## Agents

* *Agent*: perceives environment through sensors and acts upon them through actuators
* PEAP
* *Percepts*: observations of environment (made by sensors)
  + *Percept Sequence*: entire history of everything agent has perceived
* *Environment*: where agent exists
* *Action*: affects the environment
* *Performance Measure*: desired environment state(s)
* *Rational* agent: Maximises Performance Measure based on percept sequence and built-in knowledge
* *Agent Function*: maps percepts to actions

## Environment Properties (ODESD)

* Fully/Partially Observable
* can sensors capture complete state of environment at every point in time
* Deterministic/Stochastic
* does current state uniquely determine the next
* Episodic/Sequential
* do short-term actions have long-term consequences
* Static/Dynamic
* does environment change when agent does nothing
* Discrete/Continuous
* finite/infinite number of possible actions
* Single/Multi Agent
* are there other agents

**Agent Architectures**

| Simple Reflex Agent | Model-based Reflex Agent |
| --- | --- |
| only chooses action based on current percept, ignores all preceding information | maintains internal state of environment   * good for partially observable |
| Goal-Based Agent | Utility-Based Agent |
| makes decisions in order to achieve set of predefined goals, in addition to maintaining internal state | compares desirability of different environment states via utility function |

## Single-State Problems

* *Single-State*: completely observable; certain about our current state
* *Problem*:

1. Initial State
2. Actions/Operators/Successor Function
3. Goal Test
4. Path Cost (additive)

* *Solution:* sequence of actions leading from initial state to goal state
* *State Space*: all possible states in domain; must be abstracted

## Search Algorithms/Strategy

* offline, simulated exploration of state space by generating successors of already-explored states
* *Expand Function*: picks order of node expansion

Evaluation

* *Completeness*: guaranteed to find (any) solution?
* *Time Complexity*
* *Space Complexity*
* *Optimality*: find optimal solution?

|  | * : maximum branching factor * : depth of least cost solution * : maximum depth |
| --- | --- |

* *State*: representation of physical configuration of environment state
* *Node*: data structure constituting part of search tree (parent, child, depth, path cost)

## Uninformed Search Algorithms

* *Uninformed*: have access only to problem definition

| Breadth-First Search | Uniform-Cost Search |
| --- | --- |
| expand shallowest unexpanded node | expand least-cost unexpanded node |
| Depth-First Search | Depth-Limited Search |
| expand deepest unexpanded node | DFS with depth limit |
| Iterative Deepening | Bidirectional Search |
| DLS with increasing depth limits  combination of BFS and DFS | search (usually BST) simultaneously forwards from start point and backwards from goal |

## 

## Informed Search Algorithms

* *Informed*: have access to a heuristic function that estimates cost of solution from node
* *Heuristic Function*, h(n): decreases branching factor
* relax problem definition, store precomputed solution costs for subproblems, learn from experience with problem classification

| Best-First Search | Greedy Best-First Search |
| --- | --- |
| selects node according to evaluation function | expand node with minimal h(n) |
| A\* Search | Recursive Best-First Search (RBFS) |
| * f(n) = cost to reach node + h(n) * *Admissible Heuristic:* never overestimates estimated cost > true | uses limited amounts of memory; can solve problems A\* cannot solve because it runs out of memory; recursive stack |

| Criterion | Best-First | Greedy Best-First | A\* | RBFS |
| --- | --- | --- | --- | --- |
| Complete? | No | No | Yes | No |
| Time |  |  |  |  |
| Space |  |  |  |  |
| Optimal? | No | No | Yes | No |

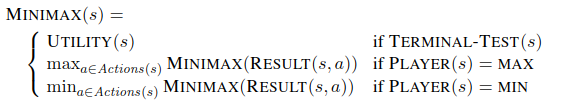
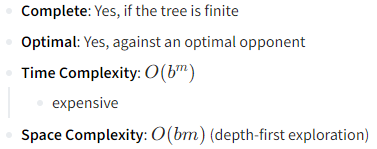
* *Hill Climbing*:
* finds for local minimum/maximum

