CS 1571 Introduction to AI
Lecture 6

Informed search methods

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

Announcements

Homework assignment 2 is out
• Due on Tuesday, September 24, 2013 before the class
• Two parts:
  – Pen and pencil part
  – Programming part (Puzzle 8): informed search methods

Course web page:
  http://www.cs.pitt.edu/~milos/courses/cs1571/
Search methods

- Uninformed search methods
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Iterative deepening (IDA)
  - Bi-directional search
  - Uniform cost search
- Informed (or heuristic) search methods:
  - Best first search with the heuristic function

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

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

queue → Arad

Arad 366

Greedy search

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

queue → Sibiu

Sibiu 253
Timisoara 329
Zerind 374
Greedy search

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

![Greedy search diagram](image_url)

Greedy search

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

![Greedy search diagram](image_url)

Goal !!!
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

Example: traveler problem with straight-line distance information

• Greedy search and optimality
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

\[ f(n) \]

- Arad 366
- queue
- Arad 366

A* search example

\[ f(n) \]

- Arad 366
- Zerind 75 140 118
- Timisoara 449 393 447
- Sibiu 393
- Timisoara 447
- Zerind 449
A* search example

Arad → Zerind: 449
Arad → Sibiu: 366
Sibiu → Timisoara: 447
Sibiu → Pitesti: 417
Sibiu → Oradea: 526
Zerind → Fagaras: 393
Zerind → Rimnicu: 413

f(n)
queue

Rimnicu: 413
Fagaras: 417
Timisoara: 447
Zerind: 449
Oradea: 526
Arad: 646
A* search example

Properties of A* search

- Completeness: ?
- Optimality: ?
- Time complexity: – ?
- Memory (space) complexity: – ?
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

overestimate
Example: traveler problem with straight-line distance information

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 !!!

$$h(n) \leq h^*(n) \text{ for all } n$$

**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 $G2$

$$f(G2) = g(G2) \text{ since } h(G2) = 0$$
$$> g(G1) \text{ since } G2 \text{ is suboptimal}$$
$$\geq f(n) \text{ since } h \text{ is admissible}$$

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)
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><img src="initial_position.png" alt="Initial Position" /></td>
<td><img src="goal_position.png" alt="Goal Position" /></td>
</tr>
</tbody>
</table>

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

Heuristics 1: number of misplaced tiles

<table>
<thead>
<tr>
<th>Initial position</th>
<th>Goal position</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="initial_position.png" alt="Initial Position" /></td>
<td><img src="goal_position.png" alt="Goal Position" /></td>
</tr>
</tbody>
</table>

$h(n)$ for the initial position: ?
### 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) for the initial position: 7

---

**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:
Admissible heuristics

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

  \[
  \text{Initial position} \hspace{1cm} \text{Goal position}
  \begin{array}{cccc}
  4 & 5 & \text{Goal} & \text{Goal} \\
  6 & 1 & 8 & 1 \\
  7 & 3 & 2 & 2 \\
  \end{array}
  \begin{array}{cccc}
  1 & 2 & 3 & \text{Goal} \\
  4 & 5 & 6 & \text{Goal} \\
  7 & 8 & \text{Goal} & \text{Goal} \\
  \end{array}
  \]

  \(h(n)\) 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
  \[\forall n \ h_1(n) \geq h_2(n)\]
- **Combination:** two or more admissible heuristics can be combined to give a new admissible heuristics
  - Assume two admissible heuristics \(h_1, h_2\)
    
    Then:
    \[h_3(n) = \max(\ h_1(n), h_2(n))\]
    is admissible