Sept 16: Optimality

Theorem: Any consistent heuristic is also admissible.

Prove it.

Announcements and To-Dos

Announcements:

1. HW 2 posted on Canvas!

Last time we learned:

1. A* and Greedy Best-First searches that use *heuristics*

$A*$ Recap:

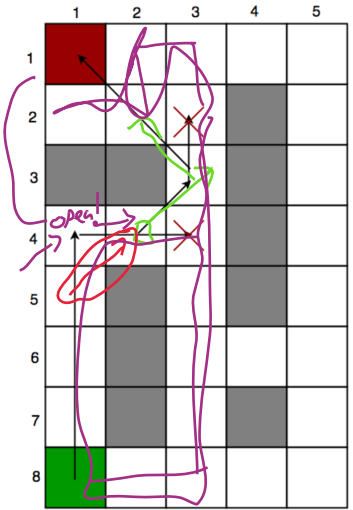
↳ BFS: queue
DFS

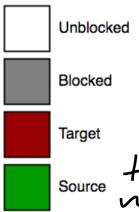
All of our search algorithms rely on a decision function f: choose the "next state" to expand with the best (lowest) evaluation of f. Supposing the cost to move from current state to state n is g,

1. UCS unfolded states in a closest-first order, or: $f(n) = g(n)$.
2. Greedy best-first algorithms used $f(n) = h(n)$ based on heuristic h.
3. A^* then uses:

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

A*:





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).

...but by what heuristic?

Heuristics Recap:

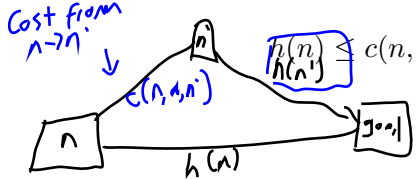
admissible

A^* is optimal on a tree or graph if h is **consistent.**

Definition: a heuristic $h(n)$ is *admissible* if it never overestimates the cost to reach the goal. Because $g(n)$ is an actual cost to reach n along a current path, f will never overestimate the cost of a solution *through* n if $g(n)$ is admissible.

Definition: a heuristic $h(n)$ is *consistent* if it satisfies the triangle inequality. In a sentence: the cost to go from n to the goal must be less than or equal to the cost to move from n to n' plus the *estimated* cost $h(n')$,

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



Opening Sol'n

from triangle: not an overestimate

Theorem: Any consistent heuristic is also admissible.

Proof: is by induction.

1 Base Case: this works for small example.
(if we are 1 step away from goal, then consistent ⇒ admissible

2 Inductive Hyp: Suppose consistent ⇒ admissible for up to k steps away. Then it also does for k+1.

Opening Sol'n

(consistent: $h(n) \leq h(n') + c(n \to n')$)

Theorem: Any consistent heuristic is also admissible.
Proof: is by induction.

Base Case: Suppose a heuristic h is consistent for $k = 1$ nodes along the shortest path from n to the goal node (so we start one step away from the finish)

Let n' be the goal, 1 step away. Then we have $h(n) \leq c(n, a, n') + h(n')$. Since n' is the goal, $h(n') = 0$ and $c(n, a, n') = h^*(n)$

(h dist-to-goal $= 0$, h^* optimal (true shortest distance))

Then:

$$\implies h(n) \leq c(n, a, n') + h(n') = h^*(n)$$

which means that h does not overestimate the actual cost to get to the goal from n, so it is admissible.

Opening Sol'n

Theorem: Any consistent heuristic is also admissible.
Proof: is by induction.

Inductive Step: Suppose *if* a heuristic h is consistent for k nodes along the shortest path from n to the goal node, then it is admissible.

Let n' be another node on the shortest path from n to the goal, k steps away. Then we have $h(n) \leq c(n, a, n') + h(n')$.

Since h is admissible along that path, $h(n') \leq h^*(n')$. Then:

$$\implies h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n') = h^*(n)$$

which means that h does not overestimate the actual cost to get to the goal from n, so it is admissible.

