# COL333/671: Introduction to AI

Semester II, 2020

## Solving Problems by Searching
## Informed Search

Rohan Paul

# Outline

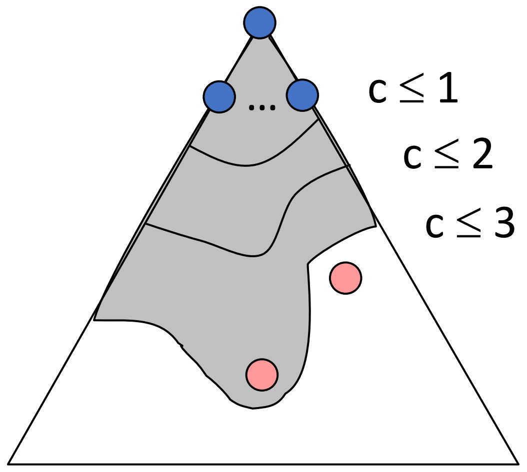
- Last Class
  - Uninformed Search
- This Class
  - Informed Search
    - Key idea behind Informed Search
    - Best First Search
    - Greedy Best First Search
    - A* Search: evaluation Function
- Reference Material
  - AIMA Ch. 3

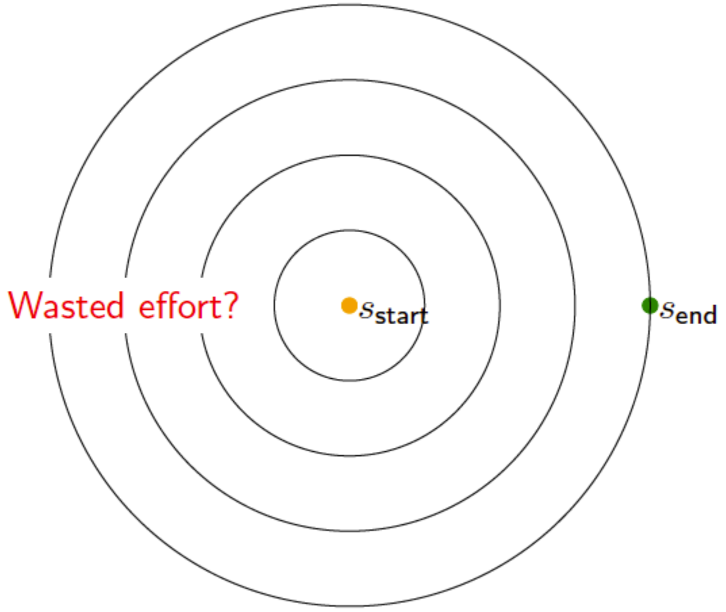
# Acknowledgement

**These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Dan Klein, Nicholas Roy and others.**

# Uninformed Search

- Uniform Cost Search
  - Expand the lowest cost path
  - Complete
  - Optimal
- Problem
  - Explores options in every “direction”
  - No information about goal location
- Informed Search
  - Use problem-specific knowledge beyond the definition of the problem to guide the search towards the goal.



# Recall: Tree Search

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     expand the chosen node, adding the resulting nodes to the frontier
```

# Best First Search

- **Best First Search**
  - Always choose the node from frontier that has the best evaluation (according to a function).
  - The search orders nodes in the frontier (via priority queue) for expansion using this evaluation.

- **Incorporate an evaluation of every node**
  - Lets say we evaluate a node with a function f() value.
  - Estimates the desirability of a node for the purposes of potentially reaching the goal. A search strategy is defined by picking the order of node expansion.
  - Expand most desirable unexpanded node. Order the nodes in frontier in decreasing order of desirability.

# Evaluation Functions for Uninformed and Informed Search

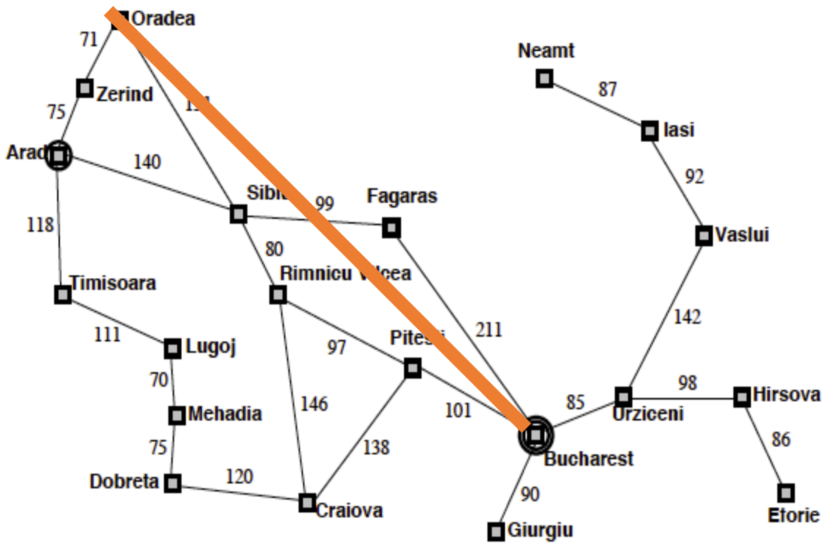
- Uninformed search methods expand nodes based on the distance of the node from the start node, $d(s_0, s)$
- Informed search methods also use **some estimate** of the distance to the goal, $d(s, s_g)$
- What if we knew the exact distance to goal $d(s, s_g)$, then we would not need to search
  - *Then there is no need to search, we could just be greedy!*
  - In practice, we do not know that exactly and must make an “estimate”.

# What is a Heuristic?

- Informally, it is an *intuition* about "approximate cost to goal"
- Even if we do not know $d(s, s_g)$ exactly, we often have some ***intuition*** about this distance. This intuition is called a heuristic, **h(n).**
- Heuristic
  - h(n) = *estimated* cost of the *cheapest* path from the state at node n to a goal state.
  - Heuristics can be *arbitrary*, *non-negative*, *problem-specific* functions.
  - Constraint, *h(n) = 0* if n is a goal.

# Example Heuristic – Path Planning

- Consider a path along a road system
- What is a reasonable heuristic?
  - The straight-line Euclidean distance from one place to another
- Is it always, right?
  - Certainly not – actual paths are rarely straight!

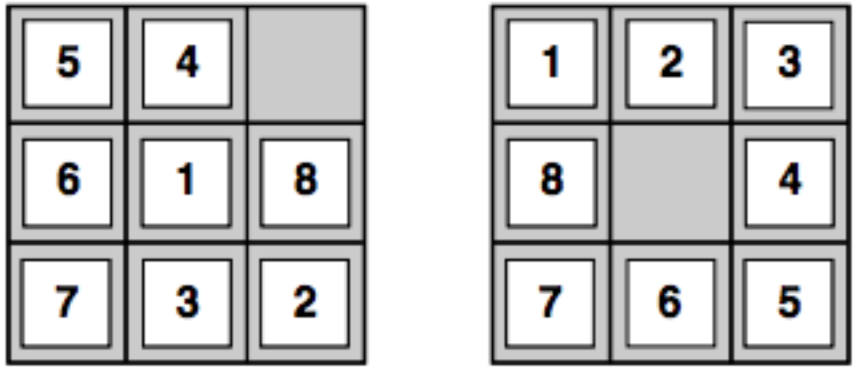



Straight−line distance to Bucharest

| | |
|---|---|
| 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(x)

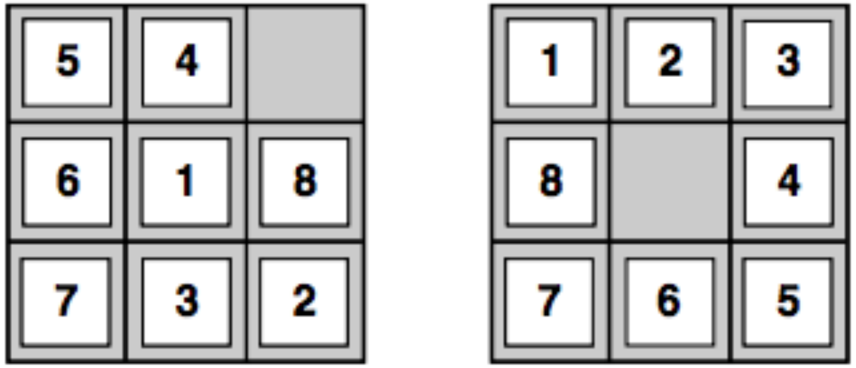
# Example Heuristic – 8 Puzzle



Start State Goal State

What would be good heuristics for this problem?

