# Announcements

- Project 1: Search
  - It's live!  Due 9/14.
  - Start early and ask questions.  It's longer than most!
- Need a partner?  Come up after class or try Piazza
- Sections: can go to any, but have priority in your own

# CS 188: Artificial Intelligence
# Fall 2011

Lecture 3: A* Search

9/1/2011

Dan Klein – UC Berkeley

Multiple slides from Stuart Russell or Andrew Moore

## Today

- A* Search
- Graph Search
- Heuristic Design

## Recap: Search

- Search problem:
  - States (configurations of the world)
  - Successor function: a function from states to lists of (state, action, cost) triples; drawn as a graph
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search Algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

# Example: Pancake Problem

**BOUNDS FOR SORTING BY PREFIX REVERSAL**

William H. GATES
*Microsoft, Albuquerque, New Mexico*

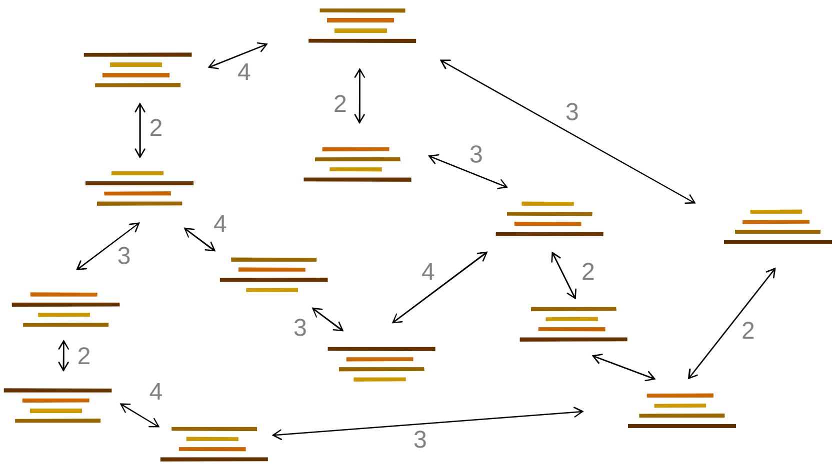
Christos H. PAPADIMITRIOU*†
*Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.*

Received 18 January 1978
Revised 28 August 1978

For a permutation $\sigma$ of the integers from 1 to $n$, let $f(\sigma)$ be the smallest number of prefix reversals that will transform $\sigma$ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all $\sigma$ in (the symmetric group) $S_n$. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ for $n$ a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2-1 \leqslant g(n) \leqslant 2n+3$.

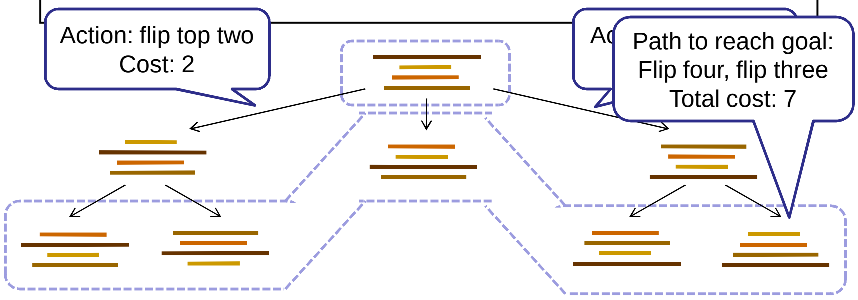
# Example: Pancake Problem

State space graph with costs as weights



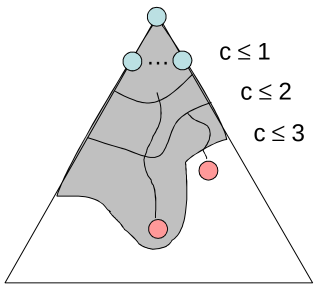
# General Tree Search

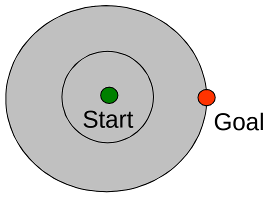
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the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




