AI - Last Min Notes

Owner	nithu
Tags	

Unit 1 - AGENTS AND BASIC SEARCH TECHNIQUES

AGENTS

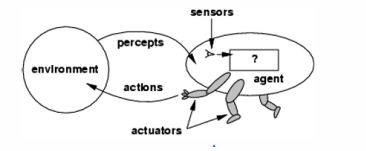
A rational agent → act to achieve the best outcome, when there is uncertainity.

Should learn from percepts and exploration

Autonomous

Agent- perceives env. thro sensors and acts thro actuators

Eg: Human Body



Consequentialism - evaluate an agent's behaviour thro consequences.

Perf. measure - criteria for success of an agent

P- Performance Measure

E- environment

A- Actuators

S-Sensors

PROPERTIES OF ENVIRONMENT

Fully observable vs partially observable(Chess vs Kriegspiel)

Single vs Multi

Deterministic vs non-deterministic(stochastic) - determine the next state with the current state vs multiple unpredictable outcomes

Episodic vs sequential - action taken once is dependent on previous

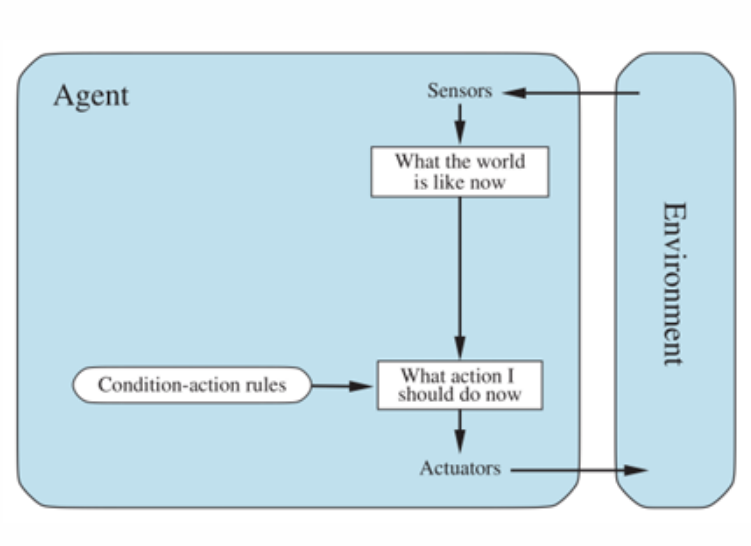
Static vs Dynamic - changing the env.

Discrete vs cont: time is discrete or cont.

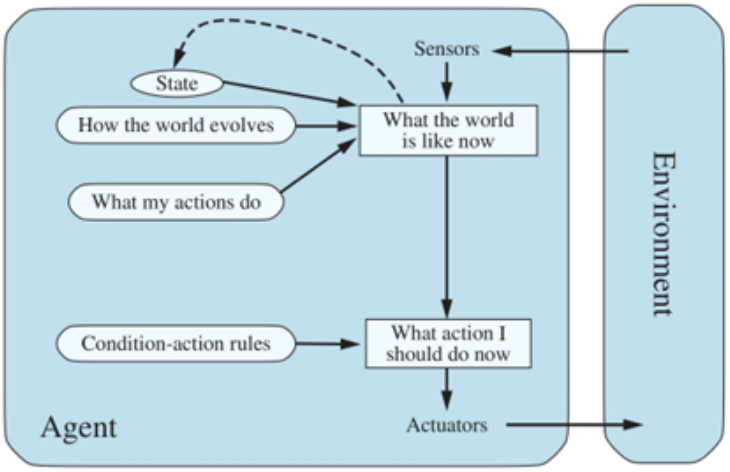
Known or unknown

TYPES OF AGENTS:

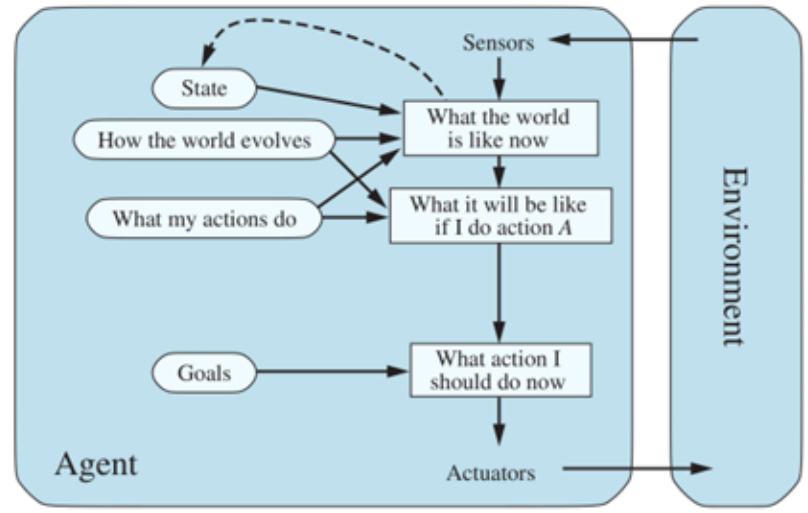
Simple reflex(Do not have mem. of past)⟶ completely dependent on present.
Condition-action rules



Model based reflex(Internal state ⟶ past states, work on change)



Goal based agents/Problem solving agents(Desirable goal to be achieved → future events and results)



Steps for goal-based agents:

1. Goal Formulation: Set of one/more world states

2.Prob.Formulation: What to consider and what not to consider

3.Search for a solution - a sequence of actions from the initial to the goal state.

