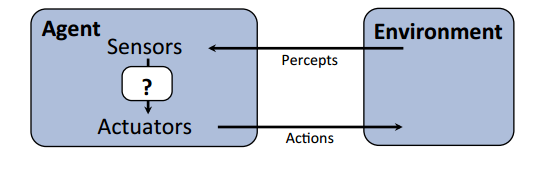
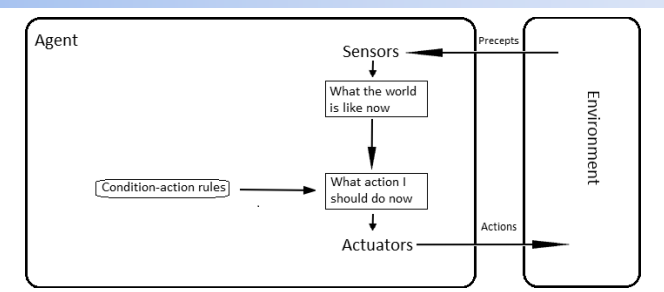
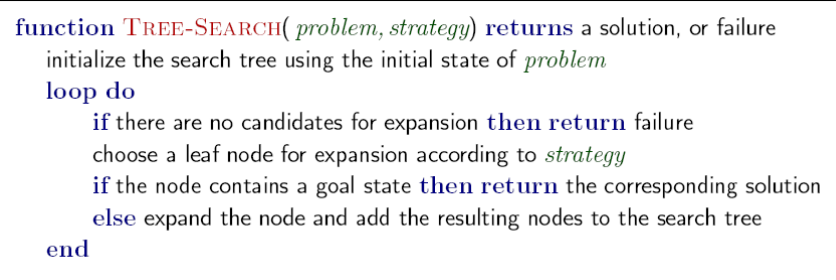
Intro:

1. Rational: maximizing the expected utility
2. Agent: an entity that perceives and acts
3. 

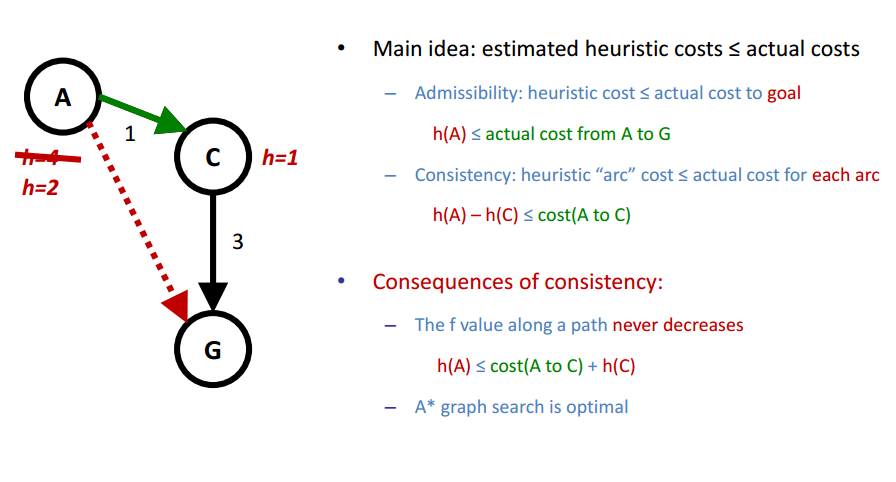
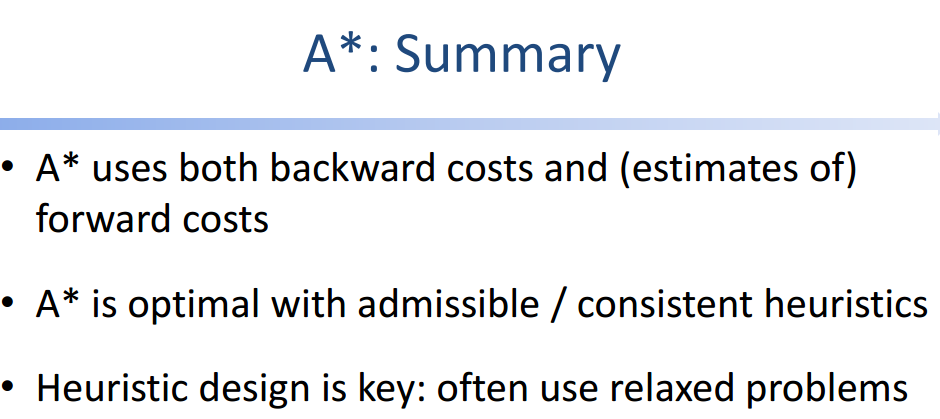
­Search:

1. Reflex Agents: Act only based on the current percept, not considering future, can be rational

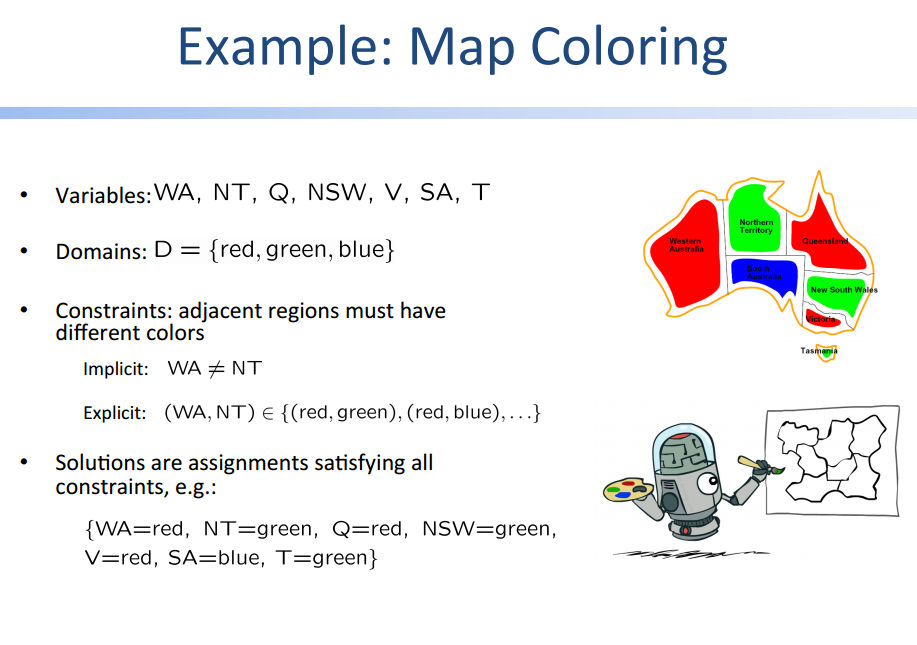


1. Planning Agents: Decision based on consequence of actions, must have a model of how the world evolves in response to actions
2. Task Environment: Performance Measure, Environment, Actuator, Sensor
3. Search Problem: A state space -> a successor function -> a start state and a goal state
4. State Space Graph: A mathematical representation of a search problem, keep track of visited state to make sure each state appears only once.
5. A search tree: Start state at root, children are successors
6. Tree Search: Fringe, Expansion, Exploration Strategy
7. Search Strategies: Completeness, time complexity, space complexity, optimality: b-> maximum branching factor, d-> depth of least cost function, m-> maximum depth of the state space
8. DFS: expand a deepest node first, fringe is a LIFO stack ==🡺 Properties: if m is infinite, DFS is complete ( prevent cycle), O(bm) time, O(bm) space, not optimal: find the leftmost solution.
9. BFS: expand a shallowest node first, FIFO fringe. Properties: O(bs) time, O(bs) space, optimal if cost is 1
10. Iterative Deepening: run DFS at increment depth. Properties: O(bs) time, O(bd) space
11. Uniform Cost Search: Expand the cheapest node first, fringe is PQ with priority as cost. Properties: effective depth-> cost of solution C, arc cost at least e, d= C/e, time O(bc/e), space O(bc/e) complete and optimal. UCS doesn’t tell about the goal location, explore in every direction

