3. Search (Part 2)

CS 3030 Lecture Notes
Yan Shi
UW-Platteville

Read: Textbook Chapter 3.7-3.9, 3.12, 4.
Heuristic Search

Problem with uninformed search:

- It ignores the domain knowledge
- Thus expand unnecessary states

Solution:

- introduce a heuristic ⇐ a "rule-of-thumb" that helps you decide when you're closer to a goal
- Example: How to solve a rubik’s cube?
Heuristics for TSP (1)

- Nearest neighbor heuristic algorithm (NN):
  - choose the nearest unvisited city as the next move
  - improved: you can generate n cycles, choose the one with the smallest path cost.

- ABDCEA: 650 miles
Heuristics for TSP (2)

- Greedy heuristic algorithm:
  - gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as
    - it doesn’t create a cycle with less than N edges, or
    - increases the degree of any node to more than 2.
  - ABDCEA: 650 miles

![Graph with cities and distances]
Heuristic Function

- **Heuristic Function**
  - \( h(n) = \) estimated cost of the cheapest path from the state at node \( n \) to a goal state

- **Shortest Path Cost Function**
  - \( g(n) = \) shortest path cost from the initial state to the state at node \( n \).

- **Evaluation Function**
  - \( f(n) = \) cost estimate used during search

- \( h(n) \) looks forward, while \( g(n) \) looks backwards.
8-puzzle Example

- Heuristic 1: count how many tiles are in the wrong place
  - \( h_1(n) = 6 \)

- Heuristic 2: sum how far each tile had to move to get to its correct state (sum of Manhattan distances)
  - \( h_2(n) = 2+0+3+1+1+1+3+4=15 \)
A heuristic is *admissible* if it *never overestimate* the cost of changing from a given state to the goal state.
- This is essential to finding a solution.
- Are the previous heuristics admissible?

**h2 dominates** h1 if h2(node) >= h1(node)
- this means h2 is *more informed* than h1, so h2 always performs more efficiently than h1.
- *This is one important criteria to choose heuristics!*

How to come use with admissible heuristic is the key to develop heuristic search methods!
- This is usually done by solving relaxed problems where new actions are allowed.
Best-Frist Search

- Choose the “best” state to expand.

```
Open ← [Start]  // states to be considered
Closed ← []   // states that have been considered
while open != []
    Next ← first(Open)
    Open ← rest(Open)
    if isgoal(Next), return SUCCESS
    let Kids = children(Next)-(Open ∪ Closed ∪ [Next])
    Closed ← Closed ∪ [Next]
    Open ← sort(append Kids, Open)
end-while
return FAIL
```

- How to decide “best’?
  - evaluation function \( f(n) \)
More on Best-First Search

- Compare to Depth-First, Breadth-First?
  - Open ← sort(append Kids, Open)  // only difference

- Do we really need to sort?
  - No, just insert the new nodes to an ordered list

- How do we record the path?
  - store parent of each node in the lists Open and Closed.

- Is uniform-cost search an instance of best-first search?
  - Yes if you consider the path cost function as the evaluation function.

- When \( f(n) = h(n) \), it is called **greedy best-first search**
Hill Climbing

- Expand to the **first** successor state that is “better” than the current node.
- **Steepest Ascent Hill Climbing**: Expand to the successor “best” state.
- Example: 1->6

![Graph showing the hill climbing procedure with nodes and distances]


red number represents the straight line distance to node 6
Hill Climbing

- Greedy local search: grab good neighbor without thinking ahead

- Problem:
  - can get stuck at a local maximum, plateau or a ridge!
  - solution: random restart or re-model the problem!

- Steepest Ascent Hill Climbing vs Greedy Best-First Search:
  - no backtracking!
Beam Search

- Breath-first search that uses a heuristic: only expand the best few successors.

- Pros:
  - efficient in memory usage so good for high branching factor search space

- Cons:
  - not exhaustive search ➔ may fail ➔ not complete
A* Search

- Uniform-cost orders by path cost, or *backward cost* $g(n)$
- Greedy best-first orders by goal proximity, or *forward cost* $h(n)$
- they both ignore one side of the cost!

- order by the sum: $f(n) = g(n) + h(n)$
  - *A* search if $h(n)$ is admissible
  - *A* search if $h(n)$ is not admissible
A* Search Example

- $f(n) = g(n) + h(n)$: $S \rightarrow A \rightarrow D \rightarrow C \rightarrow G$, path cost is 16!

