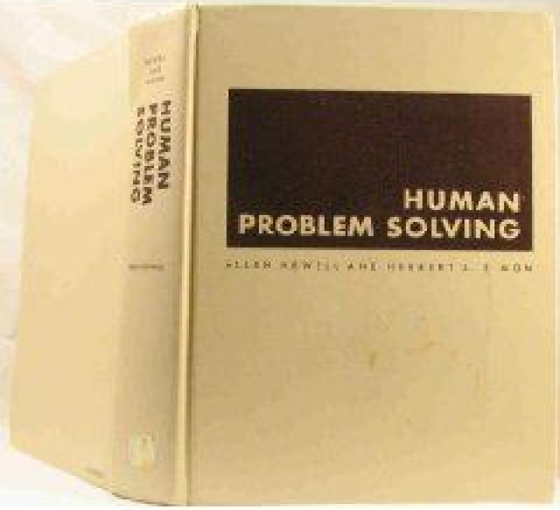
Introduction

Search is a central topic in AI.

Originated with Newell and Simon's work on problem solving; Human Problem Solving (1972).



Automated reasoning is a natural search task.

More recently: Given that almost all AI formalisms (planning, learning, etc) are NP-Complete or worse, some form of search is generally unavoidable (i.e., no smarter algorithm available).

Outline

Problem-solving agents

Problem types

Problem formulation

Example problems

Problem-solving agents

More details on “states” soon.

Problem solving agents are goal-directed agents:

1. Goal Formulation: Set of one or more (desirable) world states (e.g. checkmate in chess).
2. Problem formulation: What actions and states to consider given a goal and an initial state.
3. Search for solution: Given the problem, search for a solution --- *a sequence of actions to achieve the goal starting from the initial state*.
4. Execution of the solution

Note: Formulation feels somewhat “contrived,” but was meant to model very general (human) problem solving process.

Example: Path Finding problem

Formulate goal:

- be in Bucharest (Romania)
-

Formulate problem:

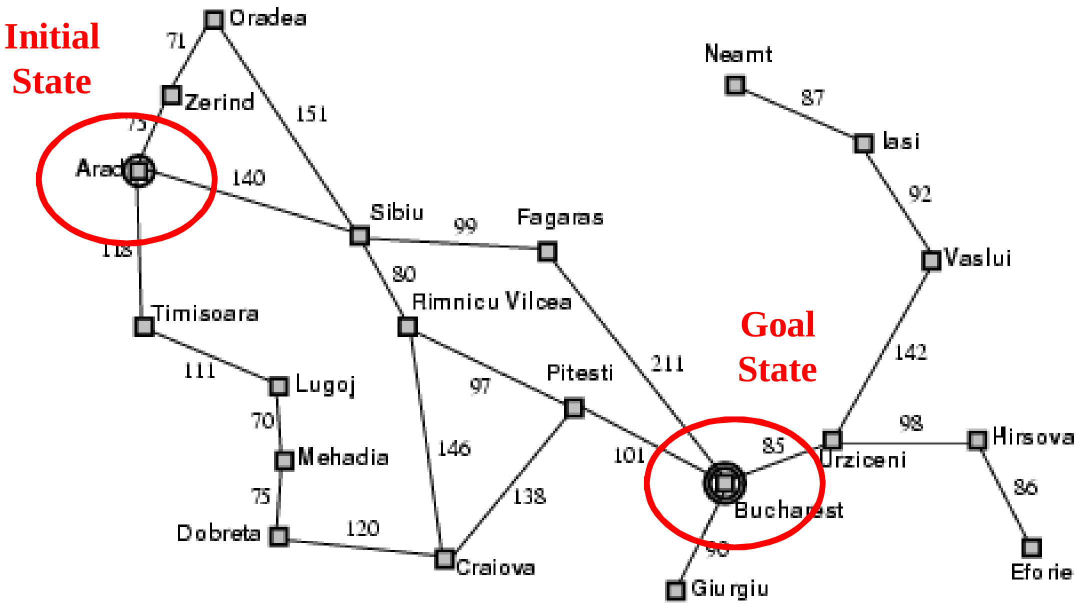
- **action**: drive between pair of connected cities (direct road)
-
- **state**: be in a city
  (20 world states)

Find solution:

- sequence of cities leading from start to goal state, e.g., **Arad, Sibiu, Fagaras, Bucharest**
-

Execution

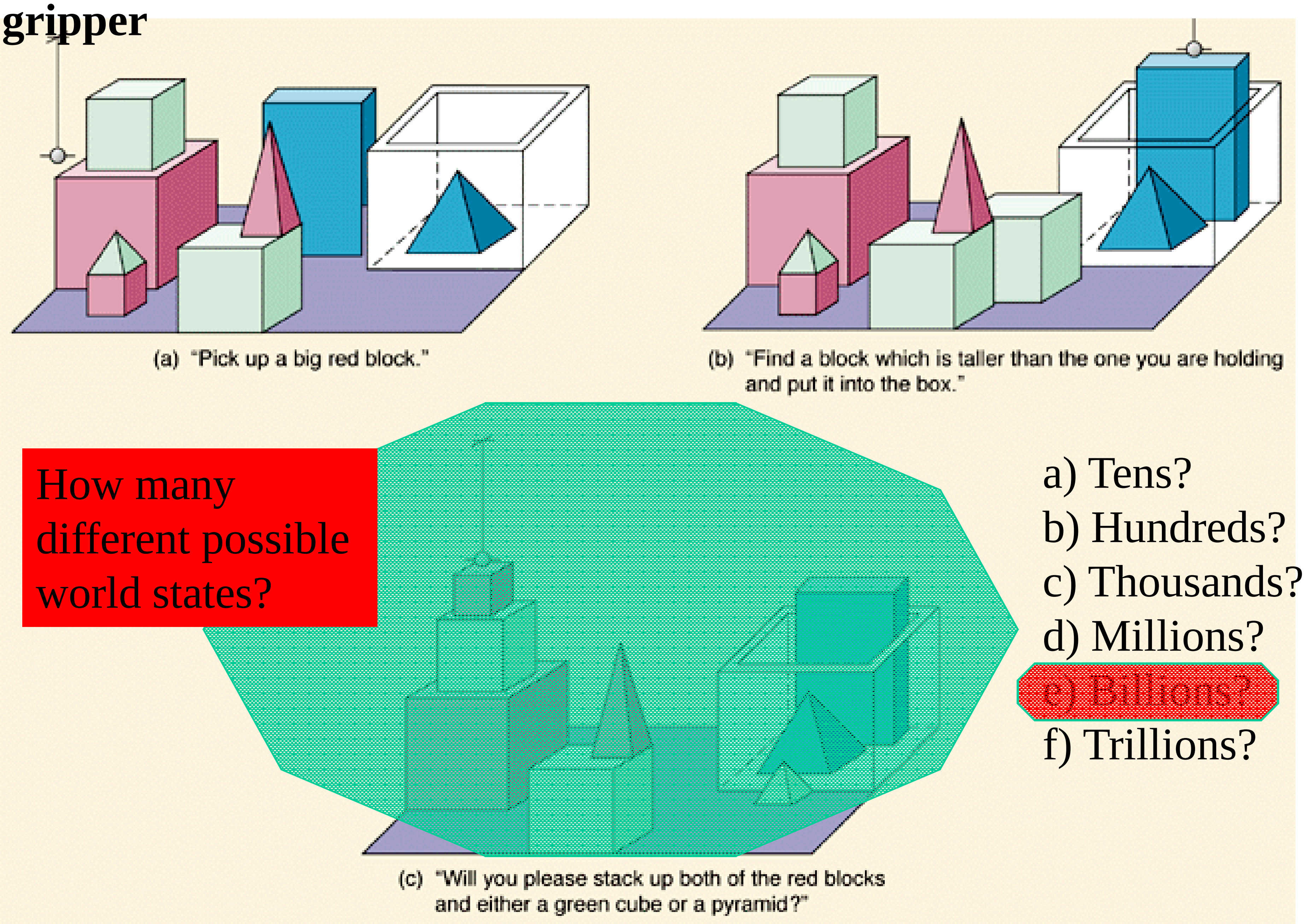
- drive from Arad to Bucharest according to the solution



