problems Solved by Transformations and Classical Planners

- **Temporally Extended Goals** expressed in LTL like “monitor room A and room B forever”: $\Box(\Diamond At(A) \wedge \Diamond At(B))$
- **Probabilistic Conformant Planning**: find action sequence that achieves goal with threshold probability
- **Off-line planning with partial observability**: Contingent planning
- **On-line planning with partial observability**: Wumpus, Minesweeper, . . .
- **Multiagent planning problems**; e.g., agent 1 and 2 need to find blocks 1 and 2 resp. hidden in some room; what they should communicate and when?
- . . .

30min Break

Outline: Probabilistic Models

- Markov Decision Processes (MDPs)
 - Models and solutions
 - Basic dynamic programming methods: Value and Policy Iteration
 - Dynamic programming + heuristic search
- Partially Observable MDPs (POMDPs)
 - Models and solutions
 - Value and policy iteration for POMDPs
 - Approximate algorithms for POMDPs
- Belief Tracking
 - Compact models
 - Basic (Flat) belief tracking
 - Particle filters and structure

Markov Decision Processes (MDPs)

Planning with Markov Decision Processes: Goal MDPs

MDPs are **fully observable**, probabilistic state models:

- state space S
 - initial state $s_0 \in S$
 - a set $S_G \subseteq S$ 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 $c(a, s) > 0$
-
- Agent always **knows** current state
 - **Solutions** are **functions (policies)** mapping states into actions
 - **Optimal** solutions minimize **expected cost** from s_0 to goal

Discounted Reward Markov Decision Processes

Another common formulation of MDPs:

- 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)
 - **discount factor** $0 < \gamma < 1$; **there are no goal states**
-
- **Solutions** are **functions (policies)** mapping states into actions
 - **Optimal** solutions max **expected discounted accumulated reward** from s_0

Expected Cost/Reward of Policy (MDPs)

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

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_i c(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i) \right]$$

where s_i is rv denoting state at time i , and 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

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_i \gamma^i r(a_i, s_i) \mid s_0 = s, a_i = \pi(s_i) \right]$$

- In both cases, **optimal value function V^*** expresses V^π for best π

Solving MDPs: Assumptions

Conditions that ensure **existence** of optimal policies and **correctness** (convergence) of some of the methods we'll see:

- For **discounted MDPs**:
 - discount factor $0 < \gamma < 1$ guarantees that everything is bounded;
e.g. discounted accumulative reward no greater than $C/(1 - \gamma)$, if $r(a, s) \leq C$ for all s and $a \in A(s)$
- For **goal MDPs**:
 - under strictly positive costs, absence of **dead-ends** is assumed so that $V^*(s) \neq \infty$ for all s
 - under general costs, other (mild) assumptions are needed

Equivalence of MDP Models

Two MDP models M and R are **equivalent** if:

- M and R have the same set of actions
- M and R have the same set of non-goal states
- there are constants $\alpha \neq 0$ and β such that for every policy π (mapping from non-goal states into actions), and for every non-goal state s :

$$V_M^\pi(s) = \alpha V_R^\pi(s) + \beta$$

- Additionally, if M and R are of **different sign**, $\alpha < 0$; otherwise $\alpha > 0$

Value functions over non-goal states are related by **linear transformation**

Consequence: if M and R are equivalent, π is optimal for M iff π is optimal for R

Equivalence-Preserving Transformations on MDP Models

A transformation is a function that maps MDP models into MDP models

- for discounted reward MDP R , $R \mapsto R + C$ adds the constant C (positive or negative) to all rewards: $V_{R+C}^\pi(s) = V_R^\pi(s) + C/(1 - \gamma)$
- $R \mapsto kR$ multiplies all costs/rewards by constant k (positive/negative). If k is negative, model kR **changes sign**. We have $V_{kR}^\pi(s) = k \times V_R^\pi(s)$
- for **discounted cost MDP** R , $R \mapsto \bar{R}$, **eliminates** discount factor by:
 - multiplying transition probabilities $P_a(s'|s)$ by γ
 - adding **new (dummy) goal state** g with transition probabilities $P_a(g|s) = (1 - \gamma)$ for all s and $a \in A(s)$

We have $V_{\bar{R}}^\pi(s) = V_R^\pi(s)$

All transformations $R \mapsto R + C$, $R \mapsto kR$ and $R \mapsto \bar{R}$ **preserve** equivalence

Solving MDPs: From Discounted Reward MDPs to Goal MDPs

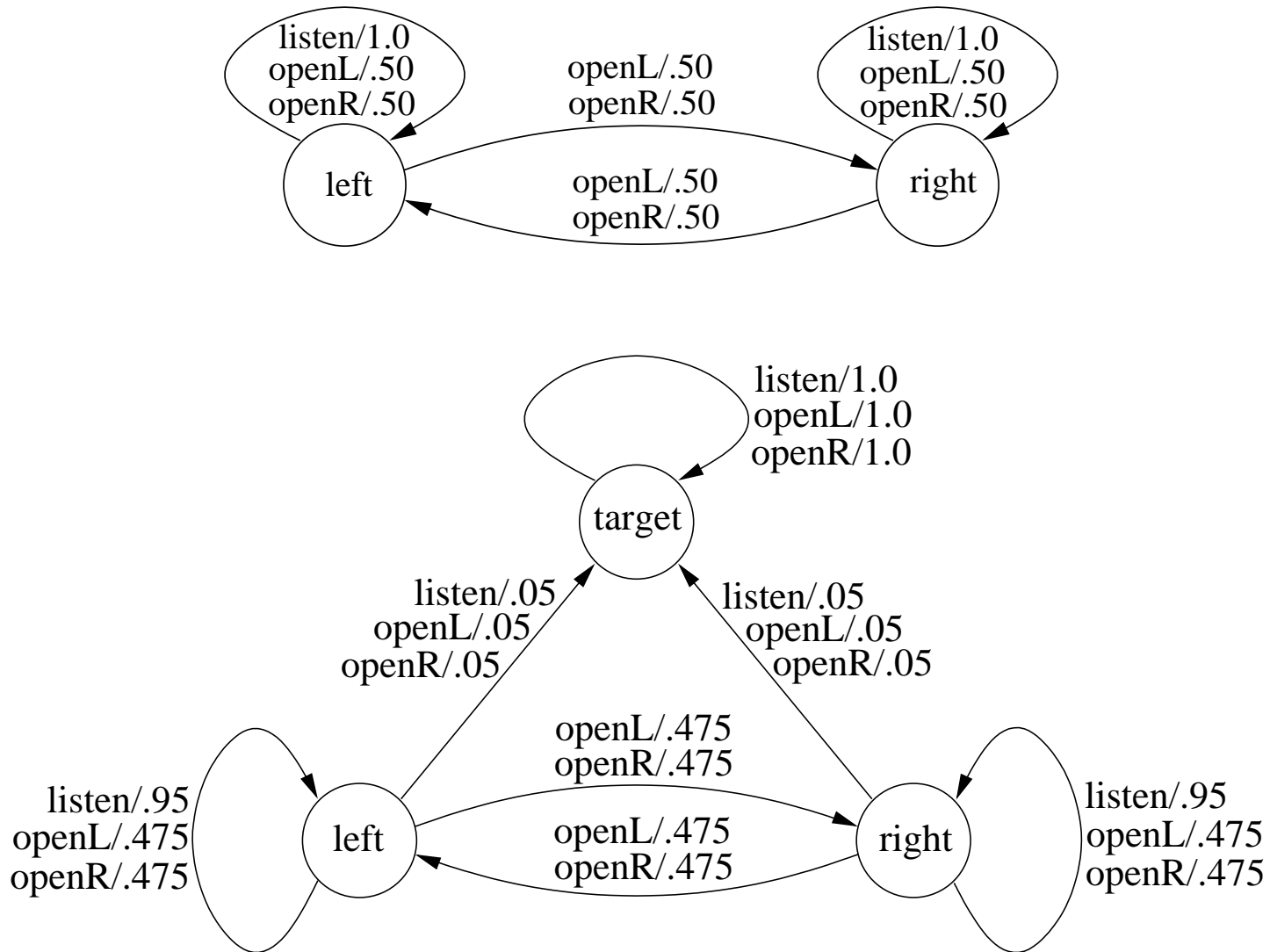
Discounted reward MDPs can be converted into **equivalent goal MDPs**
(no similar transformation known in opposite direction)

