Today’s Class

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A, A*
  - Examples
- Memory-conserving variations of A*
- Heuristic functions

"An informed search strategy—one that uses problem specific knowledge...can find solutions more efficiently than an uninformed strategy." – R&N pg. 92

Weak vs. Strong Methods

- Weak methods:
  - Extremely general, not tailored to a specific situation
- Examples
  - Means-ends analysis: the current situation and goal, then look for ways to shrink the differences between the two
  - Space splitting: try to list possible solutions to a problem, then try to rule out classes of these possibilities
  - Subgoaling: split a large problem into several smaller ones that can be solved one at a time.
- Called “weak” methods because they do not take advantage of more powerful domain-specific heuristics

Heuristic

Free On-line Dictionary of Computing*

1. **A rule of thumb, simplification, or educated guess**
2. Reduces, limits, or guides search in particular domains
3. Does not guarantee feasible solutions; often used with no theoretical guarantee

WordNet (r) 1.6*

1. Commonsense rule (or set of rules) intended to increase the probability of solving some problem

Heuristic Search

- Uninformed search is **generic**
  - Node selection depends only on shape of tree and node expansion strategy.
- Sometimes **domain knowledge** → Better decision
  - Knowledge about the specific problem

Bookkeeping

- Next lecture:
  - Python for AI
  - Eight decades of AI
  - (okay, 4)
Heuristic Search

- Romania: Arad → Bucharest (for example)

Heuristic Search

- Romania:
  - Eyeballing it → certain cities first
  - They “look closer” to where we are going

- Can domain knowledge be captured in a heuristic?

Heuristics Examples

- 8-puzzle:
  - # of tiles in wrong place
- 8-puzzle (better):
  - Sum of distances from goal
  - Captures distance and number of nodes
- Romania:
  - Straight-line distance from start node to Bucharest
  - Captures “closer to Bucharest”

Heuristic Function

- All domain-specific knowledge is encoded in heuristic function $h$
- $h$ is some estimate of how desirable a move is
  - How “close” (we think) it gets us to our goal
- Usually:
  - $h(n) \geq 0$: for all nodes $n$
  - $h(n) = 0$: $n$ is a goal node
  - $h(n) = \infty$: $n$ is a dead end (no goal can be reached from $n$)

Example Search Space Revisited

- Goal: select the best path to continue searching
- Define $h(n)$ to estimates the “goodness” of node $n$
  - $h(n) = \text{estimated cost}$ (or distance) of minimal cost path from $n$ to a goal state
- Heuristic function is:
  - An estimate of how close we are to a goal
  - Based on domain-specific information
  - Computable from the current state description
Straight Lines to Budapest (km)

Admissible Heuristics

- Admissible heuristics never overestimate cost
  - They are optimistic – think goal is closer than it is
    - $h(n) \leq h^*(n)$
  - where $h^*(n)$ is true cost to reach goal from $n$
    - $h_{LSD}(\text{Lugoj}) = 244$
  - Can there be a shorter path?
  - Using admissible heuristics guarantees that the first solution found will be optimal

Best-First Search

- A generic way of referring to informed methods
- Use an evaluation function $f(n)$ for each node
  - estimate of “desirability”
    - $f(n)$ incorporates domain-specific information
  - Different $f(n) \Rightarrow$ Different searches

Greedy Best-First Search

- Idea: always choose “closest node” to goal
  - Most likely to lead to a solution quickly
- So, evaluate nodes based only on heuristic function
  - $f(n) = h(n)$
- Sort nodes by increasing values of $f$
- Select node believed to be closest to a goal node (hence “greedy”)
  - That is, select node with smallest $f$ value

Best-First Search

- Order nodes on the list by
  - Increasing value of $f(n)$
- Expand most desirable unexpanded node
  - Implementation:
    - Order nodes in frontier in decreasing order of desirability
  - Special cases:
    - Greedy best-first search
    - A* search

Greedy Best-First Search

- Admissible?
  - Why not?
- Example:
  - Greedy search will find:
    - $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow g$; cost = 5
  - Optimal solution:
    - $a \rightarrow g \rightarrow h \rightarrow i$; cost = 3
  - Not complete (why?)
What can we say about the search space?
Beam Search

