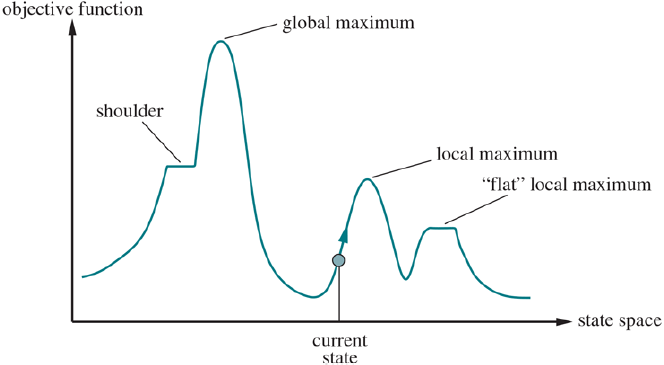
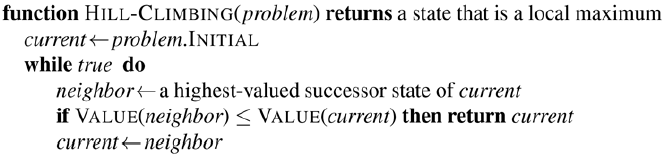
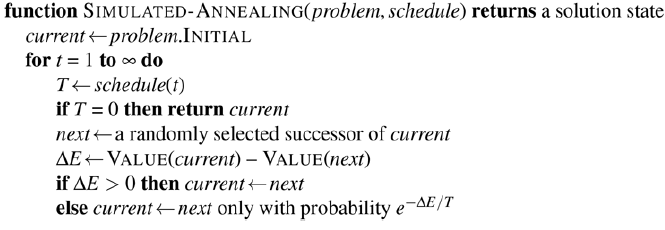
**Sometimes we care only about the final state, not the path to get there**

