CSCI 3202: Intro to Artificial Intelligence

Lecture 9: Heuristics

Rachel Cox

Department of Computer Science

Heuristics

- Using a heuristic can help solve a problem more quickly.
- There is an ”art” to deciding on a heuristic function.
- We want $h(n)$ to be admissible. But we need to keep in mind that the lower $h(n)$ is, the more nodes A* expands (making it slower.)

Heuristics

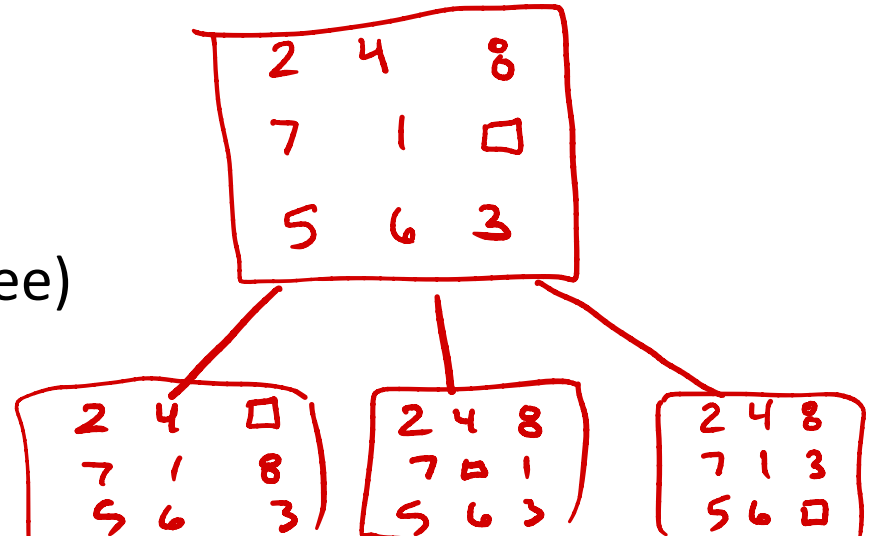
2	4	8
7	1	
5	6	3

	1	2
3	4	5
6	7	8

Branching factor $b \approx 3$

Average solution depth = 22

- BFS might expand around $3^{22} \approx 3.1 \times 10^{10}$ nodes (tree)
- Graph version: $\frac{9!}{2} \approx 180{,}000$ distinct reachable states



Heuristics

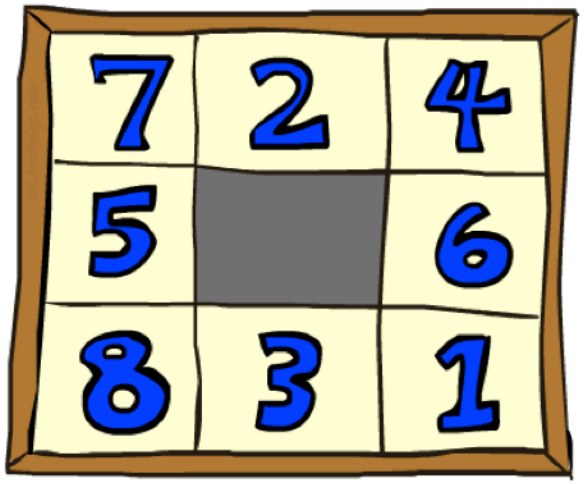
How do we come up with heuristics?

1) Generate heuristics from relaxed problems.