# Example Heuristic – 8 Puzzle

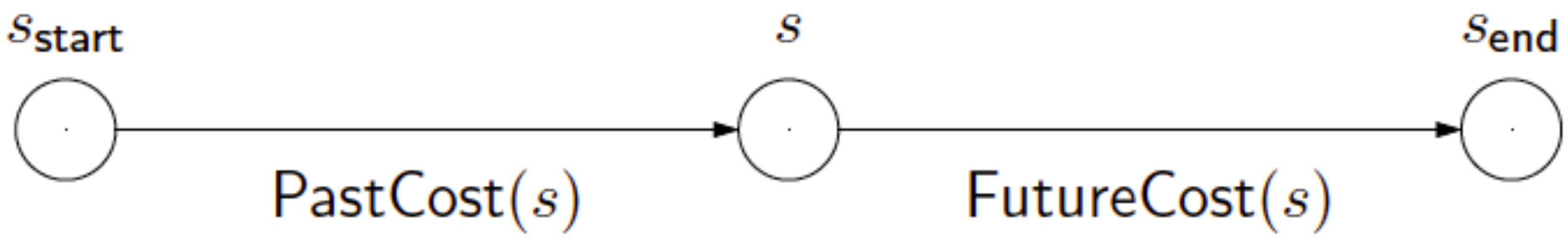


Start State Goal State

Consider the following heuristics:

- $h_1$ = number of misplaced tiles (=7 in example)
- $h_2$ = total Manhattan distance (i.e., no. of squares from desired location of each tile) (= 2+3+3+2+4+2+0+2 = 18 in example)

# *Greedy* Best-First Search

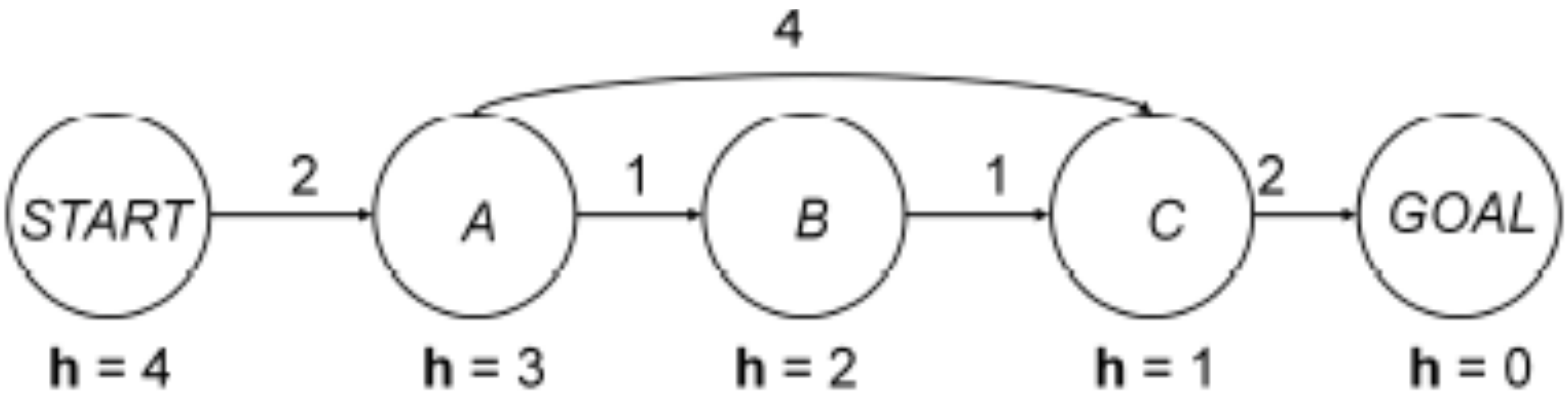



- **Best-First Search**
  - At any time, expand the most promising node on the frontier according to the evaluation function f(n).

- ***Greedy*** Best-First Search
  - Best-first search that uses h(n) as the evaluation function,
  - The evaluation function is, **f(n) = h(n)**, the estimated cost from a node n to the goal.
  - Only guided by “cost to go” (not “cost so far”).

# Greedy Best-First Search

- Which path does Greedy Best-First Search return?

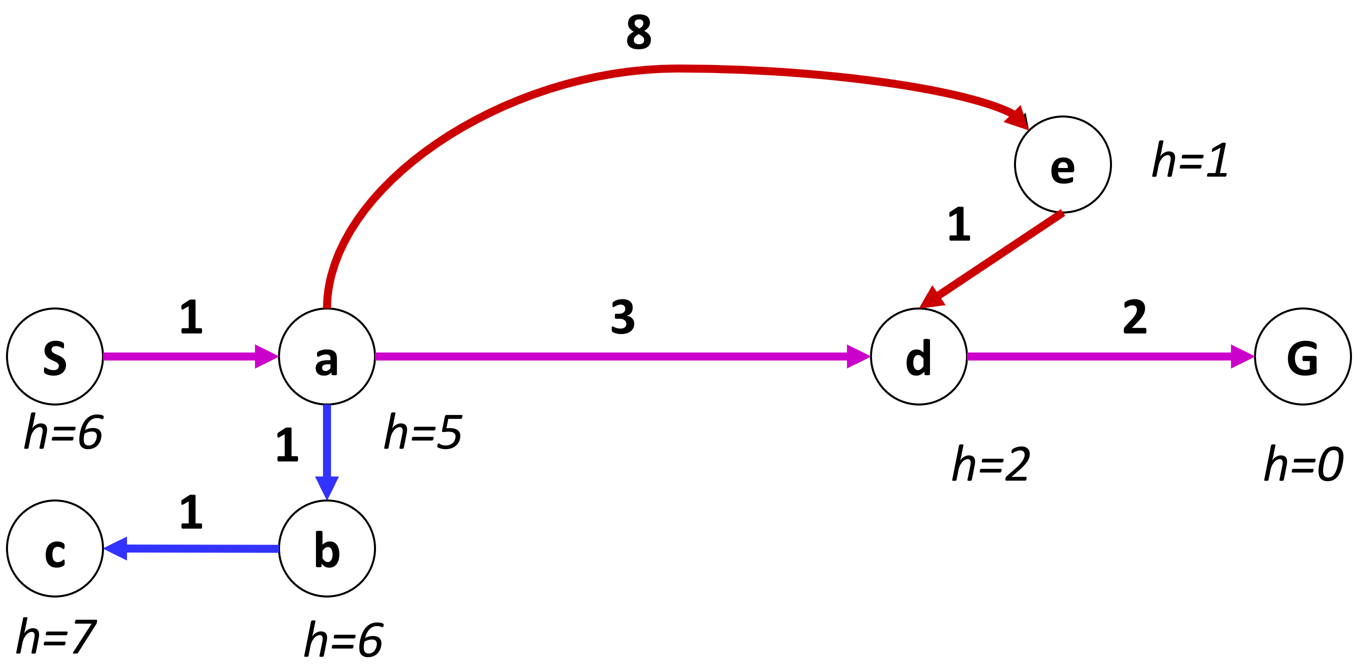


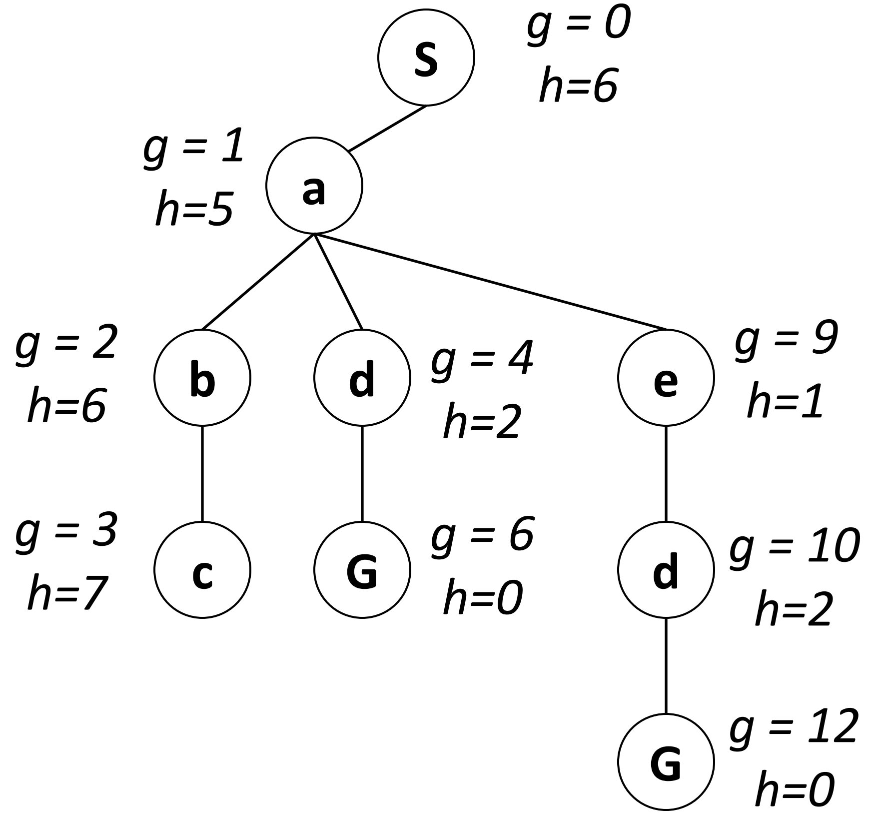
# A* Search

- Core Idea
  - Combine the greedy search (*the estimated cost to go*) with the uniform-search strategy (*the cost incurred so far*).