Environment: fully observable (map), deterministic, and the agent knows effects of each action. Is this really the case?

Note: Map is somewhat of a "toy" example. Our real interest: ***Exponentially large spaces***, with e.g. 10^100 or more states. Far beyond full search. Humans can often still handle those!
One of the mysteries of cognition.

Micro-world: The Blocks World



(a) "Pick up a big red block."

(b) "Find a block which is taller than the one you are holding and put it into the box."

(c) "Will you please stack up both of the red blocks and either a green cube or a pyramid?"

How many different possible world states?

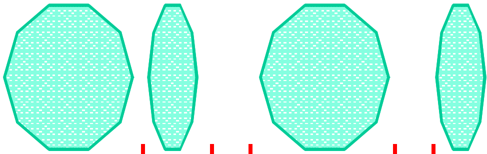
a) Tens?
b) Hundreds?
c) Thousands?
d) Millions?
e) Billions?
f) Trillions?

Size state space of blocks world example

n = 8 objects, k = 9 locations to build towers, one gripper. (One location in box.)
All objects distinguishable, order matter in towers. (Assume stackable in any order.)

Blocks: Use r-combinations approach from Rosen (section 5.5; CS-2800).

- - - - - - - - - - - - - - - - -   consider 16 = (n + k – 1) “spots”



Select k – 1 = 8 “dividers” to create locations,
(16 choose 8) ways to do this, e.g.,

| | - - - | - | | - - - | | - | Allocate n = 8 objs to remaining spots, 8! ways, e.g.,
| | 4 1 8 | 5 | | 6 3 7 | | 2 | assigns 8 objects to the 9 locations
a b c d e f g h i   based on dividers

So, total number of states (ignoring gripper): (16 choose 8) * 8! = 518,918,400
*** 9 for location gripper: > 4.5 billion states even in this toy domain!**
Search spaces grow exponentially with domain. *Still need to search them, e.g., to*

Problem types

Increasing complexity

1) Deterministic, fully observable

Agent knows exactly which state it will be in; solution is a sequence of actions.

2) Non-observable --- sensorless problem

- **Agent may have no idea where it is (no sensors); it reasons in terms of belief states; solution is a sequence actions (effects of actions certain).**

3) Nondeterministic and/or partially observable: contingency problem

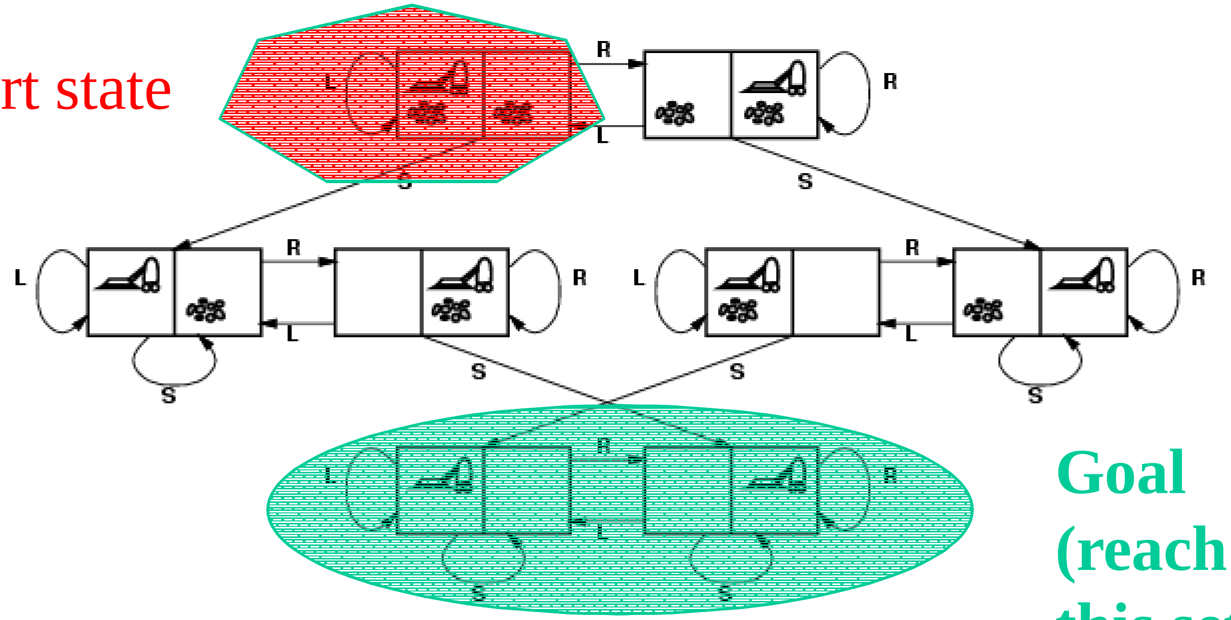
- **Actions uncertain, percepts provide new information about current state (adversarial problem if uncertainty comes from other agents).**
- **Solution is a “strategy” to reach the goal.**

4) Unknown state space and uncertain action effects: exploration problem

-- Solution is a “strategy” to reach the goal (end explore environment).

