Uninformed search methods II.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Uninformed methods

- Uninformed search methods use only information available in the problem definition
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Iterative deepening (IDA)
  - Bi-directional search
- For the minimum cost path problem:
  - Uniform cost search
Minimum cost path search

Traveler example with distances [km]

Optimal path: the shortest distance path between the initial and destination city

Searching for the minimum cost path

• General minimum cost path-search problem:
  – adds weights or costs to operators (links)

• Search strategy:
  – “Intelligent” expansion of the search tree should be driven by the cost of the current (partially) built path

• Implementation:
  – Path cost function for node $n$: $g(n)$
    • length of the path represented by the search tree branch starting at the root of the tree (initial state) to $n$
  – Search strategy:
    • Expand the leaf node with the minimum $g(n)$ first
    • Can be implemented by the priority queue
Searching for the minimum cost path

- The basic algorithm for finding the minimum cost path:
  - Dijkstra’s shortest path

- In AI, the strategy goes under the name
  - Uniform cost search

- Note:
  - When operator costs are all equal to 1 the uniform cost search is equivalent to the breadth first search BFS

Uniform cost search

- Expand the node with the minimum path cost first
- Implementation: a priority queue
Uniform cost search

Arad

Zerind 75
Sibiu 140
Timisoara 118

queue

Zerind 75
Timisoara 118
Sibiu 140

Uniform cost search

Arad

Zerind 75
Sibiu 140
Timisoara 118

queue

Timisoara 118
Sibiu 140
Oradea 146
Arad 150

150 146
Properties of the uniform cost search

- **Completeness**: Yes, assuming that operator costs are non-negative (the cost of path never decreases)
  \[ g(n) \leq g(\text{successor}(n)) \]
- **Optimality**: Yes. Returns the least-cost path.

- **Time complexity**: number of nodes with the cost \( g(n) \) smaller than the optimal cost

- **Memory (space) complexity**: number of nodes with the cost \( g(n) \) smaller than the optimal cost
Elimination of state repeats

Idea:
- A node is redundant and can be eliminated if there is another node with exactly the same state and a shorter path from the initial state

Assuming positive costs:
- If the state was already expanded, is there a shorter path to that node?
- No!

Implementation:
- Marking with the hash table

Informed (heuristic) search methods
Additional information to guide the search

• **Uninformed search methods**
  – use only the information from the problem definition; and
  – past explorations, e.g. cost of the path generated so far

• **Informed search methods**
  – incorporate additional measure of a potential of a specific state to reach the goal
  – a potential of a state (node) to reach a goal is measured by a **heuristic function**

• Heuristic function is denoted \( h(n) \)

---

**Evaluation-function driven search**

• A search strategy can be defined in terms of a **node evaluation function**
  – Similarly to the path cost for the uniform cost search

• **Evaluation function**
  – Denoted \( f(n) \)
  – Defines the **desirability of a node to be expanded next**

• **Evaluation-function driven search:**
  – **expand the node (state) with the best evaluation-function value**

• **Implementation:**
  – **priority queue** with nodes in the decreasing order of their evaluation function value
**Uniform cost search**

- **Uniform cost search (Dijkstra’s shortest path):**
  - A special case of the evaluation-function driven search
  \[ f(n) = g(n) \]
- **Path cost function** \( g(n) \);
  - path cost from the initial state to \( n \)
- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - **weights or costs** associated with operators (links).
- **Note:** Uniform cost search relies on the problem definition only
  - It is an uninformed search method

---

**Best-first search**

**Best-first search**

- incorporates a **heuristic function**, \( h(n) \), into the evaluation function \( f(n) \) to guide the search.

**Heuristic function:**

- Measures a potential of a state (node) to reach a goal
- Typically in terms of some distance to a goal estimate

**Example of a heuristic function:**

- Assume a shortest path problem with city distances on connections
- Straight-line distances between cities give additional information we can use to guide the search
Example: traveler problem with straight-line distance information

- **Straight-line distances** give an estimate of the cost of the path between the two cities

---

Best-first search

Best-first search

- incorporates a **heuristic function**, $h(n)$, into the evaluation function $f(n)$ to guide the search.

- **heuristic function**: measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):

- **Greedy search**
  \[ f(n) = h(n) \]

- **A* algorithm**
  \[ f(n) = g(n) + h(n) \]

+ **iterative deepening** version of A*: **IDA***
Greedy search method

- Evaluation function is equal to the heuristic function
  \[ f(n) = h(n) \]
- **Idea:** the node that seems to be the closest to the goal is expanded first

Greedy search

- Evaluation function is equal to the heuristic function
  \[ f(n) = h(n) \]
- **Queue:** the node with the lowest estimated cost is expanded first
Greedy search

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

```
Arad
\( 366 \)
\- Zerind 75
\- Sibiu 140
\- Timisoara 118

Zerind 374
Sibiu 253
Timisoara 329
```

The algorithm starts with Arad in the queue and continues by selecting the node with the smallest cost function value until the goal is reached.

Greedy search

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

```
Arad
\( 366 \)
\- Zerind 75
\- Sibiu 140
\- Timisoara 118

Zerind 374
Sibiu 253
Timisoara 329
```

Fagaras 178
Rimnicu V. 193
Timisoara 329
Arad 366
Zerind 374
Oradea 380

The algorithm starts with Arad in the queue and continues by selecting the node with the smallest cost function value until the goal is reached.
Properties of greedy search

- **Completeness:** No.
  