  - Minimize estimated path costs. Avoid expanding paths that are already expensive.
- Always expand node with lowest f(n) first, where
  - g(n) = **actual cost** from the initial state to n.
  - h(n) = **estimated cost** from n to the next goal.
  - **f(n) = g(n) + h(n)**, the estimated cost of the cheapest solution through n.
- Can I use any heuristic?
  - Any heuristic will *not* work. [properties soon]

# Example: UCS , Greedy and A* Search

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



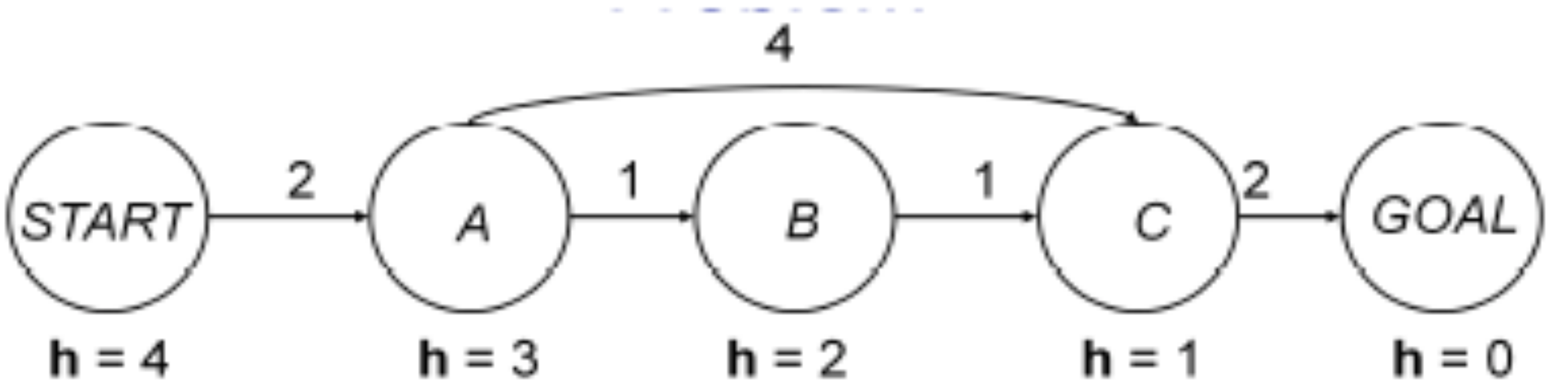




- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

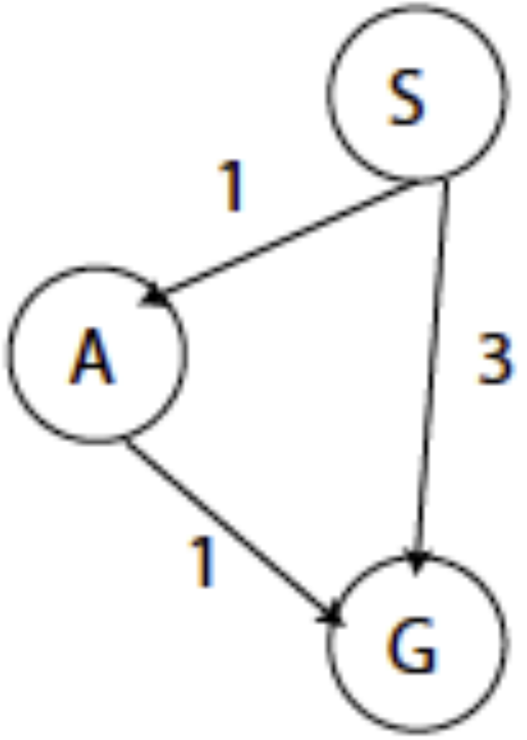
# Example

**Which path will A* search find?**



# Effect of heuristic function on search

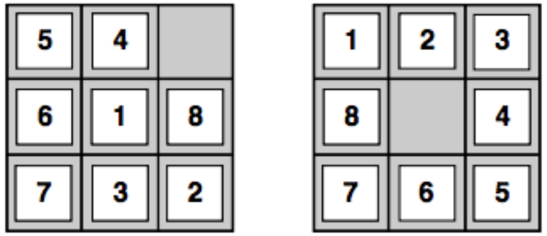
- For the following choices, would the optimal solution be found?
  - h(A) = 1
  - h(A) = 2
  - h(A) = 3
- Can we put conditions on the choice of heuristic to guarantee optimality?



# Admissible Heuristics

- Let **$h^*(n)$** be the actual **shortest path** from n to any goal state.
- Heuristic h is called ***admissible*** if **$h(n) \leq h^*(n)\ \forall n$**.
  - Admissible heuristics are ***optimistic*, **they often think that the cost to the goal is **less than the actual cost.**
- If h is admissible, then $h(g) = 0, \forall g \in G$
  - A **trivial** case of an admissible heuristic is $h(n) = 0, \forall n$.

