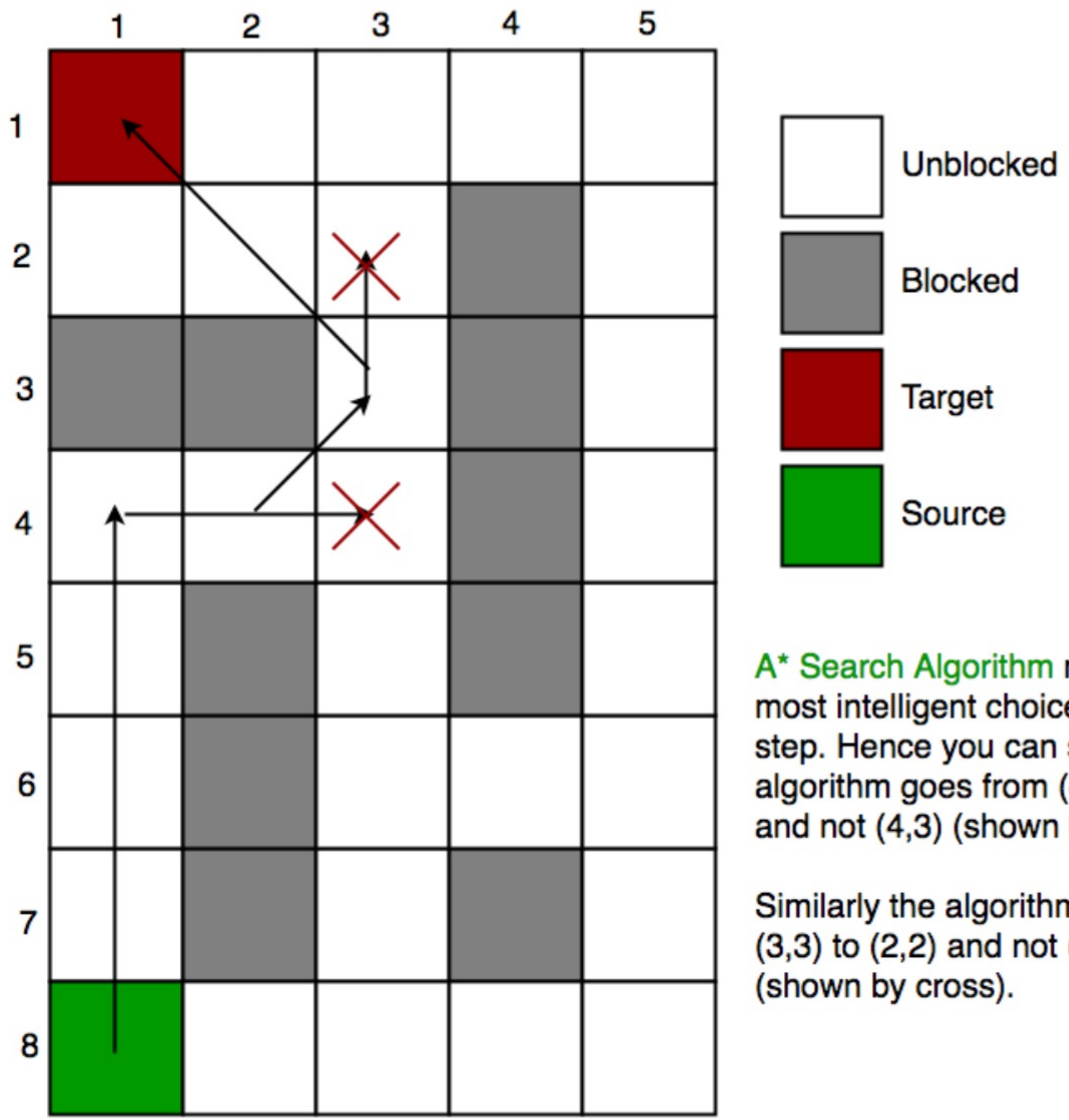
CSCI 3202: Intro to Artificial Intelligence
Lecture 10: A* Search and Heuristics

Rhonda Hoenigman
Department of Computer Science



A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).

Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Source

Optimality of A* Search

So A* is **optimal**, **complete**, and **optimally efficient**.

Why do we even care about other search algorithms?

- **Number of nodes** to expand along the goal contour is still **exponential** in depth of solution/length of solution path.

- Absolute error: $\Delta \coloneqq h^* - h$
  - h* = actual cost from root to goal
  - h = heuristic you used

- Relative error: $\epsilon \coloneqq (h^* - h)/h^*$

A* Search

Complexity depends strongly on state space characterization

- Single goal, tree, reversible actions $\rightarrow O\left(b^{\Delta}\right)$, or $O\left(b^{\epsilon d}\right)$ with constant step costs (*d* is solution depth)

  Δ typically is proportional to the path cost $h*$, so ϵ is pretty much constant (or growing with d), and we can rewrite: $O\left((b^{\epsilon})^{d}\right)$

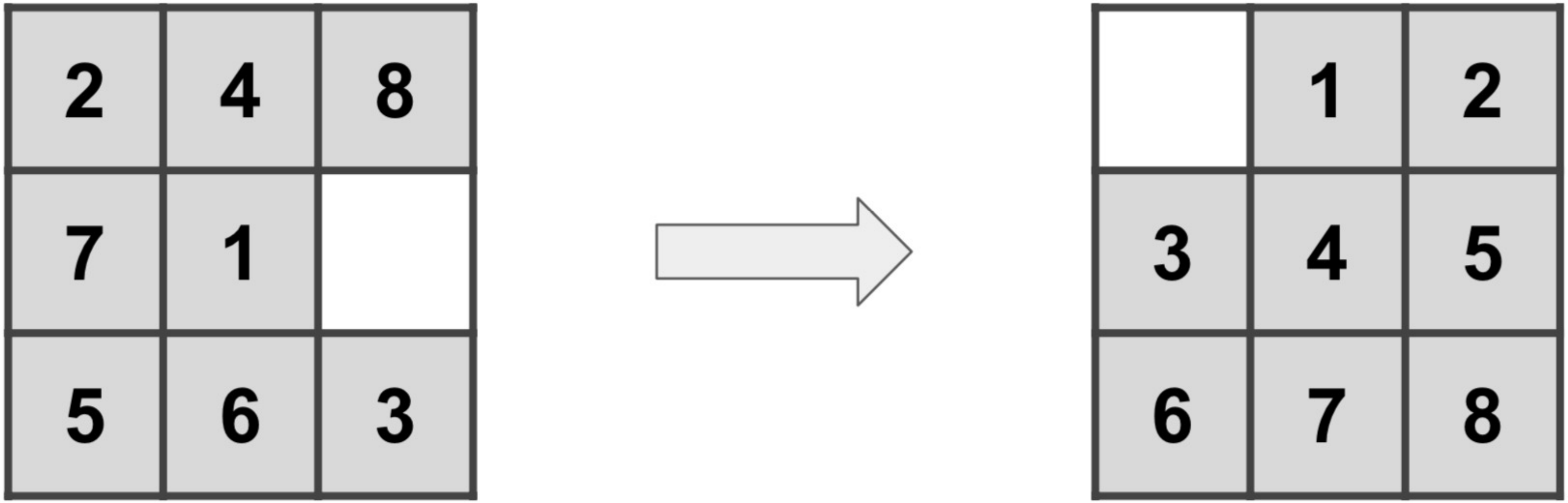
  $\rightarrow$ The effective branching factor is really b^{ϵ}.
  $\rightarrow$ Important to choose as good of a heuristic as we can.

- Many goal states/near-goal states can be a problem -- need to expand a ***lot*** of branches.

