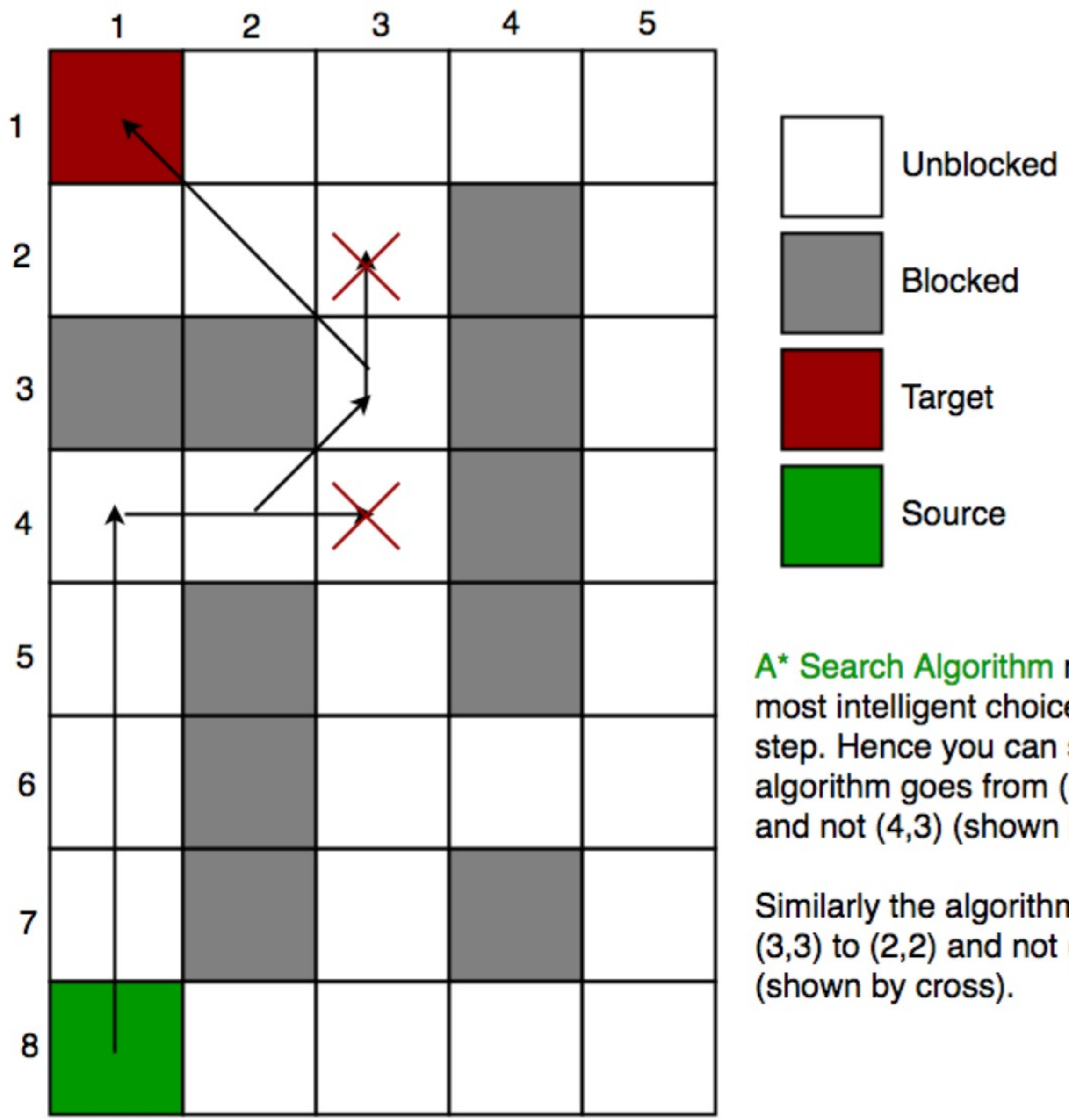
CSCI 3202: Intro to Artificial Intelligence
Lecture 9: A* Search and Optimality

Rhonda Hoenigman
Department of Computer Science



A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).

Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Source

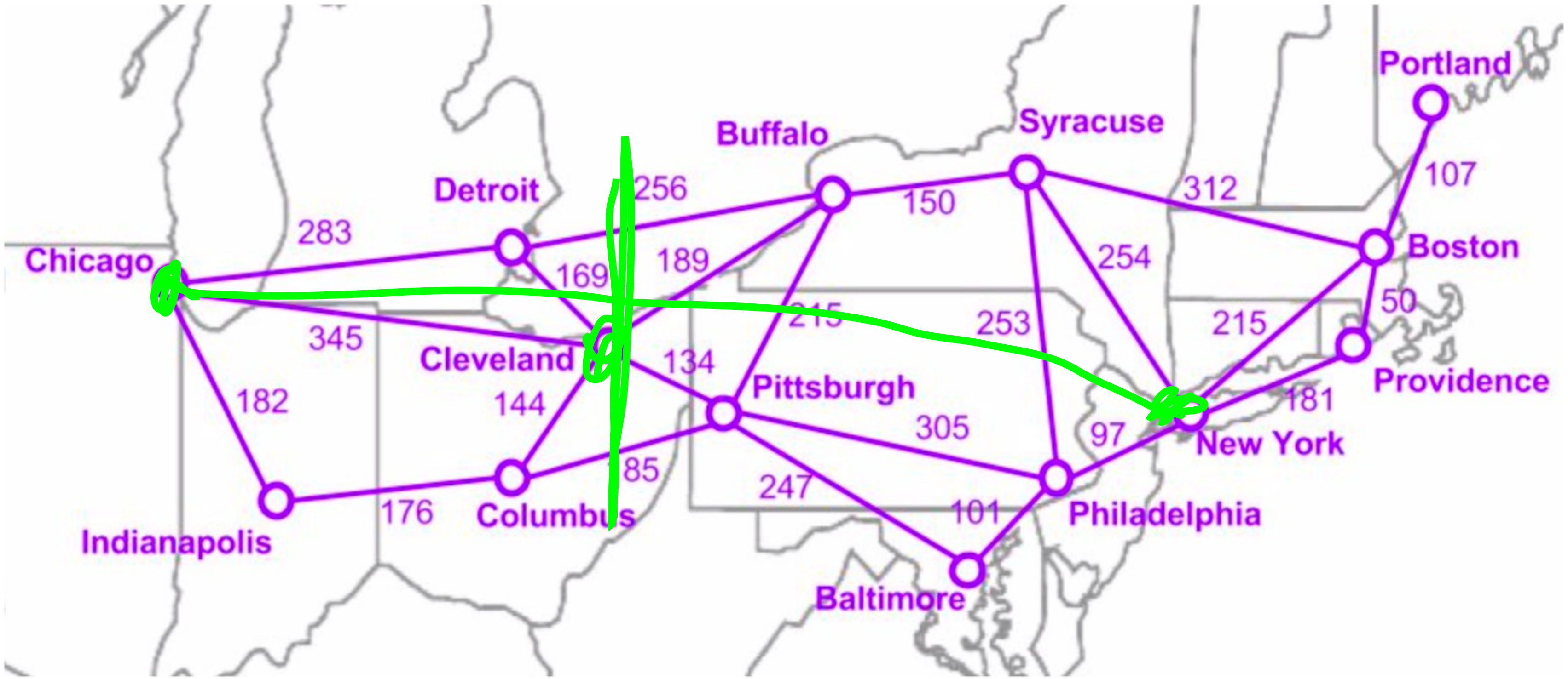
Review: Greedy best–first search

❖ First expand the path that's closest to the goal.

To determine what's closest to the goal, we need to define a heuristic function.