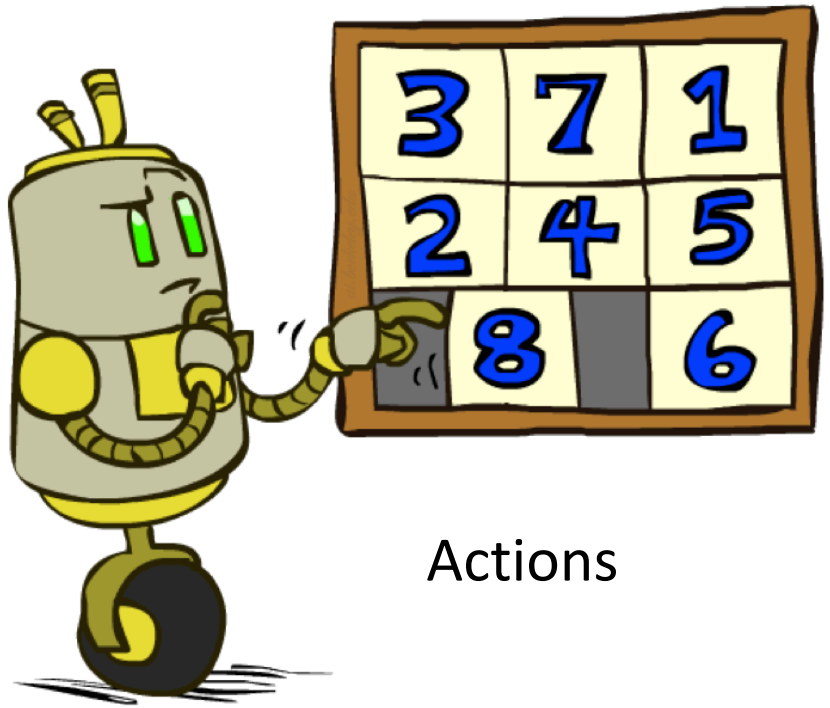
2) Generate heuristics from sub-problems.

3) Learning heuristics from experience.

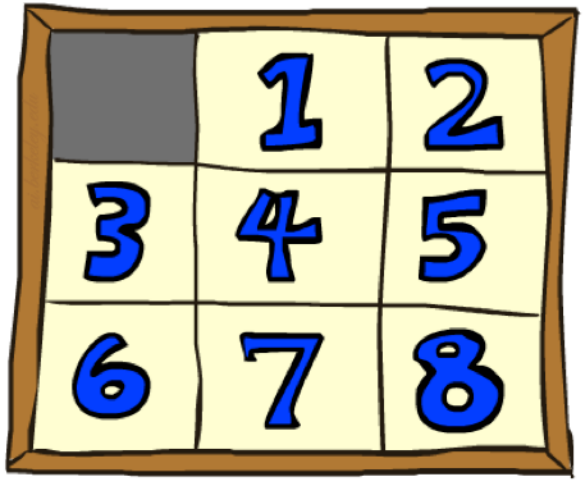
Heuristics



Start State



Actions



Goal State

- What are the states?
- How many states? 9!
- What are the actions? blank space can slide left, right, up, down
- How many successors from the start state? 4 successors
- What should the costs be?

every time we move a tile, we add 1 to the cost

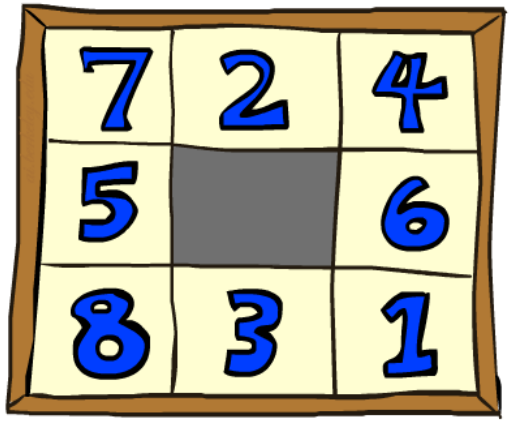
Slide source: CS 188 Berkeley

Heuristics

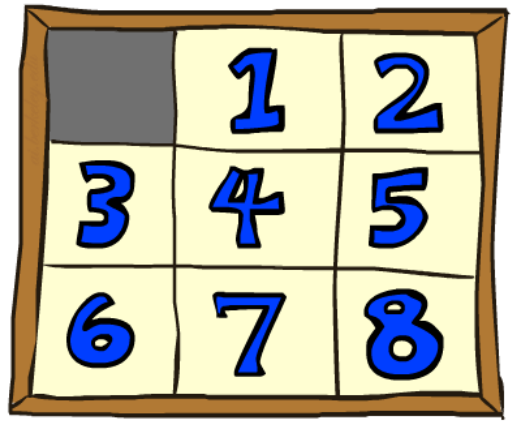
Consider a Relaxed problem - allowing the tiles to be pulled off from the board and replaced in correct order