Given discounted reward MDP model R :

1. Multiply rewards by -1 applying $R \mapsto -R$
(Result: $-R$ is cost-based and discounted MDP)
2. Add big constant C to make all costs positive using $R \mapsto R + C$ on $-R$
(Result: $-R + C$ has no rewards, only positive costs, but it has discount factor γ)
3. Eliminate discount factor using $R \mapsto \bar{R}$ over $-R + C$
(Result: $\overline{-R + C}$ is Goal MDP)

Consequence: solvers for goal MDPs can be used for discounted reward MDPs

Example of Elimination of Discount Factor $\gamma = 0.95$



Basic Dynamic Programming Methods: Value Iteration (1 of 3)

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

$$\pi_V(s) = \operatorname{argmin}_{a \in A(s)} \left[c(s, a) + \sum_{s' \in S} P_a(s'|s) V(s') \right]$$

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

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

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

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

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

Basic DP Methods: Value Iteration (2 of 3)

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

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

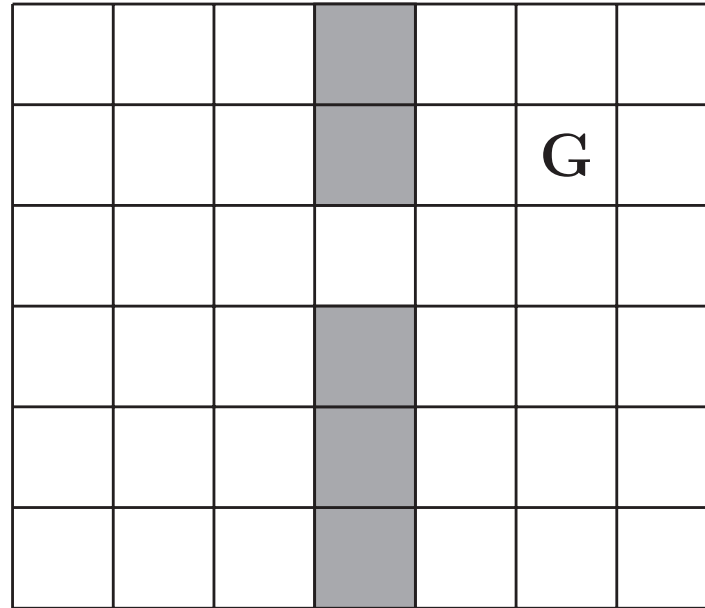
and $V_{i+1}(s) := 0$ for goal states s

- This is **parallel Value Iteration** as the values for all states are updated in each iteration

Basic DP Methods: Value Iteration (3 of 3)

- **Asymptotic convergence:** $V_i \rightarrow V^*$ as $i \rightarrow \infty$
- Parallel VI can be implemented with **two vectors** to store current and next value function
- In practice, stop when **residual** $Res = \max_s |V_{i+1}(s) - V_i(s)|$ is sufficiently small
- Bellman eq. for **discounted reward** MDPs similar, but with **max** instead of **min**, and sum multiplied by γ
- Discounted reward MDPs: **loss of early termination** bounded by $2\gamma Res / (1 - \gamma)$

Example: Value Iteration



- Agent navigates grid: 37 states, 4 actions
- Actions Up, Right, Down and Left move correctly with probability $p = 0.8$, move in **orthogonal direction** (possibly more than one dir.) with rest of probability
- Initial vector is $V_0 \equiv 0$

Example: Value Iteration

0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0

Initial value function V_0

- Agent navigates grid: 37 states, 4 actions
- Actions Up, Right, Down and Left move correctly with probability $p = 0.8$, move in **orthogonal direction** (possibly more than one) with rest of probability
- Initial vector is $V_0 \equiv 0$

Example: Value Iteration

1.0	1.0	1.0		1.0	1.0	1.0
1.0	1.0	1.0		1.0	0.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0		1.0	1.0	1.0
1.0	1.0	1.0		1.0	1.0	1.0
1.0	1.0	1.0		1.0	1.0	1.0

Second value function V_1

- Agent navigates grid: 37 states, 4 actions
- Actions Up, Right, Down and Left move correctly with probability $p = 0.8$, move in **orthogonal direction** (possibly more than one) with rest of probability
- Initial vector is $V_0 \equiv 0$

Example: Value Iteration

2.0	2.0	2.0		2.0	1.2	2.0
2.0	2.0	2.0		1.2	0.0	1.2
2.0	2.0	2.0	2.0	2.0	1.2	2.0
2.0	2.0	2.0		2.0	2.0	2.0
2.0	2.0	2.0		2.0	2.0	2.0
2.0	2.0	2.0		2.0	2.0	2.0

Third value function V_2

- Agent navigates grid: 37 states, 4 actions
- Actions Up, Right, Down and Left move correctly with probability $p = 0.8$, move in **orthogonal direction** (possibly more than one) with rest of probability
- Initial vector is $V_0 \equiv 0$

Example: Value Iteration

10.0	9.0	7.9		2.5	1.5	2.5
9.4	8.1	6.9		1.5	0.0	1.5
8.4	6.9	5.4	3.7	2.7	1.5	2.5
9.4	8.2	6.9		3.8	3.0	3.6
10.0	9.4	8.4		4.9	4.3	4.7
11.0	10.0	9.4		5.9	5.6	5.7

Value function with residual < 0.001

- Agent navigates grid: 37 states, 4 actions
- Actions Up, Right, Down and Left move correctly with probability $p = 0.8$, move in **orthogonal direction** (possibly more than one) with rest of probability
- Initial vector is $V_0 \equiv 0$

Asynchronous Value Iteration

- **Asynchronous Value Iteration** is **asynchronous** version of VI, where each iteration updates the value of **one or more states**, in any order
- Asynchronous VI converges to V^* when **all states updated infinitely often**
- It can be **implemented** with single V vector

Executions and Proper Policies

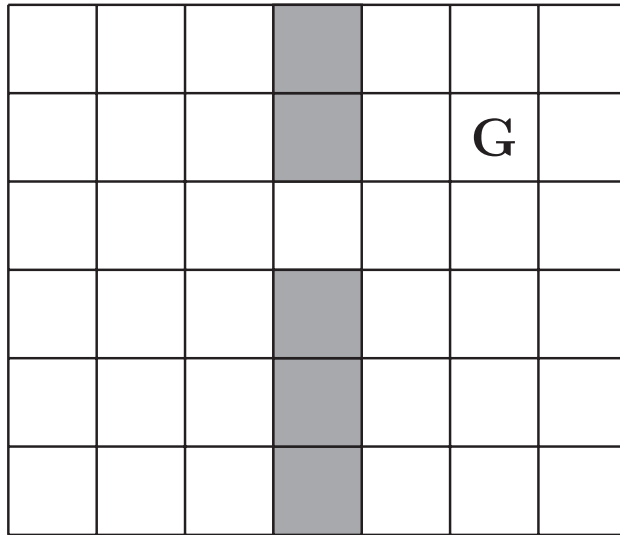
Given policy π and state s , an interleaved sequence $(s_0, a_0, s_1, \dots, a_{n-1}, s_n)$ of states and actions is a π -**execution from** s when:

- it starts at s ; i.e. $s_0 = s$
- actions are given by π ; i.e. $a_i = \pi(s_i)$ for $i = 0, 1, \dots, n - 1$
- transitions are possible; i.e. $P_{a_i}(s_{i+1}|s_i) > 0$ for $i = 0, 1, \dots, n - 1$

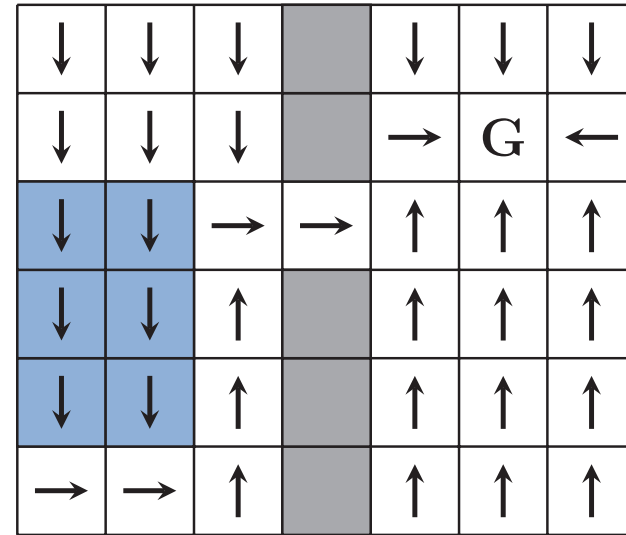
Policy π is **proper** if for every state s , there is a π -execution $(s_0, a_0, s_1, \dots, s_n)$ from s that terminates in **goal state**

The notion of proper policy only applies to goal MDPs

Example of Proper Policy



Goal MDP



Non-optimal but proper policy

Actions at each state yield **intended effect** with some probability $p > 0$

Basic DP Methods: Policy Iteration (1 of 3)

- Expected cost $V^\pi(s)$ for policy π characterized with **set of linear equations**

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

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

- **Linear equations** can be solved by standard methods, or by VI-like procedure
- **Optimal expected cost** at s , $V^*(s)$, is $\min_\pi V^\pi(s)$ and **optimal policy** is π_{V^*}
- Similar for **discounted reward** MDPs, but $c(s, a)$ replaced by $r(a, s)$, min replaced by max, and sum discounted by γ

Basic DP Methods: Policy Iteration (2 of 3)

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

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

- Given policy π , we can **strictly improve** π by changing $\pi(s)$ to a when:
 - in discounted reward MDPs: $Q^\pi(s, a) > Q^\pi(s, \pi(s))$
 - in goal MDPs: π is **proper** and $Q^\pi(s, a) < Q^\pi(s, \pi(s))$
- In goal MDPs, improved policy is **guaranteed** to remain proper when π is proper

Basic DP Methods: Policy Iteration (3 of 3)

- **Policy Iteration (PI)** computes π^* iteratively by sequence of **evaluation** and **improvement** steps:
 1. Starting with arbitrary **proper policy** π (if discounted, start with arbitrary policy)
 2. Compute $V^\pi(s)$ for all states s (**evaluation step**)
 3. Improve π by setting $\pi(s) := \operatorname{argmin}_a Q^\pi(s, a)$ for some s (**improvement step**)
 4. If π changed in 3, go back to 2
- In reward MDPs, improvement is done by setting $\pi(s) = \operatorname{argmax}_a Q^\pi(s, a)$
- PI finishes with π^* after **finite** number of iterations as set of states is **finite**, set of policies is **finite**, and each policy is **better** than previous policy for at least one state

Obtaining a Proper Policy

Different ways to get proper policy in problems without **dead-ends**

- Solve set of linear equations, one per state s :

$$V(s) = \frac{1}{|A(s)|} \sum_{a \in A(s)} \left[c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \right]$$

Define $\pi(s) = \operatorname{argmin}_{a \in A(s)} [c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s')]$

- For each state s , obtain one execution $(s_0, a_0, s_1, \dots, s_n)$ such that
 - $s_0 = s$
 - $a_i \in A(s_i)$ for $i = 0, 1, \dots, n - 1$
 - $P_{a_i}(s_{i+1}|s_i) > 0$ for $i = 0, 1, \dots, n - 1$
 - s_n is goal state

Define $\pi(s) = a_0$ for such execution

Both methods yield **proper policies** that can be used in PI

Dynamic Programming and Heuristic Search

- **DP** methods like Value and Policy Iteration are **exhaustive**: they need to maintain vectors of size $|S|$
- **Heuristic search** algorithms like A^* are **incremental**, and with good **admissible heuristics** can solve much larger problems **optimally**; e.g. Rubik's Cube

Question: Can **admissible heuristics** (lower bounds) and **initial state** s_0 be used to **focus** the updates in DP methods?

Focussed Updates in Dynamic Programming

- Given initial state s_0 , we only need policy π that is **optimal** from s_0
- Convergence to V^* **over all** s not needed to get **optimal policy** for **given** s_0
- Convergence is only required over states reachable from s_0
- Convergence of V at a state s is measured with its V -residual:

$$Res_V(s) = |V(s) - \min_{a \in A(s)} Q_V(s, a)|$$

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

Theorem. *If V is an **admissible** value function and the V -residuals over states reachable with π_V from s_0 are all zero, π_V is an **optimal policy** for s_0 (i.e. $V^\pi(s_0) = V^*(s_0)$ for $\pi = \pi_V$)*

Learning Real Time A* (LRTA*) Revisited

1. **Start** at $s := s_0$
2. **Evaluate** each action a in s as: $Q(s, a) = c(a, s) + V(s')$
3. **Apply** action a^* that minimizes $Q(s, a)$
4. **Update** $V(s)$ to $Q(s, a^*)$
5. **Observe** resulting state s'
6. **Exit** if s' is goal, else go to 2 with $s := s'$

- LRTA* can be seen as **asynchronous value iteration** algorithm for **deterministic** actions that takes advantage of **theorem above** (i.e. update in 4 is DP update)
- **Convergence** of LRTA* implies V -residuals along π_V -reachable states from s_0 are all zero
- Then: 1) $V = V^*$ along such states, 2) π_V is optimal for s_0 , but 3) π_V may not be optimal for other states (yet **irrelevant** if s_0 is given)

Real Time Dynamic Programming (RTDP) for MDPs

RTDP is a generalization of LRTA* to MDPs due to (Barto et al. 95); just adapt Bellman equation used in the **Eval** step

1. **Start** at $s := s_0$
2. **Evaluate** each action a applicable in s as

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

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

Same properties as LRTA* but over MDPs: **after repeated trials**, greedy policy π_V eventually becomes **optimal for s_0** if initial $V(s)$ is **admissible**