4.Execute

→ The process of looking for the best soln → search

→ the best seq(list of actions) → soln

TYPES OF PROBLEMS

1. Deterministic,fully observable

2.Non-observable

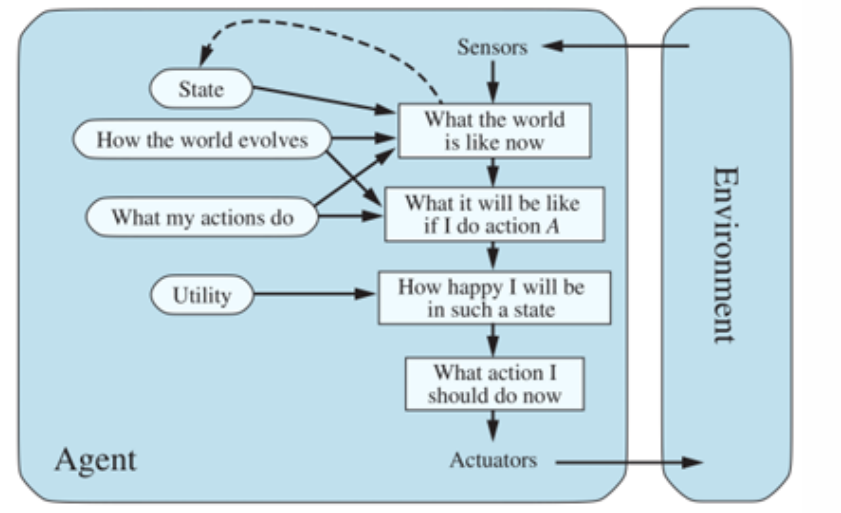
3.Non-Dete. and partially observable

4.Unknown state space

1. ***Initial state***
2. ***Actions***
3. ***Transition model or (Successor functions)***
4. ***Goal Test.***
5. ***Path Cost.***

Optimal soln → lowest path cost among the lot

Utility based agents(Alternatives → indicating a measure of success or happiness)



Learning agents(Reinforcement based)

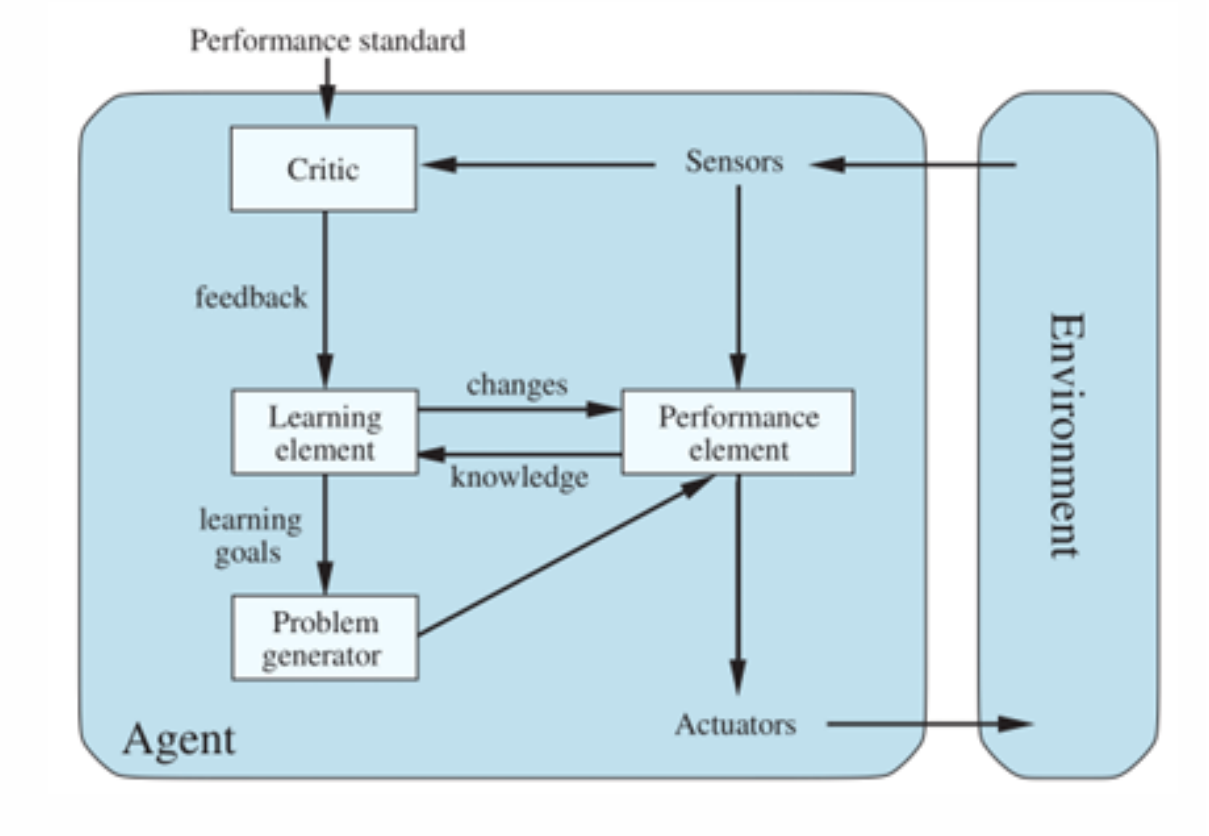


Table-lookup driven agent —: implements a lookup table for actions. Too large → 4^K possible seq.