Example: Vacuum world state space graph

Start state



Goal
(reach one in
this set of states)

states? The agent is in one of 8 possible world states.

actions? *Left, Right, Suck [simplified: left out No-op]*

goal test? No dirt at all locations (i.e., in one of bottom two states).

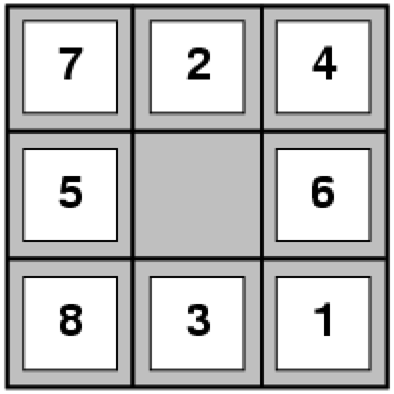
path cost? 1 per action

Minimum path from Start to Goal state: 3 actions

Alternative, longer plan: 4 actions

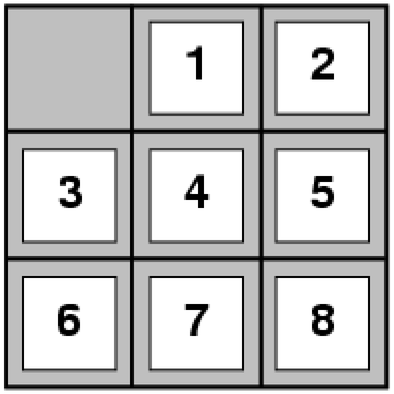
Note: path with thousands of steps before reaching goal also exist.

Example: The 8-puzzle
“sliding tile puzzle”



7	2	4
5		6
8	3	1

Start State



	1	2
3	4	5
6	7	8

Goal State

Aside:
variations
on goal state.
eg empty square
bottom right or
in middle.

states? the boards, i.e., locations of tiles

actions? move blank left, right, up, down

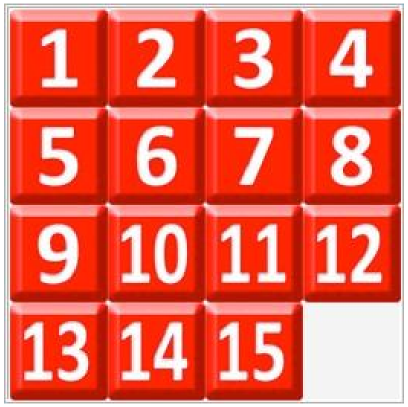
goal test? goal state (given; tiles in order)

path cost? 1 per move

Note: finding optimal solution of *n*-puzzle family is NP-hard!
Also, from certain states you can’t reach the goal.
Total number of states 9! = 362,880 (more interesting space;
not all connected… only half can reach goal state)

15-puzzle

Goal state

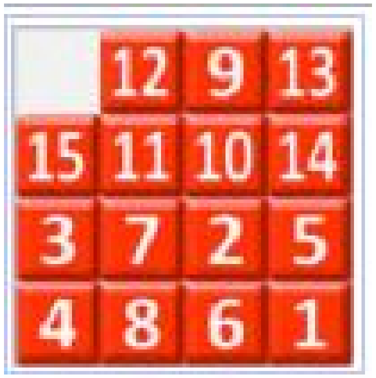


1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Search space:
16!/2 = 1.0461395 e+13, about 10 trillion.
Too large to store in RAM (>= 100 TB). A challenge to search for a path from a given board to goal.

Korf:
Disk errors
become a
problem.

Longest minimum path: 80 moves. Just 17 boards, e.g,



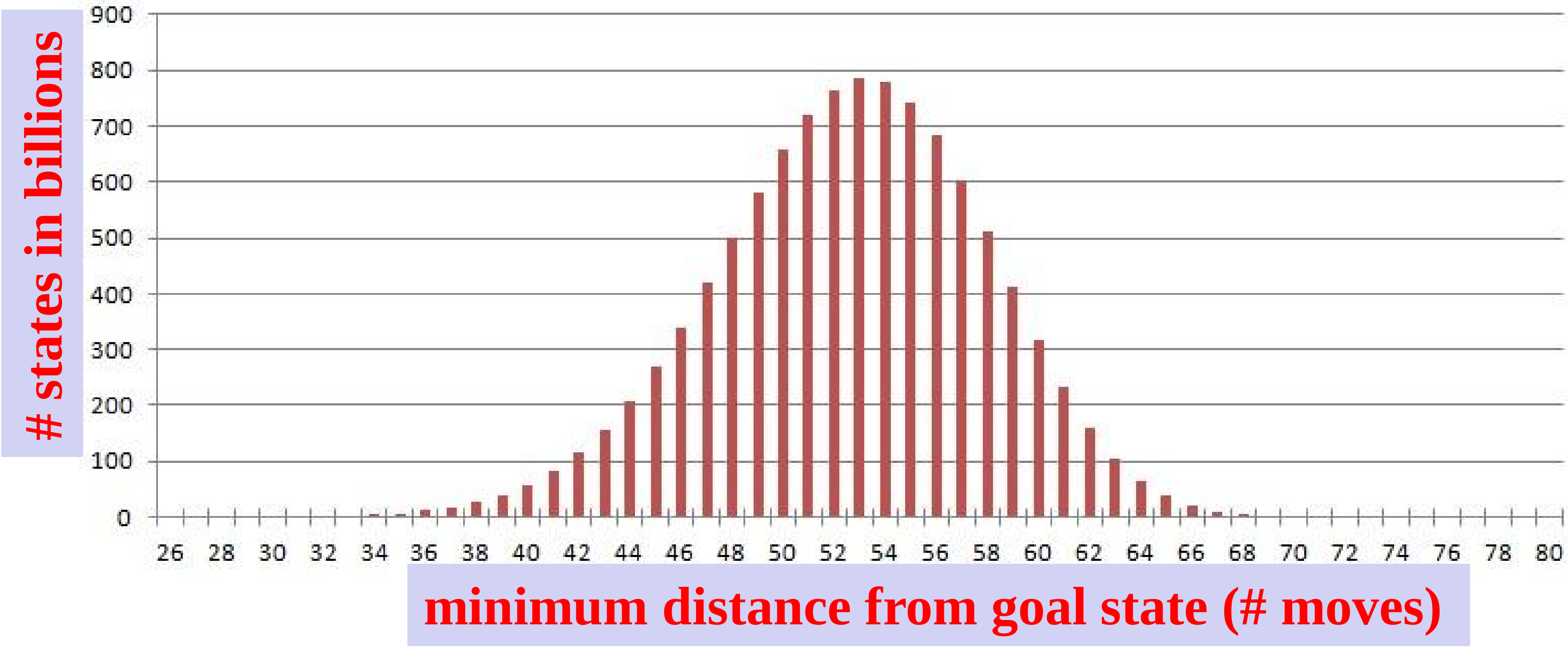
	12	9	13
15	11	10	14
3	7	2	5
4	8	6	1

Average minimum soln. length: 53.

People can find solns. But not necessarily minimum length. See solve it! (Gives strategy.)

Korf, R., and Schultze, P. 2005. Large-scale *parallel breadth-first search*. In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05). See Fifteen Puzzle Optimal Solver. With effective search: opt. solutions in seconds! Average: milliseconds.

Where are the 10 trillion states?

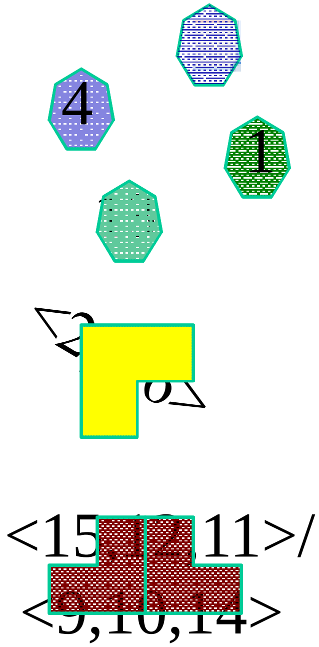


dist.	# states
0	1
1	2
2	4
3	10
4	24
5	54

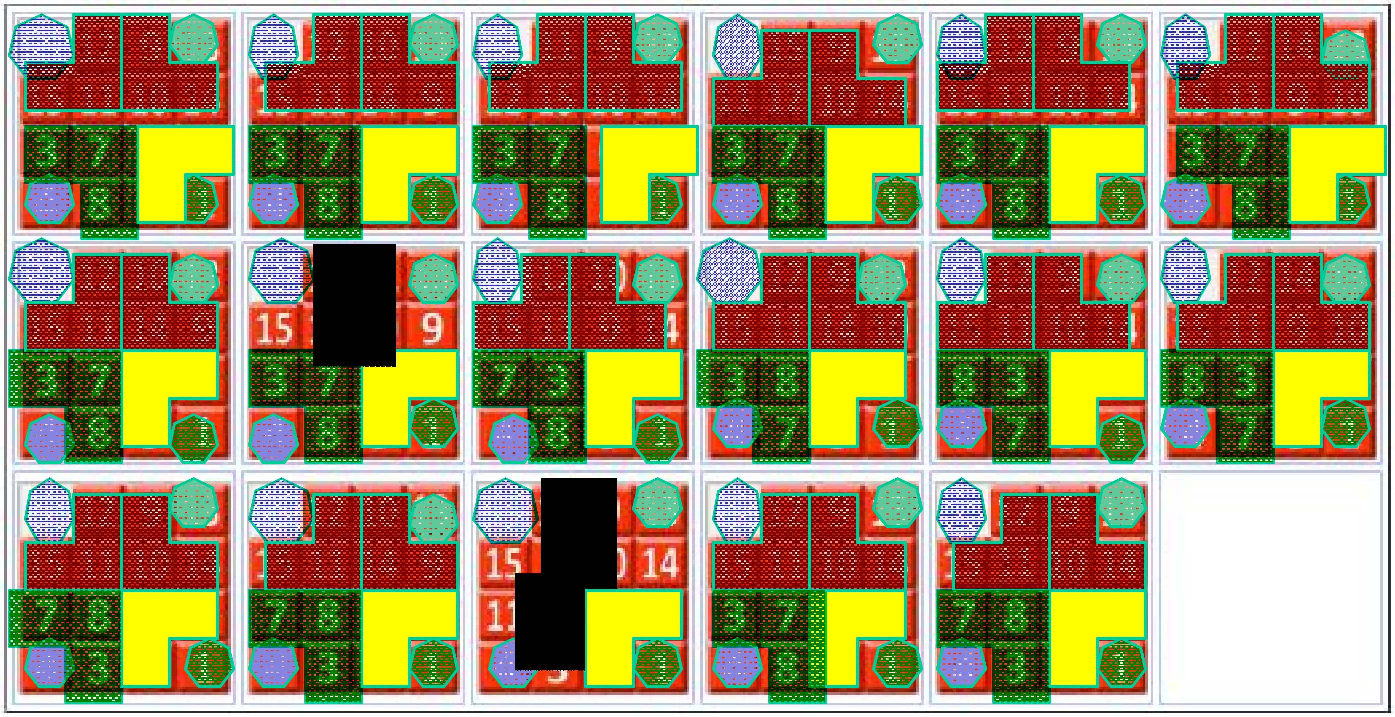
etc.

dist.	# states
76	272,198
77	26,638
78	3,406
79	70
80	17

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15



17 boards farthest away from goal state (80 moves)


[redacted]
[redacted]

What is it about these ***17 boards*** *out of over* ***10 trillion?***

Each require 80 moves to reach:

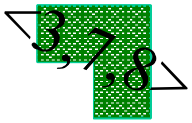
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Intriguing similarities. Each number has its own few locations.

