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

- 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
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
Planners, Models and Solvers

Problem $\Rightarrow$ Solver $\Rightarrow$ Solution

- 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 \underline{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

\[ x \lor \neg y \lor z \lor \neg w \lor \cdots \]

- 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 $C \lor L$ and unit clause $\neg L$

- **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**

\[
\text{Model} \implies \boxed{\text{Box}} \implies \text{Action sequence}
\]

- **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
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:
  - 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 \( \pi = a_0, \ldots, a_n \) that maps \( s_0 \) into \( S_G \); i.e., there are states \( s_0, \ldots, s_{n+1} \) s.t. \( s_{i+1} = f(a_i, s_i) \) and \( a_i \in A(s_i) \) for \( i = 0, \ldots, n \), 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 $P_a(o|s)$, $o \in \text{Obs}$

- 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

\[
\text{Model Instance} \rightarrow \boxed{\text{Planner}} \rightarrow \text{Controller}
\]

• 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 \( \text{Add}(o) \subseteq F \)
  - the **Delete** list \( \text{Del}(o) \subseteq F \)
  - the **Precondition** list \( \text{Pre}(o) \subseteq F \)
From Language to Models

A Strips problem \( P = \langle F, O, I, G \rangle \) determines state model \( S(P) \) where

- the states \( s \in S \) 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 \( G \subseteq s \) (\ldots)
- the actions \( a \) in \( A(s) \) are ops in \( O \) s.t. \( \text{Prec}(a) \subseteq s \)
- the next state is \( s' = s \setminus \text{Del}(a) \cup \text{Add}(a) \)
- action costs \( c(a, s) \) are all 1

- (Optimal) **Solution** of \( P \) is (optimal) solution of \( S(P) \)
- Slight language extensions often convenient (e.g., negation and conditional effects); some required for describing richer models (costs, probabilities, \ldots).
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:
  
  - $\text{Prec}(a) = \{p\}$, $\text{Add}(a) = \{q\}$, $\text{Del}(a) = \{\}$
  - $\text{Prec}(b) = \{q\}$, $\text{Add}(b) = \{r\}$, $\text{Del}(b) = \{q\}$
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: $Prec = \{p1ATrA, robotATrA\}, Add = \{p1held\}, Del = \{p1ATrA\}$
  - Pick-p1rB: $Prec = \{p1ATrB, robotATrB\}, Add = \{p1held\}, Del = \{p1ATrB\}$
  - Drop-p1rA: $Prec = \{p1held, robotATrA\}, Add = \{p1ATrA\}, Del = \{p1held\}$
  - Drop-p1rB: $Prec = \{p1held, robotATrB\}, Add = \{p1ATrB\}, Del = \{p1held\}$
  - Move-A-to-B: $Prec = \{robotATrA\}, Add = \{robotATrB\}, Del = \{robotATrA\}$
  - Move-B-to-A: $Prec = \{robotATrB\}, Add = \{robotATrA\}, Del = \{robotATrB\}$
  - Pick-p2rA: $Prec = \{p2ATrA, robotATrA\}, Add = \{p2held\}, Del = \{p2ATrA\}$
  - Pick-p2rB: $Prec = \{p2ATrB, robotATrB\}, Add = \{p2held\}, Del = \{p2ATrB\}$
  - Drop-p2rA: $Prec = \{p2held, robotATrA\}, Add = \{p2ATrA\}, Del = \{p2held\}$
  - Drop-p2rB: $Prec = \{p2held, robotATrB\}, Add = \{p2ATrB\}, Del = \{p2held\}$

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: $K = \{p_1, p_2\}$, $R = \{rA, rB\}$

