Outline

- Best-first search
  - Greedy best-first search
  - A* search
  - Heuristics
- Local search algorithms
  - Hill-climbing search
  - Beam search
  - Simulated annealing search
  - Genetic algorithms
- Constraint Satisfaction Problems

Many slides based on Russell & Norvig's slides
Artificial Intelligence: A Modern Approach
Motivation

- Uninformed search algorithms are too inefficient
  - they expand far too many unpromising paths
- Example:
  - 8-puzzle

- Average solution depth = 22
- Breadth-first search to depth 22 has to expand about $3.1 \times 10^{10}$ nodes

→ try to be more clever with what nodes to expand
Best-First Search

- Recall
  - Search strategies are characterized by the order in which they expand the nodes of the search tree
  - Uninformed tree-search algorithms sort the nodes by problem-independent methods (e.g., recency)

- Basic Idea of Best-First Search
  - use an evaluation function $f(n)$ for each node
    - estimate of the "desirability" of the node's state
    - expand most desirable unexpanded node

- Implementation
  - use Game-Tree-Search algorithm
  - order the nodes in fringe in decreasing order of desirability

- Algorithms
  - Greedy best-first search
  - A* search
Heuristic

- Greek "heurisko" (εὑρίσκω) → "I find"
  - cf. also „Eureka!“

- informally denotes a „rule of thumb“
  - i.e., knowledge that may be helpful in solving a problem
  - note that heuristics may also go wrong!

- In tree-search algorithms, a heuristic denotes a function that estimates the remaining costs until the goal is reached

- Example:
  - straight-line distances may be a good approximation for the true distances on a map of Romania
  - and are easy to obtain (ruler on the map)
    - but cannot be obtained directly from the distances on the map
Romania Example: Straight-line Distances

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobreta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>176</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy Best-First Search

- Evaluation function \( f(n) = h(n) \) (heuristic)
  - estimates the cost from node \( n \) to \( \text{goal} \)
  - e.g., \( h_{SLD}(n) \) = straight-line distance from \( n \) to Bucharest

- Greedy best-first search expands the node that appears to be closest to goal
  - according to evaluation function

- Example:
Greedy Best-First Search

- Evaluation function \( f(n) = h(n) \) (heuristic)
  - estimates the cost from node \( n \) to goal
  - e.g., \( h_{SLD}(n) \) = straight-line distance from \( n \) to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal
  - according to evaluation function
- Example:
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
  - estimates the cost from node $n$ to goal
  - e.g., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal
  - according to evaluation function
- Example:

```
Arad
  └── Sibiu
      ├── Arad 366
      │    └── Fagaras 176
      │         └── Oradea 380
      └── Timisoara 329
          └── Zerind 374
       └── Rimnicu Vlaicu 193
```
Greedy Best-First Search

- Evaluation function $f(n) = h(n)$ (heuristic)
  - estimates the cost from node $n$ to goal
  - e.g., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal
  - according to evaluation function
- Example:
Properties of Greedy Best-First Search

- **Completeness**
  - No – can get stuck in loops
  - Example: We want to get from Iasi to Fagaras
    - Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...

**Note:**
These two are different search nodes referring to the same state!

Neamt is closer to Fagaras than Vaslui
Properties of Greedy Best-First Search

- Completeness
  - No – can get stuck in loops
  - can be fixed with careful checking for duplicate states
    $\rightarrow$ complete in finite state space with repeated-state checking

- Time Complexity
  - $O(b^m)$, like depth-first search
  - but a good heuristic can give dramatic improvement
    - optimal case: best choice in each step $\rightarrow$ only $d$ steps
    - a good heuristic improves chances for encountering optimal case

- Space Complexity
  - has to keep all nodes in memory $\rightarrow$ same as time complexity

- Optimality
  - No
  - Example:
    - solution Arad $\rightarrow$ Sibiu $\rightarrow$ Fagaras $\rightarrow$ Bucharest is not optimal
A* Search

- Best-known form of best-first search

- Basic idea:
  - avoid expanding paths that are already expensive
  - evaluate complete path cost not only remaining costs

- Evaluation function: \( f(n) = g(n) + h(n) \)
  - \( g(n) \) = cost so far to reach node \( n \)
  - \( h(n) \) = estimated cost to get from \( n \) to goal
  - \( f(n) \) = estimated cost of path to goal via \( n \)
Beispiel

$g(n)$

$h(n)$

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A* Search Example

Arad

366 = 0 + 366
A* Search Example

Arad

Sibiu
393 = 140 + 253

Timisoara
447 = 118 + 329

Zerind
449 = 75 + 374
A* Search Example

Informed Search
A* Search Example

A* search is a formal algorithm that efficiently finds the least-cost path from a given start node to a given goal node in a weighted graph. It combines the advantages of Dijkstra's algorithm and the greedy best-first search algorithm. A* is widely used in pathfinding and graph traversal, the process of finding a path from a starting location to a goal location, as well as other more complex problems such as the Traveling Salesman problem.