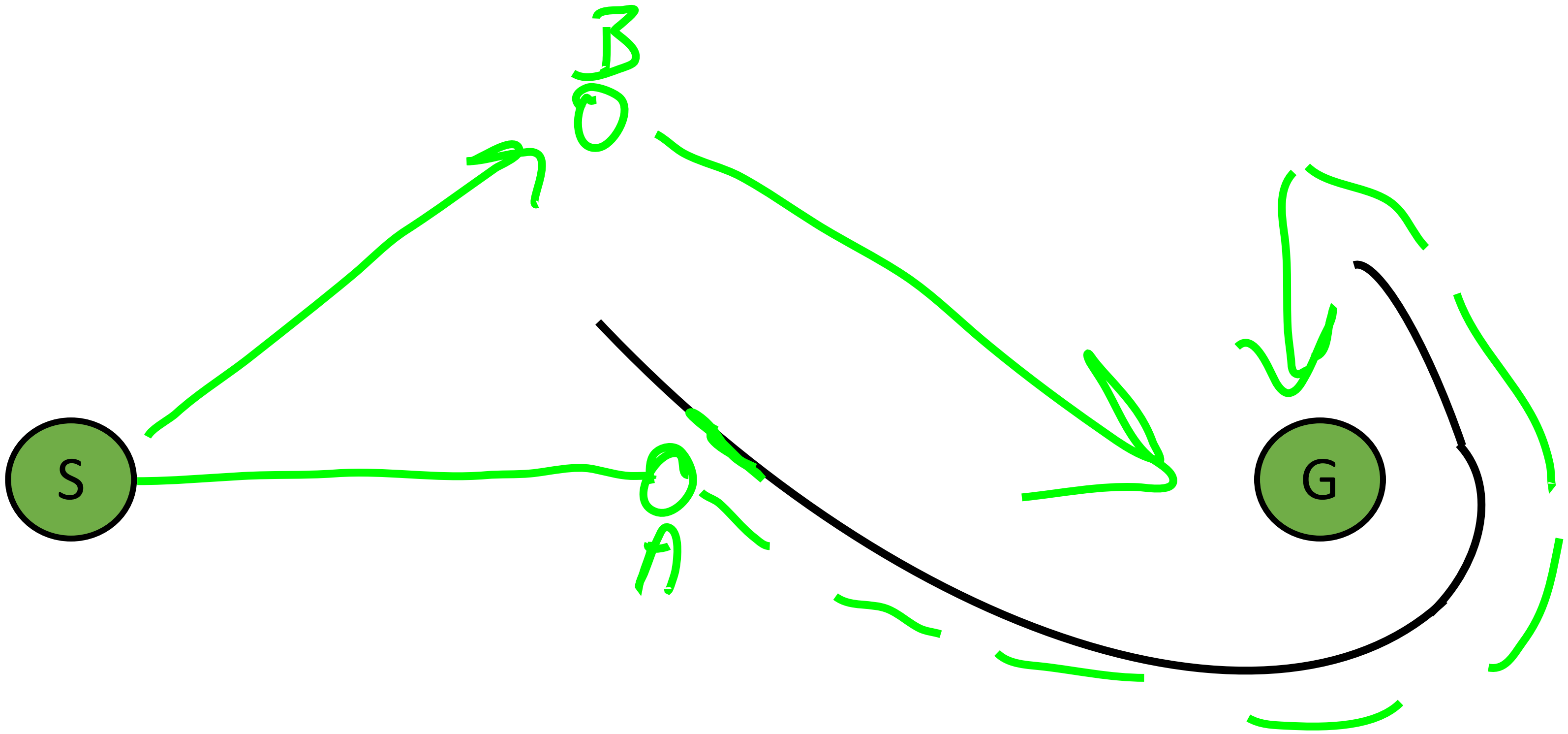
Example: For the traveling in the northeast problem, let's estimate the distance to the goal as the straight-line distance between city and the goal city.

Step costs: miles between cities along major highways



Greedy best_first search

Possible Issue: Won't necessarily find the optimal path. Can get stuck in local optimum.



A* Search

Uniform-cost search:

$f(n) = g(n)$ (cost to get to *n*)

Greedy:

$f(n) = h(n)$ (estimated cost to get from *n* to goal)

A*:

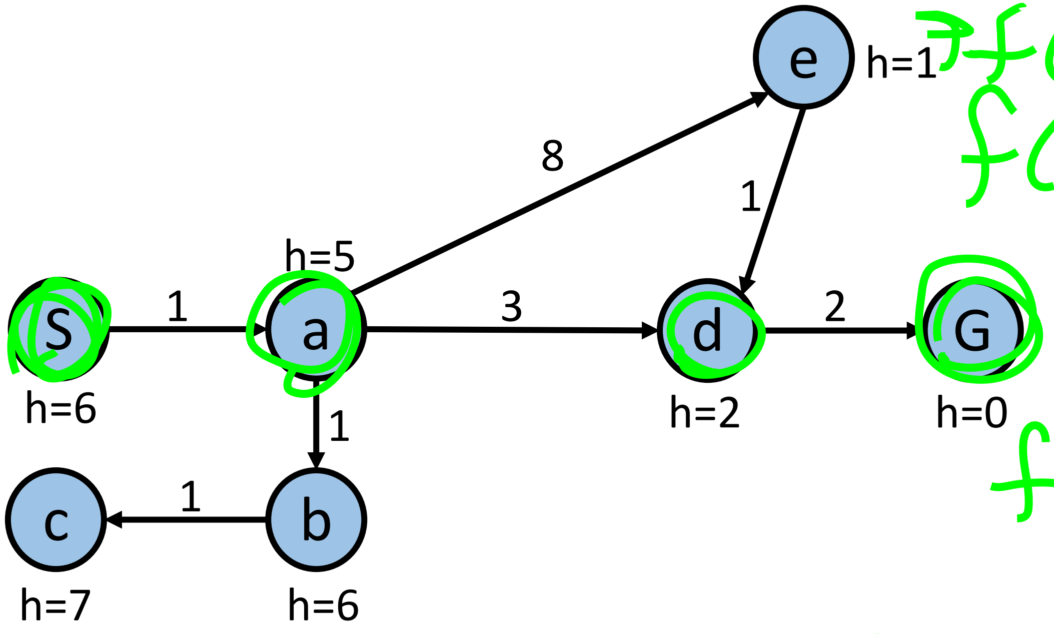
$f(n) = g(n) + h(n)$ (estimated total cost of cheapest solution through *n*)

