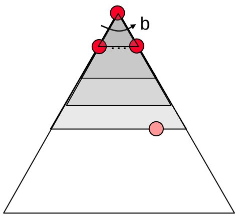
Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



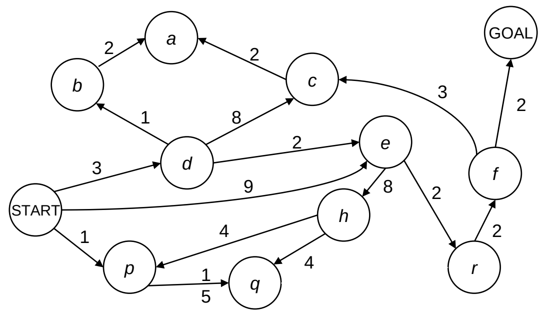
- Isn't that wastefully redundant?
  - Generally, most work happens in the lowest level searched, so not so bad!

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore first?

Cost-Sensitive Search

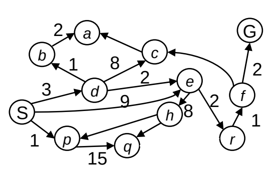


BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

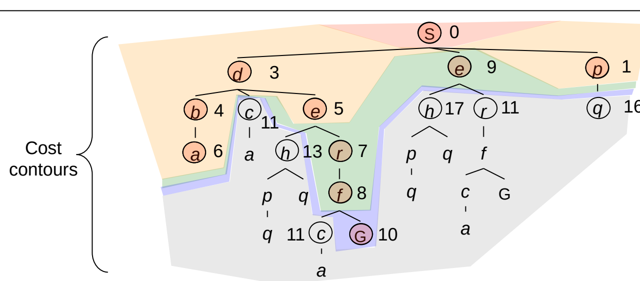
Uniform Cost Search

Strategy: expand a cheapest node on the fringe first:

Fringe is a priority queue (priority: cumulative cost)

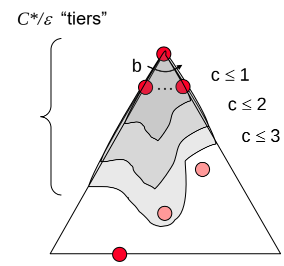


Note: This is how Dijkstra's algorithm works but it finds all shortest paths from Start



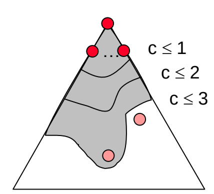
Uniform Cost Search (UCS) Properties

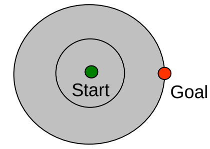
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location
- We'll fix that soon!





UCS vs Dijkstra's algorithm: Differences

The algorithms are very similar but differ in the details
Think through the potential differences:

- What is the goal of each algorithm?
- Initializing the Queue?
- Size of Queue?
- Why is UCS more desirable in our context?

The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e., collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that uses a variation on a single interface

Informed Search

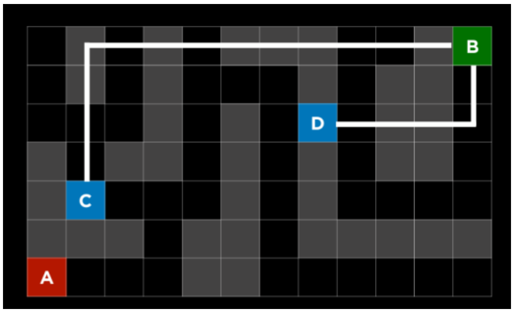
Can we do better than BFS, DFS, or UCS? **What if we have some information about where the goal is?** Up to now we have been searching in all directions and have no idea how far it is to the goal.