- Use an evaluation function $f(n) = h(n)$, but the maximum size of the nodes list is $k$, a fixed constant
- Only keeps $k$ best nodes as candidates for expansion, and throws the rest away
- More space-efficient than greedy search, but may throw away a node that is on a solution path
- Not complete
- Not admissible

Algorithm A

- Use evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = minimal-cost path from any $S$ to state $n$
- Ranks nodes on search frontier by estimated cost of solution
  - From start node, through given node, to goal
- Not complete if $h(n)$ can = $\infty$
- Not admissible

Example Search Space Revisited

Algorithm A

1. Put start node $S$ on the nodes list, called OPEN
2. If OPEN is empty, exit with failure
3. Select node in OPEN with minimal $f(n)$ and place on CLOSED
4. If $n$ is a goal node, collect path back to start; terminate
5. Expand $n$, generating all its successors, and attach to them pointers back to $n$. For each successor $n'$ of $n$
   1. If $n'$ is not already on OPEN or CLOSED
      - compute $f(n') = g(n) + c(n,n')$, $h(n') = h(n')$, $f(n') = g(n') + h(n')$
   2. If $n'$ is already on OPEN or CLOSED and if $g(n')$ is lower for the new version of $n'$, then:
      - Redirect pointers backward from $n'$ along path yielding lower $g(n')$.
      - Put $n'$ on OPEN.
Some Observations on A*

• **Perfect heuristic:** If \( h(n) = h^*(n) \) for all \( n \):
  - Only nodes on the optimal solution path will be expanded
  - No extra work will be performed

• **Null heuristic:** If \( h(n) = 0 \) for all \( n \):
  - This is an admissible heuristic
  - \( A^* \) acts like Uniform-Cost Search

The closer \( h \) is to \( h^* \), the fewer extra nodes will be expanded.

• **Better heuristic:** If \( h_1(n) < h_2(n) \leq h^*(n) \) for all non-goal nodes, \( h_2 \) is a better heuristic than \( h_1 \):
  - If \( A_1^* \) uses \( h_1 \), \( A_2^* \) uses \( h_2 \),
    - every node expanded by \( A_1^* \) is also expanded by \( A_2^* \)
  - So \( A_1 \) expands at least as many nodes as \( A_2^* \)

We say that \( A_2^* \) is better informed than \( A_1^* \).

Quick Terminology Check

• What is \( f(n) \)?
  - An evaluation function that gives...
  - A cost estimate of...
  - The distance from \( n \) to \( G \)

• What is \( h(n) \)?
  - A heuristic function that...
  - Encodes domain knowledge about...
  - The search space

• What is \( h^*(n) \)?
  - A heuristic function that gives the...
  - True cost to reach goal from \( n \)
  - Why don’t we just use that?

• What is \( g(n) \)?
  - The path cost of getting from \( S \) to \( n \)
  - Describes the “spent” costs of the current search

A* Search

• Avoid expanding paths that are already expensive
  - Combines costs-so-far with expected-costs

• \( A^* \) is **complete** iff
  - Branching factor is finite
  - Every operator has a fixed positive cost

• \( A^* \) is **admissible** iff
  - \( h(n) \) is admissible

\[ f(n) = g(n) + h(n) \]

A* Example 1

• **Idea:** Evaluate nodes by combining \( g(n) \), the cost of reaching the node, with \( h(n) \), the cost of getting from the node to the goal.

• Evaluation function \( f(n) = g(n) + h(n) \):
  - \( g(n) \) = cost so far to reach \( n \)
  - \( h(n) \) = estimated cost from \( n \) to goal
  - \( f(n) \) = estimated total cost of path through \( n \) to goal
Algorithm A*

- Algorithm A with constraint that $h(n) \leq h^*(n)$
  - $h^*(n)$ = true cost of the minimal cost path from $n$ to a goal.
- Therefore, $h(n)$ is an underestimate of the distance to the goal.
- $h()$ is admissible when $h(n) \leq h^*(n)$
  - Guarantees optimality.
- A* is complete whenever the branching factor is finite, and every operator has a fixed positive cost.
- A* is admissible.
Example Search Space Revisited

