Lecture 5: The "animal kingdom" of heuristics: Admissible, Consistent, zero, Relaxed, Dominant

Mark Hasegawa-Johnson, January 2020
With some slides by Svetlana Lazebnik, 9/2016
Distributed under CC-BY 3.0
Outline of lecture

1. Admissible heuristics
2. Consistent heuristics
3. The zero heuristic: Dijkstra’s algorithm
4. Relaxed heuristics
5. Dominant heuristics
A* Search

**Definition: A* SEARCH**

- If $h(n)$ is **admissible** ($d(n) \geq h(n)$), and
- if the frontier is a priority queue sorted according to $g(n) + h(n)$, then
- the FIRST path to goal uncovered by the tree search, path $m$, is guaranteed to be the **SHORTEST** path to goal

\[(h(n) + g(n) \geq c(m) \text{ for every node } n \text{ that is not on path } m)\]
Bad interaction between A* and the explored set

Frontier
S: g(n)+h(n)=2, parent=none

Explored Set

Select from the frontier: S
Bad interaction between A* and the explored set

Frontier
A: $g(n)+h(n)=5$, parent=S
B: $g(n)+h(n)=2$, parent=S

Explored Set
S

Select from the frontier: B
Bad interaction between A* and the explored set

Frontier
A: $g(n)+h(n)=5$, parent=S
C: $g(n)+h(n)=4$, parent=B

Explored Set
S, B

Select from the frontier: C
Bad interaction between A* and the explored set

Frontier
A: \( g(n) + h(n) = 5 \), parent = S
G: \( g(n) + h(n) = 6 \), parent = C

Explored Set
S, B, C

Select from the frontier: A
Bad interaction between A* and the explored set

Frontier
G: $g(n)+h(n)=6$, parent=C

- Now we would place C in the frontier, with parent=A and $h(n)+g(n)=3$, except that C was already in the explored set!

Explored Set
S, B, C

Select from the frontier: Would be C, but instead it’s G
Bad interaction between A* and the explored set

Return the path S, B, C, G
Path cost = 6

OOPS
Bad interaction between A* and the explored set: Three possible solutions

1. Don’t use an explored set
   - This option is OK for any finite state space, as long as you check for loops.

2. Nodes on the explored set are tagged by their $h(n)+g(n)$. If you find a node that’s already in the explored set, test to see if the new $h(n)+g(n)$ is smaller than the old one.
   - If so, put the node back on the frontier
   - If not, leave the node off the frontier

3. Use a heuristic that’s not only admissible, but also consistent.
Outline of lecture

1. Admissible heuristics
2. Consistent heuristics
3. The zero heuristic: Dijkstra’s algorithm
4. Relaxed heuristics
5. Dominant heuristics
**Definition:** A **consistent heuristic** is one for which, for every pair of nodes in the graph, 
\[ d(n) - d(p) \geq h(n) - h(p). \]

In words: the **distance between any pair of nodes** is **greater than or equal to** the **difference in their heuristics.**
A* with an inconsistent heuristic

Frontier
A: g(n)+h(n)=5, parent=S
C: g(n)+h(n)=4, parent=B

Explored Set
S, B

Select from the frontier: C
A* with a **consistent** heuristic

**Frontier**
- A: \( g(n) + h(n) = 2 \), parent = S
- C: \( g(n) + h(n) = 4 \), parent = B

**Explored Set**
- S, B

Select from the frontier: **A**
A* with a **consistent** heuristic

Frontier
.
C: \(g(n)+h(n)=2\), parent=\(A\)

Explored Set
S, B, \(A\)

Select from the frontier: \(C\)
A* with a **consistent** heuristic

Frontier

\[ G: g(n) + h(n) = 5, \text{ parent}=C \]

Explored Set

S, B, A, C

Select from the frontier: G

![Diagram](image_url)
Bad interaction between A* and the explored set: Three possible solutions

1. Don’t use an explored set.

   This works for the MP!

2. If you find a node that’s already in the explored set, test to see if the new $h(n)+g(n)$ is smaller than the old one.

   Most students find that this is the most computationally efficient solution to the multi-dots problem.