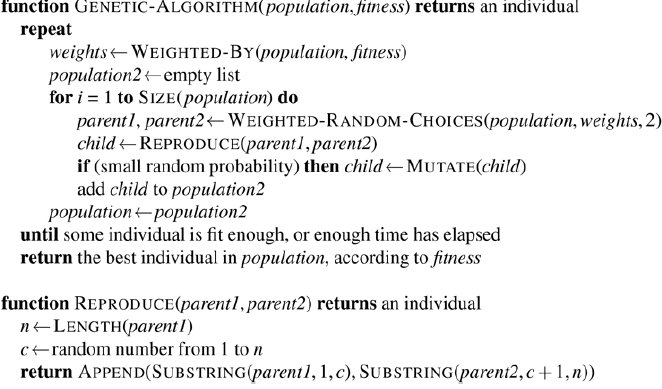
* **Local search** - operates by searching from a start state to neighboring states without keeping track of paths nor the set of states that’ve been reached
  + Not systemic
  + Use very little memory
  + Can find reasonable solutions in large/infinite spaces for which systemic algorithms are unsuitable
  + **Can solve optimization problems:** aim is to find the best state according to an objective function
* **State-space landscape** - each point (state) in the landscape has elevation defined by the value of objective function
  + If elevation corresponds to **objective function** then we want **global maximum**
    - The process is called **hill climbing**
  + If elevation corresponds to **cost** then we want **global minimum**
    - Process is called **gradient descent**
* **Hill-climbing search** - keeps track of one current state and on each iteration moves to the neighboring state with the highest value
  + heads in the direction that provides **steepest ascent**
  + **Terminates** when it reaches a peak where no neighbor has higher value
  + Does not look ahead beyond immediate neighbors
  + You can use the **negative of a heuristic** cost function as objective function
    - Climbs locally to the state with the smallest heuristic distance to the goal
  + Sometimes called **greedy local search** because it grabs good neighbor state without thinking ahead
    - Rapid progress toward solution because usually it’s easy to improve a bad state
  + Can get stuck with
    - **Local maxima**: peak that is higher than each of its neighboring states but lower than global maximum
      * Drawn to the peak then stuck
    - **Ridges**: result in a sequence of local maxima that’s difficult for greedy algorithms to navigate
    - **Plateaus:** flat area of the state-space
      * Can be flat local maximum (no uphill exit)
      * Can be a shoulder (progress is possible)
  + Steepest-ascent hill climbing gets stuck a lot but works quickly. How can we solve more problems?
    - **Sideways move** when plateau is reached hoping that plateau is a shoulder
      * But on flat local maximum, search will wander forever
      * Limit number of consecutive sideways moves
  + **Stochastic hill climbing**: chooses at random from uphill moves
    - Probability of selection can vary with the steepness of the move
    - Converges slower than steepest ascent but can find better solutions
  + **First-choice hill climbing**: stochastic hill climbing by generating successors randomly until one is generated that’s better than current state
    - Good strategy when a state has many successors
  + **Random-restart hill climbing**: conducts series of hill climbing searches from randomly generated states until goal is found
    - Complete with probability 1 because eventually generates goal state as initial state
    - If each search as a probability *p* of success, expected number of restarts required is 1/*p*
    - Expected number of steps is the cost of one successful iteration plus (1 - *p*)/*p* times the cost of failure
  + Success of hill climbing depends very much on shape of state-space landscape
    - Few local maxima and plateaus, random-restart will find a good solution quickly
    - NP-hard problems typically have exponential number of local maxima to get stuck on
      * Reasonably good local maximum can be found after small number of restarts
* **Complete-state formulation** - every state has all the components of a solution but they might not all be in the right place
* **Simulated annealing -** combining hill climbing with a random walk in a way that yields efficiency and completeness
  + **Annealing**: process used to temper or harden metals by heating them to a high temp and gradually cooling them
  + Imagine task of getting ball into the deepest crevice in a bumpy surface
    - Letting it roll it will come to rest at a local minimum
    - If we shake the surface, ball bounces out of local minimum maybe in a deeper local minimum
    - Trick is to shake hard enough to bounce ball out of local minima but not hard enough to dislodge it from global minimum
    - Simulated-annealing solution is to start by shaking hard and then gradually reduce intensity of shaking
  + Similar to hill climbing but instead of picking best move, picks *random* move
    - If move improves situation, it’s accepted
    - Otherwise algorithm accepts the move with some probability less than 1
  + Probability decreases exponentially with the ‘badness’ of the move, the amount of by which the evaluation is worsened
  + Probability also decreases as *T* goes down
    - Bad moves are more likely to be allowed at the start when *T* is high
    - Become unlikely as *T* decreases
  + If the *schedule* lowers *T* to 0 slowly enough, by property of Boltzmann distribution, all the probability is concentrated on the global maxima
    - Algorithm will find with probability approaching 1



* **Local beam search** - keeps track of *k* states rather than just one
  + Begins with *k* randomly generated states
  + Each step all successors of all *k* states are generated
  + If goal isn’t hit, selects *k* best successors from complete list and repeats
  + Seems similar to running *k* random restarts in parallel instead of in sequence
    - In random-restart each search process runs independently
    - In local beam search useful ino is passed among the parallel search threads
    - Algorithm quickly abandons bad moves and moves resources to where progress is being made
  + Can suffer from lack of diversity among *k* states since they can cluster in a small region of the state space (makes search *k* times slower than hill climbing)

**Stochastic beam search** - instead of choosing top *k* successors, choose successors with probability proportional to the successor’s value (increasing diversity)

