New rubric: AI in the news

3 minutes

Headlines this week:

Silicon Valley Business Journal: Apple on hiring spreee for AI experts


Forbes: Toyota Invests $50 Million In Artificial Intelligence Research For Vehicle Robotics

Behind Classical Search

Chapter 4
Outline

♦ Hill-climbing
♦ Simulated annealing
♦ Genetic algorithms (briefly)
♦ Local search in continuous spaces (very briefly)
♦ Searching with nondeterministic actions
♦ Searching with partial observations
Iterative improvement algorithms

In many optimization problems, path is irrelevant; the goal state itself is the solution (e.g., 8-queens)

Then state space = set of “complete” configurations;
    find optimal configuration, e.g., TSP
    or, find configuration satisfying constraints, (e.g., timetable)

In such cases, can use iterative improvement algorithms;
keep a single “current” state, try to improve it

Constant space, suitable for online as well as offline search
Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

Variants of this approach get within 1% of optimal very quickly with thousands of cities
Example: $n$-queens

Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.

Move a queen to reduce number of conflicts.

Almost always solves $n$-queens problems almost instantaneously for very large $n$, e.g., $n = 1 \text{millon}$.
Hill-climbing (or gradient ascent/descent)

“Like climbing Everest in thick fog with amnesia”

function Hill-Climbing (problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
neighbor, a node

current ← Make-Node(Initial-State[problem])
loop do
    neighbor ← a highest-valued successor of current
    if Value[neighbor] ≤ Value[current] then return State[current]
    current ← neighbor
end
Hill-climbing contd.

gets stuck (86%, 3 steps), succeeds (14%, 4 steps)

100 consecutive sideways moves - 94% (21 steps for success, 64 for failure)

Variants: stochastic hill climbing, first choice hill climbing

Random restart hill climbing: expected number of steps ???
Hill-climbing contd. (greedy local search)

Useful to consider state space landscape

- global maximum
- local maximum
- "flat" local maximum
- shoulder
- current state

Objective function

State space
Real problems
Simulated annealing

Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency

function Simulated-Annealing(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to “temperature”
local variables: current, a node
                next, a node
                T, a “temperature” controlling prob. of downward steps

current ← Make-Node(Initial-State[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← Value[next] – Value[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^{ΔE/T}
Local beam search

**Idea:** keep $k$ states instead of 1; choose top $k$ of all their successors

Not the same as $k$ searches run in parallel!
Searches that find good states recruit other searches to join them

**Problem:** quite often, all $k$ states end up on same local hill

**Idea:** choose $k$ successors randomly, biased towards good ones

Observe the close analogy to natural selection!
Genetic algorithms

= stochastic local beam search + generate successors from pairs of states

Fitness function: returns higher values for better states, e.g., no of non-attacking pairs of queens
Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components

GAs ≠ evolution: e.g., real genes encode replication machinery!
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
inputs: population, a set of individuals
        FITNESS-FN, a function that measures the fitness of an individual

repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
        population ← new_population
    until some individual is fit enough, or enough time has elapsed
return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
inputs: x, y, parent individuals

n ← LENGTH(x); c ← random number from 1 to n
return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
Continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by \((x_1, y_2), (x_2, y_2), (x_3, y_3)\)
- objective function \(f(x_1, y_2, x_2, y_2, x_3, y_3) = \) sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers \(\pm \delta\) change in each coordinate

Gradient methods compute

\[
\nabla f = \begin{pmatrix}
\frac{\partial f}{\partial x_1}, & \frac{\partial f}{\partial y_1}, & \frac{\partial f}{\partial x_2}, & \frac{\partial f}{\partial y_2}, & \frac{\partial f}{\partial x_3}, & \frac{\partial f}{\partial y_3}
\end{pmatrix}
\]

to increase/reduce \(f\), e.g., by \(x \leftarrow x + \alpha \nabla f(x)\)
Outline

♦ Hill-climbing
♦ Simulated annealing
♦ Genetic algorithms (briefly)
♦ Local search in continuous spaces (very briefly)
♦ Searching with nondeterministic actions
♦ Searching with partial observations
The erratic vacuum world

♦ When applied to a dirty square the action cleans the square and sometimes cleans up dirt in an adjacent square, too.

♦ When applied to a clean square the action sometimes deposits dirt.

Contingency plan for init state=1?

nested if-then-else statements: solution is rather a tree than a sequence
**AND-OR search trees**

**Deterministic env.** - branching is introduced by the agent’s own choices in each state (Left or Right or Suck) - **OR-node**

Standard search tree is all OR-nodes

Agent chooses action;

At least one branch must be solved

**Nondeterministic env.** - branching is introduced by the environment’s choice of outcome of each action (Suck in state 1 leads to \{5,7\}) - **AND-node**

All branches must be solved

Results(s,a) returns a set of states

Results(1,Suck) = 5,7

Results(4,Suck) = 2,4
Results(1,Right) = 2
AND-OR search trees

A solution is a sub-tree that has a goal node at every leaf
The slippery vacuum world

The movement actions sometimes fail, leaving the agent in the same location

**Cyclic solution** - keep trying right until it works

Assumption: each outcome of a nondeterministic action eventually occurs

Is it reasonable? Depends on the reason of nondeterminism (die roll vs. hotel card key)
Outline

♦ Hill-climbing
♦ Simulated annealing
♦ Genetic algorithms (briefly)
♦ Local search in continuous spaces (very briefly)
♦ Searching with nondeterministic actions
♦ Searching with partial observations
Belief state - agents current belief about the possible physical states it might be in, given the sequence of actions and percepts up to that point.