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

- $F = \{atp(?p, ?r), atr(?r), holding(?p) \mid ?p \in K, ?r \in R\}$
- $I = \{atp(p_1, rA), atp(p_2, rA), atr(rA)\}$
- $G = \{atp(p_1, rB), atp(p_2, rB)\}$,
- $O$ contains 3 action schemas; use abbrev. $P = Prec$, $A = Add$, $D = Del$
  - $\triangleright$ Pick(?p $\in K, ?r \in R$): $P : \{atp(?p, ?r), atr(?r)\}$, $A : \{holding(?p)\}$, $D : \{atp(?p, ?r)\}$
  - $\triangleright$ Drop(?p $\in K, ?r \in R$): $P : \{holding(?p), atr(?r)\}$, $A : \{atp(?p, ?r)\}$, $D : \{holding(?p)\}$
  - $\triangleright$ Move(?r_1 $\in R, ?r_2 \in R$): $P : \{atr(?r_1)\}$, $A : \{atr(?r_2)\}$, $D : \{atr(?r_1)\}$

- Grounded actions obtained by replacing variables by constants of same type
- Symbols like $atp$, $atr$ and $holding$ called predicates
- Atoms like $atp(p_1, rA)$ obtained by replacing variables in $atp(?p, ?r)$ by $p_1$ 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

\[
\text{Problem in PDDL} \implies \text{Planner} \implies \text{Plan}
\]

- 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)
   (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)
    :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))
)

(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 (at Ball1 roomB)(at Ball2 roomB))))
Computation: How to Solve Classical Planning Problems?

\[
\text{Problem in PDDL } \Longrightarrow \boxed{\text{Planner}} \longrightarrow \text{Plan}
\]

- 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 $S(P)$: 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 $S(P)$
Example: Simple Problem \( P \) and Reachable Graph \( S(P) \)

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:
  - \( \text{Prec}(a) = \{p\} \), \( \text{Add}(a) = \{q\} \), \( \text{Del}(a) = \{\} \)
  - \( \text{Prec}(b) = \{q\} \), \( \text{Add}(b) = \{r\} \), \( \text{Del}(b) = \{q\} \)

Graph associated with reachable fragment of model \( S(P) \); 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 play **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?
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:

$$h(s) \overset{\text{def}}{=} h^*_P(s)$$

- 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 $π(p)$ if $p ∈ s$
- Atom $p$ reachable $i + 1$ steps with plan $π(p_1), \ldots, π(p_n), a_p$ if
  - $p$ not reachable in $i$ steps or less, and
  - ∃ action $a_p$ that adds $p$ with preconds $p_1, \ldots, p_n$ reachable in $≤ i$ steps

Properties: relaxed plans

- Procedure terminates in number of steps bounded by number of atoms
- If atom $p$ reachable, $π(p)$ 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 $π(g)$: is FF heuristic: $h_{FF}$
Example

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

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

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

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

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

- $h_{\text{max}}(s) = n$; $h_{\text{FF}}(s) = 2n$; $h^*(s) = \infty$
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 \( p \):
  \[
  h(p; s) \overset{\text{def}}{=} \begin{cases} 
  0 & \text{if } p \in s, \\
  \min_{a \in O(p)}[\text{cost}(a) + h(Pre(a); s)] & \text{else}
  \end{cases}
  \]

- For sets of atoms \( C \), set:
  \[
  h(C; s) \overset{\text{def}}{=} \max_{r \in C} h(r; s)
  \]

- Resulting heuristic function \( h_{max}(s) \):
  \[
  h_{max}(s) \overset{\text{def}}{=} h(\text{Goals}; s)
  \]
From Max to Sum: The Additive Heuristic

• For all atoms $p$:

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

• For sets of atoms $C$, replace Max by Sum:

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

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

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

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 $S(P)$ 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 $h_{FF}(s') < h_{FF}(s)$.
  
  ▶ 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*, $f = g + W \cdot h$ 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, \( \text{clear}(A) \) 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 $\neg p$.

  $x \lor \neg y \lor z \lor \neg w$

- 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 $q \not\in I$
- **Goal:** $p_n$ for $p \in G$
- **Actions:** For $i = 0, 1, \ldots, n - 1$, and each action $a \in O$
  - $a_i \supset p_i$ for $p \in Prec(a)$
  - $a_i \supset p_{i+1}$ for each $p \in Add(a)$
  - $a_i \supset \neg p_{i+1}$ for each $p \in Del(a)$
- **Persistence:** For $i = 0, \ldots, n - 1$, and each atom $p \in F$, where $O(p^+)$ and $O(p^-)$ stand for the actions that add and delete $p$ resp.
  - $p_i \land \land_{a \in O(p^-)} \neg a_i \supset p_{i+1}$
  - $\neg p_i \land \land_{a \in O(p^+)} \neg a_i \supset \neg p_{i+1}$
- **Sequence:** For each $i = 0, \ldots, n - 1$, if $a \neq a'$, $\neg (a_i \land a'_i)$