Heuristics *are a procedure or process to achieve something that is not guaranteed to achieve the best outcome.*

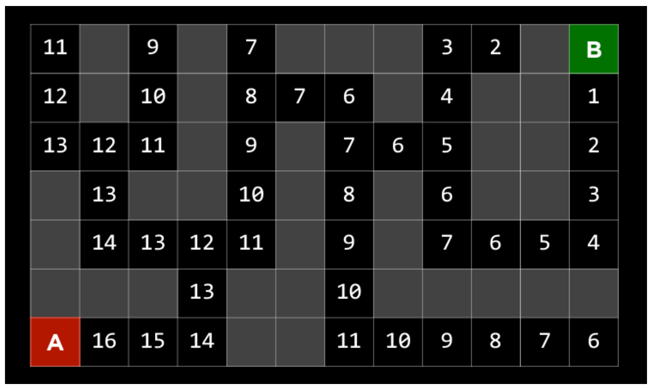
- Use a heuristic function to guide us toward the Goal
  - Greedy Best-first search
  - A* Search

Search Heuristics

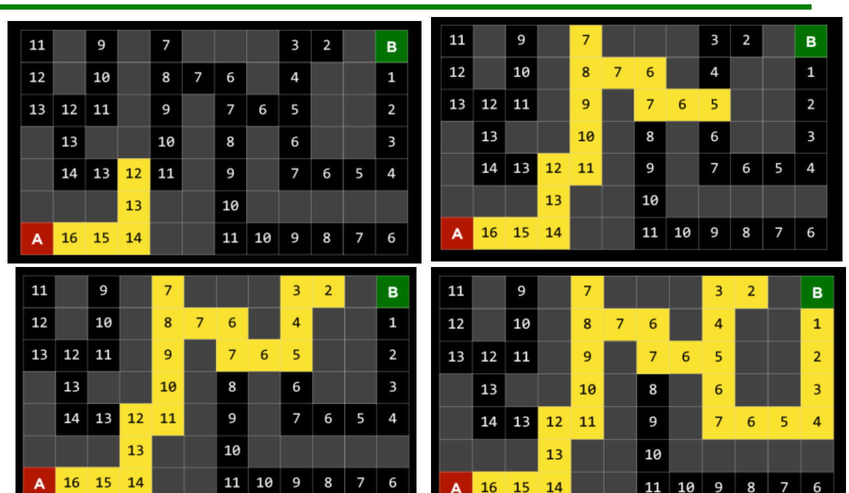
- A heuristic (in Search Problems) is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Pathing or path finding?
  - Examples: Manhattan distance, Euclidean distance for pathing



Manhattan Distances

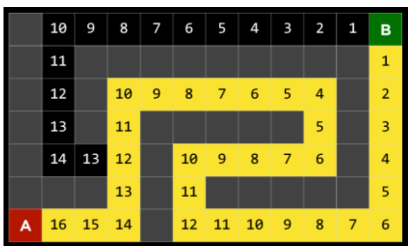


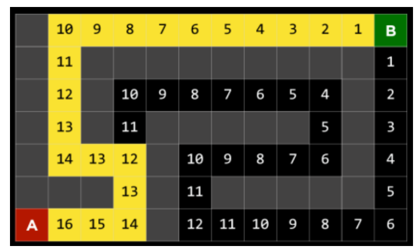
Greedy Best First Search



Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS

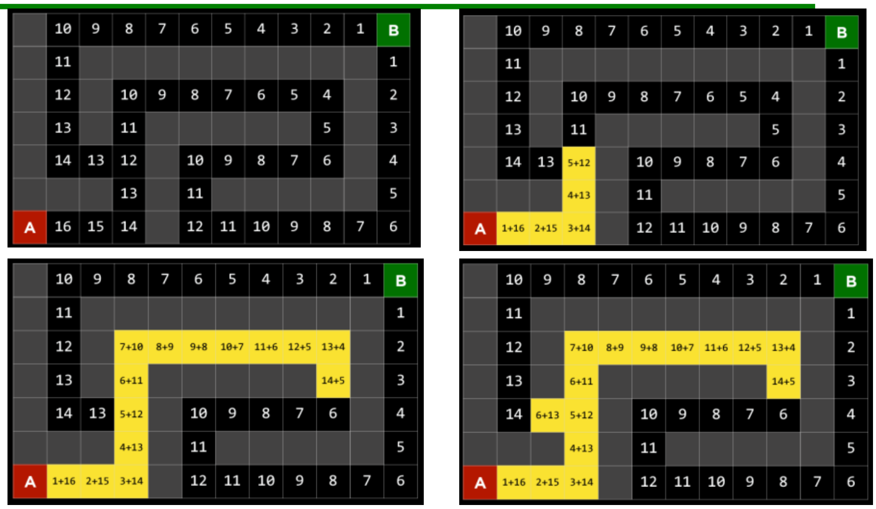




A* Search: Combine how far *traveled* with heuristic of *how far to get to goal*

- Search algorithm that expands node with lowest value of f(n) = g(n) + h(n)
  - g(n) = cost to reach node (n)
  - h(n) = estimated cost node to goal
    (heuristic ≈ what we used in greedy best-first search)