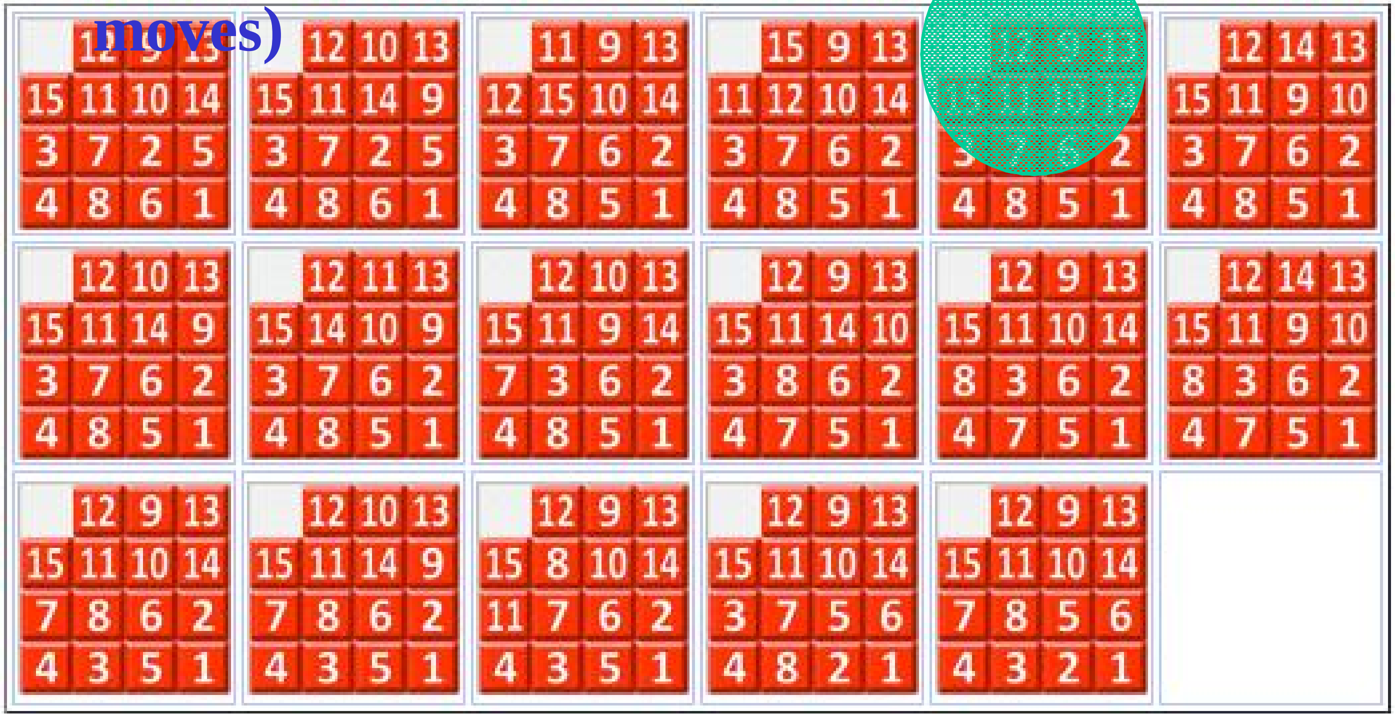
Interesting machine learning task:

Learn to recognize the hardest boards!

(Extremal Combinatorics, e.g. LeBras, Gomes, and Selman AAAI-12)



17 boards farthest away from goal state (80 moves)



	12	9	13
15	11	10	14
3	7	2	5
4	8	6	1

	12	10	13
15	11	14	9
3	7	2	5
4	8	6	1

	11	9	13
12	15	10	14
3	7	6	2
4	8	5	1

	15	9	13
11	12	10	14
3	7	6	2
4	8	5	1

	12	9	13
15	11	10	14
3	7	6	2
4	8	5	1

	12	14	13
15	11	9	10
3	7	6	2
4	8	5	1

	12	10	13
15	11	14	9
3	7	6	2
4	8	5	1

	12	11	13
15	14	10	9
3	7	6	2
4	8	5	1

	12	10	13
15	11	9	14
7	3	6	2
4	8	5	1

	12	9	13
15	11	14	10
3	8	6	2
4	7	5	1

	12	9	13
15	11	10	14
8	3	6	2
4	7	5	1

	12	14	13
15	11	9	10
8	3	6	2
4	7	5	1

	12	9	13
15	11	10	14
7	8	6	2
4	3	5	1

	12	10	13
15	11	14	9
7	8	6	2
4	3	5	1

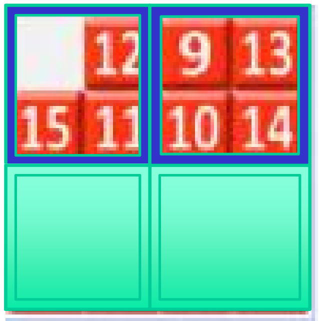
	12	9	13
15	8	10	14
11	7	6	2
4	3	5	1

	12	9	13
15	11	10	14
3	7	5	6
4	8	2	1

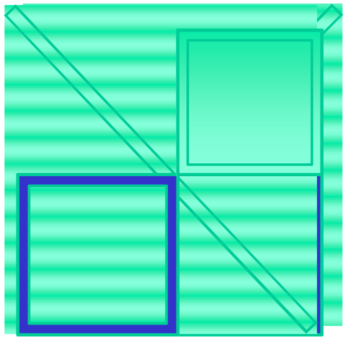
	12	9	13
15	11	10	14
7	8	5	6
4	3	2	1

Most regular extreme case:

Each quadrant reflected along diagonal. “move tiles furthest away”



