COMP 424 - Artificial Intelligence
Lecture 3: Informed Search

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

Class website:

http://cs.mcgill.ca/~jcheung/teaching/winter-2017/comp424/index.html

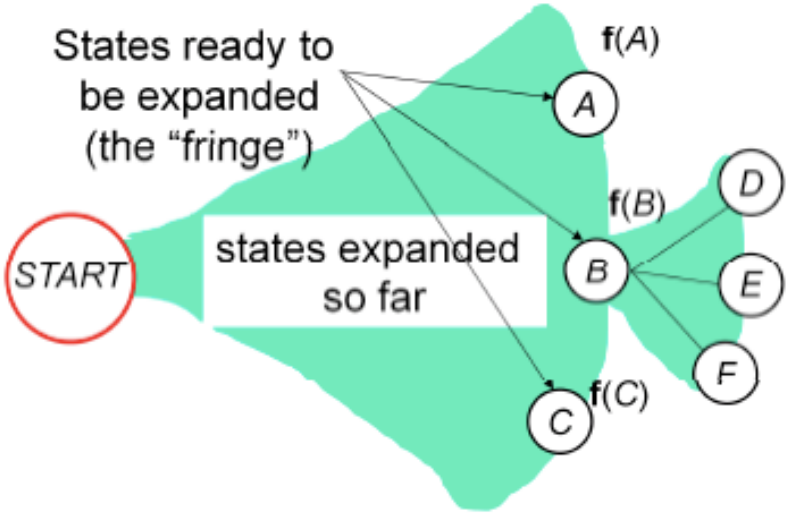
Readings: R&N Ch 3.5-3.7

Based on slides by Joelle Pineau

Quick recap

- Problems are described by states, operators, costs.
- For each state and operator, there is a set of successor states.
- In search, we maintain a set of nodes, forming a search tree.
- The fringe of the tree contains candidate nodes, and is typically maintained using a priority queue.

- **Different search algorithms use different priority functions.**

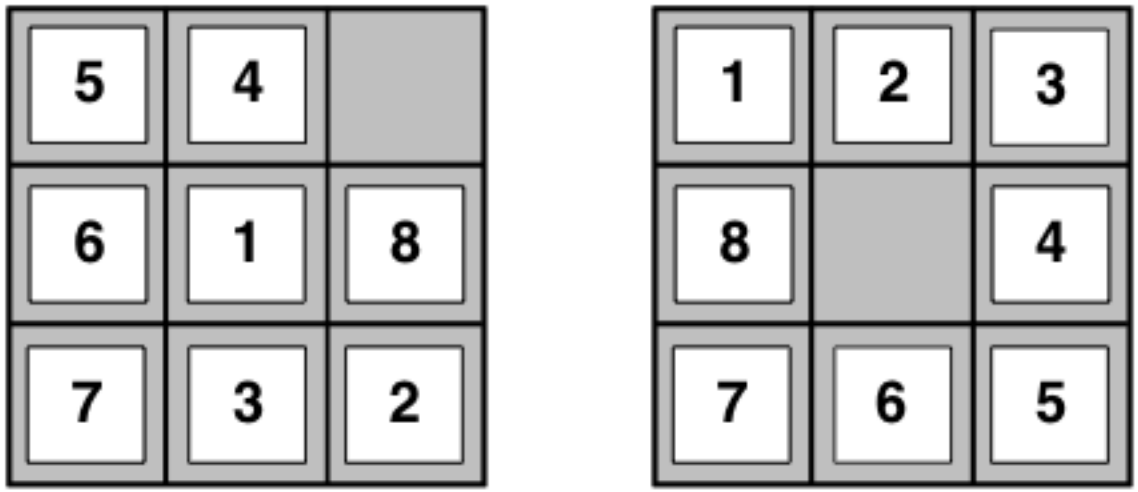


Overview of today's class

- Using heuristics to inform (guide) the search

- Informed search algorithms
  - Best-First (Greedy) Search
  - Heuristic Search
  - A* Search

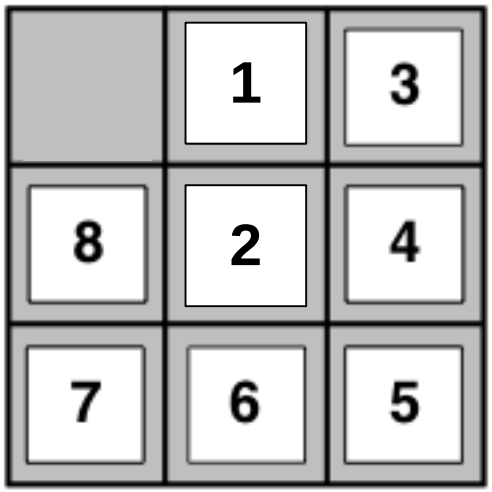
Eight-Puzzle



How would **you** solve an eight-puzzle? Do you really perform an uninformed search as we discussed last class, or do some configurations seem more promising than others?

Example

- We are very close to the goal:



Goal State

Informed Search

- Uninformed search methods expand nodes based on **distance from the start node**, $d(n_{start}, n)$

  Easy: we always know that!

- What about expanding based on **distance to the goal**, $d(n, n_{goal})$?

Informed Search

- Uninformed search methods expand nodes based on **distance from the start node**, $d(n_{start}, n)$

  Easy: we always know that!

- What about expanding based on **distance to the goal**, $d(n, n_{goal})$?
- If we know $d(n, n_{goal})$ exactly?