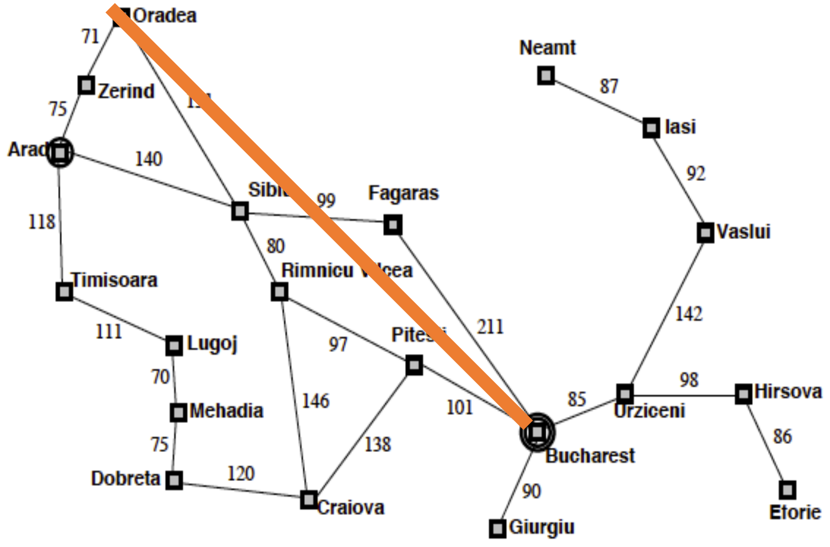
# Admissible or not?




Start State Goal State

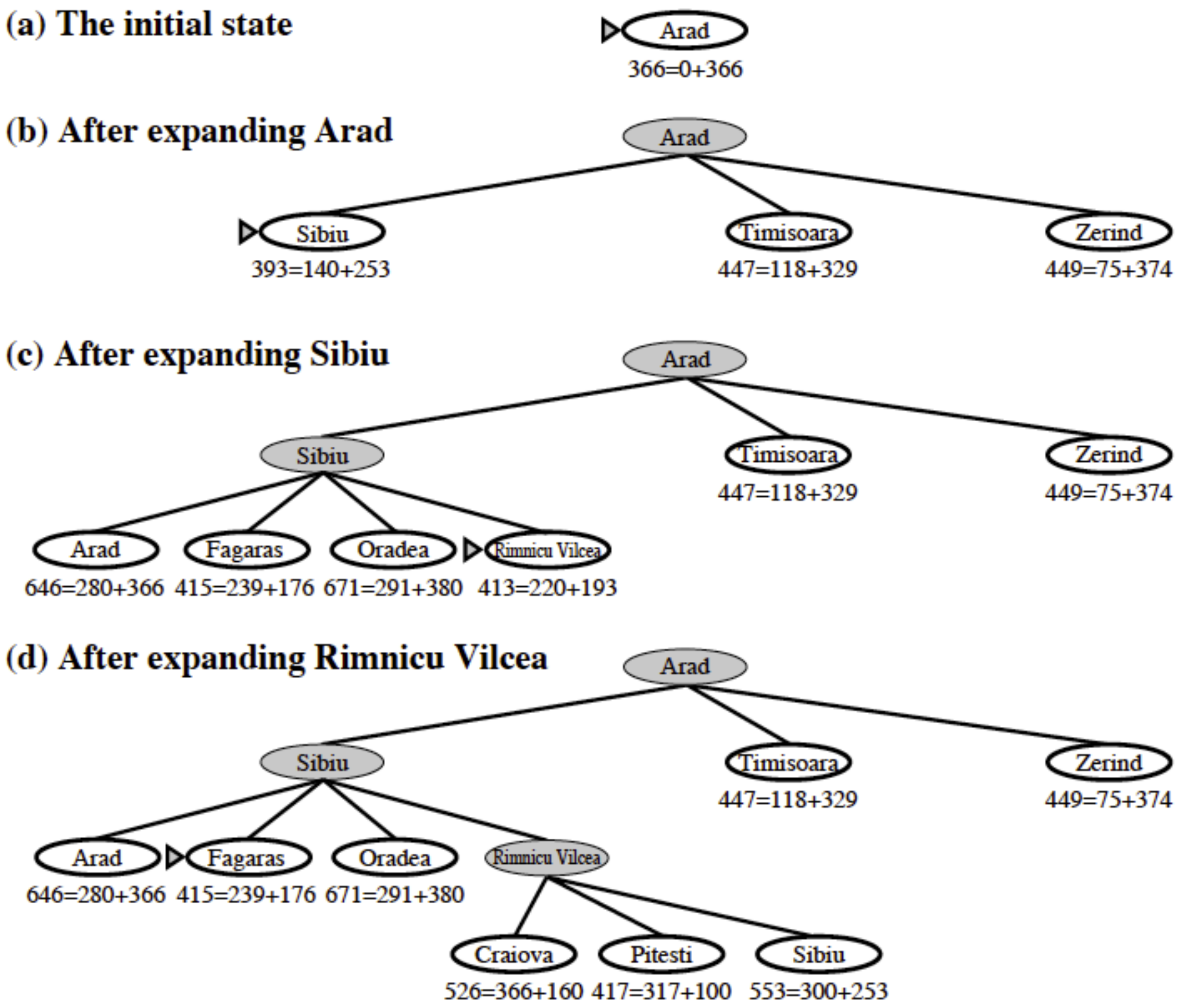
Consider the following heuristics:

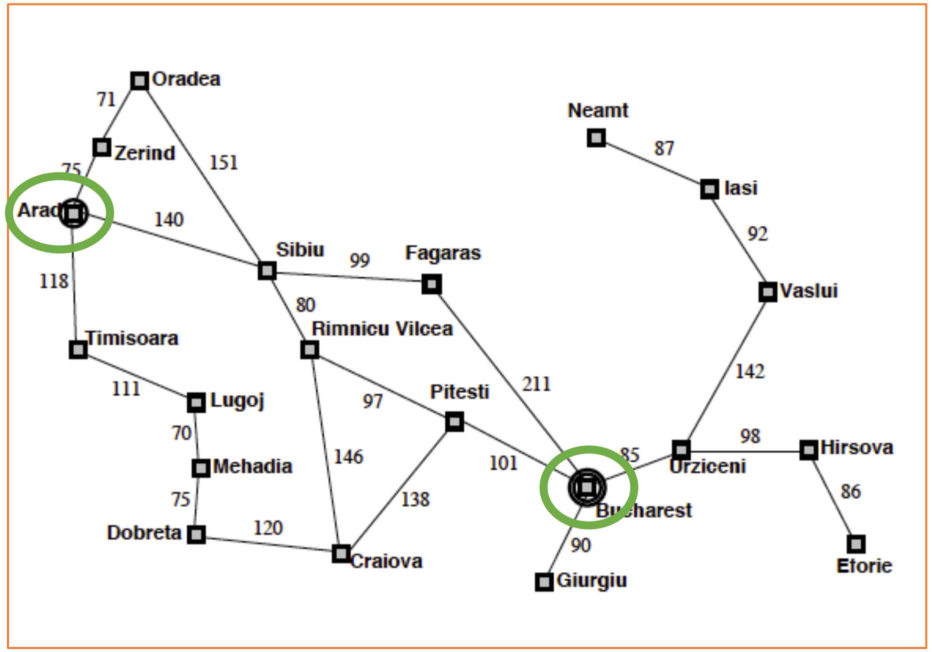
- $h_1$ = number of misplaced tiles (=7 in example)
- $h_2$ = total Manhattan distance (i.e., no. of squares from desired location of each tile) (= 2+3+3+2+4+2+0+2 = 18 in example)



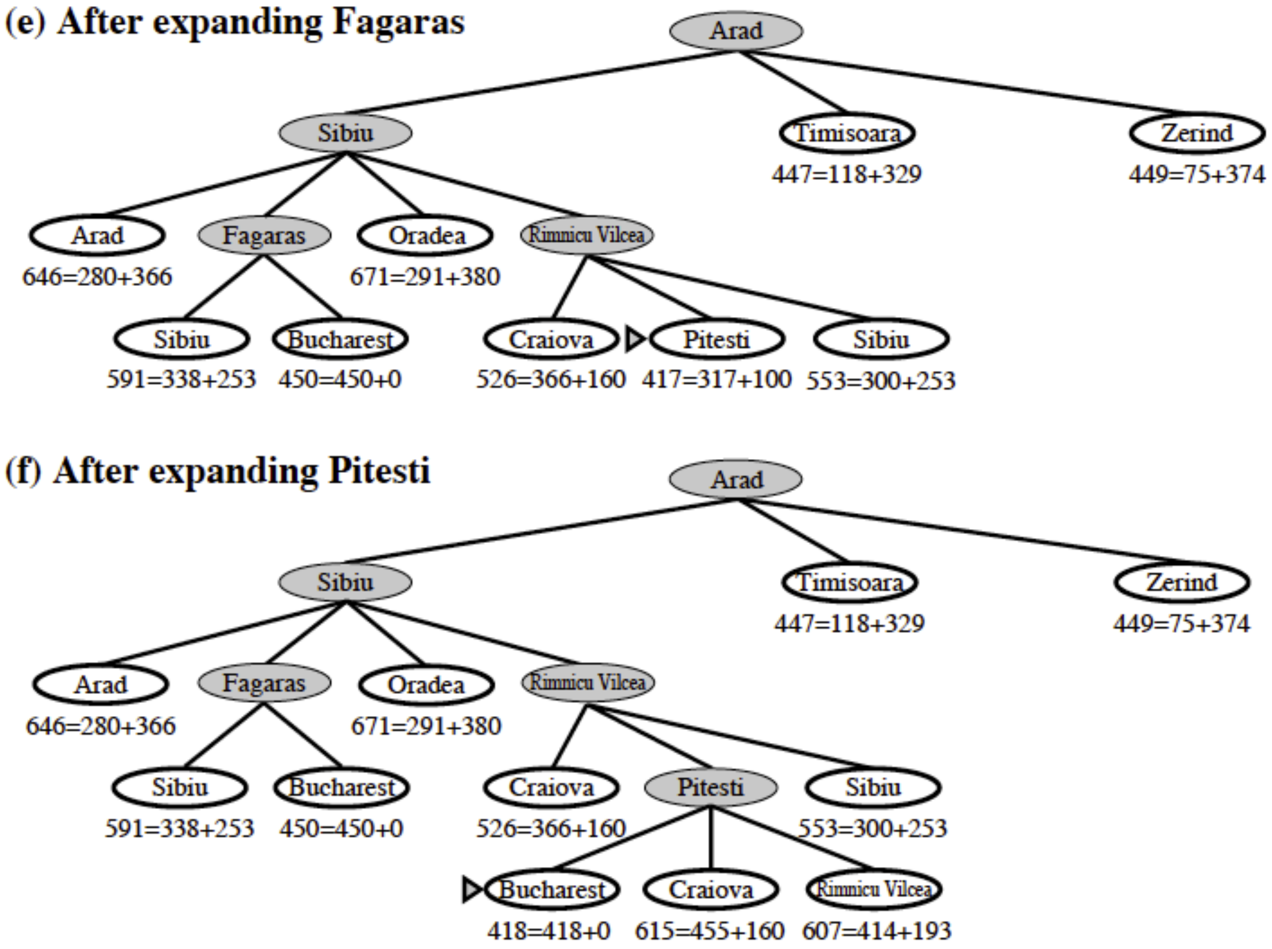

Straight line distance

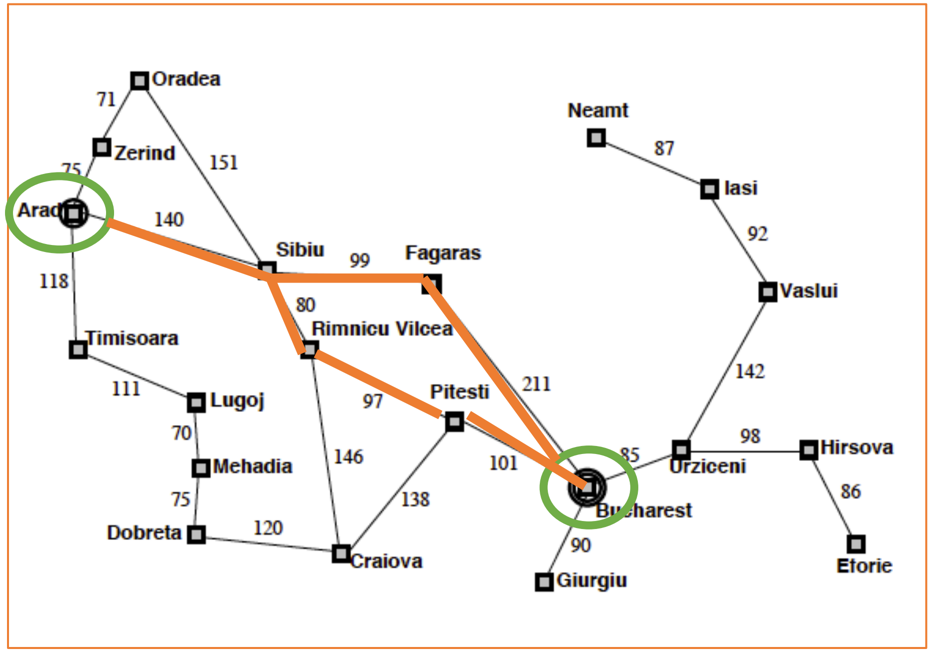
# A* Search: Route Finding Example



# A* Search: Route Finding Example



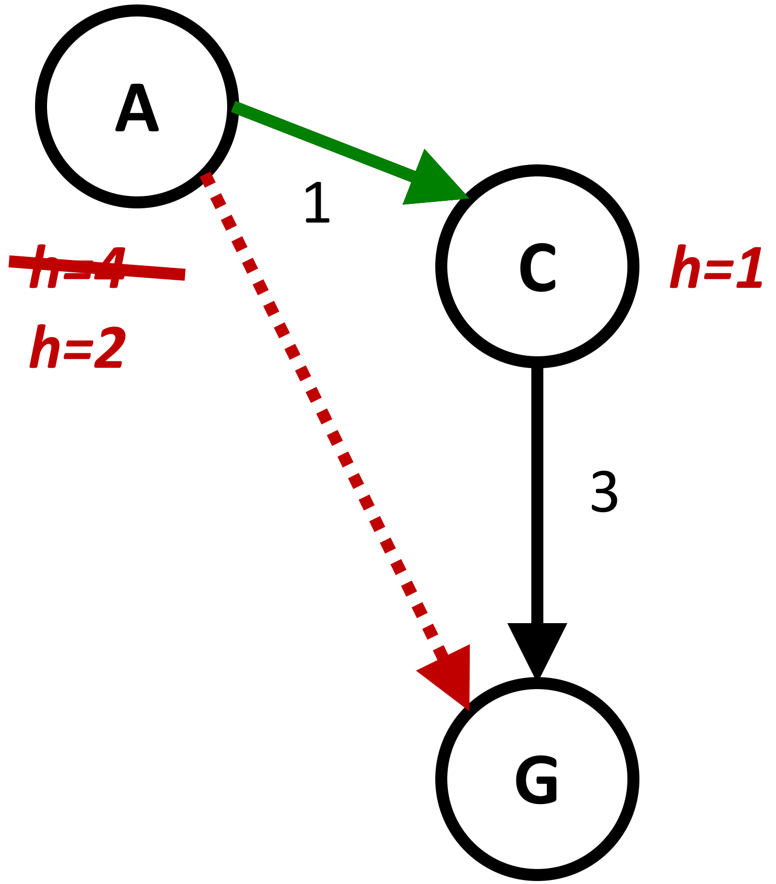




A* Tree Search will find the optimal path if the heuristic is admissible.

# Consistency (monotonicity)

- An admissible heuristic h is called consistent if for every state s and for every successor s’, **$h(s) \leq c(s, s') + h(s')$**
  - This is a version of triangle inequality
- Consistency is a **stricter requirement** than admissibility.
- If h is a consistent heuristic and all costs are non-zero, then f values **cannot decrease** along any path:
  - Claim $f(n') >= f(n)$, where n’ is the successor of n.
