COMPSCI 761: ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

INFORMED SEARCH III

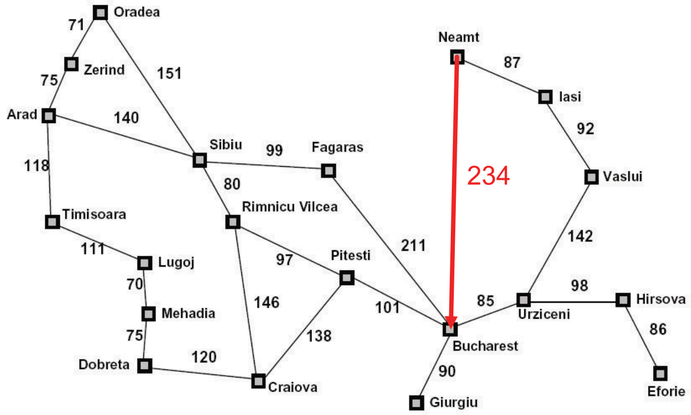
Anna Trofimova, July 2022

TODAY

- A* Search Recap
- Creating Admissible & Consistent Heuristic
- Iterative Deepening A* Search
- Informed Search Summary

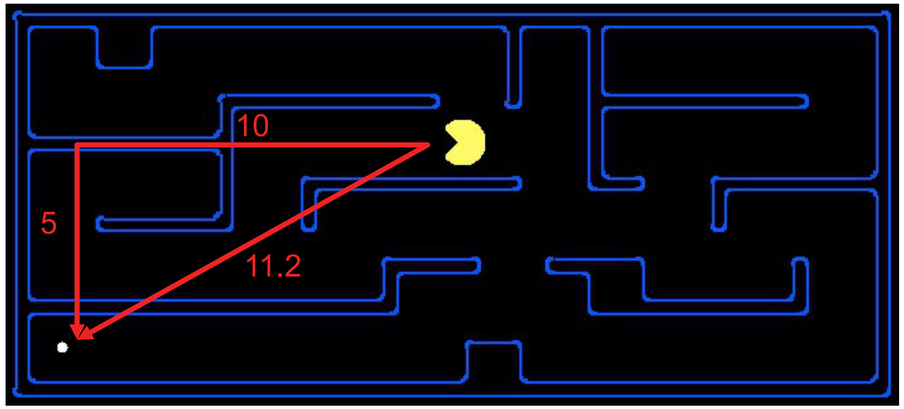
HEURISTIC

- Often, admissible heuristics are solutions to ***relaxed problems***, where some of the constraints of the original problem have been removed and thereby new actions may be available



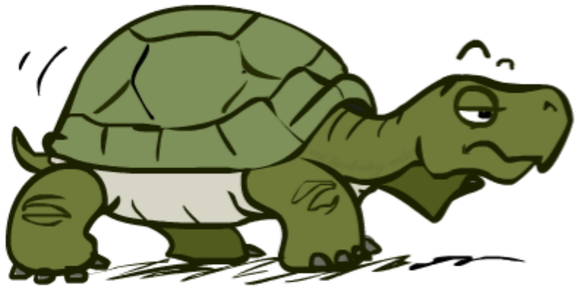
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(n)

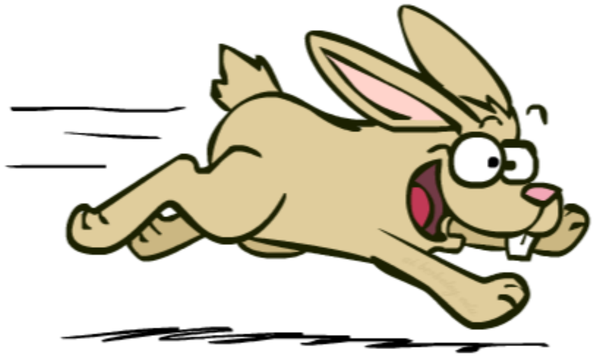


- If Problem ***P2*** is a relaxed version of ***P1*** then $h_2^*(n) \leq h_1^*(n)$ for every s, so $h_2^*(n)$ is admissible for ***P1***