* **Evolutionary algorithms** - there is a population of states in which the highest value states produce successor states that populate the next generation in a process called **recombination**
  + Evolutionary algorithms vary by
    - **Size of population**
    - **Representation of each individual**
      * In evolution strategies an individual is a sequence of real numbers and in genetic programming an individual is a computer program
    - **Mixing number** : number of parents that come together to form offspring
      * Common case is =2
      * If =1 we have stochastic beam search
    - **Selection process** for selecting individuals who become parents for next generation
      * Could select from all individuals with probability proportional to their fitness core
      * Could randomly select *n* individuals () and select the most fit ones as parents
    - **Recombination procedure**
      * Can randomly select a crossover point to split each parent strings and recombine parts to form two children
        + 1 has first part of parent 1 and second part of parent 2, vice versa for 2nd child
    - **Mutation rate**
      * Determines how offspring have random mutations to representation
      * Once offspring is generated every bit in its composition is flipped with probability equal to the mutation rate
    - **Makeup of next generation**
      * Can be newly formed offspring
      * Elitism: including a few top scoring parents from previous generation
        + Guarantees overall fitness never decreases over time
      * Culling: all individuals below given threshold are discarded
        + Can lead to a speedup
  + Often population is diverse early in process
    - crossover takes large steps in state space early (like simulated annealing)
  + After many generations of selection toward higher fitness
    - , population is less diverse and smaller steps are typical
  + Similar to stochastic beam search but with addition of **crossover**
    - Advantageous if blocks perform useful functions
    - Can be shown mathematically if blocks don’t serve purpose, crossover conveys no advantage
  + **Schema:** substring in which some positions can be left unspecified
    - **Instances:** strings that match the schema
    - If the average fitness of the instances of a schema is above the mean, then the number of instances of the schema will grow over time
  + Genetic algorithms work best when schemas correspond to meaningful components of a solution



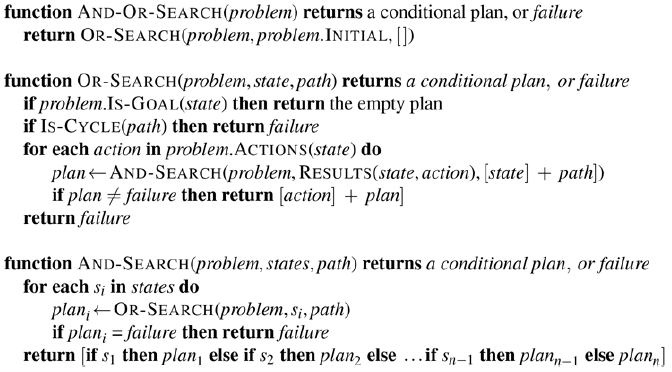
**A continuous action space has infinite branching factor and can’t be handled by most algorithm so far (except first-choice hill climbing and simulated annealing)**

* State spaces are defined by *n*-dimensional vector of variables, *x*
  + *n-dimensional* = *n* variables
* One way to deal with a continuous state space is to discretize it
* Alternatively can make branching factor finite by sampling successor states randomly ny some value
  + **Empirical gradient search:** methods that measure progress by the change in the value of the objective function between two nearby points
  + Same as steepest ascent hill climbing in a discretized version of the state space
  + Reducing over time gives more accurate solution but doesn’t necessarily converge to global optimum
* **Objective function expressed in a mathematical form to use calculus to solve the problem**
  + Some methods attempt to use the gradient of the landscape to find maximum
  + Gradient of objective function is vector that gives magnitude and direction of steepest slope
    - Sometimes can find maximum by solving =0
    - Can’t solve in closed form (can compute local gradient but not global)
  + Given locally expression for gradient can perform steepest-ascent hill climbing by updating current state according to formula
  + **Step size:**  (alpha), small constant
    - If alpha is too small, too many steps are needed
    - If alpha is too large, search overshoots maximum
  + **Line search**: extending the current gradient direction
    - E.g. doubling alpha until *f* starts to decrease
* **Newton-Raphson** **Method** - general technique for finding roots of functions
  + Solving equations of form *g*(*x*) = 0
  + Works by computing new estimate for the root *x*
  + To find max/min of *f* need to find *x* such that gradient is a zero vector
  + **Hessian matrix**:
* **Constrained optimization** - optimization problem is constrained if solutions must satisfy some hard constraints of the variables
  + Difficulty depends on nature of constraints and objective function
  + **Linear programming**: best known category of constraint optimization problems in which constraints must be linear inequalities forming a convex set and objective function is also linear
    - Special case of more general **convex optimization**
      * Allows the constraint region to be any convex region and the objective to be any function that is convex within the constraint region