base minimum

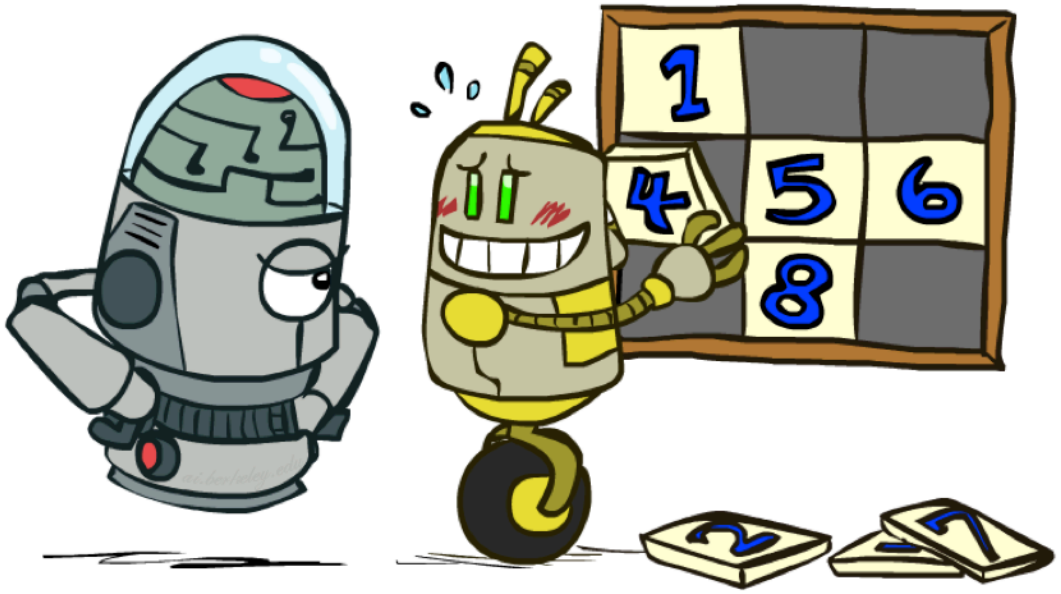
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = ? 8
- This is a *relaxed-problem* heuristic



Start State



Goal State



Slide source: CS 188 Berkeley

Relaxed problems

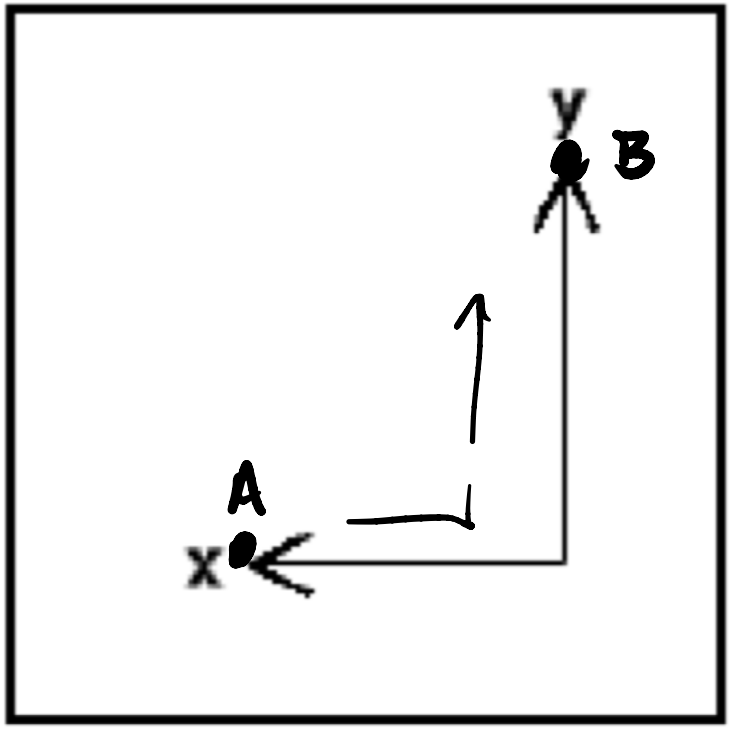
All relaxed problems are doing is adding edges to the state space graph.