- $C(P, n)$ 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
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, $\text{novelty}(s)$:

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

Iterative Width (IW):

- $\text{IW}(i)$ is a breadth-first search that prunes newly generated states $s$ with $\text{novelty}(s) > i$.
- $\text{IW}(i)$ runs is exponential in $i$, not in number of variables as normal BrFS
- $\text{IW}$ is sequence of calls $\text{IW}(i)$ for $i = 1, 2, \ldots$ over problem $P$ until problem solved or $i$ exceeds number of variables in problem
## How well does IW do? Planning with atomic goals

<table>
<thead>
<tr>
<th>#</th>
<th>Domain</th>
<th>I</th>
<th>IW(1)</th>
<th>IW(2)</th>
<th>Neither</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>8puzzle</td>
<td>400</td>
<td>55%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>2.</td>
<td>Barman</td>
<td>232</td>
<td>9%</td>
<td>0%</td>
<td>91%</td>
</tr>
<tr>
<td>3.</td>
<td>Blocks World</td>
<td>598</td>
<td>26%</td>
<td>74%</td>
<td>0%</td>
</tr>
<tr>
<td>4.</td>
<td>Cybersecure</td>
<td>86</td>
<td>65%</td>
<td>0%</td>
<td>35%</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>22.</td>
<td>Pegsol</td>
<td>964</td>
<td>92%</td>
<td>8%</td>
<td>0%</td>
</tr>
<tr>
<td>23.</td>
<td>Pipes-NonTan</td>
<td>259</td>
<td>44%</td>
<td>56%</td>
<td>0%</td>
</tr>
<tr>
<td>24.</td>
<td>Pipes-Tan</td>
<td>369</td>
<td>59%</td>
<td>37%</td>
<td>3%</td>
</tr>
<tr>
<td>25.</td>
<td>PSRsmall</td>
<td>316</td>
<td>92%</td>
<td>0%</td>
<td>8%</td>
</tr>
<tr>
<td>26.</td>
<td>Rovers</td>
<td>488</td>
<td>47%</td>
<td>53%</td>
<td>0%</td>
</tr>
<tr>
<td>27.</td>
<td>Satellite</td>
<td>308</td>
<td>11%</td>
<td>89%</td>
<td>0%</td>
</tr>
<tr>
<td>28.</td>
<td>Sc analyzer</td>
<td>624</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>33.</td>
<td>Transport</td>
<td>330</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>34.</td>
<td>Trucks</td>
<td>345</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>35.</td>
<td>Visitall</td>
<td>21859</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>36.</td>
<td>Woodworking</td>
<td>1659</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>37.</td>
<td>Zeno</td>
<td>219</td>
<td>21%</td>
<td>79%</td>
<td>0%</td>
</tr>
</tbody>
</table>

**Total/Avg:** 37921 37.0% 51.3% 11.7%

<table>
<thead>
<tr>
<th>#</th>
<th>Instances</th>
<th>IW</th>
<th>ID</th>
<th>BrFS</th>
<th>GBFS + $h_{add}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>37921</td>
<td>34627</td>
<td>9010</td>
<td>8762</td>
<td>34849</td>
</tr>
</tbody>
</table>

**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)

<table>
<thead>
<tr>
<th>Domain</th>
<th>Serialized <em>IW (SIW)</em></th>
<th>GBFS + $h_{add}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>S</td>
</tr>
<tr>
<td>8puzzle</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Blocks World</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Depots</td>
<td>22</td>
<td>21</td>
</tr>
<tr>
<td>Driver</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>Elevators</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>Freecell</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Grid</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>OpenStacksIPC6</td>
<td>30</td>
<td>26</td>
</tr>
<tr>
<td>ParcPrinter</td>
<td>30</td>
<td>9</td>
</tr>
<tr>
<td>Parking</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>Pegsol</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>Pipes-NonTan</td>
<td>50</td>
<td>45</td>
</tr>
<tr>
<td>Rovers</td>
<td>40</td>
<td>27</td>
</tr>
<tr>
<td>Sokoban</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>Storage</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>Tidybot</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>Transport</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>Visitall</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Woodworking</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

**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 \( t_0 \rightarrow t_1 \rightarrow \ldots \rightarrow t_n \) 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 \( t_0 \rightarrow t_1 \rightarrow \ldots \rightarrow t_n \) such that that the optimal plans for \( t_n \) are optimal plans for \( P \).

