Announcements

- Projects: Pacman Search up today, due 9/12
  - Due 11:59pm on Monday 9/28
  - Work in pairs or alone, hand in one per group
  - 5 slip days for projects, can use up to two per deadline

- Replacement lecturer (Prof. Geiger) for two lectures:
  - Game playing
  - Monday 9/28
  - Wednesday 9/30

Today

- A* Search
- Heuristic Design
- Local Search

Recap: Search

- Search problems:
  - States (configurations of the world)
  - Successor functions, costs, start and goal tests

- Search trees:
  - Nodes: represent paths / plans
  - Paths have costs (sum of action costs)
  \[ g(n) = \sum_{x \in p(n)} \text{cost}(x \rightarrow y) \]
  - Strategies differ (only) in fringe management

General Tree Search

- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location

Uniform Cost

- Strategy: expand lowest path cost
  - The good: UCS is complete and optimal!
  - The bad:
    - Explores options in every “direction”
    - No information about goal location
Best First (Greedy)

- Strategy: expand nodes which appear closest to goal
- Heuristic: function which maps states to distance
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS

Example: Heuristic Function

Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost $g(n)$
- Best-first orders by goal proximity, or forward cost $h(n)$

$A^*$ Search orders by the sum: $f(n) = g(n) + h(n)$

When should $A^*$ terminate?

- Should we stop when we enqueue a goal?
  - No: only stop when we dequeue a goal

Is $A^*$ Optimal?

- What went wrong?
  - Actual bad goal cost > estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics

- A heuristic $h$ is admissible (optimistic) if:
  \[ h(n) \leq h^*(n) \]
  where $h^*(n)$ is the true cost to a nearest goal
- E.g. Euclidean distance on a map problem
- Coming up with admissible heuristics is most of what’s involved in using $A^*$ in practice.
Optimality of A*: Blocking

Proof:
- What could go wrong?
- We'd have to have to pop a suboptimal goal G off the fringe before G*.
- This can't happen:
  - Imagine a suboptimal goal G is on the queue
  - Some node n which is a subpath of G* must be on the fringe (why?)
  - n will be popped before G

\[
\begin{align*}
    f(n) &\leq g(G^*) \\
    g(G^*) &< g(G) \\
    g(G) & = f(G) \\
    f(n) &< f(G)
\end{align*}
\]

Properties of A*

Uniform-Cost A* Contours

- Uniform-cost expanded in all directions

- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

Admissible Heuristics

- Most of the work is in coming up with admissible heuristics
- Inadmissible heuristics are often quite effective (especially when you have no choice)
- Very common hack: use \( \alpha \times h(n) \) for admissible \( h \), \( \alpha > 1 \) to generate a faster but less optimal inadmissible \( h' \) from admissible \( h \)

Example: 8 Puzzle

- Number of tiles misplaced?
- Why is it admissible?

\[
\begin{align*}
    h(\text{start}) & = 8 \\
    \text{This is a relaxed-problem heuristic}
\end{align*}
\]

8 Puzzle 1

\[
\begin{array}{c|c|c|c}
\text{TILES} & 112 & 6,300 & 3.6 \times 10^6 \\
\hline
\text{ID} & 13 & 39 & 227 \\
\end{array}
\]
8 Puzzle II

• 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 admissible?
• $h(\text{start}) = 3 + 1 + 2 + \ldots$
• \[= 18\]

<table>
<thead>
<tr>
<th>TILES</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAN-HATTAN</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>

Average nodes expanded when optimal path has length...

\[\begin{array}{c|c|c|c}
\text{TILES} & 4 \text{ steps} & 8 \text{ steps} & 12 \text{ steps} \\
\hline
13 & 39 & 227 \\
\end{array}\]

8 Puzzle III

• How about using the actual cost as a heuristic?
• Would it be admissible?
• Would we save on nodes expanded?
• What’s wrong with it?

• With A*: a trade-off between quality of estimate and work per node!

Trivial Heuristics, Dominance

• Dominance: $h_a \geq h_b$ if
  \[\forall n: h_a(n) \geq h_b(n)\]

• Heuristics form a semi-lattice:
  • Max of admissible heuristics is admissible
  \[h(n) = \max(h_a(n), h_b(n))\]

• Trivial heuristics
  • Bottom of lattice is the zero heuristic (what does this give us?)
  • Top of lattice is the exact heuristic

Course Scheduling

• From the university’s perspective:
  • Set of courses $\{c_1, c_2, \ldots, c_n\}$
  • Set of room / times $\{r_1, r_2, \ldots, r_m\}$
  • Each pairing $(c_i, r_j)$ has a cost $w_{ij}$
  • What’s the best assignment of courses to rooms?

