Heuristic Search

Alice Gao

Lecture 3

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

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*

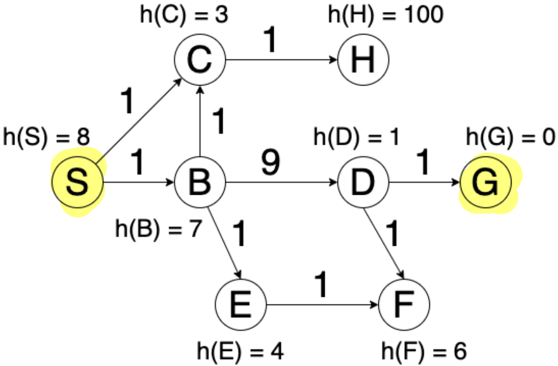
- LCFS: remove the path with the lowest cost $cost(n)$.
- GBFS: remove the path with the lowest heuristic value $h(n)$.
- A*: remove the path with the lowest cost + heuristic value $cost(n) + h(n)$.

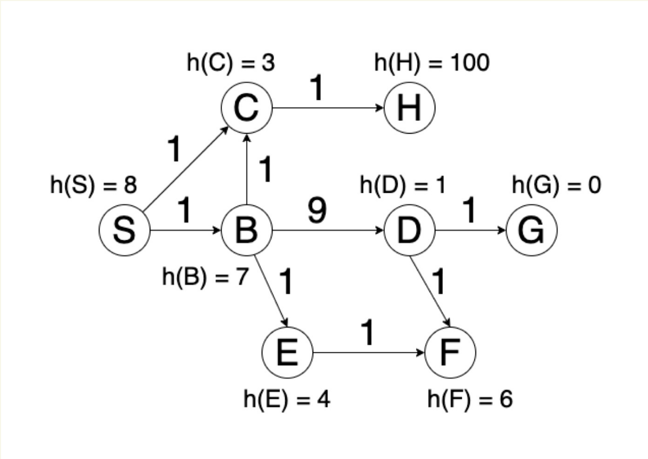
Lowest-cost-first search

- 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.

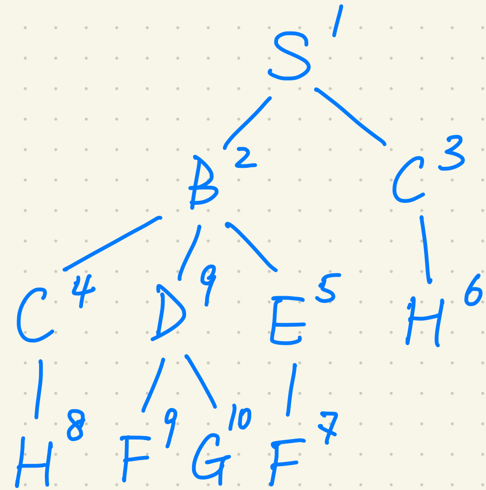




LCFS

frontier :

~~S~~	~~B~~	~~C~~	~~C~~	~~D~~	~~E~~	~~H~~	~~H~~	~~F~~	~~F~~	~~G~~
0	1	1	2	10	2	2	3	3	11	11



Properties of LCFS

- Complexity:

  time and space complexities are both exponential.

- Complete: Guaranteed to find a solution if one exists?

  Yes.

- Optimal: Guaranteed to find the optimal solution?

  Yes.

mild technical conditions required:

① branching factor is finite.

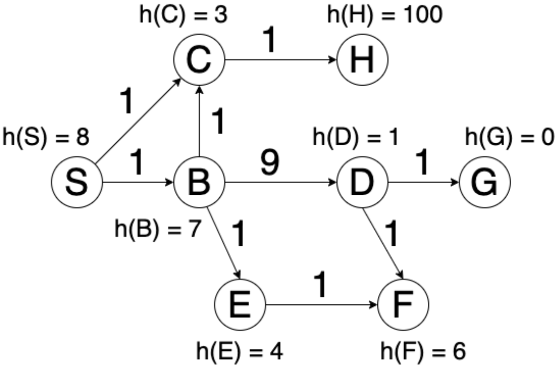
② the cost of every arc is bounded below by a positive constant.