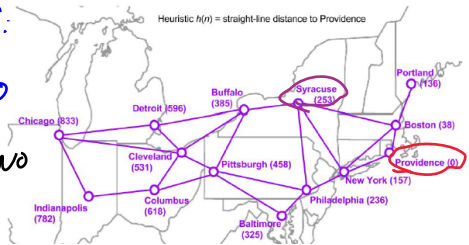
Optimality and A^*

Last time we said that A^* expands all nodes with $f(n) < C^*$ for optimal solution cost C^*.

Let's verify:

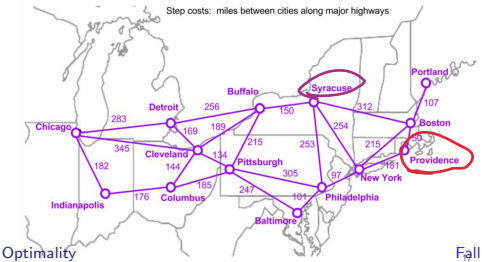
4 candidates:

	g	h_2	h_1	f:
Buff:	150	0	385	= NO
Phi:	253	0	236	= NO
NY:	234	0	177	= 411 NO
Boston:	312	0	38	= 350



What gets expanded on e.g. a route from Syracuse to Providence?

only Boston if Syracuse



Optimally Efficient

This leads to the claim that A^* is **optimally efficient** for any given heuristic: there is no other optimal algorithm that is guaranteed to expand fewer nodes than A^*.

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

Our goal is to argue that A^* is optimal as long as the heuristic is consistent. This relies on 2 major 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 Ideas

h approximates $\leq$ $h^*(n)$

$h^*(n)$: the soln cost through (n)

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

Proof:

Suppose that n' is a successor of n. **Goal:** show that $f(n') \geq f(n)$.

S'pose h consistent, n' is successor of n:

$h(n) \leq c(n \to n') + h(n')$ — Consistent (1)

$g(n') = g(n) + c(n, n')$ — Path (2)

$f(n') = g(n') + h(n')$ (3)

A^*: $= g(n) + c(n \to n') + h(n') \geq g(n) + h(n)$ (2 & 3)

$f(n') \geq g(n) + h(n) = f(n)$ (1 & 4)

Optimality Ideas

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

Proof:
Suppose that n' is a successor of n. **Goal:** show that $f(n') \geq f(n)$. Then:

$$
\begin{aligned}
g(n') &= g(n) + c(n, a, n') && \text{for some action } a \\
f(n') &= g(n') + h(n') \\
&= g(n) + c(n, a, n') + h(n') \\
&\geq g(n) + h(n) = f(n) && \text{because } h \text{ is consistent}
\end{aligned}
$$

$|a| + |b| \geq |a+b|$

Optimality Ideas

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

Proof:
Goal: contradiction.

Optimality Ideas

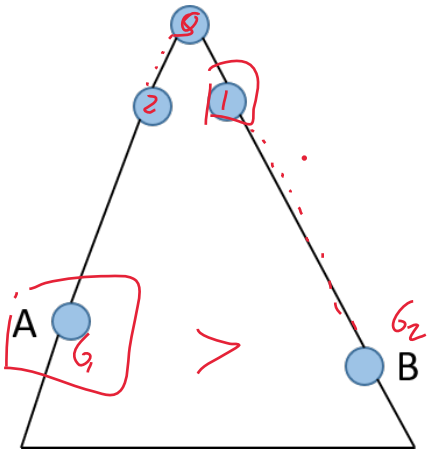
Claim: Whenever A^* selects a node n for expansion, the optimal path to that node has been found.

Proof:
Goal: contradiction. Suppose not, or that when A^* expands a node, it wasn't the optimal path.
$\implies$ there exists an unexplored lower-cost path to n.

This unexplored lower-cost path must end with a frontier node, since it hasn't been expanded yet. Call this node n'.

Since f is non-decreasing along this path, we know that $\implies f(n') \leq f(n)$, so n' should have been expanded first, and we would have found the lower-cost path to n.