A* Search: Manhattan Distance Heuristic



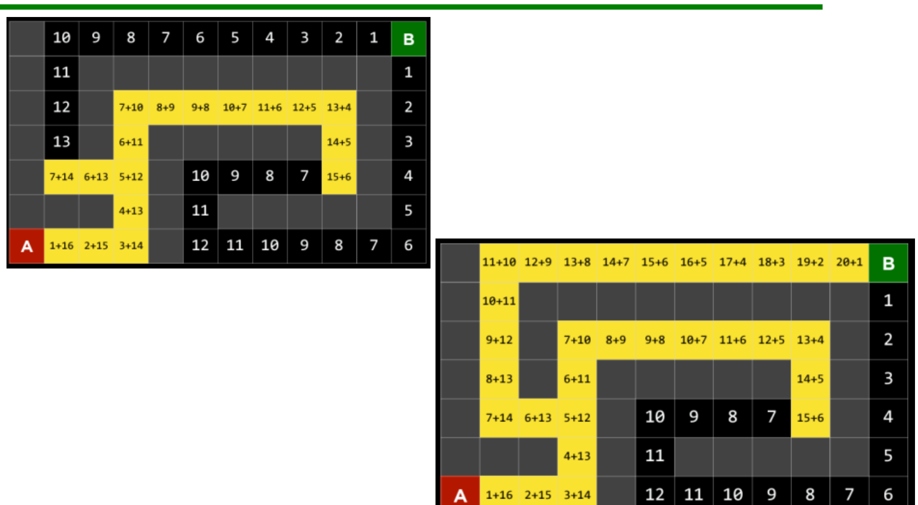
CAL POLY
SAN LUIS OBISPO

Computer Science Department

53

53

A* Search: Manhattan Distance Heuristic



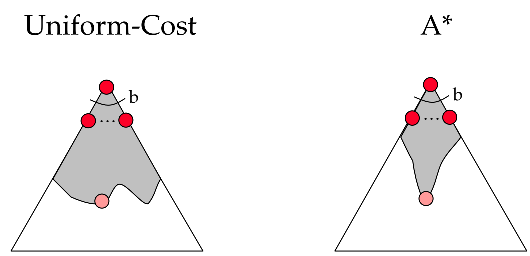
CAL POLY
SAN LUIS OBISPO

Computer Science Department

54

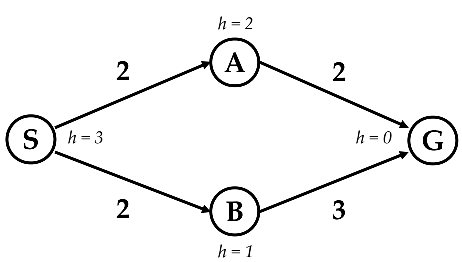
54

Properties of A*



When should A* terminate?

- Should we stop when we enqueue a goal?

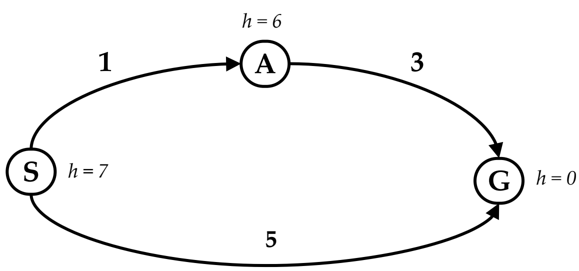


	g h +
~~S~~	~~0 3 3~~
~~S->A~~	~~2 2 4~~
~~S->B~~	~~2 1 3~~
S->B->G	5 0 5
S->A->G	4 0 4

- **No: only stop when we dequeue a goal**

Is A* Optimal?

- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!



	g h +
~~S~~	~~0+7=7~~
S->A	1+6=7
S->G	5+0=5

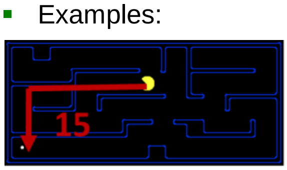
Admissible Heuristics

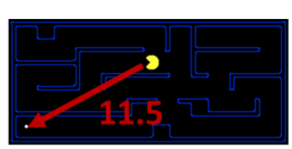
- A heuristic h is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

  where $h^*(n)$ is the true cost to a nearest goal

- Examples:





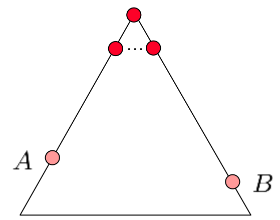
0.0

- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search

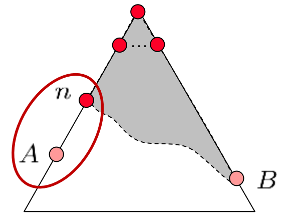
Assume:
- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:
- A will exit the fringe before B



Optimality of A* Tree Search

Proof:
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A itself)
- Claim: n will be expanded before B
  - f(n) is less or equal to f(A)

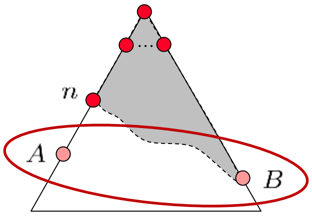


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, maybe A itself)
- Claim: n will be expanded before B
  - f(n) is less or equal to f(A)
  - f(A) is less than f(B)



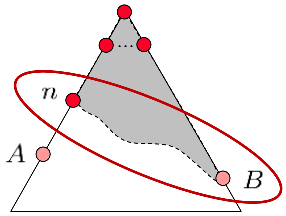
$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Optimality of A* Tree Search

Proof:

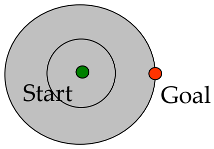
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
  - f(n) is less or equal to f(A)
  - f(A) is less than f(B)
  - n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



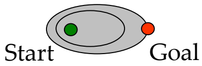
$f(n) \leq f(A) < f(B)$

UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Graph Search

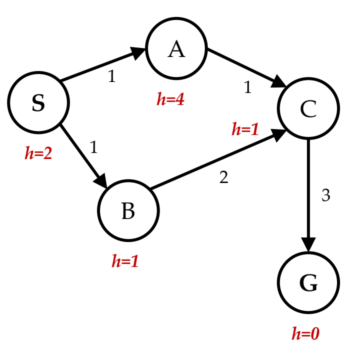
- Idea: never expand a state twice
- How to implement:
  – Tree search + set of expanded states ("closed set")
  – Expand the search tree node-by-node, but…
  – Before expanding a node, check to make sure its state has never been expanded before
  – If not new, skip it, if new add to closed set
- Important: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

Graph search

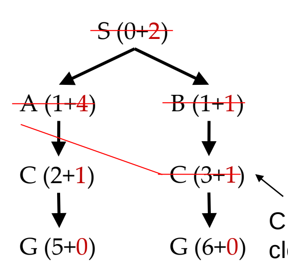
```
function Graph-Search(problem, fringe) returns a solution, or failure
   closed ← an empty set
   fringe ← Insert (Make-Node(Initial-State[problem]), fringe)
   loop do
        if fringe is empty then return failure
        node ← Remove-Front(fringe)
        if Goal-Test(problem, State[node]) then return node
        if State[node] is not in closed then
             add State[node] to closed
             fringe ← InsertAll(Expand(node, problem), fringe)
   end
```


A* Graph Search Gone Wrong?

State space graph



Search tree



Closed Set: S B C A

C is closed when removed from fringe

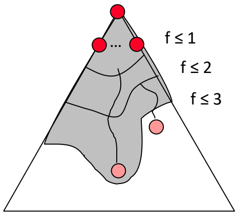
Consistency of Heuristics

- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    $h(A) \le$ actual cost from A to G = $h^*(A)$
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc
    $h(A) - h(C) \le \text{cost}(A \text{ to } C)$
- Consequences of consistency:
  - The f value along a path never decreases $h(A) \le \text{cost}(A \text{ to } C) + h(C)$
  - <u>A* graph search is optimal</u>

Optimality of A* Graph Search

Sketch of proof: consider what A* does with a consistent heuristic:

- Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
- Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
- Result: A* graph search is optimal



Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …