CSCI 3202: Intro to Artificial Intelligence

Lecture 8: Optimality

Rachel Cox

Department of Computer Science

Review

Uniform-cost search:

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

↖ cost we've already incurred to get from start state to n.

• looking backwards only

Greedy:

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

↖ future cost (estimated) from node n to goal.

• looking forwards

A*:

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

(estimated total cost of cheapest solution through *n*)

looking at cost already incurred + estimated future cost

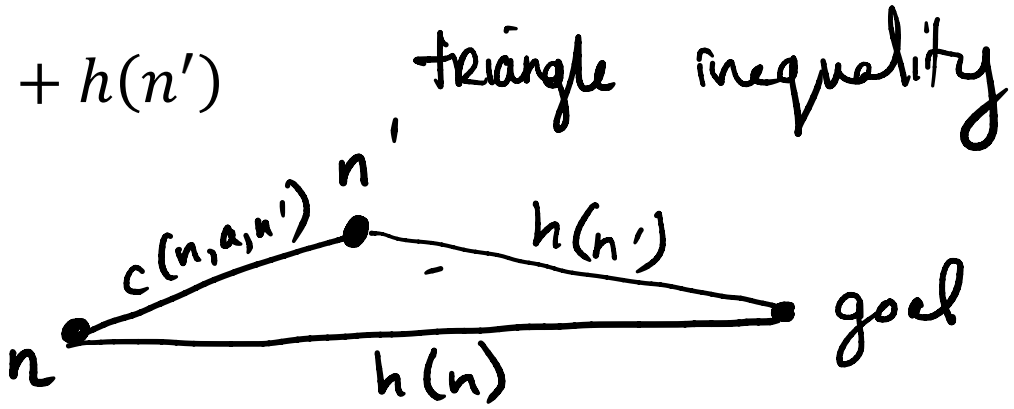
Optimality

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':

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



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.

g+h

"A*"
We won't expand a node where $f(n) > C^*$
→ "pruning"

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.

Optimality of A* Search

$p \Rightarrow q$

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

Example: Prove the above statement.

g - path cost
h - heuristic - estimated cost to goal.

Proof: Assume $h(n)$ is consistent.

Let n' be a successor of n.

$$g(n') = g(n) + c(n,a,n')$$

Add $h(n')$ to each side.

$$h(n') + g(n') = g(n) + c(n,a,n') + h(n')$$

$$f(n') = g(n) + c(n,a,n') + h(n') \quad \text{by definition of f-cost}$$

$$\geq g(n) + h(n) \quad \text{by consistent property}$$

$$= f(n) \quad \text{by definition of f-cost.}$$

$\therefore f(n') \geq f(n) \Rightarrow f$ is nondecreasing. ∎

Optimality of A* Search

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

Example: Prove the above statement.

Proof: (by contradiction) Suppose the above statement isn't true

⇒ We are expanding n, but there is some lower-cost path.

⇒ This lower cost path must end with a frontier node.
Call this frontier node of the lower-cost path n'

f is a non-decreasing function ⇒ $f(n') \leq f(n)$

⇒ n' should have been expanded first

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.

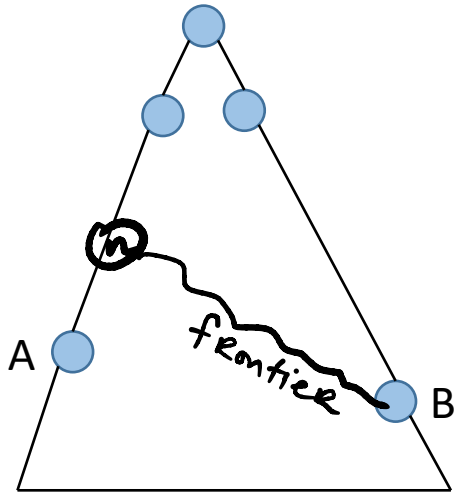
- **So the first goal node to be expanded took the lowest-cost path, and all later goal node expansions are at least as expensive.**

Optimality of A* Search

Assume:

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

$\Big)\ f(A) < f(B)$



Claim: A will exit the frontier before B.