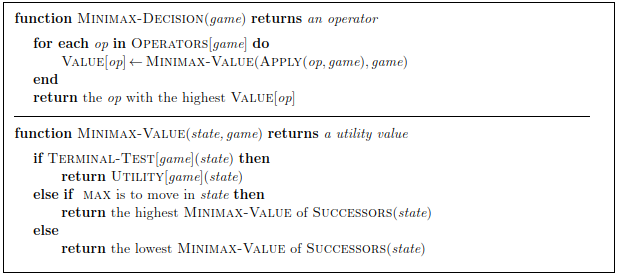
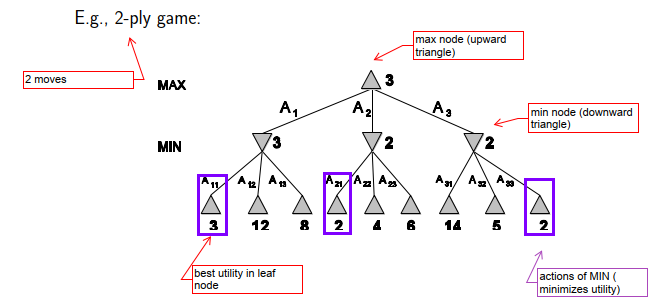
# Adversarial Search

* *Problem*:

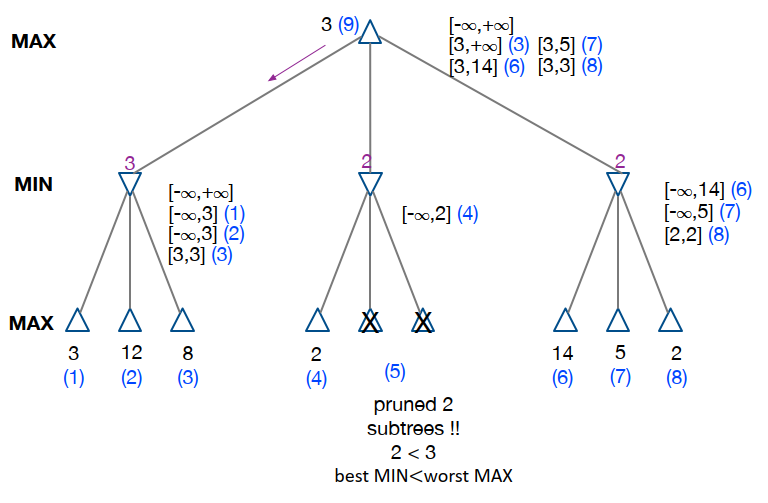
1. Initial State
2. Actions
3. Terminal Test (win/lose/draw)
4. Utility Function

## Minimax



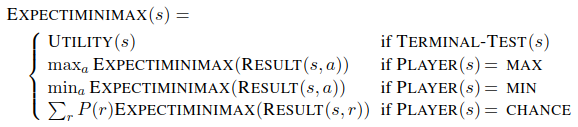


## Alpha-Beta Pruning

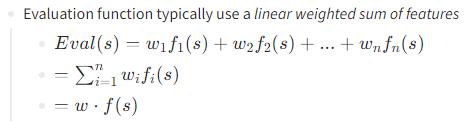


## ExpectiMinimax

handles chance nodes

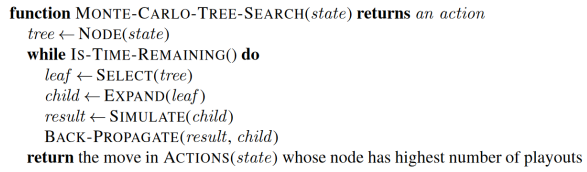


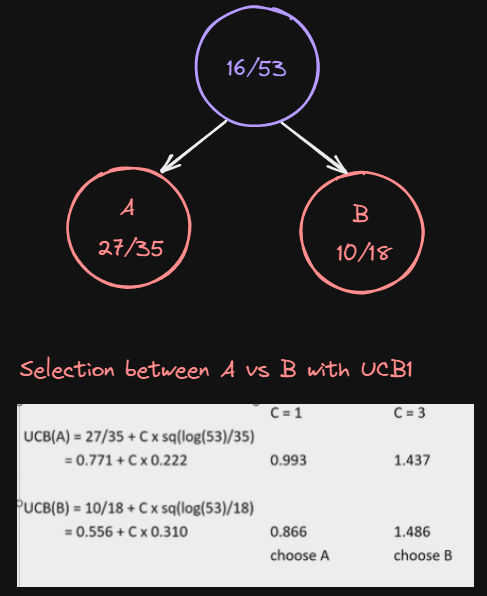
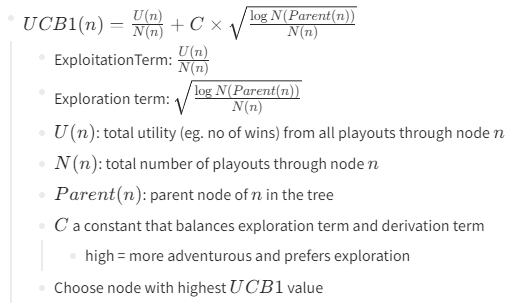
## Learning Evaluation Function Weights



