Informed search methods.

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

Announcements

- Homework assignment 1 due today
- Homework assignment 2 is out
  - Due on Wednesday, September 27, 2006
  - Two parts:
    - Pen and pencil part
    - Programming part – heuristics for (Puzzle 8)

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs1571/
Evaluation-function driven search

- A search strategy can be defined in terms of a node evaluation function
- **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

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

Queue

Arad 366

Greedy search

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

Queue

Arad 366

Zerind 374 Sibiu 253 Timisoara 329

Zerind 374

75 140 118

Sibiu 253

Timisoara 329

Zerind 374
Greedy search

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

CS 1571 Intro to AI
M. Hauskrecht

\begin{itemize}
  \item Arad
  \item Zerind
  \item Sibiu
  \item Timisoara
  \item Bucharest
  \item Fagaras
  \item Rimnicu V.
  \item Arad
  \item Sibiu
  \item Timisoara
  \item Oradea
\end{itemize}
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!
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
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
  \[ h(n) \leq h^*(n) \text{ for all } n \]

A* search example

\[ f(n) \]

Arad 366

queue

Arad 366
A* search example

\[ f(n) \]

queue

Sibiu 393
Timisoara 447
Zerind 449

CS 1571 Intro to AI
M. Hauskrecht

A* search example

\[ f(n) \]

queue

Rimnicu V. 413
Fagaras 417
Timisoara 447
Zerind 449
Oradea 526
Arad 646

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

\[ f(n) \]

\[ \text{queue} \]

\[ \text{Pitesti 415} \]
\[ \text{Fagaras 417} \]
\[ \text{Timisoara 447} \]
\[ \text{Zerind 449} \]
\[ \text{Oradea 526} \]
\[ \text{Craiova 526} \]
\[ \text{Sibiu 553} \]
\[ \text{Arad 646} \]
A* search example

- **Queue**
  - Bucharest: 418
  - Timisoara: 447
  - Zerind: 449
  - Oradea: 526
  - Craiova: 526
  - Sibiu: 553
  - **Rimnicu V.**: 607
  - Arad: 646

- **Goal**

Graph showing the A* search example with nodes and edges connecting cities like Arad, Zerind, Sibiu, Timisoara, and others, along with their values.
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>166</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>140</td>
</tr>
<tr>
<td>Dobroța</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Făgăraș</td>
<td>176</td>
</tr>
<tr>
<td>Giur˘gia</td>
<td>77</td>
</tr>
<tr>
<td>Hlășova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</td>
<td>225</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Orașenii</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>32</td>
</tr>
<tr>
<td>Râmnicu Vâlcea</td>
<td>400</td>
</tr>
<tr>
<td>Sălaj</td>
<td>353</td>
</tr>
<tr>
<td>Târgovița</td>
<td>329</td>
</tr>
<tr>
<td>Uzunți</td>
<td>140</td>
</tr>
<tr>
<td>Văcărești</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: traveler problem with straight-line distance information

- Admissible heuristics

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

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

- Admissible heuristics

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 !!!

\[ h(n) \leq h^*(n) \quad \text{for all } n \]

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: – ?
- Memory (space) complexity: – ?
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)

Admissible heuristics

- Heuristics are designed based on relaxed version of problems
- Example: the 8-puzzle problem

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

- Admissible heuristics:
  1. number of misplaced tiles
  2. Sum of distances of all tiles from their goal positions (Manhattan distance)
Admissible heuristics

**Heuristics 1:** number of misplaced tiles

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

\[ h(n) \text{ for the initial position: } ? \]
Admissible heuristics

- **Heuristic 2:** Sum of distances of all tiles from their goal positions (Manhattan distance)

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

$h(n)$ for the initial position:

\[ 2 + 3 + 3 + 1 + 1 + 2 + 0 + 2 = 14 \]

For tiles: 1 2 3 4 5 6 7 8
Admissible heuristics

- We can have multiple admissible heuristics for the same problem
- **Dominance:** Heuristic function $h_1$ dominates $h_2$ if
  \[ \forall n \quad h_1(n) \geq h_2(n) \]
- **Combination:** two or more admissible heuristics can be combined to give a new admissible heuristic
  - Assume two admissible heuristics $h_1, h_2$
  
  Then: \[ h_3(n) = \max(h_1(n), h_2(n)) \]
  
  is admissible