A* Search

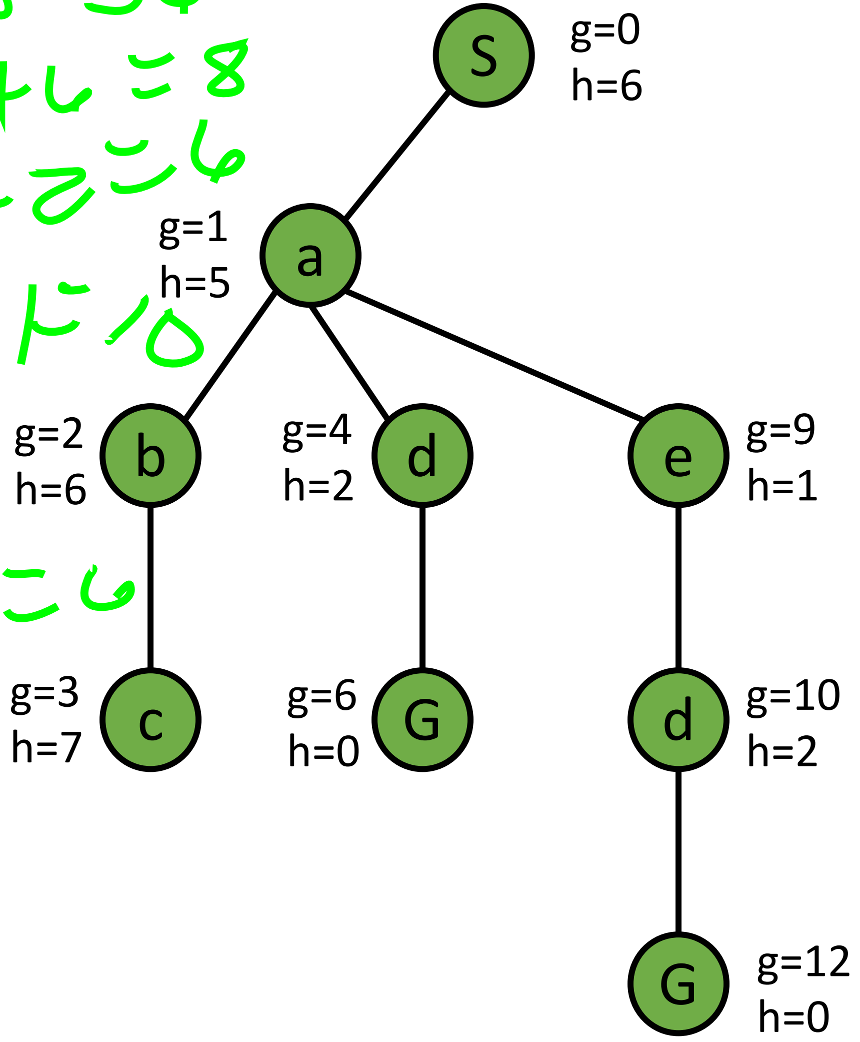
$f(n) = g(n) + h(n)$

Example: A* on the graph below.