* *Supervised Learning*:
* CONS: delayed reinforcement; credit assignment (which action responsible for win/loss?); only takes current state into consideration
* *Temporal Difference Learning*: reinforcement learning; multi-step prediction; correctness of prediction not known until several steps later
* TDLeaf(λ): uses temporal difference learning with Minimax

## MCTS

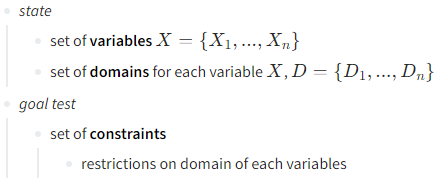


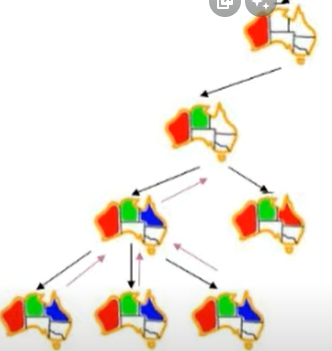
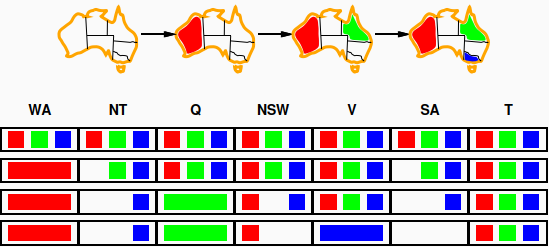
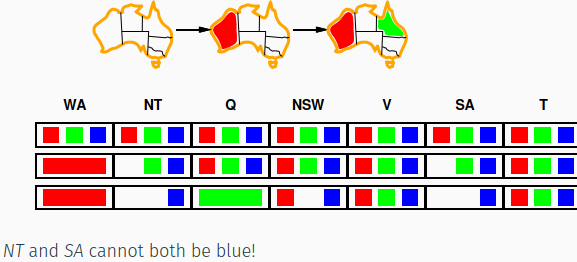
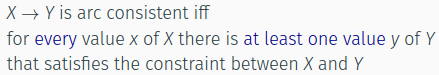


1. Expansion: create new node
2. Simulation: playout to terminal state
3. Backpropagation: update statistics

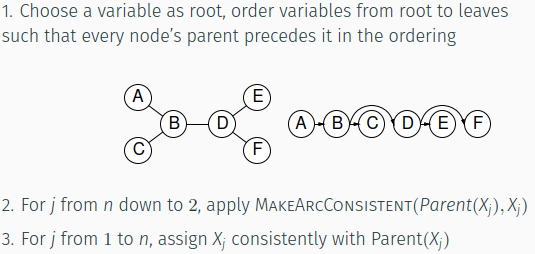
# Constraint Satisfaction Problems

* general-purpose algorithms with more power than standard search



* *Backtracking Search*: DFS with single-variable assignment
* 
* *Minimum Remaining Values (MRV)*: choose variable with fewest legal values
* *Degree Heuristic*: choose variable with most constraints on remaining values
* *Least Constraining Value*: rules out fewest values in remaining values
* *Forward Checking*: keep track of remaining legal values for unassigned variables; terminate when any variable has no legal values
* 
* *Constraint Propagation*: repeatedly enforce constraints locally – detects failures earlier (checks each variable’s arc consistency with every other variable)
* 
* *Arc Consistency*: every value for A, there is value for B such that both satisfy constraint
* 
* *Tree-structured CSPs*

| Theorem: if the constraint graph has no loops, the CSP can be solved in   * worst case CSP: |
| --- |

* 

# Bayesian Networks

| Chain Rule | Independence |
| --- | --- |
|  | A and B are independent iff  or |
| Conditional Probability | Bayes’ Rule |
|  |  |

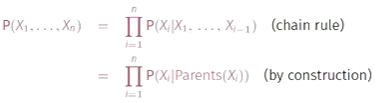
* *Bayesian Network*: set of nodes (one per variable); direct, acyclic; conditional distribution for each node given its parents,
* simplest case: conditional probability table (CPT) giving distribution over for each combination of parent values

|  | table has rows, =num parents  network requires vs for joint distribution   * 1+1+4+2+2=**10** vs -1=**31**   Full Joint Distribution: |
| --- | --- |

Construction

1. Choose an ordering of variables
2. For to

* add to the network
* select parents from such that
* Choice of parents guarantees the global semantics:



* Order: causes→symptoms

# Hidden Markov Model

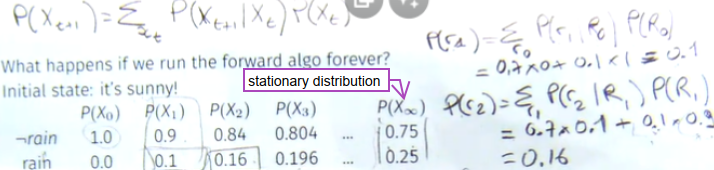
## Markov Models

* *Markov Assumption*: current state depends only on finite fixes number of states
* *First-order:* only considers previous state
* 
* *higher order*: still same probability distribution at each time step



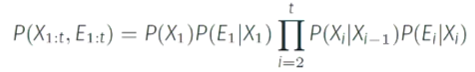
Stationary Distribution





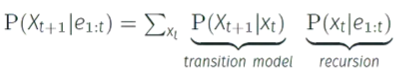
## Hidden Markov Models

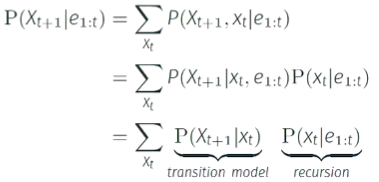
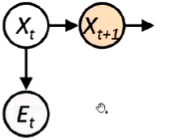
Joint Distribution



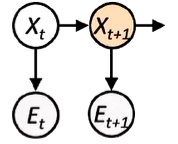
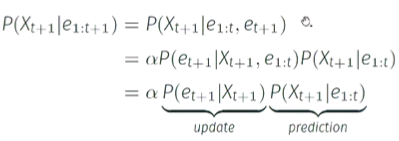
* State and Evidence independent of all past states and evidence, given the current state
* *Filtering*/*State Estimation*: with evidence gathered, can I estimate current state?
* *Prediction*: Given evidence gathered, can I determine future state?
* *Smoothing*: Given evidence up till now, can I compute a past state?
* *Most likely explanation*: given evidence gathered up till now, what are the most likely sequence of states that can explain my current state?
* *Learning*: given evidences, can we update the transition and sensor models

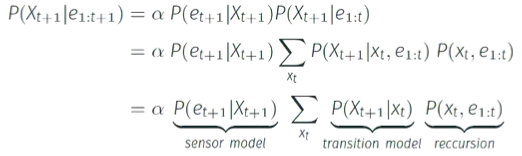
Forward Algorithm

* Prediction
* 
* Filerting
* 



Filtering





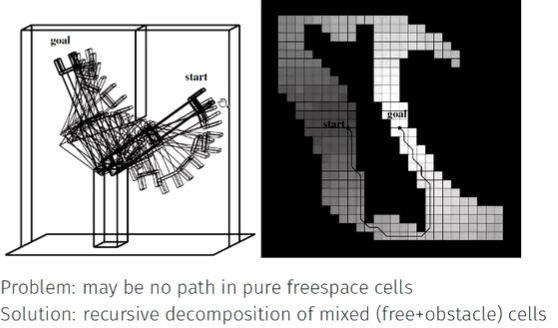
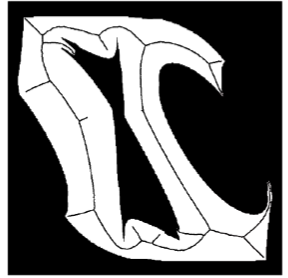
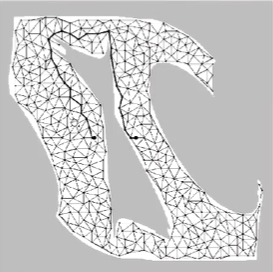
* is a normalizing constant: make probabilities sum to 1

# Robotics

Uncertainty

* *Localisation*
* *Sensor Model*: Use observations of landmark to estimate state of robot
* *Motion Model*: Update state using its movements
* *Mapping*

## Motion Planning

* *Cell Decomposition:* divide space into simple cells; each can traverse “easily” (eg. convex); discretise configuration space into cells; do search from start to end
* CONS: does not capture details in config space
* 
* *Skeletonization*: finite number of easily connected points/lines; form a graph such that any two points are connected by a path on the graph
* Voronoi Diagrams: locus of points equidistant from obstacles
* CON: does not scale well to higher dimensions
* 
* *Probabilistic Roadmap*: generate random points in C-space, keeping those in freespace; create graph by joining pairs by straight lines
* 
* CON: generating enough points to ensure that every start/goal pair is connected through the graph