# Uniform Cost Search

- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location

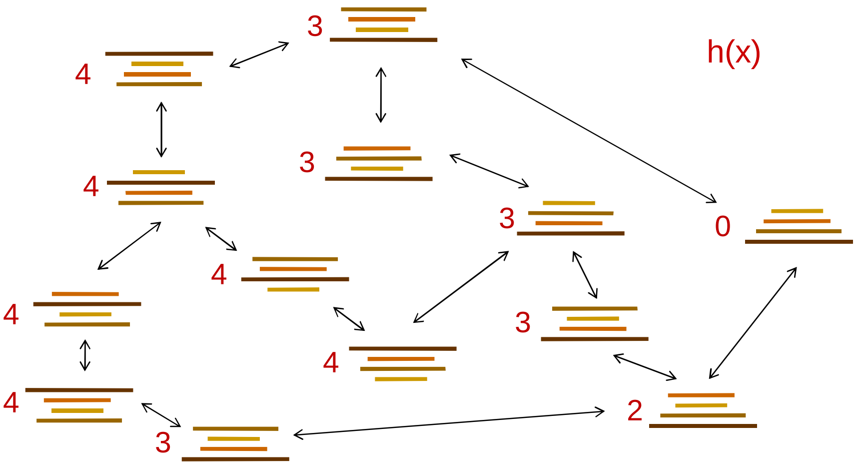


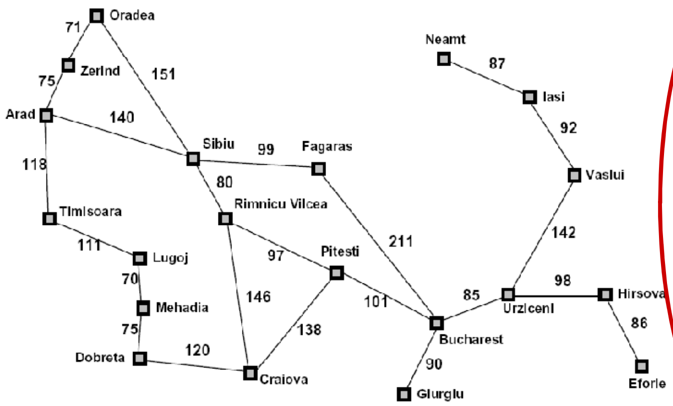

[demo: countours UCS]

# Example: Heuristic Function

Heuristic: the largest pancake that is still out of place




# Example: Heuristic Function

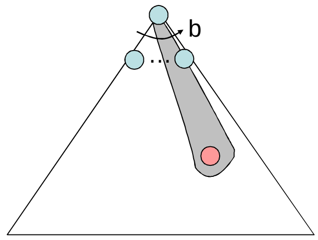


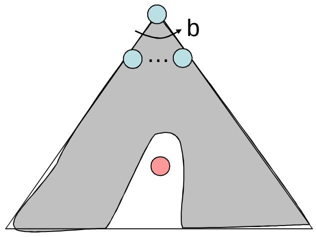

| Straight-line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

h(x)

# Best First (Greedy)

- 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

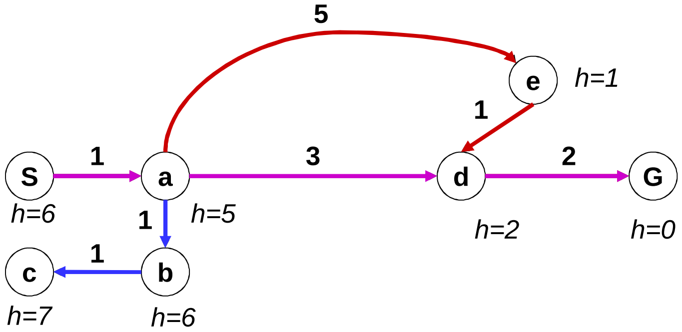



[demo: countours greedy]