    - $g(n') = g(n) + c(n, a, n')$
    - $f(n') = g(n) + c(n, a, n') + h(n') >= f(n)$

# Admissibility and Consistency




- Main idea: estimated heuristic costs ≤ actual costs
  - **Admissibility:** heuristic cost ≤ actual cost to goal

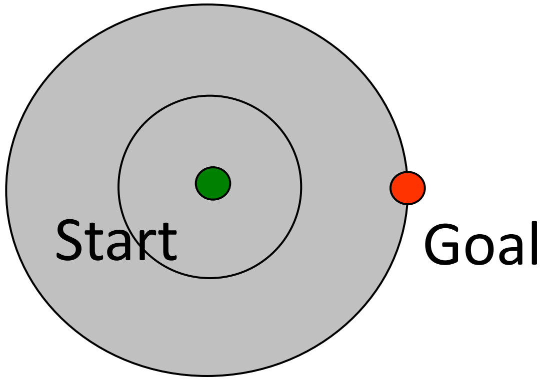
    h(A) ≤ actual cost from A to G

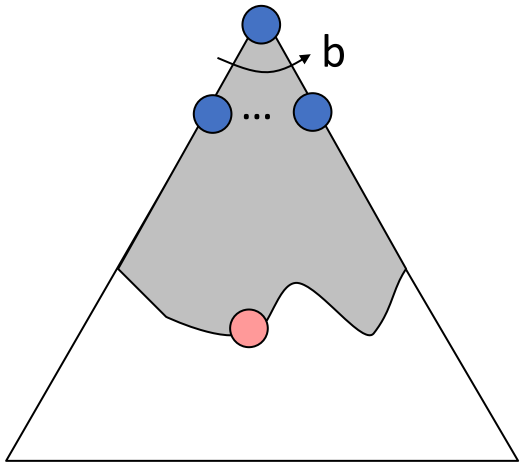
  - **Consistency:** heuristic “arc” cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

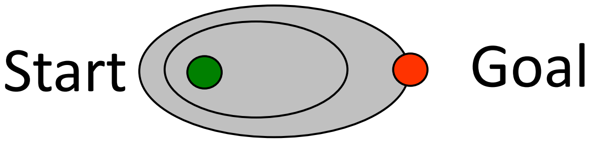
# Search Contours

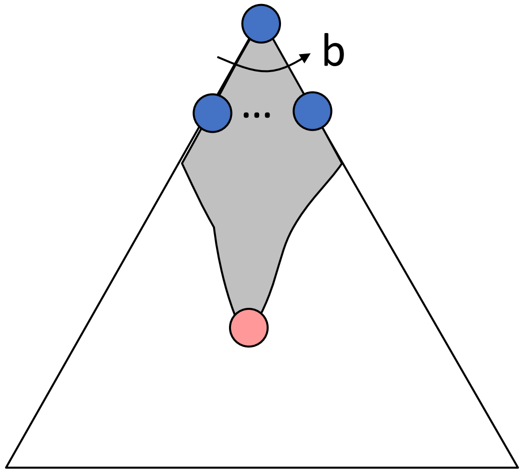
- UCS Search Contours




- A* Search Contours



# A* Search Properties

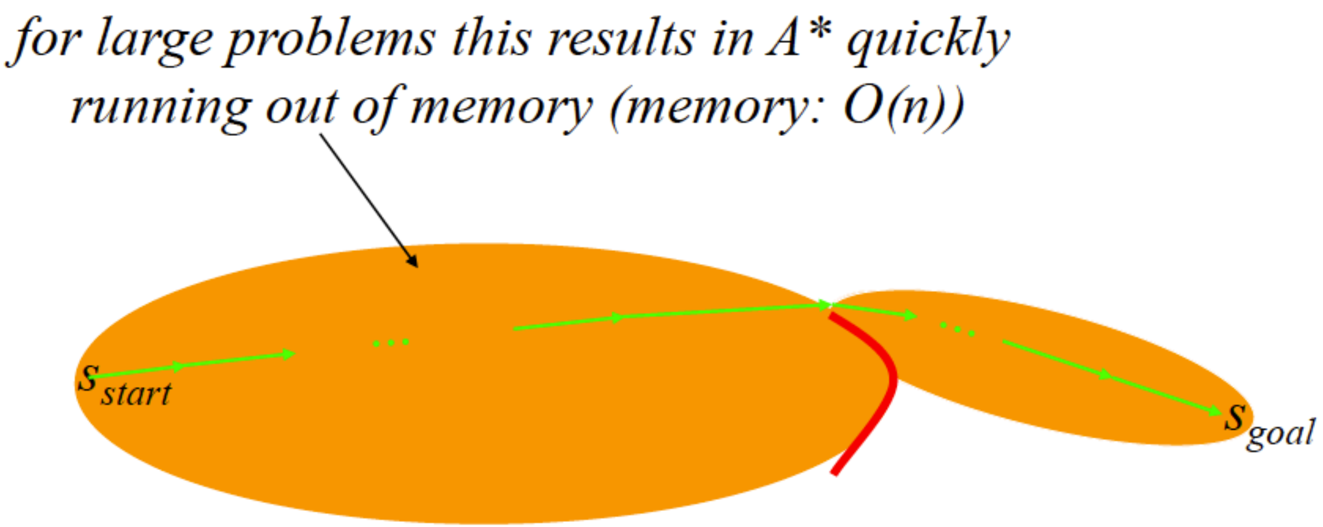
- **Optimality**
  - Tree search version of A* is optimal if the heuristic is admissible.
  - Graph search version of A* is optimal if the heuristic is consistent.

- **Completeness**
  - If a solution exists, A* will find it provided that:
    - every node has a finite number of successor nodes (b is finite).
    - there exists a positive constant $\delta > 0$ such that every step has at least cost $\delta$
    - Then there exists only a finite number of nodes with cost less than or equal to C*.

# A* Search Properties

- Exponential worst-case time and space complexity
  - Let $e = (h^* - h)/h^*$ (relative error)
  - Complexity $O(b^{ed})$ where $b^e$ is the effective branching factor.
  - 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
    - If an algorithm does not expand all nodes with $f(n) < C^*$ (the cost of the optimal solution) then there is a chance that it will miss the optimal solution.
- **Main Limitation: Space Requirement**
  - The number of states within the goal contour search space is still *exponential* in the length of the solution.

# A* Search may still take a long time to find the optimal solution



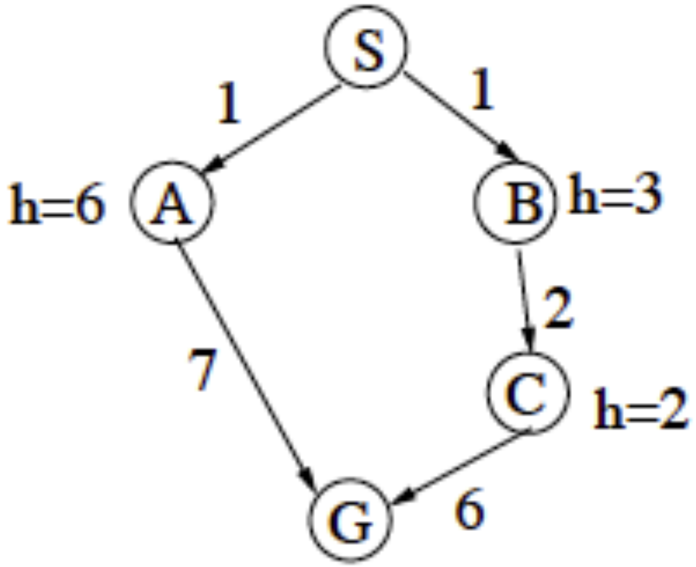

How to reduce memory requirement for A*?



Adapted from Maxim Likhachev

# Iterative Deepening A* (IDA*)

- **Idea**
  - Use an f-value limit, rather than a depth limit. Expand all nodes up to f1, f2, . . . . .
  - Keep track of the *next limit* to consider (so we will search at least one more node next time).
  - If the depth-bounded search fails, then the *next bound* is the *minimum* of the f-values that *exceeded the previous bound*.
- **Properties**
  - IDA$^*$ ]checks the same nodes as $A^*$ but recomputes them using a depth-first search instead of *storing* them.
  - IDA∗ has the *same properties* as A∗ but uses *less memory*.

