Heuristic Search

Computer Science cpsc322, Lecture 7

(Textbook Chpt 3.6)

Sept, 19, 2012

Course Announcements

Marks for Assignment0: will be posted on Connect on Fir

Assignment1: will also be posted on Fri

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

Giuseppe Tue 2 pm, my office.
Nathaniel Tomer Fri 11am, X150 (Learning Center)
Tatsuro Oya Wed 11am, X150 (Learning Center)
Mehran Kazemi Mon 11 am, X150 (Learning Center)

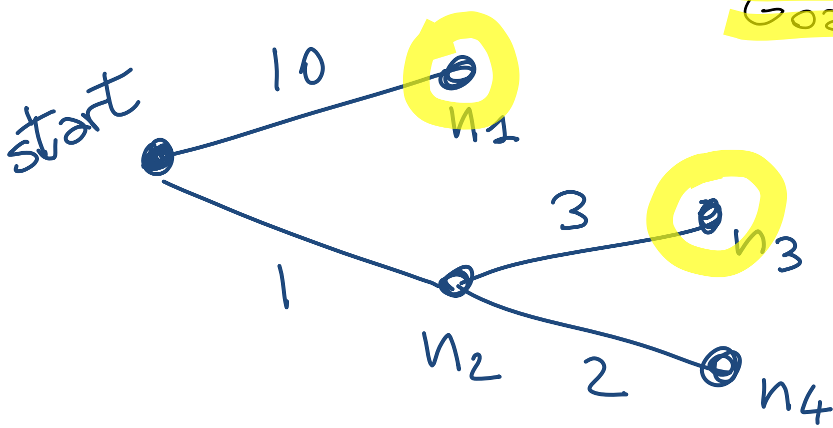
Lecture Overview

- 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.

Goal states



BFS <start, n1>

LCFS

<start, n2, n3>

- 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), (⟨n0, n3⟩ 8) , (⟨n0, n1, n4 ⟩ 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.**

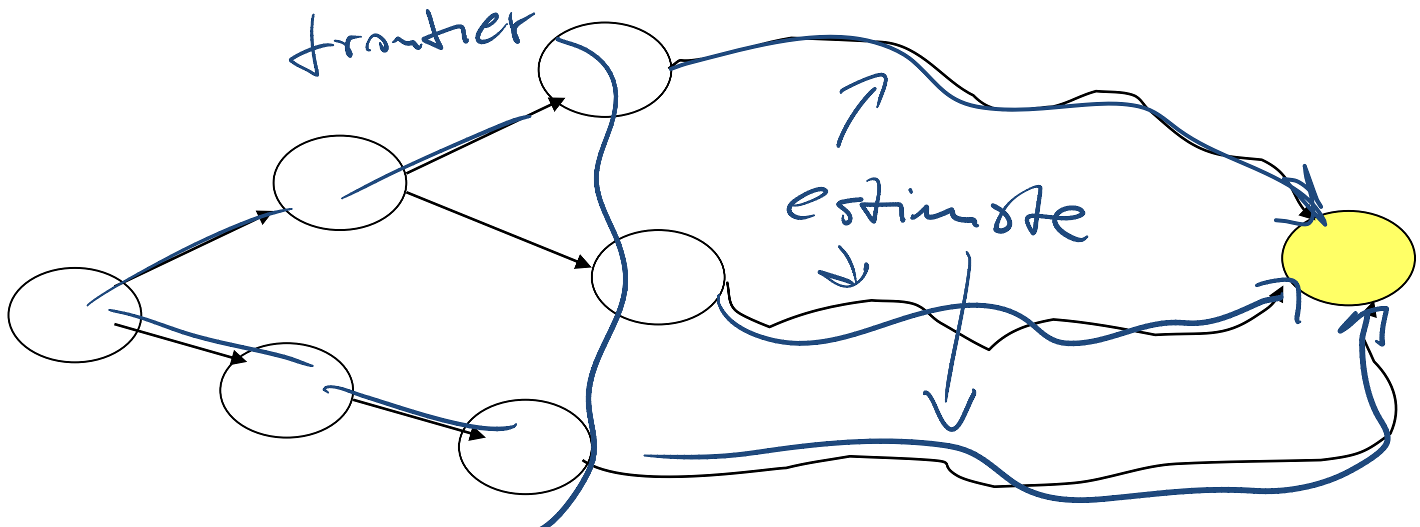
This is called a ***heuristic***

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: $h(\langle n_0, \ldots, n_k \rangle) = h(n_k)$
- 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**)

