I’m a Congressman Who Codes. A.I. Freaks Me Out.

By Ted Lieu
Jan. 23, 2023
Finally, a Fast Algorithm for Shortest Paths on Negative Graphs
ChatGPT – calm down everyone

- Q: which is bigger? an elephant or a cat?
  - A: An elephant is bigger than a cat.
- Q: which is not bigger than the other? an elephant or a cat?
  - A: Neither an elephant nor a cat is bigger than the other.
- Q: what is the largest even number?
  - A: The largest even number is 9,999,999,998.
- Q: I started with 20 dollars and gave 10 dollars to my friend. How much do we have in total?
  - A: You and your friend have a total of 30 dollars.
Recap

- A* expands the fringe node with lowest $f$ value where
  - $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost to reach $n$
  - $h(n)$ is an admissible estimate of the least cost from $n$ to a goal node: $0 \leq h(n) \leq h^*(n)$
- A* tree search is optimal
- Its performance depends heavily on the heuristic $h$
Creating Heuristics

YOU GOT

HEURISTIC UPGRADE!
Creating Admissible Heuristics

- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available.

- Problem $P_2$ is a relaxed version of $P_1$ if $\mathcal{A}_2(s) \supseteq \mathcal{A}_1(s)$ for every $s$.

- Theorem: $h_2^*(s) \leq h_1^*(s)$ for every $s$, so $h_2^*(s)$ is admissible for $P_1$. 
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- What are the step costs?

Start State

Goal State

Actions
8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$

Statistics from Andrew Moore:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS</td>
<td>112</td>
<td>6,300</td>
<td>$3.6 \times 10^6$</td>
</tr>
<tr>
<td>A*TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>
What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

- Total *Manhattan* distance

- Why is it admissible?

- $h(\text{start}) = 3 + 1 + 2 + \ldots = 18$

<table>
<thead>
<tr>
<th>Average nodes expanded when the optimal path has...</th>
<th>...4 steps</th>
<th>...8 steps</th>
<th>...12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>A*MANHATTAN</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>
Combining heuristics

- **Dominance:** $h_1 \geq h_2$ if
  \[ \forall n \ h_1(n) \geq h_2(n) \]
  - Roughly speaking, larger is better as long as both are admissible
  - The zero heuristic is pretty bad (what does A* do with $h=0$?)
  - The exact heuristic is pretty good, but usually too expensive!

- **What if we have two heuristics, neither dominates the other?**
  - Form a new heuristic by taking the max of both:
    \[ h(n) = \max(h_1(n), h_2(n)) \]
  - Max of admissible heuristics is admissible and dominates both!
Example: Knight’s moves

- Minimum number of knight’s moves to get from A to B?
  - $h_1 = \text{(Manhattan distance)}/3$
    - $h_1' = h_1$ rounded up to correct parity (even if A, B same color, odd otherwise)
  - $h_2 = \text{(Euclidean distance)}/\sqrt{5}$ (rounded up to correct parity)
  - $h_3 = \text{(max x or y shift)}/2$ (rounded up to correct parity)
  - $h(n) = \max( h_1'(n), h_2(n), h_3(n) )$ is admissible!
Optimality of A* Graph Search

This part is a bit fiddly, sorry about that
Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:

How many states within $d$ steps of start?

How many states in search tree of depth $d$?

Basic idea of graph search: don’t re-expand a state that has been expanded previously
A* Graph Search Gone Wrong?

State space graph

Search tree

Simple check against expanded set blocks C
Fancy check allows new C if cheaper than old but requires recalculating C’s descendants
Consistency of Heuristics

- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    \[ h(A) \leq h^*(A) \]
  - Consistency: heuristic “arc” cost ≤ actual cost for each arc
    \[ h(A) - h(C) \leq c(A,C) \]
    or \[ h(A) \leq c(A,C) + h(C) \] (triangle inequality)
  - Note: \( h^* \) necessarily satisfies triangle inequality

- Consequences of consistency:
  - The \( f \) value along a path never decreases:
    \[ h(A) \leq c(A,C) + h(C) \implies g(A) + h(A) \leq g(A) + c(A,C) + h(C) \]
  - A* graph search is optimal
Sketch: consider what A* does with a consistent heuristic:

- Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
- Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
- Result: A* graph search is optimal
Optimality

- **Tree search:**
  - A* is optimal if heuristic is admissible

- **Graph search:**
  - A* optimal if heuristic is consistent

- Consistency implies admissibility

- Most natural admissible heuristics tend to be consistent, especially if from relaxed problems
But...

- A* keeps the entire explored region in memory
- => will run out of space before you get bored waiting for the answer
- There are variants that use less memory (Section 3.5.5):
  - IDA* works like iterative deepening, except it uses an $f$-limit instead of a depth limit
    - On each iteration, remember the smallest $f$-value that exceeds the current limit, use as new limit
    - Very inefficient when $f$ is real-valued and each node has a unique value
  - RBFS is a recursive depth-first search that uses an $f$-limit = the $f$-value of the best alternative path available from any ancestor of the current node
    - When the limit is exceeded, the recursion unwinds but remembers the best reachable $f$-value on that branch
  - SMA* uses *all available memory* for the queue, minimizing thrashing
    - When full, drop worst node on the queue but remember its value in the parent
A*: Summary

- A* orders nodes in the queue by $f(n) = g(n) + h(n)$
- A* is optimal for trees/graphs with admissible/consistent heuristics
- Heuristic design is key: often use relaxed problems
CS 188: Artificial Intelligence

Local search

Instructors: Stuart Russell and Peyrin Kao

University of California, Berkeley
Local search algorithms

- In many optimization problems, **path** is irrelevant; the goal state is the solution.
- Then state space = set of “complete” configurations; find **configuration satisfying constraints**, e.g., n-queens problem; or, find **optimal configuration**, e.g., travelling salesperson problem.