$f(a) = 1+5 = 6$
$f(b) = 2+6 = 8$
$f(d) = 4+2 = 6$
$f(e) = 9+1 = 10$



$f(G)\ 4+2 = 6$



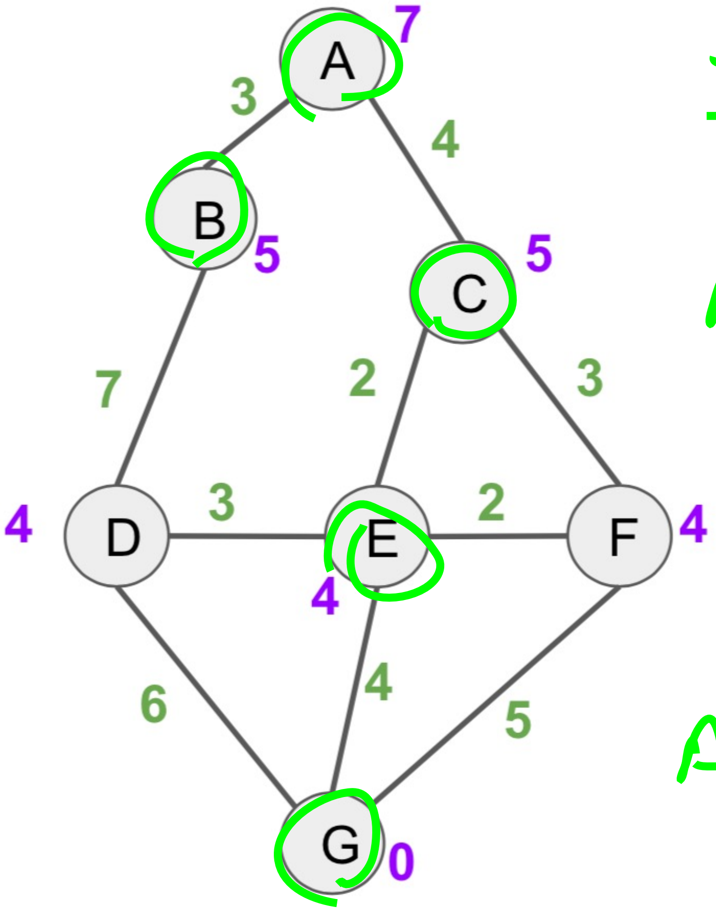
Exp
S, a
S, a, d, G

Frontier
b, ~~d~~, e

A* Search

A* Search:
- Find the cheapest path from A to G
- $h(n)$ values are given in **purple**
- Step costs are given in **green**

What are f(n) values for each node?



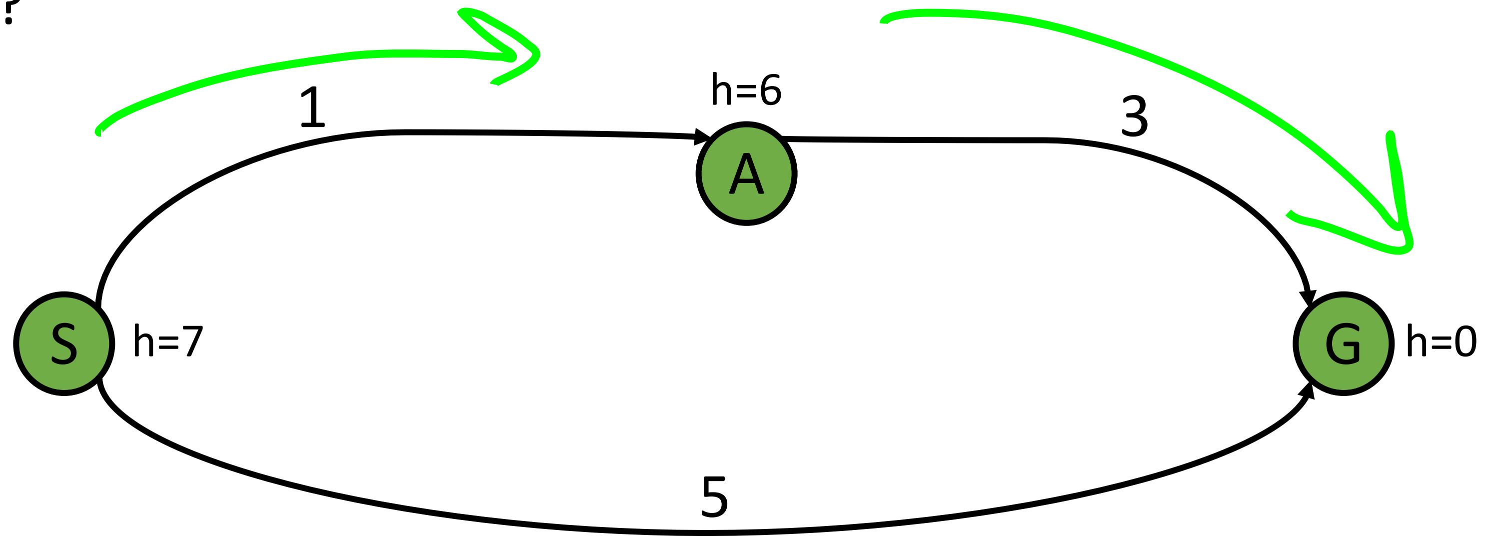
Exp	Frontier
A	(B, 8), (C, 9)
A, (B, 8)	(C, 9), (D, 14)
A, (B, 8), (C, 9)	(D, 14), (E, 10), (F, 11)
A, (B, 8), (C, 9), (E, 10)	(D, 13), (F, 11), (G, 10)
A, (B, 8), (C, 9), (E, 10), (G, 11)	

A* terminates when Goal added to explored set

A* Search

$S \to G$

Is A* optimal?



Exp	Frontier
S	(A, 7) (G, 5)
S, G	

A* Search – requirements for heuristic function

Consistent: for every node n and successor n’ of n, generated by some action a, the estimated cost of reaching the goal from n is no greater than the step cost from n to n’, plus the estimated cost of reaching the goal from n’

- That is: $h(n) \leq c(n, a, n') + h(n')$
- General **triangle inequality** between n, n’, and the goal

A heuristic h is **admissible** (optimistic) if $0 \leq h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to the nearest goal.

heuristic is an optimistic estimate of cost.

Optimality

Conditions for Optimality: Admissibility & Consistency

- $h(n)$ must be **admissible** - an admissible heuristic is one that never overestimates the cost to reach the goal.

- $h(n)$ is **consistent** if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n, a, n') + h(n')$$

Not consistent

A* Search

Search only works when:

- domain is fully observable
- domain must be known
- domain must be deterministic
- domain must be static

Relies on domain knowlege to design h(n)

implementation: use a **node**

- state - indicates state at end of path
- action - action taken to get here
- cost - total cost
- parent - pointer to another node

Total path to get to state

Edges in graph w/ weight and action

- Sum of edge weights and actions to node.

- where did we immediately come from

Optimality of A* Search

A* is **optimally efficient** for any given heuristic: No other optimal algorithm is guaranteed to expand fewer nodes than A*

- Recall: A* expands all nodes with $f(n) < C^*$, where C* is the cost of the optimal solution path.

- Any algorithm that does not expand all nodes with $f(n) < C^*$ risks missing a better solution path.

$f(n) > C^*$

Paths where $f(n) < C^*$ need to be explored for optimality

Optimality of A* Search

A* (graph) is optimal if the heuristic $h(n)$ is consistent.

Based on two key facts:

1. If $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing.

2. Whenever A* selects a node *n* for expansion, the optimal path to that node has been found.

Same as UCS and Dijkstras alg. where adding node to explored means optimal path found to that node.

Optimality of A* Search

Whenever A* selects a node n for expansion, the optimal path to that node has been found.

<u>Example</u>: Prove the above statement.

f(n) is non-decreasing

Proof by contradiction

<u>Proof</u>: suppose the statement isn't true, that we are expanding n, but a lower cost solution exists on the frontier. But, since f(n) is non-decreasing, and the lowest cost node is always selected for expansion, there won't be a lower cost solution left on the

Optimality of A* Search

frontier. Thus, a node selected must be the lowest cost path to the node.

A* (graph) is optimal if the heuristic $h(n)$ is consistent.

Based on two key facts:

1. If $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing.

2. Whenever A* selects a node *n* for expansion, the optimal path to that node has been found.

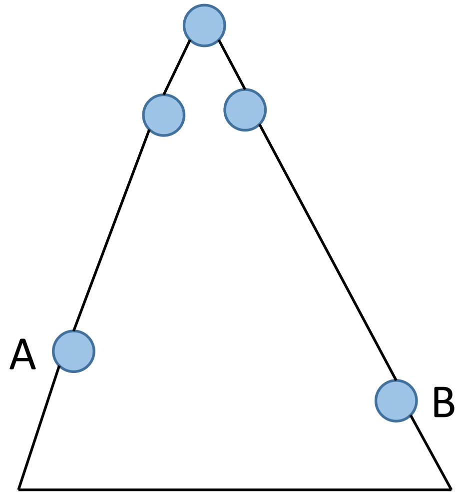
➢ **So the first goal node to be expanded took the lowest-cost path, and all later goal node expansions are at least as expensive.**

Optimality of A* Search

Assume:

- A is an optimal Goal Node
- B is a suboptimal Goal node
- h is admissible

Claim: A will exit the frontier before B.



Optimality of A* Search

So A* is **optimal**, **complete**, and **optimally efficient**.

Why do we even care about other search algorithms?

- **Number of nodes** to expand along the goal contour is still **exponential** in depth of solution/length of solution path.