RECAP: A* IS A COMBINATION OF UCS AND GREEDY



Uniform-Cost Search



Greedy Search

A*

RECAP: A* SEARCH

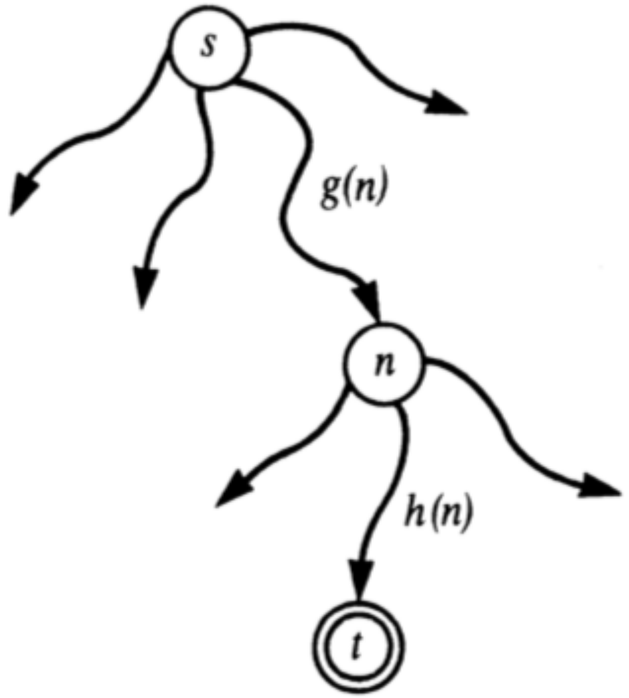
- A* Search uses evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost from initial node to node n
  - $h(n)$ = estimated cost of cheapest path from n to goal
  - $f(n)$=estimated total cost of cheapest solution through node n
- Combines uniform-cost search and greedy search
- Greedy Search minimizes $h(n)$
  - efficient but not optimal or complete
- Uniform Cost Search minimizes $g(n)$
  - optimal and complete but not efficient

HEURISTIC FUNCTION

Heuristic estimate *f(n) of* the cost of the cheapest paths from **s** to **t** via **n**: f(n) = g(n)+ h(n)

g(n) is an estimate of the cost of an optimal path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.



A* SEARCH – CONDITIONS FOR OPTIMALITY

- Heuristic ***h*** is called **admissible** if
- $\forall n\ h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to the goal

- If ***h*** is **admissible** then ***f*** **(*n*)** never overestimates the actual cost of the best solution through *n.*

- Example: $\boldsymbol{h_{SLD}}$ is admissible for delivery robot problem because the shortest path between any two points is a line.

- Theorem: A* Search finds optimal solution if ***h*****(n)** is admissible.

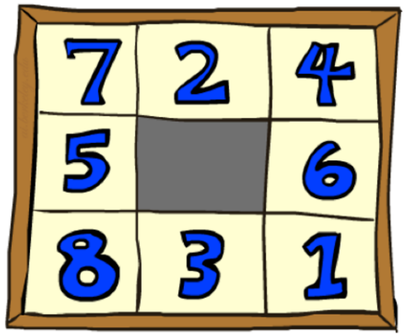
RECAP: OPTIMALITY OF A* SEARCH

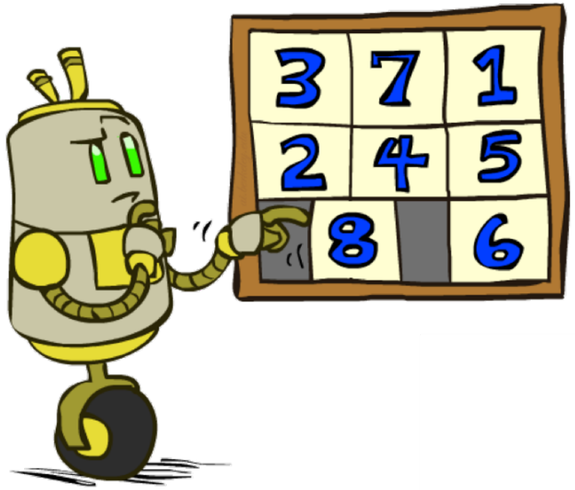
Algorithm	Completeness	Admissibility	Space	Time
A* Search	guaranteed	guaranteed if heuristic is admissible/consistent	$O(b^d)$	$O(b^d)$

- Complete: Yes, unless there are infinitely many nodes with f ≤ cost of solution
- Time: Exponential
- Space: Keeps all nodes is memory
- Optimal: Yes (assuming *h*(n) is admissible).

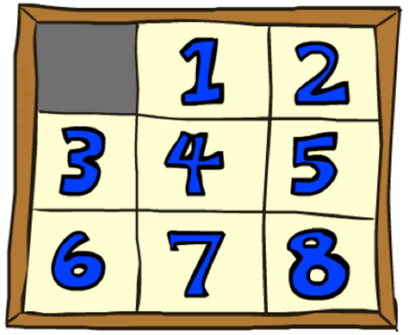
EXAMPLEL 8 PUZZLE

START STATE





GOAL STATE



- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?
-

EXAMPLEL 8 PUZZLE

- What are the states?
  The configuration of tiles on board
- How many states?
  9! = 362,880 tile configurations
- What are the actions?
  Moving an adjacent tile horizontally or vertically into the “blank” space
- How many successors from the start state?
  At most 4 successors
- What should the costs be?
  Either 1 or tile id or tile location