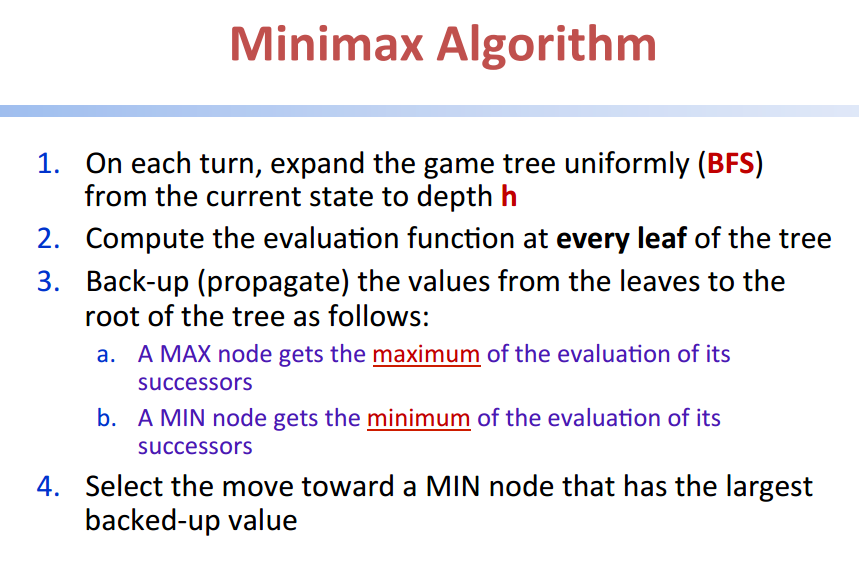
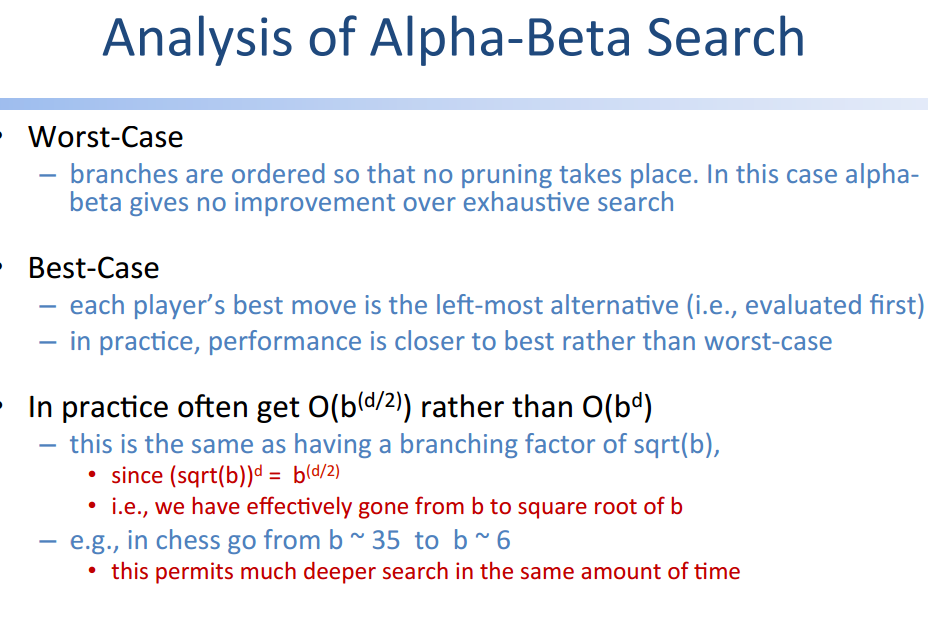
Heuristics:

1. Estimation of how close a state is to goal. Designed for a particular search problem.
2. A start search: combining UCS and Greedy f(n) = g(n) +h(n)
3. Admissibility: 0<=h(n)<=h\*(n)
4. Optimality: A is optimal solution, B is suboptimal, A will exit before B if heuristics is admissible.
5. A trade-off between quality of estimate and work per node, expand fewer node but usually do more work per node.
6. A problem with fewer restriction on the action is called a relaxed problem. Cost of a relaxed problem is a admisable heuristic
7. Consistency: heuristic “arc” cost smaller than actual cost of each arc: consequence: the f value along a path never decreases 
8. Tree Search is optimal if heuristic is admissable, graph search is optimal if heuristic is consistent.
9. 
10. Properties: Complete->unless infinite number of nodes with f<= f(G), time: bd, space: bd, optimal ,
11. Simple-Memory Bounded A\*, delete the largest f value node when memory is full
12. Search is for world with single agent, deterministic actions, fully observed state, discrete state space