A General DP + Heuristic Search Scheme for MDPs

- **Optimal** π for MDPs from s_0 can be obtained for sufficiently small $\epsilon > 0$:
 1. **Start** with admissible V ; i.e. $V(s) \leq V^*(s)$ for all states s
 2. **Repeat**: find state s π_V -reachable from 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 ϵ -convergence bounded by $\frac{1}{\epsilon} \sum_{s \in S} [V^*(s) - V(s)]$
- Like in **heuristic search**, convergence achieved **without visiting or updating** many of the states in S when initial V is **good lower bound**
- **Heuristic search MDP algorithms** like LRTDP, ILAO*, HDP, LDFS, etc. are all instances of this general schema

Scaling Up to larger MDPs: A Little Map

- **Off-line Methods:** compute complete policies or complete policies from s_0
 - **Dynamic programming:** VI, PI
 - **Heuristic search:** RTDP, LAO*, Find-and-Revise, HDP, LDFS, . . .
- **On-line Methods:** compute action to do in current state (not policy)
 - **Finite-Horizon Relaxation:** Solved anytime with UCT, RTDP, AO*, . . .
 - **Deterministic Relaxation:** Solved using classical planners like FF-Replan
- **Alternative Off-Line Methods:**
 - **FOND Relaxation:** Strong cyclic plans yield proper policies; e.g. PRP
 - **Function Approximation:** Parameterized value function; common in RL
 - **Symbolic Methods:** Compact, symbolic representation of value function

Partially Observable Markov Decision Processes (POMDPs)

Partially Observable MDPs: Goal POMDPs

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

- state space S
 - actions $A(s) \subseteq A$ applicable at each state $s \in S$
 - transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
 - **unknown** initial state: **distribution** b_0 for initial state
 - set of **observable target** states S_G
 - action costs $c(a, s) > 0$
 - **sensor model** given by **observable tokens** Ω and probabilities $P_a(o|s)$ for $o \in \Omega$
- **History** is interleaved sequence of actions and observations, beginning with action
 - **Solutions** are policies that map histories into actions
 - **Optimal** policies minimize **expected** cost to go from b_0 to **target belief state**

Discounted Reward POMDPs

A common alternative formulation of POMDPs:

- state space S
 - actions $A(s) \subseteq A$ applicable at each state $s \in S$
 - transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
 - unknown initial state: distribution b_0 for initial state
 - sensor model given by observable tokens Ω and probabilities $P_a(o|s)$ for $o \in \Omega$
 - **rewards** $r(a, s)$ (positive or negative)
 - **discount factor** $0 < \gamma < 1$; **there are no goal states**
- History is interleaved sequence of actions and observations, beginning with action
 - **Optimal** policies max **expected discounted accumulated reward** from b_0
 - Discounted reward POMDPs can be **converted** into equivalent goal POMDPs

Belief States

A belief state b is a **probability distribution** over states

A belief state **summarizes all information** contained in a history that is needed for computing optimal policies

Given history h (interleaved sequence of actions and observations), there is unique belief state b_h that summarizes the relevant information in h

However, two different histories h and h' may map to the same belief (i.e. $b_h = b_{h'}$)

Mapping Histories to Belief States

Given history h (interleaved sequence of actions and observations), there is a unique belief state b_h that summarizes the relevant information in h :

- For the **empty history** h , b_h is the distribution b_0 for the initial state
- For belief $b = b_h$ and action a , the belief for $h' = \langle h, a \rangle$ is $b_a = b_{h'}$:

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

- For $b_a = b_{h'}$ and token $o \in \Omega$, the belief for $h'' = \langle h, a, o \rangle$ is $b_a^o = b_{h''}$:

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

where $b_a(o)$ is norm. const. given by probability of observing o after $h' = \langle h, a \rangle$

POMDPs are MDPs over Belief Space

Information needed to select optimal action after history h is in belief $b = b_h$

POMDP becomes an MDP over beliefs in which policies **map beliefs into actions**

Equations that define MDP over beliefs are:

$$V(b) = \min_{a \in A(b)} c(a, b) + \sum_{o \in \Omega} b_a(o) V(b_a^o) \quad (\text{Bellman eq.})$$

$$V^\pi(b) = c(\pi(b), b) + \sum_{o \in \Omega} b_{\pi(b)}(o) V(b_{\pi(b)}^o)$$

where

$$A(b) = \cap \{A(s) : b(s) > 0\} \text{ is set of } \mathbf{applicable actions} \text{ at } b$$

$$c(a, b) = \sum_{s \in S} c(a, s) b(s) \text{ is expected cost of applying } a \text{ at } b$$

Computational Methods for POMDPs

- **Exact methods:**
 - **Value Iteration** over piecewise linear functions
 - **Policy Iteration** as iteration over finite-state controllers
- **Approximate and on-line methods:**
 - **Point-based Value Iteration methods:** VI over few belief points
 - **RTDP-Bel:** RTDP applied to discretized beliefs
 - **PO-UCT:** UCT applied to action observation histories
 - **Finite-state controllers:** synthesis of controllers of bounded size
- **Logical methods:** probabilities dropped; beliefs as sets of states
 - **Compilations** and **relaxations** for action selection
 - **Belief tracking:** for determining truth of action preconditions and goals
 - **Symbolic approaches:** for representing belief states and value functions

Value Iteration for POMDPs (1 of 2)

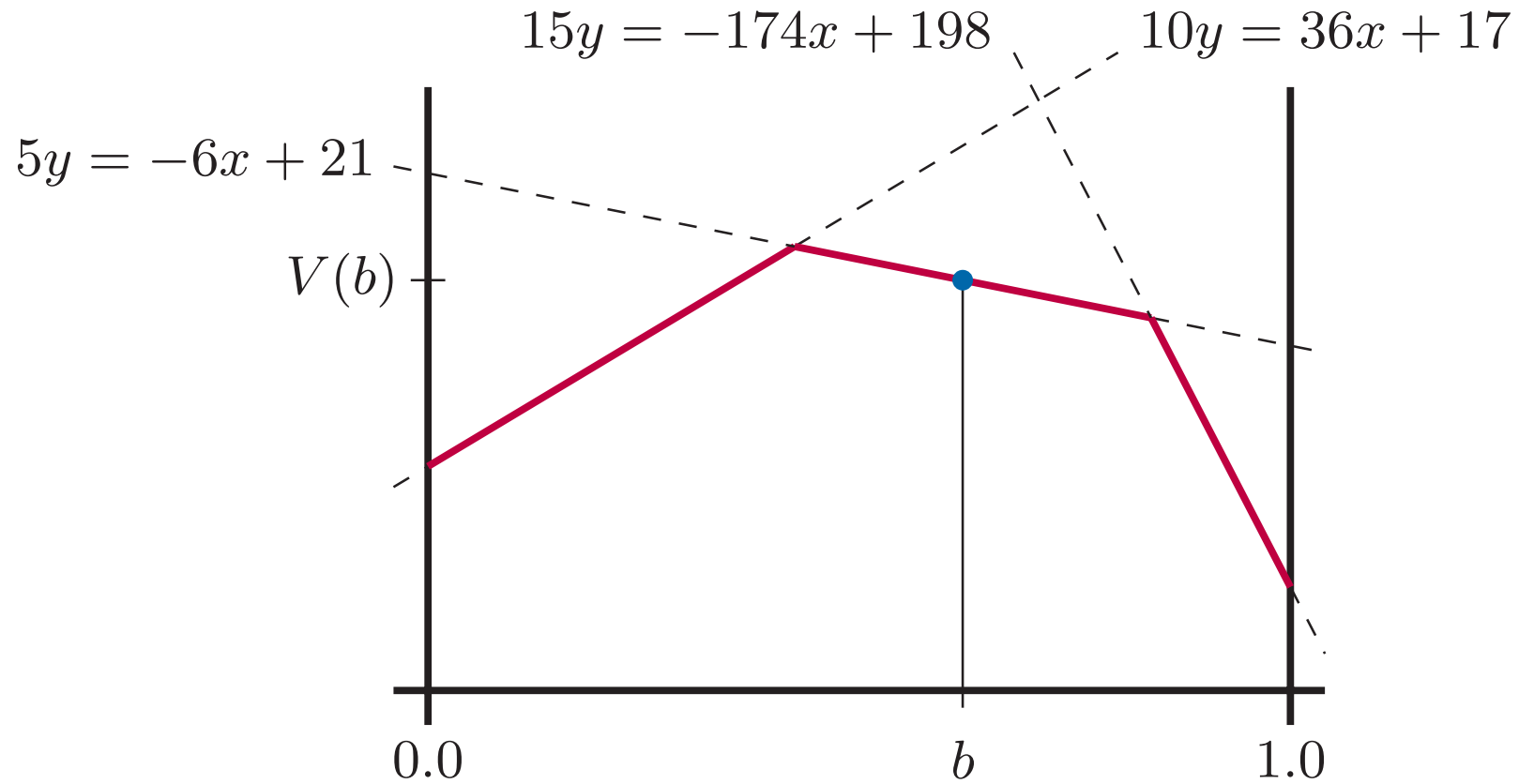
- Belief b is **goal/target belief** if $b(s) = 0$ for non-goal states s
- **Optimal** V^* given by solution to Bellman eq. with $V(b) = 0$ for goal beliefs b