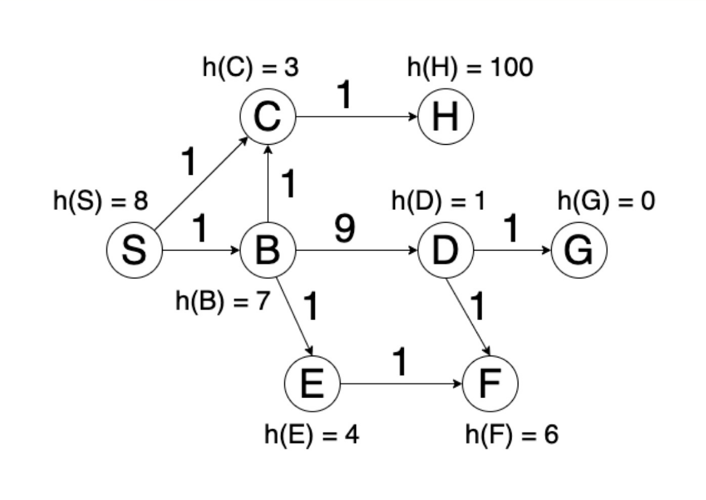
Greedy Best-First Search

- 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.

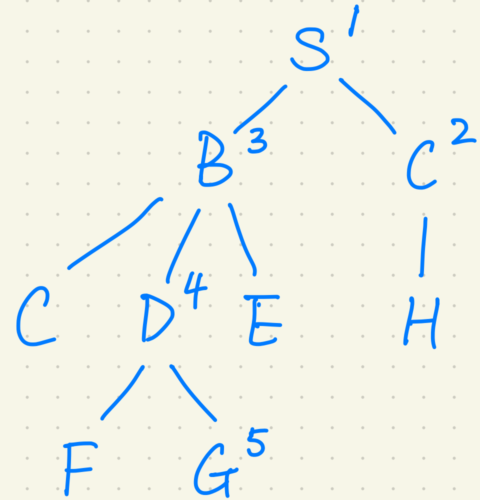




GBFS

frontier : ~~S~~ ~~B~~ ~~C~~ H C ~~D~~ E F ~~G~~

8 7 3 100 3 1 4 6 0



Properties of GBFS

- Complexity:

  space and time complexity are both exponential.

- Complete: Guaranteed to find a solution if one exists?

  No.

- Optimal: Guaranteed to find the optimal solution?

  No.

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

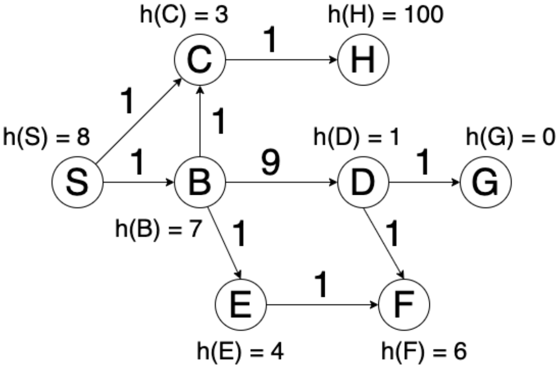
A* Search

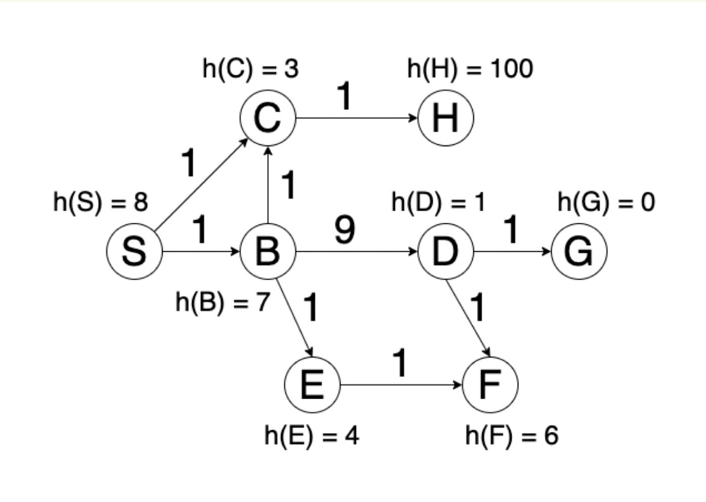
$$\underbrace{\underbrace{\text{start} \xrightarrow{\text{actual}} n}_{\text{cost}(n)} \underbrace{\xrightarrow{\text{estimate}} \text{goal}}_{h(n)}}_{f(n)}$$