$\Longrightarrow$ Any optimal solution to the original problem is also a solution to the relaxed problem.

But because of the “short-cuts” of the relaxed problem, there will typically be cheaper solutions than the full problem too.

$\Longrightarrow$ Optimal solutions of the relaxed problem are admissible heuristics

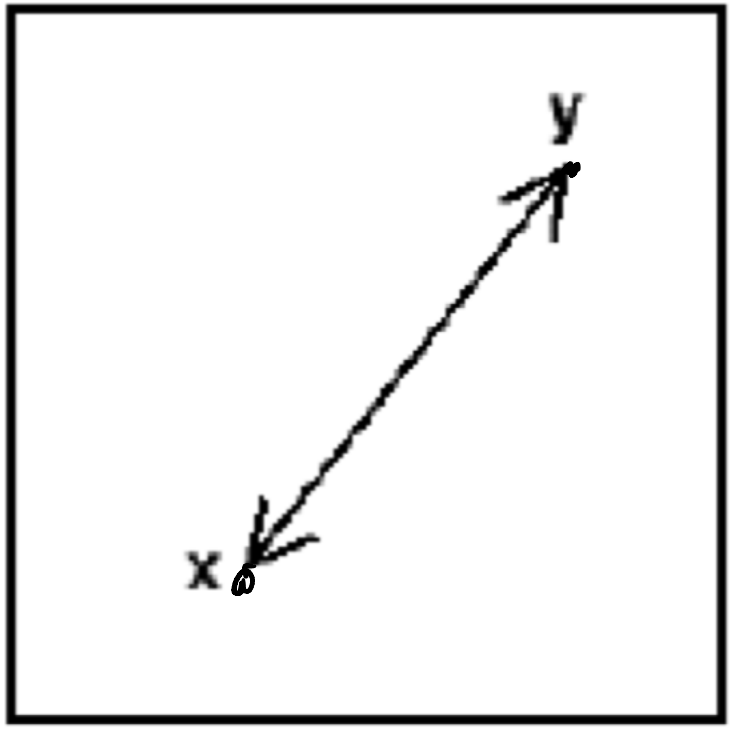
Heuristics



Manhattan

aka Taxi cab

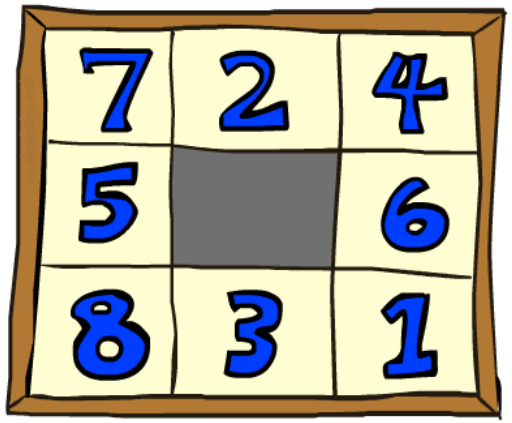
distance



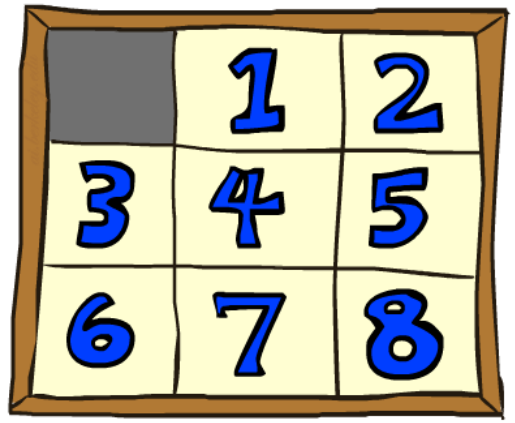
Euclidean

Heuristics

- 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 is it admissible?
- h(start) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2
  = 18