  We can loop forever. Nodes that seem to be the best choices can lead to cycles. Elimination of state repeats can solve the problem.

- **Optimality:** ?

- **Time complexity:**

- **Memory (space) complexity:**
Example: traveler problem with straight-line distance information

• Greedy search result

Total: 450

Example: traveler problem with straight-line distance information

• Greedy search and optimality

Total: 418
Properties of greedy search

- **Completeness:** No.
  We can loop forever. Nodes that seem to be the best choices can lead to cycles. Elimination of state repeats can solve the problem.

- **Optimality:** No.
  Even if we reach the goal, we may be biased by a bad heuristic estimate. **Evaluation function disregards the cost of the path built so far.**

- **Time complexity:** $O(b^m)$
  Worst case !!! But often better!

- **Memory (space) complexity:** $O(b^m)$
  Often better!

A* search

- The problem with the **greedy search** is that it can keep expanding paths that are already very expensive.
- The problem with the **uniform-cost search** is that it uses only past exploration information (path cost), no additional information is utilized

- **A* search**
  
  \[
  f(n) = g(n) + h(n)
  \]
  
  - $g(n)$ - cost of reaching the state
  - $h(n)$ - estimate of the cost from the current state to a goal
  - $f(n)$ - estimate of the path length

- **Additional A* condition:** admissible heuristic
  \[
  h(n) \leq h^*(n) \quad \text{for all } n
  \]
A* search example

Arad 366

queue

Arad 366

A* search example

Arad 366

queue

Sibiu 393
Timisoara 447
Zerind 449
A* search example

CS 1571 Intro to AI  
M. Hauskrecht
A* search example

Properties of A* search

- Completeness: Yes.
- Optimality: ?
- Time complexity: – ?
- Memory (space) complexity: – ?
Optimality of A*

- In general, a heuristic function $h(n)$:
  - It can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- Is the A* optimal for an arbitrary heuristic function?

Example: traveler problem with straight-line distance information

- Admissible heuristics

<table>
<thead>
<tr>
<th>City</th>
<th>Distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>151</td>
</tr>
<tr>
<td>Craiova</td>
<td>80</td>
</tr>
<tr>
<td>Dobroia</td>
<td>75</td>
</tr>
<tr>
<td>Iasi</td>
<td>146</td>
</tr>
<tr>
<td>Iasi</td>
<td>67</td>
</tr>
<tr>
<td>Neamt</td>
<td>92</td>
</tr>
<tr>
<td>Pitesti</td>
<td>142</td>
</tr>
<tr>
<td>Rimnicu-Valea</td>
<td>211</td>
</tr>
<tr>
<td>Sibiu</td>
<td>99</td>
</tr>
<tr>
<td>Sibi</td>
<td>60</td>
</tr>
<tr>
<td>Tulcea</td>
<td>170</td>
</tr>
<tr>
<td>Tulcea</td>
<td>190</td>
</tr>
<tr>
<td>Vaslui</td>
<td>98</td>
</tr>
<tr>
<td>Vatra-Dornei</td>
<td>88</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

Strict-line distance to Bucharest

Arad 151
Bucharest 0
Craiova 80
Dobroia 75
Iasi 146
Iasi 67
Neamt 92
Pitesti 142
Rimnicu-Valea 211
Sibiu 99
Tulcea 170
Vaslui 98
Vatra-Dornei 88
Zerind 374

overestimate
**Example: traveler problem with straight-line distance information**

- Admissible heuristics

\[
f(n) = 220 + 400 = 620
\]

\[
f(n) = 239 + 178 = 417
\]

\[
f(n) = 220 + 400 = 620
\]

\[
f(n) = 211 + 142 = 353
\]

\[
f(n) = 450
\]

**Total path: 450 is suboptimal**
Optimality of A*

- In general, a heuristic function $h(n)$:
  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- Is the A* optimal for an arbitrary heuristic function?
  - No!

Optimality of A*

- In general, a heuristic function $h(n)$:
  Can overestimate, be equal or underestimate the true distance of a node to the goal $h^*(n)$
- **Admissible heuristic condition**
  - Never overestimate the distance to the goal !!!
    
    $h(n) \leq h^*(n)$ for all $n$

**Example**: the straight-line distance in the travel problem never overestimates the actual distance

Is A* search with an admissible heuristic is optimal ??
Optimality of A* (proof)

- Let G1 be the optimal goal (with the minimum path distance). Assume that we have a sub-optimal goal G2. Let n be a node that is on the optimal path and is in the queue together with G2.

Then: \[ f(G2) = g(G2) \quad \text{since} \quad h(G2) = 0 \]
\[ > g(G1) \quad \text{since} \quad G2 \text{ is suboptimal} \]
\[ \geq f(n) \quad \text{since} \quad h \text{ is admissible} \]

And thus A* never selects G2 before n.

Properties of A* search

- Completeness: Yes.
- Optimality: Yes (with the admissible heuristic)
- Time complexity:
  - Order roughly the number of nodes with \( f(n) \) smaller than the cost of the optimal path \( g^* \)
- Memory (space) complexity:
  - Same as time complexity (all nodes in the memory)