# Combining UCS and Greedy

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

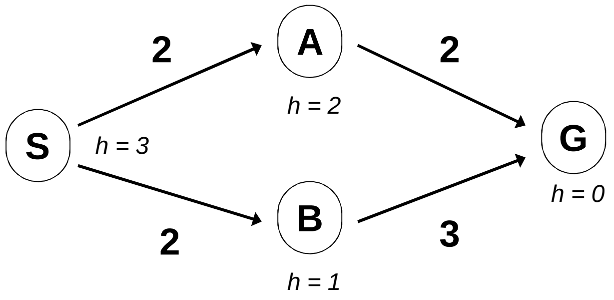



- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

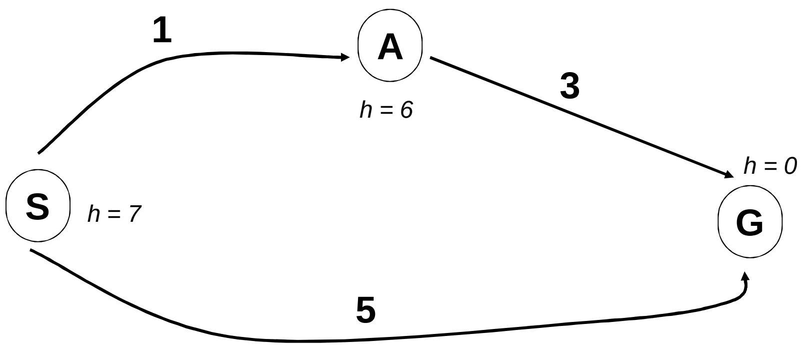
## When should A* terminate?

- Should we stop when we enqueue a goal?




- 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!

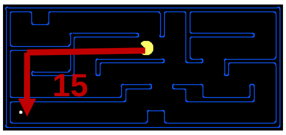
## Admissible Heuristics

- A heuristic $h$ is *admissible* (optimistic) if:

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

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

- Examples:

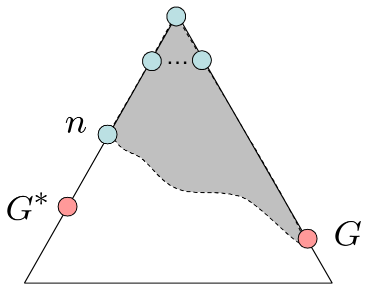




- Coming up with admissible heuristics is most of what's involved in using A* in practice.

## Optimality of A*: Blocking

Notation:

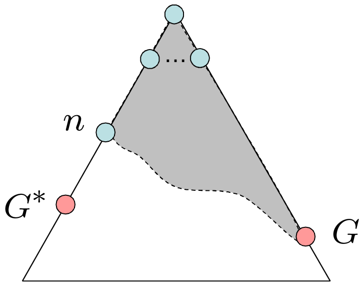
- g(n) = cost to node n
- h(n) = estimated cost from n to the nearest goal (heuristic)
- f(n) = g(n) + h(n) = estimated total cost via n
- G*: a lowest cost goal node
- G: another goal node



# Optimality of A*: Blocking

Proof:

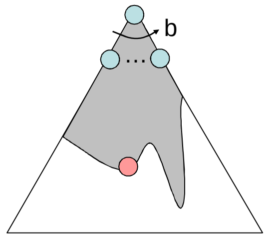
- What could go wrong?
- We'd have to have to pop a suboptimal goal G off the fringe before G*
- This can't happen:
  - Imagine a suboptimal goal G is on the queue
  - Some node *n* which is a subpath of G* must also be on the fringe (why?)
  - *n* will be popped before G



