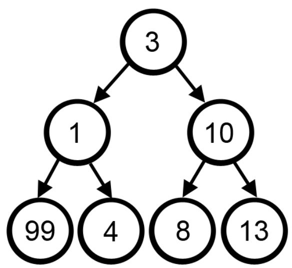
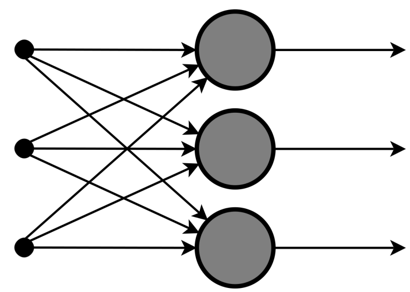
CSE 40171: Artificial Intelligence





Informed Search: Search Heuristics

Homework #2 has been released
It is due at 11:59PM on 9/30

Where do heuristics come from?

A good heuristic can go a long way…

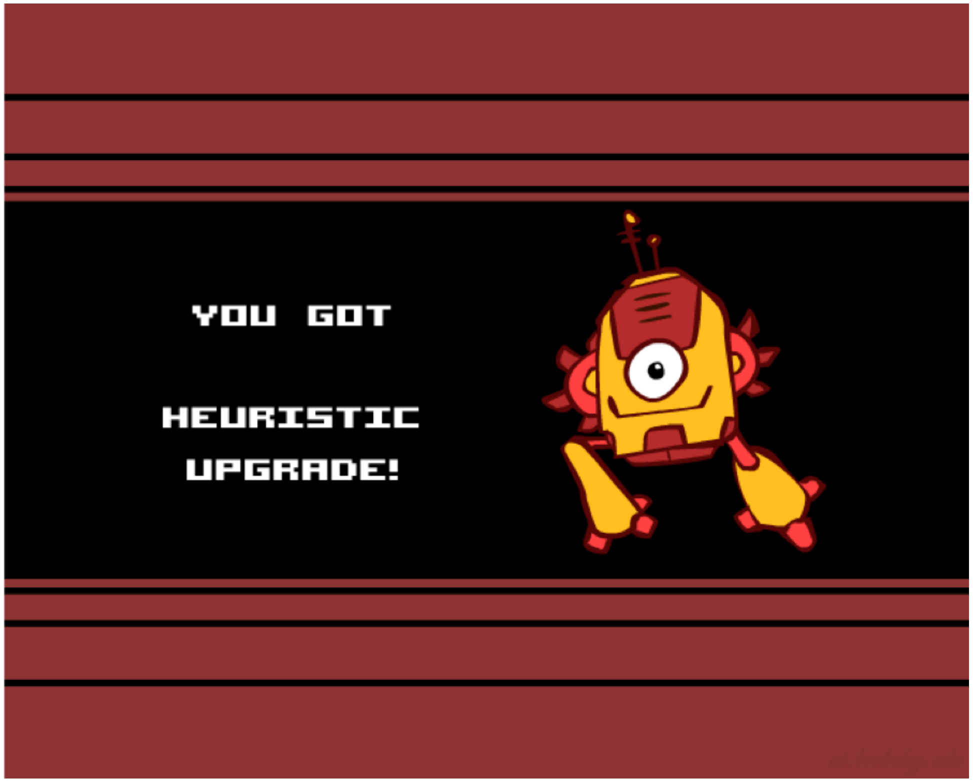
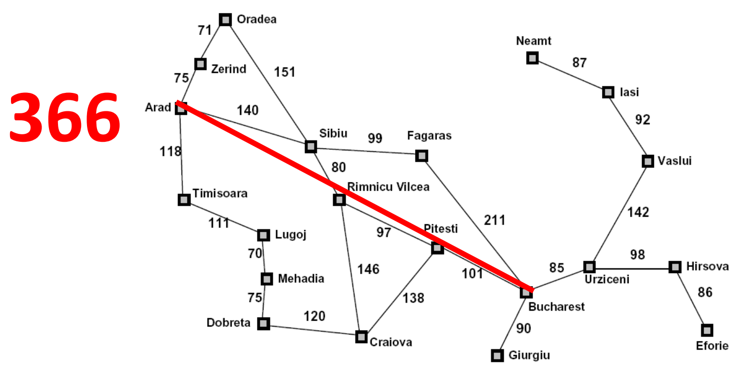


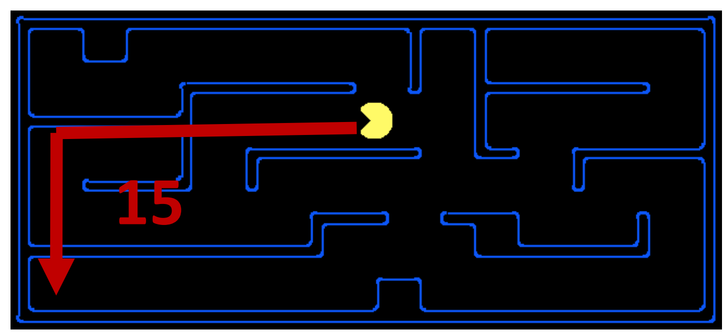
Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

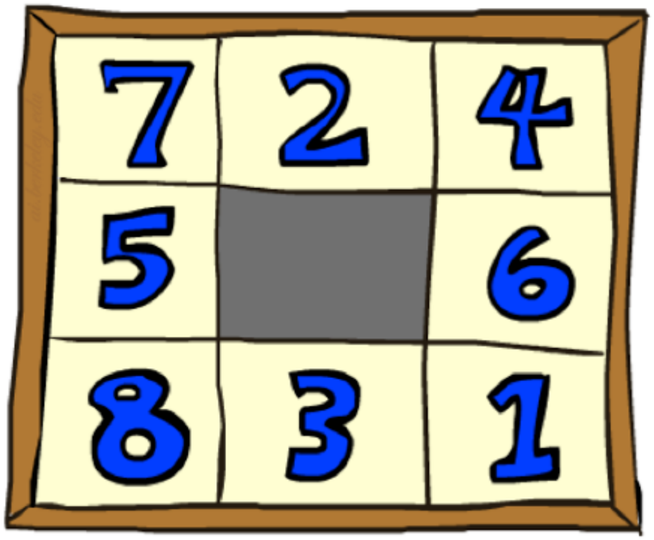
Often, admissible heuristics are solutions to relaxed problems, where new actions are available



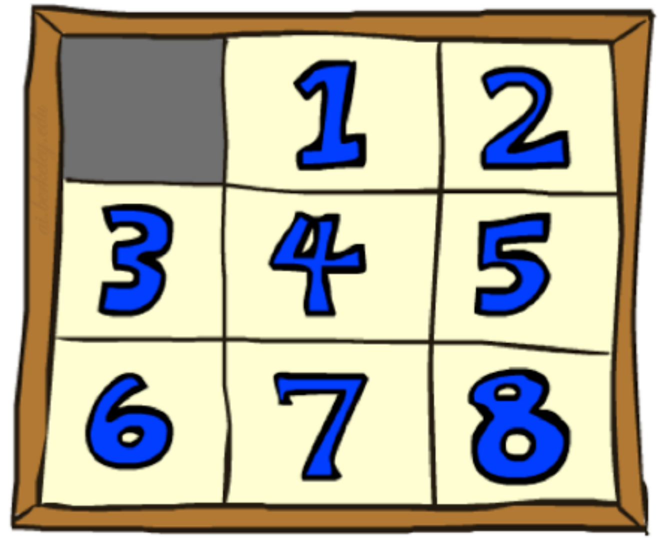


Inadmissible heuristics are often useful too

Case Study: the 8-puzzle



Start State

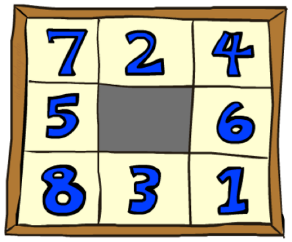


Goal State

How many steps long is the solution?

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

8-puzzle Stats



Average Solution Cost: about 22 steps

Branching Factor: about 3 steps

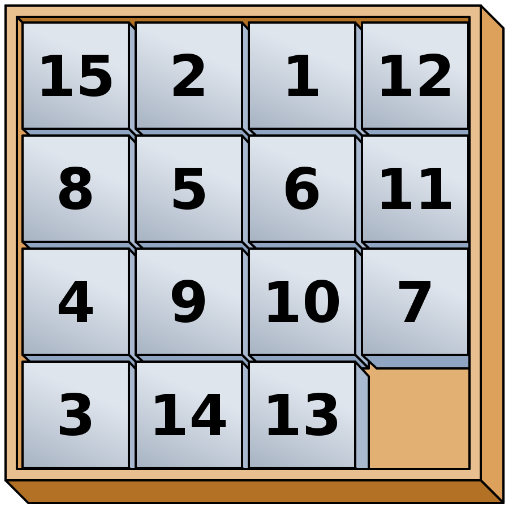
Exhaustive Tree Search: $3^{22} \approx 3.1 \times 10^{10}$ states

Graph Search: $9!/2 = 181{,}440$ states

- But the corresponding number for the *15-puzzle* is 10^{13}

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Large Numbers and Search Spaces



How large is a state space of **10 trillion** possibilities?

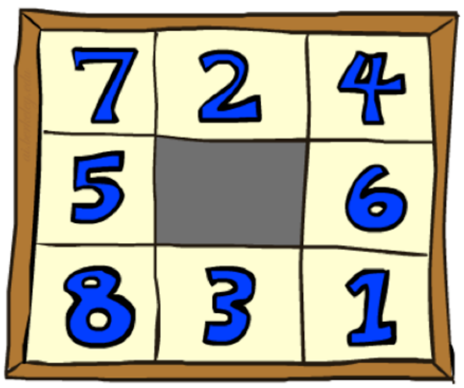
On a 3.1Ghz Core i7, it takes:

- 3m42.240s to enumerate 10^{10} states (exhaustive 8-puzzle)

Heuristics are needed to speed this up!

Heuristic Function #1

h_1 = the number of misplaced tiles



All eight tiles are out of position

Start state: $h_1 = 8$

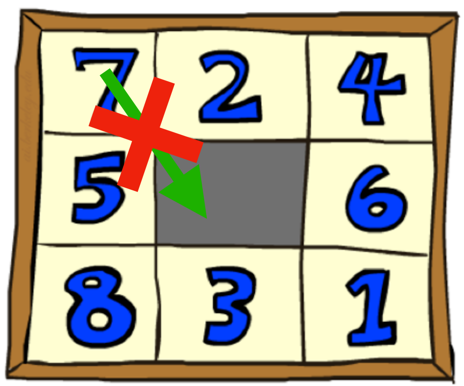
Q: Why is this heuristic admissible?

A: Any tile that is out of place must be moved at least once

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Heuristic Function #2

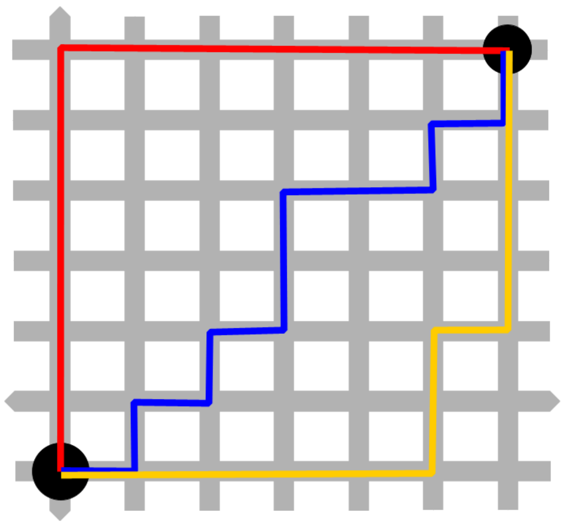
h_2 = the sum of the distances of the tiles from their goal positions



Tiles can't move along diagonals, so what do we do?

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Manhattan Distance



Manhattan distance bgiu ⓒ BY-SA 3.0 XaraX

The distance between two points in a grid based on a strictly horizontal and / or vertical path

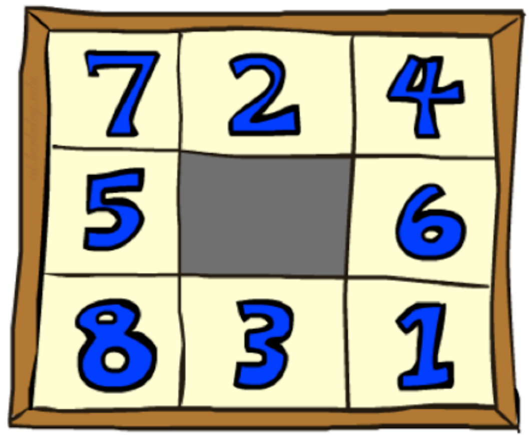
$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

where

$$\mathbf{p} = (p_1, p_2, \dots, p_n) \text{ and } \mathbf{q} = (q_1, q_2, \dots, q_n)$$

Heuristic Function #2

For this start state:



$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Q: Why is this heuristic admissible?

A: Any tile that is out of place must be moved at least once

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Effective Branching Factor

N = the total number of vertices generated by A*

d = the solution depth

b^* = the branching factor that a uniform tree of depth d would need to contain $N+1$ vertices

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Example: if A* finds a solution at depth 5 using 52 vertices, then the effective branching factor is 1.92

Effective Branching Factor

A well-designed heuristic has a value of b* close to 1

How do h_1 and h_2 stack up?

	Search Cost			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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Image Credit: Russel and Norvig

Is h_2 always better than h_1?

Yes: for any vertex v, $h_2(v) \geq h_1(v)$

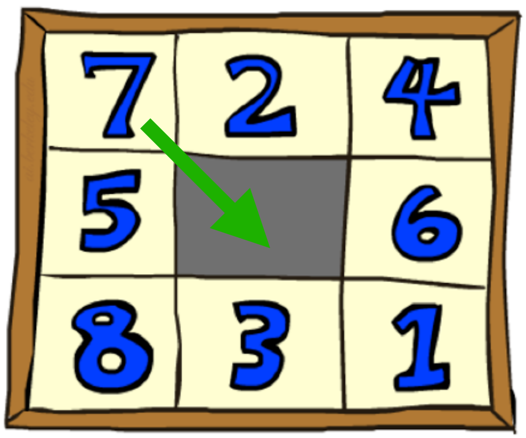
Assume C^* is the cost of the optimal solution path

Every vertex with $f(v) < C^*$ will be expanded

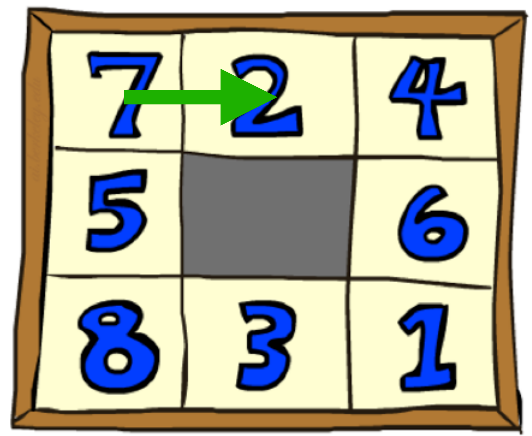
i.e., $h(v) < C^* - g(v)$ will be expanded

Every vertex expanded by h_2 will also be expanded by h_1, but h_1 may expand others as well

Relaxed Problems



What if a tile could move anywhere?



What if a tile could move one square in any direction — even onto an occupied square?

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Example: 8-puzzle

Original Problem:

A tile can move from square A to square B if
A is horizontally or vertically adjacent to B **and** B is blank

Can derive manhattan distance from this one

Three Relaxed Problems:

(a) A tile can move from square A to square B if A is adjacent to B

(b) A tile can move from square A to square B if B is blank

(c) A tile can move from square A to square B

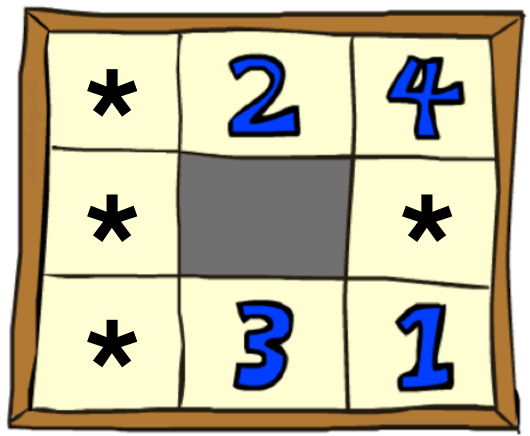
Which heuristic do we want?

If a collection of admissible heuristics $h_1, \ldots, h_m$ is available, but none of them dominates any of the others, which should we choose?

Use a composite heuristic that is most accurate on the vertex in question:

$$h(v) = \max\{h_1(v), \ldots, h_m(v)\}$$

Subproblems



The task is to get tiles 1, 2, 3, 4
into their correct positions

The cost of this subproblem is a lower bound on the cost of the complete problem.

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Pattern Databases

- Store the exact solution costs for every possible subproblem instance
- Compute an admissible heuristic h_{DB} for each complete state by looking up a corresponding subproblem configuration
- Don’t build all at once; add to DB for each new problem instance

Pattern Databases

5-6-7-8

2-4-6-8

...

Each database yields an admissible heuristic

Heuristics can be combined by taking the maximum value

Disjoint Pattern Databases

+

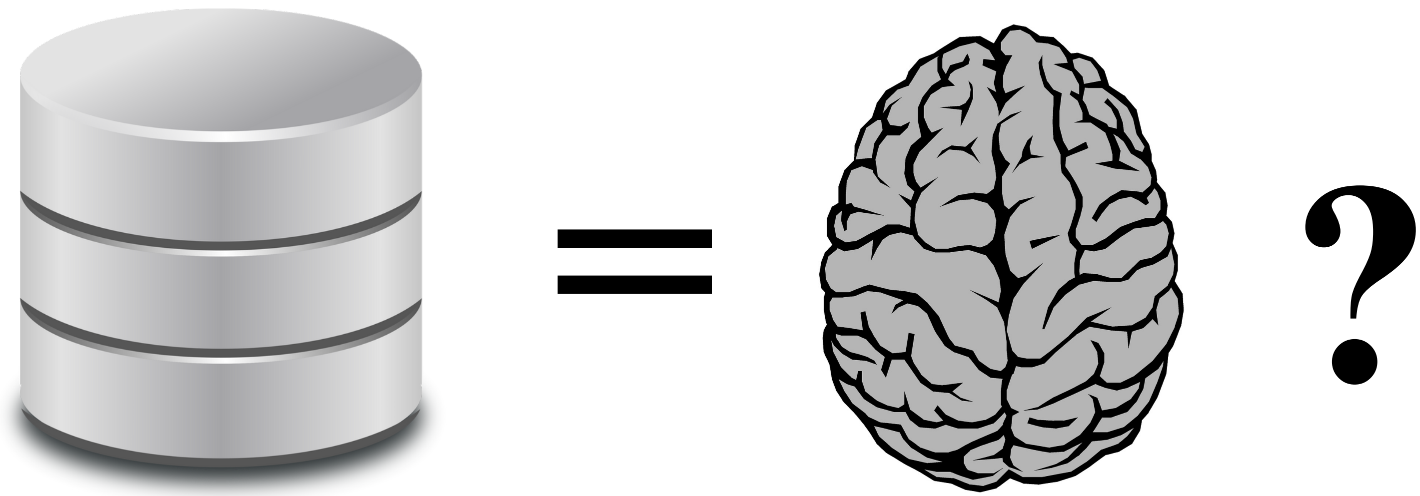
# of 1-2-3-4 moves

# of 5-6-7-8 moves

Sum is a lower bound on the cost of solving the entire problem

Speed-up achieved is several orders of magnitude

Pattern Databases

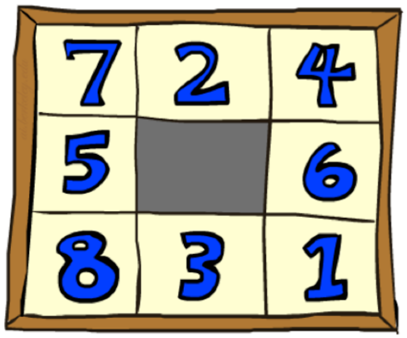


Rather shoddy strategy for modeling intelligence

Were the preceding strategies for coming up with heuristics good?

A better approach: learn from experience

Learning Heuristics From Experience



How can we do this with the 8-puzzle?

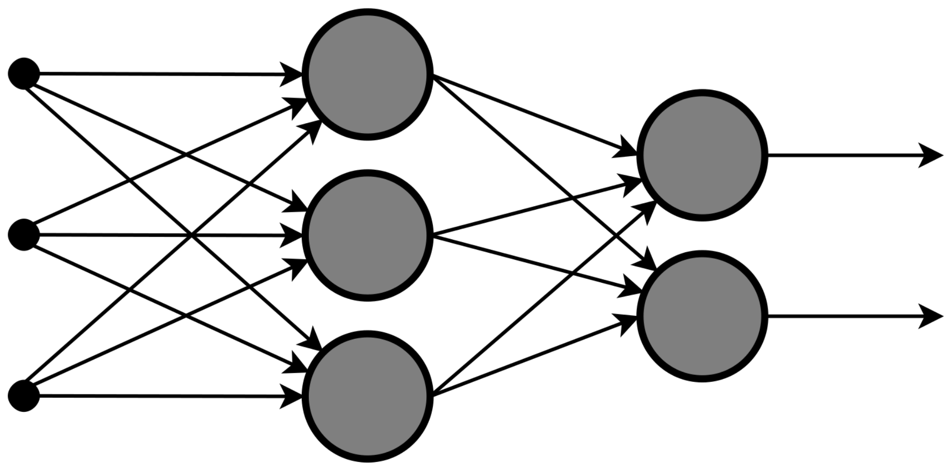
Solve a lot of 8-puzzles…

$h(v)$ can be learned from examples from optimal puzzle solutions

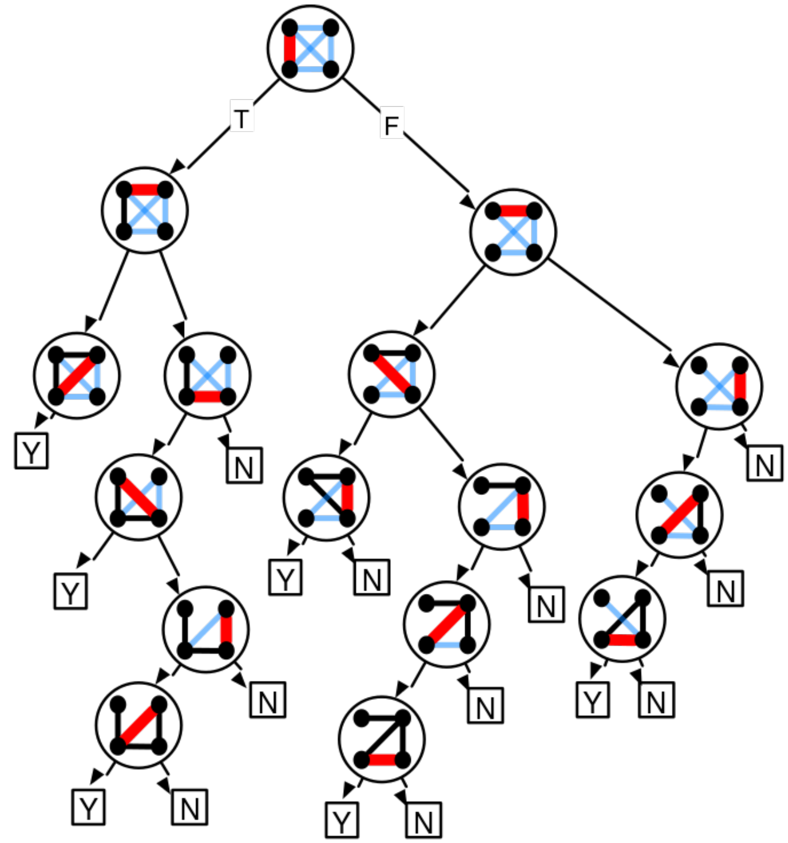
Each example consists of a state from the solution path and the cost of the solution from that point

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188

Applicable Learning Algorithms



Neural Networks



Decision tree for detecting a 3-clique in a 4-vertex graph

Decision Trees

Features

For search, learning works well when features are available that predict a state's value, rather than just a raw state description

Example for the 8-puzzle:

$x_1(v)$ = number of misplaced tiles

$x_2(v)$ = number of pairs of adjacent tiles that are not adjacent in the goal state

