Today

- A* Search
- Heuristic Design
- Local Search

Problem Graphs vs Search Trees

Each NODE in the search tree is an entire PATH in the problem graph.

Uniform Cost Problems

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location

Best-First Search

- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)
Combining Best-First and UCS

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

What happens with each ordering?
- \( A^* \) 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 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 can go wrong?
  - We'd have to pop a suboptimal goal off the fringe queue
  - Imagine a suboptimal goal \( G' \) is on the queue
  - Consider any unexpanded (fringe) node \( n \) on a shortest path to optimal \( G \)
  - \( n \) will be popped before \( G \)

Optimality of \( A^* \): Contours

- Consider what \( A^* \) does:
  - Expands nodes in increasing total value (\( f \)-contours)
  - Optimal goals have lower \( f \) value, so get expanded first
- Holds for graph search as well, but we made a different assumption. What?
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 higher f value?
- YES:
  
  \[ g = 10 \]
  
  \[ h = 10 \]
  
  \[ h = 8 \]
  
- What do we need to do to fix this?
- Consistency: \( h(n) \leq c(n, a, n') + h(n') \)
- Real cost always exceeds reduction in heuristic

UCS vs A* Contours

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

Properties of A*

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS = BFS*</td>
<td>Y</td>
<td>Y</td>
<td>O(k * b)</td>
<td>O(k * b^*</td>
</tr>
<tr>
<td>A*</td>
<td>Y</td>
<td>Y</td>
<td>O(k)</td>
<td>O(k)</td>
</tr>
</tbody>
</table>

*Assume all costs are 1  Assume one goal, non-goals have \( h(n) = g*(G) - a \)

Uniform-Cost A*

Admissible Heuristics

- Most of the work is in coming up with admissible heuristics
- Good news: usually admissible heuristics are also consistent
- More good news: inadmissible heuristics are often quite effective (especially when you have no choice)

8-Puzzle I

- Number of tiles misplaced?
- Why is it admissible?
- \( h(\text{start}) = 8 \)
- This is a relaxed problem heuristic

| Average nodes expanded when optimal path has length |
|--------|--------|--------|--------|
| 4 steps | 8 steps | 12 steps |
| ID     | 112    | 6,300  | 3.6 x 10^6 |
| TILES  | 13     | 39     | 227    |

8-Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any one direction at any time?
- Total Manhattan distance
- Why admissible?
- \( h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \)

| Average nodes expanded when optimal path has length |
|--------|--------|--------|--------|
| 4 steps | 8 steps | 12 steps |
| TILES  | 13     | 39     | 227    |
| MANHATTAN | 12 | 25 | 73    |
8-Puzzle III

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

- With A*, trade-off between quality of estimate and work per node!

Trivial Heuristics, Dominance

- Dominance:
  \[ \forall n : h_a(n) > h_c(n) \]

- 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 \( \{r_1, r_2, \ldots, r_n\} \)
  - 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?

- (Who can think of a cs170 answer to this problem?)

Other A* Applications

- Machine translation
- Statistical parsing
- Speech recognition
- Robot motion planning (next class)
- Routing problems (see homework!)
- Planning problems (see homework!)
- …

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

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)
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

Example: N-Queens

- Start wherever, move queens to reduce conflicts
- Almost always solves large n-queens nearly instantly
- How is this different from best-first search?

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?

Hill Climbing Diagram

Simulated Annealing

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

Simulated Annealing

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

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?

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
  - \( \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right) \)
  - \( x \leftarrow x + \alpha \nabla f(x) \)
- More on this next class…