Planning in Intelligent Systems: Model-based Approach to Autonomous Behavior

Master Intelligent Interactive Systems (MIIS)
Universitat Pompeu Fabra

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Tentative plan for the course

- **Intro to AI** and Automated Problem Solving
- **Classical Planning** as Heuristic Search and SAT
- **Beyond Classical Planning:** Transformations
  - ▷ *Soft goals, Conformant Planning, Finite State Controllers, Plan Recognition, Extended temporal LTL goals, . . .*
- **Planning with Uncertainty:** Markov Decision Processes (MDPs)
- **Planning with Incomplete Information:** Partially Observable MDPs (POMDPs)
- **Planning with Uncertainty and Incomplete Info:** Logical Models

– **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.

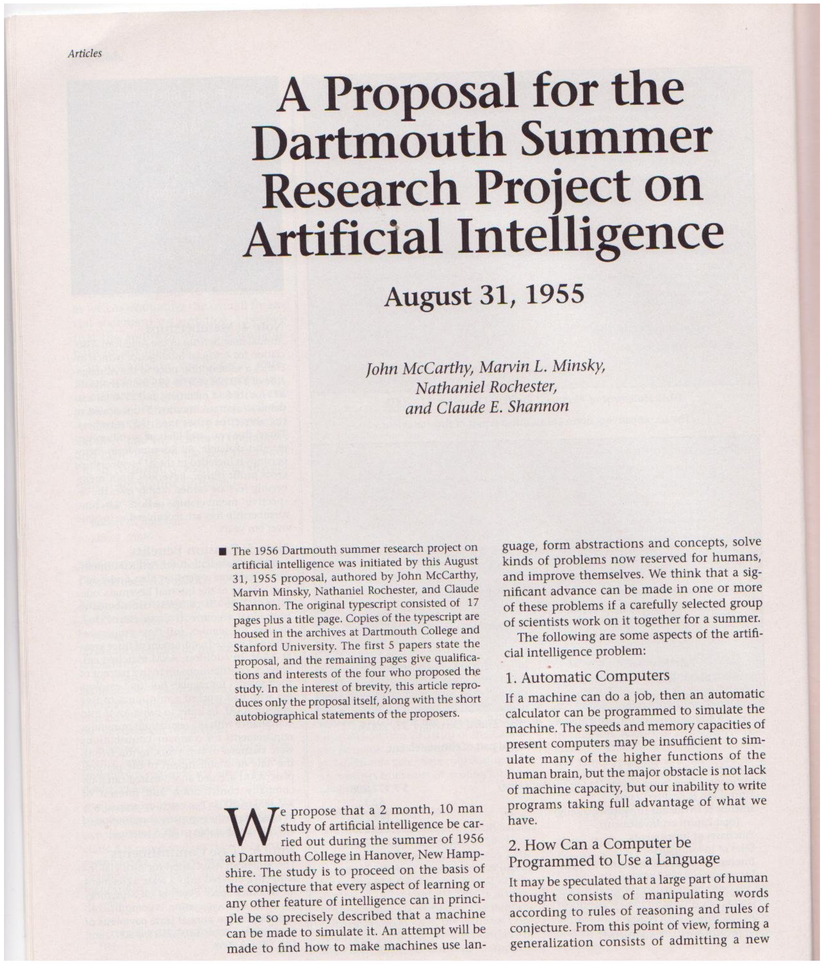
– **Initial set of slides:** `http://www.dtic.upf.edu/∼hgeffner/bsas-2013-slides.pdf`

– **Evaluation, Homework, Projects:** . . .

First Lecture

- Some AI history
- The Problem of Generality in AI
- Models and Solvers
- Intro to Planning

Darmouth 1956



Articles

A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence

August 31, 1955

John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon

■ The 1956 Dartmouth summer research project on artificial intelligence was initiated by this August 31, 1955 proposal, authored by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. The original typescript consisted of 17 pages plus a title page. Copies of the typescript are housed in the archives at Dartmouth College and Stanford University. The first 5 papers state the proposal, and the remaining pages give qualifications and interests of the four who proposed the study. In the interest of brevity, this article reproduces only the proposal itself, along with the short autobiographical statements of the proposers.

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

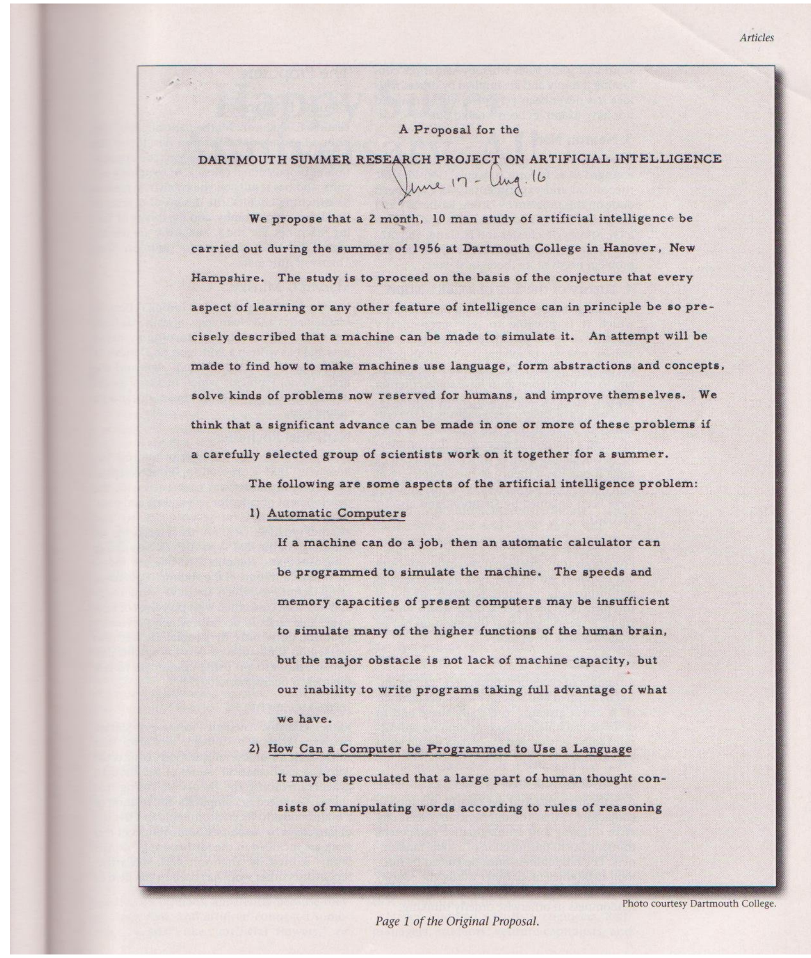
The following are some aspects of the artificial intelligence problem:

1. Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. The speeds and memory capacities of present computers may be insufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of machine capacity, but our inability to write programs taking full advantage of what we have.

2. How Can a Computer be Programmed to Use a Language

It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning and rules of conjecture. From this point of view, forming a generalization consists of admitting a new



Articles

A Proposal for the

DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE

June 17 - Aug. 16

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

The following are some aspects of the artificial intelligence problem:

1) Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. The speeds and memory capacities of present computers may be insufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of machine capacity, but our inability to write programs taking full advantage of what we have.

2) How Can a Computer be Programmed to Use a Language

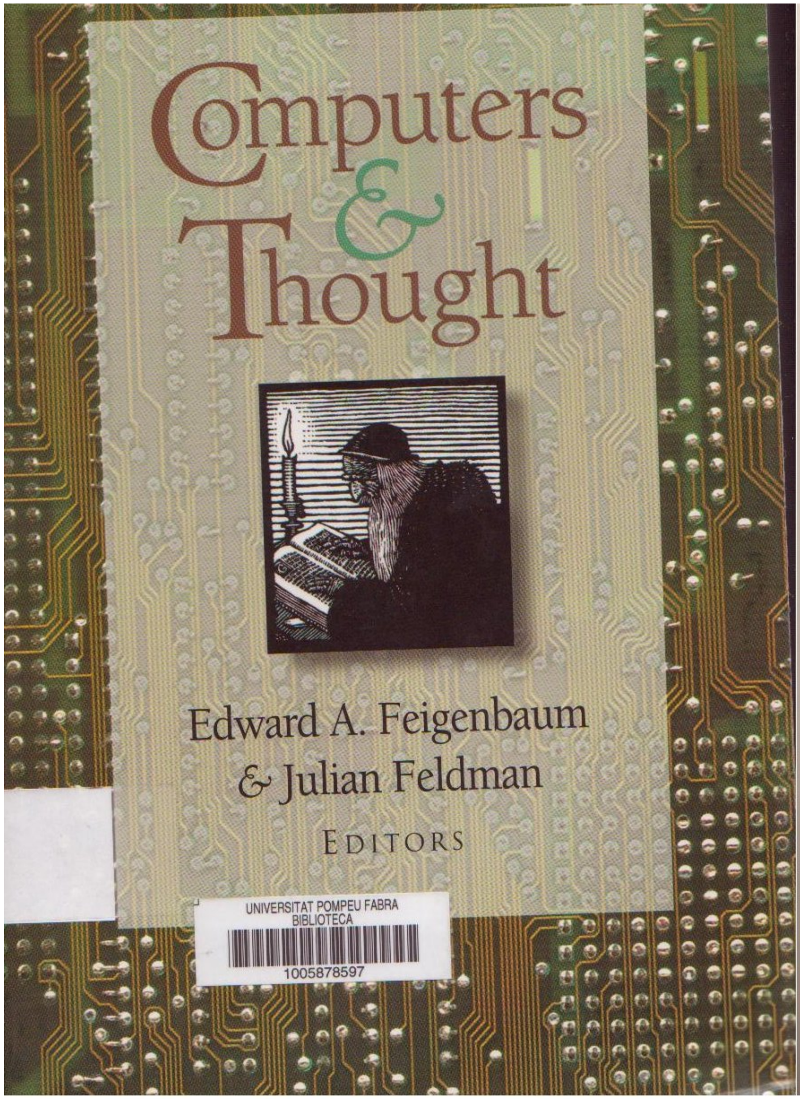
It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning

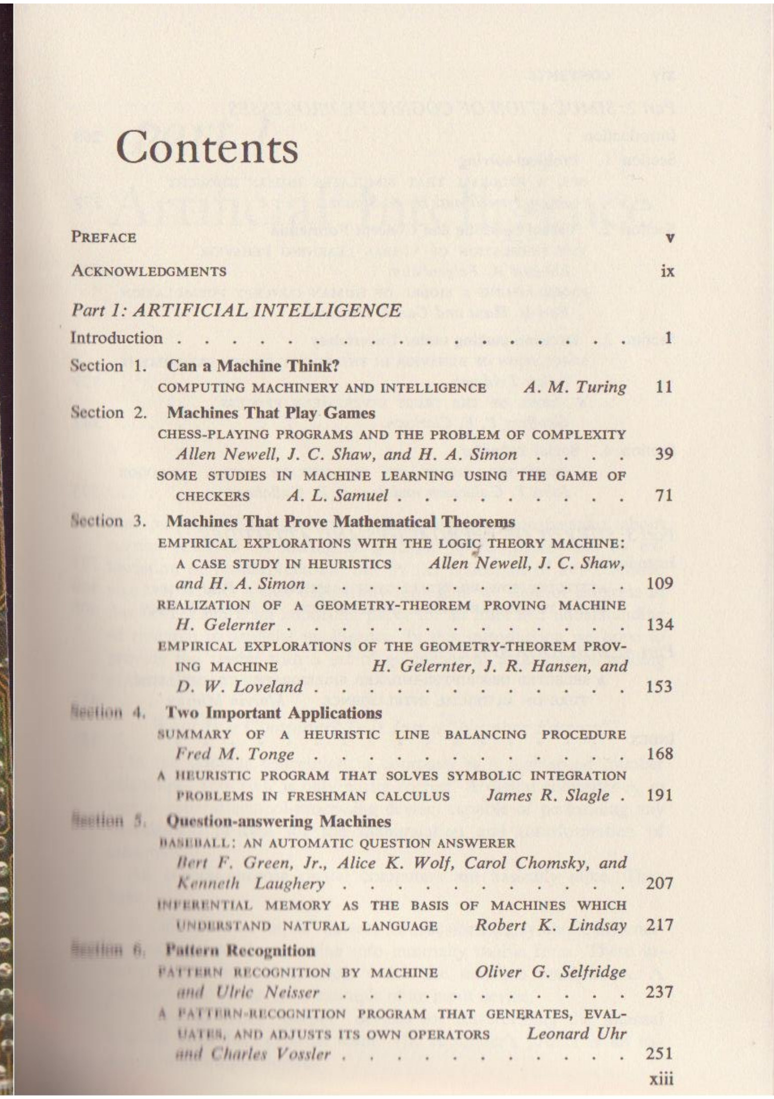
Photo courtesy Dartmouth College.

Page 1 of the Original Proposal.

"The proposal (for the meeting) is to proceed on the basis of the conjecture that every aspect of . . . intelligence can in principle be so precisely described that a machine can be made to simulate it"

Computers and Thought 1963





Contents

xiii

An early collection of AI papers and programs for playing chess and checkers, proving theorems in logic and geometry, planning, etc.

Importance of Programs in Early AI Work

In preface of 1963 edition of *Computers and Thought*

We have tried to focus on papers that report results. *In this collection, the papers . . . describe actual working computer programs . . . Because of the limited space, we chose to avoid the more speculative . . . pieces.*

In preface of 1995 AAAI edition

A critical selection criterion was that the paper had to describe . . . a running computer program . . . All else was talk, philosophy not science . . . (L)ittle has come out of the "talk".

AI, Programming, and AI Programming

Many of the key AI contributions in 60's, 70's, and early 80's had to to with **programming** and the **representation of knowledge** in **programs**:

- Lisp (Functional Programming)
- Prolog (Logic Programming)
- Rule-based Programming
- Interactive Programming Environments and Lisp Machines
- Frame, Scripts, Semantic Networks
- 'Expert Systems' Shells and Architectures

(Old) AI methodology: Theories as Programs

- For writing an AI dissertation in the 60's, 70's and 80's, it was common to:
  - ▷ pick up a task and domain X
  - ▷ analyze/introspect/find out how task is solved
  - ▷ capture this reasoning in a program
- The dissertation was then
  - ▷ a **theory** about X (scientific discovery, circuit analysis, computational humor, story understanding, etc), and
  - ▷ a **program** implementing the theory, **tested** over a few examples.

Many great ideas came out of this work . . . but there was a problem . . .

Methodological Problem: Generality

Theories expressed as programs cannot be proved wrong: when a program fails, it can always be blamed on 'missing knowledge'

Three approaches to this problem

- narrow the domain (expert systems)
  - ▷ problem: lack of generality
- accept the program is just an illustration, a demo
  - ▷ problem: limited scientific value
- fill up the missing knowledge (intuition, commonsense)
  - ▷ problem: not successful so far

AI in the 80's

The knowledge-based approach reached an **impasse** in the 80's, a time also of debates and controversies:

- **Good Old Fashioned AI** is "rule application" but intelligence is not (Haugeland)
- **Situated AI:** representation not needed and gets in the way (Brooks)
- **Neural Networks:** inference needed is not logical but probabilistic (PDP Group)

Many of these criticisms of mainstream AI partially valid then; less valid now.

Research on **models** and **solvers** over last 20-30 years provide a handle on **generality** problem in AI and related issues . . .

AI Research in 2013

Recent issues of AIJ, JAIR, AAAI or IJCAI shows papers on:

1. **SAT and Constraints**
2. **Search and Planning**
3. **Probabilistic Reasoning**
4. **Probabilistic Planning**
5. Inference in First-Order Logic
6. Machine Learning
7. Natural Language
8. Vision and Robotics
9. Multi-Agent Systems

I'll focus on 1–4: these areas often deemed about **techniques**, but more accurate to regard them as **models** and **solvers**.

Example: Solver for Linear Equations

$$\textit{Problem} \Longrightarrow \boxed{\textit{Solver}} \Longrightarrow \textit{Solution}$$

- **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 . . .

AI Models and Solvers

$$\textit{Problem} \Longrightarrow \boxed{\textit{Solver}} \Longrightarrow \textit{Solution}$$

- Some basic models and solvers currently considered in AI:
  - ▷ **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** . . .

SAT and CSPs

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

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

- 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)
- **Constraint Satisfaction Problems (CSPs)** generalize SAT by accommodating non-boolean variables as well, and constraints that are not clauses

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 $\Box$ derived, find 'causes' S of $\Box$, 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)**

Related tasks: Enumeration and Optimization SAT Problems

- **Weighted MAX-SAT**: find assignment σ that minimizes total cost $w(C)$ of violated clauses

$$\sum_{C:\sigma \not\models C} w(C)$$

- **Weighted Model Counting**: Adds up 'weights' of satisfying assignments:

$$\sum_{\sigma:\sigma \models T} \prod_{L \in \sigma} w(L)$$

SAT methods extended to these other tasks, closely connected to **probabilistic** reasoning tasks over **Bayesian Networks**:

- **Most Probable Explanation (MPE)** easily cast as Weighted MAX-SAT
- **Probability Assessment** $P(X|Obs)$ easily cast as Weighted Model Counting

Some of the best BN solvers built over these formulations . . .