$$V(b) = \min_{a \in A(b)} \left[c(a, b) + \sum_{o \in \Omega} b_a(o) V(b_a^o) \right]$$

- Problem is **infinite** and **dense** space of beliefs to update
- A **piecewise linear and concave (pwlc) function** V determined by **finite set** Γ of vectors (“pieces”):

$$V(b) = \min_{\alpha \in \Gamma} \sum_{s \in S} \alpha(s) b(s) = \min_{\alpha \in \Gamma} \alpha \cdot b$$

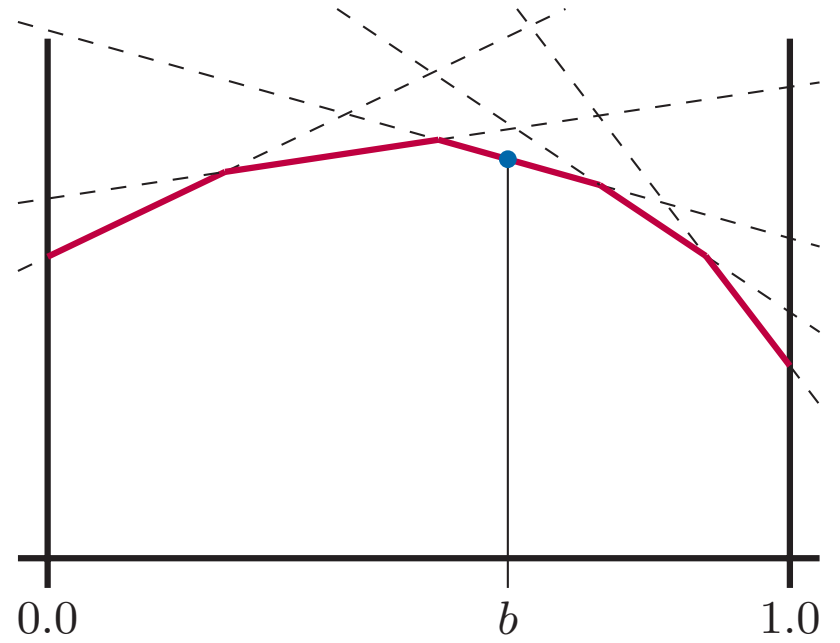
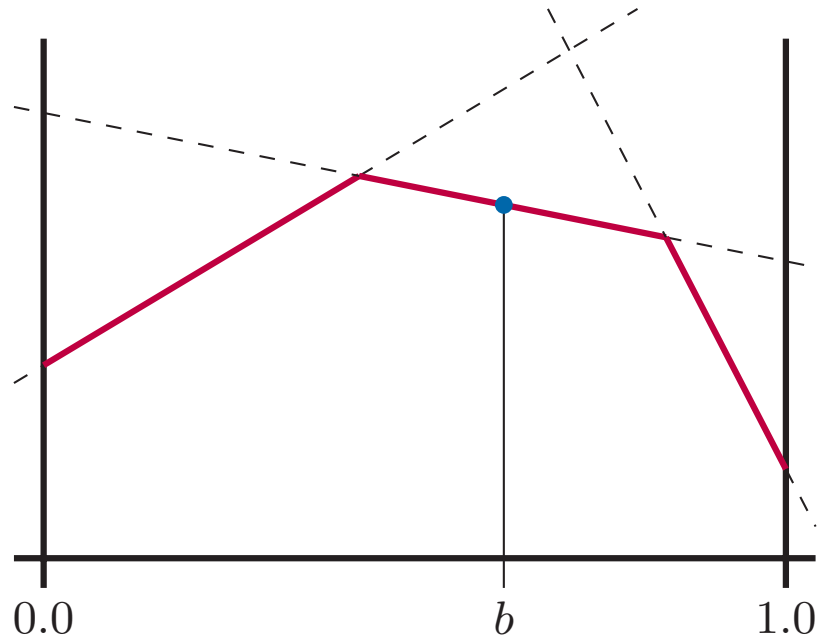
Example of PWLC Function



Value Iteration for POMDPs (2 of 2)

- If V is pwlc-function represented by Γ , we can do (parallel) DP update of V resulting in pwlc-function V' represented by Γ'
- The function $V_0 \equiv 0$ is pwlc represented by $\Gamma_0 = \{\alpha_0\}$ with $\alpha_0 \equiv 0$
- VI over belief states can be implemented as sequence of DP updates over pwlc functions V_0, V_1, V_2, \dots represented by finite sets of vectors $\Gamma_0, \Gamma_1, \Gamma_2, \dots$
- (Optional) **dominated vectors** in Γ_i detected with **LP** and **removed**
- Iterations **stopped** after reaching residual less than given $\epsilon > 0$
(Residual between V_{i+1} and V_i computed from Γ_{i+1} and Γ_i using **LP**)
- Number of “pieces” (i.e. $|\Gamma_i|$) grows **exponentially** with number of updates

Example of DP with PWLC Function



FSCs and Policy Iteration for POMDPs

- When the value function is pwlc, policies can be understood as **finite-state controllers (FSCs)**
- Advantage of such policies over functions that map beliefs into actions is that FSCs don't require **keeping track of beliefs**
- FSCs M_0, M_1, M_2, \dots are constructed in a manner that M_{k+1} is obtained from the set of vectors Γ_{k+1} for V_{k+1} and M_k
- Approach has basically the same limitations of VI, but it can be understood as a form of Policy Iteration for POMDPs
- There are ways to cast the FSC synthesis problem with given number N of controller states as a **non-linear optimization problem**

Approximate POMDP Methods: Point-Based VI (1 of 2)