Optimality Depicted



A should exit the frontier before B!

Optimally Efficient

So:

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.

The result: the first goal node to be expanded took the lowest-cost path, and all later goal node expansions are at least as expensive. That means our solution was **optimal!**

Optimality underview

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

So why ever use anything else?

1. The number of nodes to expand along the goal contour is still **exponential** in depth of the solution/length-of-path.
2. We may generate inefficiencies depending on the quality of our heuristic.
   - 2.1 **Definition:** The *absolute error* of our heuristic h usd to estimate the true cost from root to goal h^* is $\Delta := h^* - h$.
   - 2.2 **Definition:** The *relative error* of our heuristic is $\varepsilon := \frac{h^* - h}{h*}$.

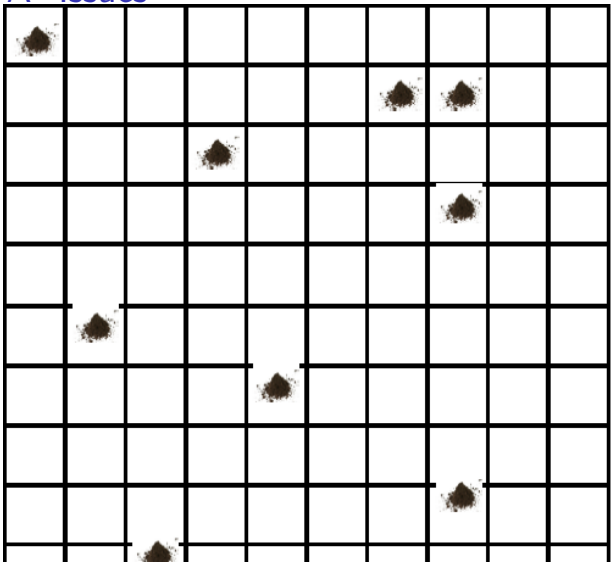
A* Efficiency

$b^{\varepsilon \cdot d} = (b^{\varepsilon})^{d} = (b^{d})^{\varepsilon}$ b^d : BFS complexity

Ultimately, the complexity of A^* depends strongly on the state space characterization and heuristic.

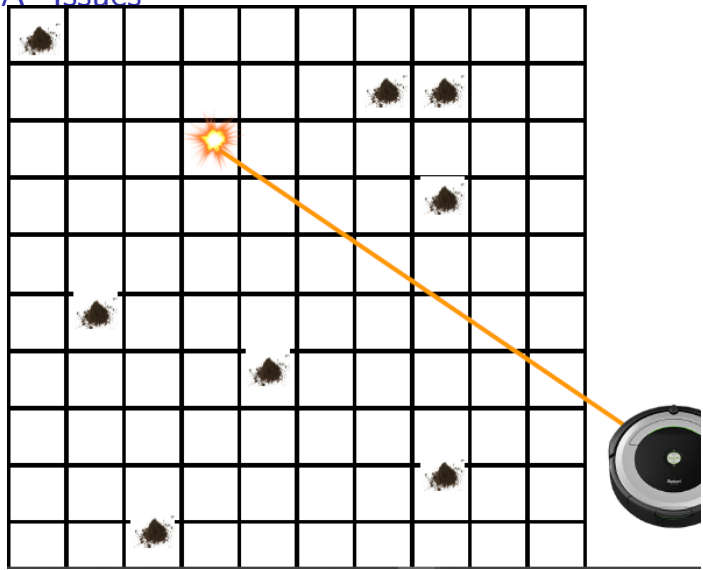
1. In the case of a single goal along a tree with reversible actions (undirected), we get $\boldsymbol{O}(b^{\Delta})$ or $\boldsymbol{O}(b^{\varepsilon d})$ if step costs are constant (and solution is at depth d).
2. Δ is proportional to the (true) path cost h^*, so ε is approximately constant - or growing slowly with d 0 and we can rewrite $\boldsymbol{O}(b^{\varepsilon d}) = \boldsymbol{O}\left((b^{\varepsilon})^{d}\right)$.
3. This looks like the cost of e.g. BFS: the (b^{ε}) term is an *effective* branching factor, so it becomes important to choose as strong of a heuristic as possible.
4. Having numerous states or near-goal states can be a problem: we might have to expand a lot of branches.

