Heuristic Search

Alice Gao

Lecture 3

Readings: RN 3.5 (esp. 3.5.2), PM 3.6, 3.7.

Outline

Learning goals

By the end of the lecture, you should be able to

- Describe motivations for applying heuristic search algorithms.
- Trace the execution of and implement the Lowest-cost-first search, Greedy best-first search and A* search algorithm.
- Describe properties of the Lowest-cost-first, Greedy best-first and A* search algorithms.
- Design an admissible heuristic function for a search problem. Describe strategies for choosing among multiple heuristic functions.
- Describes strategies for pruning a search space.

Why Heuristic Search?

How would ____ choose which one of the two states to expand?

- an uninformed search algorithm
- humans

5	3	
8	7	6
2	4	1

1	2	3
4	5	
7	8	6

Why Heuristic Search

An uninformed search algorithm

- considers every state to be the same.
- does not know which state is closer to the goal.
- may not find the optimal solution.

An heuristic search algorithm

- uses heuristics to estimate how close the state is to a goal.
- try to find the optimal solution.

The Cost Function

Suppose that we are executing a search algorithm
and we have added a path ending at n to the frontier.

$cost(n)$ is the actual cost of the path ending at n.

The Heuristic Function

Definition (search heuristic)

A **search heuristic** $h(n)$ is an estimate of the cost of the cheapest path from node n to a goal node.

In general, $h(n)$ can be arbitrary.
However, a good heuristic function has the following properties.

- problem-specific.
- non-negative.
- $h(n) = 0$ if n is a goal node.
- $h(n)$ must be easy to compute (without search).

LCFS, GBFS, and A*

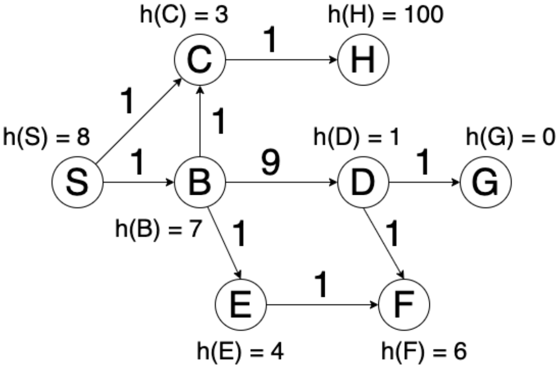
- LCFS: remove the path with the lowest cost $cost(n)$.
- GBFS: remove the path with the lowest heuristic value $h(n)$.
- A*: remove the path with the lowest cost + heuristic value $cost(n) + h(n)$.

Lowest-cost-first search

- Frontier is a priority queue ordered by $cost(n)$.
- Expand the path with the lowest $cost(n)$.

Trace LCFS on a search graph

If there is a tie, remove nodes from the frontier in alphabetical order.



Properties of LCFS

- Space and Time Complexities

Properties of LCFS

- Space and Time Complexities

  Both complexities are exponential.
  LCFS examines a lot of paths to ensure that
  it returns the optimal solution first.

Properties of LCFS

- Space and Time Complexities

  Both complexities are exponential.
  LCFS examines a lot of paths to ensure that
  it returns the optimal solution first.

- Completeness and Optimality

Properties of LCFS

- Space and Time Complexities

  Both complexities are exponential.
  LCFS examines a lot of paths to ensure that
  it returns the optimal solution first.

- Completeness and Optimality

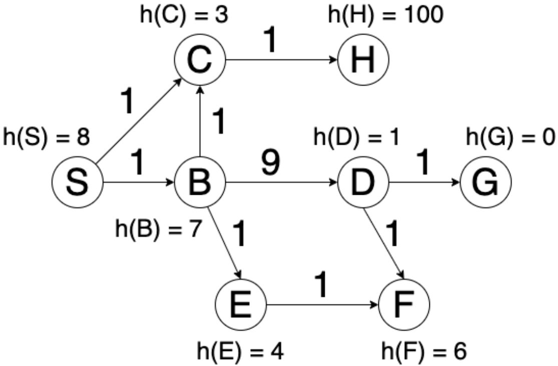
  Yes and yes under mild conditions:
  (1) The branching factor is finite.
  (2) The cost of every edge is bounded below by a positive constant.

Greedy Best-First Search

- Frontier is a priority queue ordered by $h(n)$.
- Expand the node with the lowest $h(n)$.

Trace GBFS on a search graph

If there is a tie, remove nodes from the frontier in alphabetical order.



Properties of GBFS

- Space and Time Complexities

Properties of GBFS

- Space and Time Complexities

  Both complexities are exponential.

Properties of GBFS

- Space and Time Complexities

  Both complexities are exponential.

- Completeness and Optimality

Properties of GBFS

- Space and Time Complexities

  Both complexities are exponential.

- Completeness and Optimality

  No, GBFS is not complete. It could be stuck in a cycle.
  No, GBFS is not optimal. GBFS may return a sub-optimal path first.

Properties of GBFS

- Space and Time Complexities

  Both complexities are exponential.

- Completeness and Optimality

  No, GBFS is not complete. It could be stuck in a cycle.
  No, GBFS is not optimal. GBFS may return a sub-optimal path first.

