Artificial Intelligence Planning

• Informed Search
A* (Best-First) Search: Touring Romania
Overview

• Heuristic Search Strategies
• The A* Algorithm (for Tree Search)
• Properties of A*
• Graph Search with A*
• (Good) Heuristics
Best-First Search

- an instance of the general tree search (or graph) search algorithm
  - strategy: select next node based on an evaluation function \( f \) \( : \) state space \( \rightarrow \mathbb{R} \)
  - select node with lowest value \( f(n) \)
- implementation: selectFrom(\( fringe, strategy \))
  - priority queue: maintains fringe in ascending order of \( f \)-values

Best-First Search

- an instance of the general tree search (or graph) search algorithm
  - tree or graph search: both possible; difference only lies in test for repeated states
  - strategy: select next node based on an evaluation function \( f \): state space \( \rightarrow \mathbb{R} \)
    - evaluation function: determines the search strategy
    - intuition: choose function that estimates the distance to the goal
  - select node with lowest value \( f(n) \)
    - lowest \( f \)-value means best node: hence best-first search
- implementation: selectFrom(\( fringe, strategy \))
  - priority queue: maintains fringe in ascending order of \( f \)-values
    - implementation as binary tree: nodes can be added/retrieved in log-time (still expensive)
Heuristic Functions

- **heuristic function** $h$: state space $\rightarrow \mathbb{R}$
- $h(n)$ = estimated cost of the cheapest path from node $n$ to a goal node
- if $n$ is a goal node then $h(n)$ must be 0
- heuristic function encodes problem-specific knowledge in a problem-independent way

Heuristic Functions

- **heuristic function** $h$: state space $\rightarrow \mathbb{R}$
- $h(n)$ = estimated cost of the cheapest path from node $n$ to a goal node
- if $n$ is a goal node then $h(n)$ must be 0
- heuristic function encodes problem-specific knowledge in a problem-independent way
- difference between evaluation function and heuristic function:
  - good evaluation function makes sure nodes are expanded in an order that leads straight to the optimal solution
  - good heuristic function always gives the correct distance to the nearest goal node
  - evaluation function is not problem-specific, but uses heuristic function which is problem-specific
Greedy Best-First Search

• use heuristic function as evaluation function:

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

– always expands the node that is closest to the goal node
– eats the largest chunk out of the remaining distance, hence, “greedy”
Real-World Problem: Touring in Romania

• shown: rough map of Romania
• initial state: on vacation in Arad, Romania
• goal? actions? -- “Touring Romania” cannot readily be described in terms of possible actions, goals, and path cost
Touring in Romania: Heuristic

• \( h_{SLD}(n) = \) straight-line distance to Bucharest

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Hirsova</td>
<td>151</td>
<td>Rimnicu</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Iasi</td>
<td>226</td>
<td>Vilcea</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Lugoj</td>
<td>244</td>
<td>Sibiu</td>
</tr>
<tr>
<td>Dobret</td>
<td>242</td>
<td>Mehadia</td>
<td>241</td>
<td>Timisoara</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Neamti</td>
<td>234</td>
<td>Urziceni</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Oradea</td>
<td>380</td>
<td>Vaslui</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Pitesti</td>
<td>100</td>
<td>Zerind</td>
</tr>
</tbody>
</table>

Touring in Romania: Heuristic

• \( h_{SLD}(n) = \) straight-line distance to Bucharest
  • straight-line distance: Euclidean distance
  • distance to Bucharest because our goal is to be in Bucharest

• \( h_{SLD}(\text{Bucharest}) = 0 \)

• \( h_{SLD}(\text{Fagaras}) = 176 < 211 \) driving distance

• \( h_{SLD}(n) \) cannot be computed from the problem description, it represents additional information
Greedy Best-First Search: Touring Romania

• values are values of evaluation function = heuristic function
• select Arad; expand Arad
• select Sibiu; expand Sibiu
  • Fagharas has lowest $f$-value of all fringe nodes
• select Fagharas; expand Fagharas
• select Bucharest – goal node

• for this problem: search proceeds straight to the goal node:
  • minimal search cost
  • but not the optimal path
• uniform-cost search vs. greedy best-first search: both expand node with lowest number:
  • UCS: numbers start from 0 and increase – tendency to expand earlier nodes – breadth-first tendency
  • GBFS: number start from high and decreases – tendency to expand later nodes – depth-first tendency
Overview

• Heuristic Search Strategies
• The A* Algorithm (for Tree Search)
• Properties of A*
• Graph Search with A*
• (Good) Heuristics
A* Search

