

Heuristic Search

Computer Science cpsc322, Lecture 7
(Textbook Chpt 3.6)

January, 18, 2010



Course Announcements

Posted on WebCT

- Marks for assignment0

Will be Posted on WebCT tonight

- Second Practice Exercise (uninformed Search)

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

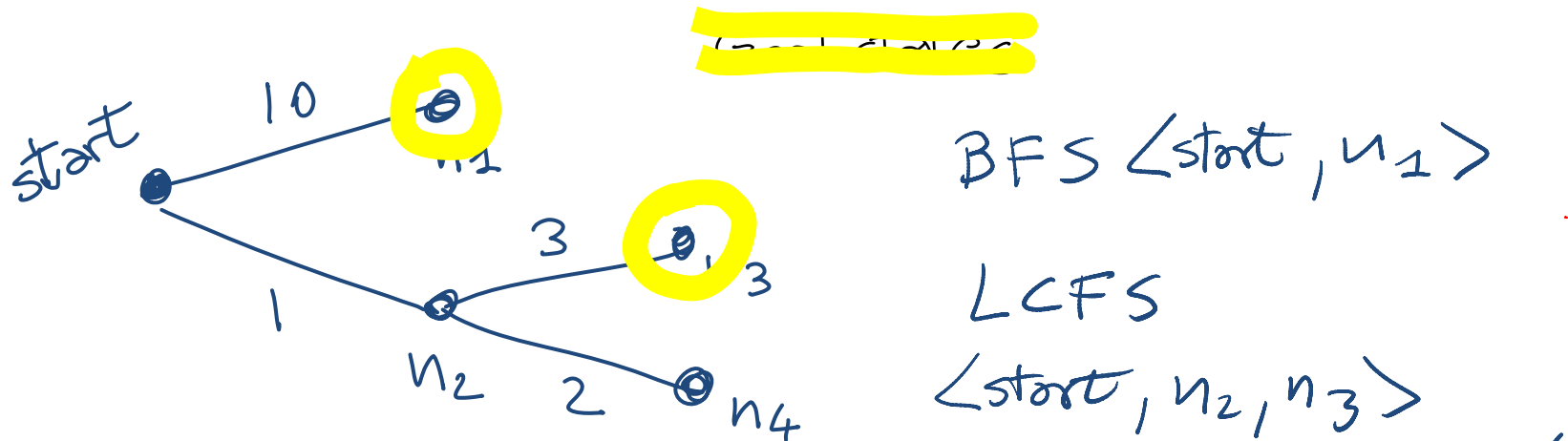
Assignment1 will be posted on Wed

Lecture Overview

- **Recap**
 - **Search with Costs**
 - **Summary Uninformed Search**
- Heuristic Search

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.



- **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 <i>Y if no cycles and finite search space</i>	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS	Y	Y	$O(b^m)$	<u>$O(mb)$</u>
LCFS	Y Costs > 0	Y Costs <u>>=0</u>	$O(b^m)$	$O(b^m)$

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

$(\langle n0, n2, \text{n3} \rangle 12), (\langle n0, \text{n3} \rangle 8), (\langle n0, n1, \text{n4} \rangle 13)$

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

Lecture Overview

- Recap
 - Search with Costs
 - Summary Uninformed Search
- Heuristic Search

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.

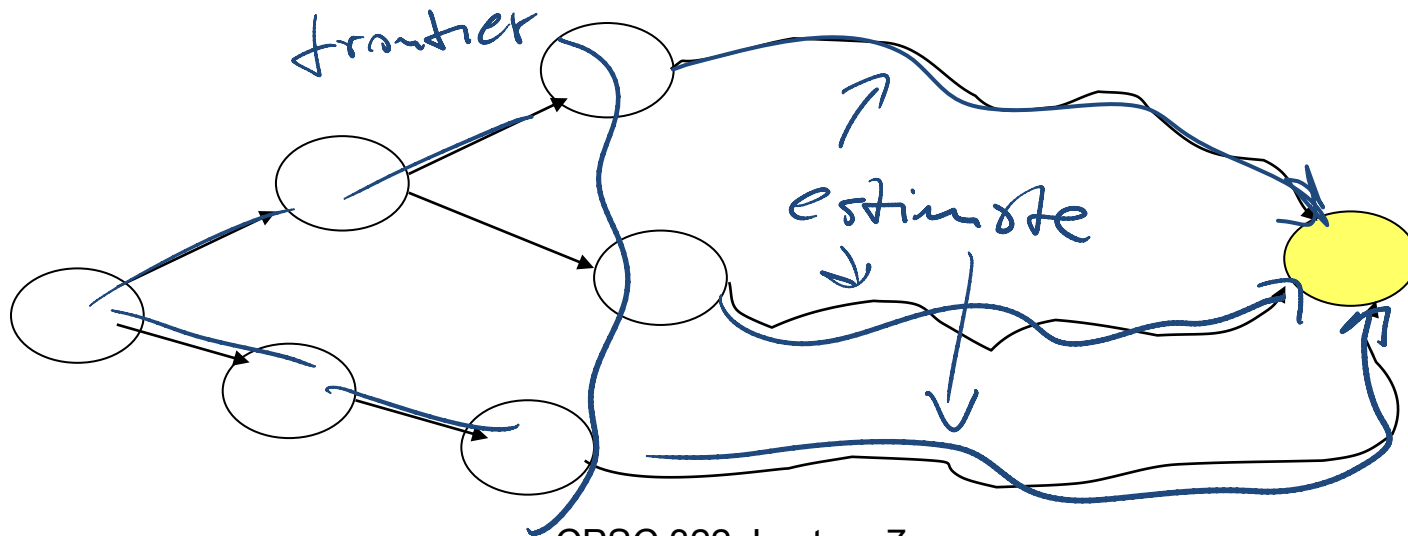
This is called a *heuristic*

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: $h(\langle n_0, \dots, n_k \rangle) = h(n_k)$
- For now think of $h(n)$ as only using 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. ←

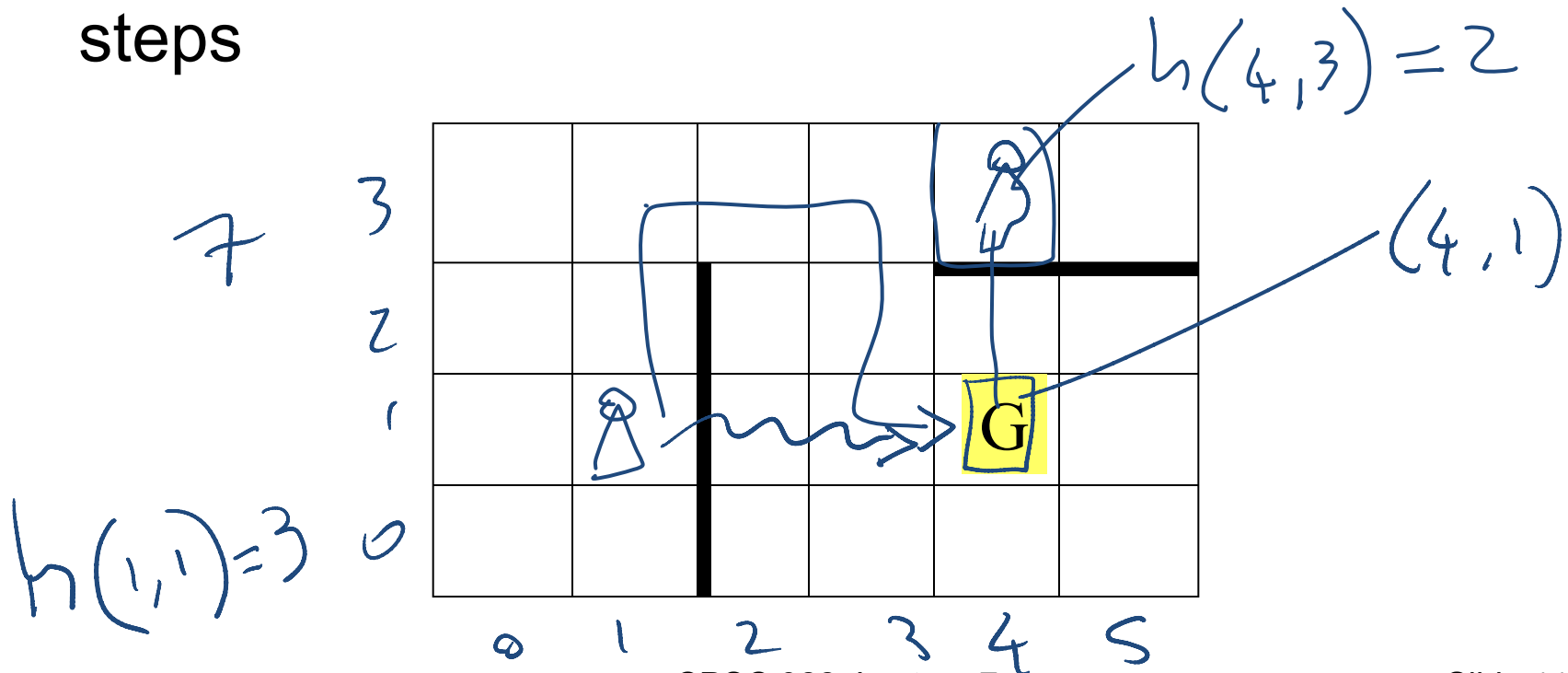
- There is never a path from n to a goal that has path length less than $h(n)$.
- another way of saying this: $h(n)$ is a **lower bound** on the cost of getting from n to the nearest goal.



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)

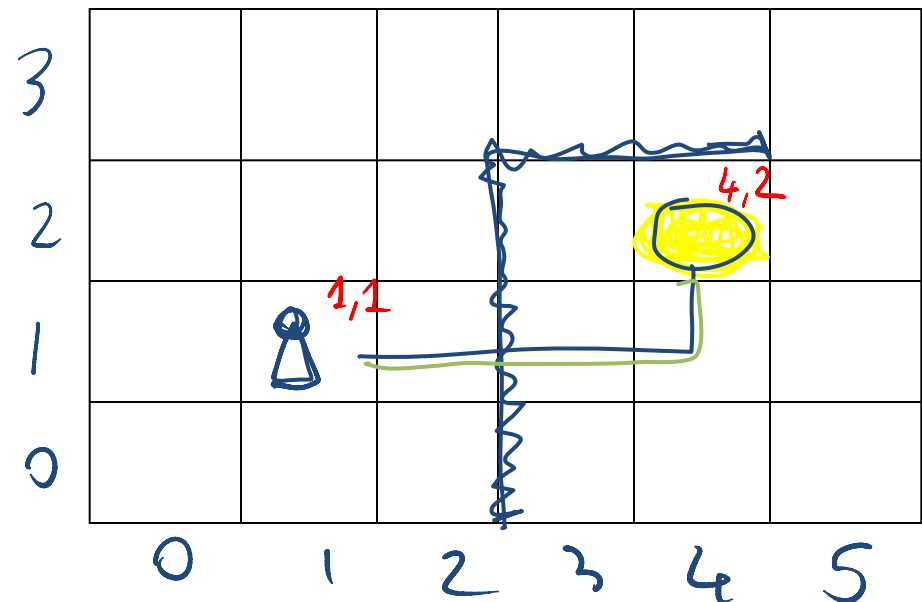
Final cost (quality of the solution) is the number of steps



Example Admissible Heuristic Functions

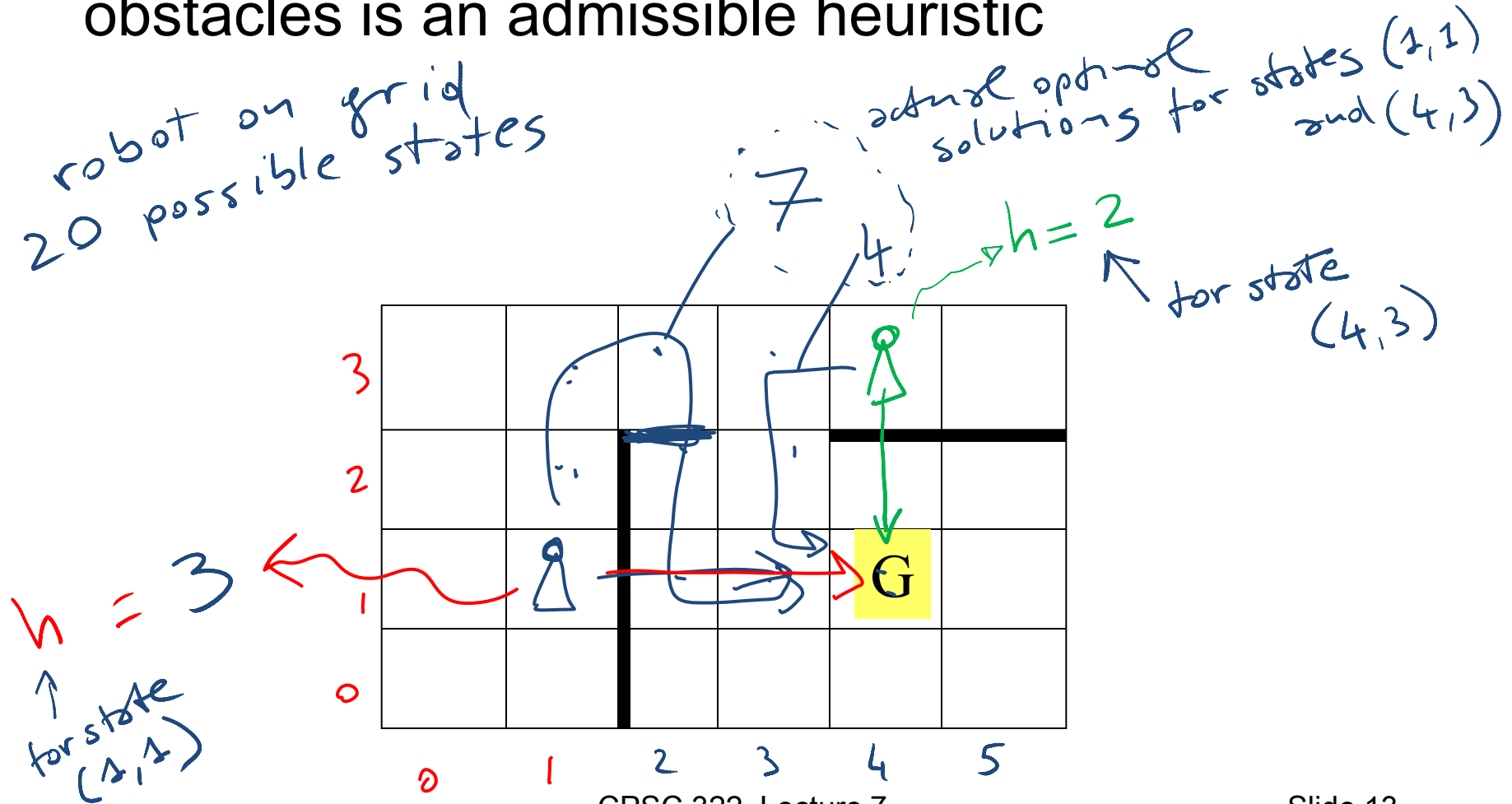
- If no obstacles, cost of optimal solution is...

$$\begin{array}{l} \text{Goal state} \\ X_G \quad Y_G \\ \text{Current state} \\ X_C \quad Y_C \\ \left\{ \begin{array}{l} |X_G - X_C| + \\ |Y_G - Y_C| \end{array} \right\} \\ \text{Manhattan distance} \end{array} \quad \begin{array}{l} |4 - 1| + \\ |2 - 1| \\ = 4 \end{array}$$



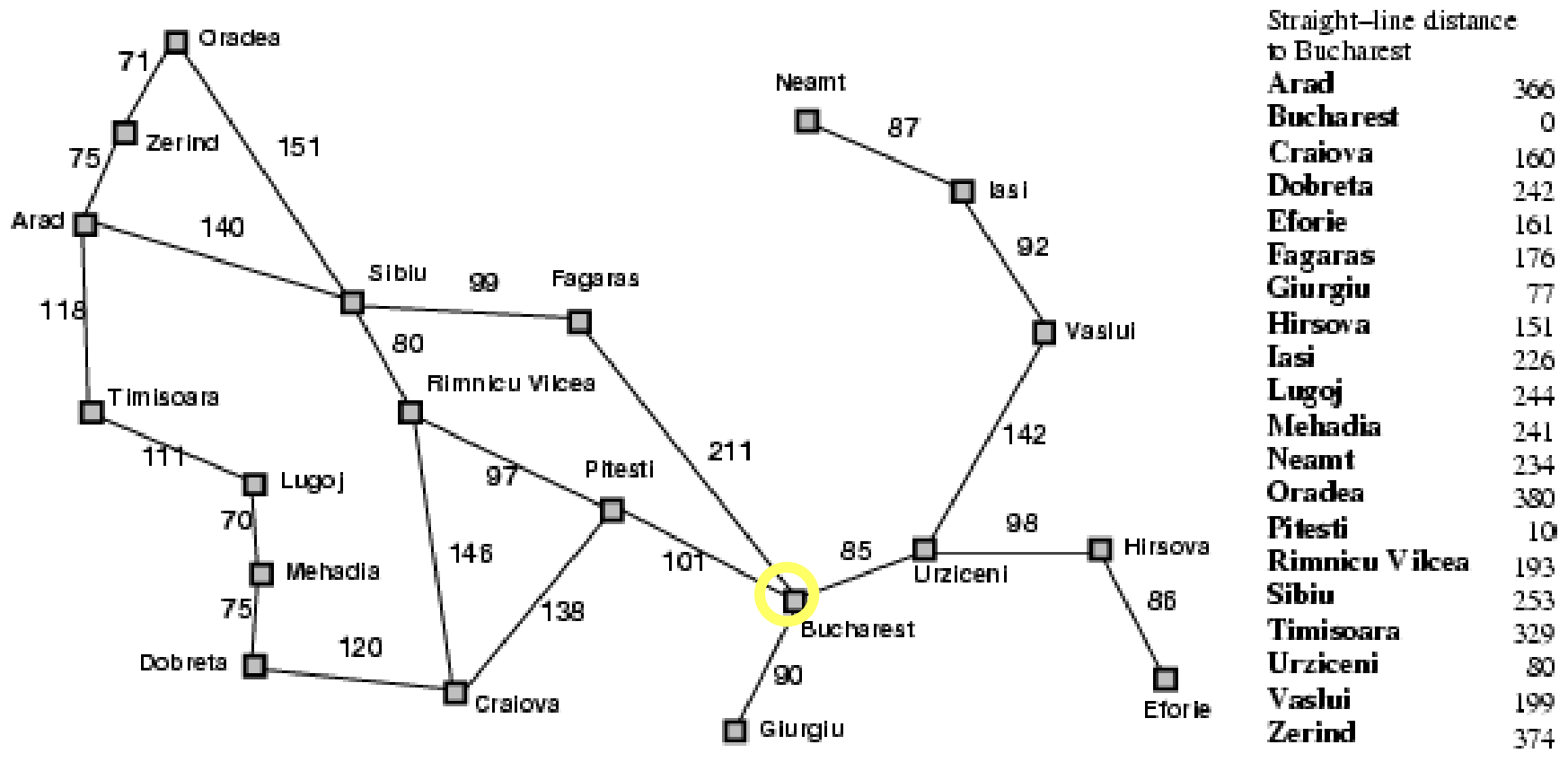
Example Admissible Heuristic Functions

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



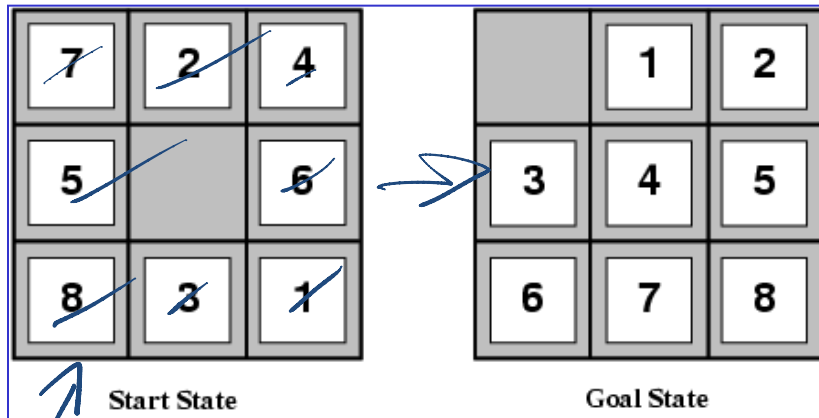
Example Admissible Heuristic Functions

- 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 $h(n)$.

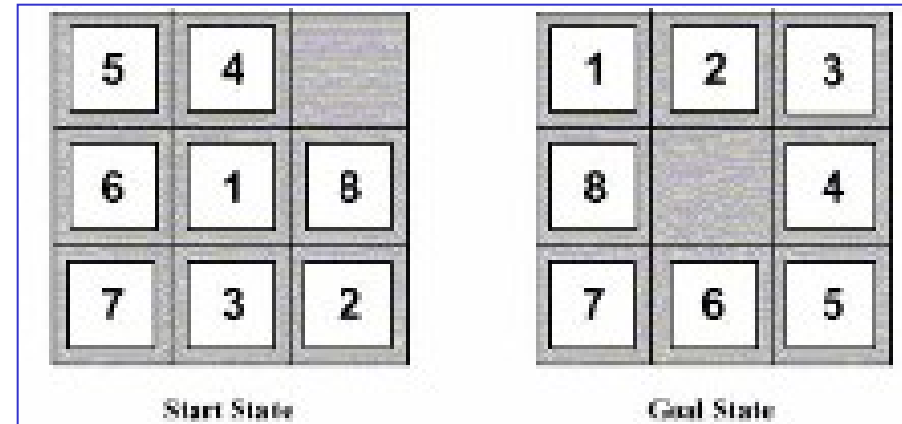


Example Heuristic Functions (1)

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



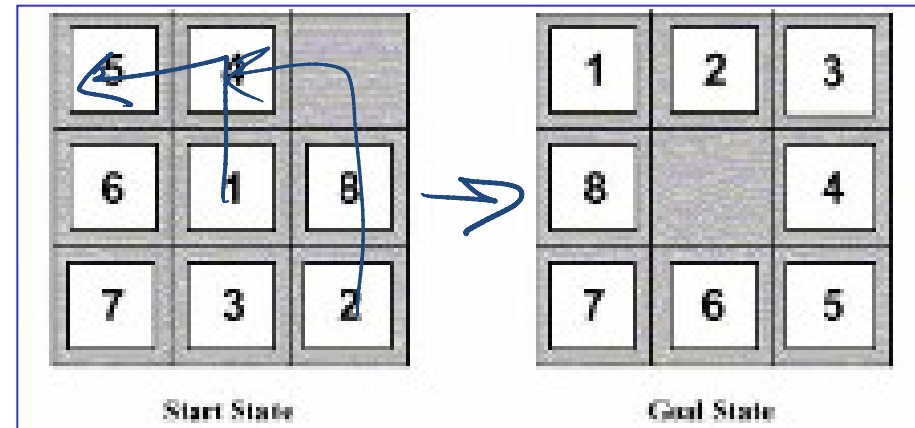
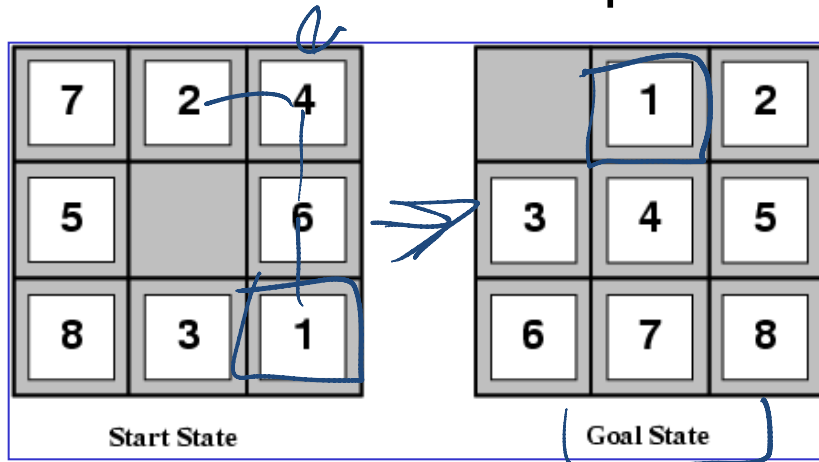
8



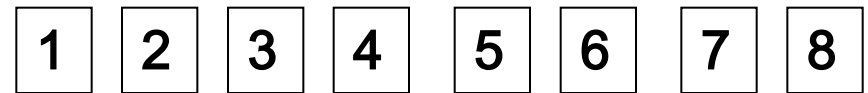
7

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



3 1 2 2 2 3 3 2 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** 

Driver: the agent **can move straight** 

8puzzle: (1) tiles **can move anywhere** 

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

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

1	2	3
8		4
7	6	5

Goal node

SubProblem

	1	3
@	2	@
@	@	4

search ↙

1	2	3
@		4
@	@	@

Heuristics: Dominance

If $h_2(n) \geq h_1(n)$ for every state n (both admissible)
then h_2 dominates h_1

Which one is better for search (why?) $h_2 \geq h_1$

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)

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

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?)

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

CPSC 322, Lecture 8



why

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

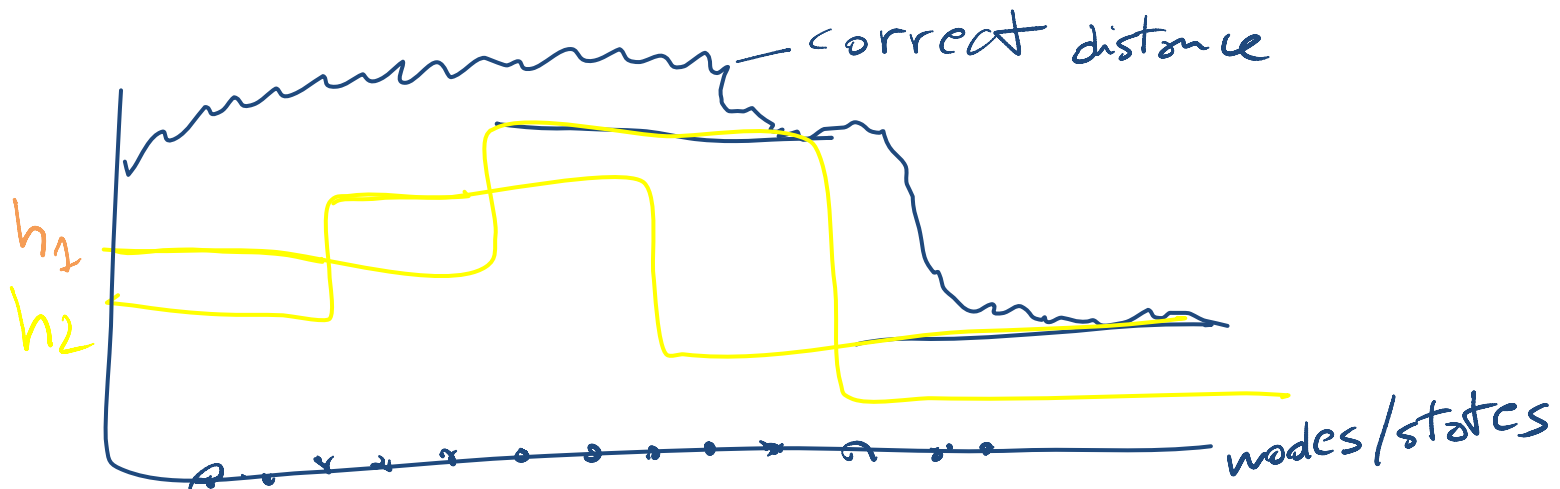
Slide 21

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



Combining Heuristics: Example


In 8-puzzle, solution cost for the 1,2,3,4 subproblem is substantially more accurate than simpler heuristic (sum of distance of each tile position from correct position)

But not in all cases

So.....

$\max(h_3, 1, 2, 3, 4, h_2)$
is a better heuristic

Learning Goals for today's class

- Construct admissible heuristics for appropriate problems.
 - Verify Heuristic Dominance. 
 - Combine admissible heuristics
-
- From previous classes
Define/read/write/trace/debug different search algorithms
 - With / Without cost
 - Informed / Uninformed

Next Class

Combining LCFS and BFS: A^* (finish 3.6)

- A^* Optimality