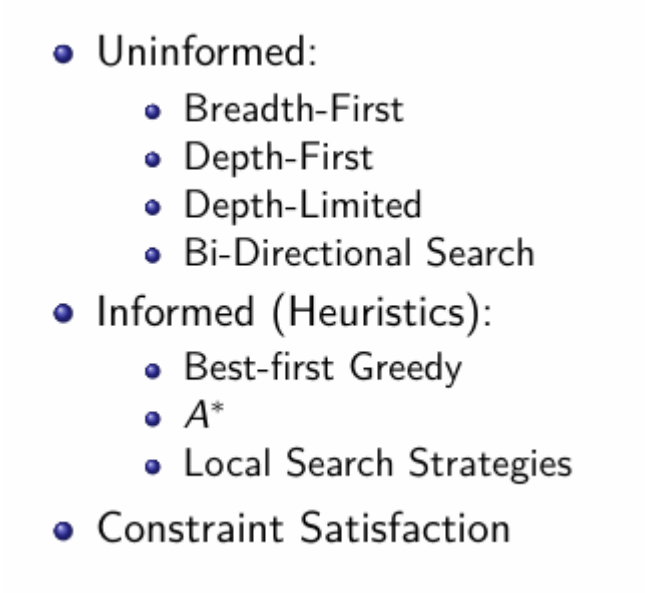
State-Space Approach: Visualize the search tree based on an initial state, set of actions

B.F → no of possible actions

State Space → all that is reachable from the initial state

State-space components: Initial state, Successor function, Goal test and Path cost

8-puzzle prob(jumbled up numbers from 1-8→ arrange it ascendingly)



Metrics of a problem:

Completeness

Time Complexity

Space Complexity

Optimality

UNINFORMED SEARCH TECH.

Tree-Search -generate successors of the explored states

Graph-search - explored and frontier(generated but not explored)

States (physical config.)vs Nodes (state, parent node, action,path cost,depth)

BFS →FIFO Queue

Complete: Yes(if B.F is finite)

Time Complexity: O(b^(d+1))
Space Complexity:O(b^(d+1))

Optimality: No

Uniform cost search → Start with the lowest cost fringe

Complete: Yes(if no negative cost)

Time Complexity: O(b^(C*/e))
Space Complexity:O(b^(C*/e))

Optimality: Yes. Monotonic cost fn.

[Use I.D idea within Uniform cost search - Iterative Lengthening Approach]

Depth-First Search → LIFO Stack

Complete: No

Time Complexity: O(b^(m))
Space Complexity:O(b(m))

Optimality: No

Bactracking → DFS version → generate one successor

Complete: No

Time Complexity: O(b^(m))
Space Complexity: O(m)

Optimality: No

What we need now?

Something with less time complexity as DFS and complete as BFS →

Depth-Limited Search : Impose a limit l on the search

l>d but not large(defeats the purpose)

Complete: No! Depends on L

Optimal: No!

Time Complexity=O(b^l)
Space Complexity: O(bl)

Iterative Deepening: Dynamically find a suitable depth-limit l and increment until goal is reached

Complete: Yes! - BFS Property

Optimal: No! - BFS Property

Time Complexity=O(b^d) - DFS Property
Space Complexity: O(bd) - DFS Property

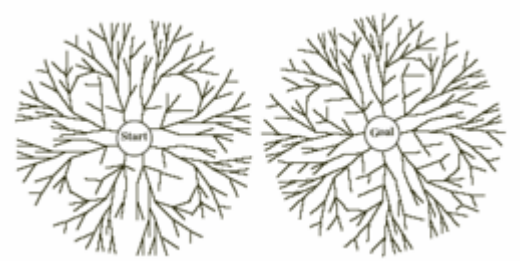
Bidirectional search: Run two search simultaneously -one forward and another backward from the goal - make them meet in middle

Time Complexity=O(b^d/2) - half the depth

Space Complexity =O(b^d/2)

Goal test → 'intersection test'

Backward search → define predecessors



UNIT 2 - HEURISTICS, INFORMED SEARCH AND LOCAL SEARCH

HEURISTICS - A* search/RBFS/BEST-FIRST - INFORMED SEARCH

BEST-first GREEDY ALGORITHM:

evaluation function=heuristics

Complete: No

Optimal:No

Time Complexity:O(b^m)

Space Complexity: O(b^m)

What if we design an evaluation fn - to determine goodness or badness of a state. → to choose the best of the lot

→ Need of heuristics h(s) → estimated cost of reaching a goal state

An admissible heuristic → never overestimates the cost → optimistic

A heuristic h is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a, means it is non-decreasing along a path. f is monotonic*

DOMINANT HEURISITCS: If h2(n) ≥ h1(n) then h2 is said to be dominant, which makes it a better heuristic for search.