• best-first search where
  \( f(n) = h(n) + g(n) \)
  – \( h(n) \) the heuristic function (as before)
  – \( g(n) \) the cost to reach the node \( n \)
• evaluation function:
  \( f(n) = \) estimated cost of the cheapest solution through \( n \)
• A* search is optimal if \( h(n) \) is **admissible**
**A* (Best-First) Search: Touring Romania**

- initial state: in Arad; values shown are evaluation function $f(n) = h(n) + g(n)$
- select Arad; expand Arad
  - lowest f-value: Sibiu (393); means: possible path through Sibiu with cost 393
- select Sibiu; expand Sibiu
  - lowest f-value: Rimnicu Vilcea (413); means: possible path through Rimnicu Vilcea with cost 413
- select Rimnicu Vilcea; expand Rimnicu Vilcea
  - lowest f-value: Fagaras (415); expanding Rimnicu Vilcea showed f-value too optimistic
- select Fagaras; expand Fagaras
  - lowest f-value: Pitesti (417); expanding Fagaras showed f-value too optimistic
- select Pitesti; expand Pitesti
  - lowest f-value: Bucharest (418)
- select Bucharest
  - goal node test succeeds

- note: search cost not minimal as for GBFS but solution is optimal
The Eight-Puzzle

initial state

<table>
<thead>
<tr>
<th>7</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

goal state

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Heuristics for the Eight-Puzzle

- $h_1$: number of misplaced tiles
- $h_2$: Manhattan block distance

- example:
  - $h_1 = 8$
    all 8 tiles are misplaced
  - $h_2 = 3+1+2+2+2+3+3+2 = 18$

Heuristics for the Eight-Puzzle

- Both heuristics are admissible;
- Cost of the optimal solution: 26; both heuristics underestimate;
Overview

• Heuristic Search Strategies
• The A* Algorithm (for Tree Search)
• Properties of A*
• Graph Search with A*
• (Good) Heuristics
## Admissible Heuristics

A heuristic \( h(n) \) is admissible if it *never overestimates* the distance from \( n \) to the nearest goal node.

- example: \( h_{SLD} \)
- A* search: If \( h(n) \) is admissible then \( f(n) \) never overestimates the true cost of a solution through \( n \).

### Admissible Heuristics

- A heuristic \( h(n) \) is admissible if it *never overestimates* the distance from \( n \) to the nearest goal node.
  - admissible heuristics usually think the nearest goal node is closer than it actually is
  - example: \( h_{SLD} \)
    - \( h_{SLD} \): shortest distance between two point is straight line, hence \( h_{SLD} \) is admissible
  - A* search: If \( h(n) \) is admissible then \( f(n) \) never overestimates the true cost of a solution through \( n \).
    - since \( f(n) = h(n) + g(n) \) and \( g(n) \) is the exact cost of reaching \( n \), \( f(n) \) cannot overestimate the true cost of a solution through \( n \)
Optimality of A* (Tree Search)

Theorem:
A* using tree search is optimal if the heuristic $h(n)$ is admissible.

• Theorem: A* using tree search is optimal if the heuristic $h(n)$ is admissible.
  • reminder: optimal means finds a minimal-path cost solution
Completeness of A*: Contours

• **contours**: sets of states that can be reached within a certain cost
  – prerequisite for drawing contours: \( f \)-values along a path are non-decreasing
• A* fans out from the start node, adding nodes in concentric bands (contours) of increasing \( f \)-values
• A* is complete: it must reach a contour that includes a goal node

Completeness of A*: Contours
• **contours**: sets of states that can be reached within a certain cost
  • imagine like contours in a topographic map (with \( f \)-value instead of altitude)
  • **prerequisite for drawing contours**: \( f \)-values along a path are non-decreasing
• A* fans out from the start node, adding nodes in concentric bands (contours) of increasing \( f \)-values
  • uniform-cost search: draws circles
  • A* search: ellipsis stretch towards the goal nodes around the optimal path; the more accurate the heuristic, the more they stretch
• A* is complete: it must reach a contour that includes a goal node
  • each contour contains only a finite number of nodes because number of actions is finite and action cost must be greater than some positive value
Touring in Romania: Contours

- contour for $f=380$ only contains initial state
- contour for $f=400$ stretches towards the goal node
- contour for $f=420$ stretches towards and includes the goal node
A*: Optimally Efficient

- A* is optimally efficient for a given heuristic function: no other optimal algorithm is guaranteed to expand fewer nodes than A*.
- any algorithm that does not expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution

A*: Optimally Efficient

- A* is optimally efficient for a given heuristic function: no other optimal algorithm is guaranteed to expand fewer nodes than A*.
  - efficiency can still be increased with a different, more accurate heuristic for a given problem
  - but: efficiency does not only depend on number of nodes expanded
- any algorithm that does not expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution
  - suppose there is a node with $f(n) < C^*$ that is not expanded before a goal node
  - then there could be a path of cost with $f(n) < C^*$ through that node which would be better than the goal node found
Overview

• Heuristic Search Strategies
• The A* Algorithm (for Tree Search)
• Properties of A*
• Graph Search with A*
• (Good) Heuristics
**A* Tree/Graph Search Algorithm**

function aStarTreeSearch(problem, h)
  fringe ← priorityQueue(new searchNode(problem.initialState))
  allNodes ← hashTable(fringe)
  loop
    if empty(fringe) then return failure
    node ← selectFrom(fringe)
    if problem.goalTest(node.state) then
      return pathTo(node)
    for successor in expand(problem, node)
      if not allNodes.contains(successor) then
        fringe ← fringe + successor @ f(successor)
        allNodes.add(successor)

**A* Tree/Graph Search Algorithm**

• steps in grey for graph search
• also needed: detecting short-cuts if heuristic is not admissible
A* and Exponential Space

- A* has worst case time and space complexity of $O(b^l)$
- exponential growth of the fringe is normal
  - exponential time complexity may be acceptable
  - exponential space complexity will exhaust any computer’s resources all too quickly

- and with the memory exhausted A* cannot continue and fails – no solution will be found
The Eight-Puzzle Search Space
Permutations of Solutions

• independent actions: for all states \( s \):
  \[ \gamma(\gamma(s,a_1),a_2) = \gamma(\gamma(s,a_2),a_1) \]

• worst case:

\[ s_i \rightarrow \ldots \rightarrow s_n \]

optimal path: \( n \) independent actions

contour for \( n-1 \)

Permutations of Solutions

• problem: \( (n-1)! \) different paths in contour
• A* may explore all these nodes before exploring \( s_g \) (all optimal, so heuristic does not help)
• note: many puzzles do not have independent actions; real-world problems often do
Overview

- Heuristic Search Strategies
- The A* Algorithm (for Tree Search)
- Properties of A*
- Graph Search with A*
- (Good) Heuristics
Heuristics

- **heuristics** are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal
- example: ripe pineapple
  - inner leafs rip out easily
  - fruit smells like pineapple

**Heuristics**

- Heuristic (colloquial): **a rule of thumb**;
- Heuristic (general term) vs. heuristic function (technical term) \( h(n) \) = estimated cost of the cheapest path from node \( n \) to a goal node;
- Another example: what made you (the student) choose this course over others?
Good Heuristics

- good heuristics
  - indicate a way to reduce the number of states that need to be evaluated
  - help obtain solutions within reasonable time constraints
- trade-off:
  - simplicity
    - must provide a simple means of discriminating between choices
  - accuracy
    - no guarantee that they identify the best course of actions
    - but they should do so sufficiently often

Good Heuristics

- Simplicity: easy to compute;
- Accuracy: should result in the best course of actions;
Finding Good Heuristics

• How can we find good heuristics for a given problem?
• Can this process be automated?

Finding Good Heuristics

• Automation would require problem description in a formal language, of course.
Heuristics from Simplified Problems

• relaxed problem: a problem with fewer restrictions on the actions than the original problem
• *The cost of an optimal solution for a relaxed problem is an admissible and consistent heuristic for the original problem.*

Heuristics from Simplified Problems

• Admissibility:
  • Optimal solution to original problem is also solution to relaxed problem;
  • Therefore: optimal solution to original problem at least as expensive as solution to relaxed problem;

• Consistency:
  • Because derived heuristic is exact cost for relaxed problem (finds “short cuts”) the triangle inequality must hold;

• ABSOLVER: program that generates heuristics based on the relaxed problem method;
  • Found best heuristic for 8-puzzle and first useful heuristic for Rubik’s cube;
8-Puzzle Actions

- a tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank
- Relaxed conditions:
  - a tile can move from square A to square B if A is adjacent to B (⇒ Manhattan distance)
  - a tile can move from square A to square B if B is blank
  - a tile can move from square A to square B (⇒ misplaced tiles)

8-Puzzle Actions

• Heuristics are estimates for costs of actions still expected, hence modify constraints on actions
Overview

- Heuristic Search Strategies
- The A* Algorithm (for Tree Search)
- Properties of A*
- Graph Search with A*
- (Good) Heuristics