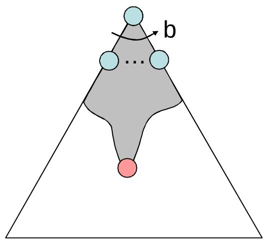

$$f(n) = g(n) + h(n)$$
$$g(n) + h(n) \leq g(G^*)$$
$$g(G^*) < g(G)$$
$$g(G) = f(G)$$
$$f(n) < f(G)$$

# Properties of A*

Uniform-Cost

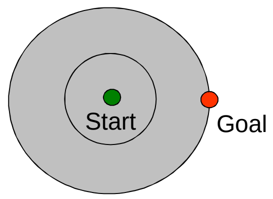



A*

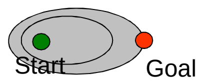


# UCS vs A* Contours

- Uniform-cost expanded in all directions



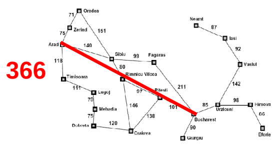

- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

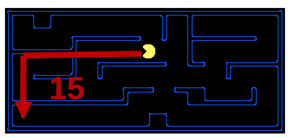



[demo: countours UCS / A*]

# Creating Admissible Heuristics

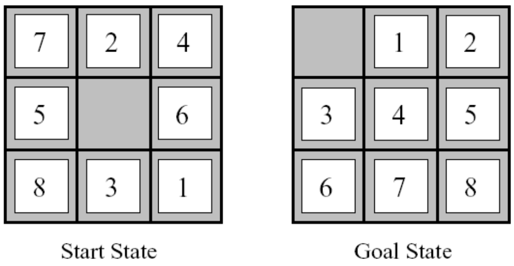
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available




- Inadmissible heuristics are often useful too (why?)

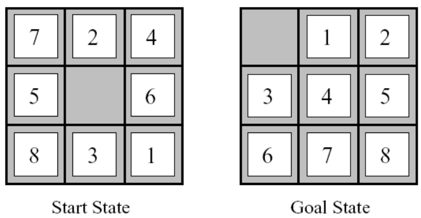
# Example: 8 Puzzle




- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

# 8 Puzzle I

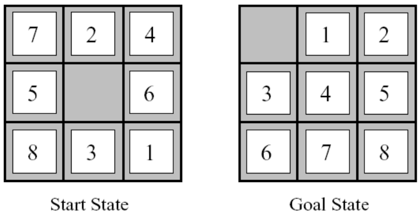
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a relaxed-problem heuristic




| | Average nodes expanded when optimal path has length… | | |
|---|---|---|---|
| | …4 steps | …8 steps | …12 steps |
| UCS | 112 | 6,300 | $3.6 \times 10^6$ |
| TILES | 13 | 39 | 227 |

# 8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?




- h(start) =

  3 + 1 + 2 + …

  = 18

| | Average nodes expanded when optimal path has length… | | |
|---|---|---|---|
| | …4 steps | …8 steps | …12 steps |
| TILES | 13 | 39 | 227 |
| MANHATTAN | 12 | 25 | 73 |

# 8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?
- With A*: a trade-off between quality of estimate and work per node!

## Trivial Heuristics, Dominance

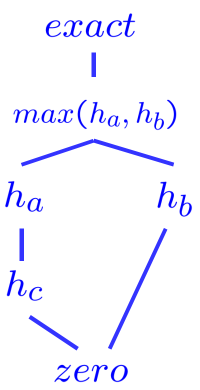
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

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

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



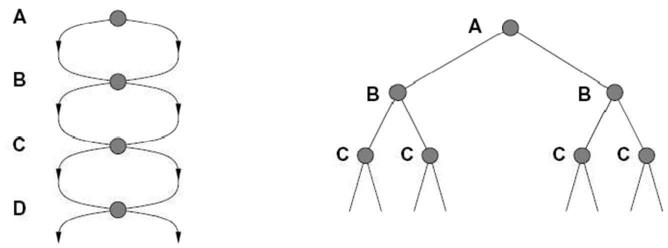

## Other A* Applications

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

[demo: plan tiny UCS / A*]