This helps in reducing the search-space by suitable generating successors, with a good branching factor.

How to get heuristics?

- Relaxed problems:
- Subproblems: like in 4-queens problems. Use the substeps to minimize the heuristic value
- Landmarks: Optimal paths between two nodes using precomputation. Use landmark points among the vertices.

Differential heuristic:

$$h_{DH}(n) = \max_{L \in Landmarks} |C^*(n, L) - C^*(goal, L)|$$

A* search:

- Optimal
- Complete
- Time complexity: O(b^ed)
- Optimally efficient(the fewest nodes are expanded)
- Space Complexity → worst case : BFS=O(b^d)
- If admissible heuristic - use A* tree-search
- If consistent heuristic - use A* graph-search

BEAM SEARCH : Limiting the frontier

Have only the k-best nodes with f-scores and discard the others or have a limit for the f-scores

ITERATIVE - DEEPENING A* → f-cost is a cutoff(like iterative lengthening)—>path-finding

RBFS : Dynamically adjusting cost-limit for a path

Track the best alternative path , when the best child exceeds, unwind

Better than IDA* , but still suffers from node regeneration.

SMA(Simplified Memory-bounded A*) → Memory use to the peak

Proceed like A* until mem is exhausted

Throw away the node with the largest f-val to get new space.

Remember the best forgotten ones in the parent

Same f-cost is an issue here

LOCAL SEARCH TECHNIQUES

Our goal state is our final aim- so sometimes it is better to move on - iterating upon optimal config. satisfying a few set conditions rather than discussing your path.

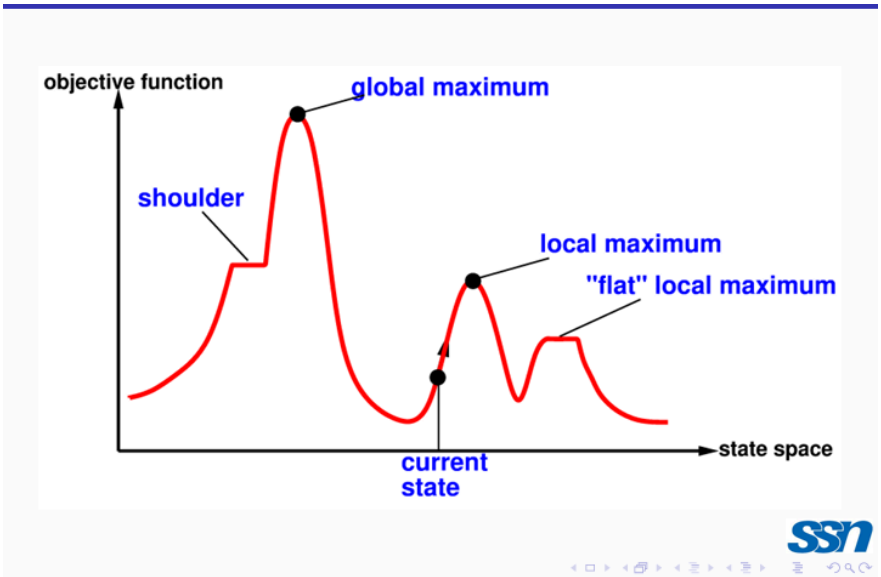
Eg: n-queens

1. Start with a state(any random placement of the n-queens)
2. Place one queen per column
3. Calculate and recalculate your heuristic
4. Iterate until the val comes at a minimum.

→ Iterative Improvement Algos.

→ Basically improve upon what you have in your hand

Hill climbing:



Problems:

1. Local Maxima/Minima - won't move further even if a better solution is available(kind of like instant gratification)
2. Ridges - Sequence of local maxima - difficult to navigate
3. Plateau - a very flat sequence or a shoulder

How to rectify?

Random sideways moves - you could escape a shoulder , but gets stuck on the maxima.

Stochastic Hill Climbing: Choose random out of all possibilities(prob. depends on steepness)

First Choice Hill Climbing: randomly generate your successors, until the immediate better option is generated