- 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.
- More or less unavoidable if the “state” is yourself (i.e., learning).
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
Heuristic for $n$-queens problem

- Goal: $n$ queens on board with no *conflicts*, i.e., no queen attacking another
- States: $n$ queens on board, one per column
- Actions: move a queen in its column
- Heuristic value function: number of conflicts
Hill-climbing algorithm

function HILL-CLIMBING(problem) returns a state
  current ← make-node(problem.initial-state)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.value ≤ current.value then
      return current.state
    current ← neighbor
  "Like climbing Everest in thick fog with amnesia"
Global and local maxima

Random restarts
- find global optimum
- duh

Random sideways moves
- Escape from shoulders
- Loop forever on flat local maxima
Hill-climbing on the 8-queens problem

- **No sideways moves:**
  - Succeeds w/ prob. 0.14
  - Average number of moves per trial:
    - 4 when succeeding, 3 when getting stuck
  - Expected total number of moves needed:
    - \(3(1-p)/p + 4 \approx 22\) moves

- **Allowing 100 sideways moves:**
  - Succeeds w/ prob. 0.94
  - Average number of moves per trial:
    - 21 when succeeding, 65 when getting stuck
  - Expected total number of moves needed:
    - \(65(1-p)/p + 21 \approx 25\) moves

Moral: algorithms with knobs to twiddle are irritating
Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state

- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty flaky, doesn’t it?
function SIMULATED-ANNEALING(problem, schedule) returns a state

current ← problem.initial-state

for t = 1 to ∞ do

    T ← schedule(t)

    if T = 0 then return current

    next ← a randomly selected successor of current

    ΔE ← next.value – current.value

    if ΔE > 0 then current ← next

    else current ← next only with probability $e^{ΔE/T}$
Simulated Annealing

- **Theoretical guarantee:**
  - Stationary distribution (Boltzmann): \( P(x) \propto e^{E(x)/T} \)
  - If \( T \) decreased slowly enough, will converge to optimal state!

- **Proof sketch**
  - Consider two adjacent states \( x, y \) with \( E(y) > E(x) \) [high is good]
  - Assume \( x \rightarrow y \) and \( y \rightarrow x \) and outdegrees \( D(x) = D(y) = D \)
  - Let \( P(x), P(y) \) be the equilibrium occupancy probabilities at \( T \)
  - Let \( P(x \rightarrow y) \) be the probability that state \( x \) transitions to state \( y \)
Occupation probability as a function of $T$
Simulated Annealing

- Is this convergence an interesting guarantee?

- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - “Slowly enough” may mean exponentially slowly
  - Random restart hillclimbing also converges to optimal state...

- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems
Local beam search

- **Basic idea:**
  - \( K \) copies of a local search algorithm, initialized randomly
  - For each iteration
    - Generate ALL successors from \( K \) current states
    - Choose best \( K \) of these to be the new current states

Or, \( K \) chosen randomly with a bias towards good ones
Beam search example ($K=4$)
Local beam search

Why is this different from \( K \) local searches in parallel?
- The searches communicate! “Come over here, the grass is greener!”

What other well-known algorithm does this remind you of?
- Evolution!
Genetic algorithms use a natural selection metaphor

- Resample $K$ individuals at each step (selection) weighted by fitness function
- Combine by pairwise crossover operators, plus mutation to give variety
Example: N-Queens

- Does crossover make sense here?
- What would mutation be?
- What would a good fitness function be?
Local search in continuous spaces
Example: Siting airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport.

Airport locations $x = (x_1, y_1), (x_2, y_2), (x_3, y_3)$

City locations $(x_c, y_c)$

$C_a =$ cities closest to airport $a$

Objective: minimize $f(x) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
Handling a continuous state/action space

1. Discretize it!
   - Define a grid with increment $\delta$, use any of the discrete algorithms

2. Choose random perturbations to the state
   a. First-choice hill-climbing: keep trying until something improves the state
   b. Simulated annealing

3. Compute gradient of $f(x)$ analytically
Finding extrema in continuous space

- Gradient vector $\nabla f(x) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, \ldots)^\top$
- For the airports, $f(x) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\partial f/\partial x_1 = \sum_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(x) = 0$
- Can sometimes solve in closed form: $x_1 = (\sum_{c \in C_1} x_c)/|C_1|$
- Is this a local or global minimum of $f$?
- Gradient descent: $x \leftarrow x - \alpha \nabla f(x)$
  - Huge range of algorithms for finding extrema using gradients
Many configuration and optimization problems can be formulated as local search.

General families of algorithms:
- Hill-climbing, continuous optimization
- Simulated annealing (and other stochastic methods)
- Local beam search: multiple interaction searches
- Genetic algorithms: break and recombine states

Many machine learning algorithms are local searches.