8 PUZZLE

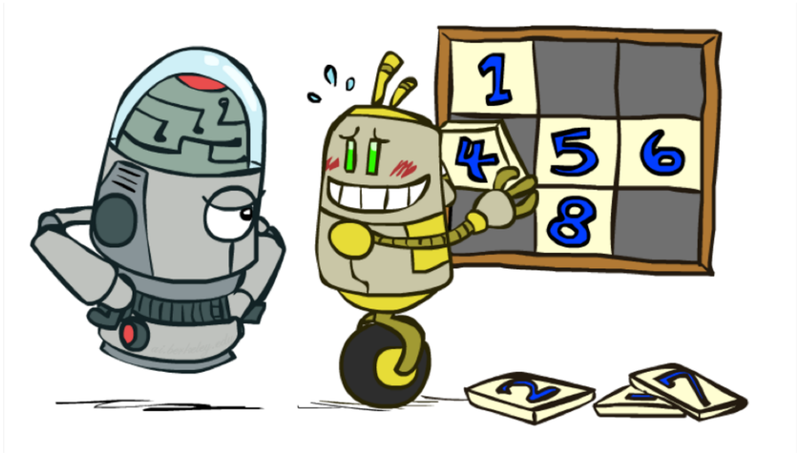
Heuristic: Number of tiles misplaced

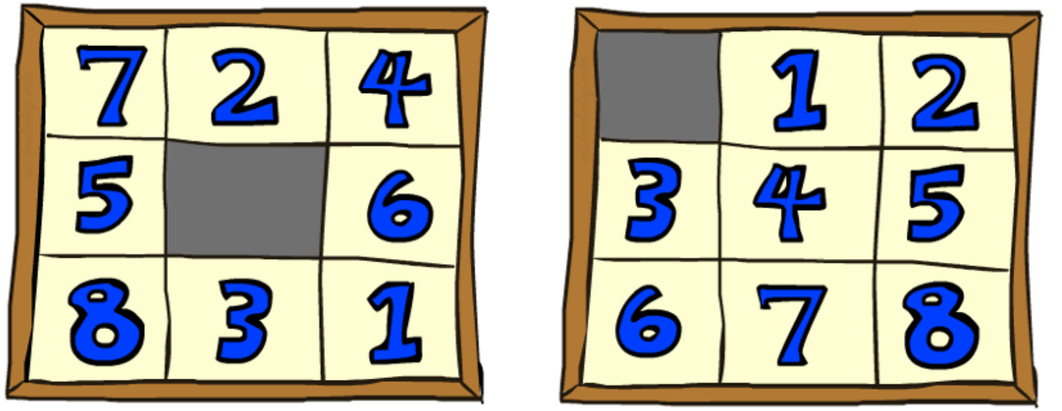
Why is it admissible?

h(start) = 8

This is a relaxed-problem heuristic

.

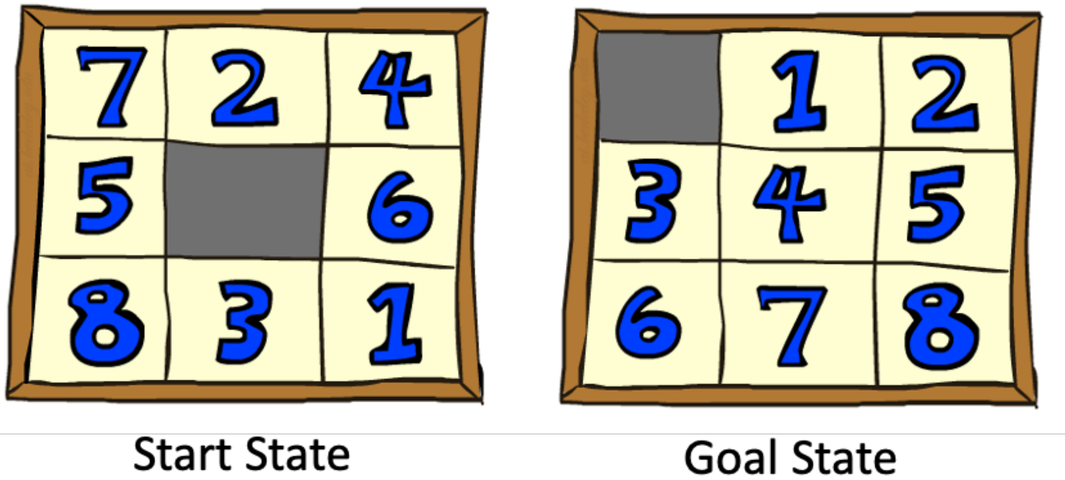




Start State Goal State

8 PUZZLE

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total Manhattan distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18

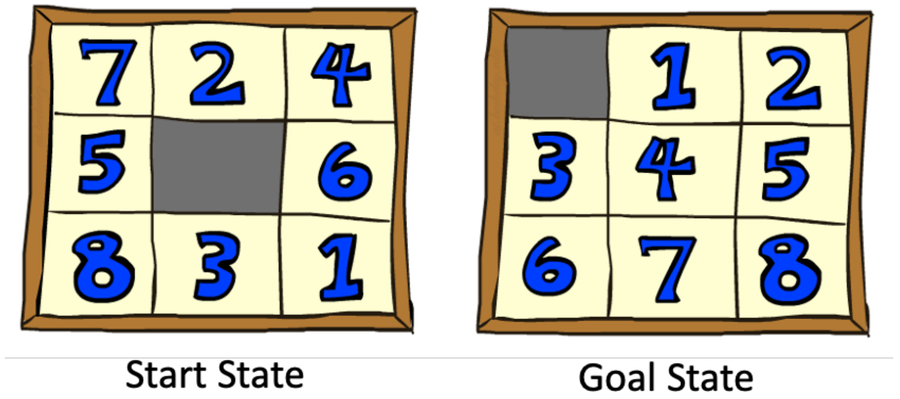


.