Simplest scenario: the agent has no sensors at all; then we add in partial sensing as well as nondeterministic actions.

Sensorless agents can be surprisingly useful, primarily because they don’t rely on sensors working properly (errors, high cost of sensing).

E.g., doctors often prescribe a broadspectrum antibiotic rather than using the contingent plan of doing an expensive blood test, then waiting for the results to come back, and then prescribing a more specific antibiotic and perhaps hospitalization because the infection has progressed too far.

Part orientation convoyer: https://www.youtube.com/watch?v=QsJzSFVAinhk
Which sequence does guarantee to reach the goal state 7?

**Coercion**: the agent can coerce the world into state 7.

To solve sensorless problems, we search in the space of belief states rather than physical states.
Specifying sensorless problems

\[ P: ACTIONS_P, RESULT_P, GOAL-TEST_P, \text{ and } \text{STEP-COST}_P \]

◊ **Belief states**: every possible set of physical states. If \( P \) has \( N \) states, then the sensorless problem has up to \( 2^N \).

◊ **Initial state**: Typically the set of all states in \( P \), although in some cases the agent will have more knowledge than this.

◊ **Actions**: Suppose the agent is in belief state \( b = s_1, s_2 \), but \( ACTIONS_P(s_1) \neq ACTIONS_P(s_2) \). What are possible actions in current state? (union vs. intersection): \( ACTIONS(b) = \bigcup_{s \in b} ACTIONS_P(s) \)

◊ **Transition model**: deterministic vs. nonderministic

\[
b' = RESULT(b, a) = \{ s' : s' = RESULT_P(s, a) \text{ and } s \in b \}\]

\[
b' = RESULTS(b, a) = \{ s' : s' = RESULTS_P(s, a) \text{ and } s \in b \}\]
Specifying sensorless problems

Predicting the next belief state

◊ **Goal test**: a belief state satisfies the goal only if all the physical states in it satisfy $\text{GOAL} - \text{TEST}_P$. The agent may accidentally achieve the goal earlier, but it won't know that it has done so.

◊ **Path cost**: If the same action can have different costs in different states, then the cost of taking an action in a given belief state could be one of several values.
Reachable belief state space for deterministic
Reachable belief states vs. possible belief states
Incremental belief state search

Builds up the solution one physical state at a time

E.g., initial belief state is \{1, 2, 3, 4, 5, 6, 7, 8\}, and we have to find an action sequence that works in all 8 states.

Finding a solution that works for state 1

Then we check if it works for state 2

If not, go back and find a different solution for state 1,
Searching with partial observations

Local-sensing vacuum world: has a position sensor and a local dirt sensor but has no sensor capable of detecting dirt in other squares.

The percept \([A, \text{dirt}]\) produces in states ...
Transition model in local sensing vacuum
Transition model in local sensing vacuum

3 stages

1. **Prediction stage** is the same as for sensorless problems: given the action \( a \) in belief state \( b \), the predicted belief state is \( \hat{b} = \text{PREDICT}(b, a) \)

2. **Observation prediction stage** determines the set of percepts \( o \) that could be observed in the predicted belief state:

\[
P\text{OSSIBLE-}\text{PERCEPTS}(\hat{b}) = \{ o : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b} \}
\]

3. **Update stage** determines, for each possible percept, the belief state that would result from the percept. The new belief state \( b_o \) is just the set of states in \( \hat{b} \) that could have produced the percept:

\[
b_o = \text{UPDATE}(\hat{b}, o) = \{ s : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b} \}
\]

Note:
◇ the updated belief state $b_o$ can be no larger than the predicted belief state $\hat{b}$

◇ the belief states for the different possible percepts will be disjoint, forming a partition of the original predicted belief state (for deterministic sensing).
Solving partially observable problems

Assuming initial percept [A, dirty] the solution is conditional plan ...

Chapter 4  35
Solving partially observable problems

[Suck, Right, if Bstate =\{6\} then Suck else [ ]].
◊ Local search methods such as hill climbing operate on complete-state formulations, keeping only a small number of nodes in memory. Several stochastic algorithms have been developed, including simulated annealing, which returns optimal solutions when given an appropriate cooling schedule.

◊ A genetic algorithm is a stochastic hill-climbing search in which a large population of states is maintained. New states are generated by mutation and by crossover, which combines pairs of states from the population.

◊ In nondeterministic environments, agents can apply ANDOR search to generate contingent plans that reach the goal regardless of which outcomes occur during execution.

◊ When the environment is partially observable, the belief state represents the set of possible states that the agent might be in.