Back to 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



Example: solve the 8-tile problem

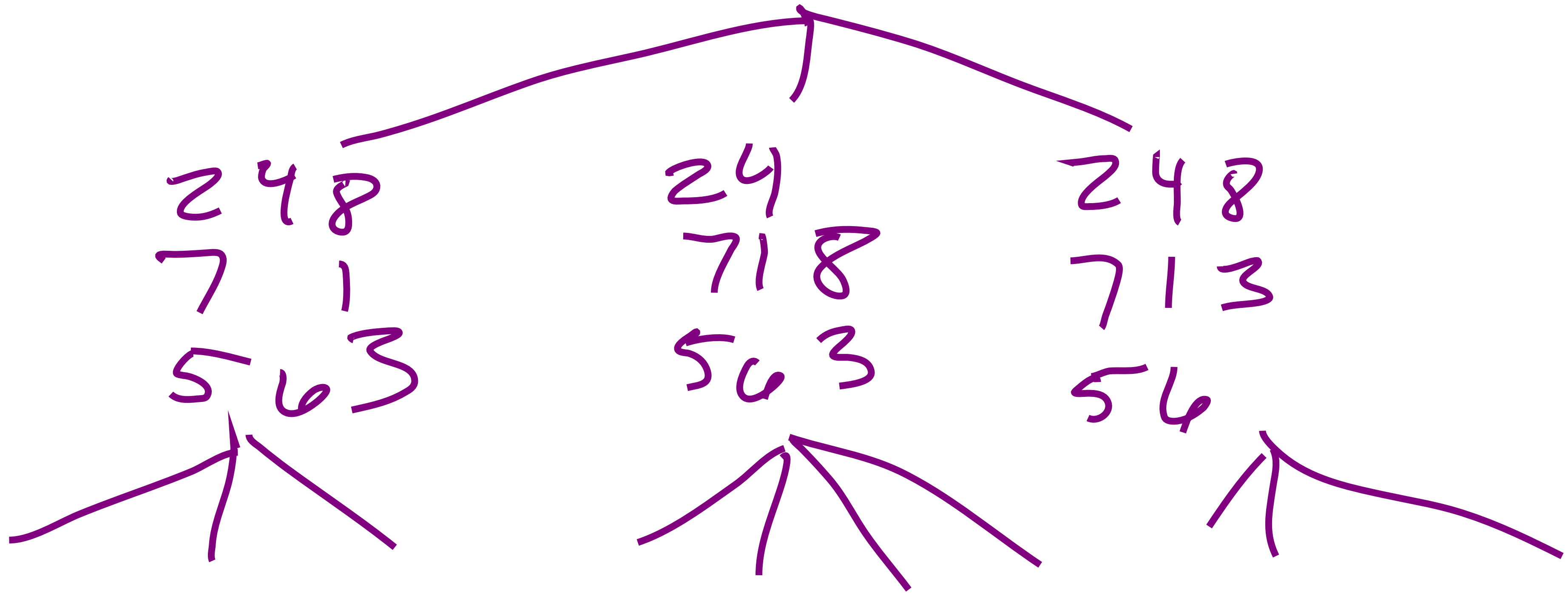
8_tile problem search tree

Is any state closer to goal than other states?

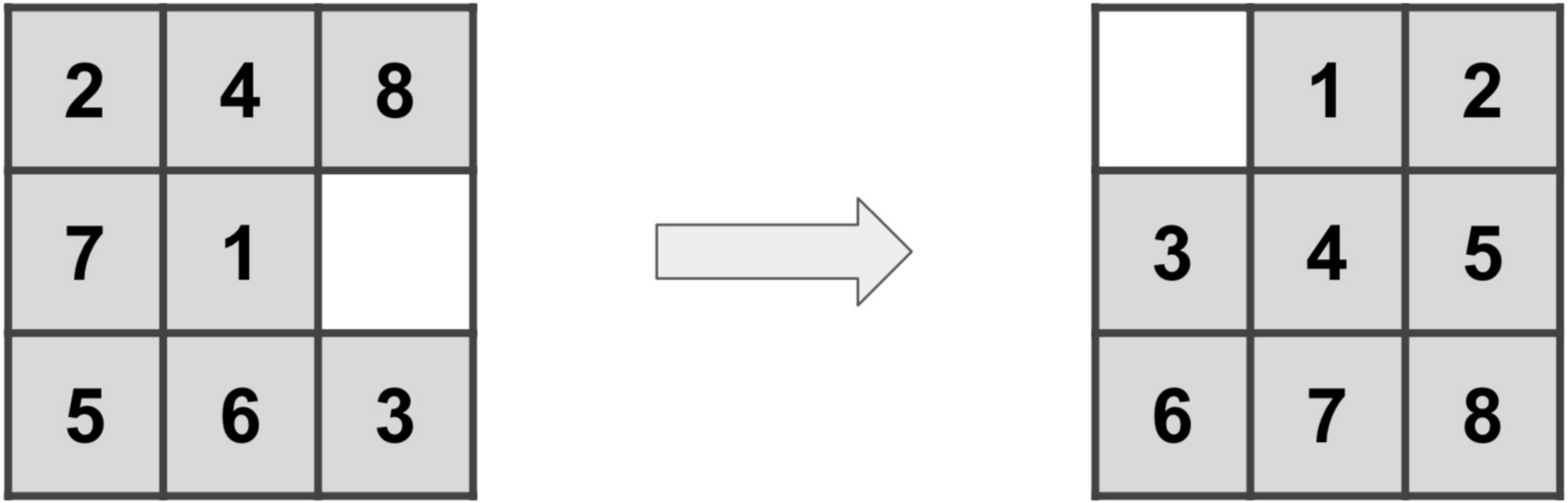
2	4	8
7	1	
5	6	3

→

	1	2
3	4	5
6	7	8



Heuristics



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.

Relax the problem constraints

2) Generate heuristics from sub-problems.

Smaller instance of larger problem. e.g. fix some tiles in place.

3) Learning heuristics from experience.

Commonly observed patterns or path length in solutions observed in data.

8-tile rules

1 tile moves at a time.

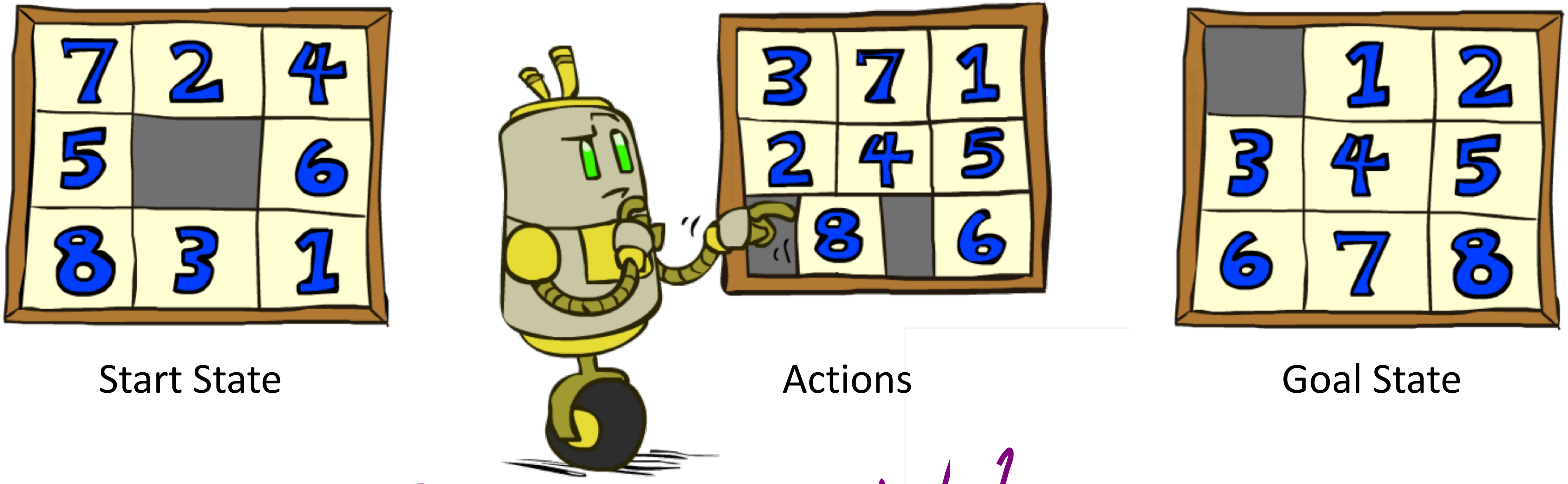
Move vertical or horizontal only.