  Easy: just expand the nodes needed to find a solution!

Informed Search

- Uninformed search methods expand nodes based on **distance from the start node**, $d(n_{start}, n)$

  Easy: we always know that!

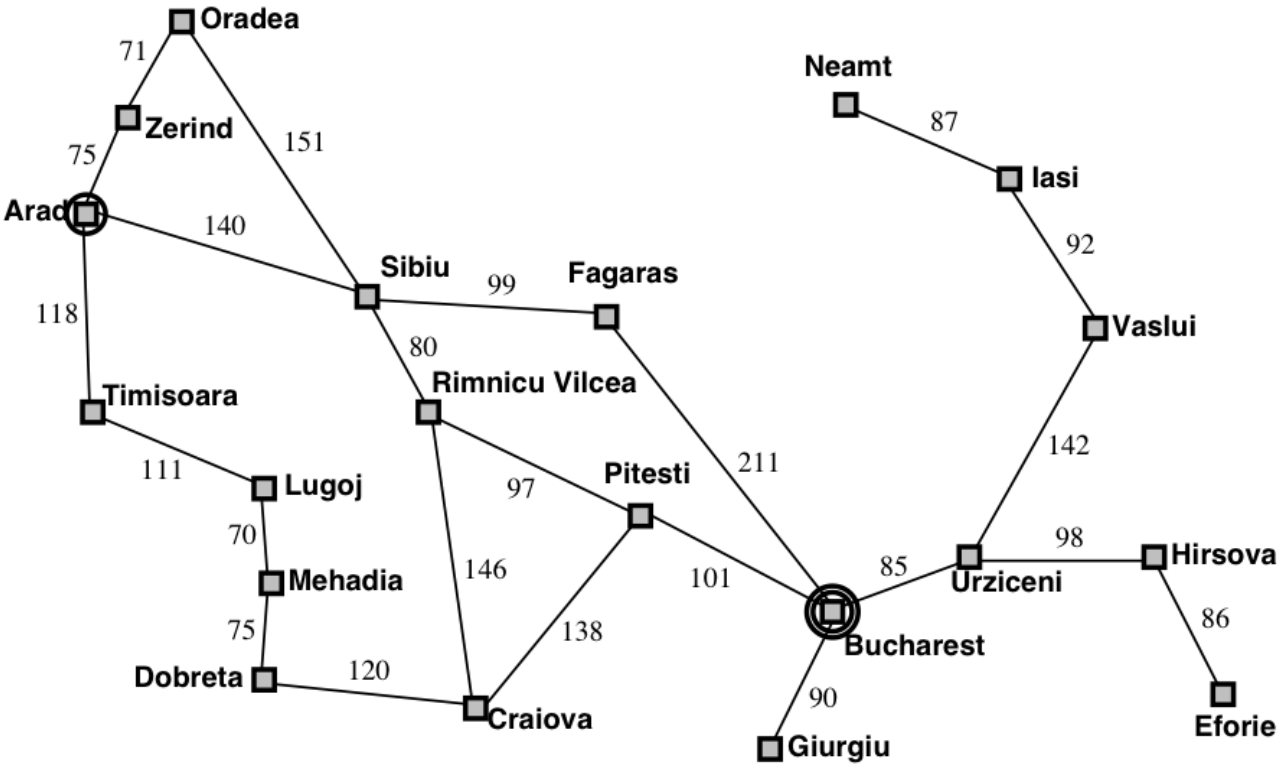
- What about expanding based on **distance to the goal**, $d(n, n_{goal})$?
- If we know $d(n, n_{goal})$ exactly?

  Easy: just expand the nodes needed to find a solution!

- If we do not know $d(n, n_{goal})$ exactly, can we use **intuition about this distance**?
  - We will call this intuition a **heuristic** ***h(n)***.

Example: Travelling through Romania

What is good heuristic for path planning problems?



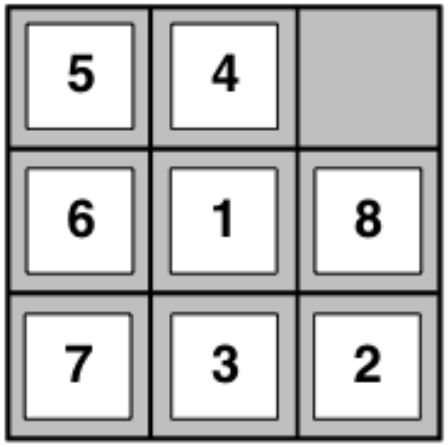
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example heuristic: path planning

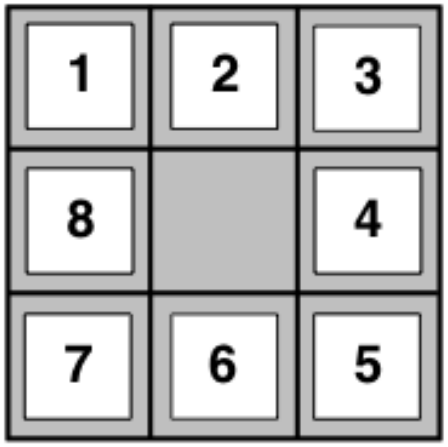
- What is a reasonable heuristic?
  - The straight-line distance between two places.
- Is it always right?
  - Clearly not - roads are rarely straight!
- But it's roughly correct.

Example heuristic: eight-puzzle

- What would be a good heuristic for this problem?



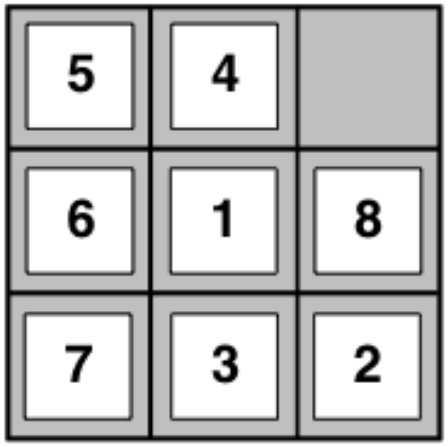
Start State



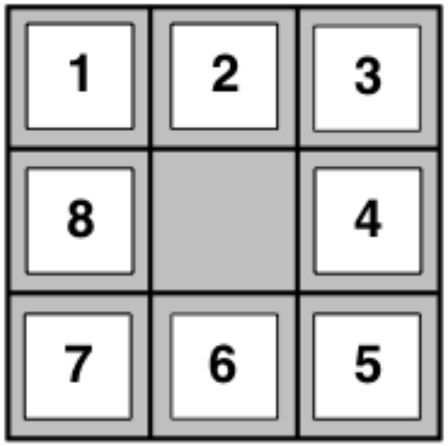
Goal State

Example heuristic: eight-puzzle

- Two options: $h_1(n)$ = number of misplaced tiles = 7
  $h_2(n)$ = total Manhattan distance
  = (2+3+3+2+4+2+0+2) = 18
- Which is better?



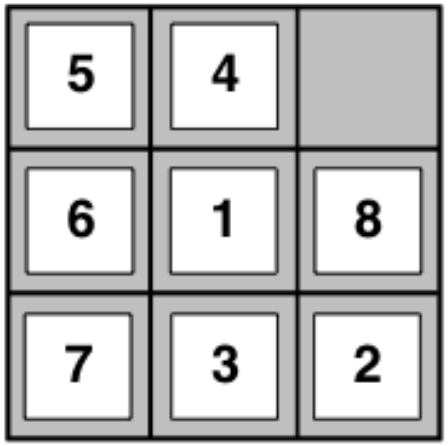
Start State



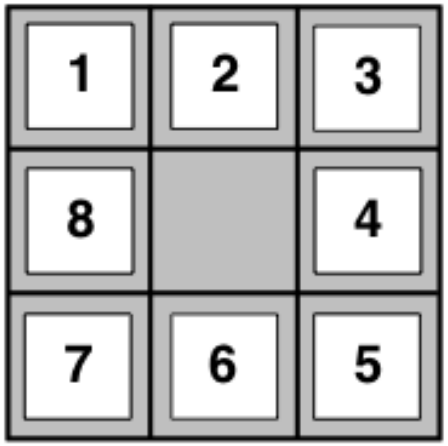
Goal State

Example heuristic: eight-puzzle

- Two options: $h_1(n)$ = number of misplaced tiles = 7
  $h_2(n)$ = total Manhattan distance
  = (2+3+3+2+4+2+0+2) = 18
- Which is better? Intuitively, $h_2(n)$ seems better : varies more across state space and its estimate is closer to the true cost.



Start State



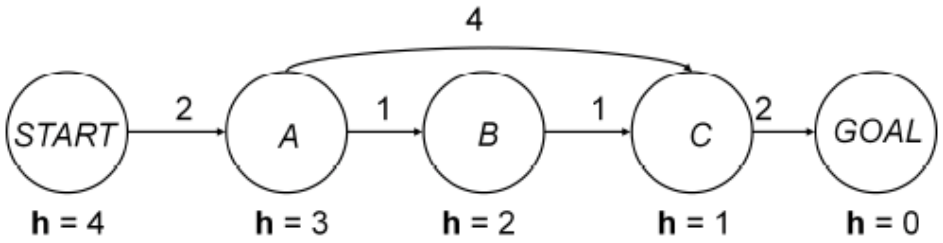
Goal State

Where do heuristics come from?

- Prior knowledge about the problem.
- Exact solution to a relaxed version of the problem.
  - E.g. If rules of eight-puzzle are changed to allow a tile to move anywhere. Then $h_1(n)$ gives the exact distance to goal.
  - E.g. If rules of eight-puzzle are changed to allow a tile to move to any adjacent square. Then $h_2(n)$ gives the exact distance to goal.
- Learning from prior experience.

Best-First Search

- **Algorithm**: expand the **most promising node** according to the **heuristic**.
- Example:

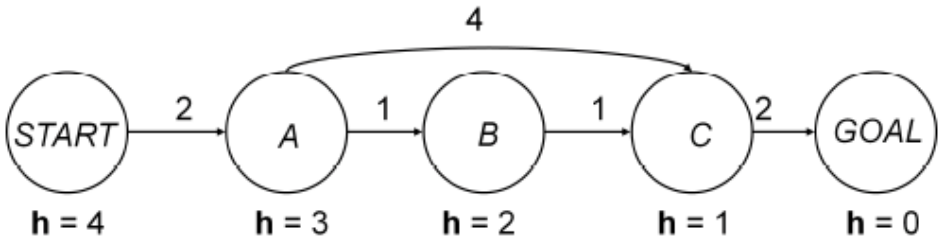


  - What is the sequence of nodes visited in Best-First Search?
  - What is the sequence of nodes visited in Uniform-Cost Search?

Best-First Search

- **Algorithm**: expand the **most promising node** according to the **heuristic**.

- Example:



  - What is the sequence of nodes visited in Best-First Search?

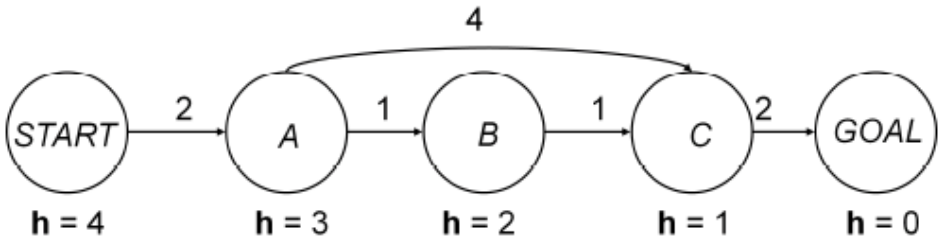
    START -> A -> C -> B -> GOAL

  - What is the sequence of nodes visited in Uniform-Cost Search?

    START -> A -> B -> C -> GOAL

Best-First Search

- **Algorithm**: expand the **most promising node** according to the **heuristic**.
- Example:



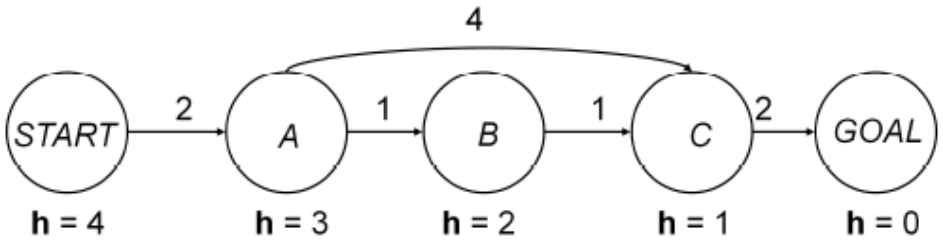
- Best-first search is similar to Breadth-first search (BFS)

  How similar depends on the goodness of the heuristic.

  If the heuristic is always 0, best-first search is the same as BFS.

Best-First Search

- **Algorithm**: expand the **most promising node** according to the **heuristic**.
- Example:



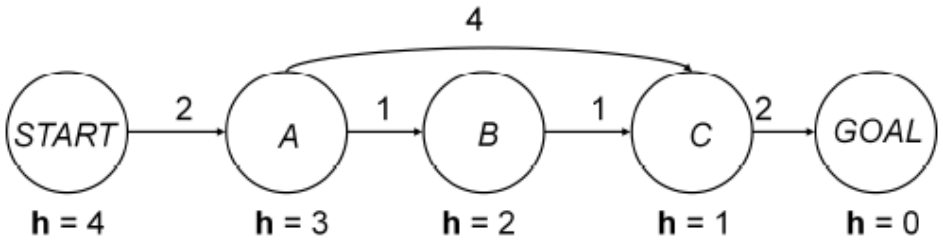
- Best-first search contrasts with uniform-cost search:

  Uniform cost search considers the **cost-so-far**.

  Best-first search considers the **cost-to-go**.

Best-First Search

- **Algorithm**: expand the **most promising node** according to the **heuristic**.
- Example:



- Best-first search is a **greedy method**.

  ↳**maximize short-term advantage** without worrying about long-term consequences.

Properties of Greedy Search

- In worst case: exponential time/space complexity (same as BFS).

  $O(b^d)$ (b=branching factor, d=solution depth).

- A good heuristic can help a lot!

  $O(bd)$ (same as DFS).

Properties of Greedy Search

- In worst case: exponential time/space complexity (same as BFS).

  $O(b^d)$ (b=branching factor, d=solution depth).

- A good heuristic can help a lot!

  $O(bd)$ (same as DFS).

- Completeness:
  - Not always complete. Can get stuck in loops.
  - Complete in finite spaces, if we check to avoid repeated states.

- Not optimal! (as seen in the example.) **=> Can we fix this?**

Fixing greedy search

- What's the problem with best-first search?
  - It is **too greedy**: does not take into account the cost so far!

- Can you suggest a solution?
  - Let *g* be the cost of the path so far
  - Let *h* be a heuristic function (estimated cost to go)

Fixing greedy search

- What's the problem with best-first search?
  - It is **too greedy**: does not take into account the cost so far!

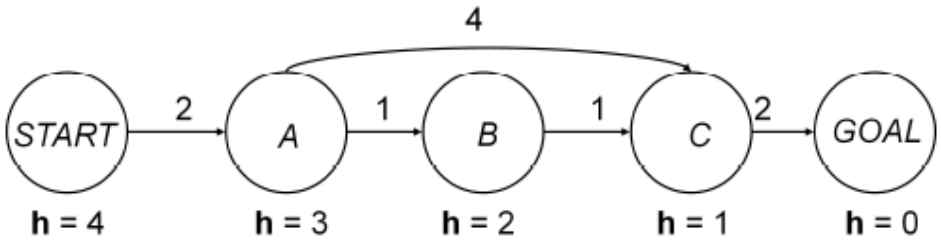
- Can you suggest a solution?
  - Let *g* be the cost of the path so far
  - Let *h* be a heuristic function (estimated cost to go)

- **Heuristic search**: best-first search, greedy with respect to *f=g+h*
  - Think of *f = g+h* as the estimate of the cost of the current path.