Proof: Suppose B is on the FRONTIER.
Suppose some ancestor of A is also on the frontier.
Call this ancestor n.

$f(n) = g(n) + h(n)$ by definition of f-cost

$f(n)$ is the estimated total cost to the goal. And $g(A)$ represents the actual cost to A

$f(n) \leq g(A)$

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

$f(n) \leq f(A)$

$\Rightarrow f(A) \leq f(B)$ By our assumption

$\Rightarrow f(n) \leq f(B)$ $\Rightarrow$ n will be expanded first
$\Rightarrow$ A will be expanded 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: $\epsilon := (h^* - h)/h^*$

A* Search

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.

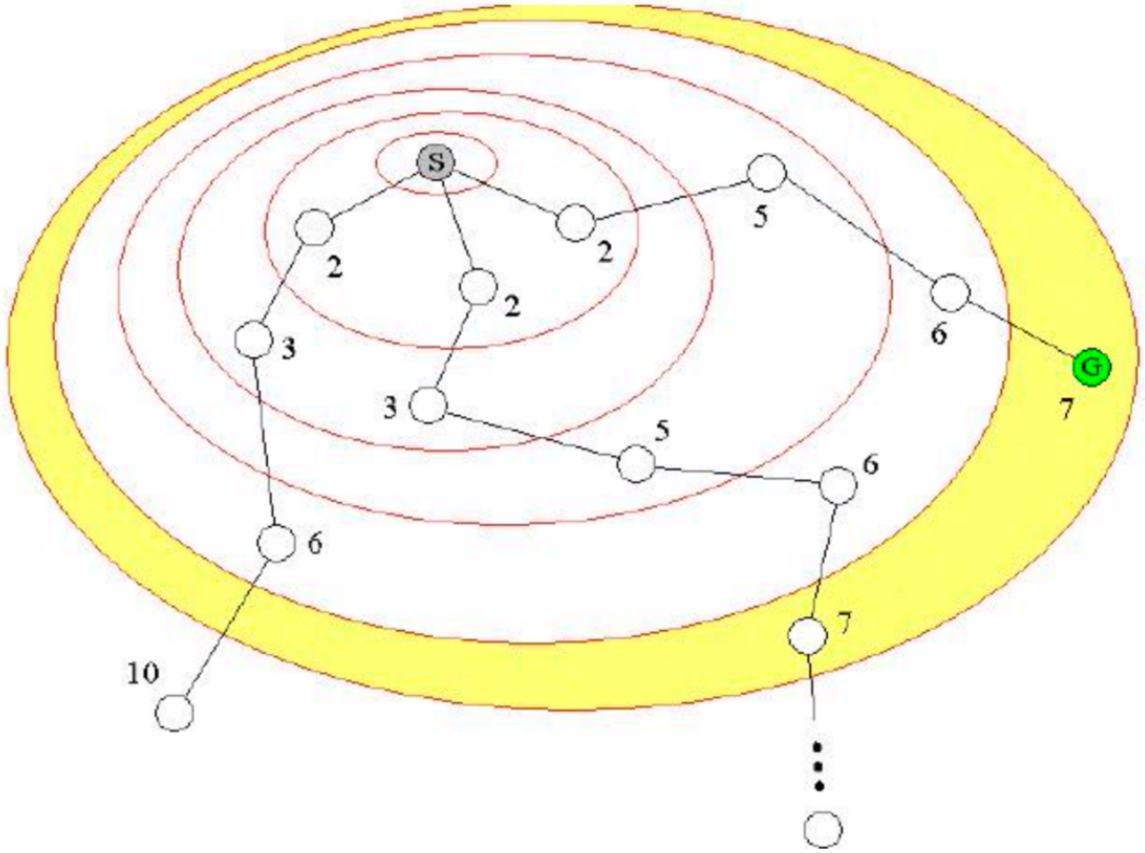
- Many goal states/near-goal states can be a problem -- need to expand a ***lot*** of branches.