Move to blank space only.

3x3 grid

A move is one square.

Heuristics



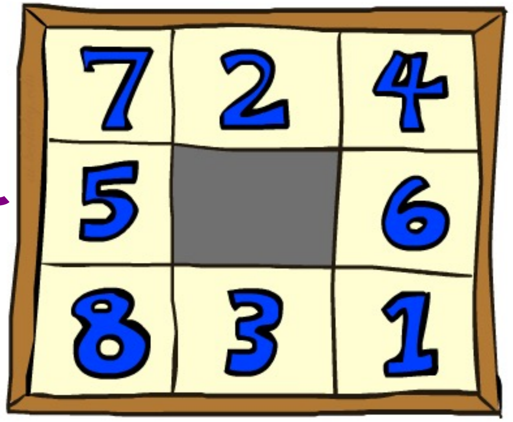
Start State   Actions   Goal State

- What are the states? Positions of all tiles
- How many states? 9!
- What are the actions? slide a tile one position
- How many successors from the start state? 4
- What should the costs be? Each move = 1, total cost is # of moves to goal state.

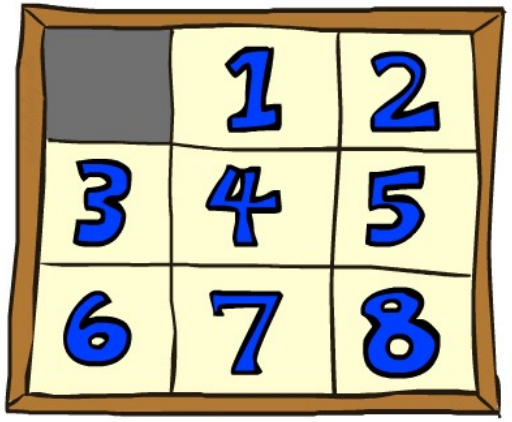
Slide source: CS 188 Berkeley

Heuristics – relaxed problem example

- Heuristic: Number of tiles misplaced
- Why is it admissible? will underestimate true cost
- h(start) = ? 9
- This is a *relaxed-problem* heuristic

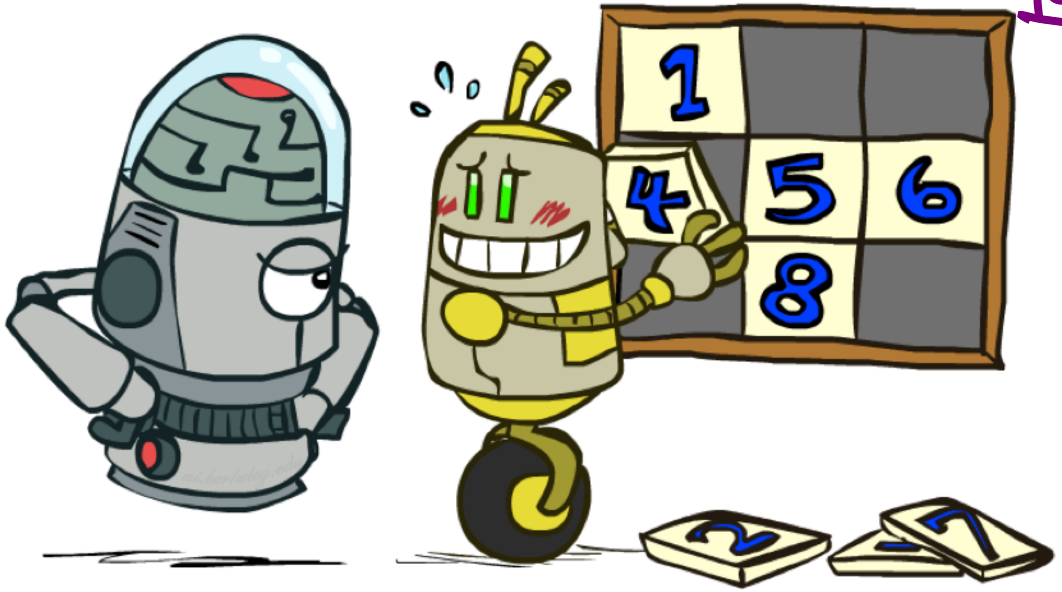


Start State



Goal State

Relaxed
Position must blank
1 square per move
Move only vert or
horizontal



Slide source: CS 188 Berkeley

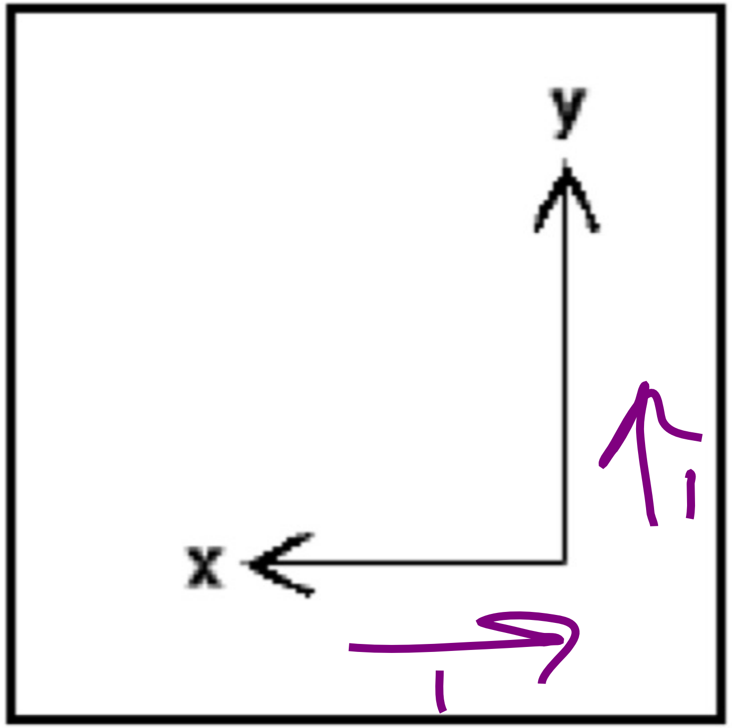
Relaxed problems

⟹ Any optimal solution to the original problem is also a solution to the relaxed problem.

Relaxed problems include “short-cuts” of the original problem – they will be cheaper solutions than the full problem.

⟹ Optimal solutions of the relaxed problem are admissible heuristics

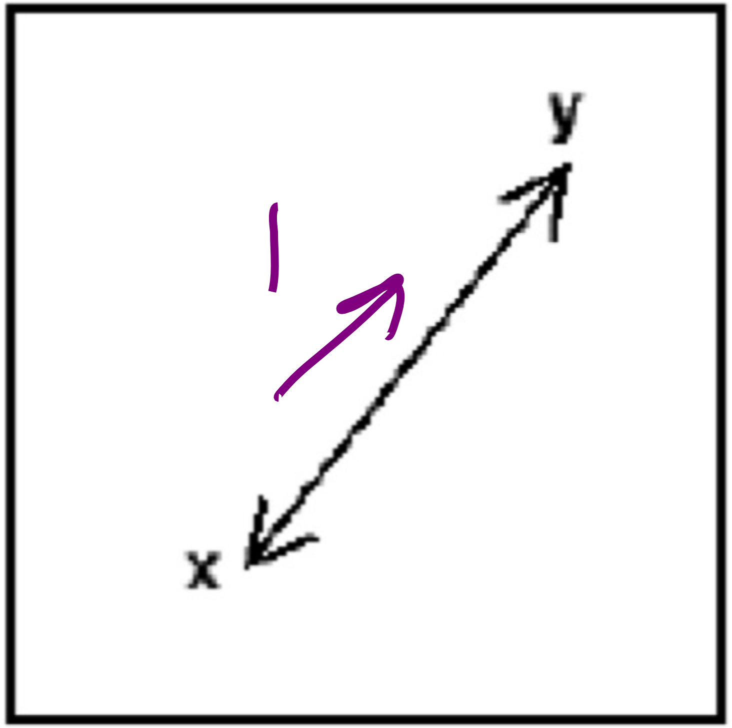
Heuristics – a distance example



Manhattan

cost = 2 to go

$x \to y$



Euclidean

cost = 1 to go

$x \to y$

Heuristics – relaxed problem example

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

• Total Euclidean dist

- Total *Manhattan* distance

Euclidean 8

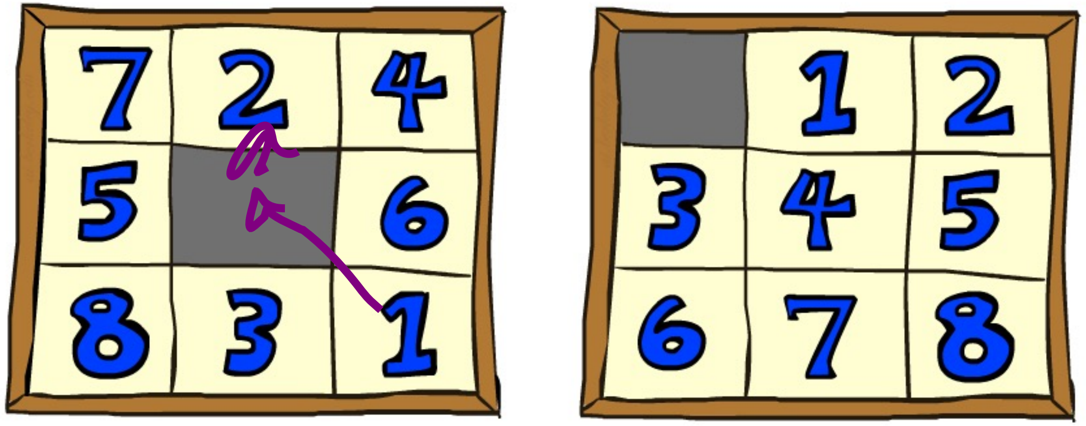
$2+1+1+1+$

$2+2+2+2$

- Why is it admissible?

- h(start) = 18

$3+1+2+2+2+3+3+2=18$ manhattan



Start State Goal State

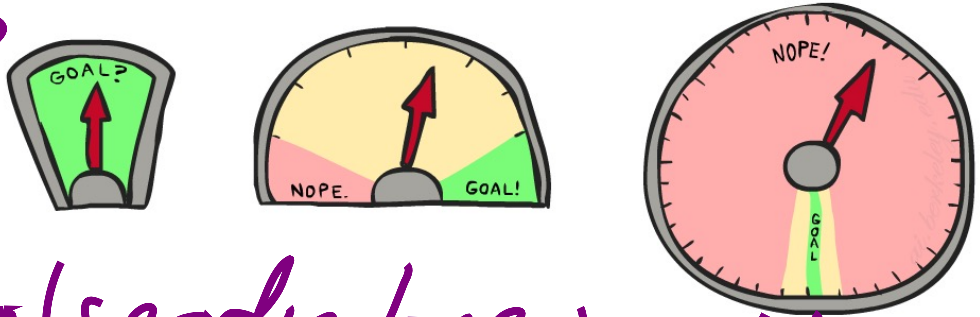
manhattan much faster than # of tiles misplaced