In A*, the algorithm is given a graph where each node represents a state or configuration of the problem, and each edge between nodes represents a transition or an action that can be taken. Each node has a cost associated with it, which is the total cost of the path from the start node to the current node. Additionally, each node has a heuristic estimate of the cost to reach the goal from the current node. The algorithm maintains an open list of nodes to be evaluated and a closed list of nodes that have already been evaluated. At each step, it selects the node with the lowest cost from the open list, evaluates it, and adds it to the closed list. If the goal node is reached, the algorithm returns the path from the start node to the goal node. Otherwise, it expands the node and adds its unvisited neighbors to the open list, updating their costs and heuristic estimates accordingly.
**A* Search Example**

Note that Pitesti will be expanded even though Bucharest is already in the fringe! This is good, because we may still find a shorter way to Budapest. Greedy Search would not do that.
A* Search Example

- **Arad**
  - **Sibiu**: 646 = 280 + 366
  - **Fagaras**: 671 = 291 + 380
  - **Oradea**: 591 = 338 + 253

- **Timisoara**: 447 = 118 + 329
  - **Rimnicu Vilcea**: 449 = 75 + 374

- **Zerind**
  - **Bucharest**: 418 = 418 + 0
  - **Craiova**: 526 = 366 + 160
  - **Pitesti**: 553 = 300 + 253

- **Sibiu**
  - **Bucharest**: 615 = 455 + 160
  - **Craiova**: 607 = 414 + 193
Properties of A*

- Completeness
  - Yes
  - unless there are infinitely many nodes with $f(n) \leq f(G)$

- Time Complexity
  - it can be shown that the number of nodes grows exponentially unless the error of the heuristic $h(n)$ is bounded by the logarithm of the value of the actual path cost $h^*(n)$, i.e.

  $$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

- Space Complexity
  - keeps all nodes in memory
  - typically the main problem with A*

- Optimality
  - ???
  - → following pages
Admissible Heuristics

A heuristic is admissible if it *never overestimates* the cost to reach the goal.

- Formally:
  - $h(n) \leq h^*(n)$ if $h^*(n)$ are the true cost from $n$ to goal

- Example:
  - Straight-Line Distances $h_{SLD}$ are an admissible heuristics for actual road distances $h^*$

- Note:
  - $h(n) \geq 0$ must also hold, so that $h(\text{goal}) = 0$
Theorem

If $h(n)$ is admissible, $A^*$ using \textsc{tree-search} is optimal.

Proof:

Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$ with path cost $C^*$.

\[ C^* = g(n) + h^*(n) \]
\[ f(n) = g(n) + h(n) \leq C^* < f(G_2) \]
\[ h(n) \leq h^*(n) \]

because $h$ admissible

Suppose some suboptimal goal $G_2$ has been generated and is in the fringe.

$g(G_2) > C^*$

because $G_2$ suboptimal

\[ f(G_2) = g(G_2) \]

because $h(G_2) = 0$

$G_2$ will never be expanded, and $G$ will be returned

(holds for all goal nodes)
Consistent Heuristics

- Graph-Search discards new paths to repeated state even though the new path may be cheaper
  → Previous proof breaks down

- 2 Solutions
  1. Add extra bookkeeping to remove the more expensive path
  2. Ensure that optimal path to any repeated state is always followed first

- Requirement for Solution 2:

  A heuristic is **consistent** if for every node \( n \) and every successor \( n' \) generated by any action \( a \) it holds that

  \[
  h(n) \leq c(n, a, n') + h(n')
  \]
Lemma 1

Every consistent heuristic is admissible.

Proof Sketch:

for all nodes $n$, in which an action $a$ leads to goal $G$

$$h(n) \leq c(n, a, G) + h(G) = h^*(n)$$

by induction on the path length from goal, this argument can be extended to all nodes, so that eventually

$$\forall n : h(n) \leq h^*(n)$$

- Note:
  - not every admissible heuristic is consistent
  - but most of them are
    - it is hard to find non-consistent admissible heuristics
Lemma 2

If \( h(n) \) is consistent, then the values of \( f(n) \) along any path are non-decreasing.

Proof:

\[
\begin{align*}
     f(n) &= g(n) + h(n) \\
     &\leq g(n) + c(n, a, n') + h(n') = \\
     & g(n) + c(n, a, n') + h(n') = g(n') + h(n') = f(n')
\end{align*}
\]
Theorem

If $h(n)$ is consistent, $A^*$ is optimal.

**Proof:**

$A^*$ expands nodes in order of increasing $f$ value

Contour labelled $f_i$ contains all nodes with $f(n) < f_i$

Contours expand gradually
Cannot expand $f_{i+1}$ until $f_i$ is finished.