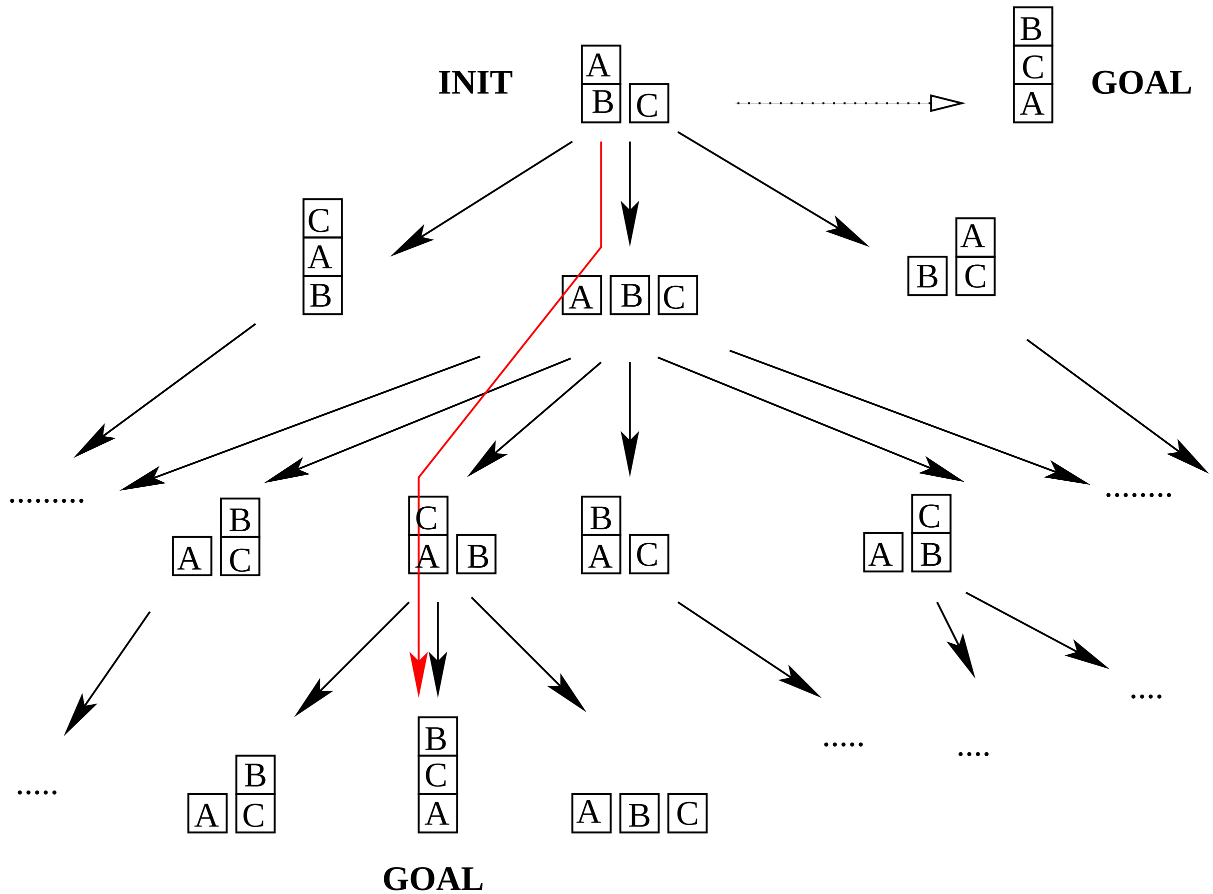
Basic (Classical) Planning Model and Task

- Planning is the **model-based approach** to autonomous behavior,
- A system 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**

$$\textit{Model} \Longrightarrow \boxed{\textit{Box}} \Longrightarrow \textit{Action sequence}$$

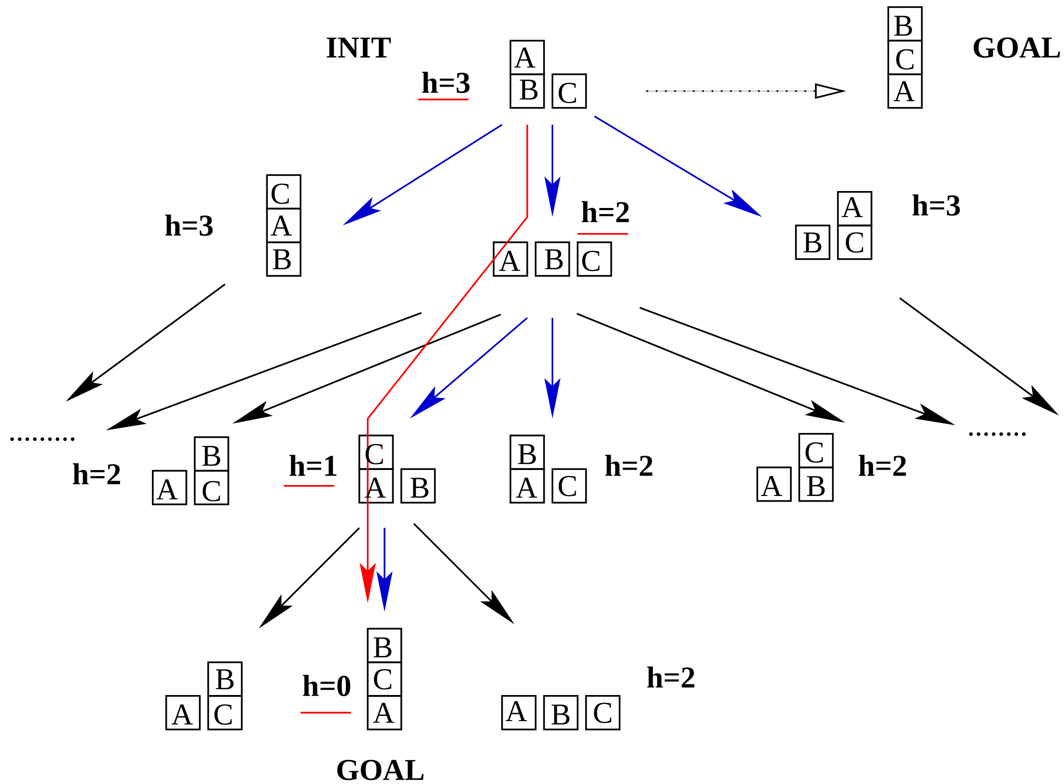
- **Complexity:** NP-hard; i.e., exponential in number of vars **in worst case**
- **Box** is generic; it should work on any domain no matter what variables are about

Concrete 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?

Problem Solved with 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).

A bit of Cog Science: Models, solvers, and inference

- We have learned a lot about **effective inference mechanisms** in last 20–30 years from work on **domain-independent** solvers

- The problem of AI in the 80's (the 'knowledge-based' approach), was probably lack of **mechanisms**, not only **knowledge**.

- Commonsense based not only on massive amounts of knowledge, but also **massive amounts of fast and effective but unconscious inference**

- This is clearly true for **Vision** and **NLP**, but likely for **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)*

The appraisals/heuristics $h(s)$ from a cognitive point of view

- they are **opaque** and thus cannot be **conscious**

  meaning of symbols in the relaxation is not the normal meaning; e.g., objects can be at many places at the same time as old locations not deleted

- they are **fast and frugal** (linear-time), but unlike the 'fast and frugal heuristics' of Gigerenzer et al. are **general**

  they apply to all problems fitting the model (planning problems)

- they play the role of **'gut feelings'** or **'emotions'** according to De Sousa 87, Damasio 94, Evans 2002, Gigerenzer 2007 . . .

  providing a guide to action while avoiding infinite regresses in the decision process

Old Debates, New Insights?

- **Logic vs. Probabilistic Inference:** don't look all that different now
- **Intelligence can't be rules all the way down:** not in planning
- **Symbolic vs. Non-Symbolic:** are (learned) BNets and MDPs 'symbolic'?
- **GOFAI vs. Mainstream AI:** is GOFAI just 'old' AI, no longer current?
- **Solvers vs. Architectures:** architectures don't "solve" anything; solvers do.
- **Mind as Architecture or Solver?** Adaptive, heuristic, multiagent POMDP solver?
- . . .

Summary: AI and Automated Problem Solving

- A **research agenda** that has emerged in last 20 years: **solvers** for a range of **intractable models**

- **Solvers** unlike other programs are **general** as they do not target individual problems but families of problems (**models**)

- The challenge is **computational**: how to scale up

- Sheer **size of problem** shouldn't be impediment to meaningful solution

- **Structure** of given problem must be recognized and **exploited**

- Lots of room for **ideas** but methodology **empirical**

- While agenda is technical, resulting ideas likely to be relevant for understanding **general intelligence** and **human cognition**

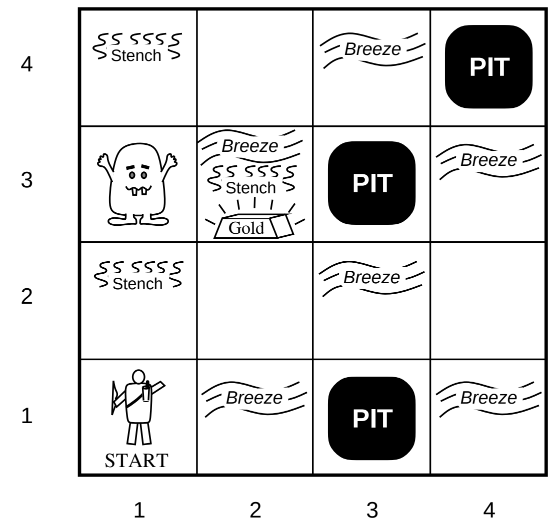
Introduction to Planning: Motivation

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

Wumpus World PEAS description

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

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



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

Sensors Breeze, Glitter, Smell

Autonomous Behavior in AI: The Control Problem

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

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

Approaches not orthogonal though; and successes and limitations in each . . .

Settings where greater autonomy required

- **Robotics**
- **Video-Games**
- **Web Service Composition**
- **Aerospace**
- ⋮

Solution 1: Programming-based Approach

Control specified by programmer; e.g.,

- *don't move into a cell if not known to be safe (no Wumpus or Pit)*
- *sense presence of Wumpus or Pits nearby if this is not known*
- *pick up gold if presence of gold detected in cell*
- . . .

Advantage: domain-knowledge easy to express

Disadvantage: cannot deal with situations not anticipated by programmer

Solution 2: Learning-based Approach

- **Unsupervised** (Reinforcement Learning):
  - ▷ penalize agent each time that it 'dies' from Wumpus or Pit
  - ▷ reward agent each time it's able to pick up the gold, . . .
- **Supervised** (Classification)
  - ▷ learn to classify actions into good or bad from info provided by teacher
- **Evolutionary:**
  - ▷ from pool of possible controllers: try them out, select the ones that do best, and mutate and recombine for a number of iterations, keeping best

Advantage: does not require much knowledge in principle

Disadvantage: in practice though, right features needed, incomplete information is problematic, and unsupervised learning is slow . . .

Solution 3: Model-Based Approach

- specify model for problem: actions, initial situation, goals, and sensors
- let a solver compute controller automatically

$$\begin{array}{c} \textit{Actions} \\ \textit{Sensors} \\ \textit{Goals} \end{array} \longrightarrow \boxed{\textit{SOLVER}} \longrightarrow \textit{CONTROLLER} \;\; \begin{array}{c} \xrightarrow{actions} \\ \xleftarrow{observations} \end{array} \;\; \textit{World}$$

Advantage: flexible, clear, and domain-independent

Disadvantage: need a model; computationally intractable

Model-based approach to intelligent behavior called **Planning** *in AI*

Basic State Model for 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** is a sequence of applicable actions that maps s_0 into S_G, and it is **optimal** if it minimizes **sum of action costs** (e.g., $\#$ of steps)

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 $a \in A(s)$

- initial **belief state** b_0
- final **belief states** b_F
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$

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

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

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

Models, Languages, and Solvers

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

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

- Many models, many solution forms: uncertainty, feedback, costs, . . .
- Models described in suitable **planning languages** (Strips, PDDL, PPDDL, . . .) where **states** represent interpretations over the language.

Language for Classical Planning: Strips

- A **problem** in Strips is a tuple $P = \langle F, O, I, G \rangle$:
  - ▷ F stands for set of all **atoms** (boolean vars)
  - ▷ O stands for set of all **operators** (actions)
  - ▷ $I \subseteq F$ stands for **initial situation**
  - ▷ $G \subseteq F$ stands for **goal situation**

- Operators $o \in O$ **represented** by
  - ▷ the **Add** list $Add(o) \subseteq F$
  - ▷ the **Delete** list $Del(o) \subseteq F$
  - ▷ the **Precondition** list $Pre(o) \subseteq F$

From Language to Models

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

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

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

– Slight language extensions often convenient (e.g., **negation** and **conditional effects**); some required for describing richer models (costs, probabilities, ...).

Example: Blocks in Strips (PDDL Syntax)

```
(define (domain BLOCKS)
  (:requirements :strips) ...
  (:action pick_up
           :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (hol
  (:action put_down
           :parameters (?x)
           :precondition (holding ?x)
           :effect  (and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x)))
  (:action stack
           :parameters (?x ?y)
           :precondition (and (holding ?x) (clear ?y))
           :effect  (and (not (holding ?x)) (not (clear ?y)) (clear ?x)(handempty)
                         (on ?x ?y))) ...
(define (problem BLOCKS_6_1)
   (:domain BLOCKS)
   (:objects F D C E B A)
   (:init (CLEAR A) (CLEAR B) ...  (ONTABLE B) ... (HANDEMPTY))
   (:goal (AND (ON E F) (ON F C) (ON C B) (ON B A) (ON A D))))
```

Example: Logistics in Strips PDDL

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

Example: 15-Puzzle in PDDL

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

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

Computation: how to solve Strips planning problems?

- **Key issue:** exploit two roles of **language:**
  - **specification:** concise model description
  - **computation:** reveal useful heuristic info

- **Two traditional approaches:** search vs. decomposition
  - explicit **search** of the state model $S(P)$ direct but not effective til recently
  - **near decomposition** of the planning problem thought a better idea

Computational Approaches to Classical Planning

- **Strips algorithm** (70's): Total ordering planning backward from Goal; work always on **top** subgoal in stack, delay rest
- **Partial Order (POCL) Planning** (80's): work on **any** subgoal, resolve threats; UCPOP 1992
- **Graphplan** (1995 – . . .): build graph containing all possible **parallel** plans up to certain length; then extract plan by searching the graph backward from Goal
- **SatPlan** (1996 – . . .): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- **Heuristic Search Planning** (1996 – . . .): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P
- **Model Checking Planning** (1998 – . . .): search state space $\mathcal{S}(P)$ with 'symbolic' BrFS where sets of states represented by formulas implemented by BDDs

State of the Art in Classical Planning

- significant **progress** since Graphplan (Blum & Furst 95)
- **empirical methodology**
  - ▷ standard PDDL language
  - ▷ planners and benchmarks available; competitions
  - ▷ focus on performance and scalability
- **large problems solved** (non-optimally)
- different **formulations** and **ideas**

We'll focus on **two formulations:**

- (Classical) Planning as **Heuristic Search**, and
- (Classical) Planning as **SAT**

Classical Planning and Heuristic Search

Models, Languages, and Solvers (Review)

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

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

- Many models, many solution forms: uncertainty, feedback, costs, . . .

- Models described in suitable **planning languages** (Strips, PDDL, PPDDL, . . .) where **states** represent interpretations over the language.

State Model for Classical Planning

- finite and discrete state space S
- an initial state $s_0 \in S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- a transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
- action costs $c(a, s) > 0$

A **solution** is a sequence of applicable actions a_i, $i = 0, \ldots, n$, that maps the initial state s_0 into a goal state $s \in S_G$; i.e., $s_{n+1} \in S_G$ and for $i = 0, \ldots, n$

$$s_{i+1} = f(a, s_i) \text{ and } a_i \in A(s_i)$$

Optimal solutions minimize total cost $\sum_{i=0}^{i=n} c(a_i, s_i)$

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 Problem P to State Model $S(P)$

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

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

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

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

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

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, DFS B&B, LRTA*, . . .*

General Search Scheme

```
Solve(Nodes)
  if Empty Nodes  -> Fail
  else Let Node = Select-Node Nodes
       Let Rest =  Nodes - Node
      if Node is Goal -> Return Solution
      else Let Children = Expand-Node Node
           Let New-Nodes = Add-Nodes Children Rest
           Solve(New-Nodes)
```

- Different algorithms obtained by suitable instantation of
  - `Select-Node` *Nodes*
  - `Add-Nodes` *New-Nodes Old-Nodes*
- Nodes are data structures that contain state and bookkeeping info; initially `Nodes` $= \{root\}$
- Notation $g(n)$, $h(n)$, $f(n)$: accumulated cost, heuristic and evaluation function; e.g. in A*, $f(n) \stackrel{\text{def}}{=} g(n) + h(n)$

Some instances of general search scheme

- **Depth-First Search** expands 'deepest' nodes n first
  - `Select-Node` $Nodes$: Select First Node in $Nodes$
  - `Add-Nodes` $New\ Old$: Puts New before Old
  - Implementation: Nodes is a **Stack** (LIFO)

- **Breadth-First Search** expands 'shallowest' nodes n first
  - `Select-Node` $Nodes$: Selects First Node in $Nodes$
  - `Add-Nodes` $New\ Old$: Puts New after Old
  - Implementation: Nodes is a **Queue** (FIFO)

Additional instances of general search scheme

- **Best First Search** expands best nodes n first; $\min f(n)$
  - ▷ `Select-Node` $Nodes$: Returns n in Nodes with min $f(n)$
  - ▷ `Add-Nodes` New Old: Performs ordered merge
  - ▷ Implementation: Nodes is a **Heap**
  - ▷ Special cases

    Uniform cost/Dijkstra: $f(n) = g(n)$
    A*: $f(n) = g(n) + h(n)$
    WA*: $f(n) = g(n) + Wh(n)$, $W \geq 1$
    Greedy Best First: $f(n) = h(n)$

- **Hill Climbing** expands best node n first and **discards others**
  - ▷ `Select-Node` $Nodes$: Returns n in Nodes with min $h(n)$
  - ▷ `Add-Nodes` New Old: Returns New; discards Old

Variations of general search scheme: DFS Bounding

```
Solve(Nodes,Bound)

  if Empty Nodes  -> Report-Best-Solution-or-Fail
  else
     Let Node = Select-Node Nodes
     Let Rest =  Nodes - Node

    if f(Node)  > Bound
           Solve(Rest,Bound)    ;;;    PRUNE NODE n

    else if Node is Goal -> Process-Solution Node Rest
         else
           Let Children = Expand-Node Node
           Let New-Nodes = Add-Nodes Children Rest
           Solve(New-Nodes,Bound)
```

Select-Node & Add-Nodes as in DFS

Some instances of general bounded search scheme

- **Iterative Deepening (ID)**
  - ▷ Uses $f(n) = g(n)$
  - ▷ Calls `Solve` with bounds 0, 1, .. til solution found
  - ▷ `Process-Solution` returns Solution

ıIterative Deepening A* (IDA*)

  - ▷ Uses $f(n) = g(n) + h(n)$
  - ▷ Calls `Solve` with bounds $f(n_0)$, $f(n_1)$, $\ldots$ where $n_0 = root$ and n_i is cheapest node pruned in iteration $i - 1$
  - ▷ `Process-Solution` returns Solution

- **Branch and Bound**
  - ▷ Uses $f(n) = g(n) + h(n)$
  - ▷ Single call to `Solve` with high (Upper) `Bound`
  - ▷ `Process-Solution:` updates `Bound` to Solution Cost minus 1 & calls `Solve(Rest,New-Bound)`

Properties of Algorithms

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

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

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

– BrFS optimal when costs are uniform

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

A*: Details, Properties

- A* stores in memory **all nodes visited**
- Nodes either in **Open** (search frontier) or **Closed**
- When nodes expanded, children looked up in **Open** and **Closed** lists
- Duplicates prevented, only best (equivalent) node kept

– A* is **optimal** in another sense: no other algorithm expands less nodes than A* with same heuristic function *(this doesn't mean that A* is always fastest)*

– A* expands 'less' nodes with **more informed heuristic**, h_2 more informed that h_1 if $0 < h_1 < h_2 \leq h^*$

– A* won't re-open nodes if heuristic is **consistent** (**monotonic**); i.e., $h(n) \leq c(n, n') + h(n')$ for children n' of n.

Practical Issues: Search in Large Spaces

- Exponential-memory algorithms like A* **not feasible** for large problems
- **Time and memory** requirements can be lowered significantly by multiplying heuristic term $h(n)$ by a constant $W > 1$ (WA*)
- Solutions **no longer optimal** but at most W times from optimal
- For large problems, only feasible optimal algorithms are **linear-Memory** algorithms such as IDA* and B&B
- Linear-memory algorithms often use **too little memory** and may visit fragments of search space many times
- It's common to extend IDA* in practice with so-called **transposition tables**
- Optimal solutions have been reported to problems with **huge state spaces** such 24-puzzle, Rubik's cube, and Sokoban (Korf, Schaeffer); e.g. $|S| > 10^{20}$

Learning Real Time A* (LRTA*)

- LRTA* is a very interesting **real-time** search algorithm (Korf 90)
- It's like a **hill-climb** or **greedy** search that **updates** the heuristic V as it moves, starting with $V = h$.

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

- Two remarkable **properties**
  - ▷ **Each trial** of LRTA gets eventually to the goal if space connected
  - ▷ **Repeated trials** eventually get to the goal **optimally**, if h **admissible**!
- Generalizes well to **stochastic actions** (MDPs)

Heuristics: where they come from?