**Theorem 1:** Domains like Blocks, Logistics, Gripper, \ldots 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

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

| # 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)

<table>
<thead>
<tr>
<th>Game</th>
<th>BrFS</th>
<th>MC</th>
<th>OLMC</th>
<th>IW(1)</th>
<th>BrFS</th>
<th>MC</th>
<th>OLMC</th>
<th>IW(1)</th>
<th>1-Look</th>
<th>RND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camel Race</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Digdug</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Firestorms</td>
<td>12</td>
<td>6</td>
<td>2</td>
<td>13</td>
<td>14</td>
<td>7</td>
<td>6</td>
<td>25</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Infection</td>
<td>20</td>
<td>21</td>
<td>19</td>
<td>22</td>
<td>21</td>
<td>19</td>
<td>22</td>
<td>21</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Firecaster</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Overload</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>20</td>
<td>17</td>
<td>3</td>
<td>5</td>
<td>23</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pacman</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Seaquest</td>
<td>13</td>
<td>13</td>
<td>15</td>
<td>9</td>
<td>11</td>
<td>17</td>
<td>22</td>
<td>9</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Whackamole</td>
<td>20</td>
<td>18</td>
<td>25</td>
<td>23</td>
<td>22</td>
<td>23</td>
<td>25</td>
<td>21</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>Eggomania</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>77</td>
<td>65</td>
<td>71</td>
<td>110</td>
<td>87</td>
<td>73</td>
<td>87</td>
<td>159</td>
<td>62</td>
<td>28</td>
</tr>
</tbody>
</table>

**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 $\pi$ is **utility of soft goals $p$ achieved** minus **plan cost**:

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

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

- For each soft goal \( p \), create **new hard goal** \( p' \) initially false, and **two new actions**:
  - \( \text{collect}(p) \) with precondition \( p \), effect \( p' \) and **cost** 0, and
  - \( \text{forgo}(p) \) with an empty precondition, effect \( p' \) and **cost** \( u(p) \)

- Plans \( \pi \) **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

- 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** \( O \)
  - \( c(G + \overline{O}) \): 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).
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
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 bel state $b_a = \{s' \mid s' \in F(a, s) \& s \in b\}$
- conformant problem becomes a path-finding problem in **belief space**

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

- **effective representation** of belief states $b$
- **effective heuristic** $h(b)$ for estimating cost in belief space

**Recent alternative:** translate into classical planning . . .
Basic Translation: Move to the 'Knowledge Level'

Given conformant problem \( P = \langle F, O, I, G \rangle \)

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

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

- \( F' = \{ KL, K\neg L \mid L \in F \} \)
- \( I' = \{ KL \mid \text{ clause } L \in I \} \)
- \( G' = \{ KL \mid L \in G \} \)
- \( O' = O \) 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 (\(\equiv\) 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 \( B \) \textbf{observable}.

\[ A \quad \quad \quad B \]

• This \textbf{finite-state controller} solves the problem

\[ q_0 \quad \text{A/Right} \quad \text{-/Right} \quad q_1 \quad \text{B/Left} \quad \text{-/Left} \]

• \textbf{FSC} is \textbf{compact} and \textbf{general}: can add noise, vary distance, etc. and still works

• Heavily \textbf{used in practice}, e.g. video-games and robotics, but \textbf{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 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”: $\square(\Diamond At(A) \land \Diamond At(B))$

- **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)
Planneding 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 $\pi$ starting at $s$, denoted as $V^\pi(s)$, is