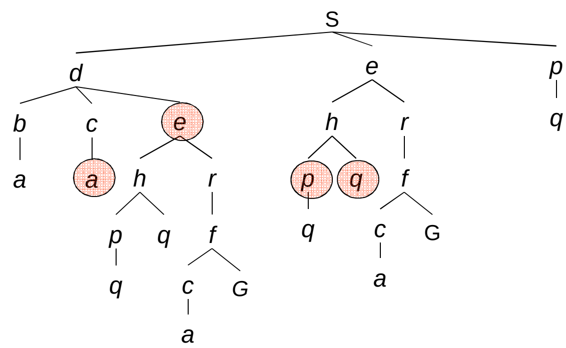
## Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work. Why?




## Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



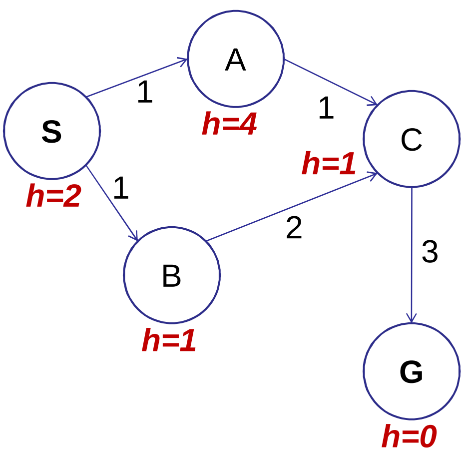
# Graph Search

- 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 is new
  - If not new, skip it
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

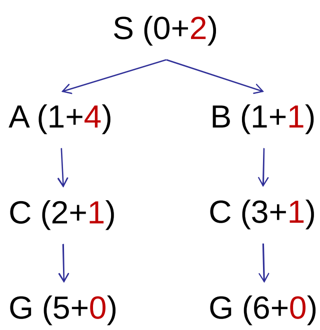
Warning: 3e book has a more complex, but also correct, variant

# A* Graph Search Gone Wrong?

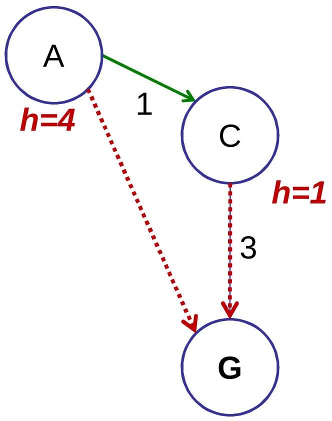
**State space graph**




**Search tree**



# Consistency of Heuristics




- Stronger than admissibility
- Definition:

  cost(A to C) + h(C) ≥ h(A)

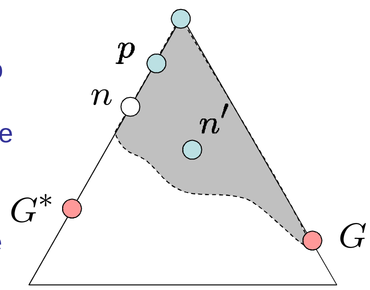
  cost(A to C) ≥ h(A) - h(C)

  real cost ≥ cost implied by heuristic
- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal

# Optimality of A* Graph Search

Proof:

- New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expanded first (disaster!)
- Take the highest such *n* in tree
- Let *p* be the ancestor of *n* that was on the queue when *n*' was popped
- *f(p) < f(n)* because of consistency
- *f(n) < f(n')* because *n'* is suboptimal
- *p* would have been expanded before *n*'
- Contradiction!



# Optimality

- Tree search:
  - A* is optimal if heuristic is admissible (and non-negative)
  - 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

# Summary: A*

- 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

# Mazeworld Demos