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

- Examples:

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

- Yet

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

We'll get more into this as we get back to planning . . .

Classical Planning as Heuristic Search

From Strips Problem P to State Model $S(P)$ (Review)

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
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Pre(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

How to solve $S(P)$?

Heuristic Search Planning

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

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

Heuristics for Classical Planning

- Key development in planning in the 90's, is automatic extraction of **heuristic functions** to guide search for plans

- The general idea was known: heuristics often **explained** as **optimal** cost functions of **relaxed** (simplified) problems (Minsky 61; Pearl 83)

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

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

- Heuristic h^+ **intractable** but easy to **approximate**; i.e.
  - ▷ *computing* **optimal** *plan for P^+ is* **intractable**, but
  - ▷ *computing a non-optimal plan for P^+ (***relaxed plan***) easy*

- State-of-the-art heuristics as in FF or LAMA still rely on P^+ . . .

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, assume **independence**:

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

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

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

Heuristic not admissible but informative and fast

Max Heuristic

- For all **atoms** p:

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

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

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

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

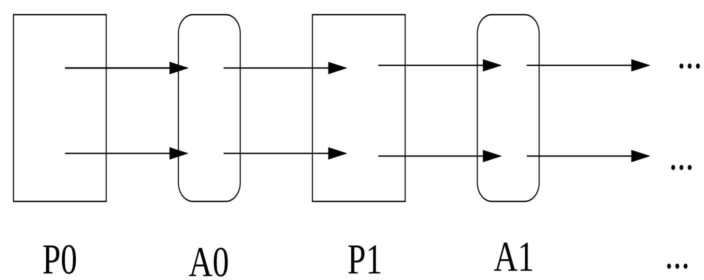
$$h_{max}(s) \stackrel{\text{def}}{=} h(Goals;s)$$

Heuristic admissible but not very informative . . .

Max Heuristic and (Relaxed) Planning Graph

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

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



– Graph implicitly **represents** max heuristic:

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

Heuristics, Relaxed Plans, and FF

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

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

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

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

- The **relaxed plan** $\pi(s)$ for the **goals** obtained by **planner FF** using $h = h_{max}$

- More accurate h obtained then from **relaxed plan** π as

$$h(s) = \sum_{a \in \pi(s)} cost(a)$$

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

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

- More recent planners like **FF** and **LAMA** go beyond this in two ways

- They exploit the structure of the heuristic and/or problem further:
  - ▷ **Helpful Actions**
  - ▷ **Landmarks**

- 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**. Not always though; try them!

Experiments with state-of-the-art classical planners

		FF			FD			PROBE			LAMA'11			BFS(f)		
Domain	I	S	Q	T	S	Q	T	S	Q	T	S	Q	T	S	Q	T
8puzzle	50	49	52.61	0.03	50	52.30	0.18	50	60.94	0.09	49	92.54	0.18	50	45.30	0.20
Barman	20	0	–	–	20	197.90	84.00	20	169.30	12.93	20	192.15	8.39	20	174.45	281.28
BlocksW	50	44	39.36	66.67	50	104.24	0.46	50	43.88	0.25	50	89.96	0.41	50	54.24	2.25
Cybersec	30	4	29.50	0.73	28	36.58	859.24	24	50.73	48.29	30	35.27	880.06	28	36.92	63.79
Depots	22	22	51.82	32.72	17	110.25	91.86	22	88.88	1.45	21	43.56	3.58	22	39.56	69.11
Driver	20	16	25.00	14.52	20	50.67	1.26	20	60.17	1.49	20	46.22	1.51	18	48.06	140.93
Elevators	30	30	85.73	1.00	30	92.57	3.20	30	107.97	26.66	30	97.07	4.69	30	129.13	93.88
Ferry	50	50	27.68	0.02	50	30.08	0.09	50	44.80	0.02	50	26.86	0.08	50	31.28	0.03
Floortile	20	5	44.20	134.29	3	39.00	6.91	5	40.50	106.97	5	40.00	8.94	7	36.50	4.15
Freecell	20	20	64.00	22.95	20	61.06	26.55	20	62.44	41.26	19	67.78	27.35	20	64.39	13.00
Grid	5	5	61.00	0.27	5	61.60	4.95	5	58.00	9.64	5	70.60	4.84	5	70.60	7.70
Gripper	50	50	76.00	0.03	50	152.62	0.17	50	152.66	0.06	50	92.76	0.15	50	152.66	0.38
Logistics	28	28	41.43	0.03	28	77.11	0.18	28	55.36	0.09	28	73.64	0.17	28	87.04	0.12
Miconic	50	50	30.38	0.03	50	39.80	0.07	50	44.80	0.01	50	31.02	0.06	50	34.46	0.01
Mprime	35	34	9.53	14.82	35	8.37	9.50	35	12.97	26.67	35	8.60	10.30	35	10.17	19.30
Mystery	30	18	6.61	0.24	19	6.86	1.87	25	7.71	1.08	22	7.29	1.70	27	7.07	0.93
NoMyst	20	4	19.75	0.23	6	22.40	1.96	5	23.20	2.73	11	23.00	1.77	19	22.60	0.78
OpenSt	30	30	155.67	6.86	30	130.11	5.97	30	134.14	64.55	30	130.18	3.49	29	125.89	129.06
OpenSt6	30	30	136.17	0.38	30	222.67	5.39	30	224.00	48.89	30	140.60	4.89	30	139.13	40.19
ParcPr	30	30	42.73	0.06	27	35.79	1.97	28	70.92	0.26	30	70.54	0.28	27	70.42	6.72
Parking	20	3	88.33	945.86	20	74.86	330.76	17	143.36	685.47	19	129.57	361.19	17	83.43	562.39
Pegsol	30	30	25.50	7.61	30	25.97	0.80	30	25.17	8.60	30	26.07	2.76	30	24.20	1.17
Pipes-N	50	35	34.34	12.77	44	75.50	7.94	45	46.73	3.18	44	54.41	11.11	47	58.39	35.97
Pipes-T	50	20	31.45	87.96	40	73.33	99.06	43	54.19	88.47	41	69.83	35.28	40	39.14	216.25
PSR-s	50	42	16.92	63.05	50	14.61	0.27	50	17.20	0.07	50	14.65	0.31	48	18.14	2.57
Rovers	40	40	100.47	31.78	40	153.18	13.69	40	131.20	24.19	40	108.53	17.90	40	126.30	44.20
Satellite	20	20	37.75	0.10	20	40.90	0.78	20	37.05	0.84	20	42.05	0.78	20	36.05	1.26
Scan	30	30	31.87	70.74	28	30.04	7.30	28	25.15	5.59	28	28.04	8.14	27	29.37	7.40
Sokoban	30	26	213.38	26.61	28	204.14	12.44	25	231.52	39.63	28	231.81	184.38	23	218.52	125.12
Storage	30	18	16.28	39.17	20	17.72	3.20	21	14.56	0.07	18	24.56	8.15	20	20.94	4.34
Tidybot	20	15	63.20	9.78	15	66.00	338.14	19	52.67	33.50	16	62.60	102.52	18	63.27	207.85
Tpp	30	28	122.29	53.23	30	127.93	16.95	30	152.53	60.95	30	205.37	18.72	30	110.13	126.03
Transport	30	29	117.41	167.10	30	97.57	12.75	30	125.63	38.87	30	215.90	76.18	30	97.57	46.64
Trucks	30	11	27.09	3.84	17	26.00	0.65	8	26.75	113.54	16	24.75	0.53	15	26.50	8.59
Visitall	20	6	450.67	38.22	7	3583.86	166.35	19	411.71	9.02	20	468.00	4.68	20	339.00	4.58
WoodW	30	17	32.35	0.22	30	57.13	18.40	30	41.13	15.93	30	79.20	12.45	30	41.13	19.12
Zeno	20	20	30.60	0.17	20	37.45	2.68	20	44.90	6.18	20	35.80	4.28	20	37.70	77.56
Summary	1150	909	67.75	51.50	1037	168.60	57.78	1052	83.64	41.28	1065	86.51	48.98	1070	74.32	63.91

Heuristic Search Planners (1997–2012)

- **HSP, 1998:** GBFS guided by **heuristic** h_{add}; solves **729** out of 1150 problems
- **FF, 2000:** Incomplete **EHC search** followed by GBFS with $h_{\rm FF}$; solves **909**
- **FD, 2004:** GBFS with **two queues:** helpful and unhelpful, ordered by $h_{\rm FF}$; **1037**
- **LAMA, 2008:** GBFS with **four queues:** helpful and unhelpful for **landmark** h too; **1065**
- **PROBE, 2011:** Plain GBFS that throws poly-time **probe** from every expanded node; solves **1072**
- **BFS(f), 2012:** Plain GBFS with $h(s) \in [1,6]$ based on helpful and **width** info, and tie-breaker based on landmark h and h_{add}

EHC, Helpful Actions, Landmark Heuristic

- **EHC:** On-line, incomplete planning algorithm: from current state s uses **breadth-first search** and **helpful actions only** to find state s' such that $h(s') < h(s)$
  - ▷ **Helpful action:** applicable action a in s is **helpful** when a adds goal or precondition of an action in **relaxed plan** from s that is not true in s

- **Landmark:** is atom p that is made true by all plans (e.g., $clear(B)$ landmark is block beneath B not well placed)
  - ▷ **Computing landmarks 1:** sufficient criterion for p being a landmark is that **relaxed problem** not solvable **without** the actions that add p.
  - ▷ **Computing landmarks 2:** complete set of landmarks for **delete-relaxation** can be computed in poly-time once, as **preprocessing**
  - ▷ **Landmark heuristic:** just count the **number of unachieved landmarks**. It extends classical **number of unachieved goals** heuristics, and achieves a complete form of **problem decomposition**

- **Multi-Queue Best First Search:** it maintains and alternates between **multiple open lists**, and doesn't leave any open list waiting for ever (fairness)

Sructure of classical planning benchmarks: why are they easy?

- Most planning benchmarks are easy although planning is NP-hard.
- Problem considered in area called *tractable planning*, but gap with existing benchmarks closed only recently
- Graphical models such as CSPs and Bayesian Network are also NP-hard, yet some easy problems can be identify with a **treewidth** measured associated with underlying graph
- CSP and BNet algorithms are **exponential in treewidth**
- **Question:** can suitable **width** notion be formulated to bound the complexity of **planning** so that easy problems turn out to have low width?

Width: Definition

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
- A valid chain $t_0 \rightarrow t_1 \rightarrow \ldots \rightarrow t_n$ **implies** G if all **optimal plans** for t_n are also **optimal plans** for G
- The **size** of the chain is the **size of largest** t_i in the chain
- **Width** of P is **size of smallest** chain that **implies** goal G of P

Theorem 1: A problem P can be solved in time exponential in its **width.**

Theorem 2: Most planning domains (Blocks, Logistics, Gripper, . . .) have a **bounded and small width**, independent of problem **size**, provided that goals are **single atoms**

Width: Basic Algorithm

The **novelty** of a newly generated state s during a search is the **size of the smallest tuple of atoms** t that is **true** in s and **false** in all previously generated states s'. If no such tuple, the novelty of s is $n+1$ where n is number of problem vars.

- *IW*(i) is **breadth-first** search that **prunes** newly generated states s when $novelty(s) > i$.

- *IW* is **sequence of calls** *IW*(i) for $i = 0, 1, 2, \ldots$ over problem P until problem solved or i exceeds number of vars in problem

IW solves P in time exponential in the width of P

Iterative Width: Experiments for Single Atomic Goals

- *IW*, while simple and blind, is a pretty **good algorithm** over benchmarks when goals restricted to **single atoms**
- This is no accident, **width** of benchmarks domains is **small** for such goals
- Tests over domains from previous IPCs. For **each instance** with N goal atoms, N **instances** created with a **single goal**
- Results quite remarkable: *IW* is much better than **blind-search**, and as good as Greedy Best-First Search with heuristic h_{add}

# Instances	*IW*	*ID*	*BrFS*	$GBFS + h_{add}$
37921	91%	24%	23%	91%

Sequential IW: Using IW Sequentially to Solve Joint Goals

SIW runs *IW* iteratively, until **one more goal** achieved (hill climbing)

		Serialized *IW* (*SIW*)				GBFS + h_{add}		
Domain	I	S	Q	T	M/Aw_e	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

Width and Structure in Planning

Notion of width doesn't explain why planners do well in most benchmarks, but it suggests that most benchmarks are 'easy' because:

- The domains have a **low width** when the goals are **single atoms**, and
- **Conjunctive goals** are easy to **serialize** in these domains

If you want 'hard' problems, then look for

- Domains that have **high width** for single atomic goals, or
- Domains with conjunctive goals are that are **not** easy to serialize

Few benchmarks appear to have high width (Hanoi), although some are not easy to serialize (e.g., Sokoban)

Classical Planning as SAT and Variations

SAT and SAT Solvers

- SAT is the problem of determining whether a set of **clauses** or **CNF formula** is satisfiable
- A clause is disjunction of **literals** where a literal is a **propositional symbol** or its **negation**

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

- Many problems can be mapped into SAT such as Planning, Scheduling, CSPs, Verification problems etc.
- SAT is an **intractable problem** (exponential in the worst case unless P=NP) yet very large SAT problems can be solved in practice
- Best SAT algorithms not based on either pure **case analysis** (model theory) or **resolution** (proof theory), but **combination** of both

Davis and Putnam Procedure for SAT

- DP (DPLL) is a sound and complete proof procedure for SAT that uses resolution in a restricted form called **unit resolution**, in which one **parent clause** must be **unit clause**

- Unit resolution is very efficient (poly-time) but **not complete** (Example: $q \vee p$, $\neg q \vee p$, $q \vee \neg p$, $\neg q \vee \neg p$)

- When **unit resolution** gets stuck, DP picks undetermined Var, and **splits** the problem in two: one where Var is true, the other where it is false (**case analysis**)

```
  DP(clauses)
    Unit-resolution(clauses)
    if Contradiction, Return False
    else if all VARS determined, Return True
*   else pick non-determined VAR, and
    Return  DP(clauses + VAR)  OR  DP(clauses + NEG VAR)
```

Currently very large SAT problems can be solved. Criterion for **var selection** is critical, as **learning from conflicts** (not shown).

Planning as SAT

- Maps planning problem $P = \langle F, O, I, G \rangle$ with horizon n into a **set of clauses** $C(P, n)$, solved by **SAT solver** (satz,chaff,. . .).

- Theory $C(P, n)$ includes vars $p_0, p_1, \ldots, p_n$ and $a_0, a_1, \ldots, a_{n-1}$ for each $p \in F$ and $a \in O$

- $C(P, n)$ satisfiable **iff** there is a plan with length bounded by n

- Such a **plan** can be read from **truth valuation** that satisfies $C(P, n)$.

Theory $C(P, n)$ for Problem $P = \langle F, O, I, G \rangle$

- **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 $i = 0, 1, \ldots, n-1$, 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 where $O(p^+)$ and $O(p^-)$ stand for the actions that add and delete p resp.

  $p_i \wedge \bigwedge_{a \in O(p^-)} \neg a_i \supset p_{i+1}$

  $\neg p_i \wedge \bigwedge_{a \in O(p^+)} \neg a_i \supset \neg p_{i+1}$
- **Seriality:** For each $i = 0, \ldots, n-1$, if $a \neq a'$, $\neg(a_i \wedge a'_i)$

– This encoding is pretty simple doesn't work too well

– Alternative encodings used: parallelism (no seriality), NO-OPs, lower bounds, . . .

– Best current SAT planners are very good

Other methods in classical planning

- Regression Planning
- Graphplan
- Partial Order Causal Link (POCL) Planning

Regression Planning

Search backward from **goal** rather than forward from initial state:

- **initial** state σ_0 is G
- a **applicable** in σ if $Add(a) \cap \sigma \neq \emptyset$ and $Del(a) \cap \sigma = \emptyset$
- resulting state is $\sigma_a = \sigma - Add(a) + Prec(a)$
- terminal states σ if $\sigma \subseteq I$

Advantages/Problems:

\+ Heuristic $h(\sigma)$ for any σ can be computed by simple aggregation (max,sum, . . .) of estimates $g(p, s_0)$ for $p \in \sigma$ computed only **once** from s_0

\- Spurious states σ not reachable from s_0 often generated (e.g., where a block is on two blocks at the same time). A good h should make $h(\sigma) = \infty$ . . .

Variation: Parallel Regression Search

Search backward from goal assuming that **non-mutex actions** can be done in **parallel**

- The regression search is similar, except that **sets** of non-mutex actions A allowed: $Add(A) = \cup_{a \in A} Add(a)$, $Del(A) = \cup_{a \in A} Del(a)$, $Prec(A) = \cup_{a \in A} Prec(a)$.
- Resulting state from regression is $\sigma_A = \sigma - Add(A) + Prec(a)$

Advantages/Problems:

+ Sometimes easier to compute optimal **parallel** plans than optimal **serial** plans

+ Some heuristics provide tighter estimates of **parallel cost** than **serial cost** (e.g., $h = h1$)

- **Branching factor** in parallel search (either forward or backward) can be very large (2^n if n applicable actions).

Parallel Regression Search with NO-OPs

- Assumes 'dummy' operator **NO-OP(p)** for each p with $Prec = Add = \{p\}$ and $Del = \emptyset$
- A set of non-mutex actions A (possibly including NO-OPs) applicable in σ if $\sigma \subseteq Add(A)$ and $Del(A) \cap \sigma = \emptyset$
- Resulting state is $\sigma = Prec(A)$
- Starting state $\sigma_0 = G$ and terminal states $\sigma \subseteq I$

Advantages/Problems:

\- More actions to deal with

\+ Enables certain compilation techniques as in Graphplan . . .

Graphplan (Blum & Furst): First Version

- Graphplan does an IDA* parallel regression search with NO-OPs over **planning graph** containing **propositional** and **action layers** P_i and A_i, $i = 0, \ldots, n$
  - P_0 contains the atoms true in I
  - A_i contains the actions whose precs are true in P_i
  - P_{i+1} contains the **positive** effects of the actions in A_i
- planning graph built til layer P_n where G appears, then **search** for plans with horizon $n-1$ invoked with $Solve(G, n)$ where
  - $Solve(G, 0)$ succeds if $G \subseteq I$ and fails otherwise, and
  - $Solve(G, n)$ mapped into $Solve(Prec(A), n-1)$, where A **is a set of non-mutex actions in layer** in A_{n-1} that covers G, i.e., $G \subseteq Add(A)$.
- If search fails, n increased by 1, and process is repeated

Graphplan: Real version

- The IDA* search is **implicit**; heuristic $h(\sigma)$ encoded in planning graph as **index of first layer P_i that contains σ**

- This heuristic, as defined above, corresponds to the $\mathbf{hmax} = \mathbf{h1}$ heuristic; Graphplan actually uses a more powerful admissible heuristic akin to h_2 . . .

- Basic idea: extend **mutex** relations to **pairs** of actions and propositions in each layer $i > 0$ as follows:
  - p and q **mutex in** P_i if p and q are in P_i and the actions in A_{i-1} that support p and q are **mutex in** A_{i-1};
  - a and a' **mutex in** A_i if a and a' are in A_i, and they are **mutex** or $Prec(a) \cup Prec(a')$ contains a **mutex in** P_i

- The **index of first layer** in planning graph that contains a set of atoms P or actions A **without** a mutex, is a **lower bound**

- Thus, search can be **started** at level in which G appears without a mutex, and $Solve(P, i)$ needs to consider only sets of actions A in A_{i-1} that **do not contain a mutex**.

