2.uniformed search ("uniformed " means only have successors() function But not which non-goal states)

- Problem solving as search : A search problem has five components: S, I, G, actions, cost)

The process to look for sequence of actions to reach the goal

- problem representation in terms of states
  - Fully observable assumption: no missing information
  - -- State Space (aka Problem Space) = all possible valid configurations of the environment
- state-space graph search formulation: (Formalizing Search in a State Space)
  - Frontier = {S}, where S is the start node
  - Loop do
    - if Frontier is empty then return failure
    - pick a node, n, from Frontier
    - if n is a goal node then return solution
    - Generate all n's successor nodes and add them all to Frontier
    - Remove n from Frontier
- closed world assumption:
  - All necessary information about a problem domain is accessible so that each state is a complete description of the world;
- expanding a node: a)generate successor b) add them and their arcs to the state space search tree
- frontier/open list: The generated, but not yet expanded, states define the Frontier (aka Open or Fringe) set
  - Different Search Strategies: The essential difference is, which state in the Frontier to expand next)
- partial solution path: a partial solution path from the start node to the given nodee
- chronological backtracking: when search hits a dead end, backs up one level at a time when search hits a dead end,

– problematic if the mistake occurs because of a bad action choice near the top of search tree

- detecting repeated states,(If State Space is Not a Tree):

When pick a node from Frontier – Remove it from Frontier – Add it to Explored – Expand node, generating all successors

4.local search: operate using a single **current node** and generally move only to neighbors of that node. greedy local search every node is a solution

- escaping local optimal

Fix by replacing fixed probability, p, that a bad move is accepted, with a probability that decreases as the search proceeds

- Boltzman's equation
  - Let ΔE = f(newNode) – f(currentNode) < 0   p = e ^(ΔE / T) (Boltzman's equation*
- cooling schedule
  - --T, the annealing "temperature," is the parameter that control the probability of taking of bad steps
  - --We gradually reduce the temperature, T(k)
  - --At each temperature, the search is allowed to proceed for a certain number of steps, L(k)
  - --The choice of parameters {T(k),L((k)} is called the cooling schedule

Complete : a complete algorithm will find a solution)

Optimal/admissible : an admissible algorithm will find a solution with minimum cost)

Time complexity: measured for worst case)

Space complexity: maximum size of frontier)

7.supervised learning

- K-nearest neighbor algorithm,
- Decision Trees
- Ockham's razor, (the Preference Bias of best decision tree is Ockham's razor:)
  - -- The simplest hypothesis that is consistent with all observations is most likely
  - – The smallest decision tree that correctly classifies all of the training examples is best
- decision tree algorithm,
- information gain, (Goal: Select the attribute that will result in the smallest expected tree size)
  - I(Y; X) = H(Y) - H(Y I X)
- max-gain,: the attribute that has the largest expected information gain
- entropy,
- conditional entropy,
- remainder, H(YIX) = Remainder(X)
- overfitting problem: As d increases, the(Mean Squared Error) on the training data improves, but prediction on test data worsens
  - – meaningless regularity is found in the data
  - – irrelevant attributes confound the true, important, distinguishing features
  - – fix by pruning some nodes in the decision tree
- Pruning
  - Randomly split the training data into TRAIN and TUNE, say 70% and 30% 2.
  - Build a full tree using only the TRAIN set 3.
  - Prune the tree using the TUNE set,
- setting parameters,
  - – most learning algorithms require setting various parameters – they must be set without looking at the Test data
  - – Common approach: use a Tuning Set
  - 1. Partition given examples into TRAIN, TUNE and TEST sets
  - 2. For each candidate parameter value, generate a decision tree using the TRAIN set
  - 3. Use the TUNE set to evaluate error rates and determine which parameter value is best
  - 4. Compute the final decision tree using the selected parameter values and both TRAIN and TUNE sets
  - 5. Use TEST to compute performance accuracy

3.informed search(use domain knowledge to guide selection of the best path to continue searching)

- Heuristic function: estimated cost of the cheapest path from the state at node *n* to a goal state. h(n)
- Evaluation function: cost estimate,
- admissible heuristic: *never overestimates* the cost to reach the goal.
- consistent heuristic, h(n) ≤ c(n, n') + h(n')
- devising heuristics: relaxing the problem; If optimality is not required, a satisficing solution is okay, (fewer expand nodes

Goal of the heuristic is then to get as close as possible, either under or over, to the actual cost)

- Heuristics are often defined by relaxing the problem,
- If h(n) = h*(n) for all n,
  - – only nodes on optimal solution path are expanded
  - – no unnecessary work is performed
- If h(n) = 0 for all n,
  - – the heuristic is admissible
  - – A* performs exactly as Uniform-Cost Search (UCS)
- If h1(n) ≤ h2(n) ≤ h*(n) for all n, then h2 dominates h1
  - – h2 is a better heuristic than h1
  - – A* using h1 (i.e., A1*) expands at least as many if not more nodes than using A* with h2