**Belief state** - set of physical states that the agent believes are possible

**In partially observable and nondeterministic environments the solution to a problem is no longer a sequence but a conditional plan (aka contingency plan, strategy) that specifies what to do depending on what percepts agent receives while executing the plan.**

* **Search with nondeterministic actions** - agent will be able to observe at runtime but doesn’t know at planning time. Solutions are trees rather than sequences
  + In deterministic environment, the only branching introduced is by agent’s own choices
    - **OR nodes**
  + In nondeterministic environment branching is also introduced by environment’s choice of outcome for each action
    - **AND nodes**
* **AND-OR tree** combines both type of nodes
  + Solution for AND-OR search problem is a subtree of the complete search tree that has
    - Goal node at every leaf
    - Specifies one action at each of its OR nodes
    - Includes every outcome branch at each of its AND nodes
  + Cycles are common in nondeterminisitc problems. How to deal?
    - If current state is identical to a state on the path from the root, return with failure
    - Doesn’t mean there is no solution from current state
      * IF there IS a noncyclic solution it must be reachable from earlier incarnation of current state so new incarnation can be discarded
    - Ensures algorithm terminates in every finite space
      * Every path must reach a goal, dead end or repeated state
  + Algorithm doesn’t check wheter current state is a repetition of state on other path from root
  + Can be explored either breadth or best first
    - Concept of heuristic must be changed to estimate cost of contingent solution rather than sequence
    - Notion of admissibility carries over



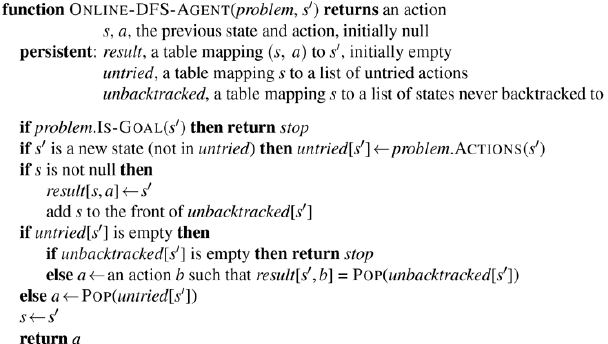
* **Some problems need a cyclic solution**: keep trying function until it works
  + When is a cyclic plan a solution
    - Minimally every leaf is a goal state
    - Leaf is reachable from every point in the plan
    - Must consider cause of nondeterminism
      * Is it due to random chance? Or unobserved fact about agent or environment?
        + Will repetition really help?

**Searching in partially observable environments requires agent’s actions to be aimed at reducing uncertainty about current state**

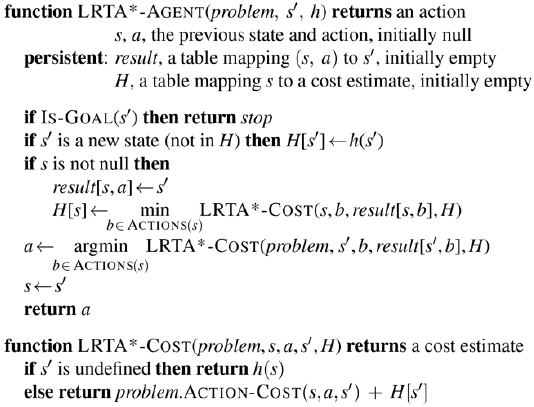
* **Sensorless (conformant) problem -** when agent’s percepts provide no information at all
  + Sometimes better even when a conditional plan with sensing is available
  + Agent can **coerce** the world into goal state
  + Solution is a sequence of actions not conditional plan
  + In a fully observable environment each belief state contains one physical state
    - But for partially observable environment we search in the space of **belief states** rather than physical states
    - In belief state space the problem is fully observable because agent always knows its own belief state
  + Percepts received after each action is predictable: Always empty
    - No contingency to plan for
    - **True even if environment is nondeterminisitc**
* **Transform physical-state problem into belief state problem**
  + **States**: belief state space contains every possible subset of physical states
    - If *P* (physical) has *N* states then belief-state has belief states although many of those may be unreachable from initial state
  + **Initial state**: typically the belief state consisting of all states in *P* but sometimes agent might have more knowledge
  + **Actions:** agent is unsure which actions are legal
    - Can we assume illegal actions have no effect on environment?
      * If true, safe to take the union of all actions in any physical state in the current belief state *b*
      * If false, only allow the intersection, the set of actions legal in all states
  + **Transition model:** For deterministic actions new belief state has one result state for each current possible states
    - With nondeterminism the new belief state consists of all possible results of applying action to any states in the current belief state
  + **Goal Test**: agent possibly achieves goal if any state *s* in the belief state satisfies goal test of underlying problem
    - Agent necessarily achieves goal if every state satisfies goal state
  + **Action Cost:** if same action can have different cost in different state, cost of taking action in a given belief state could be one of several values
    - Assume that cost of action is same in all states and so can be transferred directly from underlying physical problem
* When the belief state problem is formulation is constructed from the definition of the underlying physical problem we can solve sensorless problems with any of the ordinary search algorithms
* Newly reached states are tested to see if they were previously reached
  + If superset of belief states have been generated and found to be solvable, any subset is guaranteed to be solvable
* **Belief state space is often too vast for it to be feasible in practice**
  + Can try to represent belief state b y some more compact description
  + Can avoid standard search algorithms which treat belief states as black boxes
    - Instead look inside belief states and develop **incremental belief-state search** algorithms
      * Has to find ONE solution for every state in belief state
      * Able to detect failure quickly