Random Restarts: Expect 1/p restarts for p prob. of success

Stimulated Annealing: A variation of hill climbing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE(current) – VALUE(next)
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(−ΔE/T)
```

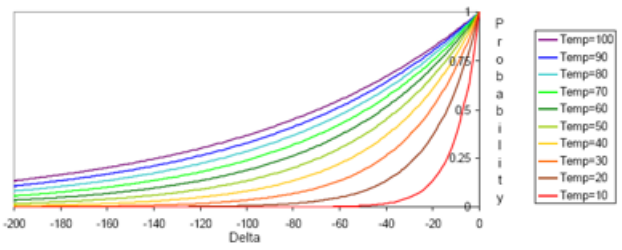
Start with a case where the prob. of a bad state is high and iterate to slowly reduce it.

Tries to explore enough early on, makes certain downhill moves for finding a good way uphill.

The prob. of moving to a higher energy sate

p = e^(-del E/kT)

if (delta>=0) accept
else if (*random* < $e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)
Temp slowly goes to 0, with multiple moves attempted at each temperature
Final runs with temp=0 (always reject bad moves) greedily "quench" the system

LOCAL BEAM SEARCH: Generate k initial states terminate if goal state else generate all successors and select best k

GENETIC ALGORITHM: state with a random pop. Produce a new gen by choosing crossover pts and mutation

Schema : a substring where some positions are left undefined.

- Given: population P and fitness-function f
- repeat
  - newP ← empty set
  - for *i* = 1 to size(P)
    x ← RandomSelection(P,f)
    y ← RandomSelection(P,f)
    child ← Reproduce(*x*,*y*)
    if (small random probability) then child ← Mutate(child)
    add *child* to newP
  - P ← newP

NON-DETERMINISM, PARTIALLY OBSERVABLE AND CONTINUOUS SPACE

In case of continuous state spaces,

Choose hill climbing or stimulated annealing because all the successors need not be generated.

The evaluation function $f(s)$ may be easily computed as:

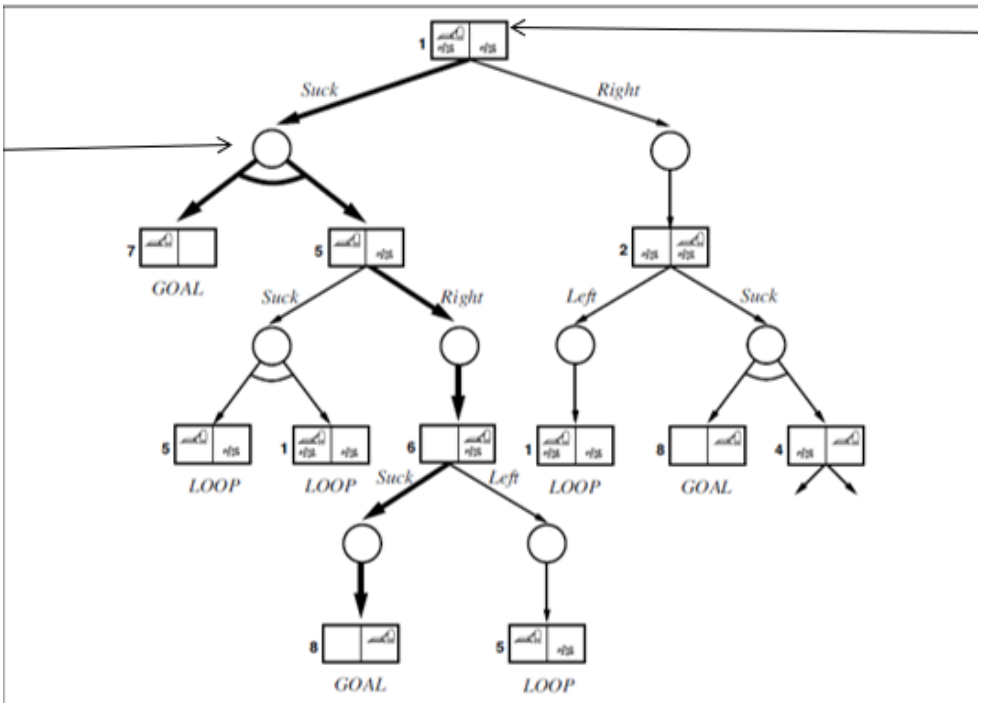
$$f\left((x_1, y_1), (x_2, y_2), (x_3, y_3)\right) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

Use empirical gradient /gradient descent to find the max/min in a landscape. Gives the steepest slope.

or take up Newton-Raphson method

The results of an action can vary in such a case . For example, suppose a vacuum cleaner cleans two squares of dirt, or deposits dirt while cleaning

Generalize the solution to a contingency plan. AND-OR TREE



In case of an partially observable environment(for eg, Kriegspiel) → the agent does not always know its state, instead goes by a belief state

ONLINE SEARCH AGENTS:
Executes actions

Acquires percepts

Deterministic and fully observable

Has to act to know the outcome

Eg: Autonomous vehicle and Search engines

```
function ONLINE-DFS-AGENT(problem, s') returns an action
        s, a, the previous state and action, initially null
  persistent: result, a table mapping (s, a) to s', initially empty
        untried, a table mapping s to a list of untried actions
        unbacktracked, a table mapping s to a list of states never backtracked to