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)

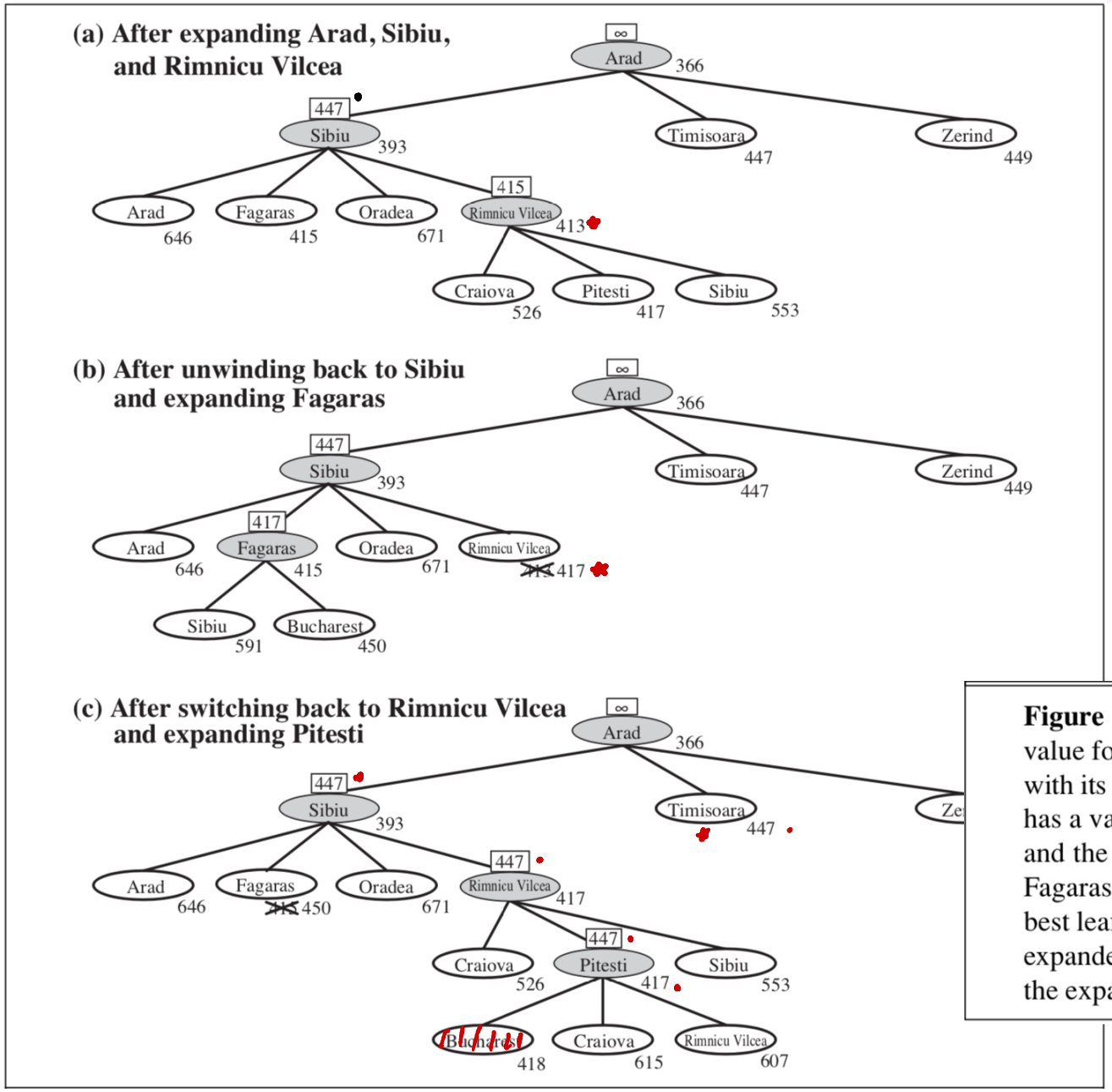


Figure 3.27 Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse 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