	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

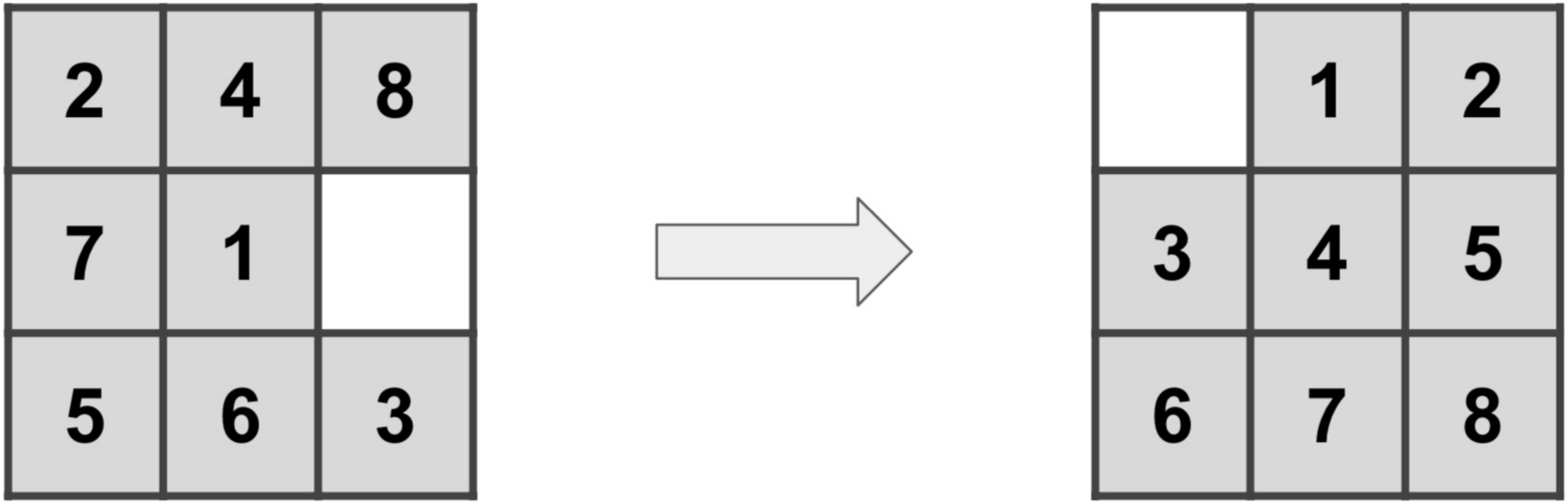
Heuristics

- How about using the *actual cost* as a heuristic?
  - Would it be admissible? yes
  - Would we save on nodes expanded? yes
  - What's wrong with it? Hard to determine. Wouldn't need a heuristic if we already knew cost



- 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

Heuristics – which one is better?



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

→ 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.

h_1 = # of tiles misplaced

h_2 = Manhattan dist

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

  h2 dominat

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.

Heuristics from sub–problems

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

2	4	*
*	1	
*	*	3

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	…

2	4	8
7	1	
5	6	3

	1	2
3	4	5
6	7	8

Heuristics from learning

Predict cost from features of the initial states:

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

	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	…

Potential issue:

- Not necessarily admissible/consistent
- Could be, depending on the features and regression constants

Next Time

Local Search