  if problem.IS-GOAL(s') then return stop
  if s' is a new state (not in untried) then untried[s'] ← problem.ACTIONS(s')
  if s is not null then
      result[s, a] ← s'
      add s to the front of unbacktracked[s']
  if untried[s'] is empty then
      if unbacktracked[s'] is empty then return stop
      else a ← an action b such that result[s', b] = POP(unbacktracked[s'])
  else a ← POP(untried[s'])
  s ← s'
  return a
```

UNIT -3 GAME THEORY - ALPHA-BETA PRUNING, MCTS, CSP

All the search techniques that we have discussed so far , will seem fruitless for a actual game

GAMES:

Perfect info+deterministic ⟶ Chess

Imperfect info + deterministic ⟶ Kriegspiel

Perfect info + non-deterministic ⟶ Backgammon

Imperfect info + non-deterministic ⟶ Poker, scrabble

ASSUMPTIONS:

- Fully observable
- Adversarial
- Agents whose actions are alternate

For eg: 35^100 possibilities for a move(non-trivial games)

Search ⟶ Path(not adversarial)

Game ⟶ a strategy(adversarial ⟶ your opp. plans as you do)

ZERO-SUM GAMES: two outcomes , one maximum/positive(+1) and one negative/minimum(-1) ⟶ Tic-tac toe

Reward+punishment(penalty) is always a constant

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s
- ACTIONS(s): set of legal moves in a state s
- RESULT(s,a): the transition model that defines the result of a move a in a state s
- IS_TERMINAL(s): Terminal test — 'true' when the game is over — terminal states
- UTILITY(s,p): Utility function — an objective function that defines the final numeric value to a player p when the game ends in a terminal state s — in chess, outcome is a win, loss, or draw, with values $+1, 0, \frac{1}{2}$

MINIMAX TREE(Adversarial Search) + PRUNING

MINIMAX Tree⟶ alternate MIN and MAX (choose lowest utility for MIN and highest utility for MAX)

Heuristic Evaluation Fn: How ‘good’ a config. is for a player

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Complete(if the tree is finite)

Optimal - if the opponent is optimal(he/she takes the best move)

Space Complexity: O(bm)

Time Complexity:O(b^m)

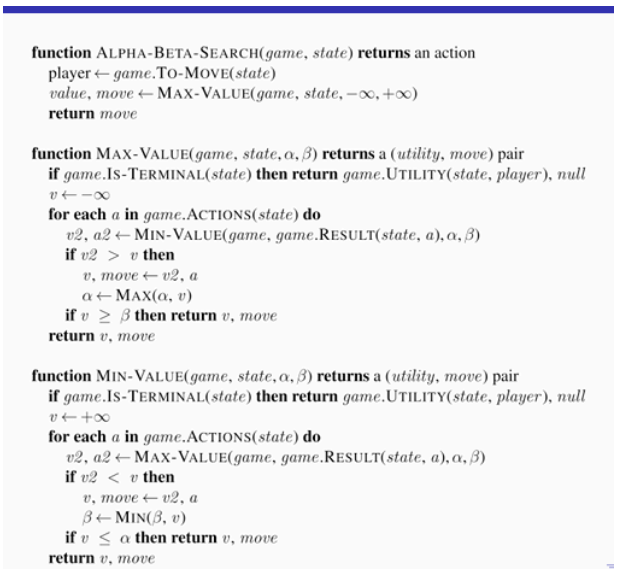
PRUNING: Discard based on a given heuristic.

ALPHA-BETA PRUNING:

- No use expanding bad nodes
- If the opp forces a bad node, then again its useless

ALPHA(Best choice for MAX) ⟶ lower bound for MAX
BETA(best choice for MIN)⟶ upper bound for MAX

PRUNING CONDITION: Alpha≥Beta



```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Implementing this retains your completeness ,optimality and space complexity

Time complexity ⟶ can be halved O(b^ m/2) ⟶ O(root(b)^m)

Eg: In chess ⟶ B.F goes from 35 to root(35) ⟶ approx 6

Worst-case ⟶ ordered branches ⟶no pruning

Best Case ⟶ the left-most child is the best move(hence it is considered first)

HOW TO MAKE THIS EFFECTIVE?

Dynamic move ordering — moves found to be best in the past
(learning) — or, iterative-deepening

Transposition table — hash the α–β values for future use

Sometimes, this may also not be useful ⟶ Cases like Chess and Go

TYPE A ⟶ heuristic evaluation in a certain depth and estimate utilities

TYPE B ⟶ ignore bad moves

Iterative Deepening Search(IDS)

- **In practice, iterative deepening search (IDS) is used**

–IDS runs depth-first search with an increasing depth-limit

–when the clock runs out we use the solution found at the previous depth limit

NON DETERMINISM IN GAMES - MCTS

Basically - u could have multiple next states and work upon belief states or possible states

Monte Carlo Tree Search:

Four basic principles:

- Selection

  Leaf node is a node which has unexplored child nodes.

$$UCB(node_i) = \overline{x}_i + c\sqrt{\frac{\log N}{n_i}}$$

  U(n)/C(n) , n ⟶node value , N⟶parent value of the node

  As no of turns go up, the second term becomes lesser⟶ least likely to be selected

  Built-in exploration and exploitation

- Expansion

  Randomly pick an unexplored node

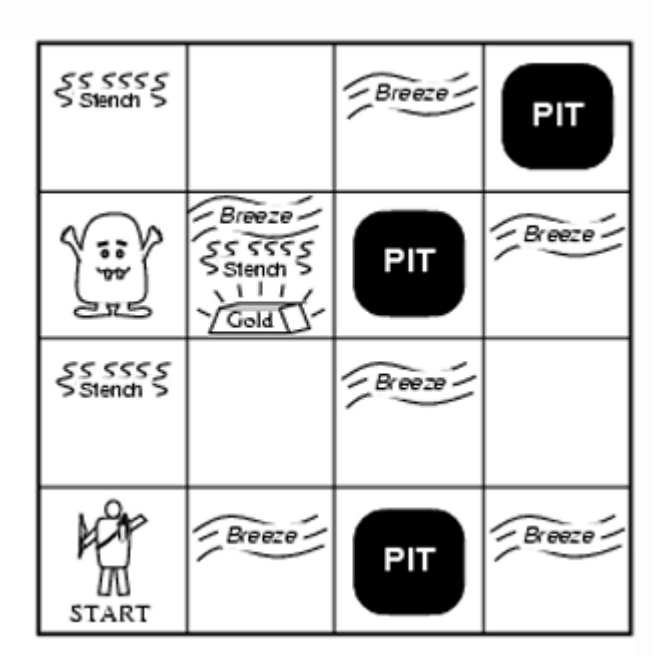
- Simulation

  Rollout ⟶ +1/0/-1

- Backpropagation

This can be considered as a Reinforcement Strategy ⟶ using Type B strategy

WUMPUS WORLD



Breeze near Pit

Stench near Wumpus

Gold la Glitter

- Partially Observable
- Determinstic
- Episodic(No)
- Static
- Discrete
- Single Agent

SAT → to check for satisfiability → exists an assignment of truth values where the proposition holds true(NP-complete)

CSP

How does it differ from Local Search/Adversarial techniques?