Start State



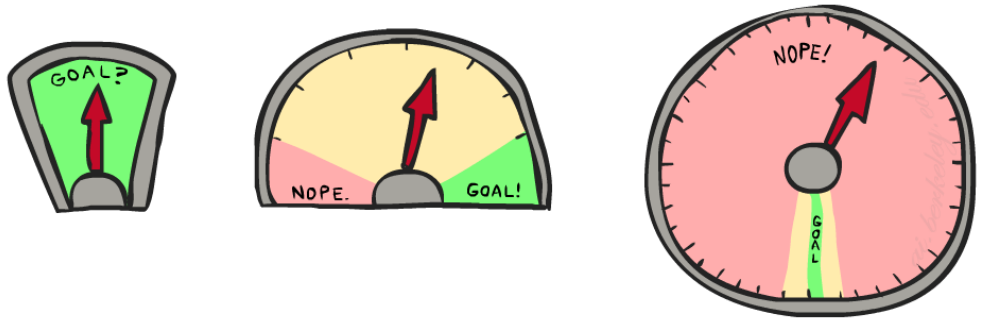
Goal State

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

Slide source: CS 188 Berkeley

Heuristics

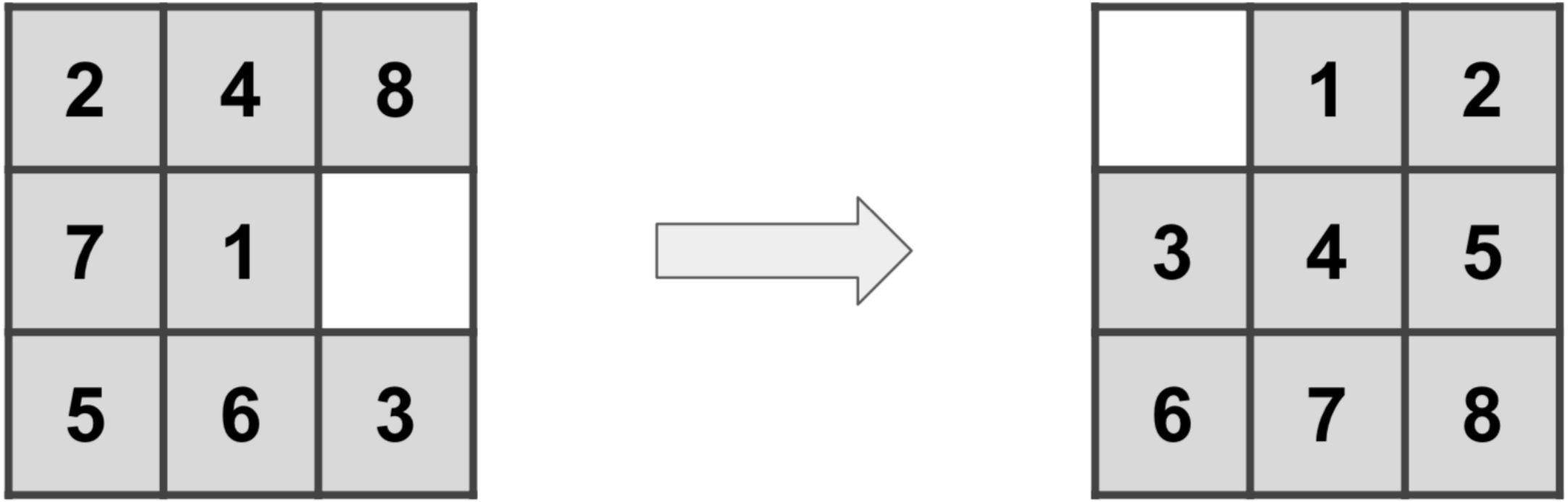
- How about using the *actual cost* as a heuristic?
  - Would it be admissible? actual cost ≤ actual cost
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Slide source: CS 188 Berkeley

Heuristics



So we want good heuristics!

- $h_1 =$ number of misplaced tiles
- $h_2 =$ sum of distances of the tiles from their goal positions

Heuristics

Reminder:

Complexity of A*: $O\left((b^{\epsilon})^{d}\right)$

$\rightarrow$ The effective branching factor is really b^{ϵ}

- Suppose A* finds a solution at depth *d and* generates *N* nodes to find it.
- b^{ϵ} is the branching factor that a uniform tree of depth *d* would have in order to contain *N*+1 nodes:

$$N + 1 = 1 + b^{\epsilon} + (b^{\epsilon})^{2} + (b^{\epsilon})^{3} + \ \ldots + (b^{\epsilon})^{d}$$

Heuristics

Depending on which heuristic we use, h_1 or h_2, the search cost (nodes generated) and b^{ϵ} will be different.