- Absolute error: $\Delta := h^* - h$
  - h* = actual cost from root to goal
  - h = heuristic you used

- Relative error: $\epsilon := (h^* - h)/h^*$

$h^* - h$

Minimize Δ by selecting an h that is close to h^*

$$\frac{(h^* - h)}{h^*}$$

A* Search

b = branching factor

Complexity depends strongly on state space characterization

- Single goal, tree, reversible actions → $O\left(b^{\Delta}\right)$, or $O\left(b^{\epsilon d}\right)$ with constant step costs (*d* is solution depth)

  Δ typically is proportional to the path cost h^*, so ϵ is pretty much constant (or growing with d), and we can rewrite: $O\left((b^{\epsilon})^{d}\right)$

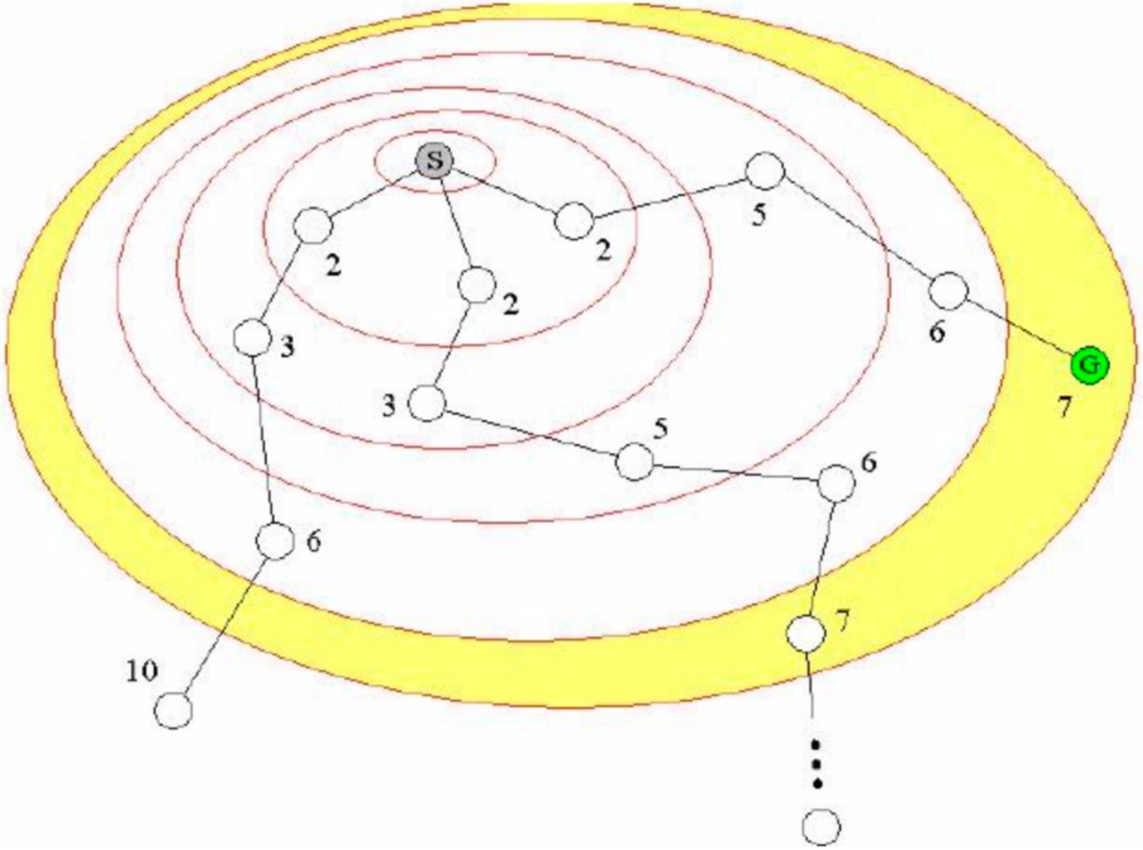
  → The effective branching factor is really b^{ϵ}.
  → Important to choose as good of a heuristic as we can.

- Many goal states/near-goal states can be a problem -- need to expand a ***lot*** of branches.

A* Search: alternatives

Space complexity can be a burden for A*

- Keeps all generated nodes in memory



- Iterative Deepening A* (IDA*)
    - Just like standard IDS, but instead of extending the search depth, extend the allowed *f* cost.
    - Search out to a particular contour

A* Search: alternatives

- **Recursive best-first search (RBFS)**
  - Like standard DFS, but keeps track of the best alternative path's *f*-cost
  - Once the path you're digging down exceeds the best alternative path, switch over to the back-up path
  - As RBFS back-tracks, each node along the back-tracked path it replaces the *f*-value with that of the cheapest child node.
    - Remembers the best leaf in the sub-tree, so RBFS knows whether it's worth it to go back down that road later.
  - Still has problems of indecision
    - Expanding a new path makes the unexpanded alternatives look better.

Recursive best–first Search（RBFS）

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
   return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

Figure 3.26 The algorithm for recursive best-first search.