Eventually
- $A^*$ expands all nodes with $f(n) < C^*$
- $A^*$ expands some nodes with $f(n) = C^*$
- $A^*$ expands no nodes with $f(n) > C^*$

How would such contours look like for uniform-cost search?
Memory-Bounded Heuristic Search

- Space is the main problem with A*
- Some solutions to A* space problems
  (maintaining completeness and optimality)
  - Iterative-deepening A* (IDA*)
    - like iterative deepening
    - cutoff information is the $f$-cost ($g + h$) instead of depth
  - Recursive best-first search (RBFS)
    - recursive algorithm that attempts to mimic standard best-first search with linear space.
    - keeps track of the $f$-value of the best alternative path available from any ancestor of the current node
  - (Simple) Memory-bounded A* ((S)MA*)
    - drop the worst leaf node when memory is full
Admissible Heuristics: 8-Puzzle

- $h_{\text{MIS}}(n) =$ number of misplaced tiles
  - admissible because each misplaced tile must be moved at least once
- $h_{\text{MAN}}(n) =$ total Manhattan distance
  - i.e., no. of squares from desired location of each tile
  - admissible because this is the minimum distance of each tile to its target square

Example:

Start State

Goal State

$h_{\text{MIS}}(\text{start}) = 8$

$h_{\text{MAN}}(\text{start}) = 18$

$h^*(\text{start}) = 26$
Effective Branching Factor

- Evaluation Measure for a search algorithm:
  - assume we searched \( N \) nodes and found solution in depth \( d \)
  - the effective branching factor \( b^* \) is the branching factor of a uniform tree of depth \( d \) with \( N+1 \) nodes, i.e.
    \[
    1 + N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d
    \]

- Measure is fairly constant for different instances of sufficiently hard problems
  - Can thus provide a good guide to the heuristic’s overall usefulness.
  - A good value of \( b^* \) is 1
Efficiency of A* Search

- Comparison of number of nodes searched by A* and Iterative Deepening Search (IDS)
  - average of 100 different 8-puzzles with different solutions
  - Note: heuristic $h_2 = h_{MAN}$ is always better than $h_1 = h_{MIS}$

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2,45</td>
<td>1,79</td>
<td>1,79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2,87</td>
<td>1,48</td>
<td>1,45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2,73</td>
<td>1,34</td>
<td>1,30</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2,80</td>
<td>1,33</td>
<td>1,24</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2,79</td>
<td>1,38</td>
<td>1,22</td>
</tr>
<tr>
<td>12</td>
<td>3644035</td>
<td>227</td>
<td>73</td>
<td>2,78</td>
<td>1,42</td>
<td>1,24</td>
</tr>
<tr>
<td>14</td>
<td>–</td>
<td>539</td>
<td>113</td>
<td>–</td>
<td>1,44</td>
<td>1,23</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1,45</td>
<td>1,25</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1,46</td>
<td>1,26</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1,47</td>
<td>1,27</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1,48</td>
<td>1,28</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1,48</td>
<td>1,26</td>
</tr>
</tbody>
</table>
Dominance

If \( h_1 \) and \( h_2 \) are admissible, \( h_2 \) dominates \( h_1 \) if \( \forall n : h_2(n) \geq h_1(n) \)

- if \( h_2 \) dominates \( h_1 \) it will perform better because it will always be closer to the optimal heuristic \( h^* \)

- Example:
  - \( h_{\text{MAN}} \) dominates \( h_{\text{MIS}} \) because if a tile is misplaced, its Manhattan distance is \( \geq 1 \)

Theorem: (Combining admissible heuristics)

If \( h_1 \) and \( h_2 \) are two admissible heuristics than

\[
    h(n) = \max(h_1(n), h_2(n))
\]

is also admissible and dominates \( h_1 \) and \( h_2 \)
Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

Examples:
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_{\text{MIS}}$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_{\text{MAN}}$ gives the shortest solution.

Thus, looking for relaxed problems is a good strategy for inventing admissible heuristics.
Pattern Databases

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
  - This cost is a lower bound on the cost of the real problem.
- Pattern databases store the **exact solution** (length) for every possible **subproblem** instance
  - constructed once for all by searching backwards from the goal and recording every possible pattern
- **Example:**
  - store exact solution costs for solving 4 tiles of the 8-puzzle
  - sample pattern:

![Start State](image)

![Goal State](image)
Learning of Heuristics

- Another way to find a heuristic is through learning from experience
- Experience:
  - states encountered when solving lots of 8-puzzles
  - states are encoded using features, so that similarities between states can be recognized
- Features:
  - for the 8-puzzle, features could, e.g. be
    - the number of misplaced tiles
    - number of pairs of adjacent tiles that are also adjacent in goal
    - ...
- An inductive learning algorithm can then be used to predict costs for other states that arise during search.
- No guarantee that the learned function is admissible!
Summary

- Heuristic functions estimate the costs of shortest paths
- Good heuristics can dramatically reduce search costs
- Greedy best-first search expands node with lowest estimated remaining cost
  - incomplete and not always optimal
- A* search minimizes the path costs so far plus the estimated remaining cost
  - complete and optimal, also optimally efficient:
    - no other search algorithm can be more efficient, because they all have search the nodes with $f(n) < C^*$
    - otherwise it could miss a solution
- Admissible search heuristics can be derived from exact solutions of reduced problems
  - problems with less constraints
  - subproblems of the original problem