Performance comparison

	Search Cost (nodes)			Effective Branching Factor		
d	IDS	A* (h_1)	A* (h_2)	IDS	A* (h_1)	A* (h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	-	539	113	-	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27

- h_2 (Manhattan distance) dominates h_1 (misplaced tiles)

Heuristics

- A* using h_2 will never expand more nodes than A* using h_1

  Every node with $f(n) < C^*$ will be expanded

  $\Longrightarrow f(n) = g(n) + h(n)$, so every node with $h(n) < C^* - g(n)$ will be expanded

  But $h_1(n) \leq h_2(n)$, which means any node expanded by A* using h_2 will be expanded by A* using h_1

So it's best to use a heuristic with higher values

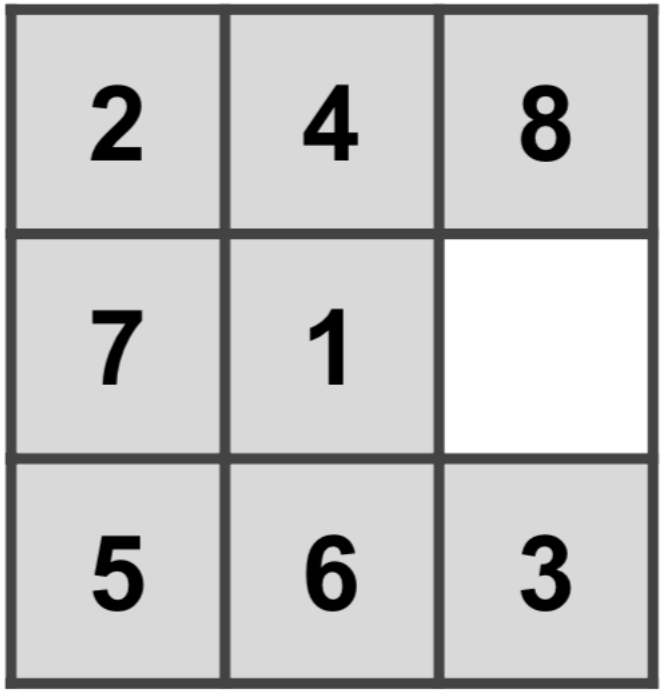
- Makes sense, because those are almost necessarily more accurate:

  Admissible → Can't overestimate ⟶ The higher they are, the better they are.

Relaxed problems

Once we have several heuristics, how do we decide which one to use?

- If any dominate, then definitely go with that.
  $\forall n: h_a(n) \geq h_c(n)$
- More generally, just take the most accurate heuristic for whatever node is in question at the moment: $h(n) = \max\{h_1(n), \ldots, h_k(n)\}$