- **Exponential blow-up** in single DP update due to update of the pwlc function at **all belief states**
- Alternative is to update the value at **selected beliefs** thus controlling the number of vectors
- State-of-the-art offline algorithms based on this idea known as **point-based VI**
- If V is pwlc given by Γ , the **point-based update** of V over belief set F is pwlc \hat{V}_F given by $\hat{\Gamma}$ that **satisfies**

$$\hat{V}_F(b) = V_{\text{Full-DP}}(b)$$

for every $b \in F$, where $V_{\text{Full-DP}}$ is the **full DP update** of V

- If F is complete set of beliefs, $\hat{V}_F = V_{\text{Full-DP}}$ and the point-based update is a full DP update

Approximate POMDP Methods: Point-Based VI (2 of 2)

- Starting with V_0 given by Γ_0 and an initial belief set F_0 , standard point-based algorithms do, for $i = 0, \dots, k$:
 - Set $F_{i+1} := F_i \cup \{ \text{backup}(V_i, b) : b \in F_i \}$
 - Set $V_{i+1} := \widehat{V}_{F_{i+1}}$

where $\text{backup}(V, b)$ is the vector that assigns value to b in $V_{\text{Full-DP}}$; i.e.

$$\text{backup}(V, b) = \underset{\alpha \in \Gamma_{\text{Full-DP}}}{\text{argmin}} \sum_{s \in S} \alpha(s) b(s)$$

and $\Gamma_{\text{Full-DP}}$ is the set of vectors that define the full DP update $V_{\text{Full-DP}}$ of V

- Key result is that $\text{backup}(V, b)$ can be computed in **polynomial time** from V (i.e. $O(|S| |O| |\Gamma|)$ where V given by Γ) without computing $\Gamma_{\text{Full-DP}}$ which is of **exponential size**

Approximate POMDP Methods: RTDP-Bel

- Goal POMDPs are goal MDPs over belief space, then RTDP can be used
- However, we can't maintain a hash table over beliefs (infinite and dense)
- RTDP-Bel **discretizes** beliefs b for writing to and reading from hash table

RTDP-BEL

% Initial value function V given by heuristic h

% Changes to V stored in hash table using discretization function $d(\cdot)$

Let $b := b_0$ the initial belief

Sample state s with probability $b(s)$

While b is not a goal belief **do**

Evaluate each action $a \in A(b)$ as: $Q(b, a) := c(a, b) + \sum_{o \in \Omega} b_a(o) V(b_a^o)$

Select best action $a^* := \operatorname{argmin}_{a \in A(b)} Q(b, a)$

Update value $V(b) := Q(b, a^*)$

Sample next state s' with probability $P_{a^*}(s'|s)$ and set $s := s'$

Sample observation o with probability $P_{a^*}(o|s)$

Update current belief $b := b_{a^*}^o$

end while


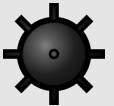


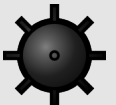
Belief Tracking

POMDPs in Compact Form







- Most of exact and approximated computational methods for POMDPs assume **explicit model**
- Interesting problems defined in terms of variables where actions/observations only affect/sense single or **small subset** of variables

Challenge: design computational methods that scale over **implicit models**

Example of POMDPs in Compact Form

	2		
	3		4
1	3		2
1		2	1

Minesweeper

Stench		Breeze	
	Glitter Breeze Stench 		Breeze
Stench		Breeze	
	Breeze		Breeze

Wumpus



SLAM

Model-Based Autonomous Behavior for POMDPs

Number of states is **exponential in number of variables**

Number of beliefs is **exponential in number of states** (logical setting) or **infinite** (probabilistic setting)

Addressing implicit POMDPs requires solving two **fundamental tasks** (both **intractable** in worst case):

- Efficient representation of belief states
 - Needed for **action selection** when policies map beliefs into actions
 - Needed for **monitoring the system**
- Algorithms for action selection (control problem)

Languages for Implicit POMDPs

Two classes of POMDPs:

- **Logic POMDPs:** no probabilities, only matters which transitions and observations are possible given actions
 - Propositional languages similar to classical planning
- **Probabilistic POMDPs:** transitions and observations specified in **factored** manner
 - Usually done with **2-layer dynamic bayesian network (2-DBN)**

(2-DBN is standard language for compact specification of probabilistic systems)

Basic (Flat) Algorithm for Belief Tracking

Task: Given initial belief b_0 , transitions $P(s'|s, a)$ and sensing $P(o|s, a)$, compute posterior $P(s_{t+1}|o_t, a_t, \dots, o_0, a_0, b_0)$ given execution $(a_0, o_0, \dots, a_t, o_t)$ from b_0

Basic algorithm: Use plain **Bayes updating** $b_{t+1} = b_a^o$ for $b = b_t$ (at state level):

- Logic POMDPs:

$$b_a^o = \{s \in b_a : \text{observation } o \text{ is possible in } s \text{ after } a\}$$

$$b_a = \{s' \in S : \text{there is } s \in b \text{ and transition } (s, a, s') \text{ is possible}\}$$

- Probabilistic POMDPs:

$$b_a^o(s) = P(o|s, a) \times b_a(s) / b_a(o) \propto P(o|s, a) \times b_a(s)$$

$$b_a(s) = \sum_{s'} P(s|s', a) b(s')$$

Complexity: Linear in number of states that is **exponential** in number of variables

Challenge: **Exploit structure** to scale up better when not worst case

Belief Tracking

- **Exact, Explicit Flat Methods**
- **Exact, Lazy Approaches for Planning**
 - Global beliefs b not strictly required, rather beliefs on **preconditions** and **goals**
 - In **logical setting**, this can be cast as **SAT** problem
 - In **probabilistic setting**, as inference problem over a **Dynamic Bayesian Network (DBN)**
 - Both approaches still **exponential** in worst case, but can be sufficiently **practical**
- **Approximations:**
 - **Particle filtering:** when uncertainty in dynamics and sensing represented by probabilities
 - **Structured methods:** exploit structure in logical and probabilistic setting to factorize belief
 - **Decomposition of joint** as product of marginals in probabilistic setting
 - **Combination** particles + decomposition in probabilistic setting given “sufficient” structure
 - **Translation-based approach** in logical setting for simple problems

Probabilistic Belief Tracking with Particles: Basic Approach

- **Particle filtering** algorithms approximate b by multi-set of **unweighted samples**
 - Prob. of $X = x$ approximated by **ratio of samples** in b where $X = x$ holds
- Multi-set B_{k+1} (approx. belief) obtained from B_k , a , and o in two steps:
 - **Sample** s_{k+1} from S with probability $P_a(s_{k+1}|s_k)$ for each s_k in B_k
 - **Re-sample** new set of samples by sampling each s_{k+1} with **weight** $P(o|s_{k+1}, a)$
- Potential problem:
 - Excessive resampling creates a problem known as **loss of diversity**
 - Particles may **die out** if many probabilities are zero
 - May require a big number of particles

Structure in Particle Filters

In some cases, samples don't need to be valuations over all variables (states)

It is sufficient to **sample a subset of variables** and then recover belief over all variables by either

- **polynomial-time** inference (e.g. in Rao-Blackwellised PFs)
- more complex, sometimes **untractable**, inference

Tradeoff because size of sampled var-subset typically larger for first method

Structure for Belief Tracking

- For each variable X , identify subset of vars that are its **immediate causes**
 - Basically, minimal subset S of variables that make X_{t+1} **independent** of the other variables at time t given the variables in $S \cup \{X\}$ and the action at time t
- Likewise, for each **observable variable** Z , identify its **immediate causes**
- The **causal context** for variable X is the **minimum** subset $S(X)$ such that:
 - X belongs to $S(X)$
 - if Z belongs to $S(X)$ and Y is immediate cause of Z , then Z belongs to $S(X)$
- The collection $\{S_1, S_2, \dots, S_n\}$ is **causal decomposition** of problem if:
 - for each variable X (state or obs. var), there is i such that $S(X) \subseteq S_i$
 - no collection with smaller subsets exists
- The **causal width** of problem is $\max_{i=1,2,\dots,n} |S(X_i)|$

Algorithms for Factored Belief Tracking

Algorithms that decompose beliefs in terms of local and independent beliefs for subproblems:

- One subproblem per context in causal decomposition
- Size of largest subproblem exponential in causal width
- Versions for logical and probabilistic setting

Algorithms can handle:

- Minesweeper
- Wumpus
- SLAM
- . . .