A* Issues



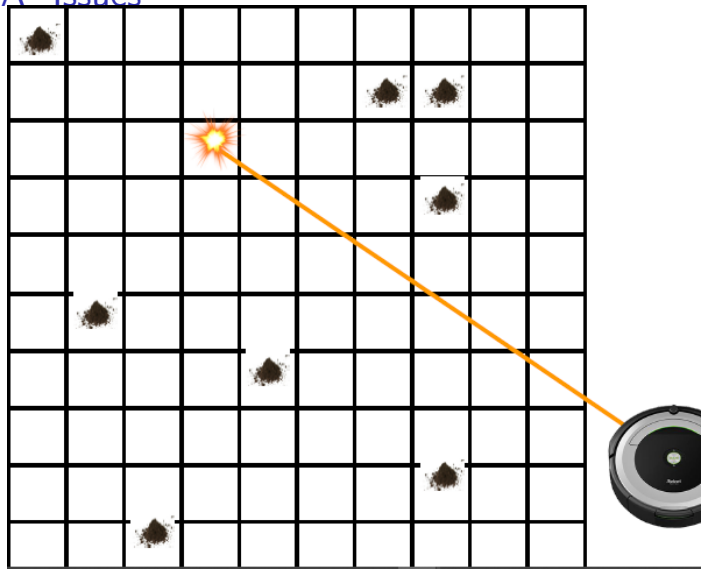
- Imagine we have a Roomba that has to clean N dirty tiles.

A* Issues



- Imagine we have a Roomba that has to clean N dirty tiles.
- But instead of moving, it just blasts tiles one at a time with a sweet laser!
- What is the optimal solution? Is it unique? How many paths will we explores

A* Issues



- Imagine we have a Roomba that has to clean N dirty tiles.
- But instead of moving, it just blasts tiles one at a time with a sweet laser!
- What is the optimal solution? Is it unique? How many paths will we explores
- **Every** solution is equal. So our A^* will unfold all 2^N states that might lead to one of the optimal solutions.

A* alternatives

Having to possibly track paths to many solutions can lead to a space complexity burden on A^*, since we keep all the generated nodes in memory.

Option: Iterative Deepening A^*.

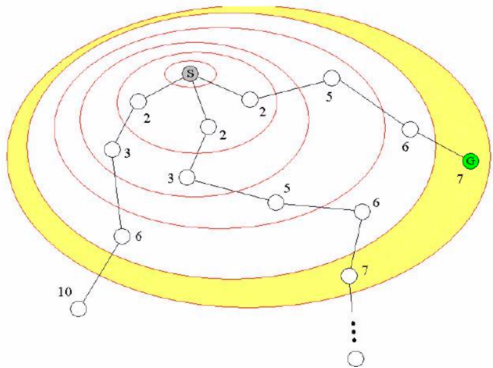
1. Same idea as iterative depth-based search.
2. Instead of expanding the search depth, extend f up to an allowed cost threshold.
3. This searches out to a particular contour one at a time, then uses prior contours' solutions to generate paths out to the next deeper contour.

A* alternatives

Having to possibly track paths to many solutions can lead to a space complexity burden on A^*, since we keep all the generated nodes in memory.

Option: Iterative Deepening A^*.

1. Same idea as iterative depth-based search.
2. Instead of expanding the search depth, extend f up to an allowed cost threshold.
3. This searches out to a particular contour one at a time, then uses prior contours' solutions to generate paths out to the next deeper contour.