Recursive best_first Search（RBFS）

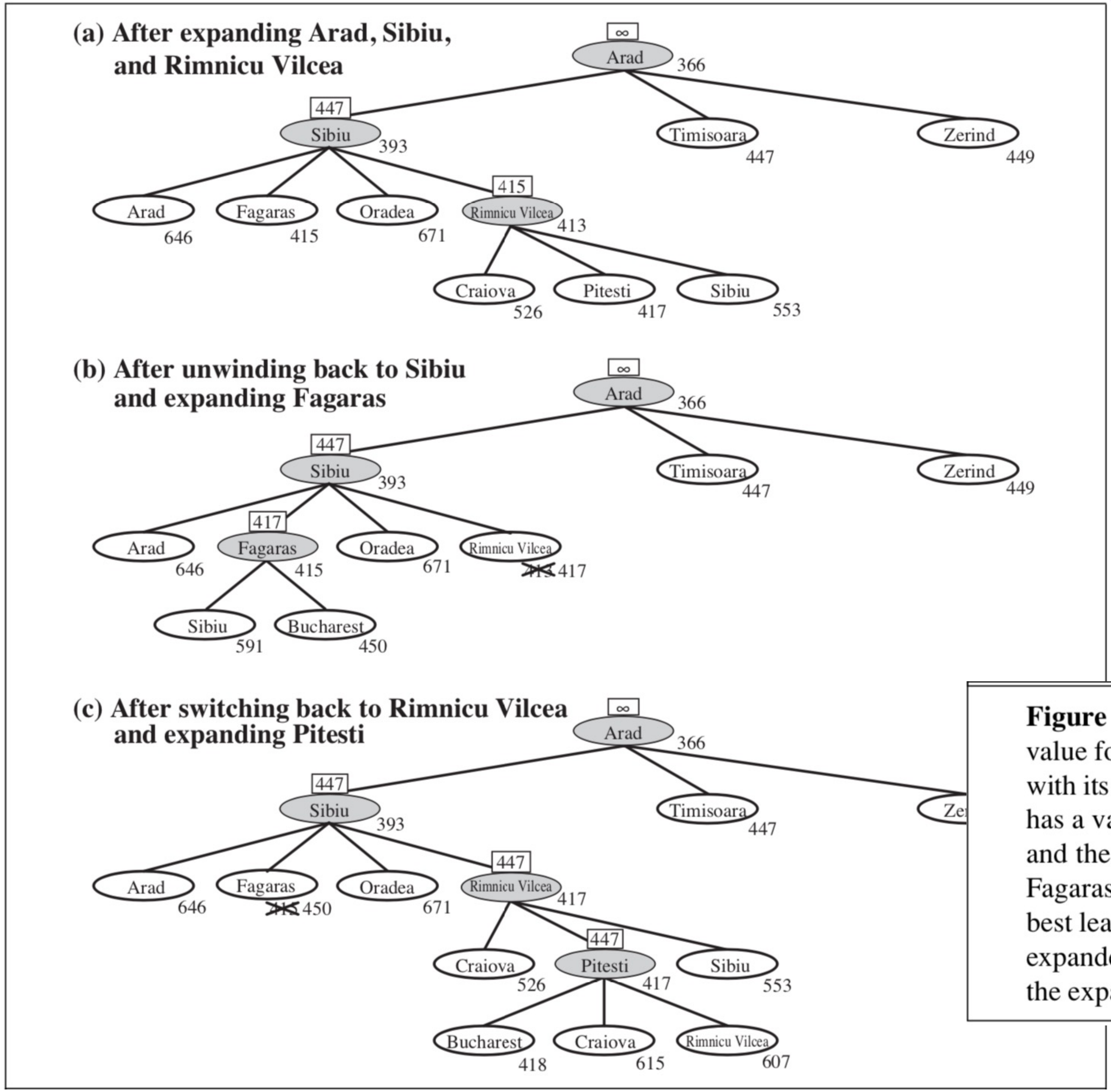


Figure 3.27 Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras). (b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best leaf value of 450. (c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path (through Timisoara) costs at least 447, the expansion continues to Bucharest.

A* Search: alternatives

- IDA* and RBFS use very little memory

    - For example:
        - Between iterations, IDA* keeps only the current *f*-cost limit
        - RBFS has memory benefits of DFS, but at the cost of potential time inefficiency

    - Those who don’t remember the past, are doomed to repeat it (maybe).

A* Search: alternatives

Memory-bounded A* and Simplified Memory-bounded A* (MA* and SMA*)

- SMA* -- general notes and subtleties
  - Expands the best leaf until memory is full
  - Then expands the best leaf and deletes the worst
  - What if all the leaves have the same *f*-value?
  - Then expand the newest leaf and delete the oldest
  - What if there is only one leaf?
  - That’s ok -- then there is a single solution path from root to goal, and it’s taking up all available memory, so we’ve failed anyway

Next Time

Heuristics