→ Rather than consider a node is a goal state alone, we deep dive into the structure of the nodes and exploit that

A solution to the CSP → assignment of values to the variable such that all constraints are satisfied.Inconsistent, if nothing as such exists.

- A *state* is an *assignment* of values to some or all variables.

–An assignment is *complete* when every variable has a value.

An assignment is *partial* when some variables have no values

For example: For n-queens , the constraints are , Qi≠Qj and |Qi-Qj|≠|i-j|

Variables: The regions

CSP - For graph coloring

Domain: The colors

Constraint: Adj.regions must have diff. colors

Types of Constraints:

Unary: X=red

Binary: X=Y

Higher order constraint: 3 or more variables

Preferences: Soft constraints

CSP ⟶ Can be expressed as a search problem

CSPs are commutative(the order doesnt matter)

DFS in CSP ⟶ backtracks

implement a constraint, backtracks when a variables has no legal values to assign

Algo:

SELECT UNASSIGNED VARIABLE

Min.Rem Values⟶most constrained variables and most likely to fail soon

If a tie, choose the var involved in the most constraints

ORDER-DOMAIN-VALUES
Least Constraining Value
Rules out the fewest choices for the variables it is in constraints with

FORWARD CHECKING: Take a constraint, change the variables ⟶ move on similarly

Sometimes, this can ignore inconsistencies

Arc Consistency

Two things are joined by an arc,all the variables in the two entities are consistent

💼 REVISE CSP

UNIT 4 - PREDICATE LOGIC

Horn Clause ⟶ At most positive literals

Definite clause ⟶ Exactly one positive literal

Goal clause⟶ No positive literals

Forward chaining ⟶ Bottom to top

Backward chaining ⟶ Top to bottom

TABLE 1 Rules of Inference.

Rule of Inference	*Tautology*	*Name*
p $p \rightarrow q$ $\therefore q$	$(p \wedge (p \rightarrow q)) \rightarrow q$	Modus ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	$(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$	Modus tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$	Resolution

---

- Negation or De Morgan's Law (we saw this last time):
  - $\neg\forall x\, P(x) \equiv \exists x\, \neg P(x)$
  - $\neg\exists x\, P(x) \equiv \forall x\, \neg P(x)$
- Distributivity:
  - $\forall x\, (P(x) \wedge Q(x)) \equiv \forall x\, P(x) \wedge \forall x\, Q(x)$
  - $\exists x\, (P(x) \vee Q(x)) \equiv \exists x\, P(x) \vee \exists x\, Q(x)$
  - Can't distribute universal quantifier over disjunciton or

UNIT 5 - UNCERTAINITY,BN

Uncertainity ⟶ partial observability, noisy sensors, uncertainity in actions and immense complexity in modelling

PROBABILITY: Model agent's degree of belief

Relates to agent's own state of knowledge , with obviously change with new evidence.

Utility Theory ⟶ represent preferences

Decision Theory = probability theory+utility theory

Atomic event: A complete specification of the state of the world.

- Mutually exclusive
- the set of all possible atomic events is exhaustive
- Entails the truth or falsehood of every proposition
- Logically equivalent to the disjunction of all atomic events

Prior probability: Or conditional probability

$$P(a \mid b) = P(a \wedge b) / P(b) \text{ if } P(b) > 0$$

$$P(a \wedge b) = P(a \mid b)\, P(b) = P(b \mid a)\, P(a)$$

Joint prob.dist: Prob. of Every atomic event of the random variables

Kolmogorov's Axioms:

$0 \le P(A) \le 1$

P(true)=1 and P(false)=0

P(AVB)=P(A)+P(B)-P(A^B)

P(~A)=1-P(A)

Chain rule = Product rule applied multiple times

Marginal Probability/Conditional Probability= P(Y)=P(Y/z)*P(z)

Time complexity: O(d^m)

Space Complexity: O(d^n)

Independence: P(A|B)=P(A) or P(B)

P(A,B)=P(A)P(B)

Bayes' rule: Posterior=P(a | b) = P(b | a) P(a) / P(b)

Naive Bayes Model=P(Y=1|X)=P(y=1)P(x1|y=1)…/P(X)

Bayes Classifier: argmaxP(Y|X1…Xn)