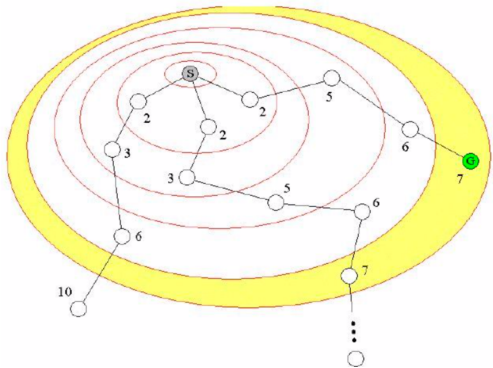


A* alternatives

Having to possibly track paths to many solutions can lead to a space complexity burden on A^*, since we keep all the generated nodes in memory.

Option: Iterative Deepening A^*.

1. Same idea as iterative depth-based search.
2. Instead of expanding the search depth, extend f up to an allowed cost threshold.
3. This searches out to a particular contour one at a time, then uses prior contours' solutions to generate paths out to the next deeper contour.



A* alternative: RBFS

Option: Recursive best-first search (RBFS).

1. Kind of like a standard DFS, but track the best alternative path's cost f as we go.
2. Once the path we're on exceeds the currently saved best-available path, switch over to a backup path.
3. As RBFS back-tracks, each node along the back-tracked path it replaces the f-value with that of the cheapest child node. Put another way: we remember the best "leaf" in the sub-tree, so RBFS knows whether or not to go back down that road later.
4. Still has some indecision and replicated-computation problems, as expanding a new path adds to costs which makes unexpanded alternatives look better.

RBFS pseudo

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 $\leftarrow$ [ ]
  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 */
    $s.f \leftarrow \max(s.g + s.h,\ node.f)$
  loop do
    best $\leftarrow$ the lowest f-value node in *successors*
    if $best.f > f\_limit$ **then return** *failure*, $best.f$
    alternative $\leftarrow$ the second-lowest f-value among *successors*
    result, $best.f \leftarrow$ RBFS(*problem*, *best*, $\min(f\_limit, alternative)$)
    if *result* $\neq$ *failure* **then return** *result*

Figure 3.26 The algorithm for recursive best-first search.

RBFS depicted

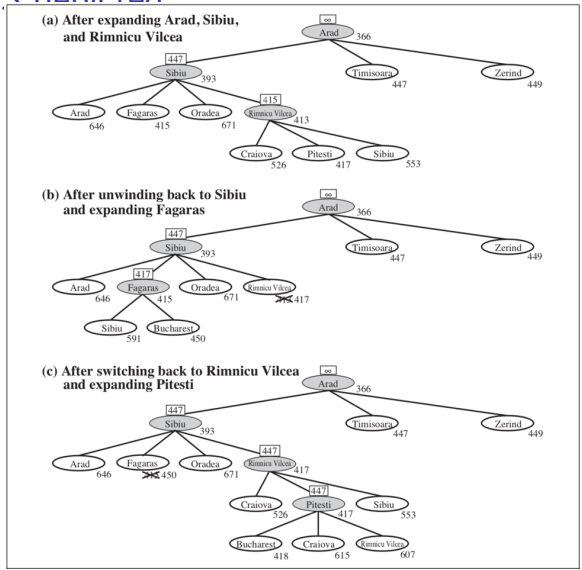


Figure 3.27 Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse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

Alternatives: complexity

Both IDA^* and RBFS use very little memory. For example, between iterations IDA^* keeps only the current $f-$cost limit, and RBFS has the memory benefits of DFS.

Unfortunately, those who do not remember the past are (maybe) doomed to repeat it!

Option: Memory-bounded A^* and Simplified Memory bounded A^* (MA^* and SMA^*.)
SMA^*:

1. Expands the best leaf *until memory is full*.
2. Then expands the best leaf and deletes the worst.
3. Concern: what if all leaves have the same $f-$ value? Then expand the newest and delete the oldest.
4. Concern: what if there is only one leaf? Then there is only one solution path from that root to the goal, and it's taking up all available memory as is. We can't solve the problem anyways.

Moving Forward

- This week:
  1. HW2 released.
  2. Additional Office Hours!
- Next time: even more on heuristics!