Heuristic search

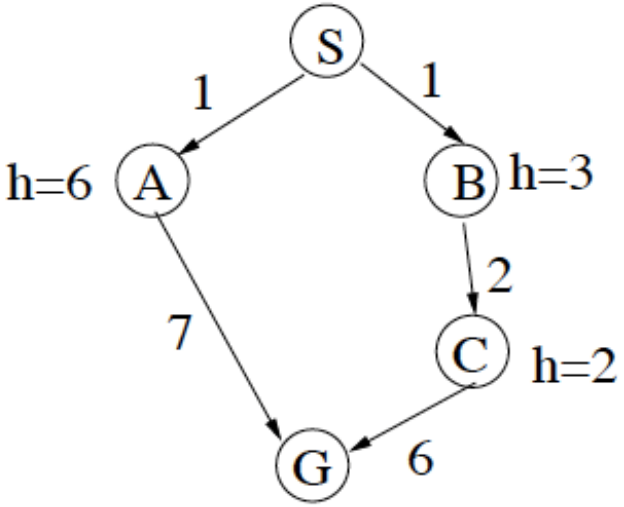
- **Algorithm**:
  - At every step, take node *n* from the front of the queue.
  - Add to queue successors *n'* with **priorities**: $f(n') = g(n')+h(n')$
  - Terminate when a goal state is popped from the queue.

- Previous example:

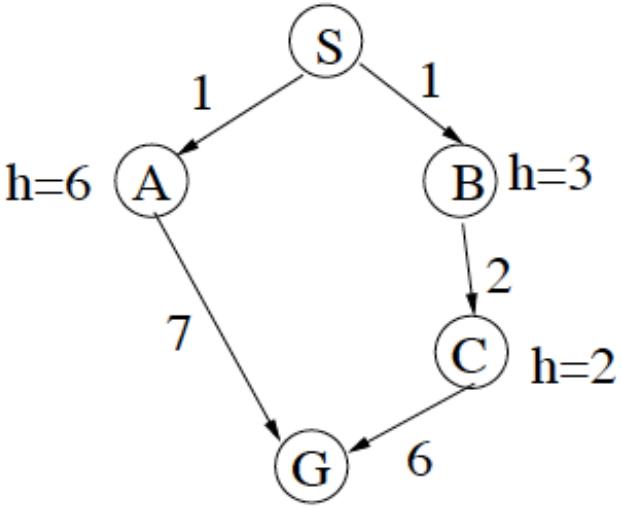


Example

Priority queue: {(S,h(S))}



Example



Priority queue: {(S,h(S))}

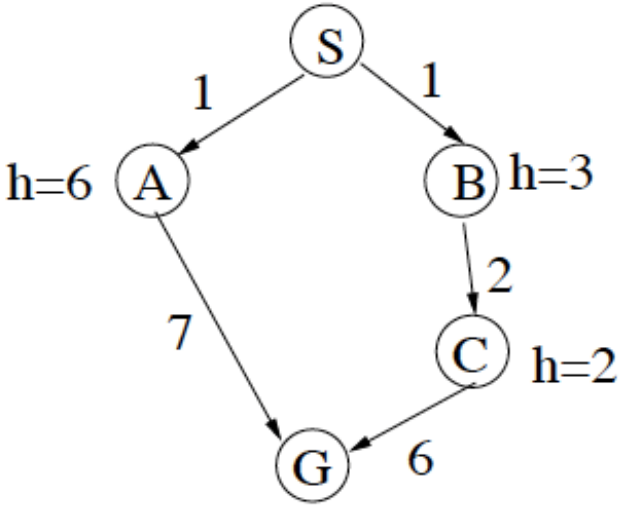
{~~(S,h(S))~~, (B,4), (A,7)}

{~~(S,h(S)), (B,4)~~, (C,5), (A,7)}

{~~(S,h(S)), (B,4), (C,5)~~, (A,7), (G,9)}

{~~(S,h(S)), (B,4), (C,5), (A,7)~~, (G,8), (G,9)} **Found!**

Example



Priority queue: {(S,h(S))}

{~~(S,h(S))~~, (B,4), (A,7)

{~~(S,h(S)), (B,4)~~, (C,5)

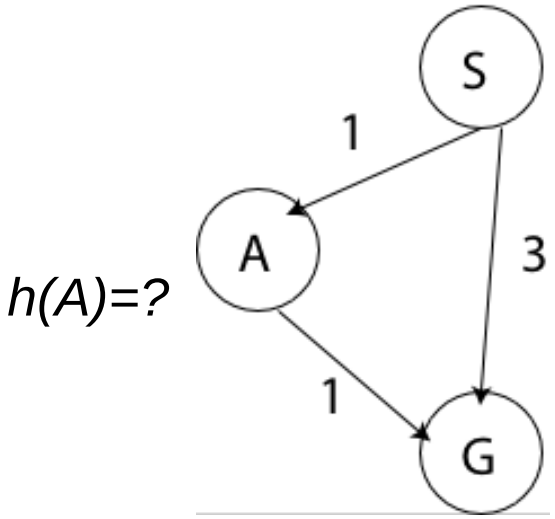
{~~(S,h(S)), (B,4), (C,5)~~, (A,7), (G,9)}

{~~(S,h(S)), (B,4), (C,5), (A,7)~~, (G,8), (G,9)} **Found!**

Remark: Continue expanding nodes after goal is found if there are unexpanded nodes that have lower cost than current path to goal.