**A little bit of sensing can go a long way**

* **Searching in partially observable environments** - problem specification will specify percept function that returns percept received by agent in given state
  + If sensing is non deterministic, can use percept function that returns set of possible percepts
  + Usually several states will produce same percept
* 3 stages of transition model between belief states for partially observable problems
  + **Prediction stage**: computes belief state resulting from the action
  + **Possible percept stage**: computes the set of percepts that could observed in the predicted belief state
  + **Update stage**: computes for each possible percept the belief state that will result from the percept
* Agent deals with possible percepts at planning time because it doesn’t know actual percepts until plan is executed
  + Observations can only help reduce uncertainty
* For deterministic sensing belief states for different possible percepts will be disjoint forming a partition of the original predicted belief stage
* Can be solved using AND-OR algorithm
  + **Solution will be a conditional plan**
  + Because we supplied belief state problem to AND-OR, it returns conditional plan that tests belief state rather than actual state
    - In partially observable environment the agent won't know actual state
  + Can improve solution by checking for previously generated belief states that are subsets or supersets of current state
  + Can service incremental search algorithms
* **Agent for partially observable environment**
  + 2 main differences between this agent and agent for fully observable deterministic environments
    - Solution will be conditional plan rather than sequence
      * Agent needs to test condition and execute appropriate branch of conditional
    - Agent will need to maintain its belief state as it performs actions and receives percepts
      * Percept is given by environment rather than calculated by agent
* **Maintaining one’s belief state is a core function of any intelligent system**
  + Also known as monitoring, filtering, state estimation
  + This computation has to happen fast as percepts are coming in
    - As environment becomes more complex agent only has time to approximate belief state
      * Maybe focusing on implications of the percept for aspects that are of interest
  + **Localization**: when agent works out where it is given a map of the world and sequence of percepts and actions
    - With nondeterministic actions, PREDICT step grows belief state but UPDATE step shrinks it back down
      * As long as percepts provide useful information
    - For environments with reasonable variation ing geography, localization often converges to a single point quickly even when actions are nondeterministic
* **Offline search algorithms** - compute complete solution before taking first action
* **Online search algorithms** - first takes action, observes environment, and computes next action
  + Good idea in dynamic/semi-dynamic environments where there's penalty for sitting around and computing
  + Helpful in nondeterministic domains because allows agent to focus computational efforts on contingencies that actually arise
  + The more an agent plans ahead, less often it will be stuck in the future
  + Agent must use actions as experiments in order to learn about the environment
* **Online search problems** - assume deterministic and fully observable environment
  + Agent has following knowledge
    - ACTIONS(*s*): legal actions in state *s*
    - *c(s, a, s’)* cost of applying action *a* in state *s* to arrive at *s’*
      * Cannot be used until agent KNOWS *s’* is the outcome
    - ISGOAL(*s)* goal test
    - Agent can’t determine RESULT(*s, a*) except by actually being in *s* and doing *a*.
    - Agent might have access to admissible heuristic *h(s)* that estimates distance from current state to goal state
  + **Competitive ratio:** Compare actual path cost with the path cost agent would incur if it knew the search space in advance
  + Vulnerable to dead ends (states from which no goal state is reachable)
  + No algorithm can avoid dead ends in all state spaces
    - No way agent could know hot to choose the correct action in both state spaces
    - **Adversary argument**: imagine adversary constructing state space putting goals and dead ends where it chooses as agent explores
    - Some actions can be irreversible, no way to return to previous state
    - Exploration algorithm is only guaranteed to work in state spaces that are safely explorable
      * Some goal state is reachable from every reachable state
  + No bounded competitive ratio can be guaranteed if there are paths of unbounded cost
    - Better to characterize performance of online search algorithm in terms of size of entire state space rather than depth of shallowest goal
* **Online search agents** - can only discover successors for a state it physically occupies
  + E.g. A\* can expand node in one part of space and expand a node somewhere completely different because expansion is simulated
  + Might be better to expand nodes in a local order (like DFS)
  + Difficulty comes when agent has tried all actions in a state
    - Agent has to backtrack in the physical world
      * Algorithm keeps another table that lists predecessor states for each state to which the agent has not yet backtracked
      * If agent runs out of states which it can backtrack, search is complete
  + For exploration, algorithm is optimal but competitive ratio could be bad
  + Only applicable in state spaces with reversible actions



* **Online local search** - hill-climbing search is already an online search algorithm
  + Base algorithm isn’t useful because agent sitting at local maxima will be stuck
  + Can’t do random restarts
  + **Random walk**: randomly select available action from current state
    - Preference can be given to an action that hasn’t been tried
    - Wille eventually find a goal or complete exploration given space is finite and safely explorable
    - Can be very slow
  + **Augmenting with memory**
    - Store *H*(*s)* “current best estimate” of the cost to reach the goal from each state that’s been visited
      * **Learning real-time A\***
        + Builds map of environment in result table
        + Updates cost estimate for the state it’s left and chooses the best move according to current cost estimate
        + **Optimism under uncertainty:** Actions that haven’t been tried in a state *s* are always assumed to lead immediately to the goal with the least possible cost
      * Guaranteed to find goal in any finite safely explorable environment
        + Not complete for infinite state spaces
        + Explores environment of *n* states in steps (worst case)



* Opportunities for learning
  + Agents map outcome of each action in each state
  + Local search agents acquire more accurate estimates of the cost of each state by using local updating rules
  + Updates eventually converge to exact values for every state, provided the agent explores space in the right way
    - Once exact values are known, pure hill climbing is optimal