Combining Features

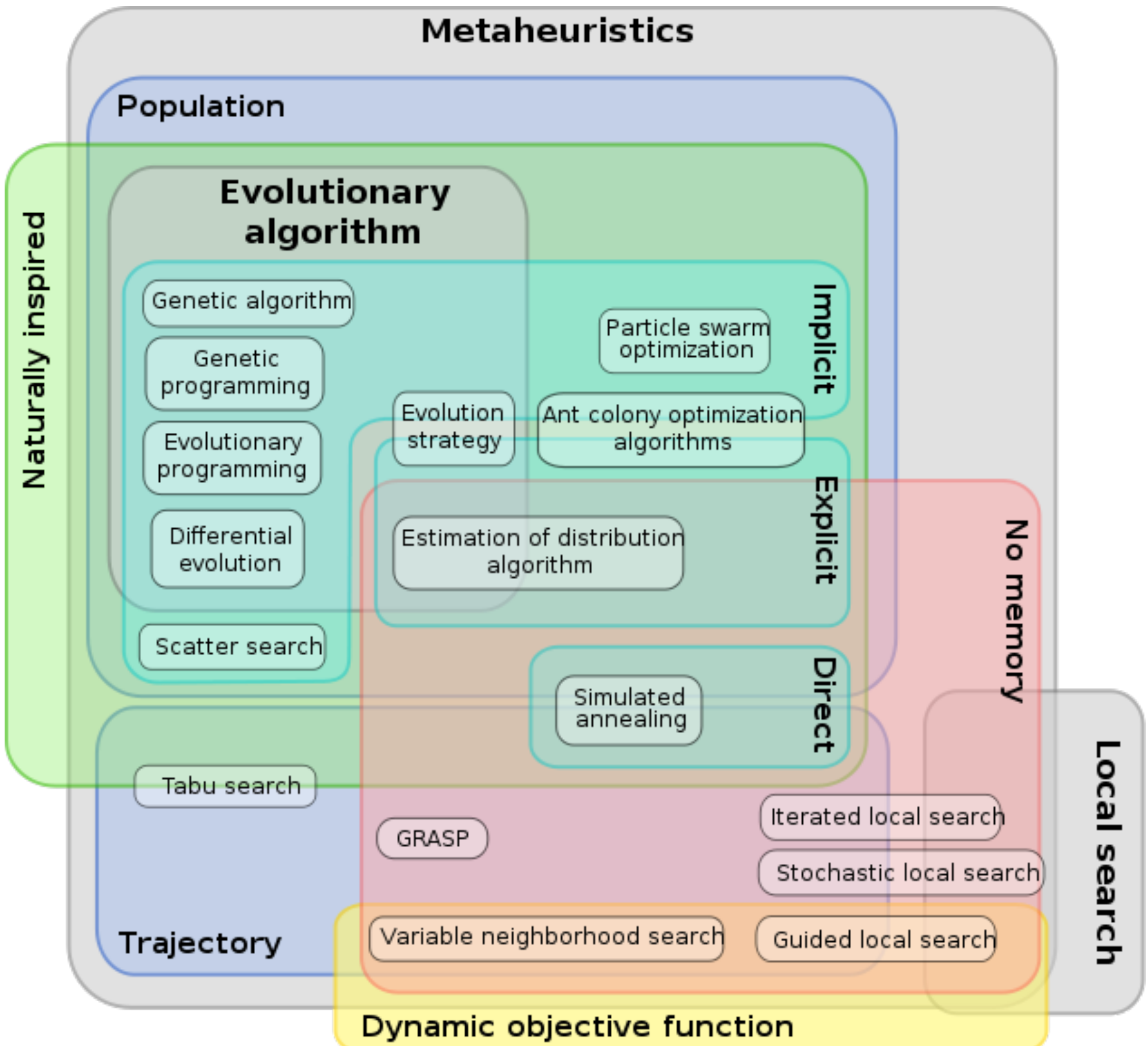
A common approach is to use a linear combination of features:

$$h(v) = \underbrace{c_1}_{\text{Constants}} x_1(v) + c_2 x_2(v)$$

Constants

c_1 and c_2 are adjusted to give the best fit to the underlying data on solution costs.

Lots of options for real problems



We will have a lot more to say about search and machine learning later…