8 PUZZLE

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

WHICH ONE IS BETTER?



Option A: Number of tiles misplaced

h(start) = 8

Option B: Manhattan distance

h(start) = 8

Option C: Actual Cost

h(start) = ?

WHAT DOES h_1 BETTER THAN h_2 MEANS

- What we'd like h_1 better than h_2 for problem P to mean is that A* solving P using h_1 takes less time/memory than A* using h_2
- However, that is hard to predict in general.
- So, instead we look at node expansions as a proxy for time/memory
- If A* expands fewer nodes using h_1 than when using h_2 we say it is "better"
- Happily, we can characterise when this is likely to happen, namely when h_1's values are not lower than h_2's values.

.

TRIVIAL HEURISTIC, DOMINANCE

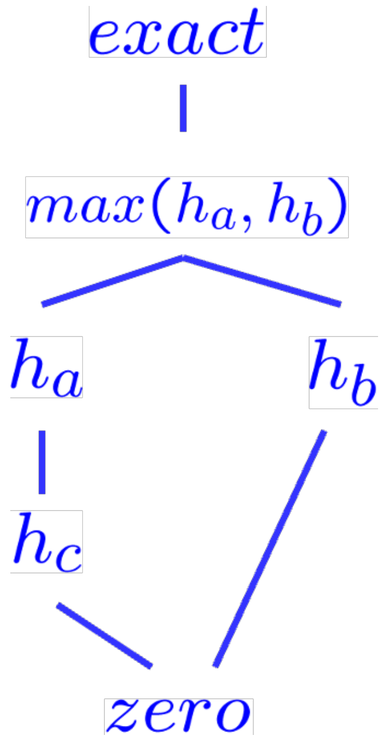
- Dominance: if $h_a(n) \geq h_c(n)$ for all *n* (both admissible) then *h*2 dominates *h*1 and is better for search. So the aim is to make the heuristic *h*() as large as possible, but without exceeding $h^*()$.

  $h_a \geq h_c$ if $\forall n: h_a(n) \geq h_c(n)$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

