

Short, short history of AI

1941- first computer

1956 AI coined—make computers think- John McCarthy

1958 LISP—symbol manipulation and recursion

60s—blocks world and stuff

70s—PROLOG (logical programming language)

1991—Chess program beats grand master—AI used in Desert Storm to good effect

Eliza—simulated psychotherapist with a patient, passed the Turing test

Pattern recognition

pattern replied

people poured lives out to—thought solved NL

Parry—Paranoid model—gets emotive, gets more hostile, psychiatrist couldn't tell diff, but paranoid people spout random stuff

Computers are superior at math type things, but cant compete in normal behavior, walk, talk ect

AI CAN:

face recognition

robotics—mostly automobile (to a point)

NLS—machine speech recognition

Expert systems—make diagnosis in narrow area

planning, scheduling (like hubble telescope)

AI CANT

understand real NL

surf web

interpret arbitrary visual input

play GO well

make real time dynamic domain plans

refocus in complex environments

life- long learning

intelligent agents see environment with **sensors** and rationally act with **effectors**

discrete agent receives **percepts** one at a time, and maps percept sequence to a series of discrete actions

rational agents:

should do whatever maximizes performance using

percept sequence

built in and acquired knowledge

info gathering

performance measure

omniscient: knows outcomes of all actions

autonomy means:

own behavior and decisions based on experience

not autonomous if guided by designer hard code—must be influenced by environment

to survive, agent must have enough built in knowledge to start and the ability to learn

software agents aka softbots

properties of environments

fully/partially observable
 sensors can tell everything need to know vs cannot tell everything

Deterministic vs Stochastic
 next state of world all determined by agents actions vs multiple,
 unpredictable outcomes
 * will agent have to deal with uncertainty

Episodic vs. Sequential
 one episode doesn't depend on next (like games) vs series of connected
 episodes (real life)

Discrete vs Continuous
 if number of precepts and actions limited, discrete. else is continuous

Single/ Multi Agent
 if has other agents in environment, needs game theory—competitive or
 cooperative

simple reflex agents: act on current precept, ignoring precept history
condition- action rule: if-then rule
 randomizing actions helps prevent infinite loops

model- based reflex agents keep internal state, keeps track of what things it cant see
 do, keeps track of self

Goal-based agents uses a goal to decide actions in situations—will it take me closer to
 goal?
search of possible actions and **planning** to reach long term goal become issues

utility- based agents compare **utility**, or how happy the agent can be at different
 choices to reach goals
utility functions maps states onto function, which describes how happy will be,
 chooses best

learning agents
learning element makes improvements
performance element selects actions
critic makes observations on performance and feeds to learning element
problem generator suggests actions that lead to learning

CHAPTER 3- UNINFORMED SEARCH

n-queens problem: n queens in chessboard, place all so no one can attack another one
8 puzzle: move numbered tiles around until reach a desired ending

World: deterministic, discrete

Closed world assumption: all relevant info is in prob descrip, we know all we need to
 know

Goal Formulation: based on current situation and performance measure
 good agents can choose from multiple choices of action then pick best—search

state space set of all states reached from goal state

path sequence of states connected by a sequence of actions

goal test decides whether a given state is a goal state

path cost assigns cost value to each path

optimal solution has lowest path cost

incremental formulation changes one thing per state (add 1 queen at a time)

complete state formulation all 8 queens on board, move them around
touring prob, traveling salesman problem, robot navigation, protein design, net searching

Searching for solutions:

Search tree/graph : find paths

search node root of search tree

expand current state to generate new set of states to move toward goal state

search strategy choice of which possible state to expand first

measuring problem solving performance

completeness, optimality, time complexity, space complexity

Uninformed/blind search: doesn't know what states are most promising

Breadth- First Search all one level expanded before next level expanded

imp- stick all newly expanded nodes on end of queue to expand, then expand next one

time complexity $O(b^{(d+1)})$ b nodes, depth d

memory complexity—HUGE make useless for complex problems

Uniform Cost Search expands node with lowest path cost first—node with lowest path cost at front of queue

can get into infinite loop if finds node with no cost action that leads back to self, so completeness is only guaranteed if all steps have a min cost

that is above 0 (min cost called ϵ)

