Informed Search

Chap. 4

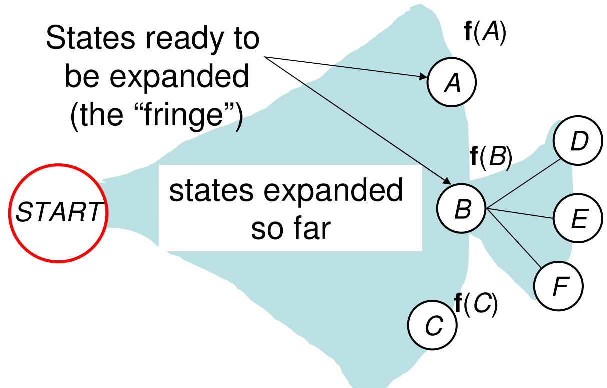
Material in part from http://www.cs.cmu.edu/~awm/tutorials

Uninformed Search Complexity

- N = Total number of states
- B = Average number of successors (branching factor)
- L = Length for start to goal with smallest number of steps
- Q = Average size of the priority queue
- $Lmax$ = Length of longest path from *START* to any state

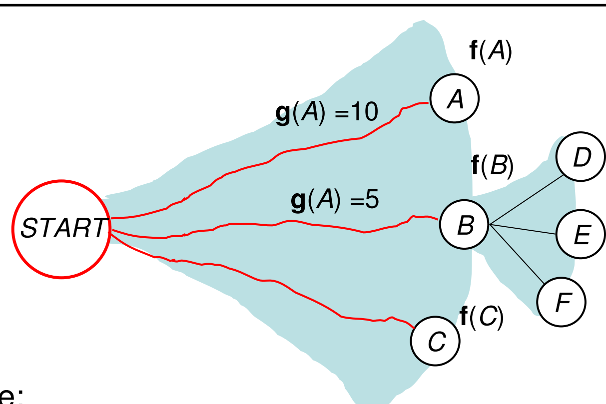
	Algorithm	Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	Y, If all trans. have same cost	$O(Min(N,B^L))$	$O(Min(N,B^L))$
BIBFS	Bi- Direction. BFS	Y	Y, If all trans. have same cost	$O(Min(N,2B^{L/2}))$	$O(Min(N,2B^{L/2}))$
UCS	Uniform Cost Search	Y, If cost > 0	Y, If cost > 0	$O(\log(Q)*B^{C/\varepsilon}))$	$O(Min(N,B^{C/\varepsilon}))$
PCDFS	Path Check DFS	Y	N	$O(B^{Lmax})$	$O(BL_{max})$
MEMD FS	Memorizing DFS	Y	N	$O(Min(N,B^{Lmax}))$	$O(Min(N,B^{Lmax}))$
IDS	Iterative Deepening	Y	Y, If all trans. have same cost	$O(B^L)$	$O(BL)$

Search Revisited



1. Store a value **f**(*s*) at each state *s*
2. Choose the state with lowest **f** to expand next
3. Insert its successors

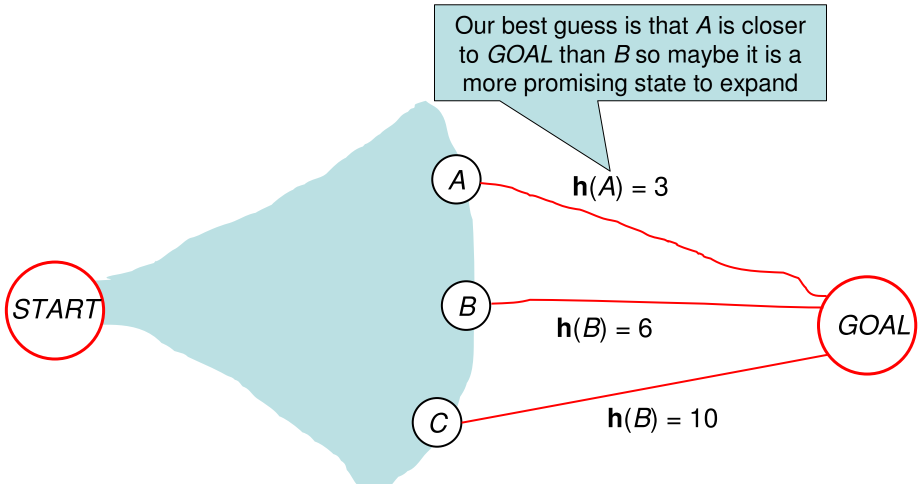
If **f**(.) is chosen carefully, we will eventually find the lowest-cost sequence



Example:

• UCS (Uniform Cost Search): **f**(*A*) = **g**(*A*) = total cost of current shortest path from *START* to *A*
• Store states awaiting expansion in a priority queue for efficient retrieval of minimum **f**
• Optimal → Guaranteed to find lowest cost sequence, *but*……

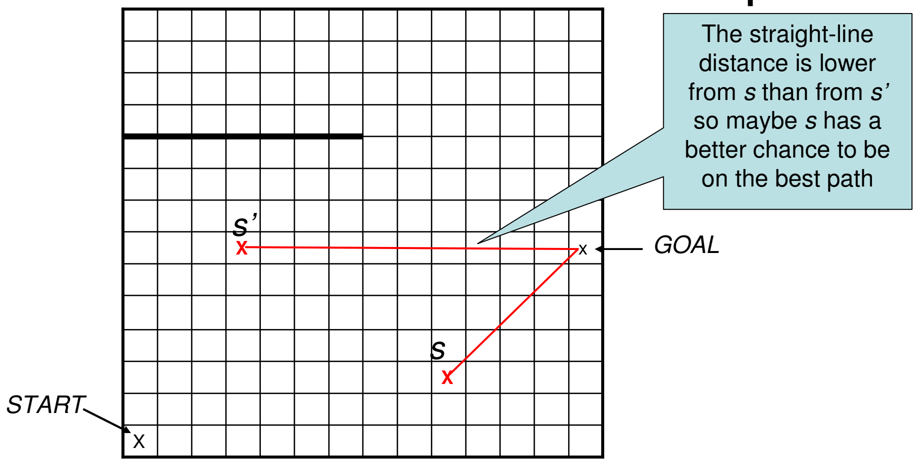
• Problem: No guidance as to how "far" any given state is from the goal
• Solution: Design a function **h**(.) that gives us an estimate of the distance between a state and the goal



Heuristic Functions

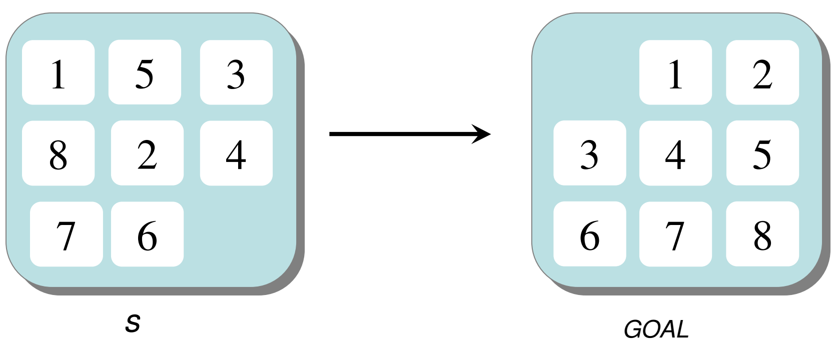
- **h**(.) is a heuristic function for the search problem
- **h**(*s*) = estimate of the cost of the shortest path from *s* to GOAL
- **h**(.) cannot be computed solely from the states and transitions in the current problem → If we could, we would already know the optimal path!
- **h**(.) is based on external knowledge about the problem → *informed* search
- Questions:
  1. Typical examples of **h**?
  2. How to use **h**?
  3. What are desirable/necessary properties of **h**?

Heuristic Functions Example



- **h**(*s*) = Euclidean distance to *GOAL*

Heuristic Functions Example



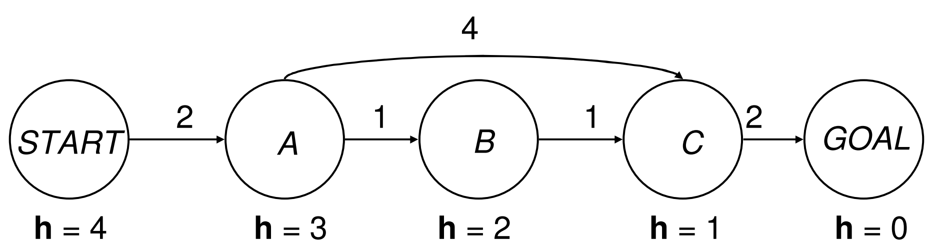
- How could we define **h**(*s*)?