- 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.

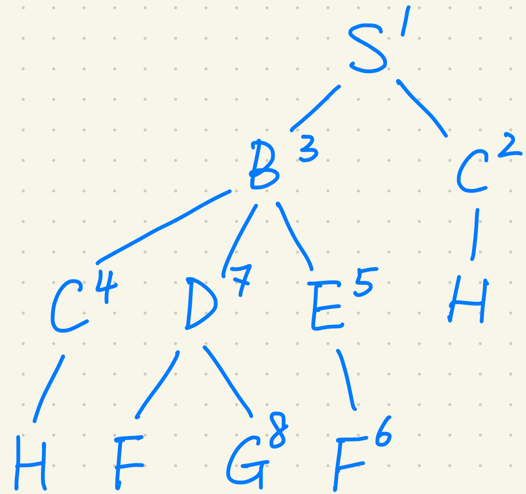




A*

frontier: ~~S~~ ~~B~~ ~~C~~ H ~~C~~ ~~D~~ ~~E~~ H ~~F~~ F ~~G~~

8 8 4 102 5 11 6 103 9 17 11



Properties of A*

- Complexity:

space and time complexities are both exponential.

- Complete: Guaranteed to find a solution if one exists?

Yes, if the heuristic satisfies a mild condition.

- Optimal: Guaranteed to find the optimal solution?

Yes, if the heuristic satisfies a mild condition.

A* is Optimal

$h^*(n)$: cost of cheapest path from n to a goal.

$$0 \leq h(n) \leq h^*(n).$$

Definition (admissible heuristic)

A heuristic $h(n)$ is admissible if it is never an overestimate of 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.*

$h(n)$ – a lower bound on the actual cost

– an optimistic estimate of the actual cost.

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.

A* could be unlucky w/ tie breaking.

Define optimal efficiency as

- expanding the minimum number of nodes n for which $f(n) \neq f^*$ where f^* is the cost of the cheapest path.

Some Heuristic Functions for 8-Puzzle

- Manhattan Distance Heuristic: h_1
  The sum of the Manhattan distances of the tiles from their goal positions
- Misplaced Tile Heuristic: h_2
  The number of tiles that are NOT in their goal positions

Both heuristic functions are admissible.

Initial State (S)

$h_1(S) = 4+3+1+2+2+0+2+2 = 16$

5 1	3 2	3
8 4	7 5	6 6
2 7	4 8	1

$h_2(S) = 7$

Goal State

1	2	3
4	5	6
7	8	

Constructing an Admissible Heuristic

- Define a relaxed problem by simplifying or removing constraints on the original problem.
- Solve the relaxed problem without search.
- 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?

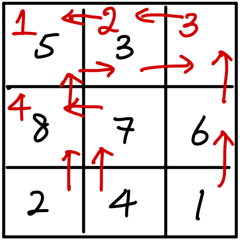
A tile can move from square A to square B if A and B are adjacent. (B may not be empty.)

(A) The Manhattan distance heuristic

(B) The Misplaced tile heuristic

(C) Another heuristic not described above

4+3+1+2
+ ....



CQ: Constructing an Admissible Heuristic

CQ: Which heuristics can we derive from the following relaxed 8-puzzle problem? (A and B may not be adjacent.)

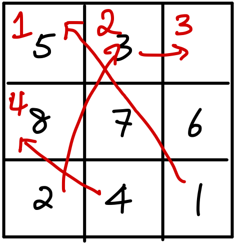
A tile can move from square A to square B. (B may not be empty.)