<table>
<thead>
<tr>
<th>Open</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S, -, 0, 16)</td>
<td></td>
</tr>
<tr>
<td>(A, S, 7, 8) (B, S, 10, 6)</td>
<td>(S, -, 0, 16)</td>
</tr>
<tr>
<td>(D, A, 13, 1) (C, D, 14, 2) (B, S, 10, 6)</td>
<td>(S, -, 0, 16) (A, S, 7, 8)</td>
</tr>
<tr>
<td>(G, C, 16, 0) (C, D, 14, 2) (B, S, 10, 6)</td>
<td>(S, -, 0, 16) (A, S, 7, 8) (D, A, 13, 1)</td>
</tr>
<tr>
<td>(C, D, 14, 2) (B, S, 10, 6)</td>
<td>(S, -, 0, 16) (A, S, 7, 8) (D, A, 13, 1)</td>
</tr>
<tr>
<td>(G, C, 16, 0)</td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>
What’s wrong with the following graph?

- h is not admissible!
- S->B->D->C->G ➜ 17
Properties of A* Search

- A* is **complete**, if
  - the search tree has a finite branching factor
  - every operator adds cost by at least $\delta > 0$ (*consistent*)
- A* is **optimal**: It always find the shortest path cost solution!
  - We can use proof by contradiction
- A* is **optimally efficient**! It will expand the fewest possible paths to find the goal!
- The complexity of A* depends on the heuristic
  - Worst case: $O(b^d)$ with constant heuristic
Implementing A* Search

- Problem: need to calculate g(n)
  - It is a search problem itself!
- Solution: **Real-time A* Search**

```
Open ← [Start]  // states to be considered
Closed ← []  // states that have been considered

// initial values
g(Start) = 0; f(Start) = h(Start)
while open != []
    Next ← first(Open)
    Open ← rest(Open)
    if isgoal(Next), return SUCCESS
    let Kids = children(Next) - (Open ⊖ Closed ⊖ [Next])
    Closed ← Closed ⊖ [Next]
    for each n ∈ children(Next) - Closed
        g'(n) = g(Next) + dist(Next, n)
        if g'(n) < g(n) or n Φ Open
            g(n) = g'(n); f(n) = g(n) + h(n); parent(n) = Next
    Open ← sort(append Kids, Open)
end-while
return FAIL
```
Real-Time A* Search Example

- $f(n) = g(n) + h(n)$: S→A→D→C→G, path cost is 16!

<table>
<thead>
<tr>
<th>Open</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(S,-,0,16)$</td>
<td>$(S,-,0,16)$</td>
</tr>
<tr>
<td>$(A,S,7,8)$ $(B,S,10,6)$</td>
<td>$(S,-,0,16)$ $(A,S,7,8)$</td>
</tr>
<tr>
<td>$(D,A,13,1)$ $(B,S,10,6)$ $(C,A,17,2)$</td>
<td>$(S,-,0,16)$ $(A,S,7,8)$ $(D,A,13,1)$</td>
</tr>
<tr>
<td>$(C,D,14,2)$ $(B,S,10,6)$ $(G,D,21,0)$</td>
<td>$(S,-,0,16)$ $(A,S,7,8)$ $(D,A,13,1)$ $(C,D,14,2)$</td>
</tr>
<tr>
<td>$(G,C,16,0)$ $(B,S,10,6)$</td>
<td>$(S,-,0,16)$ $(A,S,7,8)$ $(D,A,13,1)$ $(C,D,14,2)$</td>
</tr>
<tr>
<td>$(B,S,10,6)$</td>
<td>$(S,-,0,16)$ $(A,S,7,8)$ $(D,A,13,1)$ $(C,D,14,2)$</td>
</tr>
<tr>
<td></td>
<td>Uniform cost</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Complexity</td>
<td>exponential</td>
</tr>
<tr>
<td>Completeness</td>
<td>yes</td>
</tr>
<tr>
<td>Optimality</td>
<td></td>
</tr>
<tr>
<td>Admissibility</td>
<td>yes</td>
</tr>
<tr>
<td>Irrevocability</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Summary

- Heuristic: $h(n)$
  - Admissible
  - Dominate
  - How to come up with heuristics

- Best first search in general: $f(n)$
  - Uniform cost search: $f(n) = g(n)$
  - Greedy best first search: $f(n) = h(n)$
  - A* search: $f(n) = g(n) + h(n)$ if $h(n)$ is admissible

- Hill climbing search
- Beam search