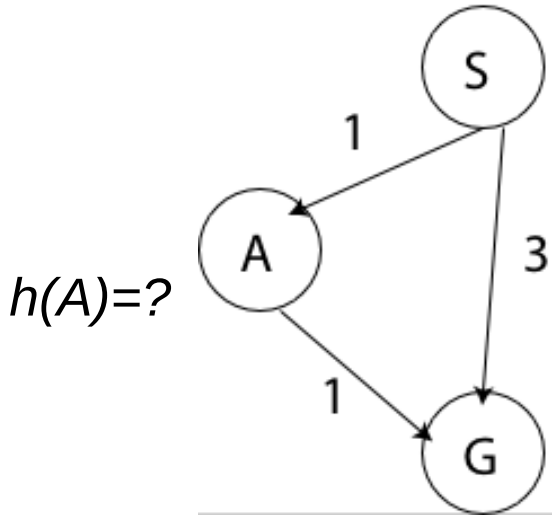
Is heuristic search optimal?

- Example:



Is heuristic search optimal?

- Example:



- Short answer: NO!
- In this example, any heuristic *h(A)>2* leads to suboptimal path (and if h(A) = 2, it depends on tie breaking).

We must put **conditions** on the choice of heuristic to guarantee optimality.

Admissible heuristics

- Let $h^*(n)$ be the shortest path from n to any goal state.
- A heuristic h is called an **admissible heuristic** if:

$$h(n) \leq h^*(n), \ \forall n.$$

- Admissible heuristics are optimistic.

Admissible heuristics

- Let $h^*(n)$ be the shortest path from n to any goal state.
- A heuristic h is called an **admissible heuristic** if:

$$h(n) \leq h^*(n), \ \forall n.$$

- Admissible heuristics are optimistic.

- Trivial case of an admissible heuristic: $h(n)=0, \ \forall n$.
  - In this case, we get uniform-cost search.

- If $h()$ is optimistic, we must have $h(g)=0, \ \forall g \in G$, the goal set.

Examples of admissible heuristics

- Robot navigation?
- 8-puzzle?

Examples of admissible heuristics

- Robot navigation? straight-line distance to goal.

- 8-puzzle? h_1: number of misplaced tiles.
  h_2: sum of Manhattan distances for each tile to its goal position.

- In general, if we get a heuristic by solving a relaxed version of the problem, we will obtain an admissible heuristic.

A* search

- **Algorithm:** Heuristic search with an admissible heuristic**.**
- Let *g* be the cost of the path so far.
- Let *h* be an admissible heuristic function.
- Perform greedy search with respect to:

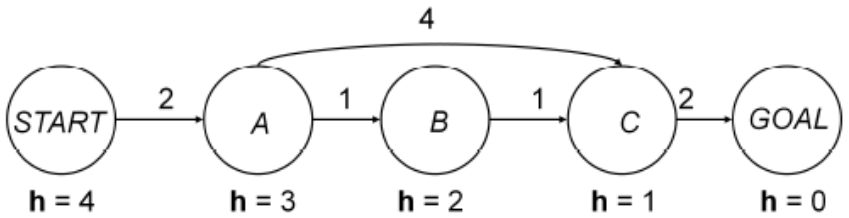
$$f = g + h$$

Consistency

- An admissible heuristic *h* is called **consistent** (or monotone*) if for every state *s* and successor *s'*, we have $h(s) \leq c(s,s') + h(s')$.
  - Think of *h* as getting "more precise" as it gets closer to the goal.

Consistency

- An admissible heuristic *h* is called **consistent** (or monotone*) if for every state *s* and successor *s'*, we have *h(s) ≤ c(s,s') + h(s').*
  - Think of *h* as getting "more precise" as it gets closer to the goal.



- This is a version of triangle inequality*, so heuristics that respect this inequality are metrics*.
- Consistency is a slightly stronger property than admissibility.

**Note: Look-up these terms if they are not familiar.*

Is A* complete?

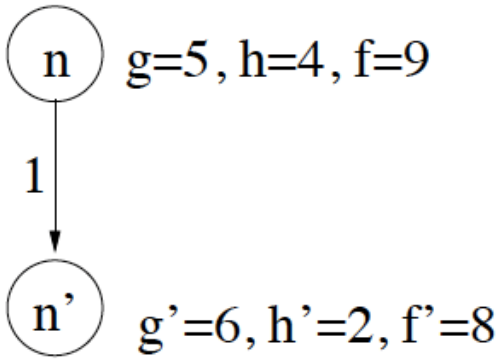
- Suppose that *h* is consistent, and all costs *c(s,s')>0*.
- This implies that *f* cannot decrease along any path:

  $f(s) = g(s) + h(s) \leq g(s) + c(s,s') + h(s') = f(s')$

- In this case, a node cannot be re-expanded.
- If a solution exists, it must have bounded cost.

- Hence A* will have to find it! So it is complete.