Demo for Logical POMDPs

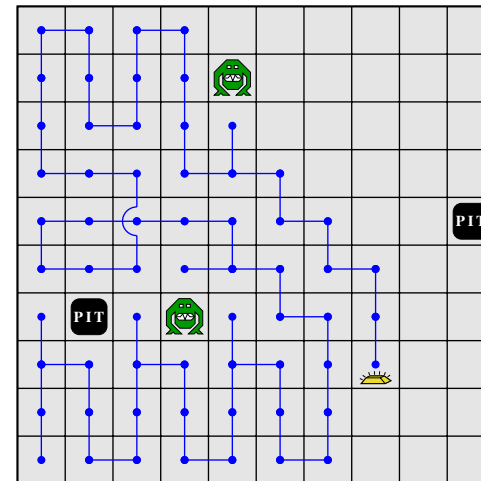
- **Minesweeper:** clear minefield by opening/tagging cells
- **Battleship:** sink ships of different sizes by firing torpedos in a grid
- **Wumpus:** find gold in a grid containing wumpuses and pits

Translation-Based Approaches

- Applies to **deterministic** models in the logical setting expressed in PDDL-like syntax
- Belief tracking problem can be “compiled” via **polynomial translations** into **classical problem**
- Translation is based on width-considerations
- Combined with $K_{T,M}$ translation for conformant planning, we can obtain **on-line solvers** for logical POMDPs

1	1	2	☠	3	3	☠	1
1	☠	2	2	☠	☠	2	1
1	2	2	2	2	2	1	
	1	☠	1				
1	2	1	1				
☠	1					1	1
1	1	★		1	2	3	☠
				1	☠	☠	2

8 × 8 Minesweeper



10 × 10 Wumpus

Wrap Up

Summary

- Planning is the **model-based** approach to autonomous behavior
- Many models and dimensions; all **intractable** in worst case
- Challenge is mainly 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**
- **Structure** and **relaxations** also crucial for **planning with sensing**
- **Promise:** solid methodology for **autonomous agent design**

Some Challenges

- **Classical Planning**

- ▷ states & heuristics $h(s)$ not **black boxes**; how to exploit **structure** further?
- ▷ **on-line planners** to compete with state-of-the-art **classical planners**

- **Probabilistic MDP & POMDP Planning**

- ▷ inference can't be at level of **states** or **belief states** but at level of **variables**

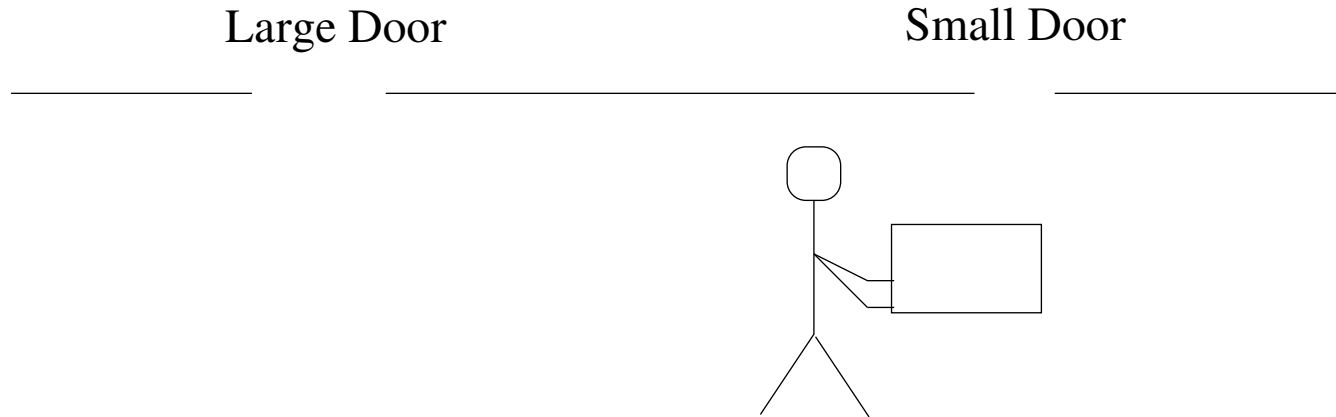
- **Multiagent Planning**

- ▷ should go long way with **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?

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 closest door

Best first search can be pretty blind: Doors Problem

					2	7	16	25	G 34
				1	2	6	15	24	34
			1	1	1	6	15	24	34
1	1	1	1		1	6	15	24	34
1									34
1	1	4	9	14	24	32	35	35	35
		2	6	12	21	31	34	34	34
			2	5	12	24	30	30	30
				2	5	13	21	24	24
					2	5	10	14	I 15

- 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 with pkg!

References

- [1] A. Albarghouthi, J. A. Baier, and S. A. McIlraith. On the use of planning technology for verification. In *Proc. ICAPS'09 Workshop VV&PS*, 2009.
- [2] A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, pages 1623–1628, 2009.
- [3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.
- [4] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5–6):593–618, 2009.
- [5] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.
- [6] A. Bauer and P. Haslum. LTL goal specifications revisited. In *Proc. 19th European Conf. on Artificial Intelligence*, pages 881–886, 2010.
- [7] P. Bertoli, A. Cimatti, M. Pistore, and P. Traverso. A framework for planning with extended goals under partial observability. In *Proc. 13th Int. Conf. on Automated Planning and Scheduling*, pages 215–225, 2003.
- [8] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific, 1995.
- [9] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [10] B. Bonet. An admissible heuristic for sas+ planning obtained from the state equation. In *Proc. 23rd Int. Joint Conf. on Artificial Intelligence*, 2013.
- [11] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. 5th Int. Conf. on Artificial Intelligence Planning Systems*, pages 52–61, 2000.
- [12] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.
- [13] B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, pages 1641–1646, 2009.