  $h(n) = max(h_a(n), h_b(n))$

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



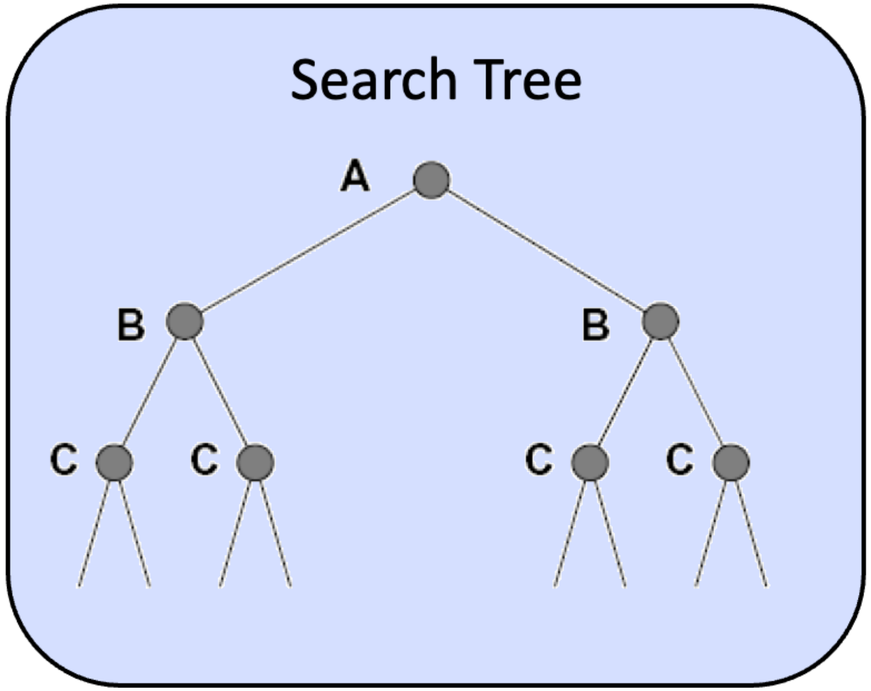
SUMMARY: HEURISTIC

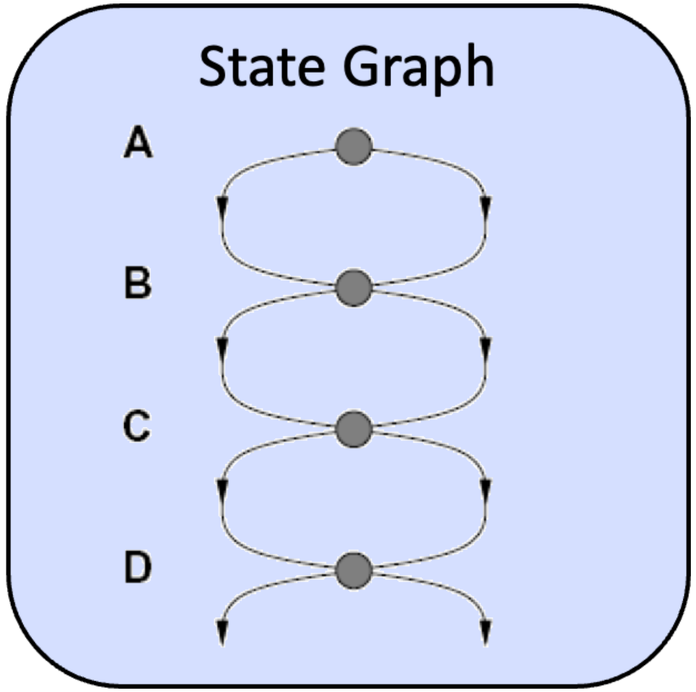
- Admissible heuristics can often be derived from the exact solution cost of a simplified or “relaxed” version of the problem. (i.e. with some of the constraints weakened or removed)
- Comparing heuristics

.

TREE SEARCH: EXTRA WORK!

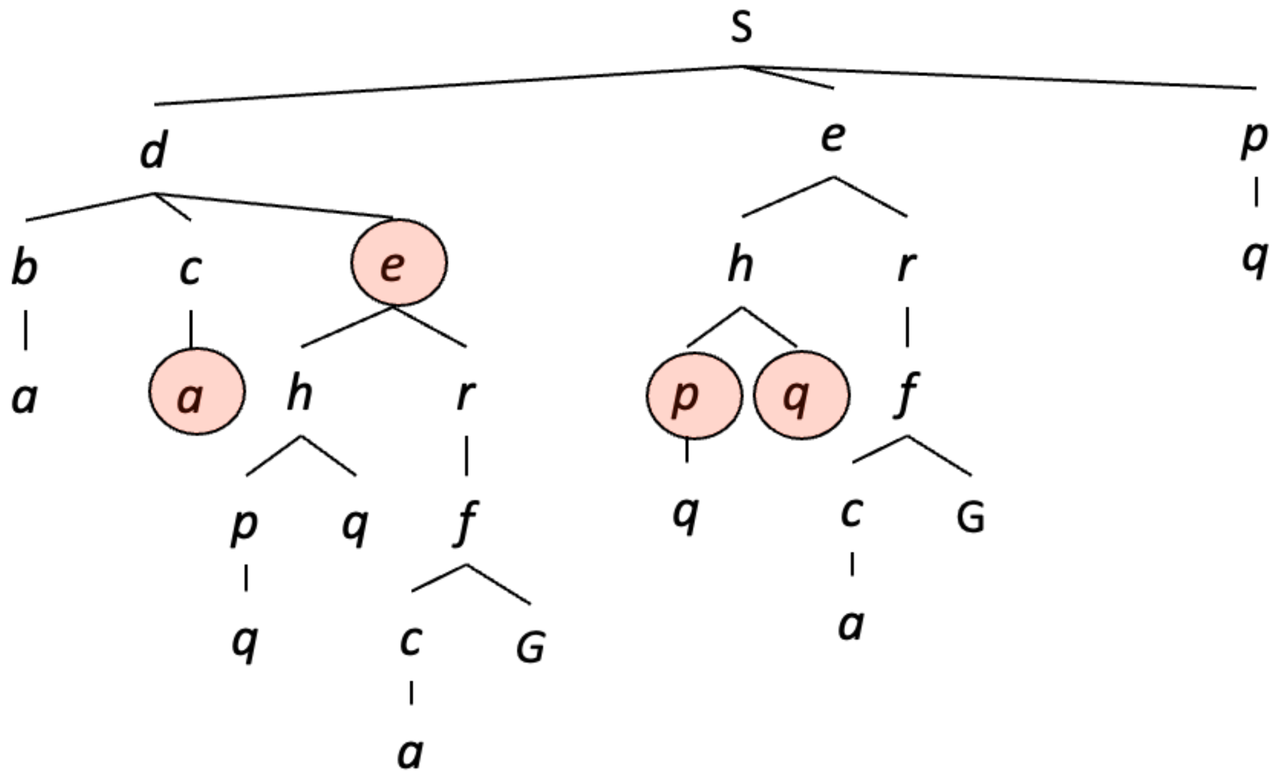
Failure to detect repeated states can cause exponentially more work.