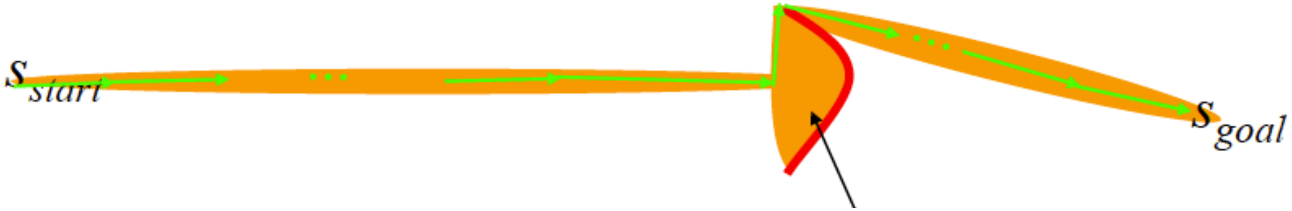



IDA* example

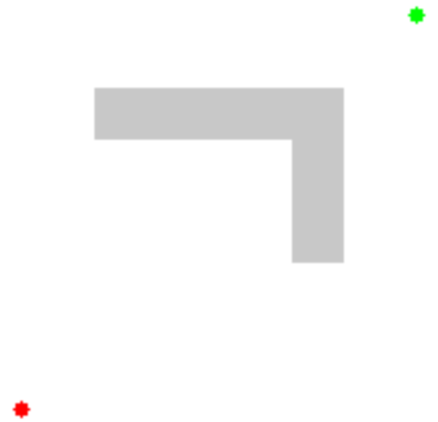
- If $f_1$ = 4, then which nodes are searched?
- If $f_2$ = 8, then which nodes are searched?

# Weighted A*

- Key Idea
  - Optimal solution requires large effort.
  - Can we quickly find sub-optimal solutions?
- Expand states in the order of
  - ***f'(n) = g(n) + w*h(n)*** values,
  - where ***w > 1.0***
  - Create a bias towards expansion of states that are closer to goal.
- Orders of magnitude faster than A*




A weighted heuristic accelerates the search by making nodes closer to the goal more attractive, giving a more depth first character.





Adapted from Maxim Likhachev

# Weighted A*

- f'(n) is *not admissible* but finds good *sub-optimal* solutions *quickly*.
- If h(n) is admissible then the sub-optimality is bounded.
  - Cost(solution) ) $\leq$ $\varepsilon\cdot$cost(optimal solution) where $\varepsilon = w - 1.0$.
- Trade off between search effort and solution quality.

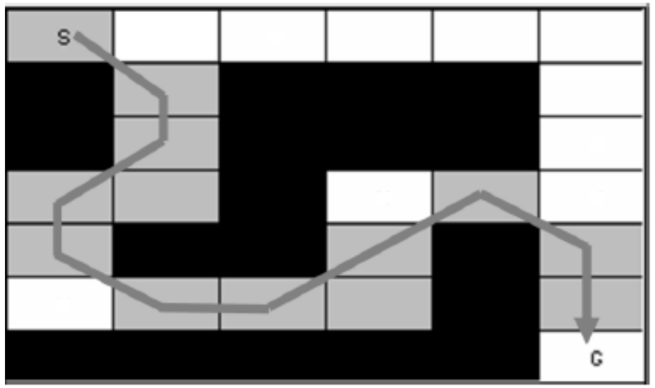


Adapted from Maxim Likhachev

# Anytime Search

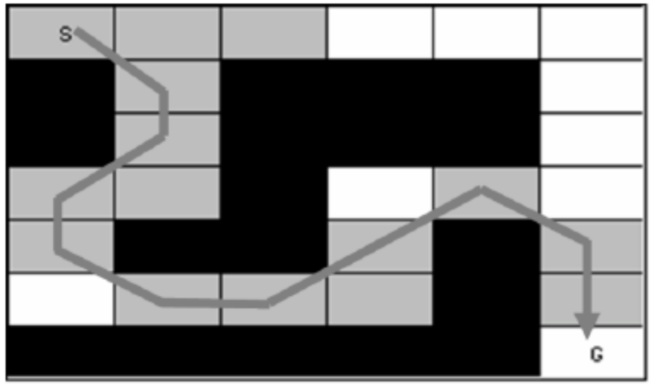
- Anytime search with weighted A*
  - find an approx. solution quickly; and then continue the search to find improved solutions and and improve the bounds on sub-optimality.
  - Run a series of weighted A* searches with decreasing *w.*

W = 2.5



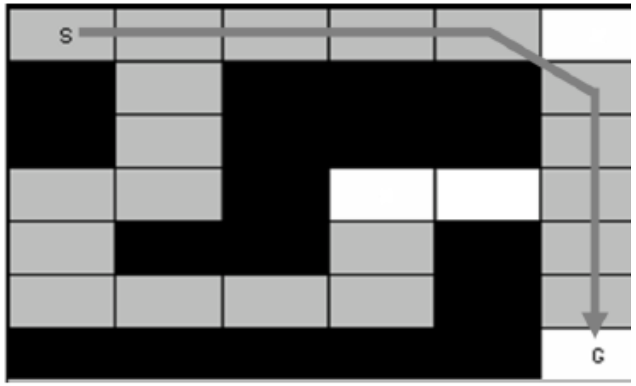

*13 expansions*
*solution=11 moves*

W = 1.5




*15 expansions*
*solution=11 moves*

W = 1.0




*20 expansions*
*solution=10 moves*



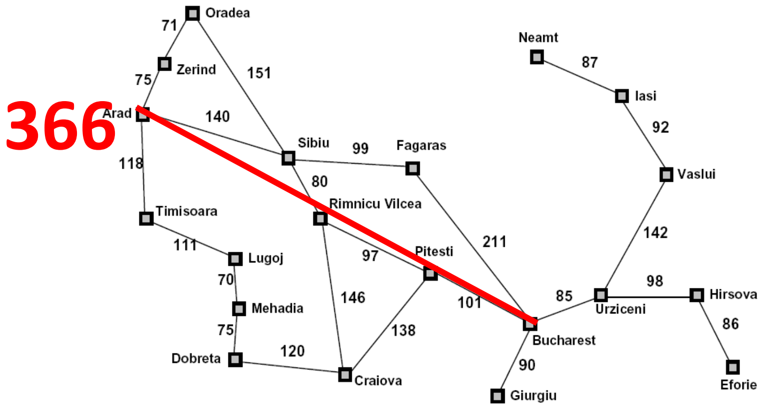
Adapted from Maxim Likhachev

# How are heuristics motivated?

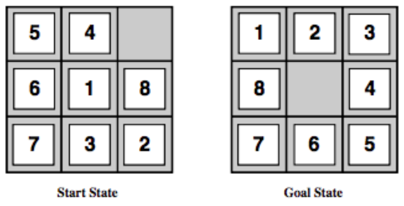
- Prior knowledge about the problem
- Exact solution cost of a relaxed version of the problem
  - E.g., If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1$ gives the shortest solution
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h_2$ gives the shortest solution
- From prior experience.

# Admissible Heuristics from Relaxed Problems

- Problem Relaxation
  - Ignore constraints/rules
  - Increase possibilities for actions.
- State space graph for the relaxed problem is a super-graph of the original state space
  - The removal of restrictions adds more edges.
  - Easier to find a solution.




Permitting straight line movement adds edges to the graph.




Consider the following heuristics:

- $h_1$ = number of misplaced tiles (=7 in example)
- $h_2$ = total Manhattan distance (i.e., no. of squares from desired location of each tile) (= 2+3+3+2+4+2+0+2 = 18 in example)

# Admissible Heuristics from Relaxed Problems

- Optimal solution in the original problem is also a solution for the relaxed problem.
- Cost of the optimal solution in the relaxed problem is an admissible heuristic in the original problem.
- Finding the optimal solution in the relaxed problem should be “easy”
  - Without performing search.
  - If decomposition is possible, it is easier to directly solve the problem.

# Effective branching factor

- Let A* generate *N* nodes to find a goal at depth *d*
- Let b* be the branching factor that a uniform tree of depth d would have in order to contain N+1 nodes.