Partial Order Planning: Regression + Decomposition. Intuition

1. recursively **decompose** regression with goal $p_1, \ldots, p_n$ into n regressions with goals p_i, $i = 1, \ldots, n$;
2. **combine** resulting plans so that they **do not interfere** with each other

E.g.: let $G = \{p, q\}$, $I = \{r\}$, and two actions

$$a1\text{: } Prec(a1) = \{r\},\ Add(a1) = \{p\},\ Del(a1) = \{r\}$$

$$a2\text{: } Prec(a2) = \{r\},\ Add(a2) = \{q\},\ Del(a2) = \{\}$$

– $P1 = \{a1\}$ is a plan for p, and $P2 = \{a2\}$ a plan for q

– Yet $a1$ in $P1$ **deletes** a precondition of $a2$

– This 'threat' can be solved by forcing $a1$ **after** $a2$, i.e., $a2 \prec a1$.

Partial Order Causal Link planning is a formulation of POP that pursues 1 and 2 concurrently

Partial Plans and Causal Links

A **partial plan** P in POCL is a triple $(Steps, \mathcal{O}, CLs)$ where

– $Steps$ is a set of **actions** a_i

– $\mathcal{O}$ is a set of **precedence constraints** $a_i \prec a_j$

– CLs is a set of **causal links** $(a1, p, a2)$ meaning that that precondition p of $a2$ is achieved by action $a1$

- POCL extends partial plans til they become **complete** (to be defined)
- **States** σ in the search are partial plans
- **Initial state** (partial plan) is $P_0 = (\{Start, End\}, \{Start \prec End\}, \{\})$, where $Start$ and End are actions that summarize I and G: $Add(Start) = I$, $Prec(End) = G$

POCL Planning Algorithm

- A partial plan $P = (Steps, \mathcal{O}, CLs)$ is **complete** when ordering $\mathcal{O}$ is **consistent** and there is no **flaw** of the form:
  - ▷ **unsupported precondition:** a precond $p \in Prec(a)$ for $a \in Steps$ s.t. no CL (a', p, a) in CLs
  - ▷ **threatened causal link:** a CL (a', p, a) for $b \in Steps$ s.t. $p \in Del(b)$ and $a' \prec b \prec a$ is consistent with $\mathcal{O}$

- POCL search **starts** with the plan $P = P_0$ above, selecting a flaw in P, and trying each one of the repairs:
  - ▷ **Flaw #1:** fixed by selecting an action a', $p \in Add(a)$, and adding a' to $Steps$, $a' \prec a$ to $\mathcal{O}$, and (a', p, a) to CLs
  - ▷ **Flaw #2:** fixed by adding $b \prec a'$ or $a \prec b$ to $\mathcal{O}$

- The **terminal states** in search are the complete plans (**solutions**) or the inconsistent ones (**dead ends**)

Status of POCL Planning

- POP/POCL dominated planning research for 10-15 years, until Graphplan
- Unlike other approaches, can work with action **schemas**
- In recent years lost favor to Graphplan/SAT/CSP/HSP
- Recent comeback combined with heuristics in RePOP and CPT
- Holds promise as **branching scheme for temporal planning**

Beyond Classical Planning: Transformations

AI Planning: Status

- The good news: **classical planning works!**
  - ▷ *Large problems solved very fast (non-optimally)*

- **Model simple but useful**
  - ▷ *Operators not primitive; can be policies themselves*
  - ▷ *Fast closed-loop replanning able to cope with uncertainty sometimes*

- Not so good; **limitations:**
  - ▷ *Does not model* **Uncertainty** *(no probabilities)*
  - ▷ *Does not deal with* **Incomplete Information** *(no sensing)*
  - ▷ *Does not accommodate* **Preferences** *(simple cost structure)*
  - ▷ . . .

Beyond Classical Planning: Two Strategies

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

  $+$: generality
  $-$: complexity

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

  $+$: efficiency
  $-$: generality

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

  ▷ compiling **soft goals** away (planning with preferences)
  ▷ compiling **uncertainty** away (conformant planning)
  ▷ deriving **finite state controllers**
  ▷ doing **plan recognition** (as opposed to plan generation)
  ▷ dealing with **temporally extended LTL goals**

Compilation of Soft Goals

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

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

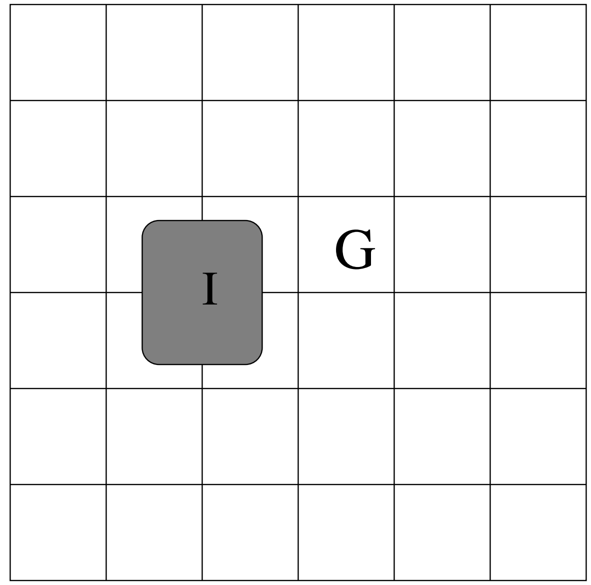
- Actions have **cost** $c(a)$, and soft goals **utility** $u(p)$
- Best plans achieve best **tradeoff** between **action costs** and **utilities**
- Model used in recent planning competitions; **net-benefit track** 2008 IPC
- Yet it turns that soft goals **do not** add expressive power, and can be **compiled away**

Compilation of Soft Goals (cont'd)

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

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

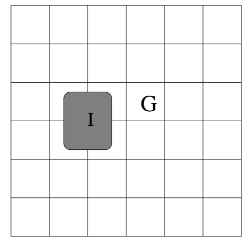
Incomplete Information: Conformant Planning



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

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

Conformant Planning: Belief State Formulation



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

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

– **effective representation** of belief states b

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

Recent alternative: translate into classical planning . . .

Basic Translation: Move to the 'Knowledge Level'

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Compiling Uncertainty Away: Properties

- General translation scheme $K_{T,M}(P)$ is always **sound**, and for suitable choice of the sets of **tags** and **merges**, it is **complete**.

- $K_{S0}(P)$ is **complete instance** of $K_{T,M}(P)$ obtained by setting T to the set of **possible initial states** of P

- $K_i(P)$ is a **polynomial instance** of $K_{T,M}(P)$ that is **complete** for problems with **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

Planning with Sensing: Models and Solutions

Problem: Starting in one of two leftmost cells, get to B; A & B **observable**

A				B

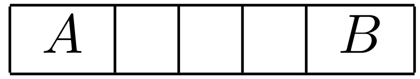
- **Contingent Planning**
  - ▷ *A* **contingent plan** *is a* **tree** *of possible executions, all leading to the goal*
  - ▷ *A contingent plan for the problem:* $\underline{R(ight), R, R, \text{ if } \neg B \text{ then } R}$

- **POMDP planning**
  - ▷ *A* **POMDP policy** *is mapping of belief states to actions, leading to goal*
  - ▷ *A POMDP policy for problem:* $\underline{\text{If } Bel \neq B, \text{ then } R}$ $\quad (2^5 - 1\ Bel$'s)

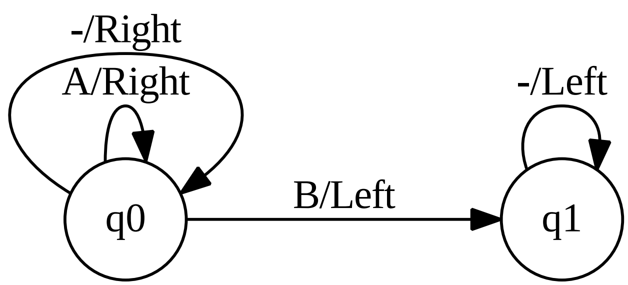
I'll focus on different **solution form**: **finite state controllers**

Finite State Controllers: Example 1

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



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



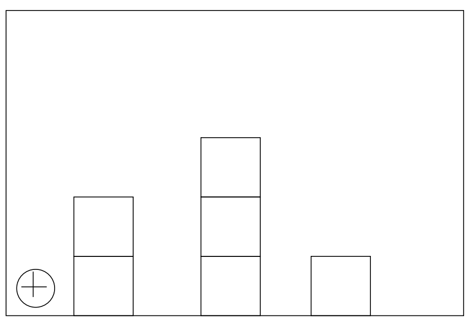
- FSC is **compact** and **general:** can add noise, vary distance, etc.

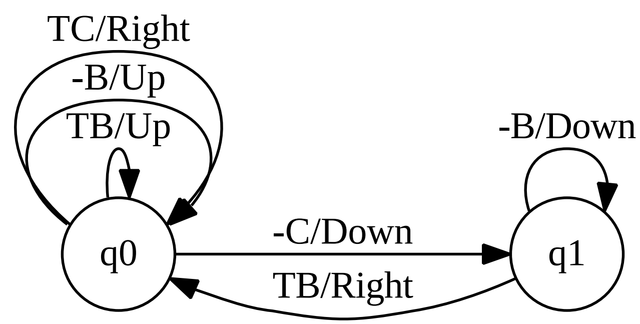
- Heavily **used in practice**, e.g. video-games and robotics, but **written by hand**

- **The Challenge:** How to get these controllers automatically

Finite State Controllers: Example 2

- **Problem** P: find **green block** using visual-marker (circle) that can move around one cell at a time
- **Observables:** Whether cell marked contains a green block (G), non-green block (B), or neither (C); and whether on table (T) or not (–)





- Controller on the right **solves** the problem, and not only that, it's **compact** and **general**: it applies to **any number of blocks** and **any configuration**!
- Controller obtained by running a **classical planner** over **transformed problem**

Deriving finite state controller from a conformant/classical plan

- **Finite state controller** $\mathcal{C}$ is a set of tuples $t = \langle q, o, a, q' \rangle$

  tuple $t = \langle q, o, a, q' \rangle$, *depicted* $q \stackrel{o/a}{\rightarrow} q'$, *tells to do action* a *when* o *is observed in controller state* q *and then to switch to* q'

- FSC $\mathcal{C}$ **solves** P if all state trajectories compatible with P and $\mathcal{C}$ reach the goal

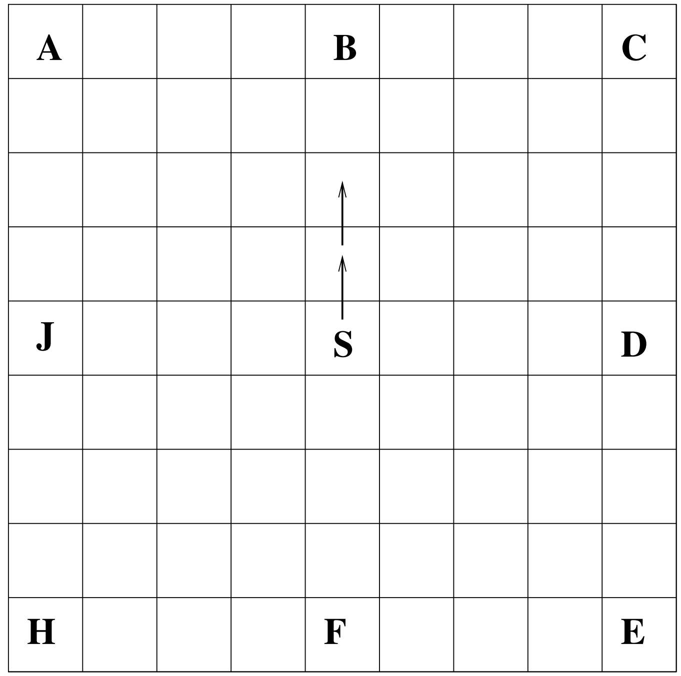
- **Transformation** maps **partially observable** P into **conformant** P_N for deriving controller with up to N states

- Conformant problem P_N contains **action** a_t for each possible **tuple** $t = \langle q, o, a, q' \rangle$

- Action a_t in P_N behaves like action a in P but **conditional** on q and o being true, and then setting q' true. Also, a_t **excludes** action $a_{t'}$ from plans if $t = \langle q, o, a, q' \rangle \neq t' = \langle q, o, a', q'' \rangle$

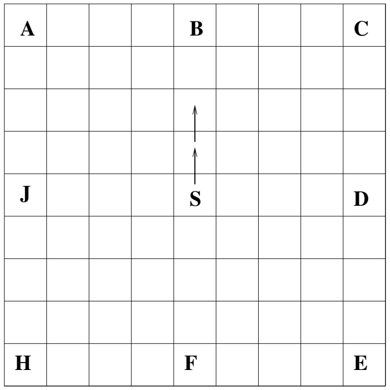
- The actions a_t in the **plan for** P_N encode a **controller that solves** P

Plan Recognition



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

Example (cont'd)



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

Posterior Probabilities from Plan Costs

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

$$\textbf{function}(c(G+\overline{O})\; - \;c(G+O))$$

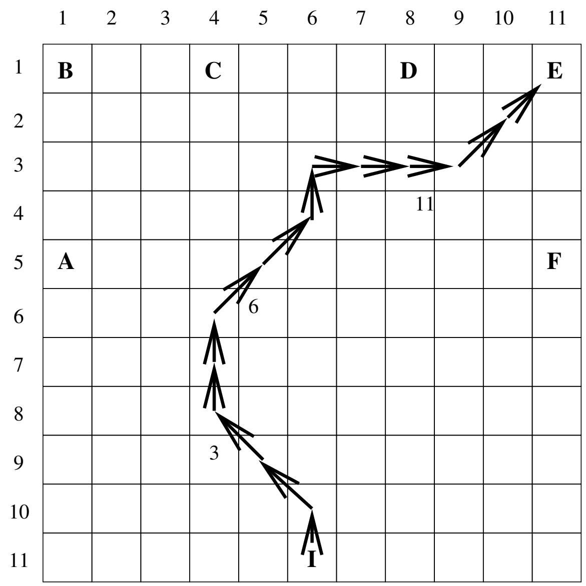
  - ▷ $c(G+O)$: cost of achieving G **while complying with** O
  - ▷ $c(G+\overline{O})$: cost of achieving G **while not complying with** O

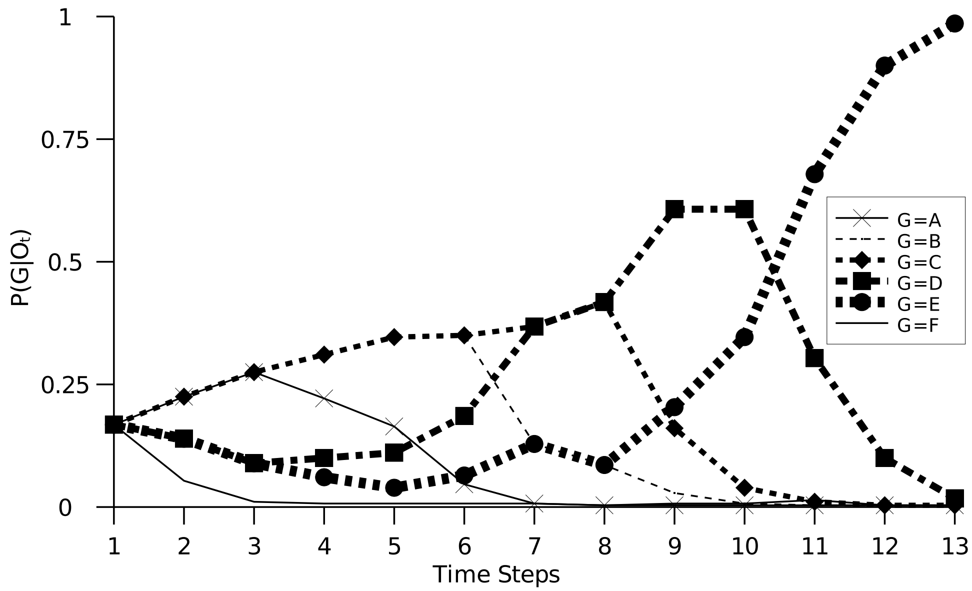
– Costs $c(G+O)$ and $c(G+\overline{O})$ computed by **classical planner** on **transformed problem** where **complying** and **not complying** with O translated into normal goals

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

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

Illustration: Noisy Walk





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

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

Extended Temporal LTL Goals

- Classical planning concerned with synthesis of **finite plans** to achieve **reachability goals**

- **Temporally extended goals** expressed in **linear temporal logic (LTL)** used to encode restrictions over whole **state trajectories** that may require **infinite plans**
  - ▷ *e.g., monitor room A and room B forever:* $\Box(\Diamond At(A) \wedge \Diamond At(B))$

- LTL extends propositional logic with **unary temporal operators** $\bigcirc$, $\Diamond$, and $\Box$, and **binary temporal operator** $\mathcal{U}$.
  - ▷ $\bigcirc\varphi$ says that φ holds at the **next** instant,
  - ▷ $\Diamond\varphi$ says that φ will **eventually** hold,
  - ▷ $\Box\varphi$ says that from current instant on φ will **always** hold, and
  - ▷ $\varphi\,\mathcal{U}\,\psi$ says that at some future instant ψ will hold and **until** then φ holds

- **Question:** How to plan to achieve/satisfy LTL goals (formulas)?

Settings for Planning with Temporal Extended LTL Goals

- 1. Classical planning for reachability goals with **finite plan** that satisfies subclass of LTL requirements
  - ▷ *most basic case; e.g., plans that don't visit states where $p \wedge q$ is true*
- 2. Planning with **deterministic actions** and known initial situation for achieving **arbitrary** LTL goals
  - ▷ *may require* **infinite 'lasso plans**
- 3. Planning with **non-deterministic actions**, **fully observable states** and **arbitrary** LTL goals
  - ▷ *this is a more interesting case; e.g., controller for 'artificial clerk' in store*
- 4. Planning with **non-deterministic actions**, **partially observable states** and **arbitrary** LTL goals
  - ▷ *this is the most general case; it subsumes* **LTL controller synthesis** which is 2-EXPTIME Complete.

Basic Case: Classical Planning with LTL constraints

- **Problem:** compute finite plan $\pi = a_0, \ldots, a_{n-1}$ for classical problem $P = \langle F, I, O, G \rangle$ s.t. resulting state sequence $s_0, \ldots, s_n$ satisfies LTL restriction φ.

- **Approach:** sequence π can be obtained as **plan** for a **transformed classical planning problem** P_φ obtained from P and LTL formula φ

- **Intuition:**

  ▷ The state trajectories that satisfy φ can be characterized as the inputs accepted by an automata A^φ

  ▷ The classical problem P_φ is the compact representation of the **product** of two automata: the automata implicitly encoded by P and the automata A^φ implicitly encoded by φ

- **Extensions:** Similar ideas have been used to address problems 2 and 3 in previous slide. Positive empirical results of planning approach in relation to symbolic methods used in formal verification and synthesis.

Summary: Transformations

- **Classical Planning** solved as **path-finding** in state state
  - ▷ Most used techniques are **heuristic search** and **SAT**
- **Beyond classical planning:** two approaches
  - ▷ **Top-down:** solvers for richer models like MDPs and POMDPs
  - ▷ **Bottom-up:** compile non-classical features away
- We have follow second approach with **transformations** to eliminate
  - ▷ **soft goals** when planning with preferences
  - ▷ **uncertainty** in conformant planning)
  - ▷ **sensing** for deriving finite-state controllers
  - ▷ **observations** for plan recognition
  - ▷ **extended temporal LTL goals**
- Other transformations used for dealing with **control knowledge**, **plan libraries**, **fault tolerant planning**, etc.