5. game playing (should consider opponent)

- Zero-sum games: one player's gain is the other player's loss. Does not mean fair
- perfect information games: each player can see the complete game state. No simultaneous decisions
- deterministic vs. stochastic games

deterministic; fully observable environments where two agents act alternately and at the end of the game are always equal and opposite

Stochastic: including a random element

- game playing as search : . (represent both computer and opponent moves)

Initial state; player; action; result; terminal state;

- ply: in minimal , consisting of two half-moves, each of which is called a **ply**
- minimax principle,(cannot be used to end of the game)
  - • high values favor the computer -- The computer assumes after it moves the opponent will choose the minimizing move
  - • low values favor the opponent -- The computer chooses the best move considering both its move and the opponent's optimal move
- static evaluation function: use heuristics to estimate the value of non-terminal states (agree with the Utility function when calculated at terminal nodes)
- alphabeta pruning, (Pruning can be used to ignore some branches)
- iterative-deepening with alpha-beta (for Dealing with Limited Time- > time can also be saved by limited depth)
  - – run alpha-beta search with depth-first search and an increasing depth-limit
  - – when time runs out, use the solution found for the last completed alpha-beta search
  - – "anytime algorithm"
- horizon effect :The computer has a limited horizon, it cannot see that this significant event could happen
- quiescence search —> used for avoid "short sightedness"(can also be solved by secondary search)
  - – when SBE value is frequently changing, look deeper than the depth-limit
  - – look for point when game "quiets down"
  - – E.g., always expand any forced sequences
- representing non-deterministic games(games involve chance,)
- chance nodes in expectimax algorithm: representing random events

6.unsupervied learning

- Inductive learning problem,: Generalize from a given set of (training) examples, make predictions for future
- unsupervised learning problem, :the agent learns patterns in the input even though no explicit feedback is supplied
- Feature: Each dimension in x is called a feature or attribute. --- x is a point in the D-dimensional feature space
- Labels: the desired prediction for an instance x
- classes,: Discrete labels:
- classification problems,: if y discrete
- inductive bias, :
  - Inductive Bias – is used when one h is chosen over another – is needed to generalize beyond the specific training examples
- Completely unbiased inductive algorithm – only memorizes training examples – can't predict anything about unseen examp
- preference bias: define a metric for comparing h's so as to determine whether one is better than another
- hierarchical agglomerative clustering algorithm, : Build a binary tree over the dataset by repeatedly merging clusters
- single linkage, / complete linkage, /average linkage,
  - shortest distance /largest distance/average distance between all pairs of members, one from each cluster
- dendrogram,: The binary tree get in hierarchical agglomerative clustering is often called a dendrogram, or taxonomy, or a hierarchy of data points
- kmeans clustering algorithm, :Specify the desired number of clusters and use an iterative algorithm to find them

Stop until cluster centers no longer change. Will terminate! Not optimal

- cluster center,:Choose ci to be the mean / centroid of all points/examples in the cluster
- distortion cluster quality. (Distortion = Sum of squared distances of each data point to its cluster center)

	description	Complete	Optimal/	Time complexity	Space complexit	nodes
BFS						
DFS						
uniform cost search),Dijkstra's	Priority Queue g(n) = cost of path					
IDS	do DFS to depth 1 and treat all children of the start node as leaves – if no solution found, do DFS to depth 2					
bidirectional search	bfs from both start and goal --Stop when Frontiers meet				O(b^(d/2))	
Greedy bfs	Use as an evaluation function, f(n) = h(n), sorting nodes in the Frontier	no	no			
Beam search	Use an evaluation function f(n) = h(n) as in Greedy bfs, and restrict the maximum size of the Frontier to a constant, k	no	no			
Best-first search	Sort nodes in the Frontier list by increasing values of an evaluation function	No	No		b^m	b^m
Algorithm a	Use as an evaluation function f(n) = g(n) + h(n),	no				
A *	all nodes n in the search space, h(n) ≤ h*(n) should terminate only when a goal is removed from the priority queue	yes	yes		O(NO.states)	
Hill climbing	1.Pick initial state s 2. Pick t in neighbors(s) with the largest f(t) 3. if f(t) ≤ f(s) then stop and return s					
hill-climbing with random restarts	When stuck, pick a random new starting state and re-run hill-climbing from there					
stochastic hill-climbing (less greedy)	1.Pick initial state, s 2. Randomly pick state t from neighbors of s 3. if f(t) better than f(s) then s = t // t better than s so move there else with small probability s = t 4. Goto Step 2 until some stopping criterion is met					
minimax algorithm				O(b^d)	O(bd)	
alpha-beta pruning	Worst Case: – ordered so that no pruning takes place Best Case: – each player's best move is visited first			In practice often get O(b^(d/2))		
Expectimax	For chance nodes: compute the average, weighted by the probabilities of each child					