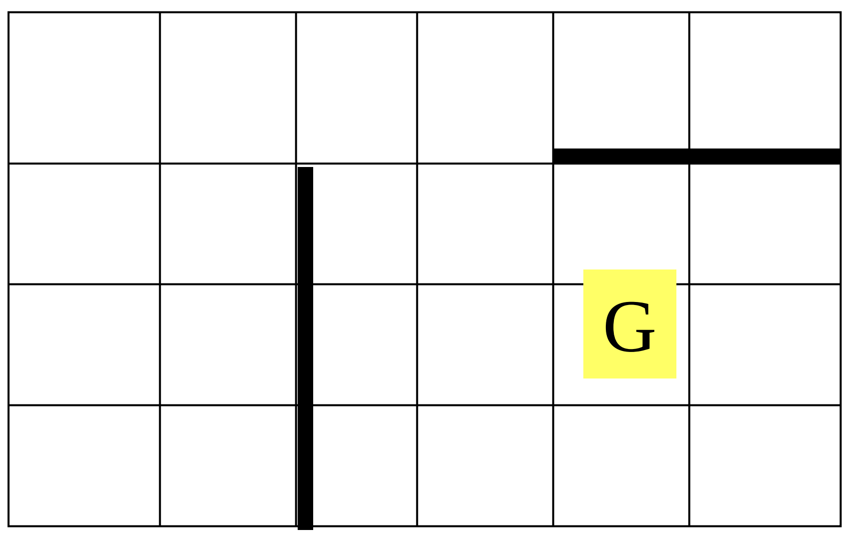
A search heuristic *h(n)* is admissible if it is never an overestimate of the cost from *n* to a goal.