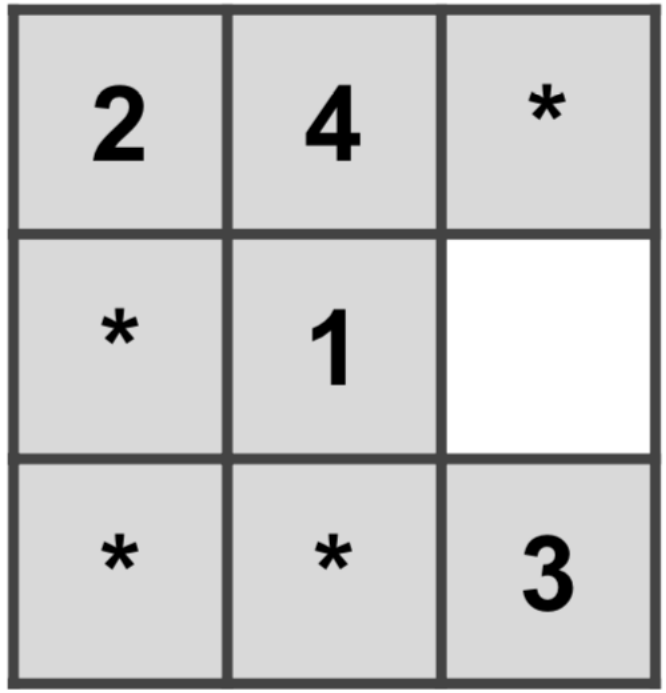


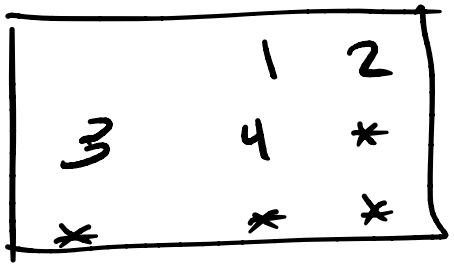
Heuristics from sub–problems

Arranging the tiles {1, 2, 3, 4} into the proper slots is a subproblem of the general 8-til problem.

- The cost of an optimal solution to this subproblem is cheaper than the cost of the optimal solution to the full problem.

- Construct a pattern database:
  - Solve each possible configuration of the subproblem
  - Store the cost of the optimal solution
  - Use this as a heuristic
  - Even better: Do this for multiple subproblems and combine the heuristics

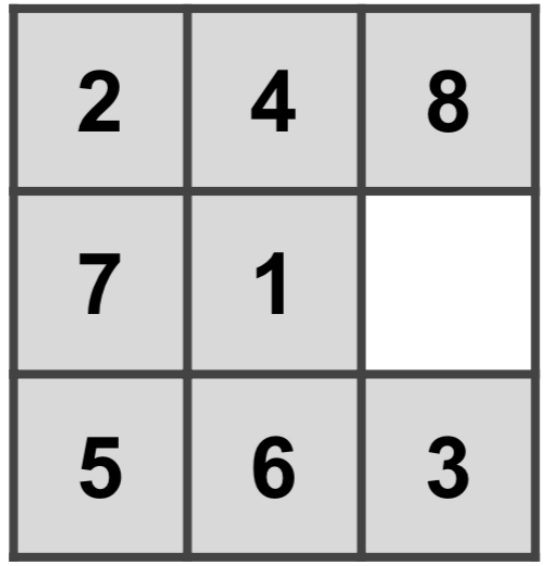




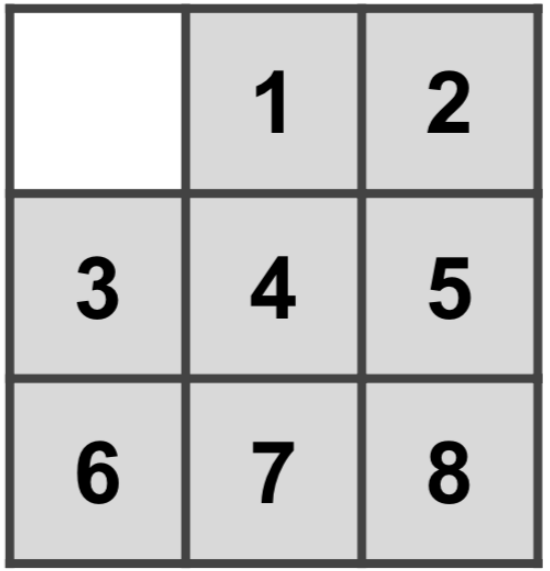
Heuristics from learning

Suppose we solved thousands of 8-tile puzzles

- Then we have a gigantic sample of initial states and of optimal solution paths.



	n_1	n_2	n_3	…
# misplaced tiles ($x_1(n)$)	2	8	5	…
# adjacent tiles that shouldn't be adjacent in goal state ($x_2(n)$)	3	6	4	…
Manhattan distance to goal ($x_3(n)$)	8	14	11	…
Cost	12	24	17	…



Heuristics from learning

Predict cost from features of the initial states:

$h(n) = c_1 x_1(n) + c_2 x_2(n) + c_3 x_3(n) + \ \ldots$

Potential issue:

- Not necessarily admissible/consistent

- Could be, depending on the features and regression constants

	n_1	n_2	n_3	…
# misplaced tiles ($x_1(n)$)	2	8	5	…
# adjacent tiles that shouldn't be adjacent in goal state ($x_2(n)$)	3	6	4	…
Manhattan distance to goal ($x_3(n)$)	8	14	11	…
Cost	12	24	17	…

Next Time

Local Search