Fixing inconsistent heuristics



- Make a small change to A*:

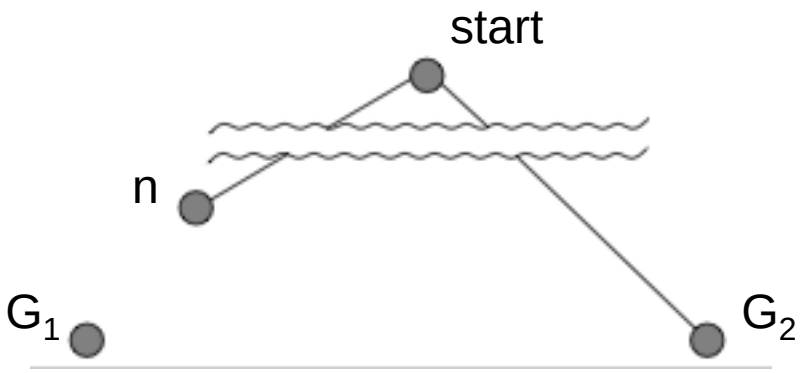
  Instead of *f(s') = g(s') + h(s')*, use

  f(s') = max { g(s')+h(s'), f(s) }

- With this change, *f* is non-decreasing along any path, and the previous argument applies.

Is A* optimal?

- Proof by contradiction. Suppose a suboptimal goal G_2 is in queue.
- Let *n* be an unexpanded node on a shortest path to optimal goal G_1.



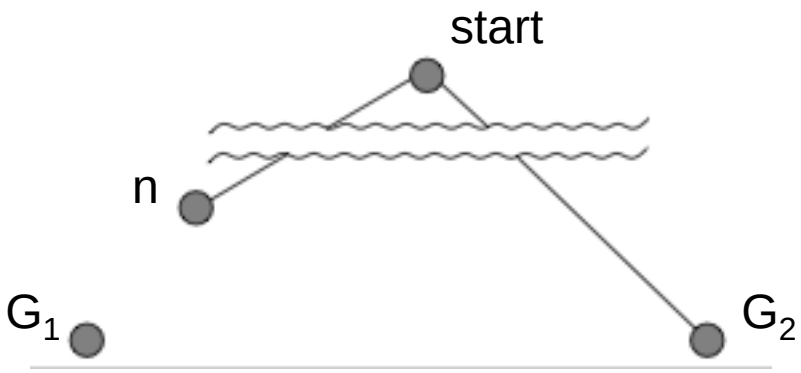
Is A* optimal?

- Proof by contradiction. Suppose a suboptimal goal G_2 is in queue.
- Let n be an unexpanded node on a shortest path to optimal goal G_1.
- We have:

  $f(G_2) = g(G_2)$ since $h(G_2)=0$

  $> g(G_1)$ since G_2 is suboptimal

  $\geq f(n)$ since h is admissible



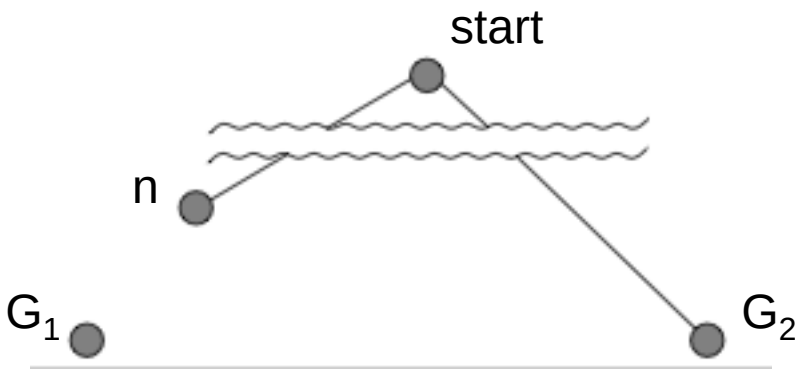
Is A* optimal?

- Proof by contradiction. Suppose a suboptimal goal G_2 is in queue.
- Let n be an unexpanded node on a shortest path to optimal goal G_1.
- We have:

  $f(G_2) = g(G_2)$ since $h(G_2)=0$

  $> g(G_1)$ since G_2 is suboptimal

  $\geq f(n)$ since h is admissible



- Since $f(G_2) > f(n)$, A* will select n for expansion before G_2.
- Similarly, all nodes on the optimal path will be chosen before G_2, so G_1 will be reached before G_2.

Dominance

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

  Intuition: $h_2(n)$ *is more informative than* $h_1(n)$

Dominance

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

  Intuition: $h_2(n)$ *is more informative than* $h_1(n)$

- Eight-puzzle example:
  - d=14 Iterative depth search = 3,473,941 nodes
    A*(h_1) = 539 nodes
    A*(h_2) = 113 nodes
  - d=24 Iterative depth search = too many nodes
    A*(h_1) = 39,135 nodes
    A*(h_2) = 1,641 nodes

Properties of A*

- Complete!
- Optimal!

- Exponential worst-case time/space complexity.
  - But with a perfect heuristic, complexity is *O(bd)*, because we only expand nodes on optimal path.
  - With a good heuristic, complexity is often sub-exponential.

- Optimally efficient: with a given *h*, no other search algorithm will be able to expand fewer nodes.

Iterative Deepening A* (IDA*)

- Same trick as we used before to avoid memory problems.
- Algorithm is basically depth-first search, but using the *f*-value to decide in which order to consider the descendants of a node.
- Use an *f*-value limit, rather than a depth limit.