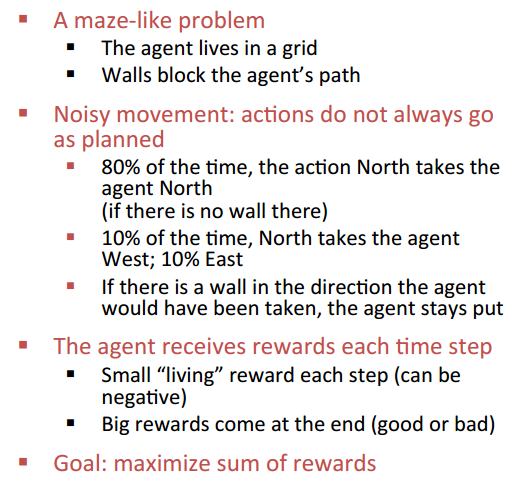
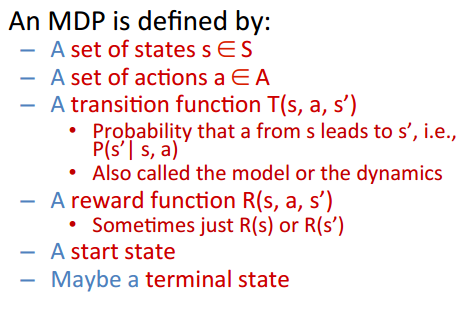
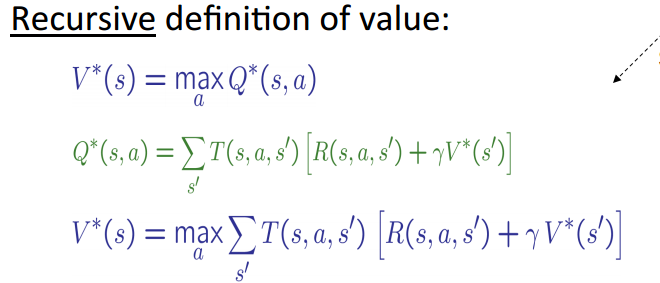
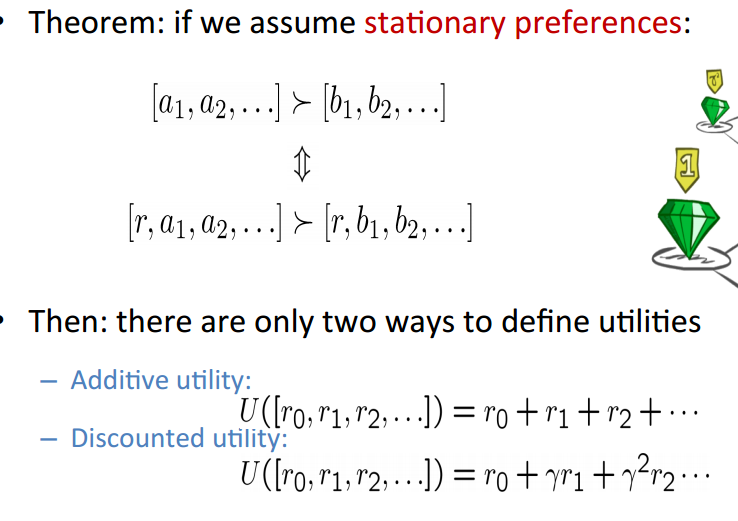
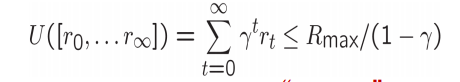
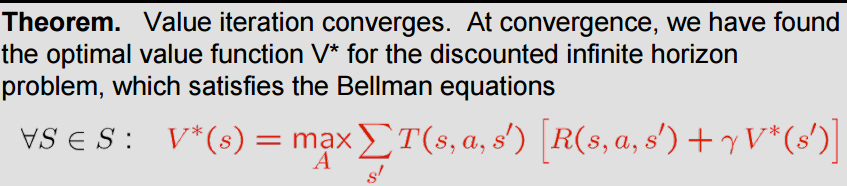
CSP (Constraint Satisfaction Problem)

1. A special subset of search problem, state is defined by variable X, with value from a domain D, goal test is a set of constraints specifying allowable combination of values of subsets of variables.
2. 
3. Standard search formulation: state defined by values assigned so far. Initial state: the empty assignment, successor function assign a value to an unassigned value, the current assignment is complete and all constraints satisfied.
4. Backtracking Search: one variable at a time, check constraints as you go. (depth first search with these two modifications)
5. Improvement: filtering -> keep track of domains for unassigned variables and cross of bad options, forward checking-> cross off options that violates constrains when added to existing assignments.
6. Arc Consistency: an arc X -> Y is consistent iff for every x in the tail there is some variables y in the head which can be assigned without violating constraint. Forward checking: Enforcing consistency of arcs pointing to each new assignment. (DELETE FORM THE TAIL)
7. Ordering: minimum remaining value: choose the variable with the fewest legal left values in its domain -> fail fast ordering; least constraining value=> choose the variables with least constrains.
8. Iterative algorithm for csp: minConflicts: apply local search to CSPs => take an assignment with unsatisfied constraints. Greedy algorithm: variable selection: randomly select any conflicted variable, value selection: min-conflicts heuristic, choose a value that violates the fewest constraints
9. Local Search improves a single option until you can’t make it better; successor function => local changes; generally, much faster and memory efficient but incomplete and suboptimal.
10. Hill climbing: move in the direction of increasing evaluation function f.=> greedy local search
11. Gradient: use the second order information to converge faster => more computation => Randomly Restart:

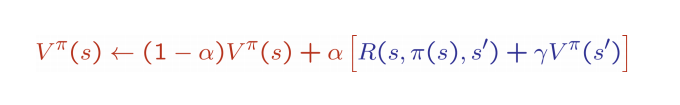
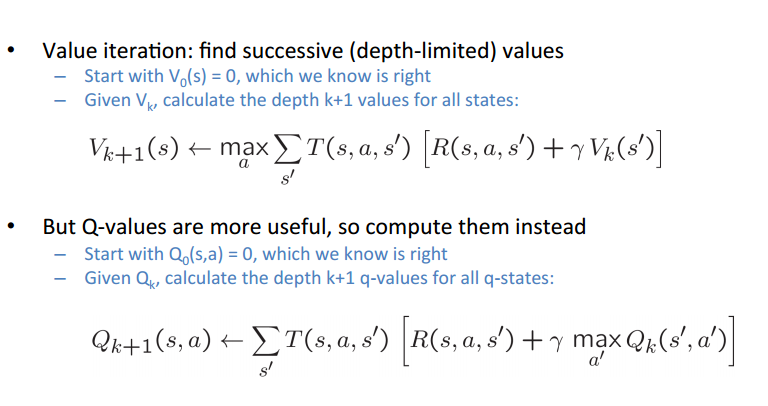
Adversarial Search:

1. Value of state: the best achievable outcome from the state;
2. Settings : two players, turn taking, deterministic, zero sum, all states observable
3. Uncertainty caused by another agent (MIN) competing with (MAX)
4. Basic Idea: Using current state as initial, build the game tree uniformly to the maximal depth h feasible in time limit, evaluate the states of the leaf nods. Backup the results from the leaf assuming the worst for MAX from MIN
5. 
6. Optimal against perfect player, time complexity O(bm), space O(bm)
7. Alpha-Beta Prunning: alpha is the highest value along the path found for MAX, beta is the lowest value found for MIN; update alpha and beta and prune the remaining branches as soon as the value is known to be worse than the current alpha or beta for MAX or MIN
8. 
9. Multiple Player: 1 MAX 2 MIN
10. Evaluation Function: weighted sum of features (return a better minmax value)
11. Expected Min-Max

Markov Decision Process:

1. AI as Planning: non-deterministic search -> MDP; AI as Learning: model of world partially known ->RL
2. 
3. 
4. Optimal Quantities: the value (utility) of a state s: V\*(c)= expected utility starting in s and acting optimally; value of a state (s,a) Q\*(c) = expected utility starting on having taken actions a from state s and thereafter acting optimally; optimal policy: optimal action from state s;
5. 
6. Discounting: its reasonable to prefer rewards now to later, value of rewards decay exponentially
7. Stationary Preference: 
8. Normative Theory of Discounting: Money should be discounted at a constant rate over time. This implies that preference will be consistent over time.
9. Infinite Utilities: If the game last forever, => Finite depth: terminate after a fixed steps, or given non stationary policies ( depending on time left); Discounting: 
10. Value Iteration Convergence: 

Reinforcement Learning:

1. Receive feedback in the form of rewards; Agent utility is defined by the reward function; must learn to act so to maximize the utility; all learning is based on observed samples of outcomes
2. Model based learning: Learn an approximate model based on experiences; solve for values as if the learned model were correct; 1) learn empirical MDP model 2) solve the learned MDP
3. Passive Reinforcement Learning: Input a policy, follow the policy and learn the state value
4. Direct Evaluation: Act according policy, every time visit a state, write down the sum of discounted rewards turned out to be; average the samples => easy to understand, doesn’t require knowledge of T, R; eventually computes the correct values; waste information about state connection
5. Temporal Difference Learning: Policy fixed => do evaluation; update V(s) each time we experience a transition, likely outcomes will contribute updates more often. Problem=> cannot turn values into a new policy => learn Q values instead
6. Active Reinforcement Learning: choose the actions now => learn the optimal policy 
7. 