



CMU 15-781

Lecture 2:

Uninformed Search

Teachers:

Emma Brunskill

Ariel Procaccia (this time)

SEARCH PROBLEMS

- A search problem has:
 - States (configurations)
 - Start state and goal states
 - Successor function: maps states to (action, state, cost) triples



EXAMPLE: PANCAKES

Discrete Mathematics 27 (1979) 47-57.
© North-Holland Publishing Company

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

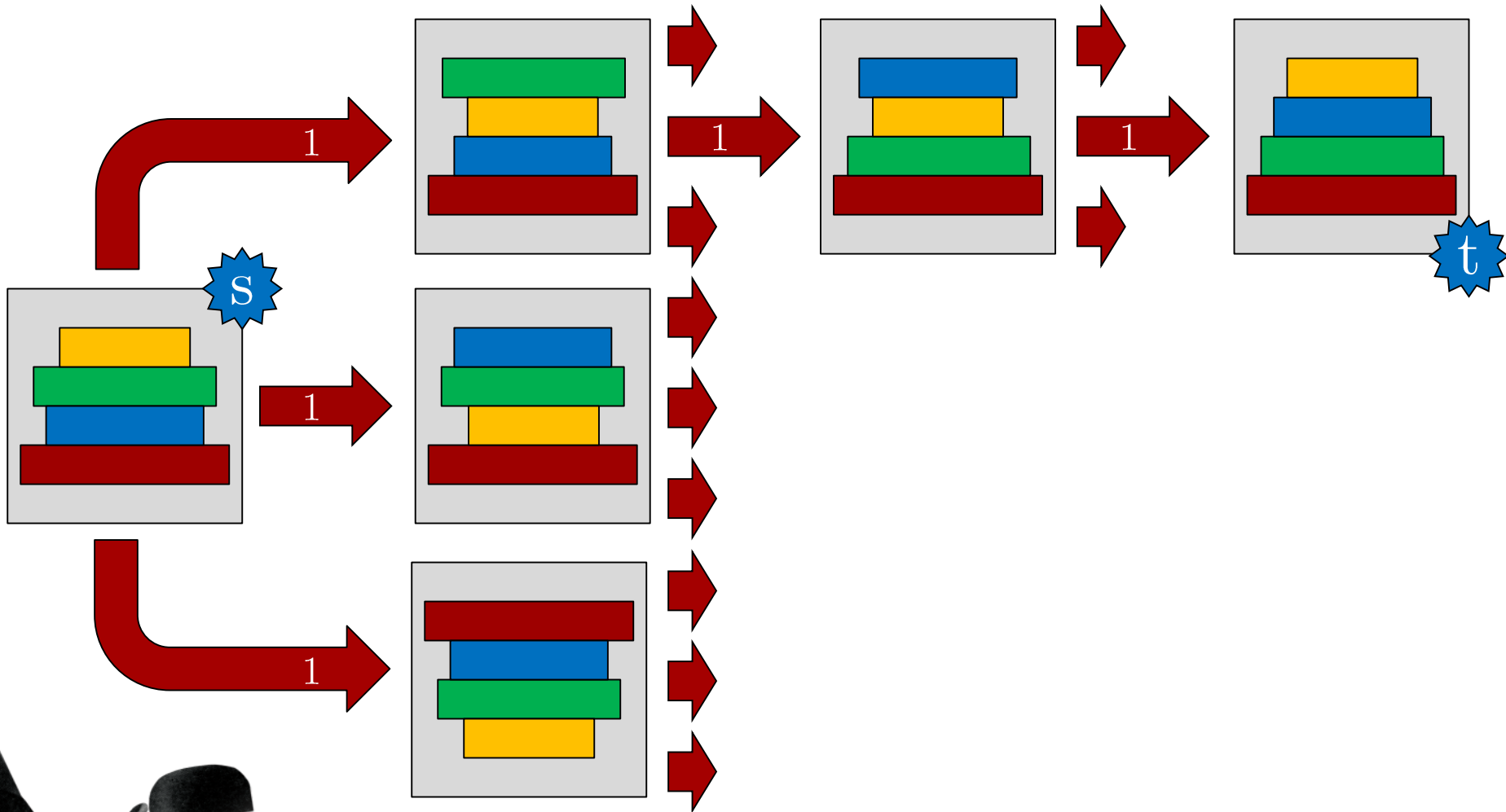
Received 18 January 1978

Revised 28 August 1978

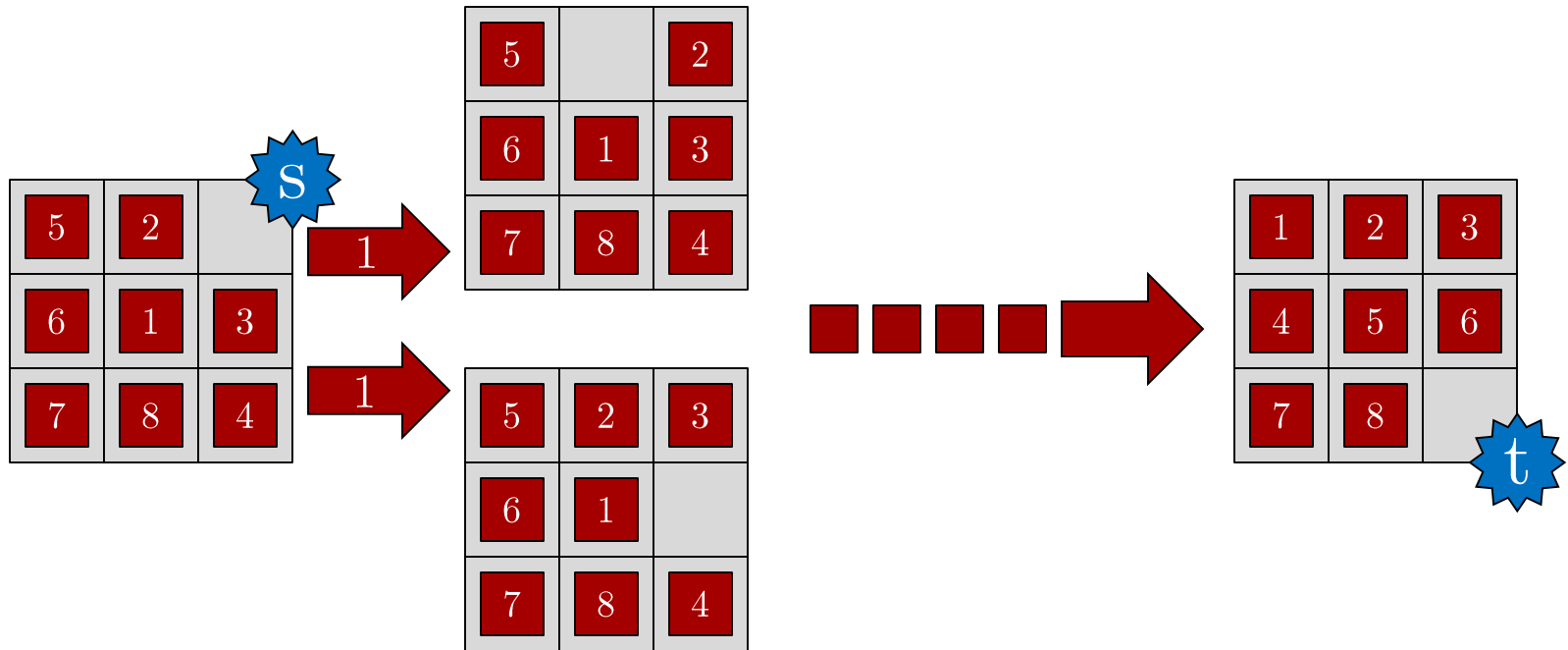
For a permutation σ of the integers from 1 to n , let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n . We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.



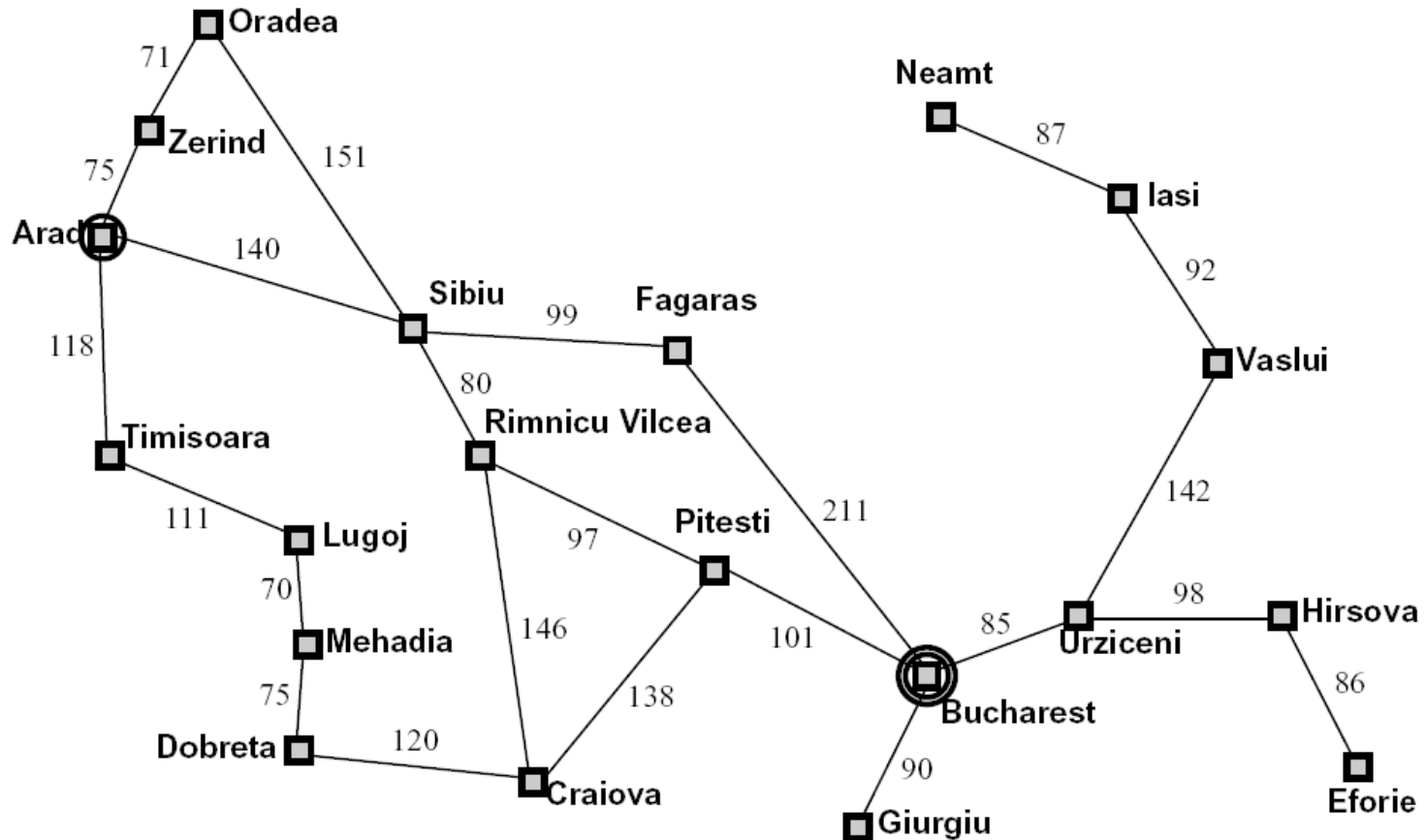
EXAMPLE: PANCAKES



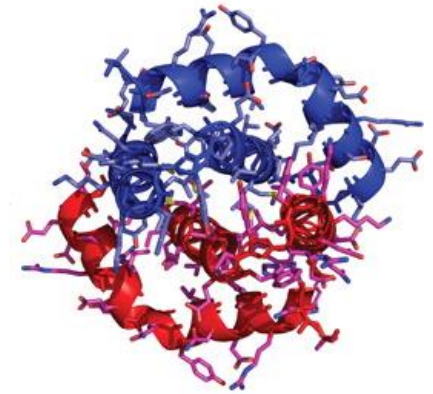
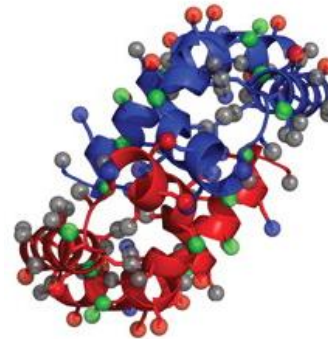
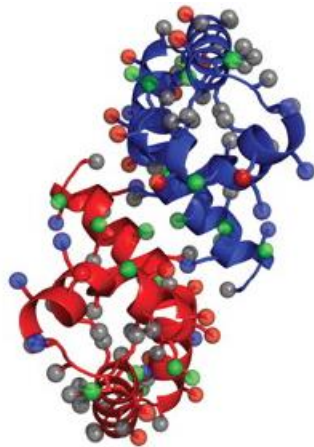
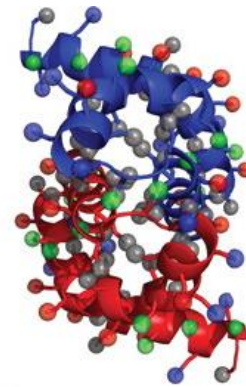
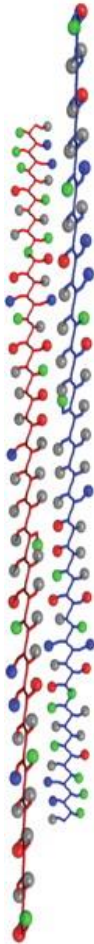
EXAMPLE: 8-PUZZLE



EXAMPLE: PATHFINDING



EXAMPLE: PROTEIN FOLDING



TREE SEARCH

function TREE-SEARCH(**problem**, **strategy**)

set of frontier nodes contains the start state of **problem**
loop

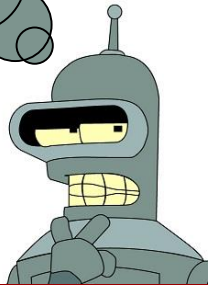
- **if** there are no frontier nodes **then return** failure
- choose a frontier node for expansion using **strategy**
