**Uninformed Search**

Fully observable – all necessary information about a problem domain is accessible so that each state is a complete description of the world; there is no missing information at any point in time

Uninformed means we only know the goal test, and the successors() function, and not which non-goal states are better

Completeness- a complete algorithm will find a solution (not all) if one exists

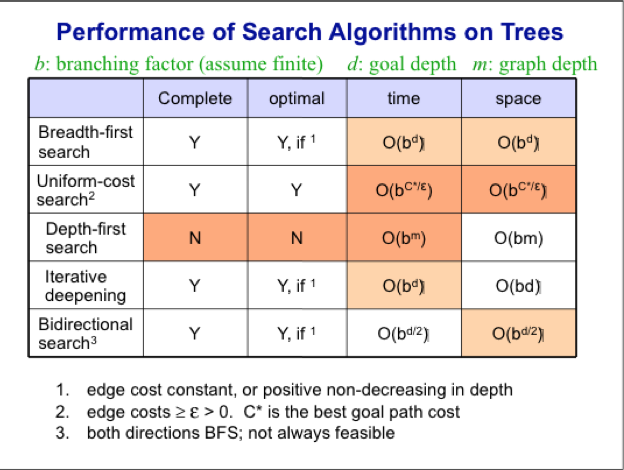
Optimality/Admissibility- an admissible algorithm will find a solution with minimum cost

Time Complexity- count number of nodes expanded

Space Complexity- maximum size of Frontier

Deterministic – games where there is no chance involved

* Tree Search Algorithm -does not detect goal when node is generated, does not detect repeated states in state space (could end in infinite loop)
* Breadth-First Search- Queue (FIFO) used for frontier, remove from front add to back, complete, optimal if constant costs or non-decreasing costs with depth
* Depth-First Search- Stack (LIFO) used for the frontier, remove from front add to front, not complete, not optimal
* Uniform-Cost Search- uses priority queue to order nodes based on path cost on the frontier list, called Dijkstra’s algo, complete and optimal
* Iterative-Deepening Search- do DFS to a certain depth and then keep increasing depth, complete and optimal as for BFS
* Goal checking for all these is done when node is removed from frontier



**Informed Search**

Informed Search- uses domain knowledge to guide selection of the best path to continue searching

Heuristics- informed guesses for the path to goal, computable from current state description, h(n) >= 0, if closer to 0, then node n is close to goal state

Consistent- if h(n) <= c(n,n’) + h(n’) where n’ is a child to node n, implies values of f along any path are non-decreasing, stronger condition than admissibility, means optimal path is found

Devising Heuristics- want an admissible heuristic that is as close to the actual cost w/o going over, must be relatively fast to compute, if h(n) = h\*(n) for all n, then only nodes on optimal solution path are expanded, any failure to decrease h will immediately cause A\* search to switch to another path

* Best-First Search- sort nodes in the Frontier by increasing value of an evaluation function, f(n)
* Greedy Best-First Search- use as an evaluation function, f(n) = h(n), using priority queue as frontier and sorting by increasing values of f, not complete, not optimal
* Beam Search- same as Greedy Best-First Search, but restrict the max size of the Frontier, more space efficient than Greedy Best-First Search, but can throw away node on solution path
* Algorithm A Search- uses evaluation function of f(n) = g(n) + h(n), where g is cost from start node to node n and h is the estimated cost from node n to goal, not optimal
* Algorithm A\* Search- same evaluation function as A Search, except add constraint that for all nodes n, h(n) <= h\*(n), where h\*(n) is the actual cost of the minimum-cost path from n to a goal, when h(n) <= h\*(n) holds for all n, h is called an admissible heuristic function, guarantees that a node on optimal path will be considered, complete and optimal, terminates only when goal is removed from priority queue, if h = 0, then same as Uniform cost search, might revisit a state and discover better path, so put node with smallest g value back in priority queue, O(number of states), can run out of memory
* Iterative Deepening A\* Search- cutoff based on f(n) rather than depth, don’t expand any node with an f-value that exceeds current threshold, complete, optimal, linear space required
* Local Searching- search for a path from start state to a goal state, then “execute” solution path’s sequence of operators, every node is a solution, the solutions that can be reached with one application of an operator are in the current solution’s neighborhood, considers next only those solutions in the neighborhood
* Hill-Climbing Search- stops at local maximum, doesn’t allow backtracking or jumping to an alternative path since no Frontier, very fast, similar to Beam Search with beam width of 1, follows the direction of the steepest ascent, can get stuck at local maximum and plateaus, there is also valley finding search which minimizes f(n)
* Stochastic Hill-Climbing- randomly select among the neighbors
* First-choice hill-climbing- randomly generate neighbors, one at a time, it neighbor is better then take that move
* Simulated Annealing (Stochastic Hill-Climbing)- helps escape local optima, but might pass the global optimum after reaching it, Let deltaE = f(newNode) – f(currentNode) < 0 and T is annealing temperature which controls frequency of acceptance of bad steps, gradually want to reduce temperature T(k), fast, simulated annealing is only guaranteed to converge to a global optimal solution in probability with an infinite number of iterations and an infinitely slow cooling rate
  + Boltsman’s equation = p = edeltaE/T
  + Cooling Schedule- choice of parameters (T, L) where L is the number of steps the search is allowed to proceed at each temperature

**Genetic Algorithms**

-another local search method, kind of hill-climbing search, similar to randomized beam search

Mechanisms of Evolutionary Change

crossover (alteration): the combination of 2 parents’ during reproduction resulting in offspring that have some traits of each parent, requires genetic diversity among parents to ensure varied offspring

mutation: the rare occurrence of errors during the process of copying chromosomes

natural selection: the fittest survive in a competitive environment resulting in better organisms

Vector- how a solution is represented

Deterministic Selection- relies heavily on evaluation/fitness function, converges fast

Proportional Fitness Selection- each individual is selected proportionally to their fitness score, even the worst individual has a chance to survive, rank selection: individual selected with a probability proportional to its rank in population sorted by fitness, proportional selection: individual selected with a probability: Fitness(individual) / ∑ Fitness for all individuals

Tournament Selection- randomly select two individuals and the one with the highest rank goes on and reproduces, defines a probability on the chances that any individual has to reproduce for the next generation equal to (2s-2r+1)/s2, where s is size of population and r is rank of the winning individual

Crowding- a potential problem associated w/ selection, occurs when the individuals that are most-fit quickly reproduce so that a large percentage of the entire population looks very similar, reduces diversity in the population

N-point Crossover- pick n dividing points in the parents’ vectors and splice together alternating segments

Uniform Crossover- the value of each element of the vector is randomly chosen from the values in the corresponding elements of the two parents

**Game Playing**

Zero-sum- one player’s gain is the other player’s loss

Discrete- states and decisions have discrete values

Finite- finite number of states and decisions

Deterministic- no chance

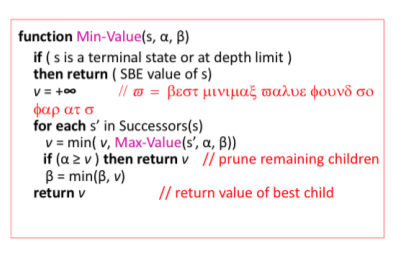
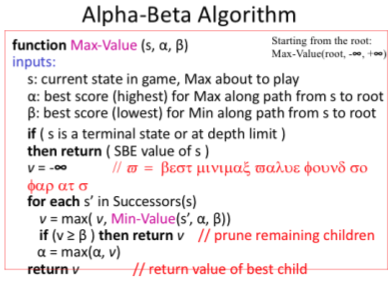
Perfect Information- each player can see the complete game state

* Greedy Search using an Evaluation Function- a utility function is used to map each terminal state of the board to a score indicating the value of that outcome to the computer, make the initial move that results in the board configuration with the maximum value
* Minimax Principle- assume both players play optimally, propagate values up the tree, DFS so O(bd) for space complexity and O(bd) for time, minimax algorithm applied to complete game tress isn’t practical so instead do depth-limited search to ply (depth) m

Static Evaluation Functions- use heuristics to estimate the value of non-terminal states

Static Board Evaluation Function- used to estimate how good the current board configuration is for the computer, reflects computer’s chances of winning from that node, isn’t perfect at estimating/scoring

* Alpha-Beta Search- pruning can be used to ignore some branches, if a>=v for some Max node ancestor, don’t visit any more of the current MIN node’s children, if v >=B for some MIN node ancestor, don’t visit any more of the current MAX node’s children, alpha cutoff means not visiting some of MIN’s childs, effectiveness depends on the order in which successors are examined, more effective if best successors are examined first, in practice often O(bd/2) rather than just O(bd), use iterative deepening search when there is a time limit for a move



The Horizon Effect- sometimes disaster lurks just beyond the search depth, to avoid this do quiescence search or secondary search

Quiescence Search- when SBE value is frequently changing, look deeper than the depth limit until game “quiets down”

Secondary Search- find best move looking to depth d, then look k steps beyond to verify that it still looks good

Non-Deterministic Games- weight scored by the probability that move occurs, instead of using max or min, compute the average, weighted by the probabilities of each child, alpha-beta pruning is less effective

* Monte Carlo Tree Search- best-first search based on random sampling of search space, concentrate search on most promising moves
* Pure Monte Carlo Tree Search- for each possible move of current player, simulate k random games by selecting moves at random for both players until game is over, count how many were wins out of each k playouts, move with most wins is selected

Exploitation- keep track of average win rate for each child from previous searches, prefer child that has previously lead to more wins

Exploration- allow for exploration of relatively unvisited children too

**Machine Learning: Introduction and Unsupervised Learning**

Supervised Learning- teacher gives a set of (x,y) pairs, training examples have known outcomes

* K-Nearest-Neighbors- save each training example as a point in Feature Space, classify a new example by giving it the same classification as its nearest neighbor in Feature Space

Inductive Bias- learning can be viewed as searching the Hypothesis Space H of possible h funciotns, Inductive Bias: used when one h is chosen over another, needed to generalize beyond the specific training examples, Completely unbiased inductive algorithm: only memorizes training examples, can’t predict anything about unseen examples

* Decision Tree- each non-leaf node has associated with it an attribute/feature, each lead node has associated with it an attribute/feature, each arc has associated with it one of the possible values of the attribute of its parent node

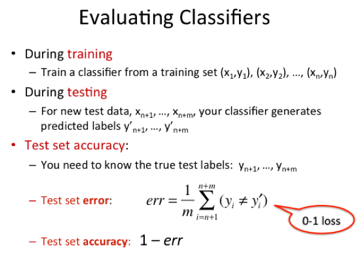
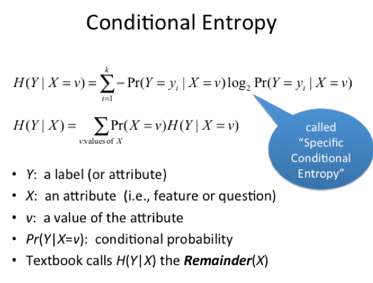
Ockham’s Razor- a possible best decision tree where the simplest hypothesis is one that is consistent with all observations and smallest decision tree that correctly classifies all of the training examples is best

Random- an attribute chosen at random

Least-Values- attribute with smallest number of possible values

Most-Values- attribute with largest number of possible values

Max-Gain- attribute with largest expected information gain

**

Overfitting- means finding “meaningless” regularity in data, fix by pruning some nodes in the decision tree or enforce a minimum number of examples at each leaf node

Tuning Set- used to prune, randomly split the training data into train and tune, build tree using only train set and prune tree using tune set

Test Set Method­- randomly choose some data to be the Test set and remaining to be Training set, build decision tree using Training set and estimate future performance using Test set

Experimental Evaluation of Performance- cross-validation, leave-one-out cross validation

Combining Classifiers: Ensemble Methods- aggregate the predictions of multiple classifiers with the goal of improving accuracy, classify it using each classifier and report as the output class the majority (for a 2-class problem) or mode classification (for k-class problems), ensemble is better when each individual classifier is accurate and diverse

Ensemble Strategies- boosting: sequential production of classifiers, where each classifier is dependent on the previous one, bagging: create classifiers using different training sets, where each training set is created by “bootstrapping”

Unsupervised Learning- only the x’s are given, unknown outcomes

Clustering­- group training sample into clusters such that examples in the same cluster are similar and examples in different clusters are different

Random Forests- uses bagging and randomized node optimization, no tree pruning and doesn’t overfit

* Hierarchical Agglomerative Clustering- build a binary tree over the data set by repeatedly merging clusters, the binary tree you get is often called a dendogram, or taxonomy, or a hierarchy of data points

Single-linkage- the shortest distance from any member of one cluster to any member of the other cluster

Complete-linkage- the largest distance from any member of one cluster to any member of the other cluster

Average-linkage- the average distance between all pairs of members, on from each cluster

* K-Means Clustering- specify the desired number of cluster and use an iterative algorithm to find them, will always terminate, not always optimal

Distortion/Inertia- sum of squared distance from each point to its cluster, smaller value corresponds to tighter clusters