3. Use a consistent heuristic.

   Do this too. Consistent: heuristic difference $\leq$ actual distance between two nodes. It’s easy to do, because $0 \leq d$. 
Outline of lecture

1. Admissible heuristics
2. Consistent heuristics
3. The zero heuristic: Dijkstra’s algorithm
4. Relaxed heuristics
5. Dominant heuristics
The trivial case: $h(n)=0$

- A heuristic is **admissible** if and only if $d(n) \geq h(n)$ for every $n$.
- A heuristic is **consistent** if and only if $d(n, p) \geq h(n) - h(p)$ for every $n$ and $p$.
- Both criteria are satisfied by $h(n) = 0$. 
Dijkstra = A* with h(n)=0

• Suppose we choose \( h(n) = 0 \)
• Then the frontier is a priority queue sorted by
  \[ g(n) + h(n) = g(n) \]
• In other words, the first node we pull from the queue is the one that’s closest to START!! (The one with minimum \( g(n) \)).
• So this is just Dijkstra’s algorithm!
Outline of lecture

1. Admissible heuristics
2. Consistent heuristics
3. The zero heuristic: Dijkstra’s algorithm
4. Relaxed heuristics
5. Dominant heuristics
Designing heuristic functions

Now we start to see things that actually resemble the multi-dot problem...

• Heuristics for the 8-puzzle
  \[ h_1(n) = \text{number of misplaced tiles} \]
  \[ h_2(n) = \text{total Manhattan distance (number of squares from desired location of each tile)} \]

\[ h_1(\text{start}) = 8 \]
\[ h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18 \]

• Are \( h_1 \) and \( h_2 \) admissible?
Heuristics from 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

• If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution

• If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
Heuristics from subproblems
This is also a trick that many students find useful for the multi-dot problem.

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions
- Can precompute and save the exact solution cost for every possible subproblem instance – *pattern database*
- If the subproblem is $O(9^4)$, and the full problem is $O(9^9)$, then you can solve as many as $9^5$ subproblems without increasing the complexity of the problem!!

---

![Start State](image1.png) ![Goal State](image2.png)
Outline of lecture

1. Admissible heuristics
2. Consistent heuristics
3. The zero heuristic: Dijkstra’s algorithm
4. Relaxed heuristics
5. Dominant heuristics
Dominance

- If $h_1$ and $h_2$ are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all $n$, (both admissible) then $h_2$ dominates $h_1$

- Which one is better for search?
  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  - Therefore, A* search with $h_1$ will expand more nodes = $h_1$ is more computationally expensive.
Dominance

• Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths):

  • $d=12$  
    BFS expands 3,644,035 nodes  
    $A^*(h_1)$ expands 227 nodes  
    $A^*(h_2)$ expands 73 nodes  

  • $d=24$  
    BFS expands 54,000,000,000 nodes  
    $A^*(h_1)$ expands 39,135 nodes  
    $A^*(h_2)$ expands 1,641 nodes
Combining heuristics

• Suppose we have a collection of admissible heuristics $h_1(n), h_2(n), ..., h_m(n)$, but none of them dominates the others

• How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), ..., h_m(n)\}$$
All search strategies. $C^*$=cost of best path.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete?</th>
<th>Optimal?</th>
<th>Time complexity</th>
<th>Space complexity</th>
<th>Implement the Frontier as a...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
<td>Queue</td>
</tr>
<tr>
<td>DFS</td>
<td>No</td>
<td>No</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
<td>Stack</td>
</tr>
<tr>
<td>UCS</td>
<td>Yes</td>
<td>Yes</td>
<td>Number of nodes w/ $g(n) \leq C^*$</td>
<td>Number of nodes w/ $g(n) \leq C^*$</td>
<td>Priority Queue sorted by $g(n)$</td>
</tr>
</tbody>
</table>
| Greedy    | No        | No       | Worst case: $O(b^m)$  
Best case: $O(bd)$ | Worse case: $O(b^m)$  
Best case: $O(bd)$ | Priority Queue sorted by $h(n)$ |
| A*        | Yes       | Yes      | Number of nodes w/ $g(n)+h(n) \leq C^*$ | Number of nodes w/ $g(n)+h(n) \leq C^*$ | Priority Queue sorted by $h(n)+g(n)$ |