- 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…

X_G Y_G Goal state

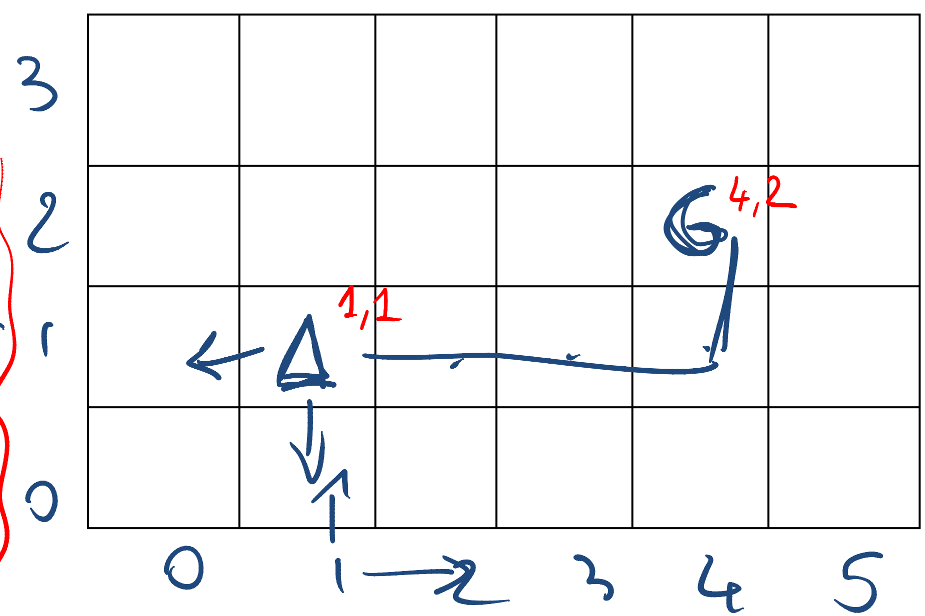
X_c Y_c Current state

$|X_G - X_c| + |Y_G - Y_c|$

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

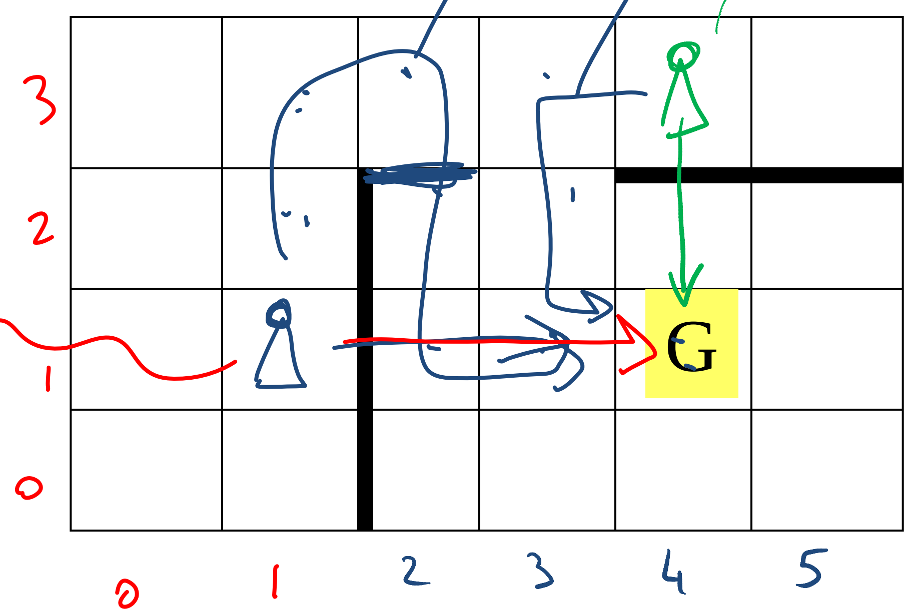
robot on grid
20 possible states

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

7

4

h = 2
for state (4,3)