Iterative Deepening A* (IDA*)

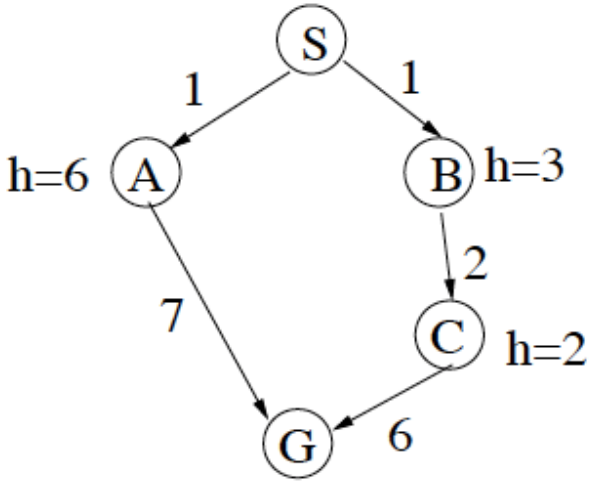
- Same trick as we used before to avoid memory problems.
- Algorithm is basically depth-first search, but using the *f*-value to decide in which order to consider the descendants of a node.
- Use an *f*-value limit, rather than a depth limit.

- IDA* has the same properties as A* but uses less memory.
- In order to avoid always expanding new nodes, old ones can be remembered if memory permits (this version is known as SMA*)

Iterative Deepening A* example

- Set $f_1 = 4$ $\Rightarrow$ only S, B are searched (no other nodes are put in the queue, because they exceed the cutoff threshold.)

- Set $f_2 = 8$ $\Rightarrow$ now S, A, B, C, G, are all searched.



Real-time search

- In dynamic environments, agents have to act before they finish thinking!
- Instead of searching for a complete path to the goal, we would like the agent to do a bit of search, then move in the direction of the currently "best" path.
- Main issue: how do we avoid cycles, if we do not have enough memory to mark states?

Real-Time A* (Korf, 1990s)

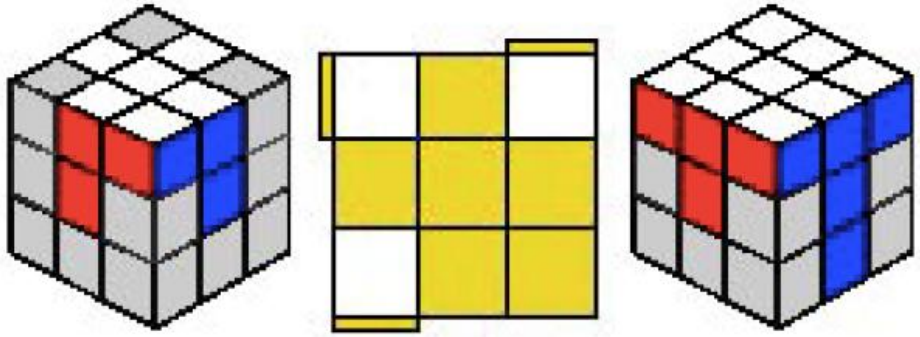
- When should the algorithm backtrack to a previously visited state *s*?
- Intuition: if the cost of backtracking to *s* and solving the problem from there is better than the cost of solving from the current state
- Korf's solution: do A* but with the *g* function equal to the cost from the current state, rather than from the start.
  - This simulates physically going back to the previous state.
- This is an execution-time algorithm!

Search improvements

- Consider Rubik's cube: 43,252,003,274,489,856,000 states!
- How do people solve this puzzle?

Changing the search problem

- People do not think at the level of individual moves!
- Instead, there are sequences of moves, designed to achieve a certain pattern (e.g. L-shape, T-shape, etc.)



- Instead of choosing individual operators, choose what subgoal to achieve next. Then solve this subgoal, pick the next one, etc.
- The solution to a subgoal is often standard, and can be reused.

Abstraction and decomposition

- **Decomposition**: The key to solving complex problems is to break them into smaller parts. Each part may be easy to solve; then put the solutions together.
- A macro-action is a sequence of actions from the original problem.
  - E.g. Making a T in Rubik's cube.

Abstraction and decomposition

- **Decomposition**: The key to solving complex problems is to break them into smaller parts. Each part may be easy to solve; then put the solutions together.
- A macro-action is a sequence of actions from the original problem.
  - E.g. Making a T in Rubik's cube.

- **Abstraction** refers to methods that ignore information, in order to speed-up computations.
  - In Rubik's cube, focus only on certain aspects and ignore rest of tiles.
- In abstraction we construct a compact representation, with many "real" states mapped into a single "abstract" state.
- Decomposition and abstraction are usually applied together.

Examples

- Landmark navigation:
  - Find a path from the current location to a well-known landmark (e.g. McGill metro).
  - Find a path between landmarks (this can be pre-computed).
  - Find a path from last landmark to destination.

Trade-offs

- By decomposing a problem and putting the solutions together, we may be giving up optimality.
- Apply this to problems that we can't solve otherwise!
- Solutions to subgoals are often cached in a database, and can be re-used (e.g. with different state/goal states).
- When we choose subgoals, we need to be careful that the overall problem still has a solution.
  - Under some conditions, sub-solutions can be pieced together to preserve completeness (Knoblock, 1990's).

Summary of informed search

- Insight: use knowledge about the problem, in the form of a heuristic.
  - The heuristic is a guess for the remaining cost to the goal.
  - A good heuristic can reduce search time from exponential to almost linear.
- Best-first search is greedy with respect to the heuristic, not complete and not optimal.
- Heuristic search is greedy with respect to $f=g+h$, where g is the cost so far and h is the estimated cost to go.
- A* is heuristic search where h is an admissible heuristic.
- A* is complete and optimal.
- **A* is a key AI algorithm**