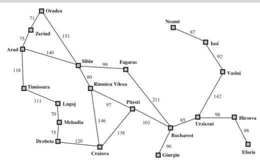
Uninformed Search 2
Informed Search
AI Class 5 (Ch. 3.5-3.7)



Based on slides by Dr. Marie desJardin. Some material also adapted from slides by Dr. Matuszek @ Villanova University, which are based on Hwee Tou Ng at Berkeley, which are based on Russell at Berkeley. Some diagrams are based on AIMA.

Dr. Cynthia Matuszek – CMSC 671

Today's Class

- Rest of blind search
- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A, A*
  - Examples
- Memory-conserving variations of A*
- Heuristic functions

"An informed search strategy—one that uses problem specific knowledge… can find solutions more efficiently then an uninformed strategy."

– R&N pg. 92

2

Questions?

3

Things to Differentiate

- Goal testing
- Expanding
- Generating

4

Blind Search (Redux)

- Last time:
  - Bread-first
  - Depth-first
  - Uniform-cost
- This time:
  - Iterative deepening
  - Bidirectional
  - Holy Grail Search

5

"Satisficing"

- Wikipedia: "**Satisficing** is … searching until an **acceptability threshold** is met"
- Contrast with **optimality**
  - Satisficable problems *do not get more benefit from* finding an optimal solution
- Ex: You have an A in the class. Studying for four hours will get you a 95 on the final. Studying for four more (eight hours) will get you a 99 on the final. What to do?
- A combination of *satisfy* and *suffice*
- Introduced by Herbert A. Simon in 1956

Another piece of **problem definition**

6

Depth-First Iterative Deepening (DFID)

1. DFS to depth 0 (i.e., treat start node as having no successors)
2. Iff no solution found, do DFS to depth 1

> until solution found do:
> DFS with depth cutoff c;
> c = c+1

- Complete
- Optimal/Admissible if all operators have the same cost
  - Otherwise, not optimal, but guarantees finding solution of shortest length
- Time complexity is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times
- Because most nodes are near the bottom of a tree, worst case time complexity is still exponential, O(bd)

7

Depth-First Iterative Deepening (DFID)

1. DFS to depth 0 (i.e., treat start node as having no successors)
2. Iff no solution found, do DFS to depth 1

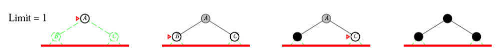
> until solution found do:
> DFS with depth cutoff c;
> c = c+1

- Complete
- Optimal/Admissible if all operators have the same cost
  - Otherwis…
- Time com… …s near the top of…
- Because n… …e complexit…

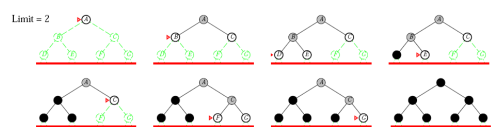
> The key: at every stage, **throw away work from previous stages** (or you don't save anything!)

8

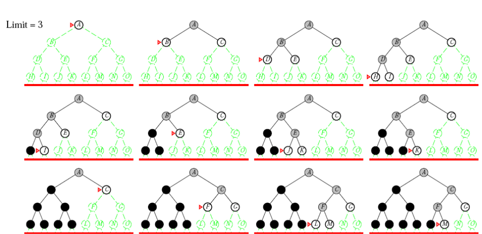
Iterative deepening search (c=1)



Iterative deepening search (c=2)



Iterative deepening search (c=3)

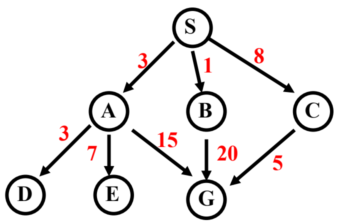


Depth-First Iterative Deepening

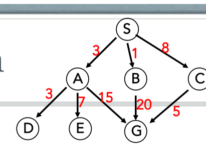
- If branching factor is *b* and solution is at depth *d*, then nodes at depth *d* are generated once, nodes at depth *d*-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db \le b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, then worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- **Linear space complexity**, O(bd), like DFS
- Has advantage of both BFS (completeness) and DFS (limited space, finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

12

Example for Illustrating Search Strategies



Depth-First Search

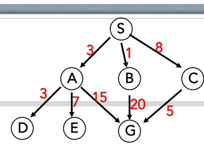


Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Depth-First Search



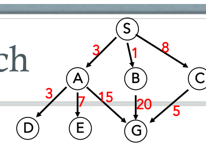
Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A	
D	
E	
G	

We won't go through these in detail, but please make sure you understand them.

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Breadth-First Search

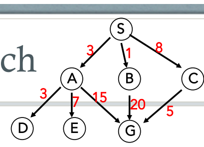


Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Uniform-Cost Search



Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }

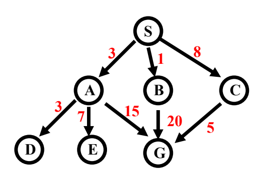
Solution path found is S C G, cost 13

Number of nodes expanded (including goal node) = 7

How they Perform

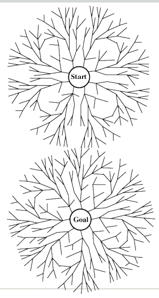
- **Depth-First Search:**
  - Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)

  This is the only uninformed search that worries about costs.
- **Iterative-Deepening Search**:
  - nodes expanded: S S A B C S A D E G



Bi-directional Search

- Alternate searching from
  - start state → goal
  - goal state → start
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states
- Requires ability to generate "predecessor" states.
- Can (sometimes) find a solution fast

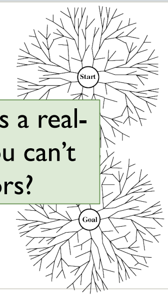


19

Bi-directional Search

- Alternate searching from
  - start state → goal
  - goal state → start
- St[...]
- W[...] u[...]
- R[...] "predecessor" states.
- Can (sometimes) find a solution fast

Thought problems: What's a real-world problem where you can't generate predecessors?



20

Comparing Search Strategies

	Complete	Optimal	Time complexity	Space complexity
Breadth first search:	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	O(bm)
Depth limited search	if l >= d	no	$O(b^l)$	O(bl)
depth first iterative deepening search	yes	yes	$O(b^d)$	O(bd)
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

b is branching factor, d is depth of the shallowest solution, m is the maximum depth of the search tree, l is the depth limit

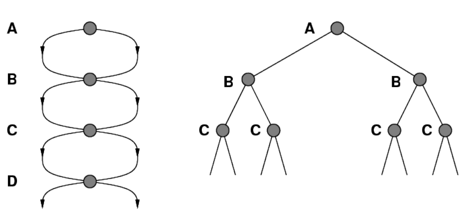
21

Avoiding Repeated States

- Ways to reduce size of state space (with increasing computational costs)
- In increasing order of effectiveness:
  1. Do not return to the state you just came from.
  2. Do not create paths with cycles in them.
  3. Do not generate any state that was ever created before.
- Effect depends on frequency of loops in state space.

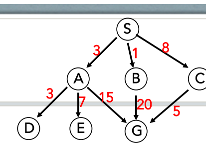
22

A State Space that Generates an Exponentially Growing Search Space



23

Holy Grail Search



Expanded node	Nodes list
	{ S^0 }
S^0	{ C^8 A^3 B^1 }
C^8	{ G^{13} A^3 B^1 }
G^{13}	{ A^3 B^1 }

Solution path found is S C G, cost 13 (optimal)

Number of nodes expanded (including goal node) = 3 (minimum possible!)

24

Holy Grail Search

Why not go straight to the solution, without any wasted detours off to the side?

<foreshadowing> **If only we knew where we were headed…** </foreshadowing>

Informed Search

"An informed search strategy—one that uses problem specific knowledge… can find solutions more efficiently then an uninformed strategy." – *R&N pg. 92*

Weak vs. Strong Methods

- **Weak methods**:
  - Extremely general, not tailored to a specific situation
- Examples
  - **Subgoaling**: split a large problem into several smaller ones that can be solved one at a time.
  - **Space splitting:** try to list possible solutions to a problem, then try to rule out *classes* of these possibilities
  - **Means-ends analysis:** consider current situation and goal, then look for ways to shrink the differences between the two
- Called "weak" methods because they do not take advantage of more powerful domain-specific heuristics

Heuristic

Free On-line Dictionary of Computing*

1. **A rule of thumb, simplification, or educated guess**
2. Reduces, limits, or guides search in particular domains
3. Does not guarantee feasible solutions; often used with no theoretical guarantee

WordNet (r) 1.6*

1. Commonsense rule (or set of rules) intended to increase the probability of solving some problem

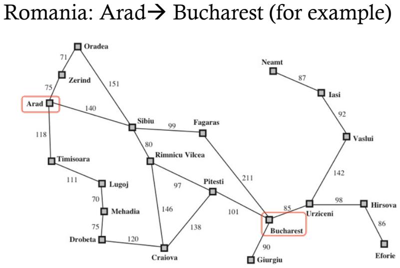
*Heavily edited for clarity

Heuristic Search

- Uninformed search is **generic**
  - Node selection depends only on shape of tree and node expansion strategy.
- Sometimes **domain knowledge** → Better decision
  - Knowledge about the specific problem

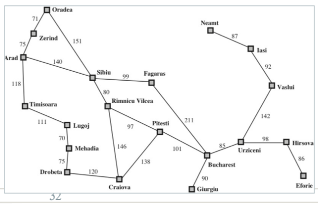
Heuristic Search

- Romania: Arad→ Bucharest (for example)



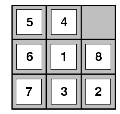
Heuristic Search

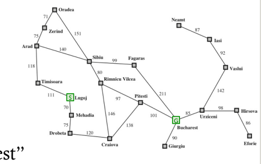
- Romania:
  - Eyeballing it → certain cities first
  - They "look closer" to where we are going
- Can domain knowledge be captured in a **heuristic?**



Heuristics Examples

- 8-puzzle:
  - # of tiles in wrong place
- 8-puzzle (better):
  - Sum of distances from goal
  - Captures distance *and* number of nodes
- Romania:
  - Straight-line distance from start node to Bucharest
  - Captures "closer to Bucharest"

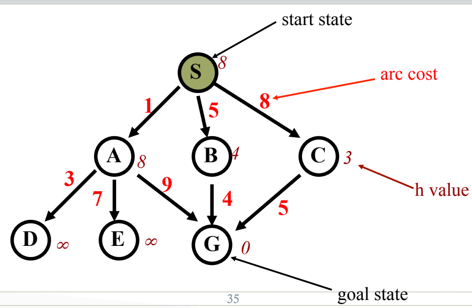




Heuristic Function

- **All** domain-specific knowledge is encoded in heuristic function h
- h is some **estimate** of how desirable a move is
  - How "close" (we think, maybe) it gets us to our goal
- Usually:
  - $h(n) \geq 0$: for all nodes n
  - $h(n) = 0$: n is a goal node
  - $h(n) = \infty$: n is a dead end (no goal can be reached from n)

Example Search Space Revisited



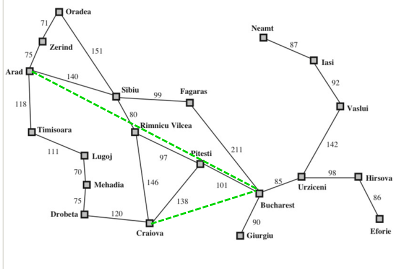
Domain Information

- Informed methods add domain-specific information!
- Goal: **select** the best path to continue searching
- Define ***h(n)*** to estimate the "goodness" of node n
  - $h(n)$ = **estimated cost** (or distance) of minimal cost path from **n to a goal state**

Is It A Heuristic?

- A **heuristic function** is:
  - An **estimate** of how close we are to a goal
    - We don't assume perfect knowledge
      - That would be holy grail search
    - The estimate can be wrong
  - Based on domain-specific information
  - Computable from the current state description

Straight Lines to Budapest (km)



	$h_{SLD}(n)$
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

R&N pg. 68, 93

Admissible Heuristics

- Admissible heuristics never overestimate cost
  - They are *optimistic* – think goal is closer than it is
  - $h(n) \leq h^*(n)$
    - where $h^*(n)$ is **true** cost to reach goal from n
  - h_{LSD}(Lugoj) = 244
    - Can there be a shorter path?
- Using admissible heuristics guarantees that the first solution found will be optimal

Best-First Search

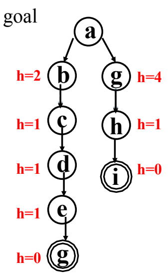
- A generic way of referring to informed methods
- Use an **evaluation function** $f(n)$ for each node → estimate of "desirability"
  - $f(n)$ incorporates domain-specific information
  - Different $f(n)$ → Different searches

Best-First Search (more)

- Order nodes on the list by
  - Increasing value of $f(n)$
- Expand **most desirable** unexpanded node
  - Implementation:
  - Order nodes in frontier in decreasing order of desirability
- Special cases:
  - Greedy best-first search
  - A* search

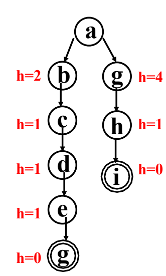
Greedy Best-First Search

- Idea: always choose "closest node" to goal
  - Most likely to lead to a solution quickly
- So, evaluate nodes based only on heuristic function
  - $f(n) = h(n)$
- Sort nodes by increasing values of f
- Select node believed to be **closest** to a goal node (hence "greedy")
  - That is, select node with smallest f value

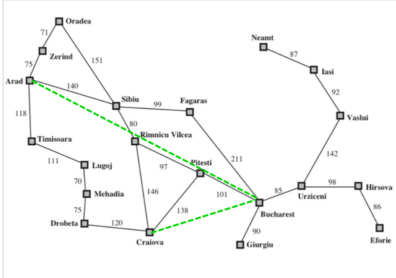


Greedy Best-First Search

- Admissible?
  - Why not?
- Example:
  - Greedy search will find:
    a→b→c→d→e→g ; cost = 5
  - Optimal solution:
    a→g→h→i ; cost = 3
- Not complete (why?)



Straight Lines to Budapest (km)

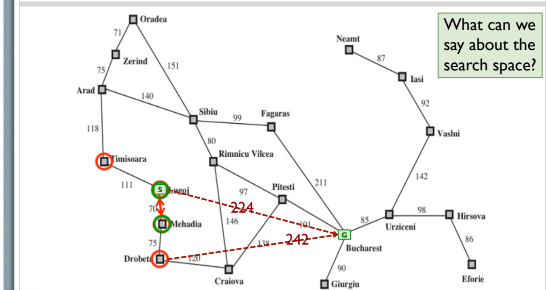


$h_{SLD}(n)$

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

R&N pg. 68, 93

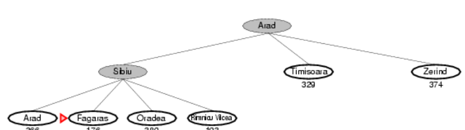
Greedy Best-First Search: Ex. 1



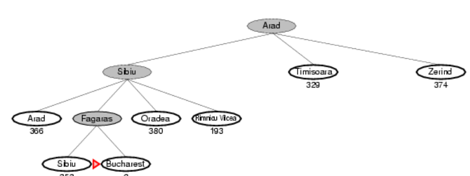
Greedy Best-First Search: Ex. 2

Greedy Best-First Search: Ex. 2

Greedy Best-First Search: Ex. 2



Greedy Best-First Search: Ex. 2



Beam Search

- Use an evaluation function $f(n) = h(n)$, but the maximum size of the nodes list is k, a fixed constant
- Only keeps k best nodes as candidates for expansion, and throws the rest away
- More space-efficient than greedy search, but may throw away a node that is on a solution path
- Not complete
- Not admissible

Quick Terminology Reminders

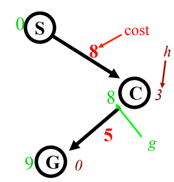
- What is $f(n)$?
  - An **evaluation function** that gives…
  - A cost estimate of...
  - The distance from n to G
- What is $h(n)$?
  - A **heuristic function** that…
  - Encodes domain knowledge about...
  - The search space
- What is $h^*(n)$?
  - A **heuristic function** that gives the…
  - **True** cost to reach goal from n
  - Why don't we just use that?
- What is $g(n)$?
  - The **path cost** of getting from S to n
  - describes the "already spent" costs of the current search

Algorithm A*

- Use evaluation function $\boldsymbol{f(n) = g(n) + h(n)}$
- $g(n)$ = minimal-cost path from any S to state n
  - That is, the cost of getting to the node **so far**
- Ranks nodes on frontier by *estimated* cost of solution
  - From start node, through given node, to goal
- Not complete if $h(n)$ can $= \infty$

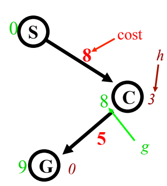
A* Search

- **Idea:** Evaluate nodes by combining g(n), the cost of reaching the node, with h(n), the cost of getting from the node to the goal.
- Evaluation function: $\boldsymbol{f(n) = g(n) + h(n)}$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

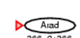


A* Search

- Avoid expanding paths that are already expensive
  - Combines costs-so-far with expected-costs
- A* is **complete** iff
  - Branching factor is finite
  - Every operator has a fixed positive cost
- A* is **admissible** iff
  - $h(n)$ is admissible

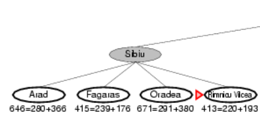


A* Example 1

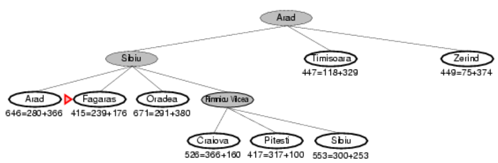


A* Example 1

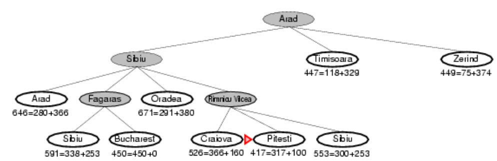
A* Example 1



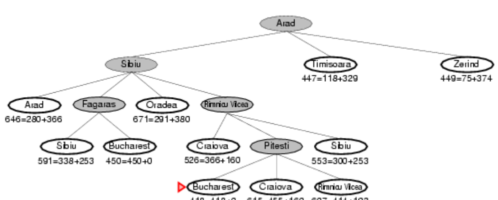
A* Example 1



A* Example 1



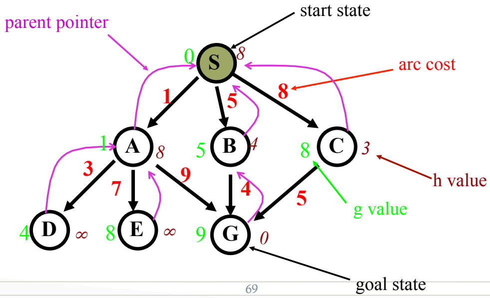
A* Example 1



Algorithm A*

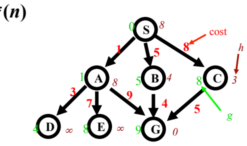
- Algorithm A with constraint that $h(n) \leq h^*(n)$
  - ***h**(*n*)** = true cost of the minimal cost path from *n* to a goal.
- Therefore, ***h*(*n*)** is an **underestimate** of the distance to the goal
- *h*() is **admissible** when $h(n) \leq h^*(n)$
  - Guarantees optimality
- A* is **complete** whenever the branching factor is finite, and every operator has a fixed positive cost
- A* is **admissible**

Example Search Space Revisited



Example

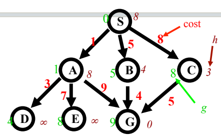
n	$g(n)$	$h(n)$	$f(n)$	$h^*(n)$
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	∞	∞	∞
E	8	∞	∞	∞
G	9	0	9	0



- $h^*(n)$ is the (hypothetical) perfect heuristic.
- Since $h(n) \leq h^*(n)$ for all n, h is admissible
- Optimal path = S B G with cost 9.

Greedy Search

$f(n) = h(n)$

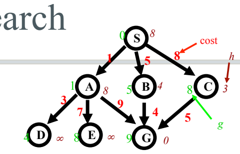


Node expanded	Node list
	{ S(8) }
S	{ C(3) B(4) A(8) }
C	{ G(0) B(4) A(8) }
G	{ B(4) A(8) }

- Solution path found is S C G, 3 nodes expanded.
- Fast!! But NOT optimal.

A* Search

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



node exp.	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }
A	{ B(9) G(10) C(11) D(∞) E(∞) }
B	{ G(9) G(10) C(11) D(inf) E(∞) }
G	{ C(11) D(∞) E(∞) }

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast, *and* optimal

Proof of the Optimality of A*

- Assume that A* has selected G_2, a goal state with a suboptimal solution ($g(G_2) > f^*$).
- We show that this is impossible.
  - Choose a node n on the optimal path to G.
  - Because $h(n)$ is admissible, $f(n) \leq f^*$.
  - If we choose G_2 instead of n for expansion, $f(G_2) \leq f(n)$.
  - This implies $f(G_2) \leq f^*$.
  - G_2 is a goal state: $h(G_2) = 0, f(G_2) = g(G_2)$.
  - Therefore $g(G_2) \leq f^*$
  - Contradiction.

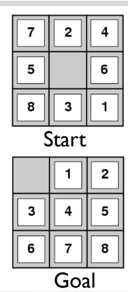
Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance
(i.e., # of squares each tile is from desired location)

- $h_1(S)$ = ?
- $h_2(S)$ = ?

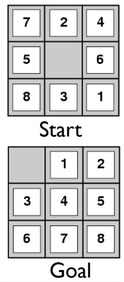


Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance
(i.e., # of squares each tile is from desired location)



- $h_1(S) = 8$
- $h_2(S) = 3+1+2+2+2+3+3+2 = 18$

Dealing with Hard Problems

- For large problems, A* often requires too much space.
- Two variations conserve memory: IDA* and SMA*
- IDA* – iterative deepening A*
  - uses successive iteration with growing limits on f. For example,
    - A* but don't consider any node n where $f(n) > 10$
    - A* but don't consider any node n where $f(n) > 20$
    - A* but don't consider any node n where $f(n) > 30$, ...
- SMA* – Simplified Memory-Bounded A*
  - uses a queue of restricted size to limit memory use.
  - throws away the "oldest" worst solution.

What's a Good Heuristic?

- If $h_1(n) < h_2(n) \leq h^*(n)$ for all n, then:
  - Both are admissible
  - h_2 is strictly better than (**dominates**) h_1.
- How do we find one?

1. **Relaxing the problem:**
   - Remove constraints to create a (much) easier problem
   - Use the solution cost for this problem as the heuristic function
2. **Combining heuristics:**
   - Take the max of several admissible heuristics
   - Still have an admissible heuristic, and it's better!

What's a Good Heuristic? (2)

3. Use statistical estimates to compute h
   - May lose admissibility
4. Identify good features, then use a learning algorithm to find a heuristic function
   - Also may lose admissibility

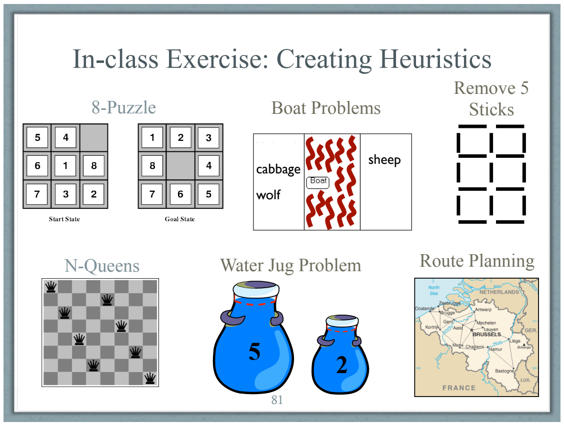
- Why are these a good idea, then?
  - Machine learning can give you answers you don't "think of"
  - Can be applied to new puzzles without human intervention
  - Often work

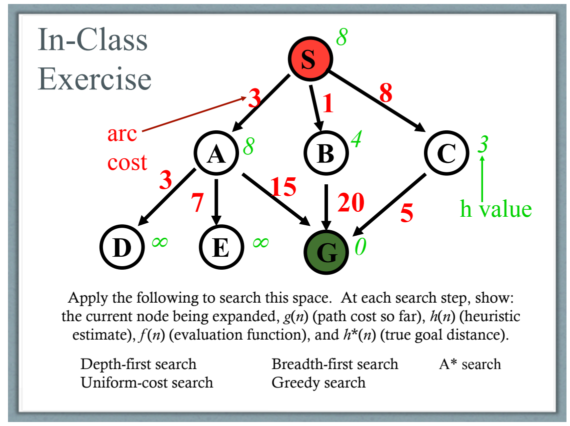
Some Examples of Heuristics?

- 8-puzzle?
  - Manhattan distance
- Driving directions?
  - Straight line distance
- Crossword puzzle?
- Making a medical diagnosis?

Summary: Informed Search

- **Best-first search:** general search where the *minimum-cost nodes* (according to some measure) are expanded first.
- **Greedy search:** uses *minimal estimated cost* $h(n)$ to the goal state as measure. Reduces search time but, is neither complete nor optimal.
- **A* search:** combines UCS and greedy search
  - $f(n) = g(n) + h(n)$
  - A* is complete and optimal, but space complexity is high.
  - Time complexity depends on the quality of the heuristic function.
- IDA* and SMA* reduce the memory requirements of A*.


In-class Exercise: Creating Heuristics
8-Puzzle
Start State
Goal State
Boat Problems
cabbage
wolf
Boat
sheep
Remove 5 Sticks
N-Queens
Water Jug Problem
5
2
Route Planning
FRANCE
81


In-Class Exercise
S
8
3
1
8
arc cost
A
8
B
4
C
3
h value
3
7
15
20
5
D
∞
E
∞
G
0
Apply the following to search this space. At each search step, show: the current node being expanded, g(n) (path cost so far), h(n) (heuristic estimate), f(n) (evaluation function), and h*(n) (true goal distance).
Depth-first search
Breadth-first search
A* search
Uniform-cost search
Greedy search