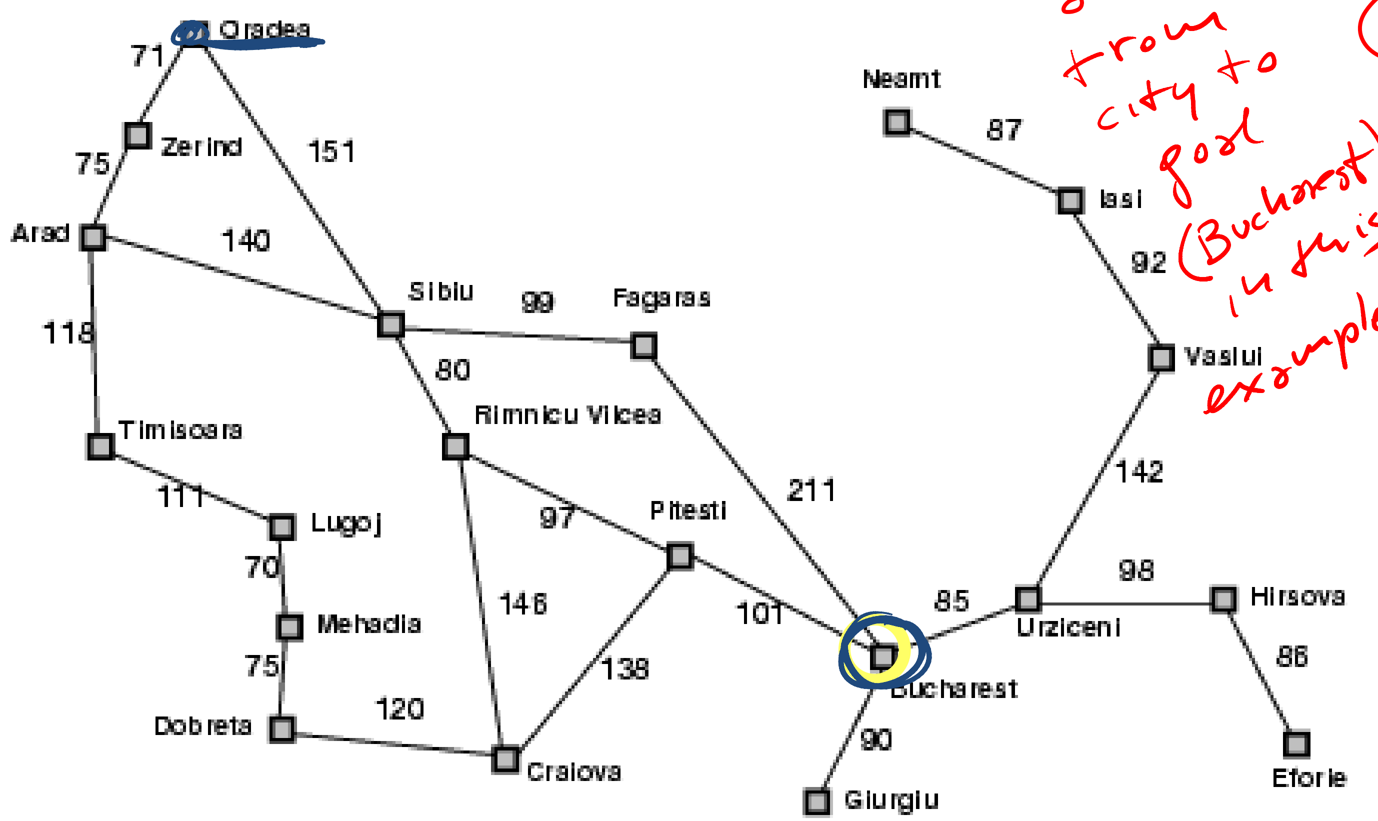


h = 3
for state (1,1)

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)*.

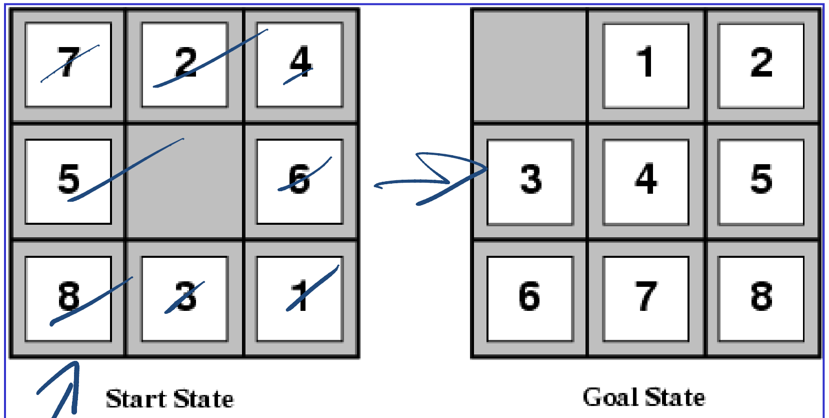
always <= than any road from city to goal (Bucharest in this example)



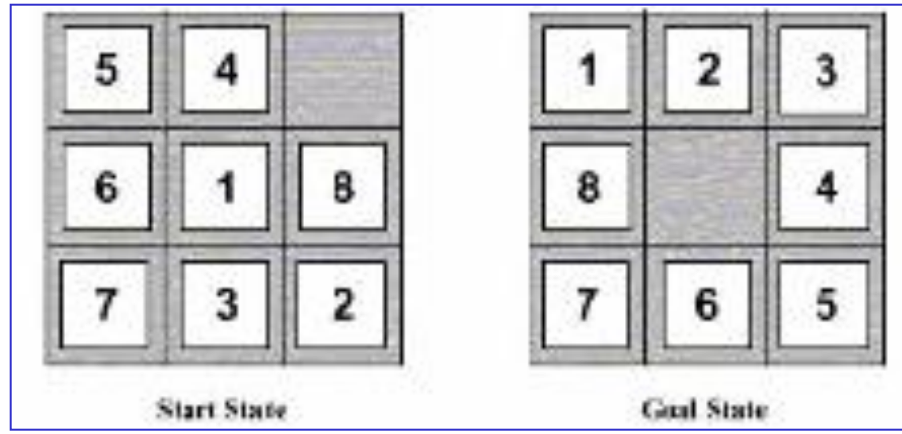
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example Heuristic Functions (1)