Learning Goals

Why Heuristic Search

LCFS, GBFS, and A*

- Lowest-Cost-First Search
- Greedy Best-First Search
- A* Search

Designing an Admissible Heuristic

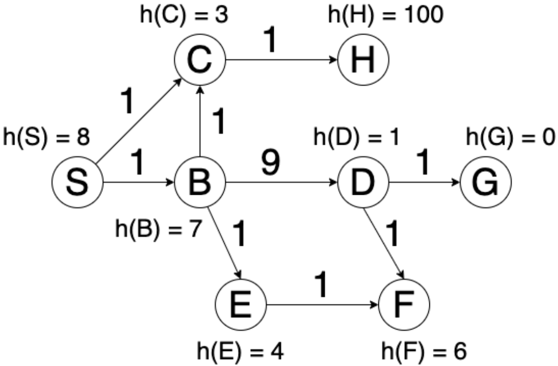
Pruning the Search Space

A* Search

- Frontier is a priority queue ordered by $f(n) = cost(n) + h(n)$.
- Expand the node with the lowest $f(n)$.

Trace A* search on a search graph

If there is a tie, remove nodes from the frontier in alphabetical order.



Properties of A* Search

- Space and Time Complexities

Properties of A* Search

- Space and Time Complexities

  Both complexities are exponential.

Properties of A* Search

- Space and Time Complexities

  Both complexities are exponential.

- Completeness and Optimality

Properties of A* Search

- Space and Time Complexities

  Both complexities are exponential.

- Completeness and Optimality

  Yes and Yes, given mild conditions on the heuristic function.

A* is Optimal

Definition (admissible heuristic)

A heuristic $h(n)$ is admissible if it never over-estimates the cost of the cheapest path from node n to a goal node.

Theorem (Optimality of A*)

If the heuristic $h(n)$ is admissible, the solution found by A is optimal.*

A* is Optimally Efficient

Among all optimal algorithms that start from the same start node and use the same heuristic, A* expands the fewest nodes.

Learning Goals

Why Heuristic Search

LCFS, GBFS, and A*

Designing an Admissible Heuristic

Pruning the Search Space

Some Heuristic Functions for 8-Puzzle

- Manhattan Distance Heuristic:
  The sum of the Manhattan distances of the tiles from their goal positions
- Misplaced Tile Heuristic:
  The number of tiles that are NOT in their goal positions

Both heuristic functions are admissible.

Initial State

5	3	
8	7	6
2	4	1

Goal State

1	2	3
4	5	6
7	8	

Constructing an Admissible Heuristic

1. Define a relaxed problem by simplifying or removing constraints on the original problem.
2. Solve the relaxed problem without search.
3. The cost of the optimal solution to the relaxed problem is an admissible heuristic for the original problem.

Constructing an Admissible Heuristic for 8-Puzzle

8-puzzle: A tile can move from square A to square B

- if A and B are adjacent, and
- B is empty.

Which heuristics can we derive from relaxed versions of this problem?

CQ: Constructing an Admissible Heuristic

CQ: Which heuristics can we derive from the following relaxed 8-puzzle problem?

A tile can move from square A to square B
if A and B are adjacent.

(A) The Manhattan distance heuristic
(B) The Misplaced tile heuristic
(C) Another heuristic not described above

CQ: Constructing an Admissible Heuristic

CQ: Which heuristics can we derive from the following relaxed 8-puzzle problem?

A tile can move from square A to square B.

(A) The Manhattan distance heuristic
(B) The Misplaced tile heuristic
(C) Another heuristic not described above

Which Heuristic is Better?

- We want a heuristic to be admissible.
- Want a heuristic to have higher values (close to h^*).
- Prefer a heuristic that is very different for different states.

Dominating Heuristic

Definition (dominating heuristic)

Given heuristics $h_1(n)$ and $h_2(n)$. $h_2(n)$ dominates $h_1(n)$ if

- $(\forall n\ (h_2(n) \geq h_1(n)))$.
- $(\exists n\ (h_2(n) > h_1(n)))$.

Theorem

If $h_2(n)$ dominates $h_1(n)$, A using h_2 will never expand more nodes than A* using h_1.*

CQ: Which Heuristic of 8-puzzle is Better?

CQ: Which of the two heuristics of the 8-puzzle is better?

(A) The Manhattan distance heuristic dominates the Misplaced tile heuristic.

(B) The Misplaced tile heuristic dominates the Manhattan distance heuristic.

(C) Neither dominates the other one.

Cycle Pruning

- What is cycle pruning?

Cycle Pruning

- What is cycle pruning?

  Whenever we realize that we are following a cycle, we should stop expanding the path.

Cycle Pruning

- What is cycle pruning?

  Whenever we realize that we are following a cycle, we should stop expanding the path.

- Why do we want to perform cycle pruning?

Cycle Pruning

- What is cycle pruning?

  Whenever we realize that we are following a cycle, we should stop expanding the path.

- Why do we want to perform cycle pruning?

  Cycles may cause an algorithm to not terminate, e.g. DFS. Exploring a cycle is a waste of time since it cannot be part of a solution.