(A) The Manhattan distance heuristic

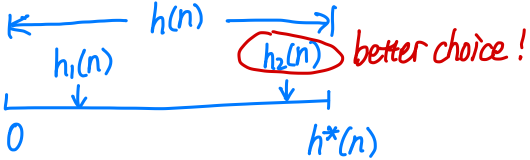
(B) The Misplaced tile heuristic

(C) Another heuristic not described above

1+1+1+1 + ... +



Which Heuristic is Better?



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

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.

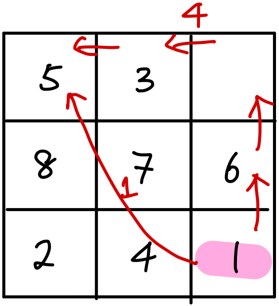
CQ: Which Heuristic of 8-puzzle is Better?

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

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

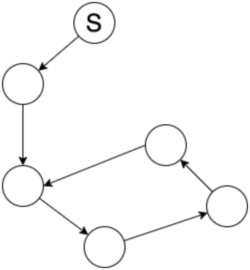
(B) The Misplaced tile heuristic dominates the Manhattan distance heuristic.

(C) I don't know....



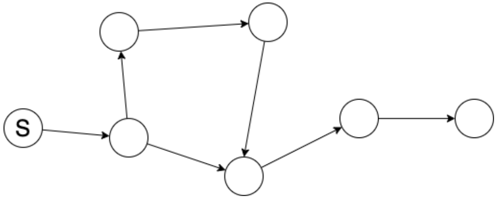
Cycle Pruning

- A cycle cannot be part of a least-cost path.
- Works well with depth-first search.
- The complexity of cycle pruning is ...



Multiple-Path Pruning

- If we have already found a path to a node, we can prune other paths to the same node.
- Subsumes a cycle check.
- Requires storing all nodes we have found paths to.
- What if a subsequent path to *n* is shorter than the first path found?



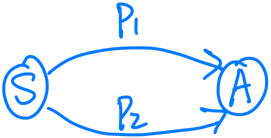
Lowest-cost-first search w/ multiple-path pruning

If we perform multiple-path pruning with lowest-cost-first search, is it possible for us to prune the optimal solution (least-cost path)?

(A) Yes, it is possible.

(B) No, it is not possible.

$cost(P_1) > cost(P_2)$



A* search w/ multiple-path pruning

Q: come up w/ a search graph s.t.
A* w/ multi-path pruning discards the optimal solution.

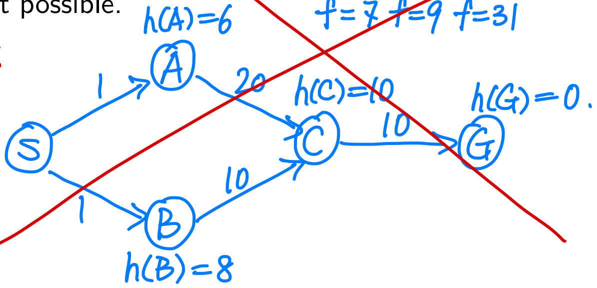
If we perform multiple-path pruning with A* search,
is it possible for us to prune the optimal solution (least-cost path)?

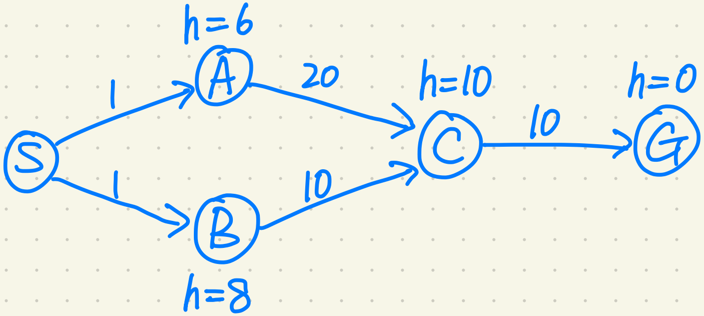
(A) Yes, it is possible.