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



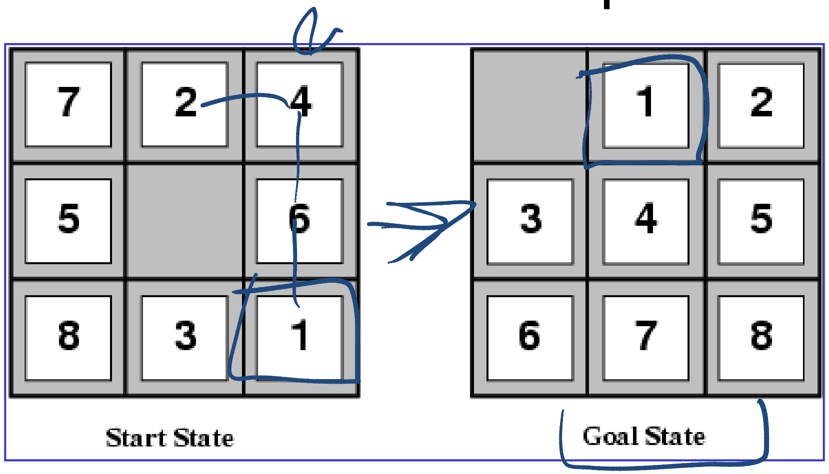
h 8

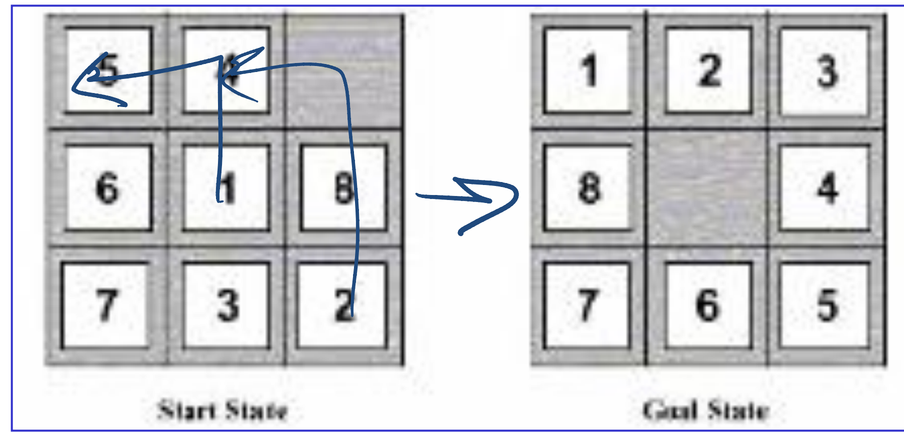


h 7

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

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

Σ 3 1 2 2 2 3 3 2

1	2	3	4	5	6	7	8
2	3						

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

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

simpler!

SubProblem

	1	3
@	2	@
@	@	4

1	2	3
8		4
7	6	5

Goal node

1	2	3
@		4
@	@	@

Goal

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 h_1
(2) tiles can move to any adjacent square h_2

(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 then h_2 dominates h_1

***Is h_2* better for search** (why?)

yes if both admissible

(2) >= (1)

yes	no	It depends

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)