• States: list of pairings
• Actions: add a legal pairing
• Costs: cost of the new pairing
• Admissible heuristics?

Other A* Applications

• Pathing / routing problems
• Resource planning problems
• Robot motion planning
• Language analysis
• Machine translation
• Speech recognition
• …

Tree Search: Extra Work?

• Failure to detect repeated states can cause exponentially more work. Why?
Graph Search

- In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)

Optimality of A* Graph Search

- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
  - Proof idea: optimal goals have lower f value, so get expanded first

Consistency

- Wait, how do we know we expand in increasing f value?
- Couldn’t we pop some node n, and find its child n’ to have lower f value?
- YES:

Optimality

- Tree search:
  - A* optimal if heuristic is admissible (and non-negative)
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- In general, natural admissible heuristics tend to be consistent

Summary: A*

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible heuristics
- Heuristic design is key: often use relaxed problems
Limited Memory Options

- Bottleneck: not enough memory to store entire fringe
- Hill-Climbing Search:
  - Only "best" node kept around, no fringe!
  - Usually prioritize successor choice by h (greedy hill climbing)
  - Compare to greedy backtracking, which still has fringe
- Beam Search (Limited Memory Search)
  - In between: keep K nodes in fringe
  - Can prioritize by h alone (greedy beam search), or h+g (limited memory A*)
  - Why not applied to UCS?
  - We’ll return to beam search later…
  - No guarantees once you limit the fringe size!

Large Scale Problems with A*

- What states get expanded?
  - All states with f-cost less than optimal goal cost
- How far "in every direction" will this be?
  - Iterative: depth grows like the heuristic “gap” (h(n) – g(n))
  - Usually at least linear in problem size
- Work exponential in depth?
- In huge problems, often A* isn’t enough
  - State space just too big
  - Can’t visit all states with f less than optimal
  - Often, can’t even store the entire fringe
- Solutions
  - Beam heuristics
  - Beam search (limited fringe size)
  - Greedy hill-climbing (fringe size = 1)

Types of Problems

- Planning problems:
  - We want a path to a solution (examples?)
  - Usually want an optimal path
  - Incremental formulations
- Identification problems:
  - We actually just want to know what the goal is (examples?)
  - Usually want an optimal goal
  - Complete-state formulations
  - Iterative improvement algorithms

Local Search Methods

- Queue-based algorithms keep fallback options (backtracking)
  - Local search: improve what you have until you can’t make it better
  - Generally much more efficient (but incomplete)

Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What’s good about it?
### Simulated Annealing

**Idea:** Escape local maxima by allowing downhill moves
- But make them rarer as time goes on

*Algorithm:*

```python
def 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 downhill 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 or EXP(-ΔE / T) > random number
        current = next
    else current = next with probability EXP(ΔE / T)
```

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

**Is this an interesting guarantee?**
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you are to make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

### Beam Search

- Like greedy search, but keep \( K \) states at all times:

**Greedy Search**

- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

**Continuous Problems**

- Placing airports in Romania
  - States: \((x_1, y_1, x_2, y_2, x_3, y_3)\)
  - Cost: sum of squared distances to closest city

### Gradient Methods

- How to deal with continuous (therefore infinite) state spaces?
- Discretization: bucket ranges of values
  - E.g. force integral coordinates
- Continuous optimization
  - E.g. gradient ascent

\[
\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)
\]

\[
x \leftarrow x + \alpha \nabla f(x)
\]

### Optimization

- Big literature in Math/CS
  - See [G22.2945-002](#)
- 2\textsuperscript{nd} order methods can speed things up
  - Need to compute Hessian
  - Slow for big problems
**Genetic Algorithms**

- Genetic algorithms use a natural selection metaphor
- Like beam search (selection), but also have pairwise crossover operators, with optional mutation
- Probably the most misunderstood, misapplied (and even maligned) technique around!

**Example: N-Queens**

- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?

**Genetic Algorithms Applications**

- Great for massive discrete problems, or when difficult to compute gradients in continuous case
- Many applications:
  - Solar arrays
  - Protein folding
  - Electric motors