MDP & POMDP Planning

Models, Languages, and Solvers (Review)

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

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

- Many models, many solution forms: uncertainty, feedback, costs, . . .
- Models described in suitable **planning languages** (Strips, PDDL, PPDDL, . . .) where **states** represent interpretations over the language.

Planning with Markov Decision Processes: Goal MDPs

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

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

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

– **Optimal** solutions minimize **expected cost** from s_0 to goal

Discounted Reward Markov Decision Processes

Another common formulation of MDPs . . .

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

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

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

Expected Cost/Reward of Policy (MDPs)

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

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

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

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

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

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

Solving MDPs: Assumptions, Generality

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

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

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

Discounted MDPs easy to convert into **Goal MDPs**; no similar transformation known in the opposite direction

Basic Dynamic Programming Methods: Value Iteration (1)

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

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

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

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

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

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

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

Basic DP Methods: Value Iteration (2)

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

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

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

(Parallel) Value Iteration and Asynchronous Value Iteration

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

Method 2: Policy Iteration

- **Expected cost** of policy π from s to goal, $V^\pi(s)$, is weighted avg of **cost** of **state trajectories** $\tau : s_0, s_1, \ldots,$ times their **probability** given π
- **Trajectory cost** is $\sum_{i=0,\infty} cost(\pi(s_i), s_i)$ and **probability** $\prod_{i=0,\infty} P_{\pi(s_i)}(s_{i+1}|s_i)$
- Expected costs $V^\pi(s)$ can also be characterized as solution to **Bellman equation**

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

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

- This set of **linear equations** can be solved analytically, or by VI-like procedure
- **Optimal expected cost** $V^*(s)$ is $\min_\pi V^\pi(s)$ and **optimal policy** is the $\arg\min$
- For **discounted reward** MDPs, all similar but with $r(s,a)$ instead of $c(a,s)$, max instead of min, and sum discounted by γ

Policy Iteration (cont'd)

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

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

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

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

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

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

Dynamic Programming: The Curse of Dimensionality

- **VI** and **PI** are exhaustive methods that need value vectors V of size $|S|$

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

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

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

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

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

Question: Can we do better?

Dynamic Programming and Heuristic Search

- **Heuristic search** algorithms like A* and IDA* manage to solve **optimally** problems with more than 10^{20} states, like Rubik's Cube and the 15-puzzle

- For this, **admissible heuristics** (lower bounds) used to **focus/prune** search

- Can admissible heuristics be used for **focusing updates** in DP methods?

- Often states **reachable** with **optimal policy** from s_0 much smaller than S

- Then convergence to V^* **over all** s not needed for **optimality** from s_0

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

Learning Real Time A* (LRTA*) Revisited

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

- LRTA* can be seen as **asynchronous value iteration** algorithm for **deterministic** actions that takes advantage of theorem above (i.e. updates $=$ DP updates)
- **Convergence** of LRTA* to V implies residuals along π_V reachable states from s_0 are all zero
- Then 1) $V = V^*$ along such states, 2) $\pi_V = \pi^*$ from s_0, but 3) $V \neq V^*$ and $\pi_V \neq \pi^*$ over other states; yet this is irrelevant given s_0

Real Time Dynamic Programming (RTDP) for MDPs

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

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

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

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

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

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

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

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

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

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

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

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

Variations on RTDP : Reinforcement Learning

Q-learning is a **model-free** version of RTDP; Q-values initialized arbitrarily and **learned by experience**

1. **Apply** action $\mathbf{a}$ that minimizes $Q(\mathbf{a}, s)$ with probability $1 - \epsilon$, with probability ϵ, choose $\mathbf{a}$ randomly
2. **Observe** resulting state s' and collect cost c
3. **Update** $Q(\mathbf{a}, s)$ to

$$(1 - \alpha)\, Q(\mathbf{a}, s) \, + \, \alpha[c + \text{min}_a Q(a, s')]$$

4. **Exit** if s' is goal, else with $s := s'$ go to 1

- Q-learning converges asympotically to **optimal** Q-values, when all actions and states visited **infinitely often**
- Q-learning solves MDPs optimally without model parameters (probabilities, costs)

Variations on RTDP : Reinforcement Learning (2)

More familiar **Q-learning** algorithm formulated for **discounted reward MDPs:**

1. **Apply** action $\mathbf{a}$ that maximizes $Q(\mathbf{a}, s)$ with probability $1-\epsilon$, with probability ϵ, choose $\mathbf{a}$ randomly
2. **Observe** resulting state s' and collect **reward** r
3. **Update** $Q(\mathbf{a}, s)$ to

$$(1-\alpha)\, Q(\mathbf{a}, s) \,+\, \alpha[r + \gamma\, \text{max}_a Q(a, s')]$$

4. **Exit** if s' is goal, else with $s := s'$ go to 1

- Q-values initialized arbitrarily
- This version solves **discounted reward MDPs**

Model-based Reinforcement Learning (RMAX)

- Planning is **model-based**: it computes behavior from model
- Reinforcement learning (RL) is **model-free**: behavior from trial-and-error
- **Model-based RL** maps **learning** into a **planning** over **optimistic model**:
  - ▷ final **nirvana state** s_N that yields max reward R is added
  - ▷ **unknown** transitions $P_a(s'|s)$ replaced by transition $P_a(s_N|s) = 1$
  - ▷ **planning over optimistic model** leads agent to unknown parts s, a of **true model**
  - ▷ statistics on rewards and transitions from s and a make these parts **eventually known** (with enough confidence)
  - ▷ optimistic model becomes less optimistic and **more accurate**, incrementally, converging eventually to **true model**

Limitations of exact MDP planning and learning algorithms

- **inference** is at the level of **states**, not **variables**: can't scale too well
- **inference** in the form of **value updates**: learning is slow

Can we do better?

We'll look at **approximate algorithms** that exploit connections to **classical planning** in both **on-line** and **off-line** settings

On-line MDP Planning by Classical Replanning

- Make **deterministic relaxation** (D-Relax); mapping diff outcomes into diff actions

$$a : C \rightarrow E_1 \; p_1 \,|\, E_2 \; p_2 \,|\, \cdots \,|\, E_n \; p_n \quad \mapsto \quad a_i : C \rightarrow E_i \;, i = 1, \ldots, n$$

- Solve D-Relax from **current state** with **classical planner**
- Follow **classical plan** until **state observed** that is not **predicted** by D-Relax
- **Replan** from that state using D-Relax, and repeat until goal found

This is **simple and often effective** but **no formal guarantees** (FF-Replan)

Proper Policies for MDPs vs. Strong Cyclic Policies for FOND

- Policy π is **proper** from s_0 in an MDP iff it leads to **goal with probability** 1
  - ▷ *e.g., hit nail until it gets into the wall*

- **Non-deterministic relaxation** of MDP is a **Fully Observable Non-Deterministic (FOND)** problem (probabilities dropped)

$$a : C \rightarrow E_1\ p_1 \,|\, E_2\ p_2 \,|\, \cdots \,|\, E_n\ p_n \quad \mapsto \quad a : C \rightarrow E_1 \,|\, E_2 \,|\, \cdots \,|\, E_n$$

- **Theorem:** π is **proper** for MDP iff π is **strong cyclic** for its FOND relaxation

- π is **strong cyclic** for FOND iff it leads to the goal assuming that dynamics is **fair**, or more formally, iff for all s **reachable** from s_0 and π, the **goal** is **reachable** from s and π

Strong, Weak, and Strong Cyclic Policies for FOND Problems

- **Pesimistic** and **optimistic** values V^π_{max} and V^π_{min} can be obtained for $a = \pi(s)$ from $V^\pi_{max}(s) = V^\pi_{min}(s) = 0$ for goal states s, and Bellman equations:

$$V^\pi_{max}(s) = c(a,s) + MAX_{s' \in F(a,s)} V^\pi_{max}(s')$$
$$V^\pi_{min}(s) = c(a,s) + MIN_{s' \in F(a,s)} V^\pi_{min}(s')$$

- π is a **strong policy** iff $V^\pi_{max}(s_0) \neq \infty$

- π is a **weak policy** iff $V^\pi_{min}(s_0) \neq \infty$

- Optimal strong and weak policies can be **computed** from V^*_{max} and V^*_{min} obtained from Bellman eqs:

$$V_{max}(s) = \min_{a \in A(s)} [c(a,s) + MAX_{s' \in F(a,s)} V_{max}(s')]$$
$$V_{min}(s) = \min_{a \in A(s)} [c(a,s) + MIN_{s' \in F(a,s)} V_{min}(s')]$$

Computing Strong Cyclic Policies – Exhaustive Method

- **Thm:** π is **strong cyclic** iff $V^{\pi}_{min}(s) \neq \infty$ for all s **reachable** from s_0 and π

- Strong cyclic π can thus be **computed** by following loop:
  - ▷ **Compute** $V^*_{min}(s)$ for all s
  - ▷ **Prune:** s' and actions $a \in A(s)$ such that $s' \in F(a, s)$ if $V^*_{min}(s') = \infty$
  - ▷ **Repeat** 1 and 2 til no more pruning
  - ▷ **Terminate:** Policy greedy in resulting V^*_{min} is **strong cyclic** from s_0
  - ▷ **Caveat:** If s_0 is pruned, no strong cyclic policy from s_0 exists

Strong Cyclic Policies using Classical Planners

- **Idea:** $V^*{}_{min}(s) \neq \infty$ in FOND problem P amounts to existence of **classical plan** from s in **deterministic relaxation** of FOND. This suggests computing **strong cyclic policies** for FOND using **classical planners**

- Consider **state-plan (SP) pairs** $\langle S, \Sigma \rangle$ where S is subset of states S and members $\sigma(s)$ of Σ are plans mapping a state $s \in S$ into goal **in DET-relaxation**. Pair $\langle S, \Sigma \rangle$ is
  - ▷ **complete** if Σ contains one plan $\sigma(s)$ for each $s \in S$
  - ▷ **consistent** if all plans $\sigma(s)$ in Σ that pass through a state s' apply same action in s', denoted $\pi(s')$
  - ▷ **closed** if $s_0 \in S$ and all states s generated by plans in Σ are in S

- **Theorem:** Policy π defined by complete state-plan pairs that are consistent and closed is **strong cyclic** for FOND problem.

- **Corollary:** compute π incrementally with **classical planner**, starting with $\langle S_0 = \{s_0\}, \Sigma_0 = \{\} \rangle$, by making pair complete and closed, while keeping it consistent. 1) Procedure may have to **backtrack**; 2) Classical planner extended to avoid **nogood** actions and states

Relaxations of MDPs for Action Selection

- **FOND relaxation** for
  - ▷ computing **proper** policies as **strong cyclic** policies
- **DET relaxation** for
  - ▷ computing **strong cyclic policies** using classical planners
  - ▷ **goal-driven action selection** in MDPs by classical replanning

We move to new relaxation/model, **finite horizon MDPs/FOND/Game Trees**, for **on-line** action selection

Finite horizon relaxation of MDPs, FOND, Game Trees

- **states** s become pairs (s, d) where d is **number of steps to go**, $d \in [0, \ldots, H]$
- **initial state** becomes (s, H) where s is **current state** and H is **given horizon**
- **terminal states** (s, d) such that s is **goal state** or $d = 0$
- **value** of terminal states s, d, $V_T(s, d)$ **available** (as in Chess)
- **state transitions** as in original problem but decrement steps-to-go d
- **action costs** or **rewards** can be arbitrary (including zero in Game Trees)

Idea: Select action a in current state s such that a is **best** in s relative to **finite-horizon relaxation**. Solution to finite-horizon problem yields form of **lookahead**

Solving Finite-Horizon Problem Exhaustively

- **Optimal value function** $V^*(s,d)$ can be computed backward from **terminal states** where $V^*(s,d) = V_T(s,d)$

MDPs:	$V^*(s,d) = \min_{a \in A(s)}[c(a,s) + \sum_{s' \in S} P_a(s'\vert s) V^*(s', d-1)]$
FONDs:	$V^*(s,d) = \min_{a \in A(s)}[c(a,s) + MAX_{s' \in F(a,s)} V^*(s', d-1)]$
Game Trees:	$V^*(s,d) = \min_{a \in A(s)}[MAX_{s' \in F(a,s)} V^*(s', d-1)]$

- Procedure called **backward induction** (minimax search for GTs); implemented by **depth-first search** of **AND/OR graph** (**Game Tree**) associated with **finite-horizon MDP, FOND** or **Game**

- This is **single iteration** of **Value Iteration**; sufficient when MDP/FOND is **acyclic** and **updates** ordered back from leaves

- Still **exponential** in **horizon** H . . .

Heuristic and Monte Carlo Tree Search for Solving Finite-Horizon Problem

Acyclic AND/OR graph can be solved in other ways:

- **Heuristic Search**
  - ▷ **AO*** is generalization of A* for AND/OR graphs
- **Game Tree Search**
  - ▷ **Alpha-Beta pruning** improves minimax with **lower** and **upper bounds**
- **Monte Carlo Tree Search (MCTS)**
  - ▷ **UCT** (Upper Bound Confidence for Trees) generalizes algorithm UCB for **multi-arm bandits** to trees

Two Player Games: Game Tree Search

- Heuristic search and planning concerned with **action selection** in problems where **initial state is given** and **changes follow from actions of single agent**

- How to plan for a goal **in the presence of other agents?**

- It depends on **what the other agents want**
  - ▷ The other agents may be there to help **(cooperative agents)**
  - ▷ The other agents may be there to hinder **(adversarial agents)**
  - ▷ The other agents may be there for their own goals **(not necessarily cooperative or adversarial)**

- **Game Trees** provide model for **2-player, adversarial, sequential games**

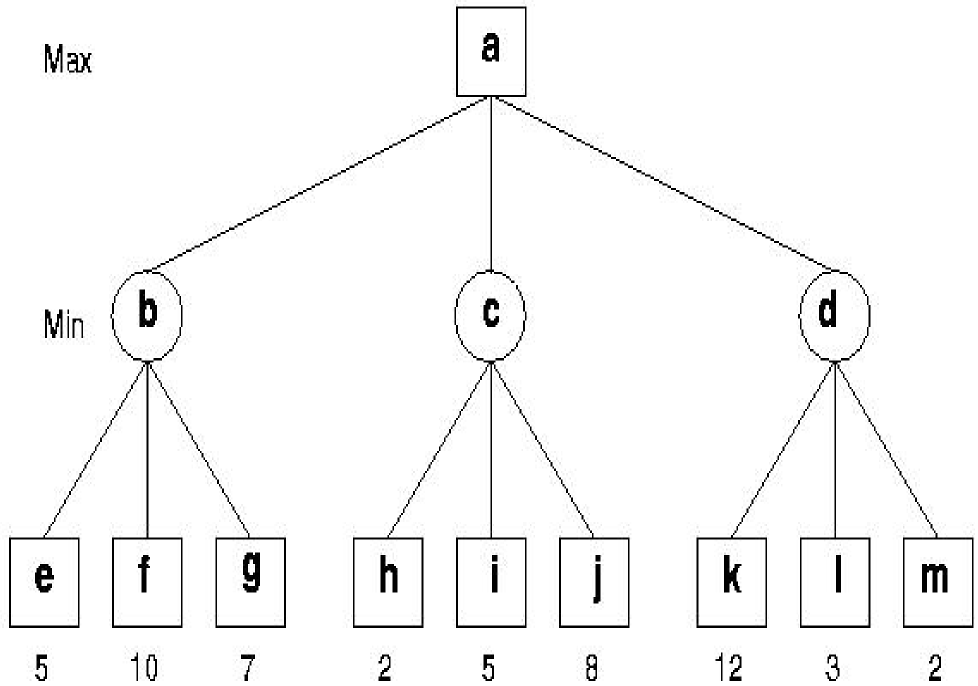
- Yet, most interactions in life are **not** adversarial. We will come back to them later.

Game Trees

Games trees suitable for adversarial (zero-sum) games like **tic-tac-toe**, **checkers**, **chess**, etc. They are made up of **three components:**

- **Three types of nodes**
  - ▷ **Max Nodes:** represent options for **Max-Player**
  - ▷ **Min Nodes:** represent options for the other player, called **Min**
  - ▷ **Terminal Nodes:** represent **final outcomes**, no options
- **Structure**
  - ▷ **Root node** is either MAX or MIN (we assume MAX, and values from MAX's perspective)
  - ▷ **Children** of MAX nodes are MIN nodes or Terminal nodes
  - ▷ **Children** of MIN nodes are MAX nodes or Terminal nodes
  - ▷ **Terminal** nodes, and only them, have no children
- **Evaluation function**
  - ▷ **Values** or **payoffs** associated with terminal nodes

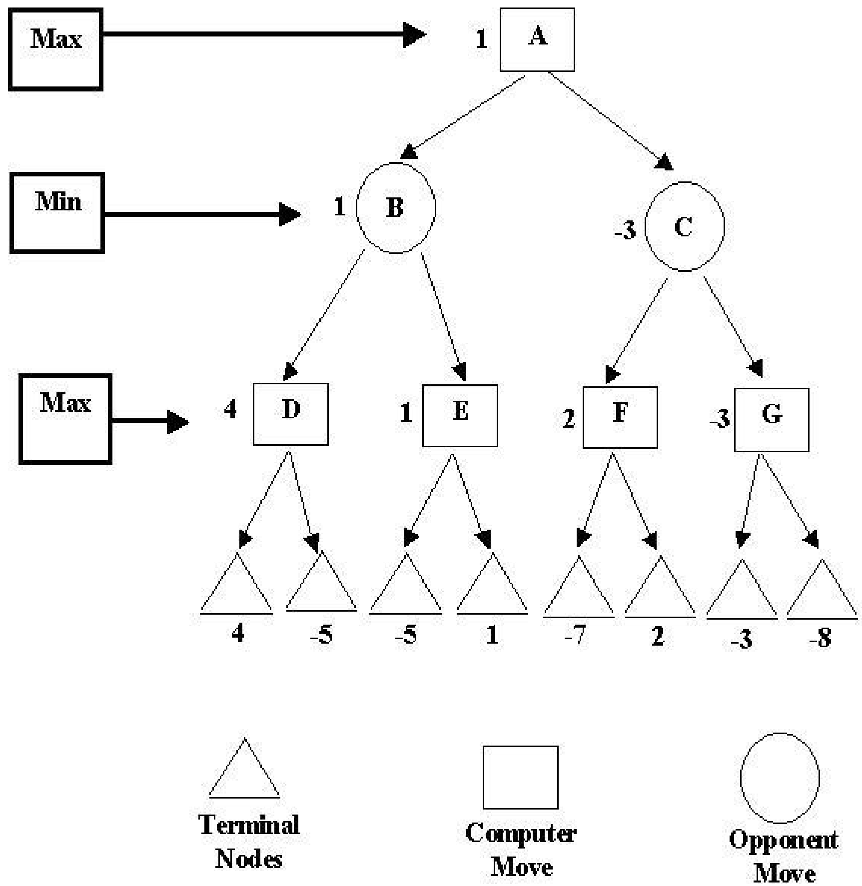
Game Tree – Example



What should MAX player do in this game?

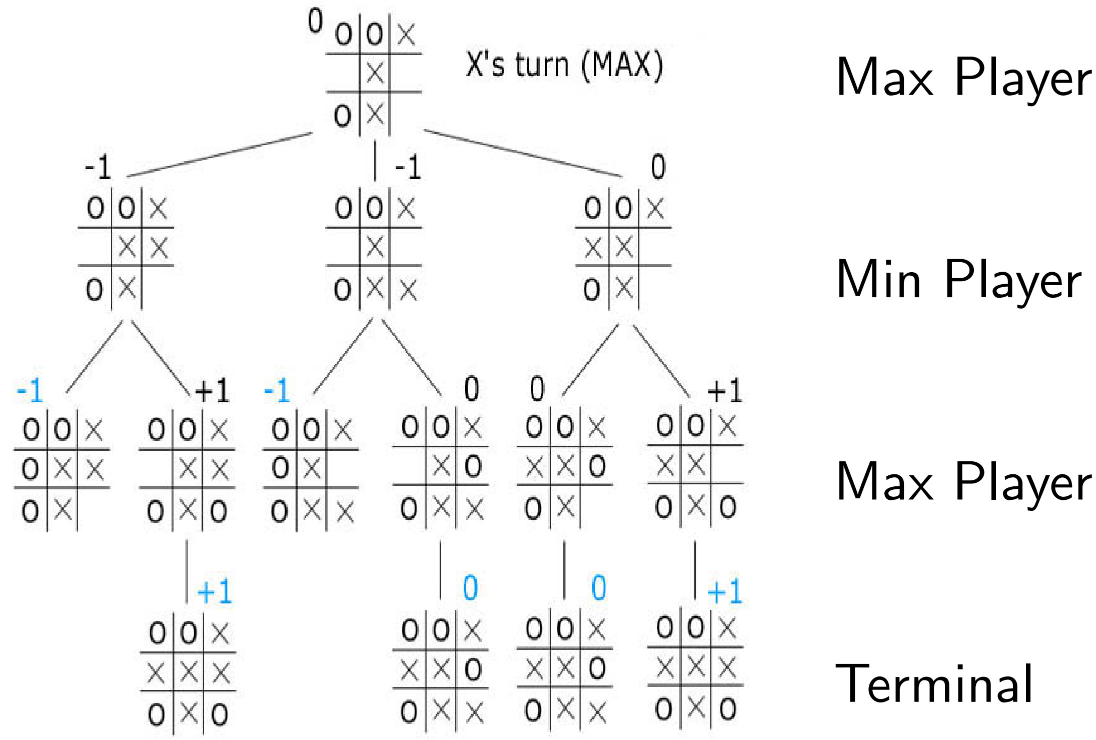
- MAX player has **three** options: left, middle, right
- What's the **Minimum payoff** that MAX player can ensure for each move?
- What should **MIN player** do if games reaches **b**, **c**, or **d**?

Game Trees – Another Example



- What should MAX player do **initially** in this game?
- What's the **minimum payoff** that he can ensure for each of the possible initial moves?
- What's the **minimum payoff** that he can ensure at the root of the game?

Game Trees – Fragment of Tic-Tac-Toe



- What should **MAX player** (crosses) do in given situation (root)?
- **Terminal nodes** represent final situations (wins, losses, ties for MAX)
- **Value** for win, loss, and tie set to $+1$, 0, -1.

Minimax (and Maximin) Algorithm for Game Trees

- Game tree evaluated **bottom up**:
  - ▷ Value of **Terminal Nodes** is **given value** of terminal
  - ▷ Value of **Min Node** is **Minimum** value of its childrenn
  - ▷ Value of **Max Node** is **Maximum** value of its children
- Value of **Root node** is value that **MAX** can guarantee
- **Best initial action** is the one that leads to child with that value

– This can be all computed by means of a **Depth-first Search**

– Algorithm called then **Maximin**, if root is MAX, or **Minimax** if MIN

– **Time complexity** is $O(b^d)$ where d is number of levels (depth) and b is branching factor (avg number of moves per node)

– **Space complexity** of these algorithms is $O(b \cdot d)$

Using and Improving Minimax over Huge Game Trees

- **Minimax** algorithms explore **complete game tree**
- This is possible in **Tic-tac-toe** but not in **Chess** or **Checkers**
- In those cases, a depth d is set, and nodes at such depth treated as **terminal**
- **Best first move** in resulting tree executed, and process is repeated after **opponent move**
- **Evaluation function** that sets values of such (non-final) **terminals** designed by hand or learned
- E.g., Value of chess board can reflect aspects such as **piece advantage**, **control of center**, etc.
- Quality of play depends on **depth** d used and **evaluation function** for terminals
- Good checkers and chess playing programs incoporate extensions for using **non-uniform depths** and searching tree **non-exhaustively**

Minimax with Alpha-Beta Pruning

- **Depth-first search** for evaluating game tree can be improved substantially in performance

- **Two observations**:

  ▷ When some of the children of a MAX node have been evaluated, there is a **lower bound** α on how much MAX can get, even if the other children have not been evaluated yet
  ▷ When some of the children of a MIN node have been evaluated, there is in turn an **upper bound** β on how much MAX can get, if the other children have not been evaluated yet

- **Two optimizations** that return same value and same moves:

  ▷ Skip rest of MIN children in depth-first search when $\beta \leq \alpha$ (β cut-off)
  ▷ Skip rest of MAX children in depth-first search when $\beta \leq \alpha$ (α cut-off)

- In practice, **Alpha-Beta pruning** makes $O(b^d)$ search in $O(b^{d/2})$ time, meaning **quality** given by depth d but time as if depth was $d/2$.

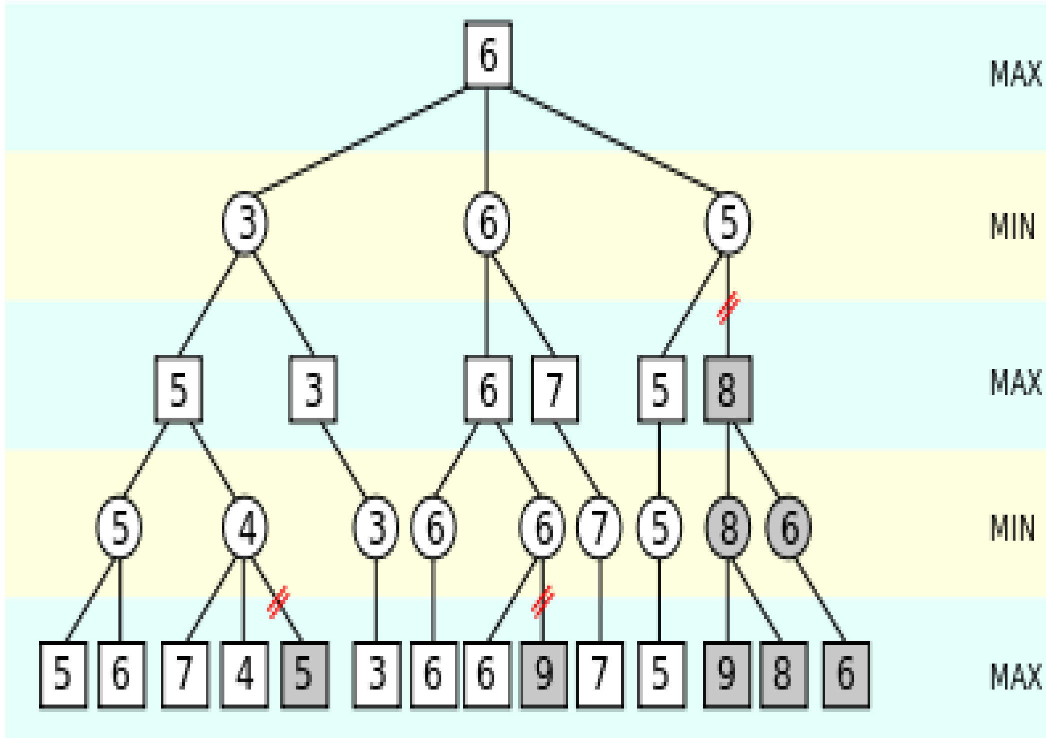
Pseudo-Code Maximin with Alpha-Beta Pruning

```
function alphabeta(node, depth, alpha, beta, maximizingPlayer)
  if depth = 0 or node is a terminal node
       return  value of node

  if maximizingPlayer
       for each child of node
         alpha := max(alpha, alphabeta(child, depth - 1, alpha, beta, FALSE))
         if beta <=  alpha
             break (* beta cut-off *)
       return alpha
  else
       for each child of node
         beta := min(beta, alphabeta(child, depth - 1, alpha, beta, TRUE))
         if beta <=  alpha
             break (* alpha cut-off *)
       return beta

(* Initial call *)
alphabeta(origin, depth, -infinity, +infinity, TRUE)
```

Example Alpha-Beta Pruning



- Game tree with alpha cut-offs shown
- Depth-first search assumed from left to right (leftmost child evaluated always first, etc)
- Pruned parts of the tree can be skipped without affecting value or best action at root

AO* for Acyclic AND/OR Graphs

- Acyclic AND/OR graph M to be solved is **implicit**
- AO* makes part of M explicit **incrementally**
- For this, AO* maintains **explicit graph** G that **initially** contains root node only
- G is expanded **incrementally** by selecting **tip node** in **best solution graph** of G that is **not terminal** in M, and expanding it in G
- The value of such tip nodes given by **heuristic function**, until they are expanded when their value is function of their **children**
- AO* **terminates** when **best solution graph** of G contains no such tips
- Upon termination, best solution graph of G is **optimal** if heuristic is admissible (lower bound of V^*)

AO* Code for Finite-Horizon MDPs

AO*: G is explicit graph, initially empty; h is heuristic function.

Initialization

- Insert node (s, H) in G where s is the initial state
- Initialize $V(s, H) := h(s, H)$
- Initialize **best partial graph** to G

Loop

- **Select** non-terminal tip (s, d) in **best partial graph.** If no such node, **Exit**.
- **Expand** node (s, d): for each $a \in A(s)$, add node (a, s, d) as child of (s, d), and for each s' with $P_a(s'|s) > 0$, add node $(s', d-1)$ as child of (a, s, d). Initialize values $V(s', d-1)$ for new nodes $(s', d-1)$ to $V_T(s', d-1)$ if terminal, else to $h(s', d-1)$
- **Update** ancestor AND and OR nodes of (s, d) in G, bottom-up as:

$$Q(a, s, d) := c(a, s) + \textstyle\sum_{s'} P_a(s'|s)V(s', d-1),$$

$$V(s, d) := \min_{a \in A(s)} Q(a, s, d).$$

- **Revise best partial graph** by updating best actions in ancestor OR-nodes (s, d) to any action a such that $V(s, d) = Q(a, s, d)$, maintaining marked action if still best.

AO* Limitations and Variations

- AO* is **not** optimal if graph is **not** acyclic or heuristic **not** admissible
- For general MDPs and FONDs, **backward induction** step in AO* needs to be replaced by **full value iteration**. This is expensive (LAO*)
- A* is **anytime optimal** even without an admissible heuristic, if A* **not stopped** after first solution found
- Same trick does **not** work for AO*
- For AO* being **anytime optimal**, AO* needs to select nodes that are **not** part of **best solution graph** (exploitation/exploration tradeoff; AOT)
- Common source of **non-admissible heuristics** from use of **base policies** π_0 to initialize values by means of **rollouts:** simulation of the base policy

UCT for Finite-Horizon Problems and Relaxations

- UCT is instance of Monte-Carlo Tree Search (MCTS) algorithms where best action selected from $Q(a, s)$ values computed by
  - ▷ running **simulations** (actual model not required)
  - ▷ **averaging** resulting values
  - ▷ using these values for **action selection in simulations** (adaptive Monte-Carlo)
- In practice, UCT not used with **heuristic functions** but **base policies**
- UCT made big splash in **game of Go** and many other games and tasks
- Like AO*, UCT builds **explicit graph** G incrementally starting with root node only, but in a different way . . .

UCT from the perspective of AO*: Four differences

- **Selection** of tip node to expand in G follows **simulation from root**: first state generated that is not in G, is added
- **Value** of new nodes obtained from **rollout**: simulation of **base policy** from node
- **Propagation** of values up the graph G following **Monte Carlo** updates as opposed to **Bellman updates**
- **Choice of action** in simulations inside graph G not greedy in $Q(a,s)$ but greedy in $Q(a,s)$ extended with **exploration bonus** as:

$$Q(a,s) + C\sqrt{2\log N(s,d)/N(a,s,d)}$$

  where C is exploration constant, and $N(s,d)$ and $N(a,s,d)$ track number of simulations that pass through (s,d) and (s,d,a)
- **Exploration bonus** ensures that all actions tried in all states infinitely often at rates that optimize regret in multi-arm setting.

UCT Code for Finite-Horizon Discounted Reward MDPs

UCT(s, d): G is explicit graph, initially empty; π is base policy; C is exploration constant.

- If $d = 0$ or s is terminal, Return $V_T(s, d)$
- If node (s, d) is not in explicit graph G, then
  - ▷ Add node (s, d) to explicit graph G
  - ▷ Initialize $N(s, d) := 0$ and $N(a, s, d) := 0$ for all $a \in A(s)$
  - ▷ Initialize $Q(a, s, d) := 0$ for all $a \in A(s)$
  - ▷ Obtain sampled accumulated discounted reward $r(\pi, s, d)$ by simulating base policy π for d steps starting at s
  - ▷ Return $r(\pi, s, d)$
- If node (s, d) is in explicit graph G,
  - ▷ Select action $a = \text{argmax}_{a \in A(s)}[Q(a, s, d) + C\sqrt{2 \log N(s, d)/N(a, s, d)}]$
  - ▷ Sample state s' with probability $P_a(s'|s)$
  - ▷ Let $nv = r(s, a) + \gamma\text{UCT}(s', d - 1)$
  - ▷ Increment $N(s, d)$ and $N(a, s, d)$
  - ▷ Set $Q(a, s, d) := Q(a, s, d) + [nv - Q(a, s, d)]/N(a, s, d)$
  - ▷ Return nv

Summary MDPs and Related Fully Observable Models

- **DP methods** like **VI** and **PI** are optimal but exhaustive

- **Heuristic search** methods like RTDP, LAO*, and **Find-and-Revise** can be optimal without being exhaustive

- **Relaxations:**

  ▷ **FOND relax** for computing **proper policies**
  ▷ **DET relax** for computing **strong cyclic policies** and **greedy** actions
  ▷ **Finite-Horizon relaxation** for **on-line** action selection

- Solutions to **finite-horizon problems**:

  ▷ Exhaustive **backward induction**, **minimax search**, ...
  ▷ Heuristic search **AO*** and variations
  ▷ Monte-Carlo methods: **UCT**, **RTDP**, ..

Partially Observable MDPs: Goal 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 $a \in A(s)$
- initial **belief state** b_0
- set of **observable target** states S_G
- action costs $c(a, s) > 0$
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$

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

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

– **Optimal** policies minimize **expected** cost to go from b_0 to target bel state.

Discounted Reward POMDPs

A common alternative formulation of POMDPs:

- states $s \in S$
- actions $A(s) \subseteq A$
- transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$
- initial **belief state** b_0
- **sensor model** given by probabilities $P_a(o|s)$, $o \in Obs$
- **rewards** $r(a, s)$ positive or negative
- **discount factor** $0 < \gamma < 1$; **there is no goal**

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

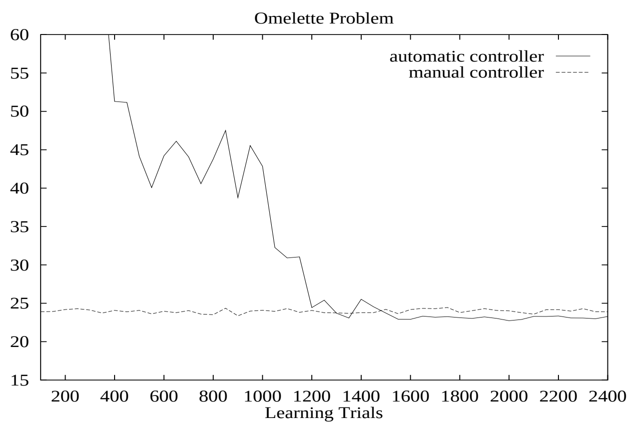
– **Optimal** solutions max **expected discounted accumulated reward** from b_0

Example: Omelette

- Representation in GPT (incomplete):

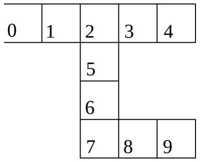
Action:	$\mathbf{grab} - \mathbf{egg}()$
Precond:	$\neg holding$
Effects:	$holding := \mathbf{true}$
	$good? := (\mathbf{true}\ 0.5\ ;\ \mathbf{false}\ 0.5)$
Action:	**clean**(bowl:BOWL)
Precond:	$\neg holding$
Effects:	$ngood(bowl) := 0\ \ ,\ nbad(bowl) := 0$
Action:	$\mathbf{inspect}(bowl : BOWL)$
Effect:	$\mathbf{obs}(nbad(bowl) > 0)$

- Performance of resulting controller (2000 trials in 192 sec)

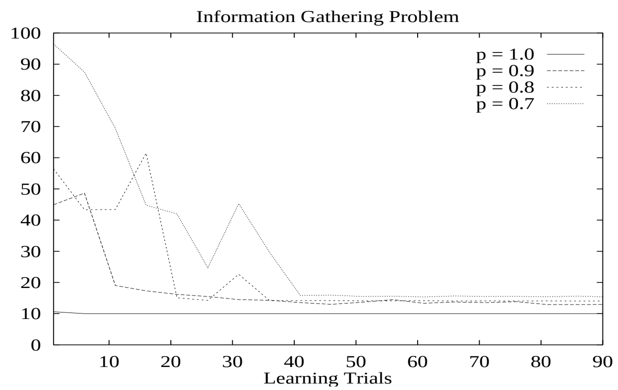


Example: Hell or Paradise; Info Gathering

- initial position is 6
- $goal$ and $penalty$ at either 0 or 4; which one not known
- noisy map at position 9

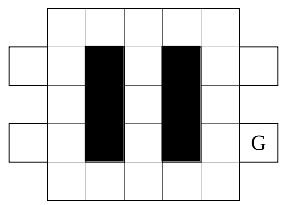


Action:	$\mathbf{go-up}()$; same for down,left,right
Precond:	$\textsc{free}(\textsc{up}(pos))$
Effects:	$pos := \textsc{up}(pos)$
Action:	$*$
Effects:	$pos = pos9 \ \rightarrow \ \mathbf{obs}(ptr)$
	$pos = goal \ \rightarrow \ \mathbf{obs}(goal)$
Costs:	$pos = penalty \ \rightarrow \ 50.0$
Ramif:	$\mathbf{true} \ \rightarrow \ ptr = (goal \ p \ ; \ penalty \ 1-p)$
Init:	$pos = pos6$; $goal = pos0 \ \vee \ goal = pos4$
	$penalty = pos0 \ \vee \ penalty = pos4$; $goal \neq penalty$
Goal:	$pos = goal$



Examples: Robot Navigation as a POMDP

- **states:** $[x, y; \theta]$
- **actions** $rotate$ $+90$ and -90, $move$
- **costs:** uniform except when hitting walls
- **transitions:** e.g, $P_{move}([2, 3; 90] \mid [2, 2; 90]) = .7$, if $[2, 3]$ is empty, . . .