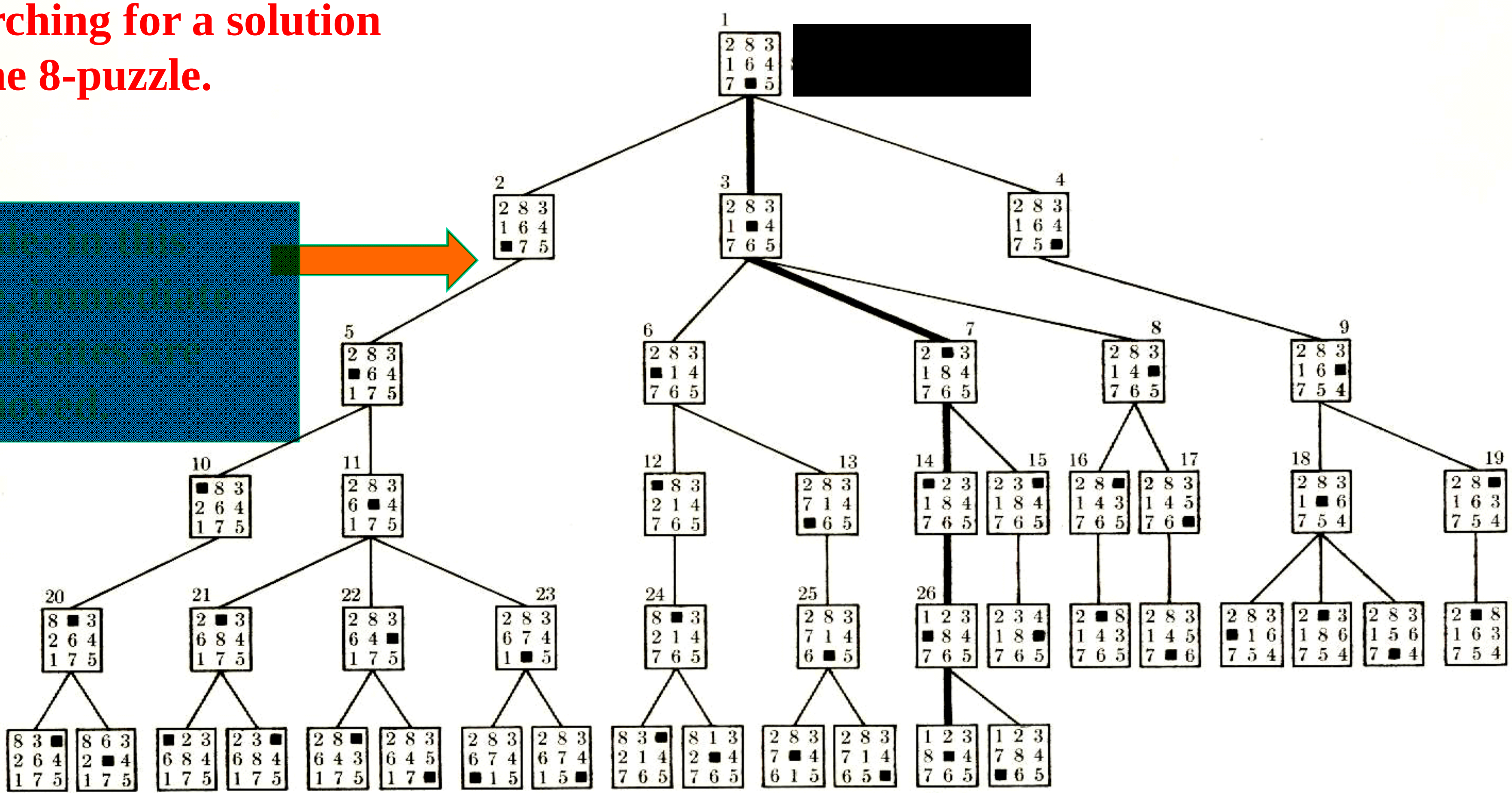
Goal state



Thanks to Jonathan GS

Searching for a solution to the 8-puzzle.

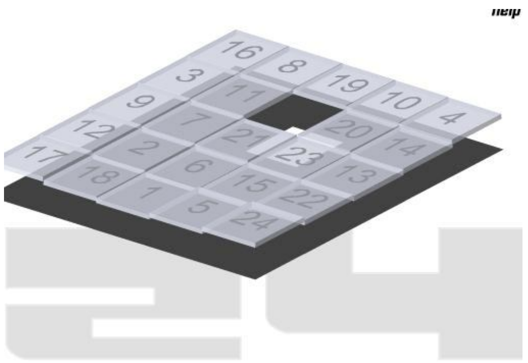
Aside: in this tree, immediate duplicates are removed.



[redacted]

[redacted]

Branching factor 1, 2, or 3 (max). So, approx. 2 --- # nodes roughly doubles at each level. *Number states of explored nodes grows exponentially with depth.*



For 15-puzzle, hard initial states: 80 levels deep, requires exploring approx. 2^80 ≈ 10^24 states.

If we block all duplicates, we get closer to 10 trillion (the number of distinct states: still a lot!).

Really only barely feasible on compute cluster with lots of memory and compute time. (Raw numbers for 24 puzzle, truly infeasible.)

Can we avoid generating all these boards? Do with much less search? (Key: bring average branching factor down.)

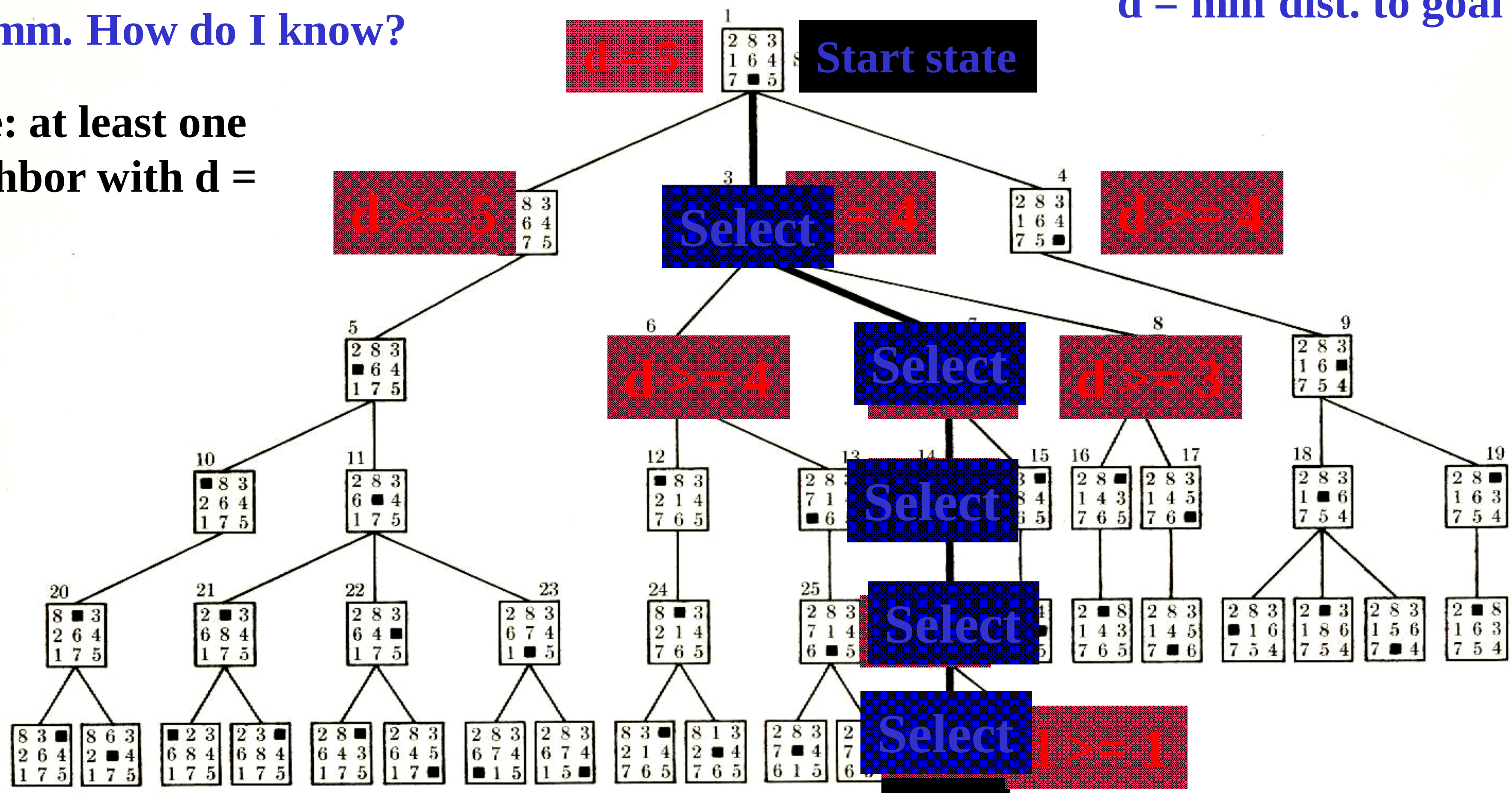
Gedanken experiment: Assume that you knew for each state, the minimum number of moves to the final goal state. (Table too big, but assume there is some formula/algorithm based on the board pattern that gives this number for each board and quickly.)