- [14] B. Bonet and H. Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. 22nd Int. Joint Conf. on Artificial Intelligence*, pages 1936–1941, 2011.
- [15] B. Bonet and H. Geffner. Action selection for MDPs: Anytime AO* versus UCT. In *Proc. 26th Conf. on Artificial Intelligence*, pages 1749–1755, 2012.
- [16] B. Bonet and H. Geffner. Causal belief decomposition for planning with sensing: Completeness and practical approximation. In *Proc. 23rd Int. Joint Conf. on Artificial Intelligence*, 2013.
- [17] B. Bonet and H. Geffner. Flexible and scalable partially observable planning with linear translations. In *Proc. AAAI*, 2014.
- [18] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. 19th Int. Conf. on Automated Planning and Scheduling*, pages 34–41, 2009.
- [19] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. 17th Int. Joint Conf. on Artificial Intelligence*, pages 690–700, 2001.
- [20] M. Bowling, R. Jensen, and M. Veloso. A formalization of equilibria for multiagent planning. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence*, pages 1460–1462, 2003.
- [21] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. 14th Conf. on Uncertainty on Artificial Intelligence*, pages 33–42, 1998.
- [22] R. I. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz. Planning games. In *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, pages 73–78, 2009.
- [23] R. I. Brafman and G. Shani. A multi-path compilation approach to contingent planning. In *Proc. 26th Conf. on Artificial Intelligence*, pages 1868–1874, 2012.
- [24] R. I. Brafman and G. Shani. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research*, 1(45):565–600, 2012.
- [25] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.
- [26] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

- [27] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1):35–84, 2003.
- [28] A. Cimatti, M. Roveri, and P. Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159:127–206, 2004.
- [29] S. Cresswell and A. M. Coddington. Compilation of LTL goal formulas into PDDL. In *Proc. 16th European Conf. on Artificial Intelligence*, pages 985–986, 2004.
- [30] M. Daniele, P. Traverso, and M. Y. Vardi. Strong cyclic planning revisited. In *Proc. 5th European Conf. on Planning*, pages 35–48, 1999.
- [31] G. de Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. 5th European Conf. on Planning*, pages 226–238, 1999.
- [32] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In *Proc. 11th Nat. Conf. on Artificial Intelligence*, pages 574–579, 1993.
- [33] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proc. 16th Conf. on Uncertainty on Artificial Intelligence*, pages 176–183, 2000.
- [34] S. Edelkamp. On the compilation of plan constraints and preferences. In *Proc. 16th Int. Conf. on Automated Planning and Scheduling*, pages 374–377, 2006.
- [35] S. Edelkamp and S. Schrödl. *Heuristic Search – Theory and Applications*. Academic Press, 2012.
- [36] Z. Feng and E. A. Hansen. Symbolic heuristic search for factored Markov decision processes. In *Proc. 16th Nat. Conf. on Artificial Intelligence*, pages 455–460, 1999.
- [37] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.
- [38] M. Fox and D. Long. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.
- [39] H. Geffner. *Artificial Intelligence: From Programs to Solvers*. *AI Communications*, 2014.
- [40] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods in Automated Planning*. Morgan & Claypool, 2013.

- [41] T. Geffner and H. Geffner. Width-based planning for general video-game playing. In *Proc. AIIDE*, 2015.
- [42] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.
- [43] E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. 14th Conf. on Uncertainty on Artificial Intelligence*, pages 211–219, 1998.
- [44] E. A. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.
- [45] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. 5th European Conf. on Planning*, pages 308–318, 1999.
- [46] M. Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.
- [47] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proc. 19th Int. Conf. on Automated Planning and Scheduling*, pages 162–169, 2009.
- [48] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proc. 17th Int. Conf. on Automated Planning and Scheduling*, pages 176–183, 2007.
- [49] J. Hoffmann, C. Gomes, B. Selman, and H. A. Kautz. SAT encodings of state-space reachability problems in numeric domains. In *Proc. 20th Int. Joint Conf. on Artificial Intelligence*, pages 1918–1923, 2007.
- [50] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
- [51] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.
- [52] F. Kabanza and S. Thiébaux. Search control in planning for temporally extended goals. In *Proc. 15th Int. Conf. on Automated Planning and Scheduling*, pages 130–139, 2005.
- [53] L. P. Kaelbling, M. L. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
- [54] E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proc. 21st Int. Joint Conf. on Artificial Intelligence*, pages 1728–1733, 2009.

- [55] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. 13th Nat. Conf. on Artificial Intelligence*, pages 1194–1201, 1996.
- [56] H. A. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, pages 318–327, 1999.
- [57] T. Keller and P. Eyerich. PROST: Probabilistic planning based on UCT. In *Proc. 22nd Int. Conf. on Automated Planning and Scheduling*, pages 119–127, 2012.
- [58] E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proc. 18th European Conf. on Artificial Intelligence*, pages 588–592, 2008.
- [59] E. Keyder and H. Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009.
- [60] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. 17th European Conf. on Machine Learning*, pages 282–293, 2006.
- [61] S. Koenig and X. Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.
- [62] A. Kolobov, P. Dai, Mausam, and D. S. Weld. Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Proc. 22nd Int. Conf. on Automated Planning and Scheduling*, pages 146–154, 2012.
- [63] A. Kolobov, Mausam, and D. S. Weld. LRTDP versus UCT for online probabilistic planning. In *Proc. 26th Conf. on Artificial Intelligence*, pages 1786–1792, 2012.
- [64] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.
- [65] U. Kuter, D. S. Nau, E. Reisner, and R. P. Goldman. Using classical planners to solve nondeterministic planning problems. In *Proc. 18th Int. Conf. on Automated Planning and Scheduling*, pages 190–197, 2008.
- [66] N. Lipovetzky and H. Geffner. Width and serialization of classical planning problems. In *Proc. 20th European Conf. on Artificial Intelligence*, pages 540–545, 2012.
- [67] N. Lipovetzky, M. Ramirez, and H. Geffner. Classical planning with simulators: Results on the atari video games. In *Proc. IJCAI*, 2015.

- [68] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool, 2012.
- [69] D. V. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1–2):111–159, 1999.
- [70] C. Muise, V. Belle, and S. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proc. AAAI*, 2014.
- [71] C. Muise, S. A. McIlraith, and J. C. Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. 22nd Int. Conf. on Automated Planning and Scheduling*, pages 172–180, 2012.
- [72] A. Newell and H. A. Simon. GPS, a program that simulates human thought. In H. Billing, editor, *Lernende Automaten*, pages 109–124. R. Oldenbourg, 1961.
- [73] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.
- [74] H. Palacios and H. Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.
- [75] F. Patrizi, N. Lipovetzky, G. de Giacomo, and H. Geffner. Computing infinite plans for LTL goals using a classical planner. In *Proc. 22nd Int. Joint Conf. on Artificial Intelligence*, pages 2003–2008, 2011.
- [76] F. Patrizi, N. Lipovetzky, and H. Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proc. 23rd Int. Joint Conf. on Artificial Intelligence*, 2013.
- [77] J. Pineau, G. J. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.
- [78] F. Pommerening, G. Röger, M. Helmert, and B. Bonet. LP-based heuristics for cost-optimal planning. In *Proc. 24th Int. Conf. on Automated Planning and Scheduling*, 2014.
- [79] M. Ramírez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proc. 24th Conf. on Artificial Intelligence*, pages 1121–1126, 2010.
- [80] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

- [81] J. Rintanen. Complexity of planning with partial observability. In *Proc. 14th Int. Conf. on Automated Planning and Scheduling*, pages 345–354, 2004.
- [82] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.
- [83] G. Röger and M. Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proc. 20th Int. Conf. on Automated Planning and Scheduling*, pages 246–249, 2010.
- [84] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.
- [85] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based POMDP solvers. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 1–51, 2012. Online-First Article.
- [86] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Proc. 24th Annual Conf. on Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.
- [87] E. Sondik. The optimal control of partially observable Markov decision processes over the infinite horizon: discounted costs. *Operations Research*, 26(2):282–304, 1978.
- [88] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.
- [89] R. Taig and R. Brafman. Compiling conformant probabilistic planning problems into classical planning. In *ICAPS*, 2013.
- [90] S. Yoon, A. Fern, and R. Givan. FF-replan: A baseline for probabilistic planning. In *Proc. 17th Int. Conf. on Automated Planning and Scheduling*, pages 352–359, 2007.