First Attempt: Greedy Best First Search

- Simplest use of heuristic function: Always select the node with smallest **h**(.) for expansion (i.e., **f**(*s*) = **h**(*s*))

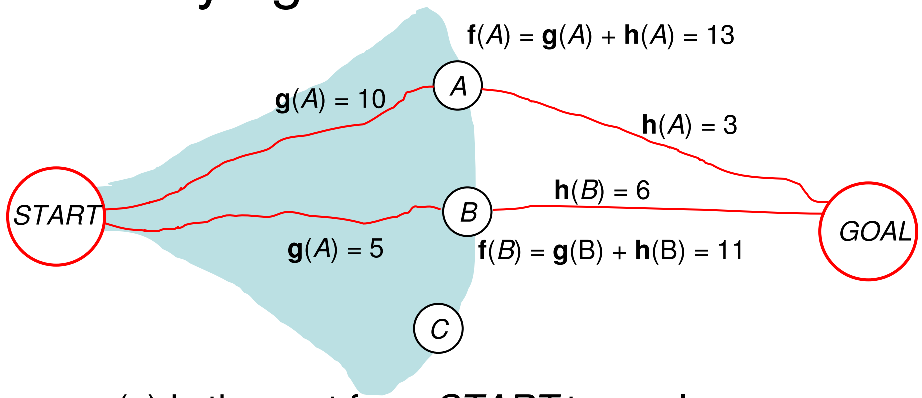
Initialize *PQ*
Insert *START* with value **h**(*START*) in *PQ*
While (*PQ* not empty and no goal state is in *PQ*)
    Pop the state *s* with the minimum value of **h** from *PQ*
    For all *s'* in **succs**(*s*)
        If *s'* is not already in *PQ* and has not already been visited
            Insert *s*' in *PQ* with value **h**(*s*')

Problem



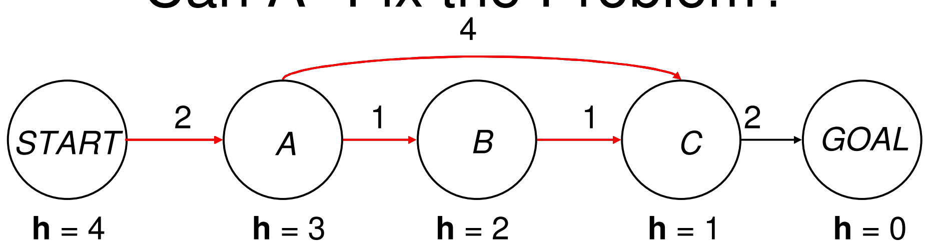
- What solution do we find in this case?
- Greedy search clearly not optimal, even though the heuristic function is non-stupid

Trying to Fix the Problem



- **g**(*s*) is the cost from *START* to *s* only
- **h**(*s*) estimates the cost from *s* to *GOAL*
- Key insight: **g**(*s*) + **h**(*s*) estimates the ***total*** cost of the cheapest path from START to GOAL going through *s*
- → A* algorithm

Can A* Fix the Problem?



{(*START*,4)}

{(*A*,5)}

(**f**(*A*) = **h**(*A*) + **g**(*A*) = 3 + **g**(*START*) + **cost**(*START*, *A*) = 3 + 0 + 2)

{(*B*,5) (*C*,7)}

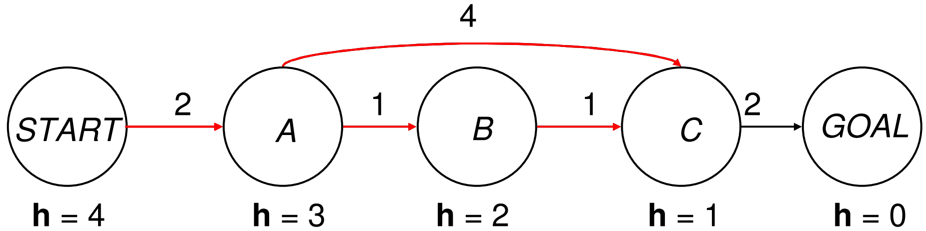
(**f**(*C*) = **h**(*C*) + **g**(*C*) = 1 + **g**(*A*) + **cost**(*A*, *C*) = 1 + 2 + 4)

{(*C*,5)}

(**f**(*C*) = **h**(*C*) + **g**(*C*) = 1 + **g**(*B*) + **cost**(*B*, *C*) = 1 + 3 + 1)

{(*GOAL*,6)}

Can A* Fix the Problem?