why

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

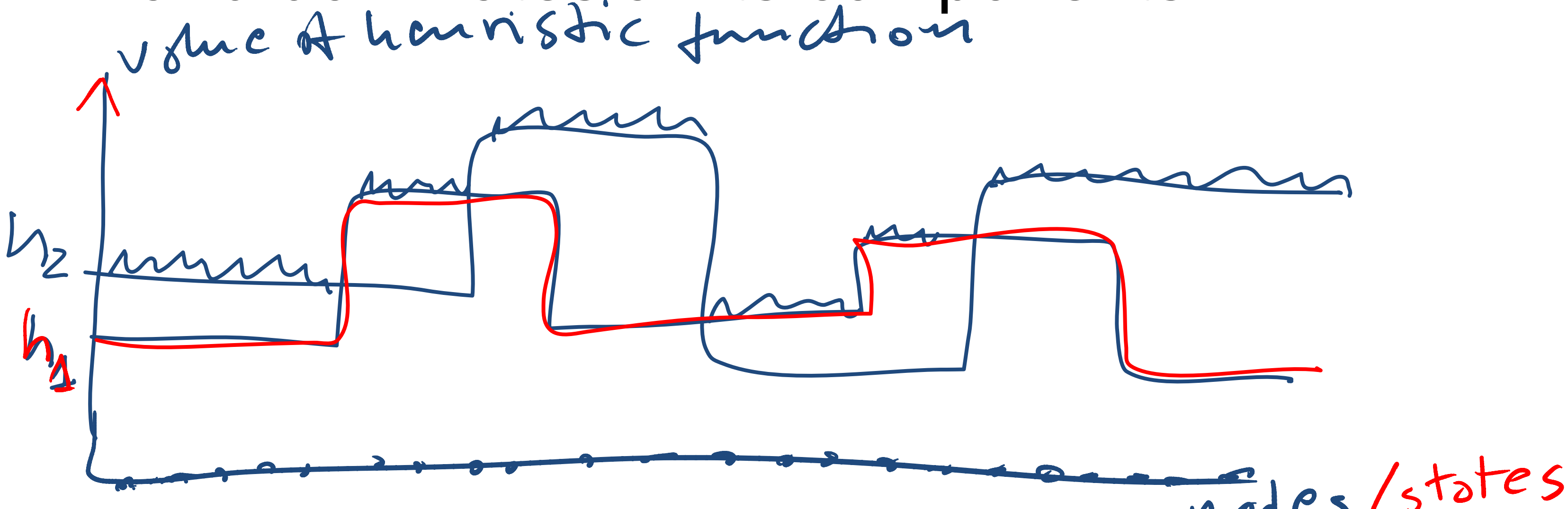
Combining Heuristics

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



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

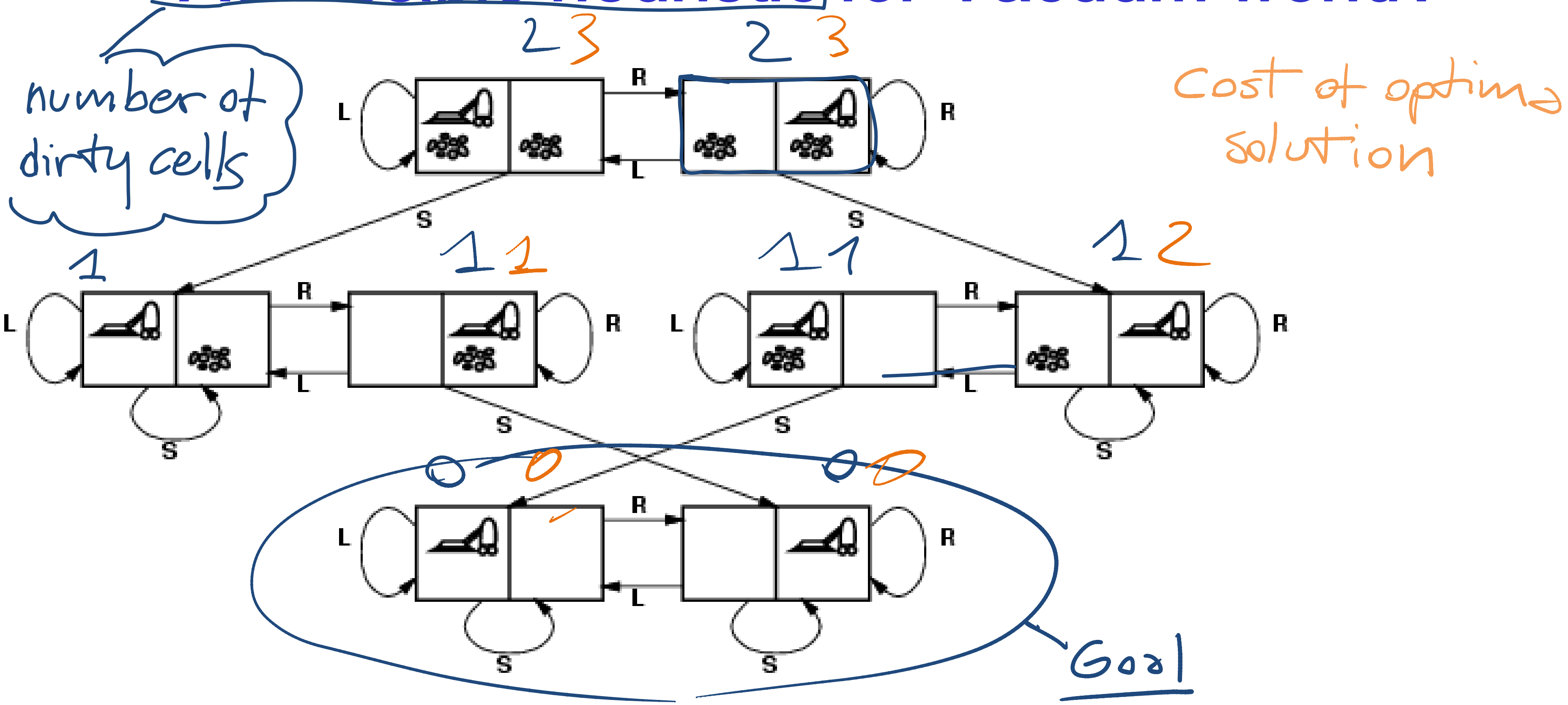
of each tile from its position in goal

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

Learning Goals for today's class

- **Construct admissible heuristics** for a given problem.
- **Verify Heuristic Dominance.**
- **Combine admissible heuristics**

• From previous classes
Define/read/write/trace/debug different search algorithms
  - With / Without cost
  - Informed / Uninformed

Next Class

- Best-First Search
- Combining LCFS and BFS: A*   (finish 3.6)
- A* Optimality