The no of parameters for modelling go down significantly

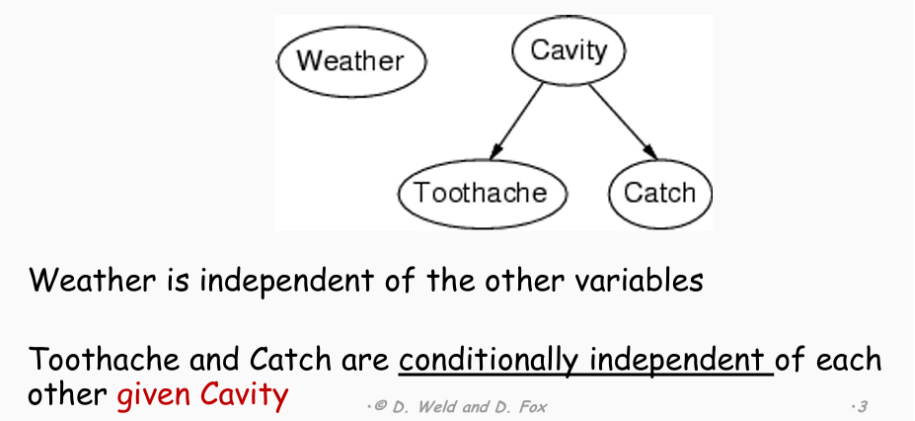
Applications: Spam classification,Weather,Medical diagnosis

→ Assesses diagnostic probability from casual probability

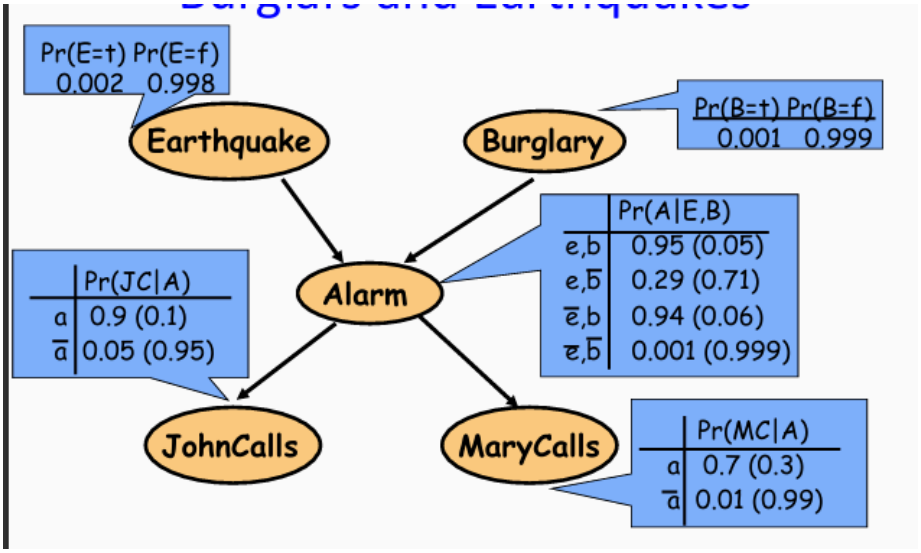
Bayes Nets(BN)

→ Join dist P over a set of variables requires exponential space for representation and inference

Graphical representation off conditional independence relations→ BNs



→ It is basically a directed , acylic graph, with nodes, one per R.V



$P(JC,MC,A,E,B) = P(JC|A) \cdot P(MC|A) \cdot P(A|B,E) \cdot P(E) \cdot P(B)$

Full Joint Dist(Chain Rule)

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Par(X_i))$$

Edges denote probabilistic influence

Marginal Prob from BN: Pr(Z\E) ⟶ Z: Query variable

E: Evidence variable

everything else: hidden

Variable Elimination: Eliminate everything until there is only the query variable

⟶ join all the factors

⟶ sum out the influence

⟶ exploit the product form

Inference in a BN is NP-hard (3-SAT into BN)

Probability =1/2^n

⟶ Polynomial time and compare with inference in CSPs

BN IN A GENERATIVE MODEL

Generate one variable in topological order

Stochastic Simulation ⟶ one after other

P(b/c)=# of live samples with B=b/total # of live samples

Rejection Sampling ⟶ Returns only consistent posterior estimates

Likelihood weighting ⟶ fix evidence variables, sample only non-evidence, weigh each sample by likelihood of evidence

S(z,e) =pi P(z\Parents(Z))

w(z,e)=pi P(e\Parents(E))

Weighted Sampling Prob = S(z,e)w(z,e)

=P(z,e)

$$P(b|c) = \frac{weight\ of\ samples\ with\ B=b}{total\ weight\ of\ samples}$$

MCMC with Gibbs Sampling

- Fix the observed variables
- Randomly assign values to the non-observed
- Random walk ⟶Pick X, Calc P(X=true\other variables), Set X=true

Basically cal P(x|MB(X))

MB ⟶ Markov Blanket

A variable is independent of all others given its Markov Blanket

$$P(X) = \alpha P(X \mid Parents(X)) \prod_{Y \in Children(X)} P(Y \mid Parents(Y))$$

$$P(X|E) = \frac{number\ of\ samples\ with\ X=x}{total\ number\ of\ samples}$$

- Advantages:
  - No samples are discarded
  - No problem with samples of low weight
  - Can be implemented very efficiently
    - 10K samples @ second
- Disadvantages:
  - Can get stuck if relationship between two variables is *deterministic*
  - Many variations have been devised to make MCMC more robust