$O(b^{\lceil c^*/\epsilon \rceil})$ c* cost of optimal solution (can be more than b* when step costs are equivalent, is b^d)

Depth- First Search: goes to deepest node—a leaf w. no possible successors, backs up, expands when finds an expandable, ect

low memory requirements, can remove fully explored nodes

backtracking search, variant

depth- bound cuts off search below fixed depth D, ensures terminates, else can get into infinite loop

bad choice early on screws you for a lot of time

Depth- limited search: uses depth- bound, nodes beyond l are treated as no successors

can lead to incompleteness if solution is below l

diameter of a state space: can you reach any element in d steps?

Iterative deepening depth first search: same as DFS with a depth limit, but loops, increasing

depth limit with each unsuccessful search until finds goal state or cannot expand nodes

because most tree nodes are in the bottom of the tree, is ok if generates top nodes many times

time complexity: series $b^d + 2b^{(d-1)} + \dots + db \leq b^d / (1-1/b)^2 = O(b^d)$

Bidirectional Search: run 1 search from start and other from goal, stopping when 2 meet

from goal must apply operations backward

problem must be **symmetric**

time: $O(b^{(d/2)})$ memory: $O(b^d)$

Avoiding Repeated States:

DFS: only nodes in memory are in path, compare to avoid looping paths

closed list: list of all expanded nodes to compare new nodes to, so don't research old paths

open list: fringe of unexpanded nodes

uniform-cost or BFS can't discard a new option because of early apparent path cost—can lose possible solution

Searching with partial info

1) **Sensorless/conformant problem:** agent has no sensors, so doesn't know exact initial state

acts to put self in known state, called **belief state** ex: series of moves always puts

in certain state

2) **Contingency Problems** if environment is partially observable or results of actions uncertain, agent must consider new info after all actions—defined contingencies which must be planned for

3) **Exploration Problems:** Agent doesn't know state and actions for environment and must try to figure them out—the extreme contingency problem

INFORMED SEARCH AND EXPLORATION

informed search one that uses knowledge about problem beyond that in prob description
state space search search to find state space = to goal state by making explicit (expanding) a certain number of nodes

best-first search where node is selected for expansion by **evaluation function** $f(n)$ measures distance from goal

heuristic function takes a node and examines state at that node returns guessed cost of cheapest path

from n to goal— $h(n)$ —also domain knowledge that clues in algorithm what to search first

evaluate a search algorithm with: Completeness (can it find a solution?) Time Complexity, Space

Complexity, Optimality./Admissibility of Solution

Greedy Best-First Search evals nodes using only $f(n)$, example of $f(n)$ for route finding—straight distance from goal

A* Search $f(n) = g(n) + h(n)$ $g(n)$ as cost to get from goal to node, $h(n)$ est cost cheapest solution thru n

is optimal id $h(n)$ is **admissible heuristic** meaning never overestimates cost to reach goal and if

consistency if cost from a to goal is no greater than cost of a to a' to goal if a' is on the way to the goal from a

creates **contours** meaning circles in state space of where for ex, all nodes that cost in a range

A* expands all nodes where $f(n) < c^*$ (cost of optimal solution path)

A* thus **prunes** all nodes where $f(n) > C^*$ because are useless, retains optimality and helps O

A* is **optimally efficient**, expands fewer or equal nodes to all other searches because any other algorithm that doesn't expand $f(n) > C^*$ can miss a solution

Even for A*, nodes in a contour can still be exponential, run out of money and time

Memory Bounded Heuristic Searches

Iterative Deepening A* (IDA*) cut off search by dropping nodes with est value above a point,

gradually increase est value

Improves space cost

Recursive Best First Search RBFS keeps track of f-value of ancestors on path. If curr path exceeds base f-value limit, returns up list to try a diff path using stored f-values to decide where to start. is optimal if h(n) is admissible
often regenerates nodes too much
space complexity is $O(bd)$

IDA* and RBFS use too little memory—between recursions dump all estimated nodes, lose all that time