- **initial** b_0: e.g,, uniform over set of states
- **goal** G: cell marked G
- **observations:** presence or absence of wall with probs that depend on position of robot, walls, etc

Equivalence of (PO)MDPs

- Let the **sign** of a POMDP be **positive** if cost-based and **negative** if reward-based

- Let $V^{\pi}_{M}(b)$ be expected cost (reward) from b in positive (negative) POMDP M

- Define **equivalence** of any two POMDPs as follows; assuming goal states are absorbing, cost-free, and observable:

Definition 1. *POMDPs R and M* **equivalent** *if have same set of non-goal states, and there are constants α and β s.t. for every π and non-target bel b,*

$$V^{\pi}_{R}(b) = \alpha V^{\pi}_{M}(b) + \beta$$

with $\alpha > 0$ if R and M have same sign, and $\alpha < 0$ otherwise.

Intuition: If R and M are equivalent, they have same optimal policies and same 'preferences' over policies

Equivalence Preserving Transformations

- A transformation that maps a POMDP M into M' is **equivalence-preserving** if M and M' are equivalent.

- Three **equivalence-preserving transformation** among POMDP's

  1. $R \mapsto R + C$: addition of C ($+$ or $-$) to all rewards/costs
  2. $R \mapsto kR$: multiplication by $k \neq 0$ ($+$ or $-$) of rewards/costs
  3. $R \mapsto \overline{R}$: elimination of discount factor by adding goal state t s.t.

$$P_a(t|s) = 1 - \gamma \ , \ P_a(s'|s) = \gamma P_a^R(s'|s) \ ; \ O_a(t|t) = 1 \ , \ O_a(s|t) = 0$$

Theorem 2. *Let R be a* **discounted reward-based** POMDP, *and C a constant that bounds all rewards in R from above; i.e. $C > \max_{a,s} r(a,s)$. Then, $M = \overline{-R + C}$ is a* **goal** POMDP *equivalent to R.*

POMDPs are MDPs over Belief Space

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

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

- The probability of observing o then is:

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

- . . . and the new belief is

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

Computational Methods for POMDPs

- **Exact Methods**
  - ▷ **Value Iteration** over Piece-wise linear functions
- **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
- **Logical Methods:** beliefs represented by sets of states
  - ▷ **Compilations** and **relaxations** for action selection
  - ▷ **Belief tracking:** for determining truth of action preconditions and goals

Value Iteration for POMDPs (1)

- Value V^π of a **policy** π given by solution to Bellman equation with $V^\pi(b) = 0$ when b is a **goal belief**:

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

- **Optimal value function** V^* given by solution to Bellman optimality equations with $V(b) = 0$ for goal beliefs b

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

- In both cases, $c(a,b) = \sum_{s \in S} c(a,s) b(s)$

- The computational problem for VI is that there is **infinite** and **dense** space of beliefs to update

VI over Piecewise Linear Value Functions

- **Sondik's observation** (1973): if VI starts from a **piecewise linear and concave (pwlc) function** V_0, then all V_k resulting from updates remain *pwlc*.

- A *pwlc* function f on the continuous belief space over S is a finite combination of linear functions:

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

- Resulting **implementation** of VI: start with set Γ_0 of vectors encoding V_0 and compute **new set of vectors** Γ_{i+1} encoding V_{i+1} that corresponds to **full update** of V_i

$$\Gamma_{i+1} = Function(\Gamma_i, \text{ POMDP pars})$$

- **Problem:** Size $|\Gamma_{i+1}|$ grows **exponentially** with $|A| \, |\Gamma_i|^{|O|}$

Computing New Vectors Γ_{k+1} from Old Γ_k (abbreviated)

Use 'choice functions' $\nu(o)$ that pick vector $\nu(o)$ from Γ_k; $|\Gamma_k|^O$ such functions. Expression $\nu(o)(s)$ stands for $\alpha(s)$:

$$
\begin{aligned}
V_{k+1}(b) &= \min_{a\in A}\left[c(a,b) + \gamma \textstyle\sum_{o\in O} b_a(o) V_k(b_a^o)\right] \\
&= \min_{a\in A}\left[c(a,b) + \gamma \textstyle\sum_{o\in O} b_a(o) \left[\min_{\alpha\in\Gamma_k} \sum_{s\in S} b_a^o(s)\alpha(s)\right]\right] \\
&= \min_{a\in A}\left[c(a,b) + \min_{\nu\in\mathcal{V}_k} \gamma \textstyle\sum_{o\in O} b_a(o) \sum_{s\in S} b_a^o(s)\nu(o)(s)\right] \\
&= \min_{a\in A}\left[c(a,b) + \min_{\nu\in\mathcal{V}_k} \gamma \textstyle\sum_{s\in S} \sum_{o\in O} \nu(o)(s) b_a(o) b_a^o(s)\right] \\
&= \min_{a\in A}\left[c(a,b) + \min_{\nu\in\mathcal{V}_k} \gamma \textstyle\sum_{s,o} \nu(o)(s) \sum_{s'} b(s') P_a(s|s') P_a(o|s)\right] \\
&= \min_{a\in A} \min_{\nu\in\mathcal{V}_k} \left[c(a,b) + \gamma \textstyle\sum_{s,o} \nu(o)(s) \sum_{s'} b(s') P_a(s|s') P_a(o|s)\right] \\
&= \min_{a\in A,\nu\in\mathcal{V}_k} \textstyle\sum_{s'} b(s') \left[c(a,s') + \gamma \sum_{s,o} \nu(o)(s) P_a(s|s') P_a(o|s)\right] \\
&= \min_{\alpha\in\Gamma_{k+1}} \textstyle\sum_{s\in S} b(s)\alpha(s)
\end{aligned}
$$

where $\Gamma_{k+1} \stackrel{\text{def}}{=} \{\alpha_{a,\nu} \mid a \in A, \nu \in \mathcal{V}_k\}$ is the collection of vectors $\alpha_{a,\nu}$

$$
\alpha_{a,\nu}(s) \quad \stackrel{\text{def}}{=} \quad c(a,s) + \gamma \textstyle\sum_{s',o} \nu(o)(s') P_a(s'|s) P_a(o|s') \,.
$$

Approximate Methods: Pointed-based Value Iteration

- Point-based algorithms compute smaller set of vector $\Gamma_{k+1}(F)$ from Γ_k, one for each belief b in a **given set** F
- Vector $backup(V, b) \in \Gamma_{k+1}(F)$ is the one in Γ_{k+1} that represents value of b:

$$backup(V, b) = \operatorname*{argmin}_{\alpha \in \Gamma_{k+1}} \sum_{s \in S} b(s)\alpha(s)$$

- Key point is that computation of $backup(V, b)$ does not require Γ_{k+1}, and is **polynomial** in $|A|$, $|O|$ and $|S|$
- Point-based POMDP algorithms differ in choice of points F, initial value vectors, and termination
- Some methods make use of known initial belief state b_0 and lower/upper bounds

Computation of $backup(V, b)$ (sketch)

$$
\begin{aligned}
backup(V,b) &= \textstyle\sum_{o\in O} b_a(o) V(b_a^o) \\
&= \textstyle\sum_{o\in O} b_a(o) \big[\min_{\alpha\in\Gamma} \sum_{s\in S} b_a^o(s)\alpha(s)\big] \\
&= \textstyle\sum_{o\in O} \big[\min_{\alpha\in\Gamma} \sum_{s,s',s''\in S} P_a(o|s'')P_a(s''|s')b(s')\alpha(s)\big] \\
&= \textstyle\sum_{o\in O} \big[\min_{\alpha\in\Gamma} \sum_{s'\in S} g_{a,o}^{\alpha}(s')b(s')\big]
\end{aligned}
$$

where $g_{a,o}^{\alpha}(s) = \sum_{s'\in S} \alpha(s')P_a(o|s')P_a(s'|s)$.

Using $\min_{\alpha\in\Gamma} g_{a,o}^{\alpha} \cdot b = [\mathrm{argmin}\{g_{a,o}^{\alpha}\cdot b | g_{a,o}^{\alpha}, \alpha \in \Gamma\}] \cdot b$

$$
\begin{aligned}
\textstyle\sum_{o\in O} b_a(o)V(b_a^o) &= \textstyle\sum_{o\in O} \big[\min_{\alpha\in\Gamma} g_{a,o}^{\alpha}\cdot b\big] \\
&= \textstyle\sum_{o\in O} \big\{ \big[\mathrm{argmin}\{g_{a,o}^{\alpha}\cdot b \,|\, g_{a,o}^{\alpha},\ \alpha\in\Gamma\}\big]\cdot b \big\} \\
&= \big[\textstyle\sum_{o\in O} \mathrm{argmin}\{g_{a,o}^{\alpha}\cdot b \,|\, g_{a,o}^{\alpha},\ \alpha\in\Gamma\}\big]\cdot b
\end{aligned}
$$

If $g_{a,b}(s) \stackrel{\text{def}}{=} c(a,s) + \gamma \sum_{o\in O} g_{a,o}^{b}(s)$ where $g_{a,o}^{b} = \mathrm{argmin}_{g_{a,o}^{\alpha}}(g_{a,o}^{\alpha}\cdot b)$,

$$
\begin{aligned}
backup(V,b) &= \min_{a\in A}\ c(a,b) + \gamma\big[\textstyle\sum_{o\in O} \mathrm{argmin}\{g_{a,o}^{\alpha}\cdot b \,|\, g_{a,o}^{\alpha},\ \alpha\in\Gamma\}\big]\cdot b \\
&= \min_{a\in A}\ \big[g_{a,b}\cdot b\big] = \mathrm{argmin}_{g_{a,b}}\ \big[g_{a,b}\cdot b\big]
\end{aligned}
$$

Approximate POMDP Methods: RTDP-Bel

Since POMDPs are MDPs over belief space, RTDP algorithm for POMDPs becomes

RTDP-BEL

% *Initial value function* V *given by heuristic* h
% *Changes to* V *stored in a 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: $Q(a, b) := c(a, b) + \sum_{o \in O} b_a(o) V(b_a^o)$
  Select best action $\mathbf{a} := \mathrm{argmin}_{a \in A(b)} Q(a, b)$
  Update value $V(b) := Q(\mathbf{a}, b)$
  Sample next state s' with probability $P_{\mathbf{a}}(s'|s)$ and set $s := s'$
  Sample observation o with probability $P_{\mathbf{a}}(o|s)$
  Update current belief $b := b_a^o$
end while

RTDP-Bel **discretizes** beliefs b for writing to and reading from hash table

UCT for POMDPs: PO-UCT

- PO-UCT is **on-line** POMDP algorithm based on UCT
- Like UCT, PO-UCT uses **simulations** for choosing action to do next in **finite-horizon relaxation**
- Nodes in the tree (explicit graph G) constructed by PO-UCT, however, are not **belief states** but **histories**
- The **histories** stand for the possible sequences of **actions** and **observations** $a_1, o_1, a_2, o_2, \ldots$
- Still, **approximation** b' of current belief b needed to construct the simulations
- **Approximation of belief** b associated with history h obtained from the **simulations** compatible with h
- This is a type of **particle filter** belief tracking scheme; more about this below ...

$\text{Search}(h)$ **Code for PO-UCT**

repeat

  Sample s according to b_0 if $h = \langle\rangle$ or $B(h)$ otherwise

  $\text{Simulate}(s, h, 0)$

until time is up

return $\text{argmin}_a\, V(\langle ha \rangle)$

$\text{Simulate}(s, h, depth)$

if $\gamma^{depth} < \epsilon$ **then return** 0

if h does not appear in tree T **then**

  for all action $a \in A$ **do**

    Insert $\langle ha \rangle$ in tree as $T(\langle ha \rangle) := \langle 0, 0, \emptyset \rangle$

  end for

  return $\text{Rollout}(s, h, depth)$

end if

$a^* := \text{argmin}_a\, V(\langle ha \rangle) - C\sqrt{\log N(h)/N(\langle ha \rangle)}$

Sample (s', o, c) using simulator with state s and action a^*

$Cost := c + \gamma \cdot \text{Simulate}(s', \langle hao \rangle, 1 + depth)$

$B(h) := B(h) \cup \{s\}$

Increment $N(h)$ and $N(\langle ha \rangle)$

$V(\langle ha \rangle) := V(\langle ha \rangle) + [Cost - V(\langle ha \rangle)]/N(\langle ha \rangle)$

return $Cost$

$\text{Rollout}(s, h, depth)$

if $\gamma^{depth} < \epsilon$ **then return** 0

Let $a := \pi(h)$

Sample (s', o, c) using simulator with state s and action a

return $c + \gamma \cdot \text{Rollout}(s', \langle hao \rangle, 1 + depth)$

Logical Approaches for Planning with Sensing

- **Uncertainty** in dynamics and sensing represented by **sets of states**

- **Belief space** is thus **finite**; methods that apply to **fully observable non-deterministic** models (FOND), apply thus to **partially observable** ones (POND)

- This includes **dynamic programming**, **heuristic search**, and **on-line** methods, with **states** replaced by **belief states**

- For **deterministic partially observable models**, **transformations** have been developed for converting them into:
  - ▷ **fully observable non-deterministic (FOND) models**: exponential in problem **width** parameter
  - ▷ **classical (deterministic) problems**: doubly exponential in number of variables (used for heuristic action selection)

- Transformations build on transformations for mapping **deterministic conformant problems** into **classical** ones

Focus: On-line Planning for Planning with Sensing

- Two problems in **on-line** planning
  - ▷ **action selection:** what action to do next
  - ▷ **belief tracking:** needed to determine **applicable actions** and **if goal achieved**

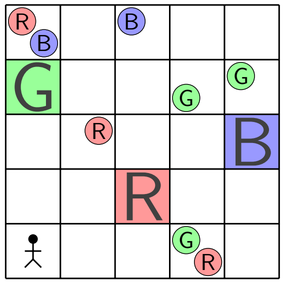
- Both problems **intractable** in worst case

- We will look for solutions, including **approximate** solutions to each one of them

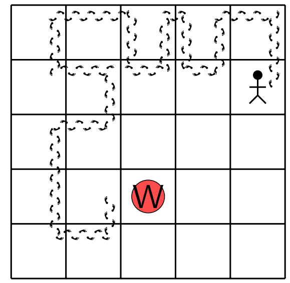
Action Selection for On-line Planning: Main Approaches

- **Heuristics** $h(b)$ for estimating cost in belief space
  - ▷ **Reachability** h**:** from $h(s)$ that estimate cost in state space, approx. $h(b)$ in various ways (e.g., $h(b) = \max_{s \in b} h(s)$, $h(b) = \sum_{s \in b} h(s)$, etc. **Problem:** no too informed, don't point to need to **get info**
  - ▷ **Cardinality** h**:** In translations, if $K(X = x)$ is **goal** (know what $X = x$ is true), then $K(X \neq x')$, where x' is another value of X, is an **implicit goal**. Can be used like **landmark heuristics**. Also, cardinality measure $|b|$ can be informative too.
- **Transformations**: **deterministic** problems of planning with sensing P can be mapped into **equivalent** classical planning problem P'
  - ▷ **Action sequences** that solve P' encode the **action trees** that solve P
  - ▷ While translation is **exponential** in number of initial states of P, it can be used to **suggest actions**; e.g., **sample** a few initial states from P and discard others
- **Relaxations: deterministic** problems of planning with sensing P can be mapped into (non-equivalent) classical planning problem P' by letting agent control **sensing outcomes**
  - ▷ This is a form of planning under **optimism**; in robotics, known as **free-space** assumptions
  - ▷ Related to FF-replan approach to MDPs where **optimism** is about **action outcomes**

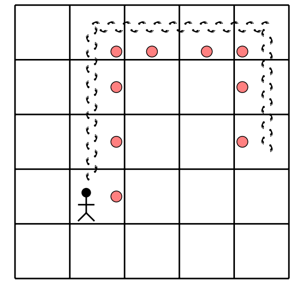
Examples (Bonet & G., IJCAI-2011)



colored-balls



kill-wumpus



trail

- **freespace**: agent needs to reach a final position by moving through unknown terrain. Each cell is blocked or clear. As agent moves, it senses status of adjacent cells.
- **doors**: agent moves in grid to reach final position, crossing doors whose positions can be sensed.
- **wumpus**: the agent moves in a grid to reach a final position that contains a number of deadly *wumpuses* that must be avoided.
- **kill-wumpus**: a variation of the above in which the agent must locate and kill the *single* wumpus in the grid.
- **colored-balls**: the agent navigates a grid to pick up and deliver balls of different colors to destinations that depend on the color of the ball. Initially, the positions and colors are unknown.
- **trail**: the agent must follow a trail of stones. The positions and quantity of stones are unknown and the agent cannot get off trail but observes the presence of stones in the nearby cells.

Second Problem in On-Line Planning: Belief Tracking

- **Exact, Explicit Flat Methods**
  - ▷ Computing next belief b_a^o from belief b, action a, and observation o is **exponential** in number of states, in both **probabilistic** and **logical** settings
- **Exact, Lazy Approaches**
  - ▷ Beliefs b not strictly required; what **is** required is to know whether **preconditions** and **goals** hold in b
  - ▷ In **logical setting**, this problem can be cast as **SAT** problem in theory obtained from initial belief, actions done, and obs gathered
  - ▷ In **probabilistic setting**, problem can be cast similarly a **Dynamic Bayesian Network** inference problem
  - ▷ Both approaches still **exponential** in worst case, but can be sufficiently **practical**
- **Approximations:** to be considered next
  - ▷ **Particle filtering:** when uncertainty in dynamics and sensing represented by probabilities
  - ▷ **Structured methods:** when uncertainty in dynamics and sensing represented by sets

Probabilistic Belief Tracking with Particles: Basic Approach

- Computing next belief b_a^o is exponential in $|S|$

$$
\begin{aligned}
b_a^o(s) &= P_a(o|s)b_a(s)/b_a(o) \\
b_a(s) &= \textstyle\sum_{s' \in S} P_a(s|s')b(s') \\
b_a(o) &= \textstyle\sum_{s \in S} P_a(o|s)b_a(s)
\end{aligned}
$$

- **Particle filtering** algorithms approximate b by (multi)set of **unweighted samples**
  - ▷ probability of $X = x$ approximated by **ratio** of **samples** in b where $X = x$ is true
- Approx. belief B_{k+1} obtained from (multi)set of samples B_k, action a, and obs o in two steps:
  - ▷ Sample s_{k+1} from S with probability $P_a(s_{k+1}|s_k)$ for each s_k in S_k
  - ▷ (Re)Sample new set of samples by sampling each s_{k+1} with weight $P(o|s_{s+1})$
- Potential problem is that **particles may die out** if many probabilities are zero

Structural Belief Tracking: Exploiting Relevance

- In worst case, belief tracking is **exponential** in number of variables
- Yet, often it's possible to do much better by **exploiting structure** of given problem
- Earlier translation that maps **deterministic** conformant planning into **classical planning** is exponential in **width** parameter
- This implies that **belief tracking** over such problems is also exponential in problem **width**
- We now deal with **non-deterministic problems with sensing** where similar **bound** established
- We don't do translations, just **belief tracking** ...