GRAPH SEARCH

In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

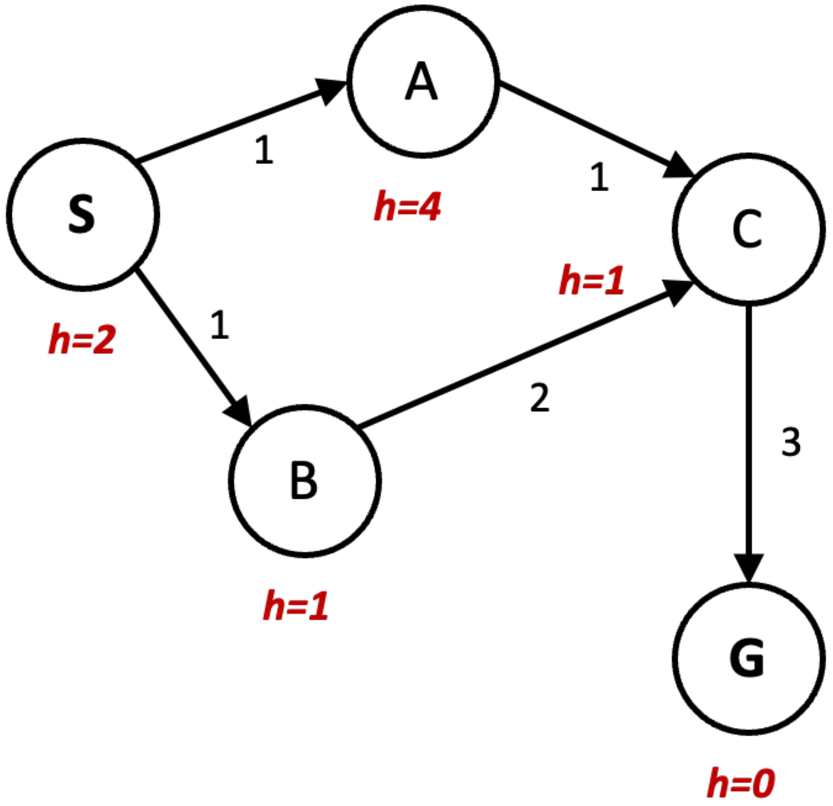


GRAPH SEARCH

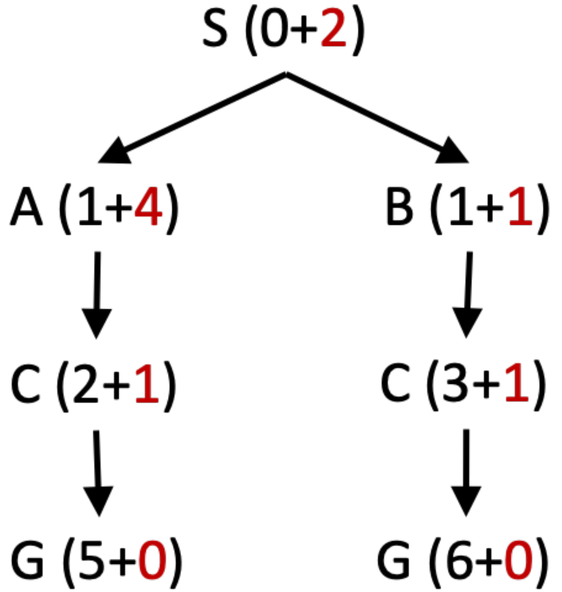
- Idea: never **expand** a state twice (if not necessary)
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new expand and add to closed set
- **Important**: store the closed set as a set (hash table), **not a list**
- Can graph search wreck completeness? Why/why not?
- How about optimality?

A* GRAPH SEARCH GONE WRONG?

State space graph

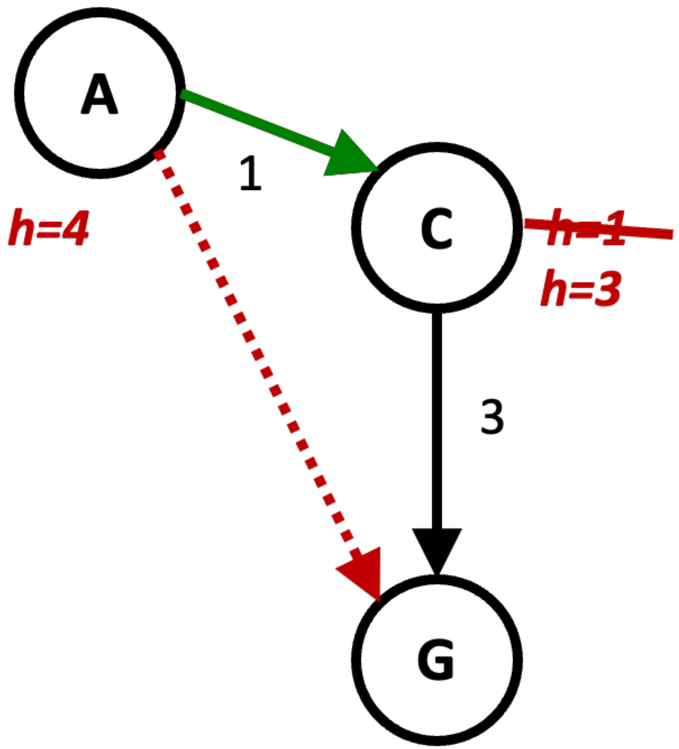


Search tree



C might need to be re-opened!!!!

CONSISTENCY OF HEURISTIC



Admissibility: heuristic cost ≤ actual cost to goal

Consistency: heuristic “edge” cost ≤ actual cost for each edge

$$|h(A) - h(C)| \leq c(A, C)$$

Or $h(C) - c(A, C) \leq h(A) \leq c(A, C) + h(C)$ (triangle inequality)

Consequences of consistency:

- The f value along a path never decreases:

$$h(A) \leq c(A, C) + h(C) \Rightarrow g(A) + h(A) \leq g(A) + c(A, C) + h(C)$$

- A* graph search is optimal (no-reopenings)

SYNONYMS

- *Fringe, frontier, & open list* all mean the same thing, the nodes that have been generated but not yet expanded. This is used to keep the generated nodes in the order they will be expanded.