MA*, **SMA*** uses increasing max value like IDA* and restricts queue size to fit in a certain memory space like beam search

metalevel learning: learns from messups to avoid re-exploring bad subtrees

HEURISTIC FUNCTIONS

effective branching factor a way to measure effectiveness of heuristic funcs, take # nodes expanded to reach goal and arrange in balanced queue of depth d, want b^* close to 1, means went directly to goal

dominates if h_1 dominates h_2 its more efficient w. # nodes expanded

relaxed problem a prob with fewer restrictions cost is admissible heuristic for original prob, if exact cost for relaxed, will be consistent

subproblem can use sol cost for subproblem as even more accurate heuristic

pattern databases store exact cost of subproblem instance

inductive learning algorithms an algorithm that takes a series of optimal solutions and predicts costs for other searches—heuristic func

features properties of a state space given to inductive learning algorithm that help it figure stuff out

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

now want answers to set of probs that don't care about solution path, only solution like 8 Queens

Local Search Algorithms use **current state** not paths generally don't bother with paths
optimizes problems want to find best state according to an **objective function** (measures goodness of state)

state space/landscape looking at location (state) and elevation (cost defined by heuristic or objective func)

global minimum lowest place in state space landscape—goal if elevation corresponds to cost

global max highest spot in state space, desired if elevation corresponds to eval func

complete local search algorithm always finds goals if they exist

optimal local search algorithm always finds global min.max

local beam search generates k random states and develops towards goal, each new generation chooses k best... shoves info on good moves, can get stuck in one part of landscape

stochastic beam search chooses k random states and develops from k parents and k children were prob of choice of a state increases, the states value increases

Genetic Algorithm varies on sto. beam. search, but instead of generating child space from 1 parent, combines 2

fitness function chooses which states best to reproduce, which affects probability of reproduction

crossover points are randomly chosen in pair to be mated

mutation w. every location, small prob of mutation how useful are they? no one knows, but

COOLNESS FACTOR!

schema where part of state can be left unspecified—states that match it are **instances**

hill climbing generate landscape, start at random point and go up steepest include to reach local max (or min if set up that way) of heuristic

can get stuck in plateaus

can get to take random steps on plateaus but must record initial states or else a- >b- >a- >b

sorta like greedy search or beam search with width of 1

w/ good search space is very fast

remedies—random restart and problem reformulation—pick new ops or change state representation

stochastic uphill (chooses a random uphill move)

first choice uphill climbing randomly generates successors til a better state is found

simulated annealing exploits analogy between as metal cools into crystalline structure and search for min or max in more general sense –bounces around into goal (local min) uses control parameter T (temperature) as time goes on in search less likely to take a worse step (a step away from goal)

if T is lowered slowly enough, SA is complete and admissible

CONSTRAINT SATISFACTION PROBLEMS

constraint satisfaction problem a set of variables and a set of constraints, state is assignments to some or all variables

get a finite set of vars, a domain of possible values, a set of constraints

consistent assignment a state which violates no constraints

solution a complete assignment that satisfies all constraints

objective function an optional evaluation function that says whether is good or not

all solutions are complete at depth n , so DFS popular

complete state formulation local searches, start with complete state, and adjust til works

continuous domain problems including real time operations

constraints can be unary (one variable) or binary (involving 2 variables) or more...

cryptarithmic ex of many constraints

constraint hypergraph can draw higher order constraint sections

constraint network a set of variables and each with an associated domain

Binary CSP all constraints are binary or unary—can use graphs to solve by systematic search, guess and

check or **Backtracking**

Backtracking consider vars in some order, pick an empty var, assign a consistent value, keep doing so til get stuck and then backtrack and reassign

problem thrashing keeps trying same illegal moves

can explore areas of search space for a long time with little likelihood of success

Consistency Checking w. backtracking

node consistency node x is node, consistent if every value in x;s domain is consistent with x's unary constraint

arc consistency if for every values x of x there is a value y for y that satisfies constraint represented by arc

a graph is arc consistent if all arcs are arc consistent

constraint propagation to repeatedly reduce domain of each variable to be consistent w/ arcs

note, if cons network cant be made arc cons a domain will collapse to null and no solution!