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

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 $\pi$

- In discounted reward MDPs, expected discounted reward from $s$ is

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

- In both cases, optimal value function $V^*$ expresses $V^\pi$ for best $\pi$
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 $\beta$ such that for every policy $\pi$ (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, $a < 0$; otherwise $a > 0$

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

**Consequence:** if $M$ and $R$ are equivalent, $\pi$ is optimal for $M$ iff $\pi$ 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: $V^\pi_{R+C}(s) = V^\pi_R(s) + C/(1-\gamma)$

- $R \mapsto kR$ multiplies all costs/rewards by constant $k$ (positive/negative). If $k$ is negative, model $kR$ changes sign. We have $V^\pi_{kR}(s) = k \times V^\pi_R(s)$

- for discounted cost MDP $R$, $R \mapsto \bar{R}$, eliminates discount factor by:
  - multiplying transition probabilities $P_a(s'|s)$ by $\gamma$
  - 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 $V^\pi_{\bar{R}}(s) = V^\pi_R(s)$

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

3. Eliminate discount factor using $R \mapsto \overline{R}$ over $-R + C$
   (Result: $-R + C$ 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** $\pi_V$ for $V = V^*$ is optimal:

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

- 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 \( \gamma \)

- 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

- 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 $V_0 = 0$
**Example: Value Iteration**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**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

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

<table>
<thead>
<tr>
<th></th>
<th>10.0</th>
<th>9.0</th>
<th>7.9</th>
<th>2.5</th>
<th>1.5</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4</td>
<td>8.4</td>
<td>6.9</td>
<td>5.4</td>
<td>3.7</td>
<td>2.7</td>
<td>1.5</td>
</tr>
<tr>
<td>8.4</td>
<td>6.9</td>
<td>5.4</td>
<td>3.7</td>
<td>2.7</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>9.4</td>
<td>8.2</td>
<td>6.9</td>
<td>3.8</td>
<td>3.0</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td>9.4</td>
<td>8.4</td>
<td>4.9</td>
<td>4.3</td>
<td>4.7</td>
<td></td>
</tr>
<tr>
<td>11.0</td>
<td>10.0</td>
<td>9.4</td>
<td>5.9</td>
<td>5.6</td>
<td>5.7</td>
<td></td>
</tr>
</tbody>
</table>

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 $V_0 = 0$
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 $\pi$ and state $s$, an interleaved sequence $(s_0, a_0, s_1, \ldots, a_{n-1}, s_n)$ of states and actions is a $\pi$-execution from $s$ when:

– it starts at $s$; i.e. $s_0 = s$

– actions are given by $\pi$; i.e. $a_i = \pi(s_i)$ for $i = 0, 1, \ldots, n - 1$

– transitions are possible; i.e. $P_{a_i}(s_{i+1}|s_i) > 0$ for $i = 0, 1, \ldots, n - 1$

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

The notion of proper policy only applies to goal MDPs
Example of 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 $\pi$ characterized with \textbf{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

- \textbf{Linear equations} can be solved by standard methods, or by VI-like procedure

- \textbf{Optimal expected cost} at $s$, $V^*(s)$, is $\min_\pi V^\pi(s)$ and \textbf{optimal policy} is $\pi_{V^*}$

- Similar for \textbf{discounted reward} MDPs, but $c(s, a)$ replaced by $r(a, s)$, min replaced by max, and sum discounted by $\gamma$
• Let $Q^\pi(s, a)$ be **expected cost** from $s$ when doing $a$ first and following $\pi$:

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

• Given policy $\pi$, we can **strictly improve** $\pi$ by changing $\pi(s)$ to $a$ when:

  -- in discounted reward MDPs: $Q^\pi(s, a) > Q^\pi(s, \pi(s))$

  -- in goal MDPs: $\pi$ is **proper** and $Q^\pi(s, a) < Q^\pi(s, \pi(s))$

• In goal MDPs, improved policy is **guaranteed** to remain proper when $\pi$ is proper
• **Policy Iteration (PI)** computes $\pi^*$ iteratively by sequence of *evaluation* and *improvement* steps:

1. Starting with arbitrary **proper policy** $\pi$ (if discounted, start with arbitrary policy)
2. Compute $V^\pi(s)$ for all states $s$ (**evaluation step**)
3. Improve $\pi$ by setting $\pi(s) := \text{argmin}_a Q^\pi(s, a)$ for some $s$ (**improvement step**)
4. If $\pi$ changed in 3, go back to 2

• In reward MDPs, improvement is done by setting $\pi(s) = \text{argmax}_a Q^\pi(s, a)$

• PI finishes with $\pi^*$ 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 $\pi(s) = \arg\min_{a \in A(s)} [c(a, s) + \sum_{s' \in S} P_a(s'|s)V(s')]$

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

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

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 $\pi_V$ from $s_0$ are all zero, $\pi_V$ 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 $\pi_V$-reachable states from $s_0$ are all zero
- Then: 1) $V = V^*$ along such states, 2) $\pi_V$ is optimal for $s_0$, but 3) $\pi_V$ 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 \( \pi_V \) eventually becomes optimal for \( s_0 \) if initial \( V(s) \) is admissible
A General DP + Heuristic Search Scheme for MDPs

- **Optimal** $\pi$ 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 $s$ $\pi_V$-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 $\epsilon$-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 $\Omega$ 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 $\Omega$ 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
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:

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

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

where

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

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

\[ 15y = -174x + 198 \]
\[ 10y = 36x + 17 \]
\[ 5y = -6x + 21 \]

\( V(b) \)
Value Iteration for POMDPs (2 of 2)

• If $V$ is pwlc-function represented by $\Gamma$, we can do (parallel) DP update of $V$ resulting in pwlc-function $V'$ represented by $\Gamma'$. 

• 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 $V_0, V_1, V_2, \ldots$ represented by finite sets of vectors $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$. 

• (Optional) dominated vectors in $\Gamma_i$ 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 $\Gamma_{i+1}$ and $\Gamma_i$ 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 $M_0, M_1, M_2, \ldots$ are constructed in a manner that $M_{k+1}$ is obtained from the set of vectors $\Gamma_{k+1}$ 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 $\Gamma$, 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 $\Gamma_0$ and an initial belief set $F_0$, standard point-based algorithms do, for $i = 0, \ldots, k$:
  - Set $F_{i+1} := F_i \cup \{ \text{backup}(V_i, b) : b \in F_i \}$
  - Set $V_{i+1} := \hat{V}_{F_{i+1}}$

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

$$
\text{backup}(V, b) = \operatorname{arg\,min}_{\alpha \in \Gamma_{\text{Full-DP}}} \sum_{s \in S} \alpha(s) b(s)
$$

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 $\Gamma$) 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

\[
\text{RTDP-Bel}
\]

\[
\% \text{ Initial value function } V \text{ given by heuristic } h
\]
\[
\% \text{ Changes to } V \text{ stored in hash table using discretization function } d(\cdot)
\]

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 $a^* := \arg\min_{a \in A(b)} Q(b, a)$
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

**Example: Wumpus and Minesweeper**

**Minesweeper**

- **Stench**
- **Breeze**
- **Glitter**
- **PIT**

**Wumpus**

- **Stench**
- **Breeze**
- **PIT**

**SLAM**

1-Line SLAM:
- Agent moves left/right in a noisy way
- Agent senses color beneath its cell in a noisy way
- Task is to construct underlying color map: requires simultaneous localization and mapping
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 $P(s_{t+1}|o_t,a_t,\ldots,o_0,a_0,b_0)$ given execution $(a_0,o_0,\ldots,a_t,o_t)$ 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 $\{S_1, S_2, \ldots, S_n\}$ 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 $\max_{i=1,2,\ldots,n} |S(X_i)|$
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

<table>
<thead>
<tr>
<th>Instance</th>
<th>Width</th>
<th>Length</th>
<th>Total Prep</th>
<th>Exec</th>
<th>Beam Tracking</th>
<th>Arc-Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>8x8 Minesweeper</td>
<td>8</td>
<td>163</td>
<td>213</td>
<td>3.0</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>10x10 Wumpus</td>
<td>10</td>
<td>156</td>
<td>213</td>
<td>6.5</td>
<td>1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
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
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