Using the minimum distance information, is there a clever way to find a minimum length sequence of moves leading from the start state to the goal state? How?

Hmm. How do I know?

d = min dist. to goal

Note: at least one neighbor with d = 4.



A breadth-first search tree. (More detail soon.)

Branching factor approx. 2. So, with “distance oracle” we only need to explore approx. 2 * (min. solution length).

For 15-puzzle, hard initial states: 80 levels deep, requires exploring approx. 2^80 ≈ 10^24 states.

But, with distance oracle, we only need to explore roughly 80 * 2 = 160 states! (only linear in size of solution length)

We may not have the exact distance function ("perfect heuristics"), but

we can still "guide" the search using an approximate distance function.

This is the key idea behind "heuristic search" or "knowledge-based search."

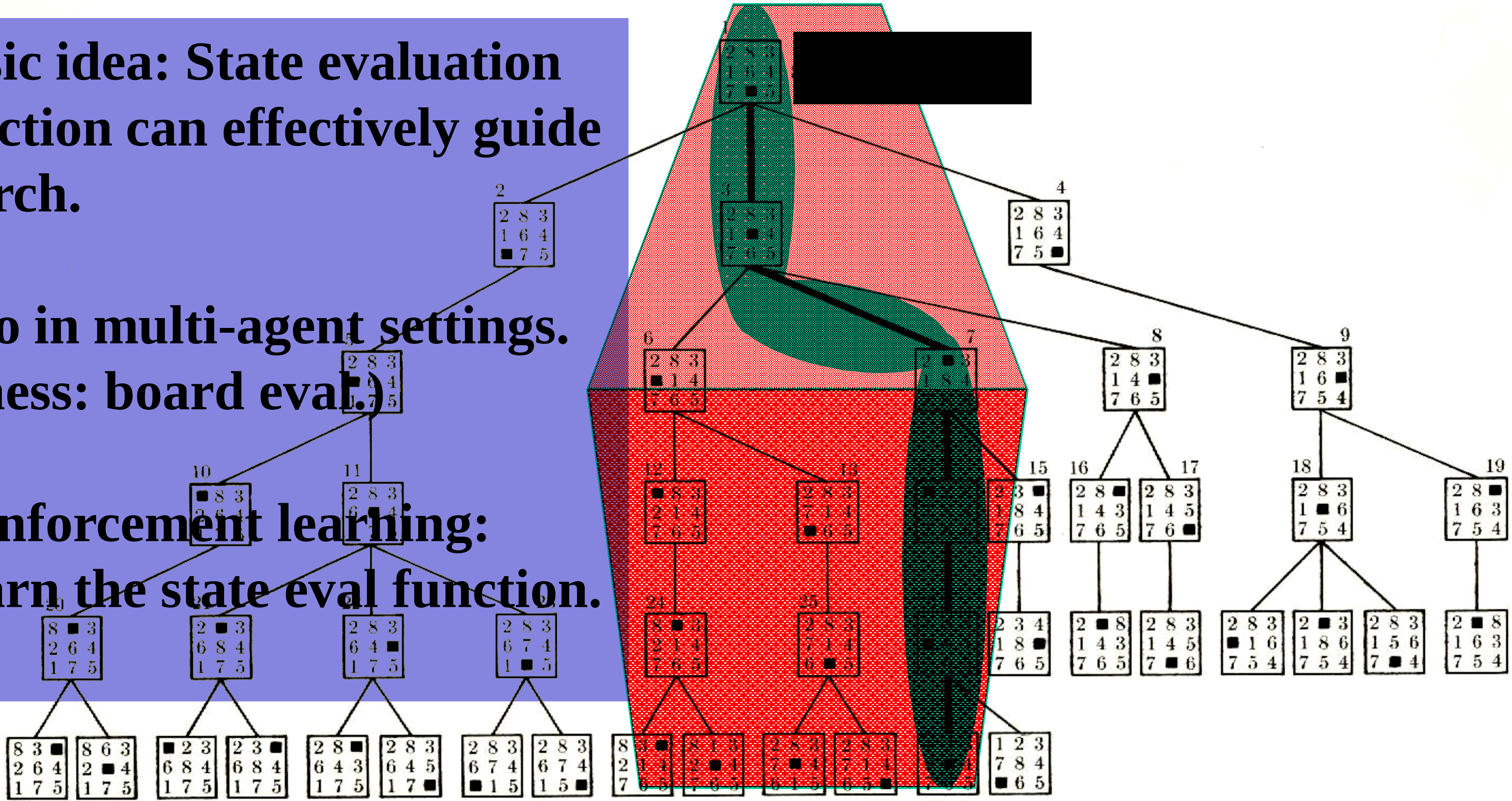
We use knowledge / heuristic information about the distance to the goal to

guide our search process. We can go from exponential to polynomial or even

Basic idea: State evaluation function can effectively guide search.

