Last Time: A* Search

- Use an evaluation function \( f(n) \) for node \( n \).
  \[ f(n) = \text{estimated total cost of path thru } n \text{ to goal} \]
- \( f(n) = g(n) + h(n) \)
  - \( g(n) \): cost so far to reach \( n \)
  - \( h(n) \): estimated cost from \( n \) to goal
- Always choose the node from frontier that has the lowest \( f \) value.
  Frontier = priority queue
A* vs. Uniform Cost Search vs. Dijkstra

• All three are optimal but differ in search strategy, time/space complexity, and goals
• A* uses $f(n) = g(n) + h(n)$ to find shortest path to a single goal
• Uniform cost search uses $f(n) = g(n)$ to find shortest path to a single goal
• Dijkstra’s algorithm also uses $f(n) = g(n)$ but finds shortest paths to all nodes

A* vs. Uniform Cost Search vs. Dijkstra

• A* expands mainly toward the goal with the help of the heuristic function
• Uniform-cost and Dijkstra expand in all directions
• A* can be more efficient if the heuristic is good
Uniform Cost Pac-Man

A* Pac-Man with Manhattan distance heuristic
Recall: Admissible Heuristics

- $A^*$ uses $f(x) = g(x) + h(x)$
- $g$: cost so far
- $h$: underestimate of remaining costs

Proved last time: If $h(n)$ admissible, $A^*$ optimal

e.g., $h_{SLD}$ is an admissible heuristic for the route finding problem

More heuristic functions

For the 8-puzzle (get to goal state with smallest # of moves), what are some heuristic functions?

- $h_1(n) = ?$
- $h_2(n) = ?$

![Start State](image1)

![Goal State](image2)
Example heuristic functions

Examples:
• $h_1(n) =$ number of misplaced tiles
• $h_2(n) =$ total Manhattan distance (no. of squares from desired location of each tile)

• $h_1(S) = ?$
• $h_2(S) = ?$

Example heuristics

Examples:
• $h_1(n) =$ number of misplaced tiles
• $h_2(n) =$ total Manhattan distance (no. of squares from desired location of each tile)

• $h_1(S) = ?$ 8
• $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

• Are these admissible heuristics?
Dominance

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$
- $h_2$ is better for search (why?)
  Getting closer to the actual cost to goal

- Does one dominate the other for:
  $h_1(n) =$ number of misplaced tiles
  $h_2(n) =$ total Manhattan distance

Dominance

- For 8-puzzle heuristics $h_1$ and $h_2$, typical search costs (average number of nodes expanded for solution depth $d$):

  - $d=12$  
    IDS = 3,644,035 nodes  
    $A^*(h_1) = 227$ nodes  
    $A^*(h_2) = 73$ nodes

  - $d=24$  
    IDS = too many nodes to fit in memory  
    $A^*(h_1) = 39,135$ nodes  
    $A^*(h_2) = 1,641$ nodes
For many problems, A* can still require too much memory

**Iterative-Deepening A*** (IDA***)

- Less memory required compared to A*
- Like iterative-deepening search, but...
- Depth bound modified to be an \( f\)-limit
  
  Start with \( \text{limit} = \text{h}(\text{start}) \)
  
  Prune any node if \( f(\text{node}) > \text{f-limit} \)
  
  Next \( f\)-limit=min-cost of any node pruned
That’s cool yo but how do you derive heuristic functions?

Relaxed Problems

• Derive admissible heuristic from exact cost of a solution to a relaxed version of problem

For route planning, what is a relaxed problem?

Relax requirement that car has to stay on road
→ Straight Line Distance becomes optimal cost

• Cost of optimal soln to relaxed problem ≤ cost of optimal soln for real problem
Heuristics for eight puzzle

Original: Tile can move from location A to B if A is horizontally or vertically next to B and B is blank

Relaxed 1: Tile can move from any loc A to any loc B
Cost = $h_1 = \text{number of misplaced tiles}$

Relaxed 2: Tile can move from loc A to loc B if A is horizontally or vertically next to B
Cost = $h_2 = \text{total Manhattan distance}$

• What can we relax?
Need for Better Heuristics

Performance of $h_2$ (Manhattan Distance Heuristic)

- 8 Puzzle: < 1 second
- 15 Puzzle: 1 minute
- 24 Puzzle: 65000 years

Can we do better?

Creating New Heuristics

- Given admissible heuristics $h_1$, $h_2$, ..., $h_m$, none of them dominating any other, how to choose the best?

- Answer: No need to choose only one! Use:
  \[ h(n) = \max \{ h_1(n), h_2(n), ..., h_m(n) \} \]
  - $h$ is admissible (why?)
  - $h$ dominates each individual $h_i$ (by construction)
**Pattern Databases** [Culberson & Schaeffer 1996]

- **Idea:** Use solution cost of a subproblem as heuristic. For 8-puzzle: pick any subset of tiles
  - E.g., 3 tiles
- **Precompute a table**
  - Compute optimal cost of solving just these tiles
    - This is a lower bound on actual cost with all tiles
  - For all possible configurations of these tiles
    - Could be several million
  - Use breadth first search back from goal state
    - State = position of just these tiles (& blank)
- **Admissible heuristic** $h_{DB}$ for complete state = cost of corresponding sub-problem state in database

---

**Combining Multiple Databases**

- **Repeat for another subset of tiles**
  - Precompute multiple tables
- **How to combine table values?**
  - Use the max trick!
- **E.g. Optimal solutions to Rubik’s cube**
  - First found w/ IDA* using pattern DB heuristics
  - Multiple DBs were used (diff subsets of cubies)
  - Most problems solved optimally in 1 day
  - Compare with 574,000 years for IDS
Drawbacks of Standard Pattern DBs

• Since we can only take max
  Diminishing returns on additional DBs

• Would like to be able to add values
  • But not exceed the actual solution cost (admissible)
  • How?

Disjoint Pattern DBs

• Partition tiles into disjoint sets
  For each set, precompute table
  Don’t count moves of tiles not in set
  • This makes sure costs are disjoint
  • Can be added without overestimating!
  • E.g. 8 tile DB has 519 million entries
  • And 7 tile DB has 58 million

• During search
  Look up costs for each set in DB
  Add values to get heuristic function value

  Manhattan distance is a special case of this idea where each set is a single tile
Performance of Disjoint PDBs

- **15 Puzzle:** 2000x speedup vs Manhattan dist
  IDA* with the two DBs solves 15 Puzzles optimally in 30 milliseconds

- **24 Puzzle:** 12 millionx speedup vs Manhattan
  IDA* can solve random instances in 2 days
  Uses DBs for 4 disjoint sets as shown
  Each DB has 128 million entries
  Without PDBs: 65,000 years

Adapted from Richard Korf presentation

Next Time

- Local search
- Gaming search and searching for Games
- To do: Project #1, Reading