(B) No, it is not possible.

frontier : ~~S~~, ~~SA~~, ~~SB~~, SAC, ~~SBC~~
f=7 f=9 f=31

This example is wrong. Check the pages below for correct examples





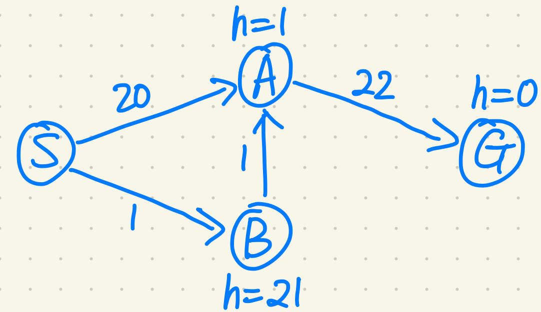
explored: S, A, B, C, G

frontier: ① ~~S~~, ② ~~SA~~, ③ ~~SB~~, SAC, ④ ~~SBC~~, ⑤ ~~SBCG~~

7 9 31 21 21

path found: SBCG (21) optimal solution

SACG (31)



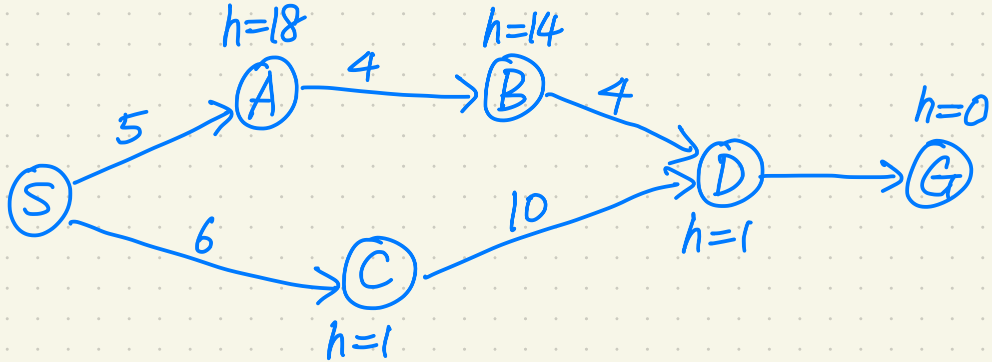
explored: S, A, B, G

frontier: ① ~~S~~, ② ~~SA~~, ③ ~~SB~~, ⑤ ~~SAG~~, ④ ~~SBA~~

21 22 42 3

path found: SAG (42)

optimal solution: SBAG (24)



explored: S, C, D, A, B, G

	1	4	2	3	7	5	6
frontier:	~~S~~,	~~SA~~,	~~SC~~,	~~SCD~~,	~~SCDG~~,	~~SAB~~,	~~SABD~~
		23	7	17	26	23	14

path found: SCDG (26)

optimal solution: SABDG (23)

Find Optimal Solution w/ Multiple-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 the least-cost path to a node is found first.

A* search w/ multiple-path pruning

How can we ensure that A* with multiple-path pruning is optimal?

- Ensure that we find the least-cost path to every node first.
- Admissible heuristic guarantees the above for a goal node, but not for other nodes.
- We need the heuristic to satisfy the monotone restriction: → consistent.

$$\textit{for any arc } \langle m, n\rangle,\ h(m) - h(n) \leq \textit{cost}(m, n).$$

if n is a goal node, h(m) ≤ cost(m, n) → admissible heuristic.

If the heuristic satisfies the monotone restriction, A* search with multiple-path pruning is optimal.

h(m) − h(n) = the heuristic estimate of the path cost from m to n.

Summary of Search Strategies

Strategy	Frontier Selection	Halts?	Space	Time
Depth-first	Last node added	No	Linear	Exp
Breadth-first	First node added	Yes	Exp	Exp
Lowest-cost-first	min $cost(n)$	Yes	Exp	Exp
Greedy Best-first	min $h(n)$	No	Exp	Exp
A*	min $cost(n) + h(n)$	Yes	Exp	Exp

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.