C is placed in the queue with backpointers {*A*,*START*}

{(*START*,4)}

{(*A*,5)}

(**f**(*A*) = **h**(*A*) + **g**(*A*) = 3 + **g**(*START*) + **cost**(*START*, *A*) = 3 + 0 + 2)

{(*B*,5) (*C*,7)}

A lower value of **f**(*C*) is found with backpointers {*B*,*A*,*START*}

(**f**(*C*) = **h**(*C*) + **g**(*C*) = 1 + **g**(*A*) + **cost**(*A*, *C*)

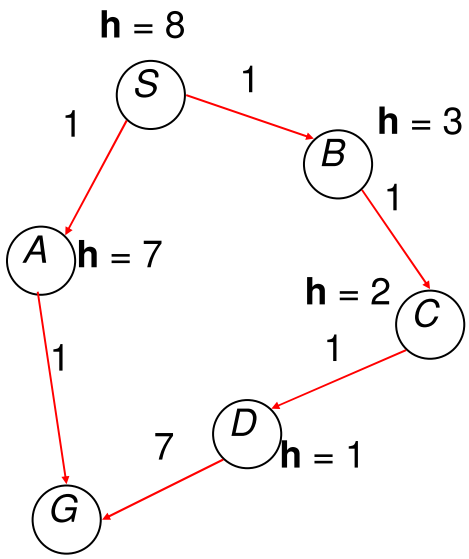
{(*C*,5)}

(**f**(*C*) = **h**(*C*) + **g**(*C*) = 1 + **g**(*B*) + **cost**(*B*, *C*) = 1 + 3 + 1)

{(*GOAL*,6)}

- Termination condition
- Revisiting states
- Algorithm
- Optimality
- Avoiding revisiting states
- Choosing good heuristics
- Reducing memory usage

A* Termination Condition



Queue:

{(B,4) (A,8)}

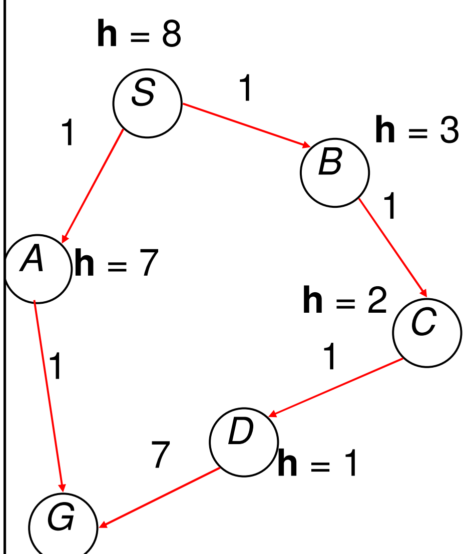
{(C,4) (A,8)}

{(D,4) (A,8)}

{(A,8) (G,10)}

- Stop when GOAL is popped from the queue!

A* Termination Condition



Queue:

{(B,4) (A,8)}

{(C,4) (A,8)}

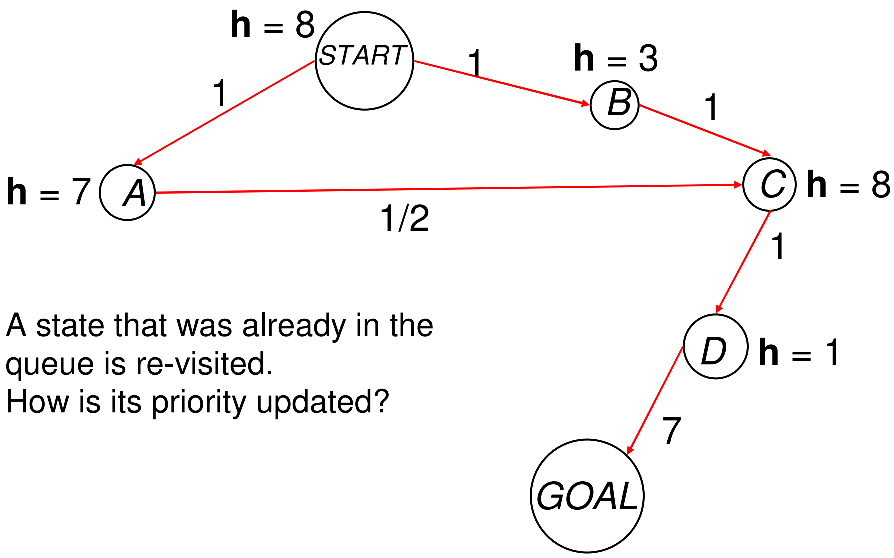
{(D,4) (A,8)}

{(A,8) (G,10)}

We have encountered *G* before we have a chance to visit the branch going through *A*. The problem is that at each step we use only an estimate of the path cost to the goal

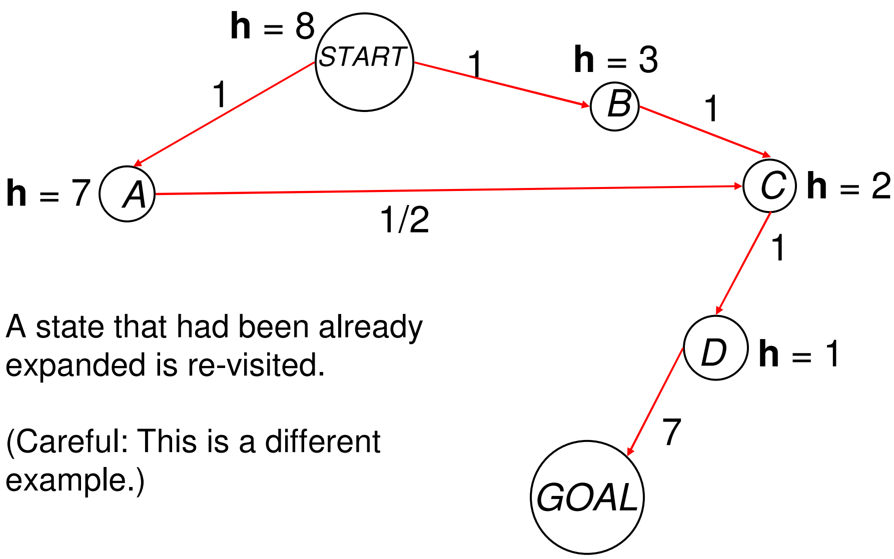
- Stop when GOAL is popped from the queue!

Revisiting States



A state that was already in the queue is re-visited.
How is its priority updated?

Revisiting States



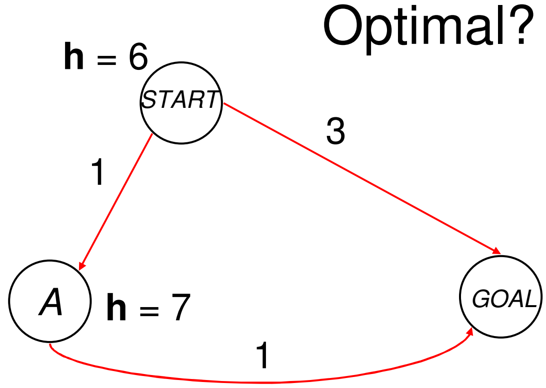
A state that had been already expanded is re-visited.

(Careful: This is a different example.)

Pop state *s* with lowest **f**(*s*) in queue
If *s* = *GOAL*
  return *SUCCESS*
Else expand *s*:
  For all *s'* in **succs** (*s*):
    f' = **g**(*s*') + **h**(*s'*) = **g**(*s*) + **cost**(*s*,*s'*) + **h**(*s'*)
    If (s' not seen before OR
      s' previously expanded with **f**(*s'*) > *f'* OR
      s' in PQ with with **f**(*s'*) > *f'*)
        Promote/Insert *s'* with new value *f'* in *PQ*
        previous(*s*') ← *s*
    Else
        Ignore *s*' (because it has been visited and its current path cost **f**(*s'*) is still the lowest path cost from *START* to *s*')

A* Algorithm (inside loop)

Under what Conditions is A* Optimal?



{(S*TART*,6)}
{(G*OAL*,3) (A,8)}

Final path:
{*START*, *GOAL*}
with cost = 3

- Problem: **h**(.) is a poor estimate of path cost to the goal state

Admissible Heuristics

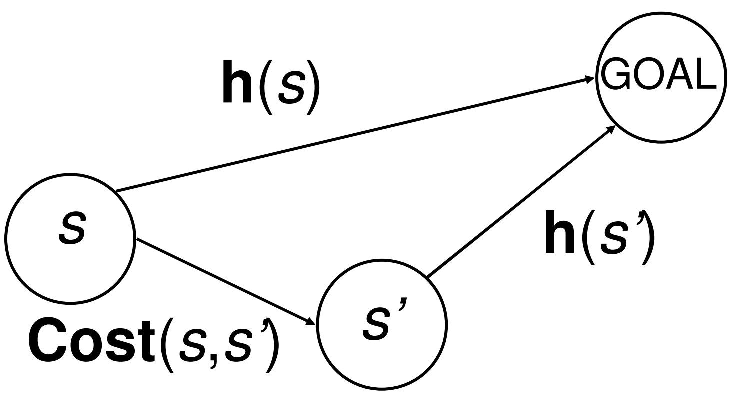
- Define **h***(*s*) = the true minimal cost to the goal from *s*
- **h** is admissible if

h(*s*) <= **h***(*s*) for all states *s*

- In words: An admissible heuristic never overestimates the cost to the goal. "Optimistic" estimate of cost to goal.

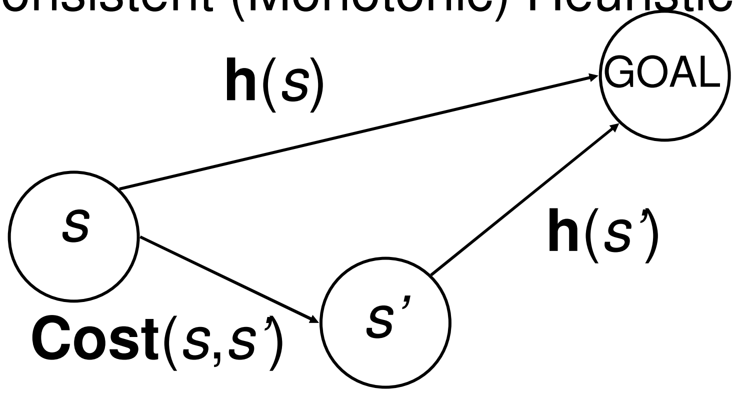
A* is guaranteed to find the optimal path if **h** is admissible

Consistent (Monotonic) Heuristics



h(*s*) <= **h**(*s'*) + **cost**(*s*,*s'*)

Consistent (Monotonic) Heuristics



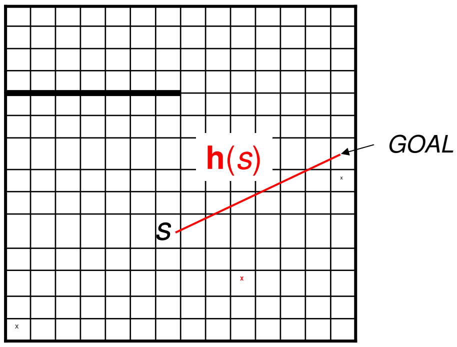
Sort of triangular inequality implies that path cost always increases + need to expand node only once

h(*s*) <= **h**(*s'*) + **cost**(*s*,*s'*)

Pop state *s* with lowest **f**(*s*) in queue
If *s* = *GOAL*
  return *SUCCESS*
Else expand *s*:
  For all *s'* in **succs** (*s*):
    f' = **g**(*s'*) + **h**(*s'*) = **g**(*s*) + **cost**(*s*,*s'*) + **h**(*s'*)
    If (s' not seen before OR
      ~~s' previously expanded with **f**(*s'*) > *f'* OR~~
      s' in PQ with with **f**(*s'*) > *f'*)
        Promote/Insert *s'* with new value *f'* in *PQ*
        previous(*s'*) ← *s*
    Else
        Ignore *s'* (because it has been visited and its current path cost **f**(*s'*) is still the lowest path cost from *START* to *s'*)

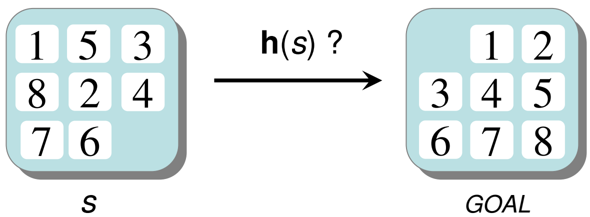
If **h** is consistent

Examples



For the navigation problem: The length of the shortest path is at least the distance between *s* and *GOAL* → Euclidean distance is an admissible heuristic

What about the puzzle?



Comparing Heuristics

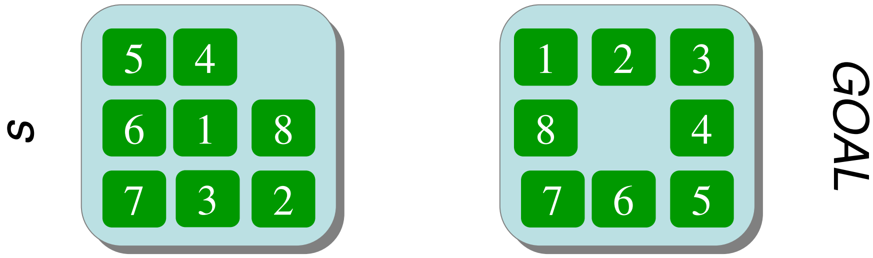
h$_1$ = misplaced tiles

h$_2$ = Manhattan distance

	L = 4 steps	*L* = 8 steps	*L* = 12 steps
Iterative Deepening	112	6,300	3.6×10^6
A* with heuristic **h**$_1$	13	39	227
A* with heuristic **h**$_2$	12	25	73

- Overestimates A* performance because of the tendency of IDS to expand states repeatedly
- Number of states expanded does not include log() time access to queue

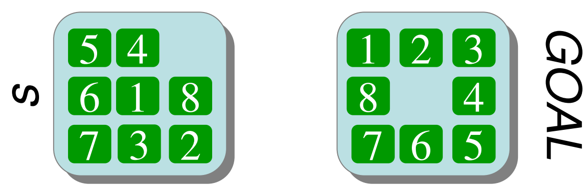
Example from Russell&Norvig



$\mathbf{h}_1(s) = 7$

$\mathbf{h}_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$

Comparing Heuristics



$\mathbf{h}_1(s) = 7$

$\mathbf{h}_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$

$\mathbf{h}_2$ is larger than $\mathbf{h}_1$ and, at same time, A* seems to be more efficient with $\mathbf{h}_2$.

Is there a connection between these two observations?

$\mathbf{h}_2$ *dominates* $\mathbf{h}_1$ if $\mathbf{h}_2(s) >= \mathbf{h}_1(s)$ for all s

For any two heuristics $\mathbf{h}_2$ and $\mathbf{h}_1$:
If $\mathbf{h}_2$ dominates $\mathbf{h}_1$ then A* is more efficient (expands fewer states) with $\mathbf{h}_2$

Intuition: since $\mathbf{h} <= \mathbf{h}^*$, a larger $\mathbf{h}$ is a better approximation of the true path cost

Limitations

- Computation: In the worst case, we may have to explore all the states → O(*N*)
- The good news: A* is optimally efficient → For a given **h**(.), no other optimal algorithm will expand fewer nodes
- The bad news: Storage is also potentially large → O(*N*)

IDS (Iterative Deepening Search)

- Need to make DFS optimal
- IDS (Iterative Deepening Search):
  - Run DFS by searching only path of length 1 (DFS stops if length of path is greater than 1)
  - If that doesn't find a solution, try again by running DFS on paths of length 2 or less
  - If that doesn't find a solution, try again by running DFS on paths of length 3 or less
  - ………..
  - Continue until a solution is found

Example: IDA* (Iterative Deepening A*)

- Same idea as Iterative Deepening DFS except use **f**(*s*) to control depth of search instead of the number of transitions
- Example, *assuming integer costs*:

1. Run DFS, stopping at states *s* such that **f**(*s*) > 0
   Stop if goal reached
2. Run DFS, stopping at states *s* such that **f**(*s*) > 1
   Stop if goal reached
3. Run DFS, stopping at states *s* such that **f**(*s*) > 2
   Stop if goal reached

……..Keep going by increasing the limit on **f** by 1 every time

- Complete (assuming we use loop-avoiding DFS)
- Optimal
- More expensive in computation cost than A*
- Memory order *L* as in DFS

Summary

- Informed search and heuristics
- First attempt: Best-First Greedy search
- A* algorithm
  - Optimality
  - Condition on heuristic functions
  - Completeness
  - Limitations, space complexity issues
  - Extensions

Nils Nilsson. Problem Solving Methods in Artificial Intelligence. McGraw Hill (1971)
Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving (1984)
Chapters 3&4 Russel & Norvig