Constraint trees there exists an arrangement such that every node has 0-2 parents

if constraint tree is node and arc cons can be solved without backtracking

if no tree, try interleaving constraint propagation and backtracking

Variable ordering assign hard ones (highly constrained) early, leaving easy ones for later

Maximum cardinality ordering chose variables cheaper to compute by decreasing cardinality

Fail-First Principle choose vairable w/ fewest values first

static ffp use domain size of vars

dynamic ffp at each pt in search sleect w/ fewest remaining values

Maximal Stable Set find largest set of variables w. no constraint between them and save for last- harder to do

Cycle- cutset tree creation find a set of variables that once instantiated leave a tree of uninstantiated variables, solve these, then solve tree w.out backtracking

Tree composition construct a tree structured set of connected subproblems

GAME PLAYING

evaluation function evals goodness of game position, diff heuristic where eval is est cost from start to goal from given node

Zero- Sum Assumption allows us to use a single eval to describe goodness of board, neg bad for me good for opponent, ect

Game trees (minimax trees)

prob spaces represented as trees

state eval func rates a board state

arcs are possible legal moves

if my turn, is a root as MAX, otherwise MIN

one level is all mins moves, then next is all max's moves, ect

expand to one depth/ply of lookahead back up tree and collect backed up values

alpha- beta pruning

at each max node, $\alpha(n) >$ max value of any child found so far—cause MIN wont choose sucky moves so dont need to expand them

worst—no pruning, b^d leaf nodes w b children and d- ply search

best case, $(2b)^{(d/2)}$ can search twice as deep as minimax alone

Deep Blue avg branching factor 6 instead of 32 for chess

Chance Games

Backgammon- 2 player with uncertainty

must use probability and outcomes, high branching factor w/ each branch, rolls, probability of all

expected values, eval function reflects value of board state value with probability

KNOWLEDGE BASED AGENTS

includes **knowledge base** (set of facts about the world) and inferences to draw conclusions from

sentence represents a piece of knowledge

knowledge representation language expresses sentences

architecture knowledge level most abstract- what agent knows

logical level formal level actual sentences

implementation physical representation of sentences in a logical level

Wumpus World

goal is to make knowledge in **computer tractable** form

knowledge rep is defined by **syntax** (symbols) and **semantics** (facts to which sentences refer)

Sound Heuristic if solution is returned is real

complete heuristic if solution exists, is found

sound inference method if derives a sentence, sentence is really implied

complete inference method can in finite time derive all sentences from knowledge base

PROPOSITIONAL LOGIC

logical constants are t and f

\wedge and \vee or \Rightarrow implies \Leftrightarrow is equiv \sim not \models entails (if p is t, so is q)

truth tables and Venn diagrams

Horn Clause a clause with at most one positive literal – string of ands ($p \wedge q$) or string of ($\sim p \vee q$) they

are equivalent

entailment (if this then these are true—problem solving) vs **derivation** (proof, steps get you there)

can't deal with individuals or patterns

FIRST ORDER LOGIC

Model checking if KB then S? Generate and test

generate all positive models

if VM, S, then s is **provably true**

if VM \sim S then S is **provably false**

Inference rules for FOL

modus ponens ect

New: universal elimination

FOL Inferences harder

variables may have infinite # values from domains

thus potentially infinite ways to apply

Gödel's Completeness Thm there are true statements that cannot be proven from thms

Generalized Modus Ponens

$$\forall x P(x) \rightarrow Q(x)$$

P(A) (if a is an instance of x)

Q(A)

Conjunctive Normal Form “a conjunction of disjunctions” a bunch of statements joined by ors and parens and those are joined by ands. And nots if you want. By convention, remove ands and put ors on separate lines, and then ands are implicit—how represent a knowledge base

Forward Chaining: takes knowledge base and derives more info from it and keeps on going until cant keep deriving stuff

Backward Chaining: start with statement want to prove and goes backwards til reaches axioms

Resolution/refutation have knowledge base and want to prove something is true, assume is false, then

use backwards or forwards chaining until you find a contradiction to something in knowledge base, and then you know its not true from knowledge base, is false, and opposite must be true

Putting FOL into Conjunctive Normal Form:

1) get rid of if and only ifs, two ifs