Search w/ Cycle Pruning

- How do we perform cycle pruning?

Search w/ Cycle Pruning

- How do we perform cycle pruning?

Algorithm 2 Search w/ Cycle Pruning

...
for every neighbour n of n_k **do**
    if $n \notin \langle n_0, \ldots, n_k \rangle$ **then**
        add $\langle n_0, \ldots, n_k, n \rangle$ to frontier;
...

Search w/ Cycle Pruning

- How do we perform cycle pruning?

Algorithm 3 Search w/ Cycle Pruning

...
for every neighbour n of n_k **do**
    if $n \notin \langle n_0, \ldots, n_k \rangle$ **then**
        add $\langle n_0, \ldots, n_k, n \rangle$ to frontier;
...

- What is the complexity of cycle pruning for DFS and BFS?

Search w/ Cycle Pruning

- How do we perform cycle pruning?

Algorithm 4 Search w/ Cycle Pruning

```
...
for every neighbour n of n_k do
    if n ∉ ⟨n_0, ..., n_k⟩ then
        add ⟨n_0, ..., n_k, n⟩ to frontier;
...
```

- What is the complexity of cycle pruning for DFS and BFS?

  DFS: constant time. remember the nodes on the current path.
  BFS: linear in path length.

Multiple-Path Pruning

- Why do we want to perform multi-path pruning?

Multiple-Path Pruning

- Why do we want to perform multi-path pruning?

  If we have already found a path to a node, we can discard other paths to the same node.

Multiple-Path Pruning

- Why do we want to perform multi-path pruning?

  If we have already found a path to a node, we can discard other paths to the same node.

- What is the relationship between cycle pruning and multi-path pruning?

Multiple-Path Pruning

- Why do we want to perform multi-path pruning?

  If we have already found a path to a node, we can discard other paths to the same node.

- What is the relationship between cycle pruning and multi-path pruning?

  Cycle pruning is a special case of multi-path pruning.

Search w/ Multi-Path Pruning

How do we perform multi-path pruning?

Search w/ Multi-Path Pruning

How do we perform multi-path pruning?

Algorithm 6 Search w/ Multi-Path Pruning

```
procedure SEARCH(Graph, Start node s, Goal test goal(n))
    frontier := {⟨s⟩};
    explored := {};
    while frontier is not empty do
        select and remove path ⟨n_0, ..., n_k⟩ from frontier;
        if n_k ∉ explored then
            add n_k to explored
            if goal(n_k) then
                return ⟨n_0, ..., n_k⟩;
            for every neighbour n of n_k do
                add ⟨n_0, ..., n_k, n⟩ to frontier;
    return no solution
```

A problem with multi-path pruning

- Multi-path pruning says that we keep the first path to a node and discard the rest.
- What if the first path to a node is not the least-cost path?
- Can multi-path pruning cause a search algorithm to fail to find the optimal solution?

Lowest-cost-first search w/ multi-path pruning

Can Lowest-Cost-First Search with multi-path pruning discard the optimal solution?

(A) Yes, it is possible.

(B) No, it is not possible.

A* search w/ multi-path pruning

Can A* search with an admissible heuristic and multi-path pruning discard the optimal solution?

(A) Yes, it is possible.

(B) No, it is not possible.

Finding optimal solution w/ multi-path pruning

What if a subsequent path to n is shorter than the first path found?

- Remove all paths from the frontier that use the longer path.
- Change the initial segment of the paths on the frontier to use the shorter path.
- Make sure that we find the least-cost path to a node first.

Consistent Heuristic

- An admissible heuristic requires that:
  For any node m and any goal node g,

$$h(m) - h(g) \leq cost(m, g).$$

Consistent Heuristic

- An admissible heuristic requires that:
  For any node m and any goal node g,

$$h(m) - h(g) \leq cost(m, g).$$

- To ensure that A* with multi-path pruning is optimal,
  we need a consistent heuristic function: For any two nodes m and n,

$$h(m) - h(n) \leq cost(m, n).$$

Consistent Heuristic

- An admissible heuristic requires that:
  For any node m and any goal node g,

  $$h(m) - h(g) \leq cost(m, g).$$

- To ensure that A* with multi-path pruning is optimal,
  we need a consistent heuristic function: For any two nodes m and n,

  $$h(m) - h(n) \leq cost(m, n).$$

- A consistent heuristic satisfies the monotone restriction:
  For any edge from m to n,

  $$h(m) - h(n) \leq cost(m, n).$$

Constructing Consistent Heuristics

- Most admissible heuristic functions are consistent.
- It's challenging to come up with a heuristic function that is admissible but not consistent.

Revisiting the learning goals

By the end of the lecture, you should be able to

- Describe motivations for applying heuristic search algorithms.
- Trace the execution of and implement the Lowest-cost-first search, Greedy best-first search and A* search algorithm.
- Describe properties of the Lowest-cost-first, Greedy best-first and A* search algorithms.
- Design an admissible heuristic function for a search problem. Describe strategies for choosing among multiple heuristic functions.
- Describes strategies for pruning a search space.