Example

Greedy Search

A* Search

Proof of the Optimality of A*

Admissible heuristics

- Assume that A* has selected $G_2$, a goal state with a suboptimal solution ($g(G_2) > f^*$).
- We show that this is impossible.
  - Choose a node $n$ on the optimal path to $G$.
  - Because $h(n)$ is admissible, $f(n) \leq f^*$.
  - If we choose $G_2$ instead of $n$ for expansion, $f(G_2) \leq f(n)$.
  - This implies $f(G_2) \leq f^*$.
  - $G_2$ is a goal state: $h(G_2) = 0$, $f(G_2) = g(G_2)$.
  - Therefore $g(G_2) \leq f^*$.
  - Contradiction.

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance
  (i.e., # of squares each tile is from desired location)

  \[
  h_1(n) = \sum \text{misplaced tiles} \\
  h_2(n) = \sum \text{Manhattan distance}
  \]
Admissible heuristics

E.g., for the 8-puzzle:

- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = total Manhattan distance (i.e., # of squares each tile is from desired location)

- \( h_1(S) = 8 \)
- \( h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \)

Dealing with Hard Problems

- For large problems, A* often requires too much space.
- Two variations conserve memory: IDA* and SMA*
- IDA* – iterative deepening A*
  - uses successive iteration with growing limits on \( f \). For example,
    - A* but don’t consider any node \( n \) where \( f(n) > 10 \)
    - A* but don’t consider any node \( n \) where \( f(n) > 20 \)
    - A* but don’t consider any node \( n \) where \( f(n) > 30 \), ...
- SMA* – Simplified Memory-Bounded A*
  - uses a queue of restricted size to limit memory use.
  - throws away the “oldest” worst solution.

What’s a Good Heuristic?

- If \( h_1(n) < h_2(n) \leq h^*(n) \) for all \( n \), then:
  - Both are admissible
  - \( h_2 \) is strictly better than (dominates) \( h_1 \).
- How do we find one?
  1. **Relaxing the problem:**
     - Remove constraints to create a (much) easier problem
     - Use the solution cost for this problem as the heuristic function
  2. **Combining heuristics:**
     - Take the max of several admissible heuristics
     - Still have an admissible heuristic, and it’s better!

What’s a Good Heuristic? (2)

3. Use statistical estimates to compute \( h \)
   - May lose admissibility
4. Identify good features, then use a learning algorithm to find a heuristic function
   - Also may lose admissibility
- Why are these a good idea, then?
  - Machine learning can give you answers you don’t “think of”
  - Can be applied to new puzzles without human intervention
  - Often work

Some Examples of Heuristics?

- 8-puzzle?
  - Manhattan distance
- Driving directions?
  - Straight line distance
- Crossword puzzle?
- Making a medical diagnosis?

Summary: Informed Search

- **Best-first search:** general search where the minimum-cost nodes (according to some measure) are expanded first.
- **Greedy search:** uses minimal estimated cost \( h(n) \) to the goal state as measure. Reduces search time but is neither complete nor optimal.
- **A* search:** combines UCS and greedy search
  - \( f(n) = g(n) + h(n) \)
  - A* is complete and optimal, but space complexity is high.
  - Time complexity depends on the quality of the heuristic function.
- IDA* and SMA* reduce the memory requirements of A*.
In-class Exercise: Creating Heuristics

8-Puzzle  Boat Problems  Remove 5 Sticks

N-Queens  Water Jug Problem  Route Planning

Apply the following to search this space. At each search step, show:
- the current node being expanded,
- \( g(n) \) (path cost so far),
- \( h(n) \) (heuristic estimate),
- \( f(n) \) (evaluation function), and
- \( h^*(n) \) (true goal distance).

Depth-first search  Breadth-first search  A* search
Uniform-cost search  Greedy search

In-Class Exercise

Apply the following to search this space. At each search step, show:
- the current node being expanded,
- \( g(n) \) (path cost so far),
- \( h(n) \) (heuristic estimate),
- \( f(n) \) (evaluation function), and
- \( h^*(n) \) (true goal distance).