Also in multi-agent settings. (Chess: board eval.)

Reinforcement learning: Learn the state eval function.



[redacted]

[redacted]

Perfect "heuristics," eliminates search.

Approximate heuristics, significantly reduces search.
Best (provably) use of search heuristic info: A* search (soon).

State evaluation functions or "heuristics"

Provide guidance in terms of what action to take next.

General principle: Consider all neighboring states, reachable via some action. Then select the action that leads to the state with the highest utility (evaluation value). This is a fully greedy approach.

Because eval function is often only an estimate of the true state value, greedy search may not find the optimum path to the goal.

By adding some search with certain guarantees on the approximation, we can still get optimal behavior (A* search) (i.e. finding the optimal path to the solution). Overall result: generally exponentially less search required.

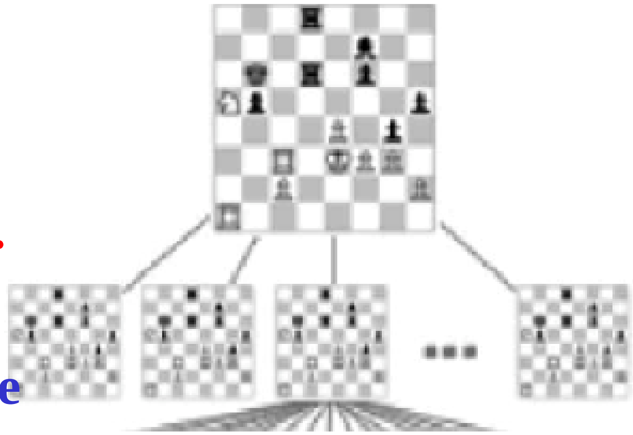
N-puzzle heuristics ("State evaluation function" wrt to goal to be reached):

1) Manhattan Distance: For each tile the number of grid units between its current location and its goal location are counted and the values for all tiles are summed up. (underestimate; too "loose"; not very powerful)

2) Felner, Ariel, Korf, Richard E., Hanan, Sarit, Additive Pattern Database Heuristics, Journal of Artificial Intelligence Research 22 (2004) 279-318. The 78 Pattern Database heuristic takes a lot of memory but solves a random instance of the 15-puzzle within a few milliseconds on average. An optimal solution for the 80 moves cases takes a few seconds each. So, thousands of nodes considered instead of many billions.

Note: many approx. heuristics ("conservative" / underestimates to goal) combined with search can still find optimal solutions.

State evaluation function (or utility value) is a very general and useful idea.



Example:

- In chess, given a board, what would be the perfect evaluation value that you would want to know? (Assume the perspective of White player.)

A: f(board) → {+1, 0, -1}, with +1 for guaranteed win for White,
0 draw under perfect play, and
-1 loss under perfect play.

Perfect play: all powerful opponent.

Given f, how would you play then?

In practice, we only know (so far) of an approximation of f.

f(board) → [-1,+1]   (interval from -1 to +1)

based on “values” of chess pieces, e.g., pawn 1 point, rook 5 points.

State evaluation function (or utility value) is a very general and useful idea.

Examples:

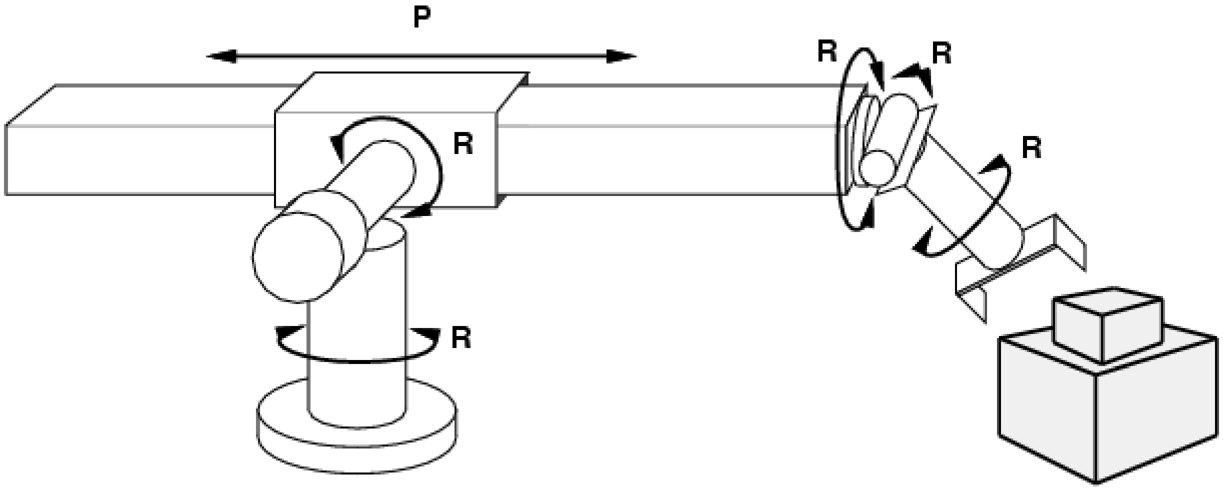
- **TD-Gammon backgammon player. Neural net was trained to find approximately optimal state (board) evaluation values (range [-1,+1]). (Tesauro 1995)**

- **“Robocopter” --- automated helic trained state evaluation function. State given by features, such as, position, orientation, speed, and rotors position and speed. Possibl actions change rotors speed and position. Evaluation assigns value in [-1,+1] to capture stability.**

© Reuters

(Abbeel, Coates, and Ng 2008)

Example: Robotic assembly



states?: real-valued coordinates of robot joint angles
parts of the object to be assembled

actions?: continuous motions of robot joints

goal test?: complete assembly

path cost?: time to execute

Other example search tasks

VLSI layout: positioning millions of components and connections on a chip to minimize area, circuit delays, etc.

Robot navigation / planning

Automatic assembly of complex objects

Protein design: sequence of amino acids that will fold into the 3-dimensional protein with the right properties.

Literally thousands of combinatorial search / reasoning / parsing / matching problems can be formulated as search problems in exponential size state spaces.