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

	Deterministic	Stochastic (cha nce)
Fully observable	Checkers, Chess, Go, OtheUo	Backgammon. Monopoly
Partially observable	stratego, battleship	Bridge, Poker, scrabble

$$\underbrace{\frac{w_i}{n_i}}_{\text{Exploitation term}} + c\underbrace{\sqrt{\frac{\ln t}{n_i}}}_{\text{Exploration term}}$$

BFS: if all operators (i.e., arcs) have the same constant cost, or costs are positive, non-decreasing with depth – otherwise, not optimal but does guarantee finding solution of shortest length (i.e., fewest arcs)
DFS: Can find long solutions quickly if lucky) (May not terminate without a depth bound)
IDS: ptimality as stated for BFS . I Trades a little time for a huge reduction in space; "Anytime" algorithm: good for response-time critical applications
Has advantages of BFS – completeness – optimality as stated for BFS
Has advantages of DFS – limited space – in practice, even with redundant effort it still finds longer paths more quickly than BFS
stochastic hill-climbing: bad move: when probability that decreases as the search progress (T decreases) and with the badness of move (ΔE worsens)

A*: the tree-search version of A* is optimal if h(n) is admissible, while the graph-search version is optimal if h(n) is consistent. (big search space —> iterative deepening A*
Simulated annealing: The probability decreases : 1) with the "badness" of the move—the amount ΔE by which the evaluation is worsened. 2)"temperature" T goes down: "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases. If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1

Hill climbing: Solution found by HC is totally determined by the starting point
When the neighbor is large: • Randomly generate neighbors, one at a time • If neighbor is better, take the move
Simulated Annealing: fast, increase chance to find global optimum
As ΔE -> -∞, p-> 0 i.e., as move gets worse, probability of taking it decreases exponentially •
As T -> 0, p ->0 i.e., as "temperature," T, decreases probability of taking bad move decreases
l ΔE << T if badness of move is small compared to T, move is likely to be accepted l
ΔE >> T if badness of move is large compared to T, move is unlikely to be accepted
Tips:

- starting temp must be high to escape local optimal
- With an infinitely slow cooling rate, SA finds the global optimum with probability 1

Effectiveness of Alpha-Beta Search: depends on the order in which successors are exam
Worst Case: ordered so that no pruning takes place
Best Case: – each player's best move is visited first
Non-Deterministic Games: increases branching factor;
Value of look-ahead diminishes: less effective for alpha-beta pruning'
Improve performance in game playing: reduce depth(better sbe, learn good features); reduce breath (explore subset of possible moves use randomized exploration)

Monte Carlo tree search(Best-first search based on random sampling of search space; focus on good moves
simulate k random games by selecting moves at random for both players until game over (called playouts); count how many were wins out of each k playouts; move with most wins is selected (infinite moves and infinite length)
Recursively build search tree, where each round consists of:
1. Selection: Starting at root, successively select best child nodes using scoring method until leaf node L reached
2. Expansion: Create and add best (or random) new child node, C, of L
3. Simulation: Perform a (random) playout from C
4. Backpropagation: Update score at C and all of C's ancestors in search tree based on playout results

Smarter Initialization of K-Means Clusters:
1.Run k-Means multiple times with different starting, random cluster centers
2.**pick a random point x1 from the dataset 1. Find a point x2 far from x1 in the dataset
Knn: pick k
Split data into training and tuning sets; Classify tuning set with different values of k; Pick the k that produces the smallest tuning-set error
K fold cross evaluation:
1.Divide all examples into K disjoint subsets E = E1, E2, ..., EK
2. For each i = 1, ..., K : let TEST set = Ei and TRAIN set = E - Ei ; build decision tree using TRAIN set ; determine accuracy Acci using TEST set
3. Compute K-fold cross-validation estimate of performance = mean accuracy
leave_one out cross evaluation:(use when have small dataset)
For i = 1 to N do // N = number of examples
1.Let (xi , yi) be the i th example
2.Remove (xi , yi) from the dataset
3. Train on the remaining N-1 examples
4. Compute accuracy on i th example (Accuracy = mean accuracy on all N runs)
Random forest: collection of independently-trained binary decision trees
Hard to interpret compared to decision tree, but efficient and can handle large data
1.For each tree, pick a randomly sampled subset of training data
2. Randomly choose a subset of features and thresholds at each node
3. Pick the feature and threshold that give the largest information gain
4. Recurse until a certain accuracy is achieved or depth-bound reached
Ensemble strategies
1. Boosting
2. Bagging
Given N training examples, generate separate training sets by choosing n examples with replacement from all N examples • Called "taking a bootstrap sample" or "randomizing the training set" • Construct a separate classifier using the n examples in each training set