Model for Non-Deterministic Contingent Planning

Contingent model $\mathcal{S} = \langle S, S_0, S_G, A, F, O \rangle$ given by

- finite **state space** S
- non-empty subset of **initial states** $S_0 \subseteq S$
- non-empty subset of **goal states** $S_G \subseteq S$
- **actions** A where $A(s) \subseteq A$ are the actions applicable at state s
- **non-deterministic** transitions $F(s, a) \subseteq S$ for $s \in S, a \in A(s)$
- **non-determinisitc** sensor model $O(s', a) \subseteq O$ for $s' \in S, a \in A$

Language

Model expressed in **compact form** as tuple $P = \langle V, A, I, G, V', W \rangle$:

- V is set of **multi-valued variables**, each X has finite domain D_X
- A is set of actions; each action $a \in A$ has precondition $Pre(a)$ and conditional **non-deterministic** effects $C \rightarrow E^1 | \cdots | E^n$
- Sets of V-literals I and G defining the initial and goal states
- V' is set of observable variables (not necessarily disjoint from V). Observations o are **valuations** over V'
- **Sensing model** is formula $W_a(\ell)$ for each $a \in A$ and observable literal ℓ that is true in states that follow a where ℓ may be observed

A literal is an atom of the form '$X = x$' or '$X \neq x$'

Example: Wumpus

$$
\begin{aligned}
\mathit{rotate\text{-}left}: &\quad heading = 0 \rightarrow heading := 1 \\
&\quad heading = 1 \rightarrow heading := 2 \\
&\quad \dots \\
\mathit{rotate\text{-}right}: &\quad \dots \\
\mathit{move\text{-}forward}: &\quad heading = 0 \,\wedge\, pos = (x, y) \rightarrow pos := (x, y+1) \\
&\quad \dots \\
\mathit{grab\text{-}gold}: &\quad \mathit{gold\text{-}pos} = (x, y) \,\wedge\, pos = (x, y) \rightarrow \mathit{gold\text{-}pos} := \text{hand} \\
W_a(stench_{x,y} = true) = &\quad wump_{x-1,y} \,\vee\, wump_{x,y+1} \,\vee\, wump_{x,y-1} \,\vee\, wump_{x+1,y} \\
W_a(breeze_{x,y} = true) = &\quad pit_{x-1,y} \,\vee\, pit_{x,y+1} \,\vee\, pit_{x,y-1} \,\vee\, pit_{x+1,y} \\
W_a(glitter_{x,y} = true) = &\quad \big[\mathit{gold\text{-}pos} = (x, y) \,\wedge\, pos = (x, y)\big] \\
W_a(dead_{x,y} = true) = &\quad \big[pos = (x, y) \,\wedge\, (pit_{x,y} \vee wump_{x,y})\big]
\end{aligned}
$$

Key Ideas: Belief Tracking for Planning

- Don't need to keep track of **global beliefs** b; beliefs b_X about **precondition** and **goal variables** X suffice
- Beliefs b_X obtained by applying **plain belief tracking** to smaller **projected subproblems** P_X
- Subproblem P_X only involves state variables that are **relevant** to X
- Resulting algorithm, **Factored Belief Tracking**, is sound and complete for planning, and exponential in **width of** P:
  - ▷ maximum number of state variables that are all relevant to a given precondition or goal variable X
- Furthermore, variant of this idea can be used to yield **practical belief tracking** that is **sound**, **efficient**, and **powerful**, but not **complete**

Decompositions for Belief Tracking

- A decomposition of problem P is pair $D = \langle T, B \rangle$ where
  - ▷ T is subset of **target** variables, and
  - ▷ $B(X)$ for X in T is a subset of state variables
- Decomposition $D = \langle T, B \rangle$ decomposes P in subproblems P_X:
  - ▷ one subproblem P_X for each variable X in T
  - ▷ subproblem P_X involves only the state variables in $B(X)$
- **Factored Decomposition:**
  - ▷ $F = \langle T_F, B_F \rangle$ where T_F are state variables appearing in preconditions or goals, and $B_F(X)$ are all variables that are **relevant** to X
- **Causal Decomposition:**
  - ▷ $C = \langle T_C, B_C \rangle$ where T_C are variables in preconditions or goals, or **observables**, and $B_C(X)$ are all variables **causally relevant** to X

Complete Tracking over Causal Decomposition

- Belief tracking over causal decomposition is **incomplete** because two beliefs b_X and b_Y associated with target vars X and Y may interact and are not independent

- Algorithm can be made **complete** by enforcing **consistency** of beliefs:

$$b_X := \Pi_{B_{C(X)}} \bowtie \{(b_Y)_a^o : Y \in T_C \text{ and relevant to } X\}$$

- Resulting algorithm is **complete** for **causally decomposable problems**, **space exponential** in causal width, but **time exponential** in width

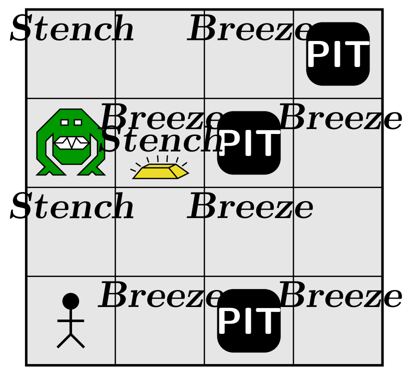
- **Beam tracking** replaces **global** by **local consistency** until **fix point**:

$$b_X := \Pi_{B_{C(X)}}(\, b_X^{i+1} \bowtie b_Y^{i+1})$$

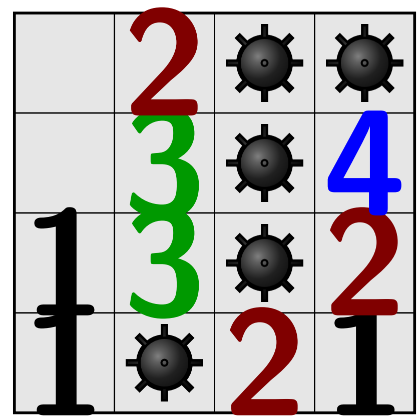
- Beam tracking is **time and space exponential** in **causal width**; sound but not complete

- See our IJCAI-2013 for experiments in Wumpus, Minesweeper, Battleship

Example: Wumpus and Minesweeper



Wumpus



Minesweeper

Two issues in Multiagent Planning: Introduction

- What's **optimal** or **best agent behavior** in presence of other actors?
  - ▷ **Game Theory**
- How to deal with **beliefs about the beliefs** of the other agents
  - ▷ **(Multiagent) Epistemic Logics**

Reference: Multiagent systems: Algorithmic, game-theoretic, and logical foundations, Yoav Shoham, Kevin Leyton-Brown, Cambridge University Press, 2008.

Essentials of Game Theory

- We have looked at **2-player, 0-sum** games like Chess, Checkers, etc
- In such games, what's good/best for one player, it's bad/worst for the other
- Best strategies maximize reward in "worst case" (adversarial); **maximim**
- Best action obtained then by single iteration of value iteration (**backward induction**), **alpha-beta pruning**, etc
- Yet, not all **games** or **interactions** are **0-sum** . . .

A more general “game” in extended (tree) form

- The game:
  - ▷ A plays first: A can choose (1,1) and **terminate**, **or** pass “ball” to B
  - ▷ B can then choose (2,2) and **terminate**, **or** pass “ball” back to A
  - ▷ A can finally choose (4,1) or (3,3), in both cases **terminating** the game

- This game can be depicted by **binary tree** . . .

- If game terminates with (n, m), player A gets reward n, and B gets m.

- In 0-sum games, $n + m = 0$; in constant-sum (equivalent), $n + m = constant$

Question: What should player A do?

Games in Normal Form

- **Game:** Agents, Actions, Payoffs (Utilities)
- **Example:** Agents "Row" and "Column"; Actions C and D; Entry n, m means n for Row, m for Column

	Action C	Action D
Action C	10,10	6,4
Action D	7,5	5,7

How should agents play?. Key notions:

- **Dominance:** C is better for Row **no matter** what Column does
- **Best-response:** D is **best response** for Column if Row plays D
- **Pareto optimal outcome:** if no entry makes **both** players better off
- **Strategies: Pure** strategy selects action; **mixed** strategy prob distribution
- **Equilibrium:** Two strategies in (Nash) eq. if each one **best-response** to other

"Solution concept" usually given by **Nash Equilibrium**; but NE is **wrong**; sometimes too weak, sometimes too strong.

Problems of Nash Equilibrium: Examples

Hi-Low game: **two equilibria** (C,C) and (D,D)

	Action C	Action D
Action C	10,10	0,0
Action D	0,0	5,5

Prisoner's Dilemma: deeper problems

	Action C	Action D
Action C	10,10	0,20
Action D	20,0	5,5

- **Only equilibrium** is (D,D) which results into only **non-opt pareto** 5,5
- Actually, problem is **deeper**: D dominates C for each player
- Yet, lots of people plays C (C understood as Cooperation; D as Defection)
- Prisoner's dilemma is very pervasive; **it's everywhere.**
- **What lessons to draw from this dilemma?**
- At least that **selfish behavior** can lead to **poor collective outcomes**

Lessons from and variations of Prisoner's Dilemma

Ultimatum is a sequential game: 100e to be split. First "offers" second X. If second accepts; he gets X and first gets $100 - X$. What much should First "offer"? What does Nash Eq. predict? **What people do?**

Homo vs. Homo-Economicus: Fortunately, people don't behave according to "economic/Nash rationality"

Evolution to the rescue?

- We are born **social beings**
- The invention of the **individual** is very recent according to our evolutionary history
- **Cooperation** is key but needs defense from **free-riders** that unravel benefits of cooperation
- How to defend from cheaters? **Punishments** of various sorts (like ostracism)
- Evolution can favor this form of **strong reciprocity** if competition among **groups** (us vs. them)
- Strategies like **tit-for-tat** performed well in **Iterated Prisioner's Dilemma competitions** (Axelrod)

Evolutionary Game Theory

- Assumes **initial population** given by numbers $n_i(0)$ of players of different strategies σ_i

- Uses **(replicator) dynamic model** where population $n_i(t+1)$ of $\sigma - i$ players at time $t+1$ is a function of how well they do on average at time t relative to the other players, given the population of players at time t
  - ▷ how do the populations **evolve** in time?
  - ▷ which populations are **stable** to **mutants** in population?

Example:

- Cake-splitting; demand-X players, $X = 10, 20, \ldots, 100$, percent demanded by cake.
- Pick two elements randomly from population: X and Y, iff $X + Y \leq 100$ then they get X and Y, else zero.
- Assume initial population: $1/3$ of 40's, 50's, and 70's resp.
- Population of demand-X_i at $t+1$: $E_t(X_i)/\sum_i E_t(X_i)$

Reference: Evolution and the Social Contract, Brian Skyrms, Cambridge University Press, 2006

Beliefs in Multiagent Systems: Example

- n-children $n > 1$
- k of these children have mud in their forehead
- they can see whether others are muddy but not themselves
- Father comes and announces: "(at least) one of you is muddy"
- "If you know you are muddy, say it, else say don't know"
- **Result:** After expressing ignorance (saying "don't know") $k - 1$ times, *the children that are muddy will know that they are muddy*
- Many puzzles like this; what's sort of reasoning is involved?

Reasoning about Multiagent (Nested) Beliefs: Language

- In **propositional logic**, formulas like $(\neg p \vee (\neg q \wedge p))$ made up of symbols like p and q, the the propositional connectives "$\neg$, $\wedge$, $\vee$, ...

- A formula A is **valid** iff it is true in every interpretation w; i.e., $\models_w A$ where
  - ▷ $\models_w p$ if p is true in w for symbols p
  - ▷ $\models_w (\neg A)$ if $\not\models_w A$
  - ▷ $\models_w (A \vee B)$ if $\models_w A$ or $\models_w B$
  - ▷ ...

- In **multiagent epistemic logics**,
  - ▷ symbols like p, q, ... are formulas,
  - ▷ $K_i A$ is a formula if A is a formula and i is an agent (i **knows** A is true)
  - ▷ propositional combination of formulas are formulas

Example: If m_i represents child i is muddy; formula $K_1 m_1 \vee K_1 \neg m_1$ represents that child 1 knows whether he is muddy or not.

Reasoning about Multiagent (Nested) Beliefs: Semantics (1)

- In muddy children, each child can see whether each other children is muddy but whether he himself is muddy
- If the **actual world** w is $m_1, \neg m_2, \neg m_3$, then agent 1 will **not know** initially whether the actual world is w or w' that is like w but with m_1 false
- After father announces $m_1 \vee m_2 \vee m_3$, however, assuming $n = 3$, **if actual world is** w, then **agent** 1 **will know it**
- As a result, initially $\not\models_w K_1 m_1$, but after announcement $\models_w K_1 m_1$

Reasoning about Multiagent (Nested) Beliefs: Semantics (2)

- Formally, a **Kripke structure** is a tuple $\langle W, R, V \rangle$, where:
  - ▷ W is the set of worlds w
  - ▷ $R_i(w)$: is the set of world deemed possible by **agent** i in world w
  - ▷ $V(w)$ is the truth valuation associated with world w.

- Formula A is **true** in world w of **structure** $\mathcal{K} = \langle W, R, V \rangle$, $\mathcal{K}, w \models A$, iff
  - ▷ if A is true in $V(w)$ when A is a propositional symbol
  - ▷ $\mathcal{K}, w \not\models B$ when A is $\neg B$
  - ▷ $\mathcal{K}, w \models B$ or $\mathcal{K}, w \models C$ when A is $B \vee C$
  - ▷ $\mathcal{K}, w \models B$ and $\mathcal{K}, w \models C$ when A is $B \wedge C$
  - ▷ $\mathcal{K}, w' \models B$ for all $w' \in R_i(w)$ when A is $K_i B$

- Formula A is **valid** in **structure** $\mathcal{K}$, $\mathcal{K} \models A$, if $\mathcal{K}, w \models A$ for all w in W

- Formula A is **valid**, $\models A$, if $\mathcal{K} \models A$ for $\mathcal{K}$

Example: Muddy Children

. . .

Wrapping Up: The Map

- **Intro to AI** and Automated Problem Solving
- **Classical Planning** as Heuristic Search and SAT
- **Beyond Classical Planning:** Transformations
  - ▷ *Soft goals, Conformant Planning, Finite State Controllers, Plan Recognition, Extended temporal LTL goals, . . .*
- **Planning with Uncertainty:** Markov Decision Processes (MDPs)
- **Planning with Incomplete Information:** Partially Observable MDPs (POMDPs)
- **Planning with Uncertainty and Incomplete Info:** Logical Models

– **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.

– **Slides:** `http://www.dc.uba.ar/materias/planning/slides.pdf`, `http://www.dtic.upf.edu/~hgeffner/bsas-2013-slides.pdf`

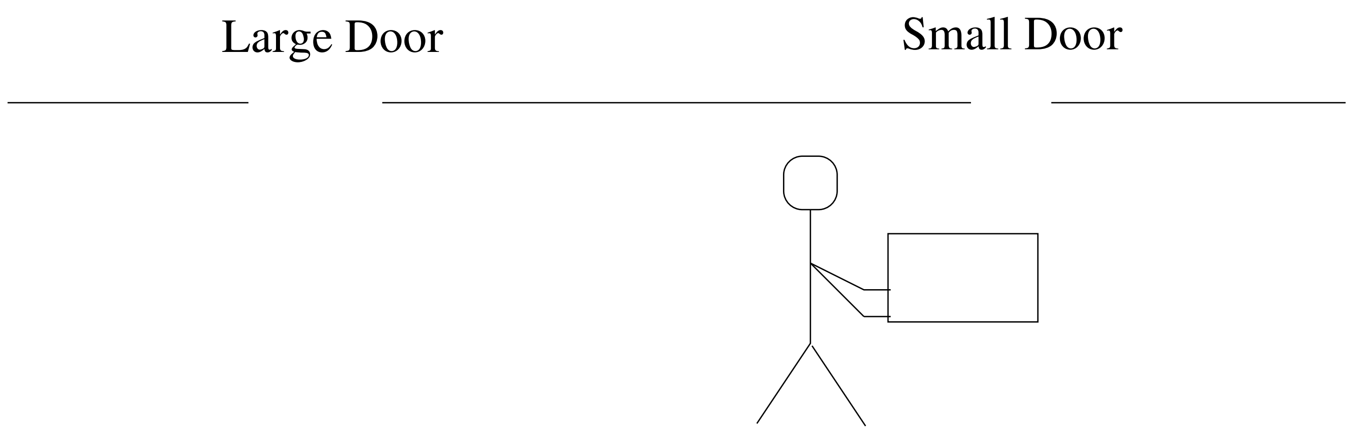
Summary

- 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:** a solid methodology for **autonomous agent design**

Some Challenges

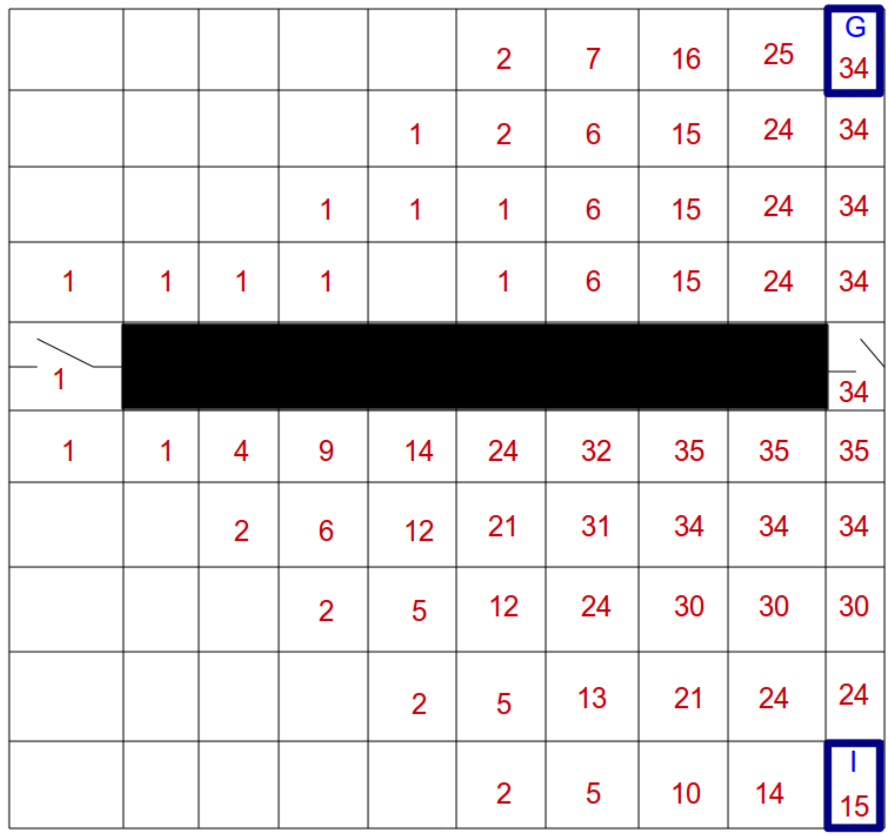
- **Classical Planning**
  - ▷ states & heuristics $h(s)$ not **black boxes**; how to exploit **structure** further?
  - ▷ **on-line planners** to compete with state-of-the-art **classical planners**
- **Probabilistic MDP & POMDP Planning**
  - ▷ inference can't be at level of **states** or **belief states** but at level of **variables**
- **Multi-agent 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



- 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!

Problem is not only computational: Multi-agent Planning

- Single-agent planning is computationally hard but target behavior is clearly defined
- What about **planning in the presence of other agents that plan**?
- Number of subtleties arise, **even when possible plans involve one action**
- E.g., Prisioners' Dilemma where **utility matrix** for 2 players is

	Action A	Action B
Action A	10,10	0,20
Action B	20,0	5,5

How should each of the players act? Why is a dilemma? Social agents?