$$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

$$N+1 = ((b^*)^{d+1} - 1)/(b^* - 1)$$

$$N \approx (b^*)^d \Rightarrow b^* \approx \sqrt[d]{N}$$

- Varies across problem instances, but nearly constant for hard problems.
- A measure of a heuristic's overall usefulness. A way to compare different heuristics.

# Comparing Heuristics

Effective branching factors for A* search for the 8-puzzle:

Comparison of two heuristics: Misplaced tiles ($h_1$) and Manhattan distance ($h_2$)

Heuristic ($h_2$) expands fewer nodes and has a lower effective branching factor

- $d$ = distance from goal
- Average over 100 instances

| | Search Cost (nodes generated) | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A*($h_1$) | A*($h_2$) | IDS | A*($h_1$) | A*($h_2$) |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | - | 539 | 113 | - | 1.44 | 1.23 |
| 16 | - | 1301 | 211 | - | 1.45 | 1.25 |
| 18 | - | 3056 | 363 | - | 1.46 | 1.26 |
| 20 | - | 7276 | 676 | - | 1.47 | 1.47 |
| 22 | - | 18094 | 1219 | - | 1.48 | 1.28 |
| 24 | - | 39135 | 1641 | - | 1.48 | 1.26 |



Reference: AIMA

# Dominance

- Heuristic function $h_2$ (strictly) dominates $h_1$ if
  - both are admissible and
  - for every node n, $h_2(n)$ is (strictly) greater than $h_1(n)$.
- A* search with a dominating heuristic function $h_2$ will never expand more nodes that A* with $h_1$.
- Domination leads to efficiency
  - Prefer heuristics with higher values, they lead to fewer expansions and more goal-directedness during search.

Typical search costs:

$d = 14$ IDS = 3,473,941 nodes
$\boxed{\begin{array}{l} \text{A}^*(h_1) = 539 \text{ nodes} \\ \text{A}^*(h_2) = 113 \text{ nodes} \end{array}}$

$d = 14$ IDS = too many nodes
$\boxed{\begin{array}{l} \text{A}^*(h_1) = 39{,}135 \text{ nodes} \\ \text{A}^*(h_2) = 1{,}641 \text{ nodes} \end{array}}$

# Combining admissible heuristics

- Heuristic design process
  - We may have a set of heuristics but not a single "clearly best" heuristic.
  - Have a set of heuristics for a problem and none of them dominates any of the other.

- Combining heuristics
  - Can use a composite heuristic
  - Max of admissible heuristics is admissible when the component heuristics are admissible.
  - The composite heuristic dominates the component heuristic.

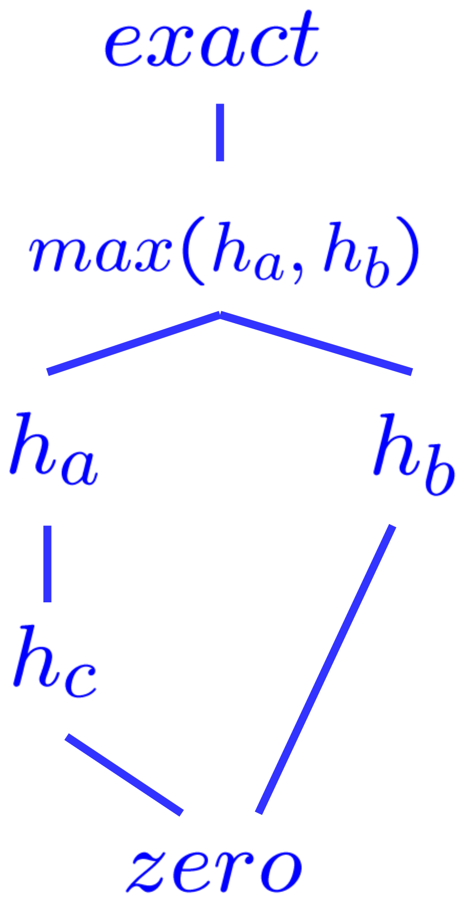
$$h(n) = max(h_a(n), h_b(n))$$



Slide adapted from Dan Klein

# Combining admissible heuristics

- Heuristics form a semi-lattice structure
  - Some heuristics can be compared to others via dominance.
  - There may be others not comparable.
  - Can create composites by combining component heuristics.
- Bottom of lattice is the zero heuristic
  - No or little computation effort
  - Not useful during search
- Top of lattice is the exact heuristic
  - A lot of computation effort
  - Really useful during search (give the exact cost)

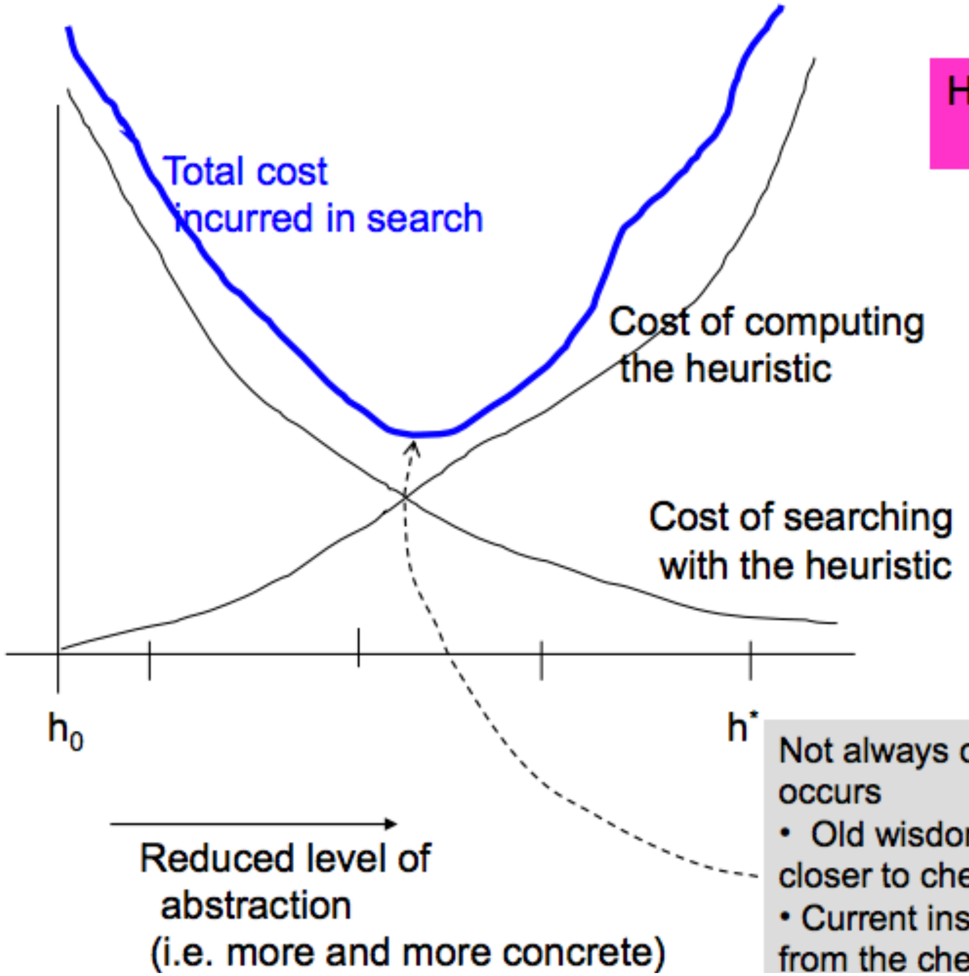





Slide adapted from Dan Klein

# Trade off

Effectiveness of the heuristic (reduced search time with the heuristic) vs. effort required to compute the heuristic




How informed should the heuristic be?

Not always clear where the total minimum occurs

- Old wisdom was that the global min was closer to cheaper heuristics
- Current insights are that it may well be far from the cheaper heuristics for many problems



Slide adapted from Mausam