Heuristic Search and Advanced Methods

Computer Science cpsc322, Lecture 3

(Textbook Chpt 3.6 – 3.7)

May, 15, 2012

Course Announcements

Posted on WebCT

- Assignment1 (due on Thurs!)

If you are confused about basic search algorithm, different search strategies….. Check learning goals at the end of lectures. Please come to office hours

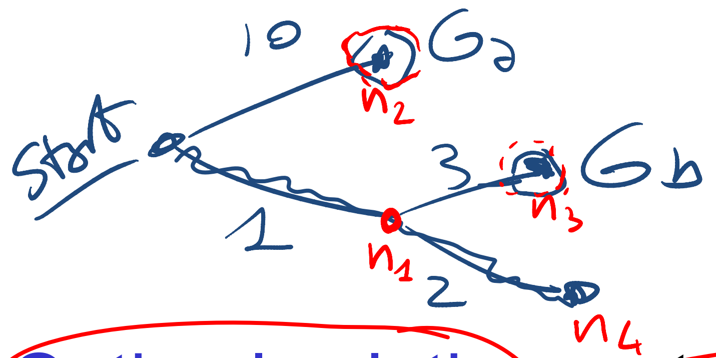
- **Work on Graph Searching Practice Ex:**
  - Exercise 3.C: heuristic search
  - Exercise 3.D: search
  - Exercise 3.E: branch and bound search

Lecture Overview

- **Recap Uninformed Cost**
- **Heuristic Search**
  - **Best-First Search**
  - **A* and its Optimality**
- **Advanced Methods**
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Recap: Search with Costs

- Sometimes there are costs associated with arcs.
  - The cost of a path is the sum of the costs of its arcs.



BFS would return $\langle start, n_2 \rangle$ as the solution

LCFS would return $\langle start, n_1, n_3 \rangle$

- Optimal solution: not the one that minimizes the *number of links*, but the one that minimizes *cost*
- Lowest-Cost-First Search: expand paths from the frontier in order of their costs.

Recap Uninformed Search

	Complete	Optimal	Time	Space
DFS	N Y if no cycles and finite search space	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y Costs > 0	Y Costs >=0	$O(b^m)$	$O(b^m)$

Recap Uninformed Search

- Why are all these strategies called uninformed?

Because they do not consider any information about the states (end nodes) to decide which path to expand first on the frontier

eg

✓ ✓ ✓

(⟨n0, n2, **n3**⟩ 12), $\underbrace{(\langle n0, \mathbf{n3}\rangle\ 8)}_{29}$, (⟨n0, n1, $\underbrace{\mathbf{n4}}_{1}$⟩ 13)

In other words, they are general they do not take into account the specific nature of the problem.

Heuristic Search

Uninformed/Blind search algorithms do not take into account the goal until they are at a goal node.

Often there is extra knowledge that can be used to guide the search: an *estimate* of the distance from node *n* to a goal node.

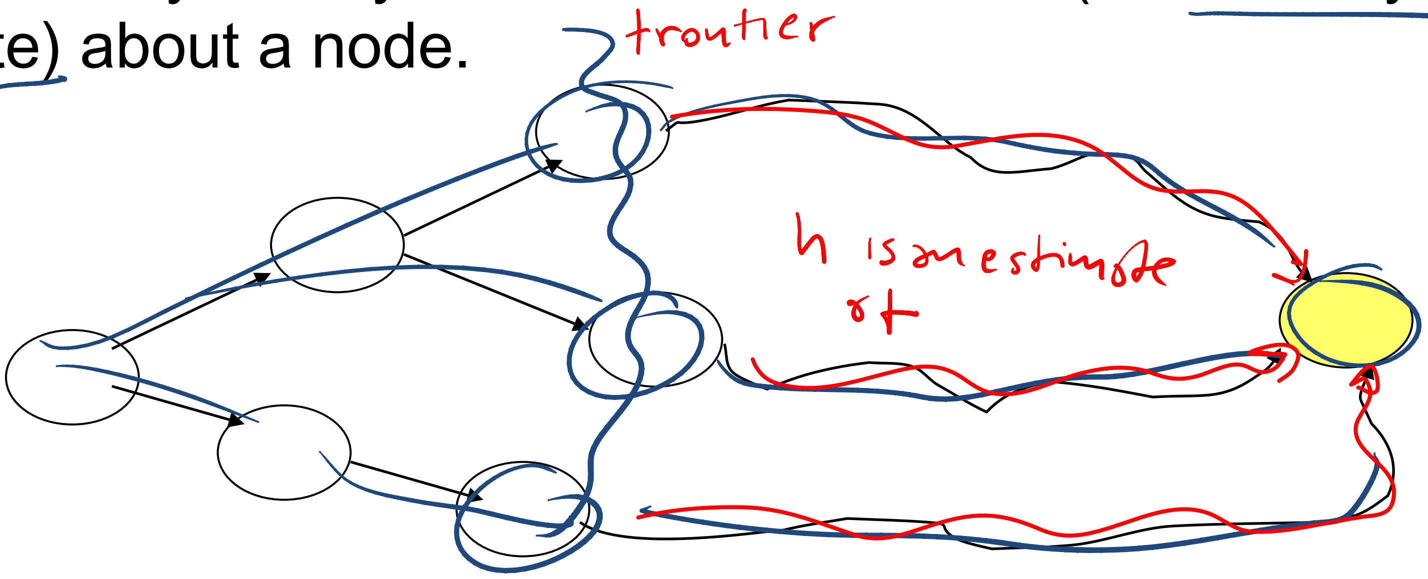
This is called a *heuristic*

More formally

Definition (search heuristic)

A search heuristic $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node.

- h can be extended to paths: $h(\langle n_0,\ldots,n_k\rangle)=h(n_k)$
- $h(n)$ uses only readily obtainable information (that is easy to compute) about a node.



More formally (cont.)

Definition (**admissible heuristic**)

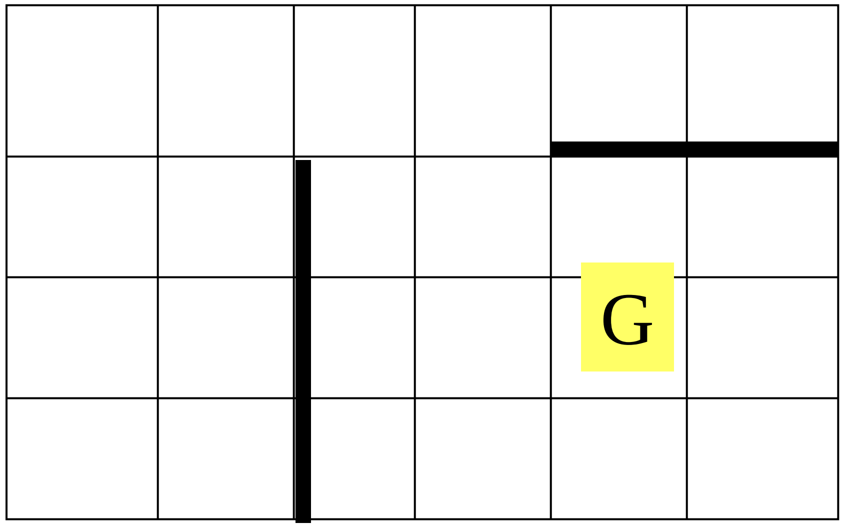
A search heuristic *h(n)* is admissible if it is never an overestimate of the cost from *n* to a goal.

- There is never a path from *n* to a goal that has path length less than *h(n)*.
- another way of saying this: *h(n)* is a lower bound on the cost of getting from *n* to the nearest goal.

Example Admissible Heuristic Functions

Search problem: robot has to find a route from start location to goal location on a grid (discrete space with obstacles)

Final cost (quality of the solution) is the number of steps



Example Admissible Heuristic Functions

If no obstacles, cost of optimal solution is…

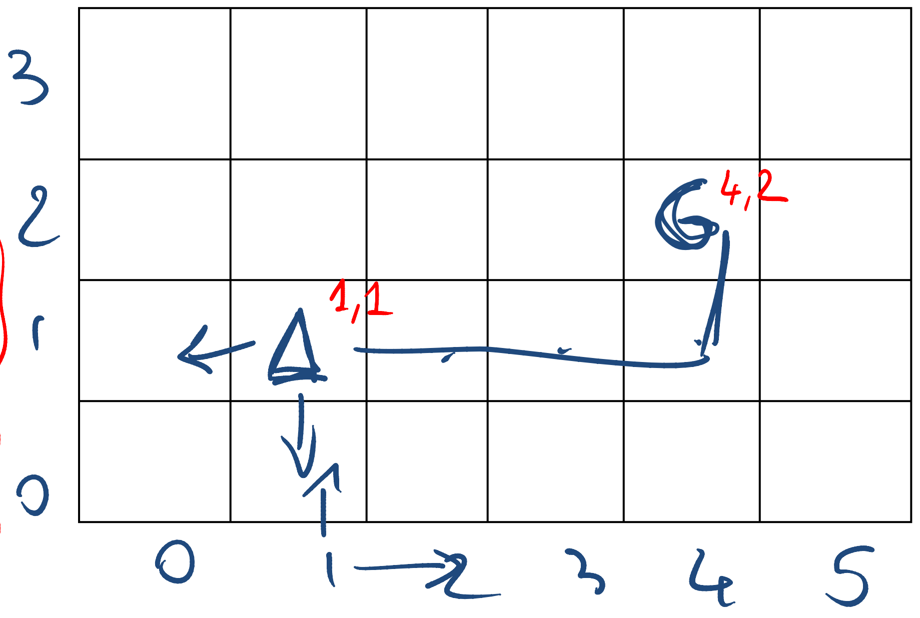
$X_G \quad Y_G$ Goal state

$X_C \quad Y_C$ Current state

$|X_G - X_C| + |Y_G + Y_C|$

In example $|4-1| + |2-1|$

Manhattan distance $= 4$

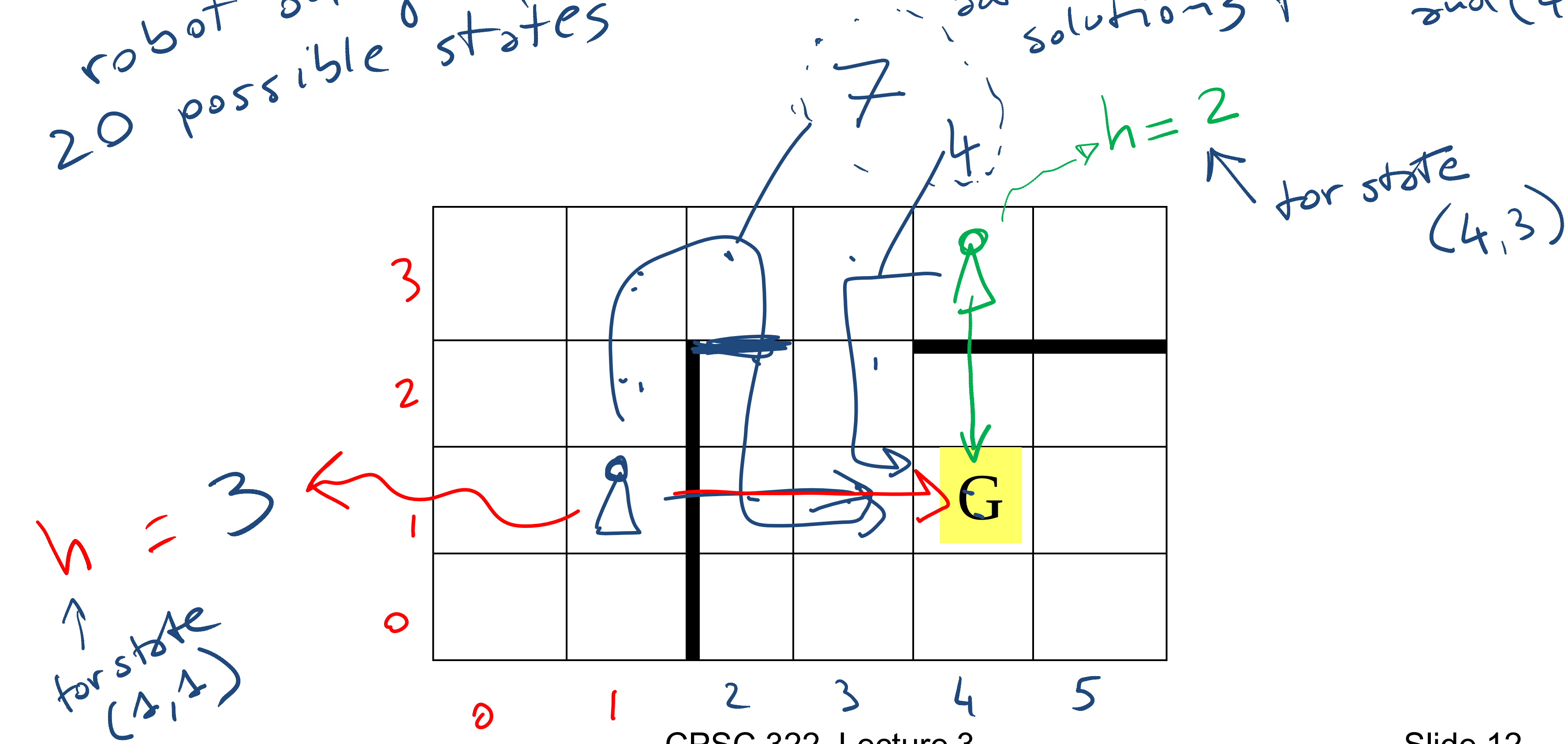


Example Admissible Heuristic Functions

If there are obstacle, the optimal solution without obstacles is an admissible heuristic

robot on grid
20 possible states

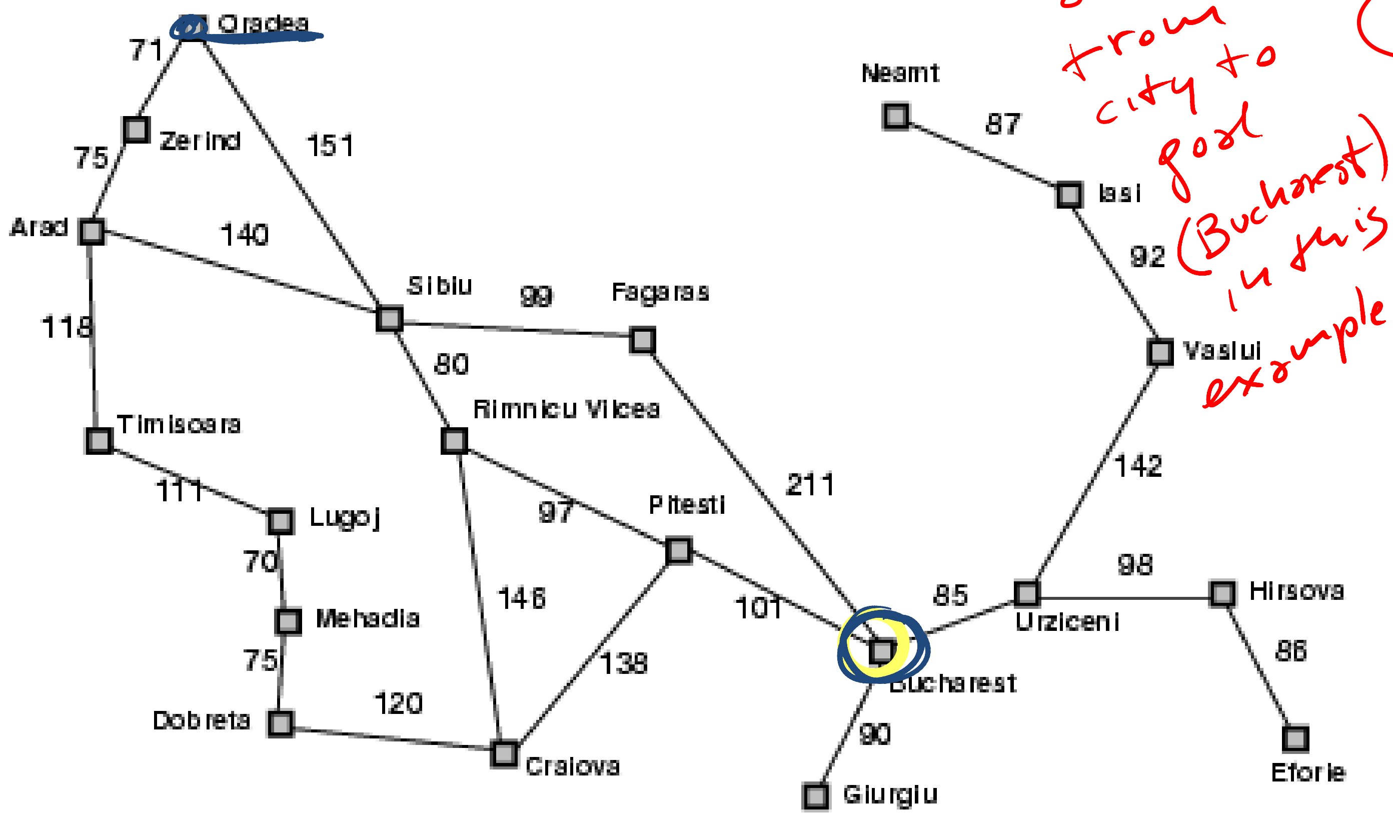
actual optimal solutions for states (1,1) and (4,3)



Example Admissible Heuristic Functions

- Similarly, If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from *n* to the closest goal as the value of *h(n)*.

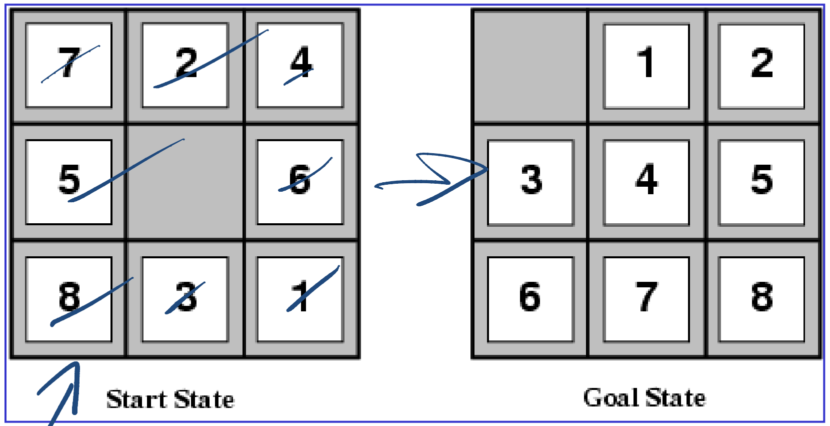
always <= than any road from city to goal (Bucharest in this example)

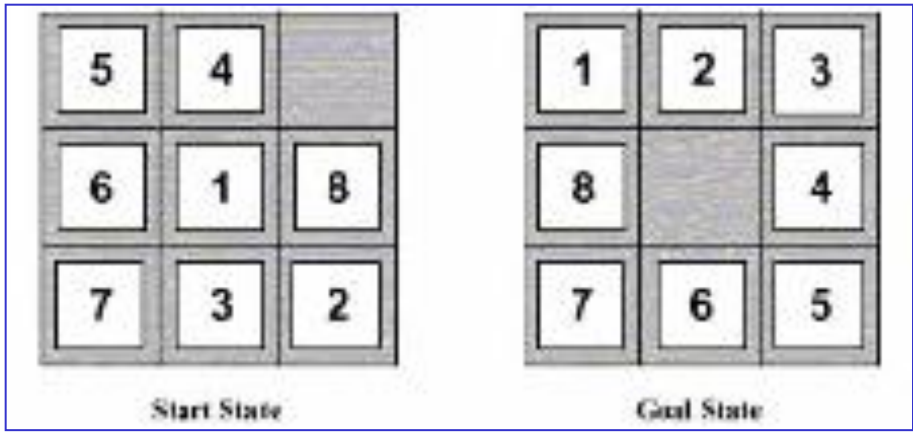


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example Heuristic Functions (1)

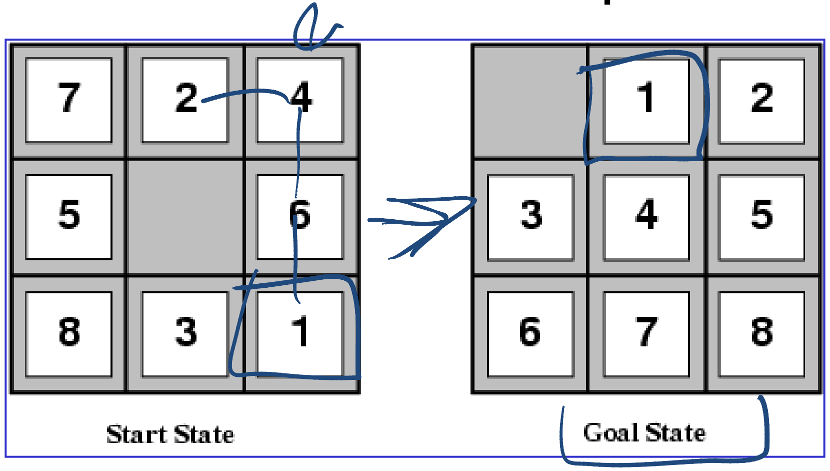
- In the 8-puzzle, we can use the number of misplaced tiles

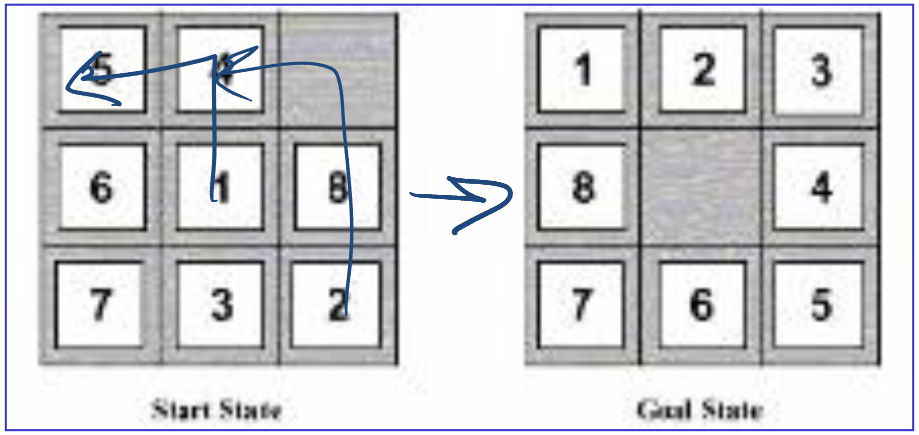




Example Heuristic Functions (2)

- Another one we can use the number of moves between each tile's current position and its position in the solution





tiles

1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
Σ 3	1	2	2	2	3	3	2	Σ	2	3						

= 18

How to Construct a Heuristic

You identify relaxed version of the problem:

- where one or more constraints have been dropped
- problem with fewer restrictions on the actions

Robot: the agent can move through walls $\leftarrow$

Driver: the agent can move straight $\leftarrow$

8puzzle: (1) tiles can move anywhere $\leftarrow$

(2) tiles can move to $\underline{\text{any adjacent square}}$ $\leftarrow$

Result: The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem (because it is always weakly less costly to solve a less constrained problem!)

How to Construct a Heuristic (cont.)

You should identify constraints which, when dropped, make the problem extremely easy to solve

- this is important because heuristics are not useful if they're as hard to solve as the original problem!

This was the case in our examples

Robot: ***allowing*** the agent to move through walls. Optimal solution to this relaxed problem is Manhattan distance

Driver: ***allowing*** the agent to move straight. Optimal solution to this relaxed problem is straight-line distance

8puzzle: (1) tiles **can move anywhere** Optimal solution to this relaxed problem is number of misplaced tiles

(2) tiles can move to **any adjacent square….**

Another approach to construct heuristics

Solution cost for a subproblem

1 2 3 4

Original Problem

	1	3
8	2	5
7	6	4

Current node

simpler!

SubProblem

	1	3
@	2	@
@	@	4

1	2	3
8		4
7	6	5

Goal node

1	2	3
@		4
@	@	@

Goal

Heuristics: Dominance

state

If $h_2(n) \geq h_1(n)$ for all n (both admissible)

then h_2 dominates h_1

h_2 is better for search (why?)

(2) >= (1)

why

8puzzle: (1) tiles can move anywhere

(2) tiles can move to any adjacent square

(Original problem: tiles can move to an adjacent square if it is empty)

Iterative deepening (not using any heuristic)

search costs for the 8-puzzle (average number of paths expanded):

→ depth of solution

d=12 IDS = 3,644,035 paths
$A^*(h_1)$ = 227 paths
$A^*(h_2)$ = 73 paths

d=24 IDS = too many paths
$A^*(h_1)$ = 39,135 paths
$A^*(h_2)$ = 1,641 paths

	h_1	h_2
If tile in correct position	0	0
If tile 1 move from correct position	1	1
otherwise	1	>1

Combining Heuristics

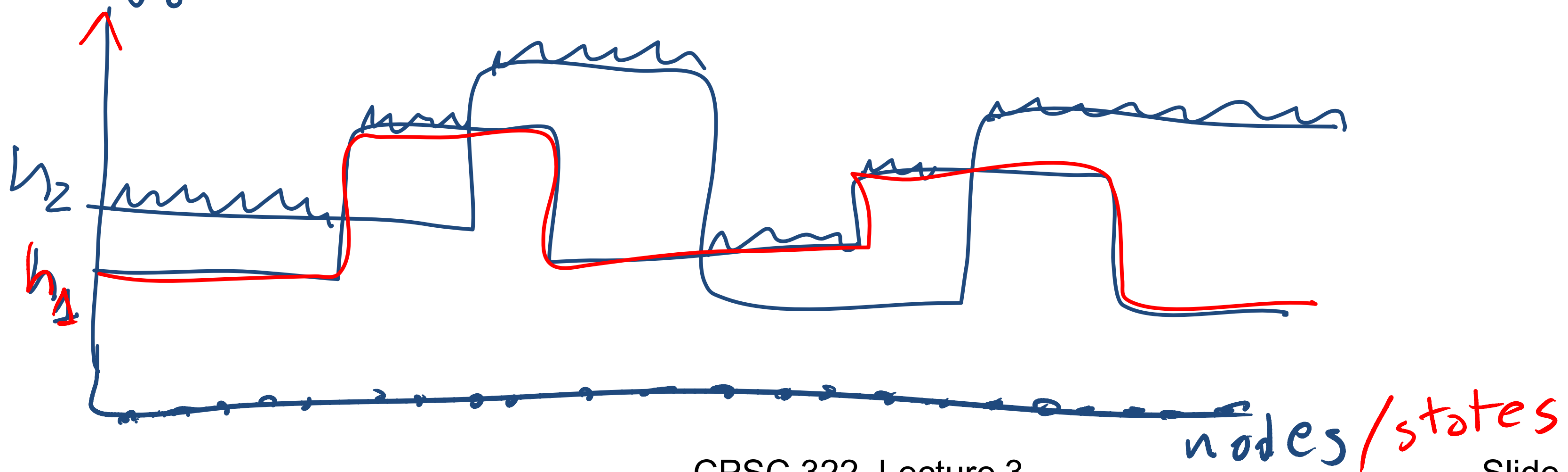
How to combine heuristics when there is no dominance?

If $h_1(n)$ is admissible and $h_2(n)$ is also admissible then

h(n)= max(h_1, h_2) …………………. is also admissible

… and dominates all its components

value of heuristic function



Combining Heuristics: Example

In 8-puzzle, solution cost for the 1,2,3,4 subproblem is substantially more accurate than Manhattan distance **in some cases**

sum of

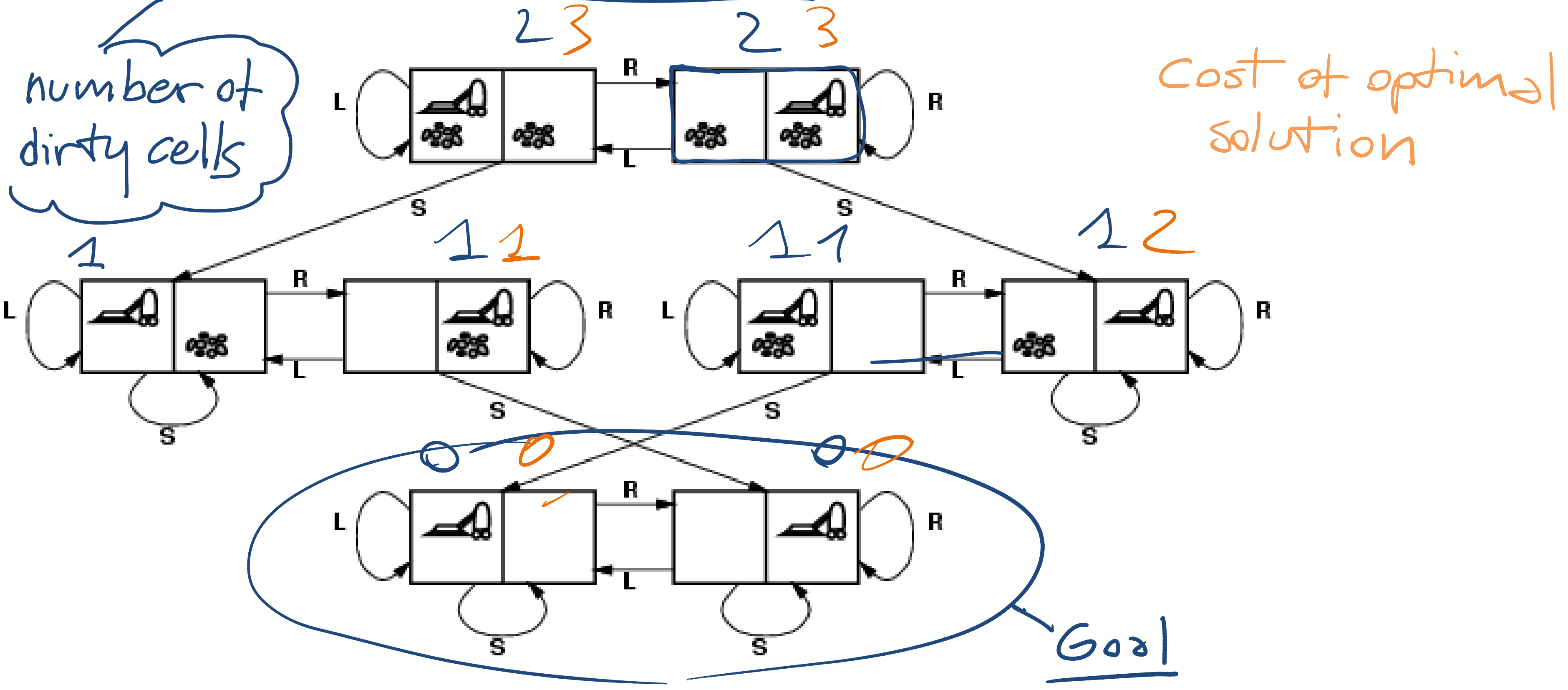
of each tile from its position in goal

So…..

max

better heuristic

Admissible heuristic for Vacuum world?



states? Where it is dirty and robot location

actions? *Left*, *Right*, *Suck*

Possible goal test? no dirt at all locations

Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Best-First Search

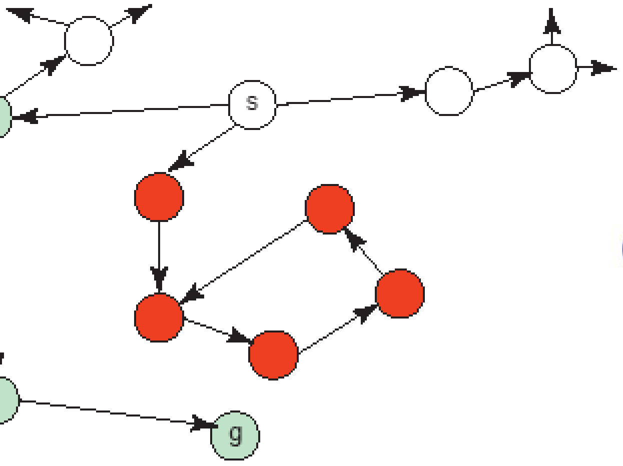
- **Idea:** select the path whose end is closest to a goal according to the heuristic function.

- **Best-First search** selects a path on the frontier with minimal *h*-value (for the end node). ←

- It treats the frontier as a priority queue ordered by *h*. (similar to ?) LCFS ⟷ by cost
- This is a greedy approach: it always takes the path which appears locally best

Analysis of Best-First Search

- Complete no: a low heuristic value can mean that a cycle gets followed forever.



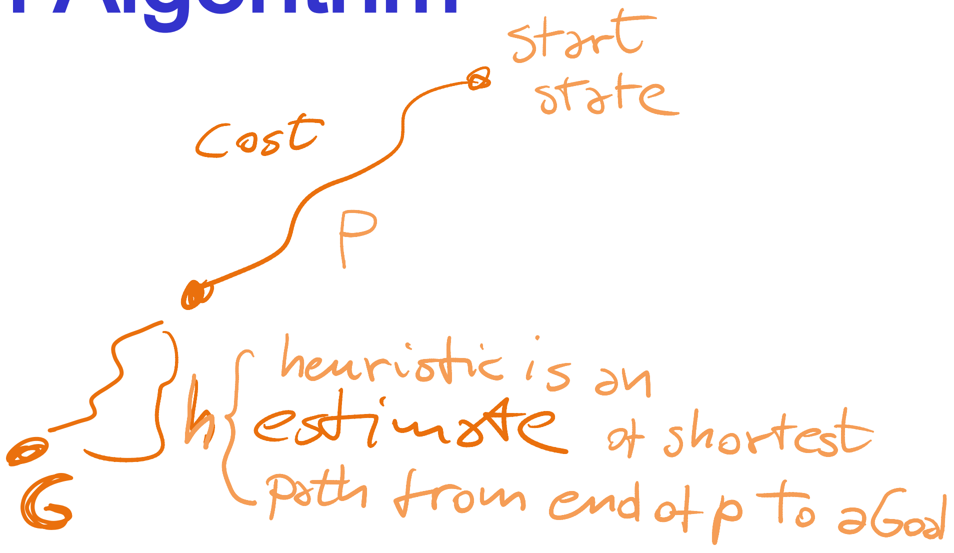
- Optimal: no (why not?)
- Time complexity is $O(b^m)$
- Space complexity is $O(b^m)$

worst case

Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

A^* Search Algorithm

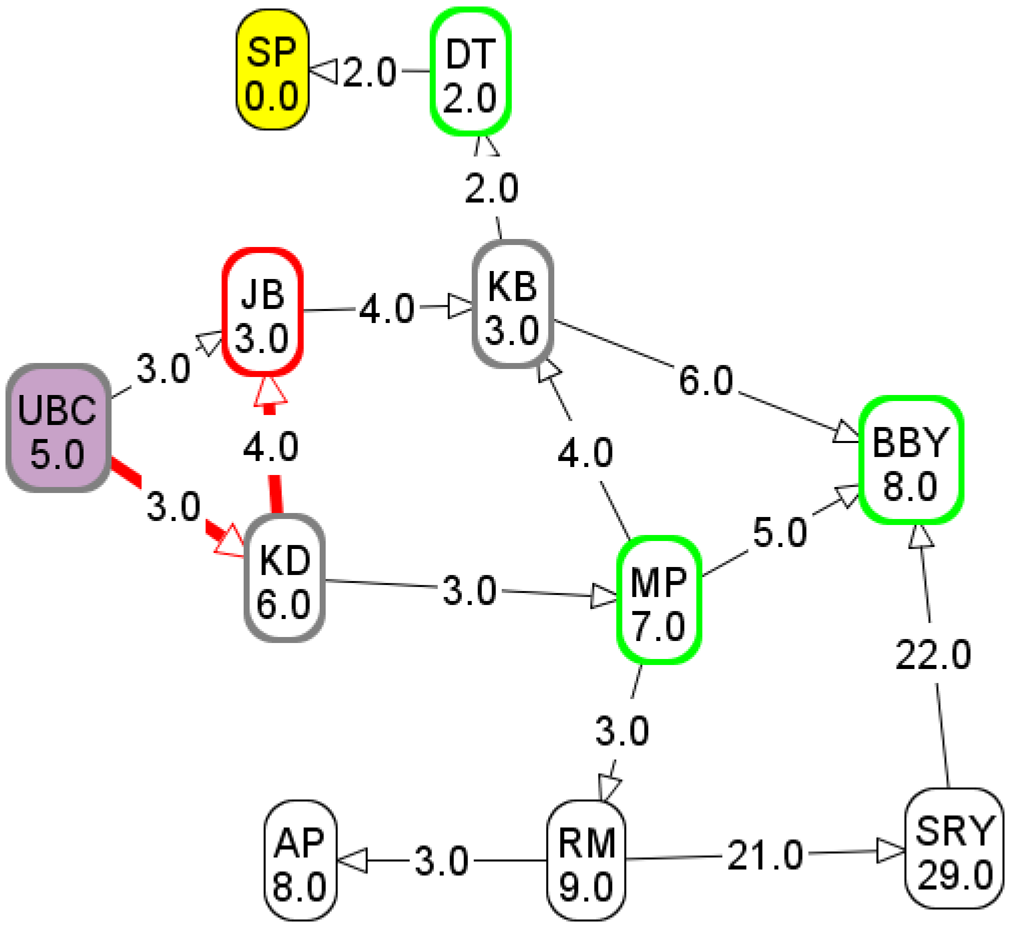

start state
cost
p
h
heuristic is an estimate of shortest path from end of p to a Goal
G

- A^* is a mix of:
  - **lowest-cost-first** and
  - **best-first search**

- A^* treats the frontier as a priority queue ordered by $\underbrace{f(p) = cost(p) + h(p)}$ is an estimate

- It always selects the node on the frontier with the lowest...... estimated ..total..........distance.

Computing f-values



F-value of ubc → kd → jb? 6 9 [✓ 10] 11

Analysis of A^*

for all states heuristic is equal to 0

$\forall s \; h(s) = 0$

Let's assume that arc costs are strictly positive.

- Time complexity is $O(b^m)$
  - the heuristic could be completely uninformative and the edge costs could all be the same, meaning that A^* does the same thing as…. BFS
- Space complexity is $O(b^m)$ like BFS, A^* maintains a frontier which grows with the size of the tree
- Completeness: $\boxed{\text{yes.}}$
- Optimality: ??

Optimality of A^*

If A^* returns a solution, that solution is guaranteed to be optimal, as long as

When

- the branching factor is finite
- arc costs are strictly positive
- *h(n)* is an underestimate of the length of the shortest path from *n* to a goal node, and is non-negative

admissible

Theorem

If A^* selects a path *p* as the solution,
p is the shortest (i.e., lowest-cost) path.

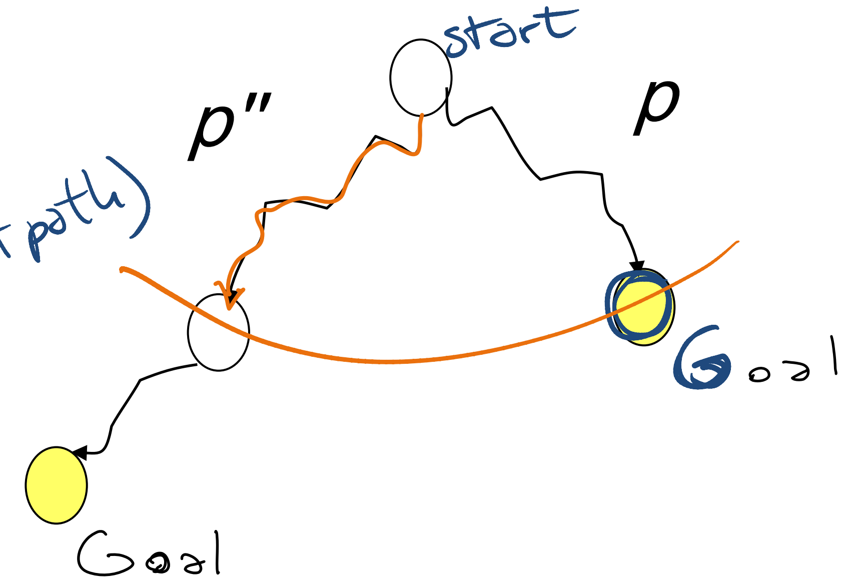
Why is *A** optimal?

- A* returns *p*
- Assume for contradiction that some other path *p′* is actually the shortest path to a goal
- Consider the moment when *p* is chosen from the frontier. Some part of path *p′* will also be on the frontier; let's call this partial path *p″*.

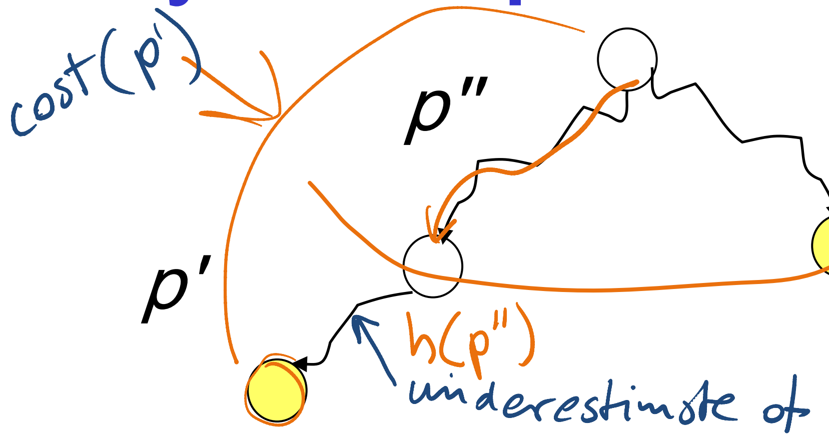
$\text{cost}(p) > \text{cost}(p')$

Think!

for any path from start to any state there is always a sub path (of that path) on the frontier



Why is A^* optimal? (cont')



$cost(p) + h(p) \leq cost(p'') + h(p'')$

- Because p was expanded before p'', $f(p) \leq f(p'')$
- Because p is a goal, $h(p) = 0$ Thus $cost(p) \leq cost(p'') + h(p'')$ ①
- Because h is admissible, $cost(p'') + h(p'') \leq cost(p')$ for any path p' to a goal that extends p'' ②

combining ① and ②

- Thus $cost(p) \leq cost(p')$ for any other path p' to a goal.

$cost(p') < cost(p)$

This contradicts our assumption that p' is the shortest path.

Optimal efficiency of A^*

- In fact, we can prove something even stronger about A^*: in a sense (given the particular heuristic that is available) **no search algorithm could do better!**

- Optimal Efficiency: Among **all optimal algorithms** that **start from the same start node** and **use the same heuristic** ***h***, A^* expands the minimal number of paths.

Sample A* applications

- **An Efficient A* Search Algorithm For Statistical** Machine Translation. 2001
- **The Generalized A* Architecture.** Journal of Artificial Intelligence Research (2007)
  - Machine Vision … Here we consider a new compositional model for finding salient curves.
- **Factored A*search for models over sequences and trees** International Conference on AI. 2003…. It starts saying… *The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate…* applied to NLP and BioInformatics

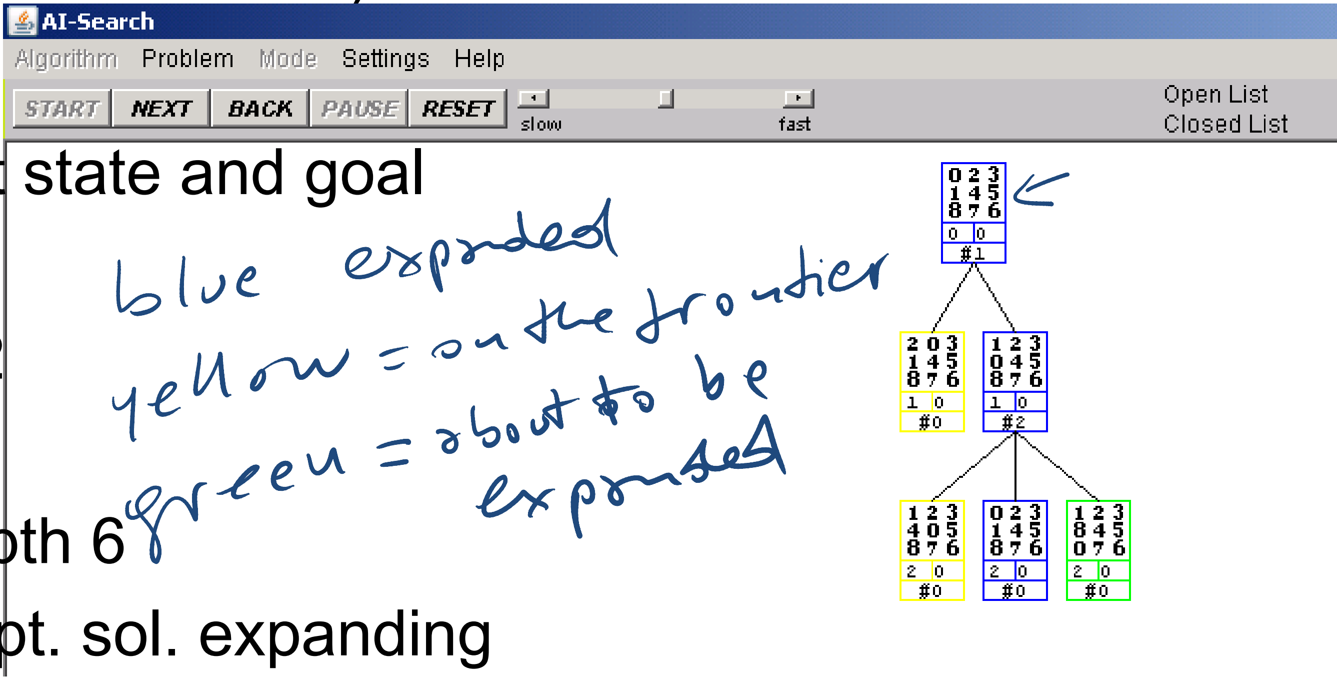
(Natural Language Processing)

DFS, BFS, A* Animation Example

- The AI-Search animation system

http://www.cs.rmit.edu.au/AI-Search/Product/

- *To examine Search strategies when they are applied to the 8puzzle*
- Compare only DFS, BFS and A* (with only the two heuristics we saw in class)



blue expanded
yellow = on the frontier
green = about to be expanded

- With default start state and goal
- DFS will find

Solution at depth 32

- BFS will find

Optimal solution depth 6

- A* will also find opt. sol. expanding much less nodes

nPuzzles are not always solvable

Half of the starting positions for the *n*-puzzle are impossible to resolve (for more info on 8puzzle) http://www.isle.org/~sbay/ics171/project/unsolvable

- So experiment with the AI-Search animation system with the default configurations.
- If you want to try new ones keep in mind that you may pick unsolvable problems

Learning Goals for today's class (part 1)

- Construct admissible heuristics for appropriate problems.
- Verify Heuristic Dominance.
- Combine admissible heuristics

- Define/read/write/trace/debug different search algorithms
  - With / Without cost
  - Informed / Uninformed

- Formally prove A* optimality

Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

mid 30 mins
BREAK
projector OFF!

Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Branch-and-Bound Search

- What is the biggest advantage of A*?

uses heuristics

- What is the biggest problem with A*?

space

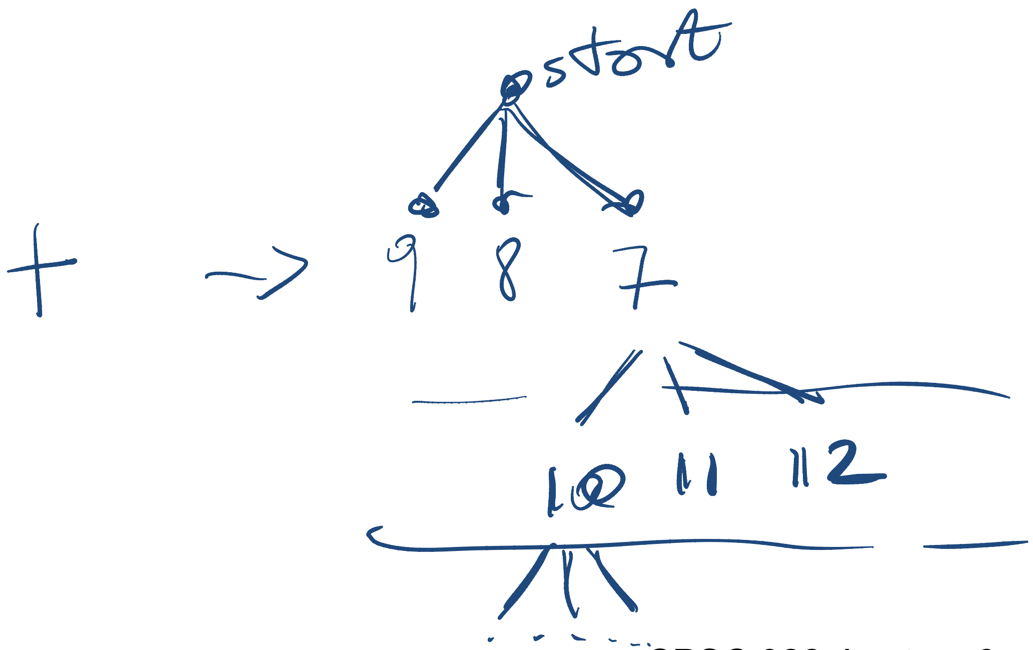
- Possible Solution:

DFS + h

Branch-and-Bound Search Algorithm

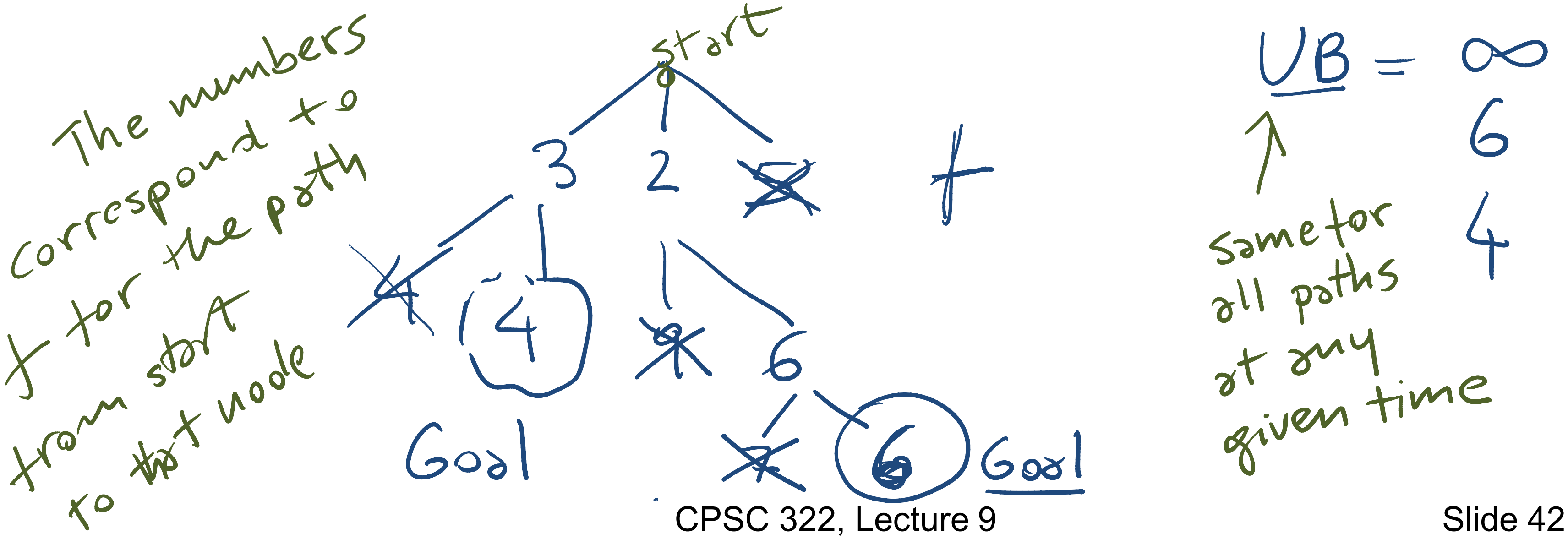
- Follow exactly the same search path as **depth-first search**
  - treat the frontier as a stack: expand the most-recently added path first
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic

we can use
$f = c + h$



Branch-and-Bound Search Algorithm

- Keep track of a lower bound and upper bound on solution cost at each path
  - lower bound: $LB(p) = \boxed{f(p)} = cost(p) + h(p)$
  - upper bound: $UB =$ cost of the best solution found so far.
    - ✓ if no solution has been found yet, set the upper bound to ∞.
- When a path p is selected for expansion:
  - if $LB(p) \geq UB$, remove p from frontier without expanding it (pruning)
  - else expand p, adding all of its neighbors to the frontier



The numbers correspond to f for the path from start to that node
start
3
2
~~5~~
f
~~4~~
4
Goal
~~9~~
6
~~7~~
6
Goal
UB = ∞
6
4
Same for all paths at any given time

Branch-and-Bound Analysis

- Complete ?

yes | no | It depends

- Optimal ?

yes | no | It depends

- Space complexity?

$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$

- Time complexity?

Lecture Overview

- **Recap Uninformed Cost**
- **Heuristic Search**
  - **Best-First Search**
  - **A* and its Optimality**
- **Advanced Methods**
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Other A^* Enhancements

The main problem with A^* is that it uses exponential space. Branch and bound was one way around this problem. Are there others?

- Iterative Deepening A^* $\swarrow$ IDA^*
  ……..
- Memory-bounded A^*

(Heuristic) Iterative Deepening – IDA*

B & B can still get stuck in infinite (extremely long) paths

- Search depth-first, but to a fixed depth/bound
  - if you don't find a solution, increase the depth tolerance and try again

    then update with the lowest f that passed the previous bound
  - depth is measured in.....f...........

    start node $f(\text{start}) = h(\text{start})$
- Counter-intuitively, the asymptotic complexity is not changed, even though we visit paths multiple times (*go back to slides on uninformed IDS*)

$$\left(\frac{b}{b-1}\right)^2$$

Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal:

yes no It depends

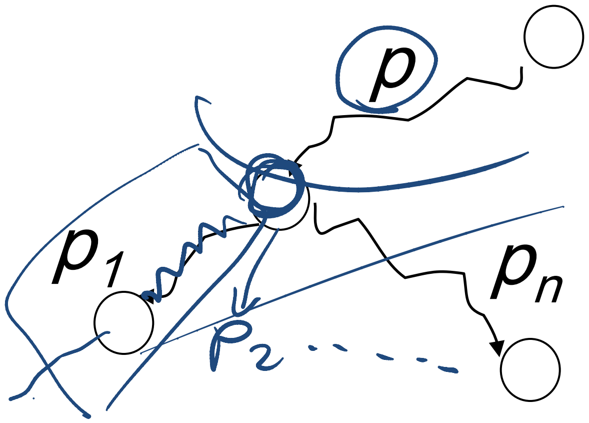
- Time complexity:

- Space complexity:

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

Memory-bounded A^*

- Iterative deepening A* and B & B use a tiny amount of memory
- **what if we've got more memory to use?**
- keep as much of the fringe in memory as we can
- if we have to delete something:
  - delete the worst paths (with ....highest.......f.........)
  - ``back them up'' to a common ancestor



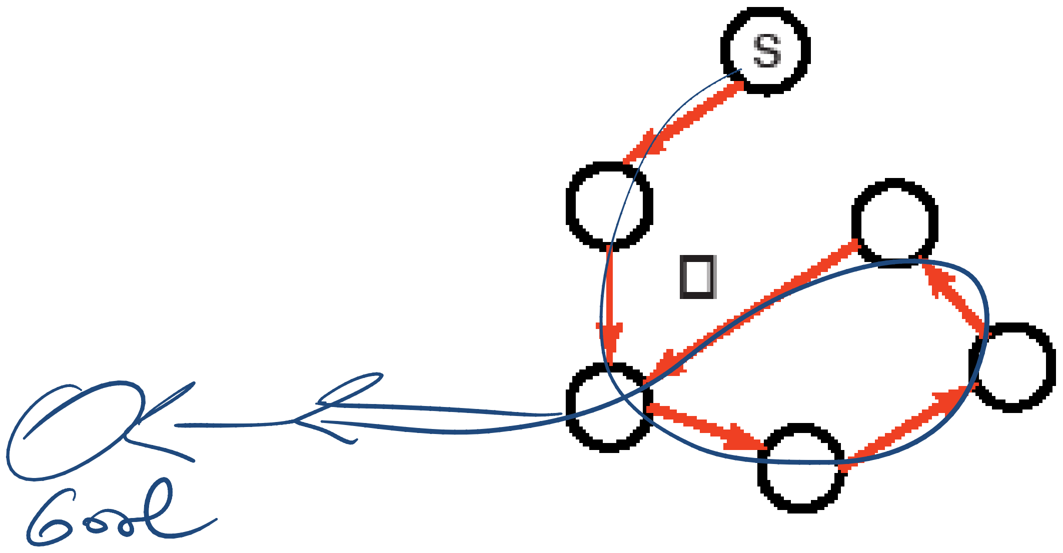
$$h(p) = \max\left(\min \left[cost(p_i) - cost(p)\right] + h(p_i)\right)$$

old h(p)

Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Cycle Checking



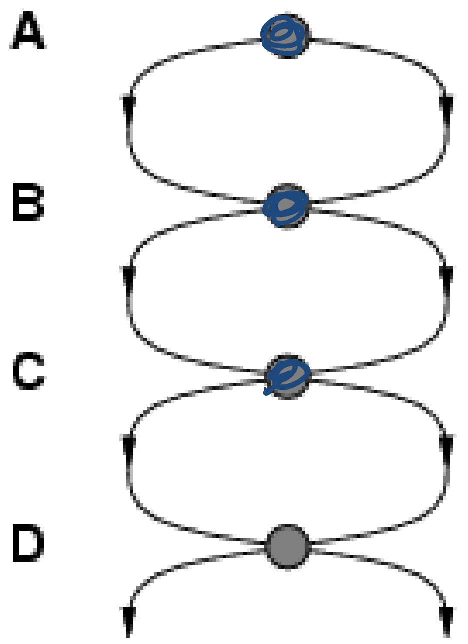
You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

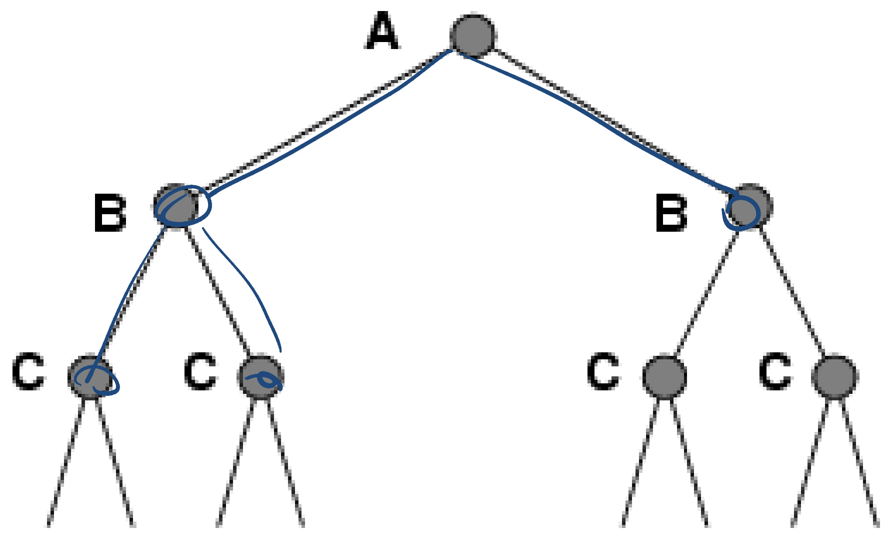
- The time is ...linear..................... in path length.

$\langle n_0, n_1, n_2 \ldots n_k \rangle$
n_2
n_i
n_j

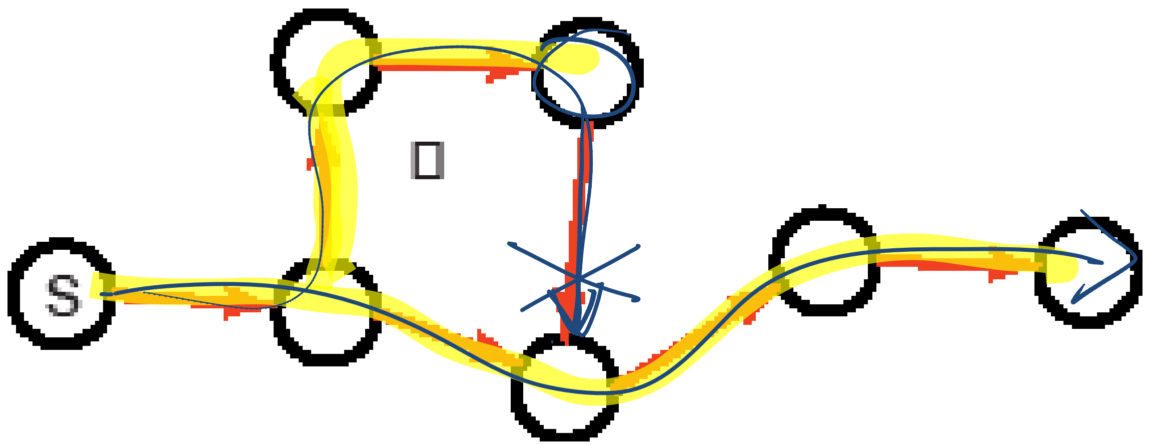
Repeated States / Multiple Paths

Failure to detect repeated states can turn a linear problem into an exponential one!





Multiple-Path Pruning

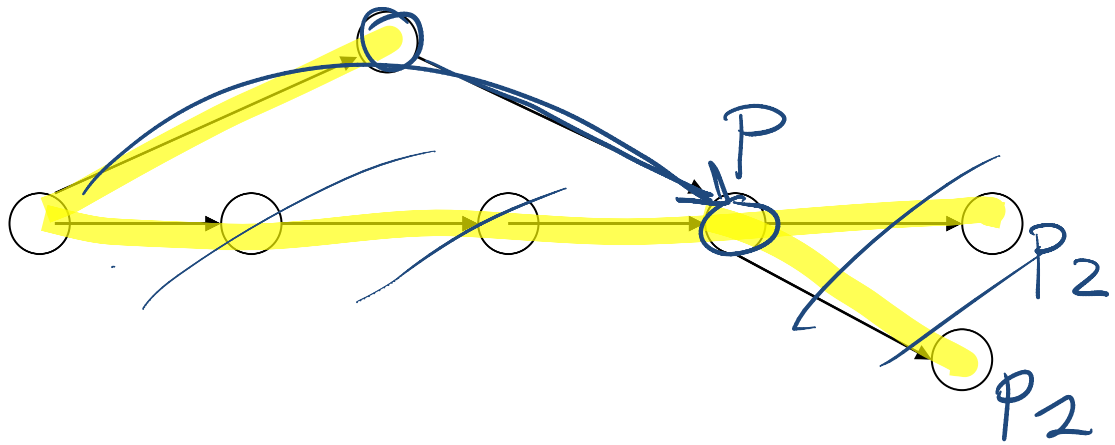


- You can prune a path to node *n* that you have already found a path to
- (if the new path is longer – more costly).

Multiple-Path Pruning & Optimal Solutions

Problem: what if a subsequent path to *n* is shorter than the first path to *n* ?

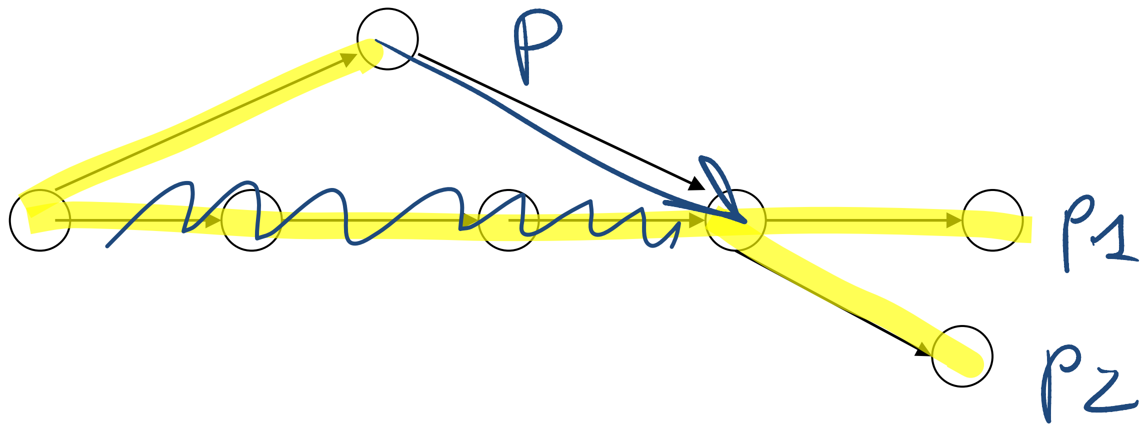
- You can remove all paths from the frontier that use the longer path. (as these can't be optimal)



Multiple-Path Pruning & Optimal Solutions

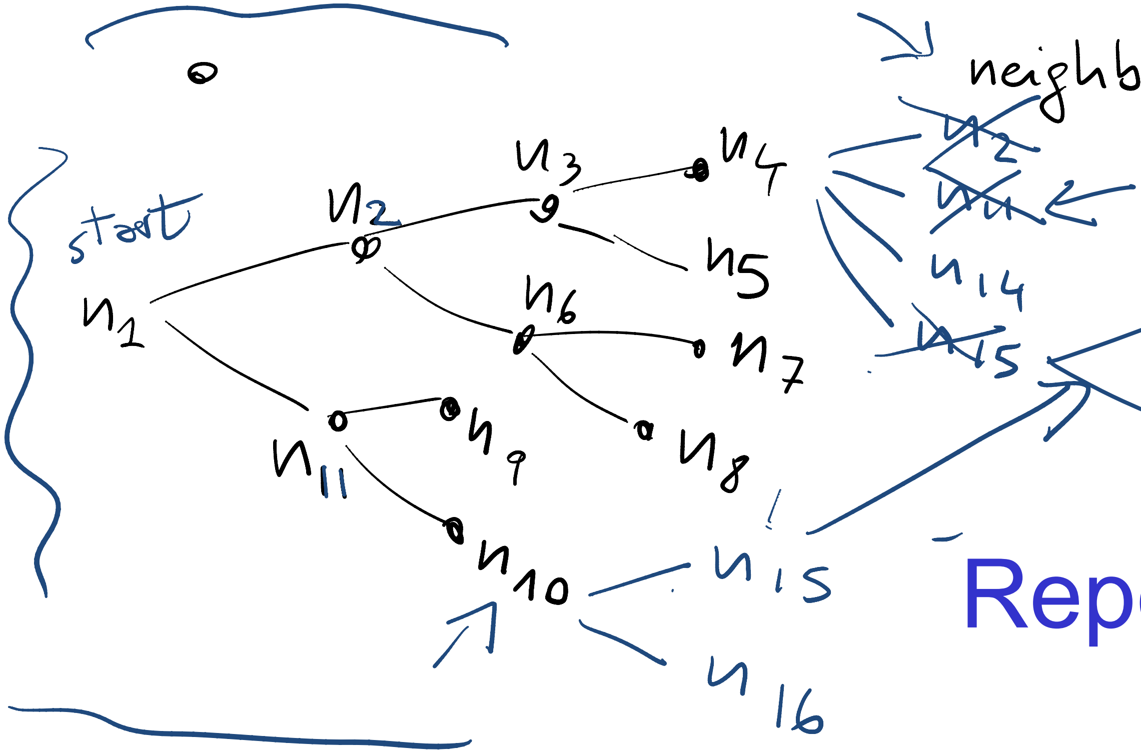
Problem: what if a subsequent path to *n* is shorter than the first path to *n* ?

- You can change the initial segment of the paths on the frontier to use the shorter path.



Example

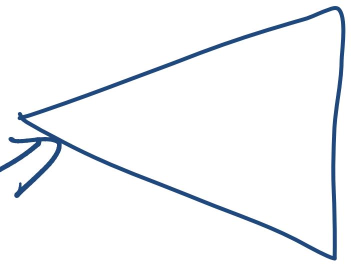
Pruning Cycles



neighbors of $n_4 = \{n_2, n_{11}, n_{14}, n_{15}\}$

Repeated States

neighbors of $n_{10} = \{n_{15}, n_{16}\}$



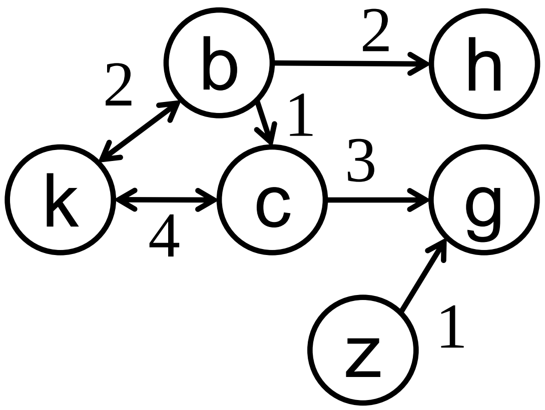
Lecture Overview

- Recap Uninformed Cost
- Heuristic Search
  - Best-First Search
  - A* and its Optimality
- Advanced Methods
  - Branch & Bound
  - A^* tricks
  - Pruning Cycles and Repeated States
  - Dynamic Programming

Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The actual distance of the shortest path from node n to a goal g
  - This is the perfect search heuristic h

- dist(g) = 0
- dist(z) = 1
- dist(c) = 3
- dist(b) = 4 6 7 ∞
- dist(k) = ? 6 7 ∞
- dist(h) = ?



- How could we implement that?

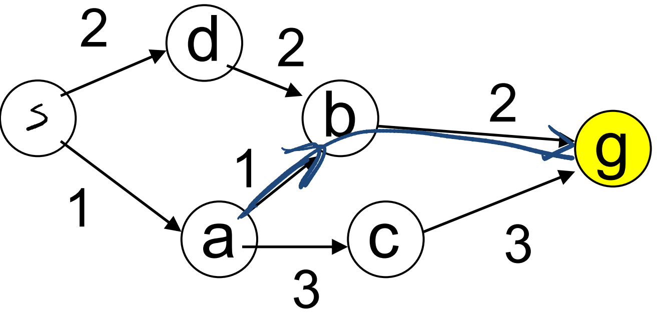
Dynamic Programming

This can be built backwards from the goal:

This can be built backwards from the goal:

$$\underline{dist(n)} = \begin{cases} \underline{0} & \text{if } \underline{is\_goal(n)}, \\ \min_{\langle \underline{n}, m \rangle \in A} (\underline{cost(n,m)} + \boxed{dist(m)}) & otherwise \end{cases}$$

all the neighbors m



	dist(n)
g	0

$dist(b) = \min[(2+0)] = 2$

$dist(c) = \min[(3+0)] = 3$

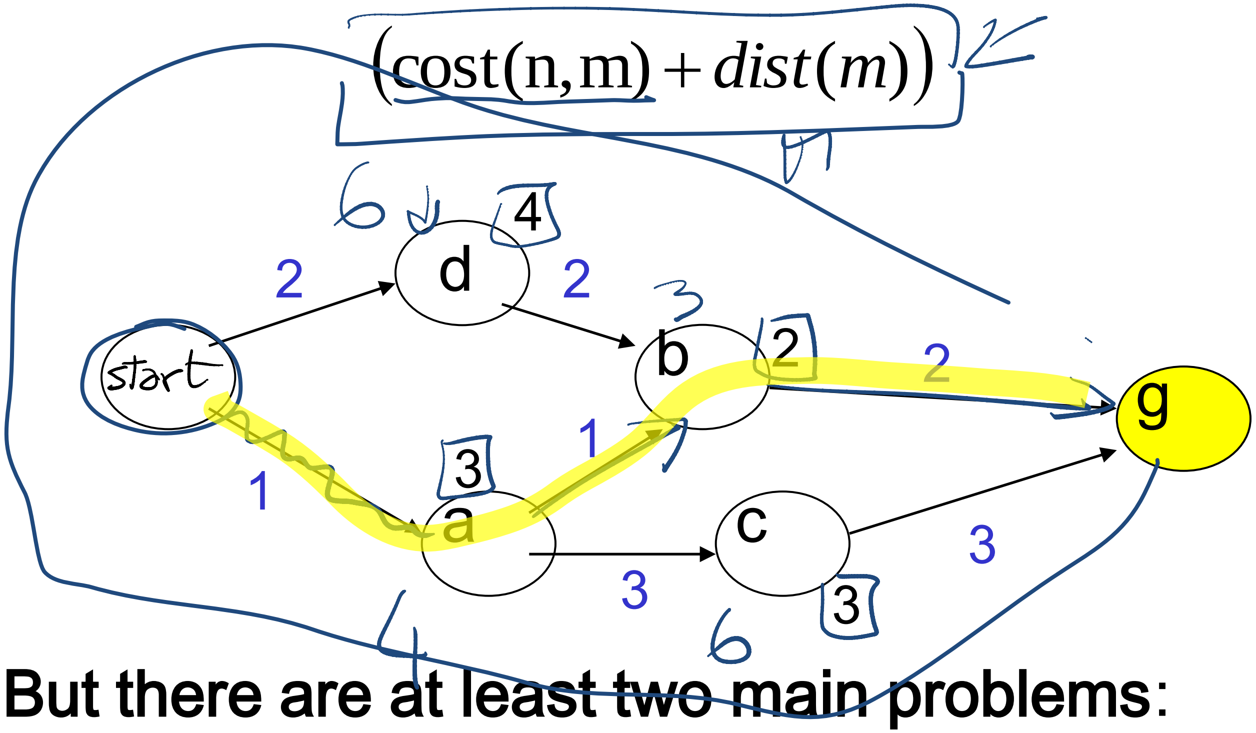
$dist(a) = \min[(3+3), (1+2)] = 3$

Dynamic Programming

This can be used locally to determine what to do.

From each node *n* go to its neighbor which minimizes

$$\boxed{\left(\underline{\text{cost}(n,m)} + dist(m)\right)}$$



But there are at least two main problems:

- You need enough space to store the graph.
- The *dist* function needs to be recomputed for each goal

Learning Goals for today's class

- Define/read/write/trace/debug different search algorithms
    - With / Without cost
    - Informed / Uninformed

- Pruning cycles and Repeated States

- Implement Dynamic Programming approach

Recap Search

	Selection	Complete	Optimal	Time	Space
DFS	LIFO	N	N	$O(b^m)$	$O(mb)$
BFS	FIFO	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	LIFO	Y	Y	$\boxed{O(b^m)}$	$O(mb)$
LCFS	min cost	Y	Y	$O(b^m)$	$O(b^m)$
BFS	min h	N	N	$O(b^m)$	$O(b^m)$
A*	min f=c+h	Y	Y	$O(b^m)$	$O(b^m)$
B&B	LIFO + pruning f	N	Y	$O(b^m)$	$O(mb)$
IDA*	LIFO +	Y	Y	$O(b^m)$	$O(mb)$
MBA*	min f	N	Y	$O(b^m)$	$O(b^m)$

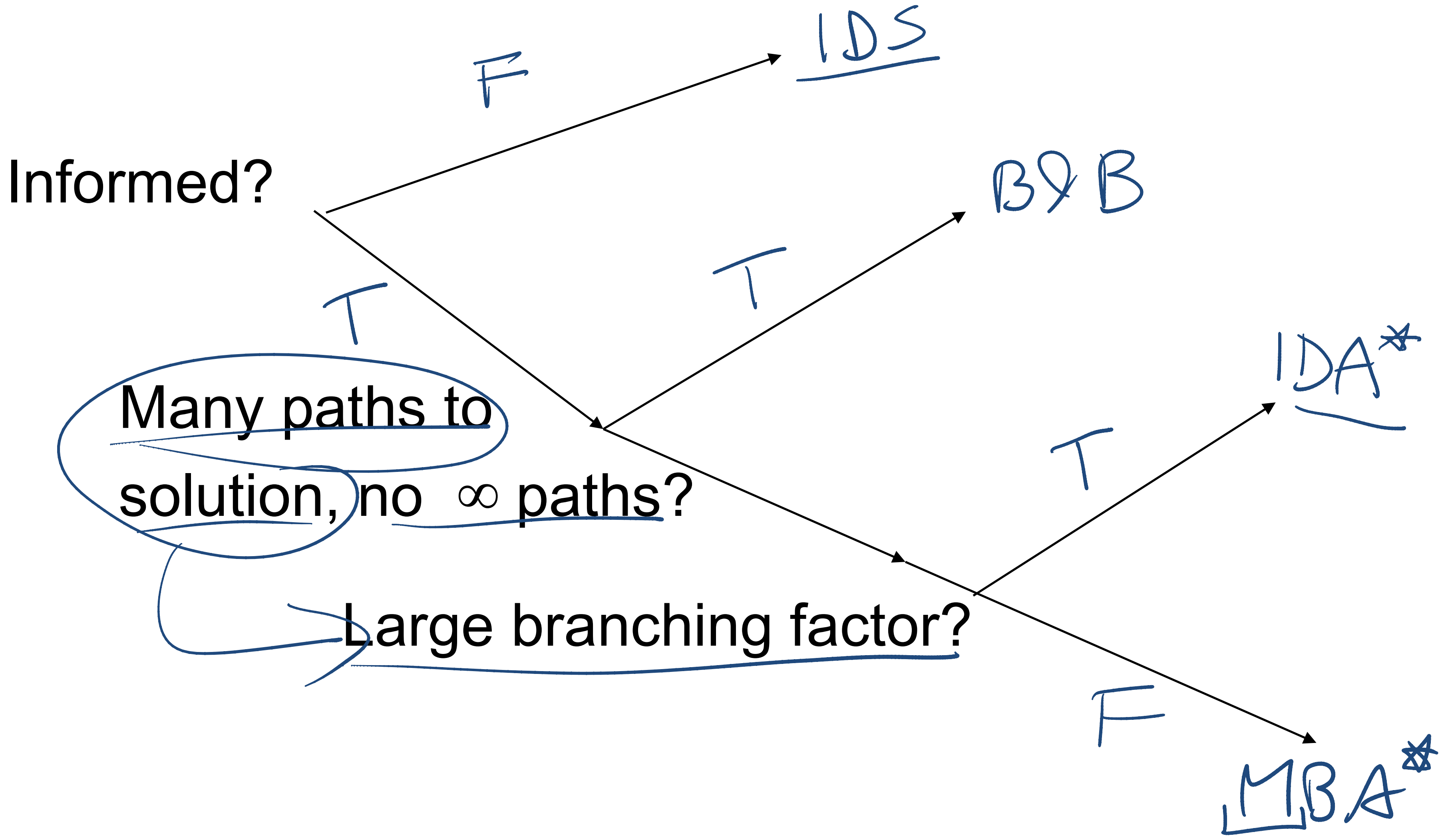
Recap Search (some qualifications)

	Complete	Optimal	Time	Space
DFS	N	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y	Y ? $c \geq 0$	$O(b^m)$	$O(b^m)$
BFS	N	N	$O(b^m)$	$O(b^m)$
A*	Y	Y ? h admissible	$O(b^m)$	$O(b^m)$
B&B	N	Y ?	$O(b^m)$	$O(mb)$
IDA*	Y	Y	$O(b^m)$	$O(mb)$
MBA*	N	Y	$O(b^m)$	$O(b^m)$

Search in Practice

	Complete	Optimal	Time	Space
DFS	N	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y	Y	$O(b^m)$	$O(b^m)$
BFS	N	N	$O(b^m)$	$O(b^m)$
A*	Y	Y	$O(b^m)$	$O(b^m)$
B&B	N	Y	$O(b^m)$	$O(mb)$
IDA*	Y	Y	$O(b^m)$	$O(mb)$
MBA*	N	Y	$O(b^m)$	$O(b^m)$
BDS	Y	Y	$O(b^{m/2})$	$O(b^{m/2})$

Search in Practice (cont')



For next class

Posted on WebCT

- Assignment1 (due this Thurs!)

If you are confused about basic search algorithm, different search strategies….. Check learning goals at the end of lectures. Please come to office hours

- **Work on Graph Searching Practice Ex:**
  - Exercise 3.C: heuristic search
  - Exercise 3.D: search
  - Exercise 3.E: branch and bound search

- **Read textbook:**
  - **4.1- 4.6 we start CSPs**

Branch-and-Bound Analysis

- Completeness: no, for the same reasons that DFS isn't complete
  - however, for many problems of interest there are no infinite paths and no cycles
  - hence, for many problems B&B is complete
- Time complexity: $O(b^m)$
- Space complexity:.....$O(bm)$
  - Branch & Bound has the same space complexity as....DFS
  - this is a big improvement over ...A*.........!
- Optimality: ..yes......

Memory-bounded A*

- Iterative deepening A* and B & B use little memory
- What if we have some more memory (but not enough for regular A*)?
  - Do A* and keep as much of the frontier in memory as possible
  - When running out of memory
    - ✓ delete worst path (highest f value) from frontier
    - ✓ Back the path up to a common ancestor
    - ✓ Subtree gets regenerated only when all other paths have been shown to be worse than the “forgotten” path
- Complete and optimal if solution is at depth manageable for available memory

Memory-bounded A^*

Details of the algorithm are beyond the scope of this course but

- It is complete if the solution is at a depth manageable by the available memory
- Optimal under the same conditions
    - Otherwise it returns the next reachable solution
- Often used in practice for, is considered one of the best algorithms for finding optimal solutions
- It can be bogged down by having to switch back and forth among a set of candidate solution paths, of which only a few fit in memory