- *Explored & closed list* both mean the states that have been expanded. This is used both to reconstruct the solution path through back-pointers and to do duplicate state elimination.

- *Successors & children* both mean the nodes that all the outgoing edges of a node connect to.

.

A* - WHAT TO DO IF *h* IS NOT CONSISTENT

- So far, we have assumed that *h* is consistent (& thus admissible) what if *h* is not consistent (but still admissible)??
- If *h* is not consistent then we have no guarantee that the 1st time state *s* is picked to be expanded that its cheapest path from *i* has been found!
- This means that if *s* is already in the closed list, a cheaper path may be found & this new path will need to "replace" the old one.

.

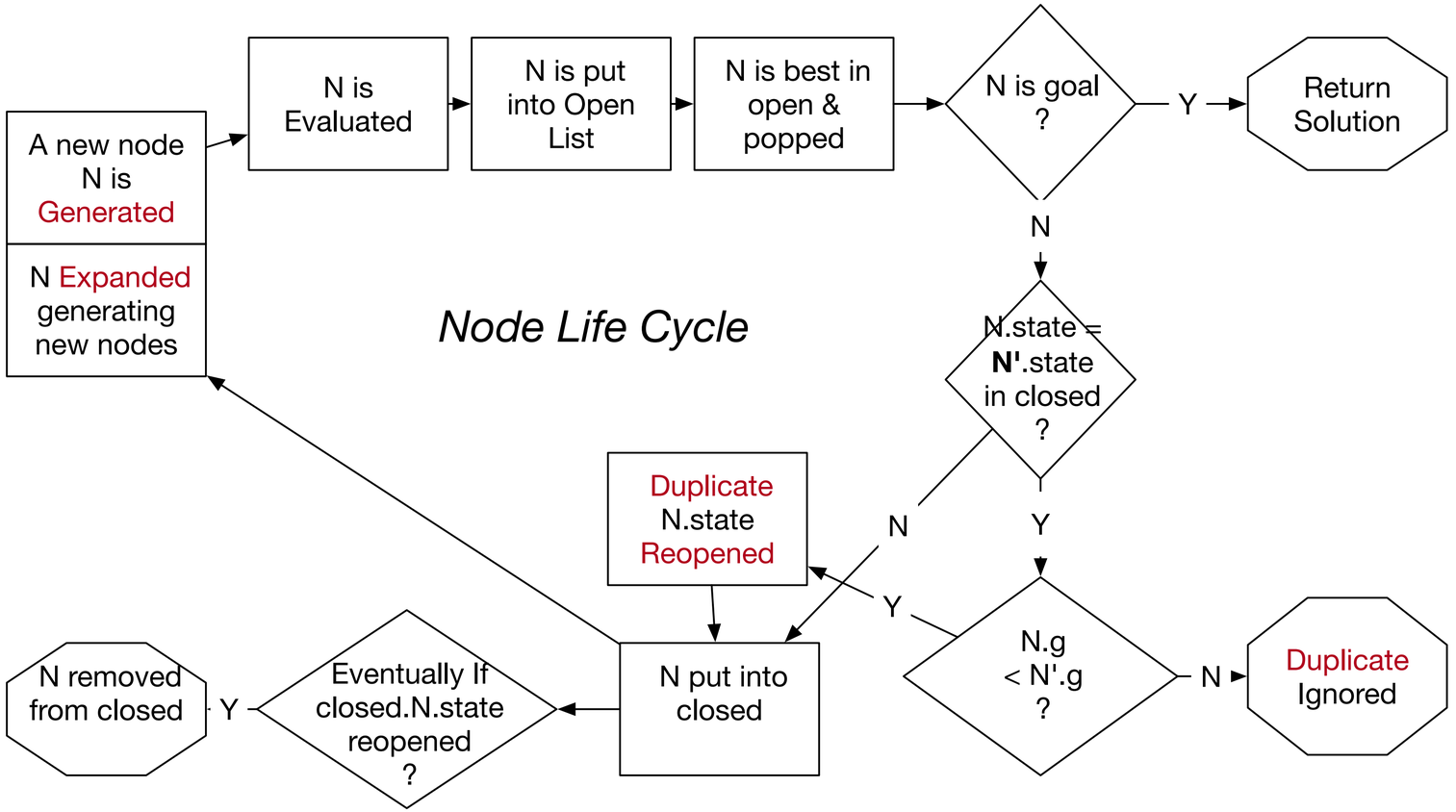
AN A* THAT REQUIRES A CONSISTENT *h*

```
function Graph-Search(problem) return solution or fail
    closed <- {}
    open <- {newNode(problem.initState)}
    loop do
       if open = {} then return fail
       n <- remove-best(open)
       if goal(n.state) then return n
       if n.state is in closed then continue
      closed =+ n
      open =+ successorsOf(n)
```

AN A* THAT DOESN'T REQUIRES A CONSISTENT *h*

```
function Graph-Search(problem) return solution or fail
    closed <- {}
    open <- {newNode(problem.initState)}
    loop do
       if open = {} then return fail
       n <- remove-best(open)
       if goal(n.state) then return n
       if n.state is in closed then
           if g(n.state) < g(closed[n.state]) then remove n.state from closed
           else continue
      closed =+ n
      open =+ successorsOf(n)
```

NODE LIFE CYCLE



SUMMARY: A*

A* uses both backward costs *g* & (estimates of) forward costs *h*

A* is optimal with admissible / consistent heuristics

Heuristic design is key: often use relaxed problems

.

ITERATIVE DEEPENING A* SEARCH

- Iterative Deepening A* is a low-memory variant of A* which performs a series of depth-first searches but cuts off each search when the sum $f() = g() + h()$ exceeds some pre-defined threshold.

- The threshold is steadily increased with each successive search. It starts at the estimate of the cost at the initial state, and increases for each iteration of the algorithm. At each iteration, the threshold used for the next iteration is the minimum cost of all values that exceeded the current threshold.

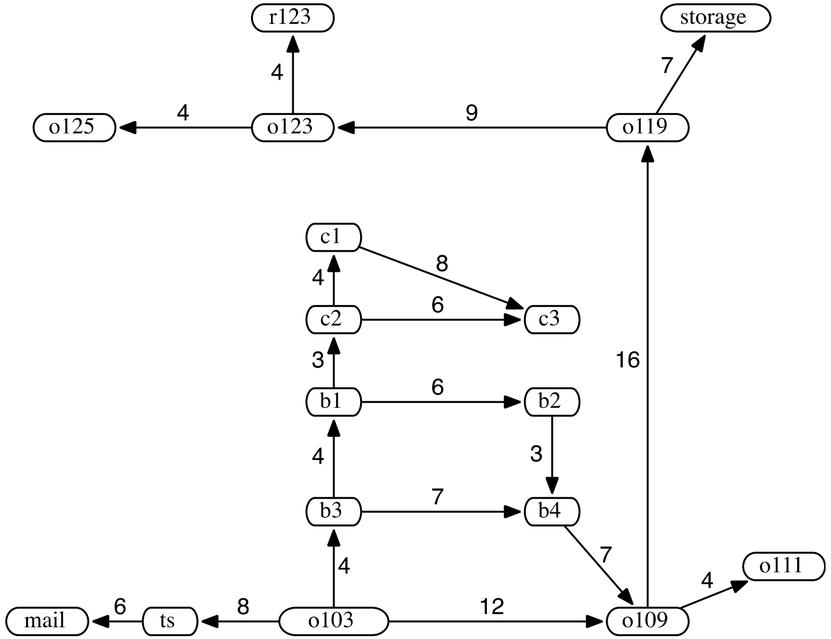
- IDA* is asymptotically as efficient as A* for domains where the number of states grows exponentially.

OPTIMALITY OF ITERATIVE DEEPENING A* SEARCH

Algorithm	Completeness	Admissibility	Space	Time
Iterative Deepening A* Search	guaranteed	guaranteed if heuristic is admissible/consistent	O(bd)	$O(b^d)$

- Complete: Yes, unless there are infinitely many nodes with f ≤ cost of solution
- Time: Exponential
- Space: it need only store a stack of nodes which represents the branch of the tree it is expanding
- Optimal: Yes (assuming *h*(n) is admissible).

IDA* SEARCH - THE DELIVERY ROBOT



$$
\begin{array}{rcrcrc}
h(mail) & = & 26 & h(ts) & = & 23 \\
h(o109) & = & 24 & h(o111) & = & 27 \\
h(o123) & = & 4 & h(o125) & = & 6 \\
h(b1) & = & 13 & h(b2) & = & 15 \\
h(b4) & = & 18 & h(c1) & = & 6 \\
h(c3) & = & 12 & h(storage) & = & 12
\end{array}
\qquad
\begin{array}{rcrcrc}
h(o103) & = & 21 \\
h(o119) & = & 11 \\
h(r123) & = & 0 \\
h(b3) & = & 17 \\
h(c2) & = & 10
\end{array}
$$

IS IDA* BETTER THAN A*

- Short answer: memory-wise yes, time-wise sometimes
- Long answer:
  - IDA* uses a priority queue
  - IDA* doesn’t need to deal with neither open or closed lists, but A* does
  - Runtime will depend on how frequently duplicate states will be encountered and iterations are done
  - If you don’t have much memory, but can wait for the solution - use IDA*

SUMMARY OF INFORMED SEARCH

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimise cost from current node *n* to the goal.
- A* combines the advantages of Uniform-Cost Search and Greedy Search
- A* is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for A*. IDA* is a low-memory variant.

- Informed search makes use of problem-specific knowledge to guide progress of search

- This can lead to a significant improvement in performance

- Much research has gone into admissible heuristics
  - Even on the automatic generation of admissible heuristics