- **if** the node contains a goal **then return** the corresponding solution
- **else** expand the node and add the resulting nodes to the set of frontier nodes



TREE SEARCH

- Tree search can expand the same nodes again and again
- In a rectangular grid:
 - Search tree of depth d has 4^d leaves
 - There are only $4d$ states within d steps of any given state

Algorithms that forget their history are doomed to repeat it!



GRAPH SEARCH

function GRAPH-SEARCH(**problem**, **strategy**)

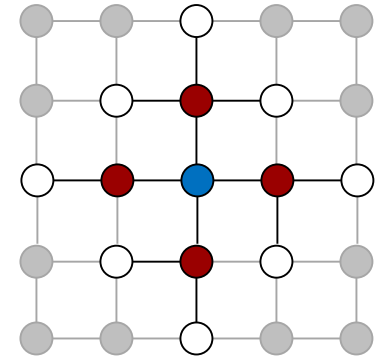
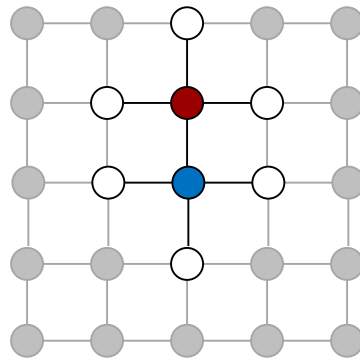
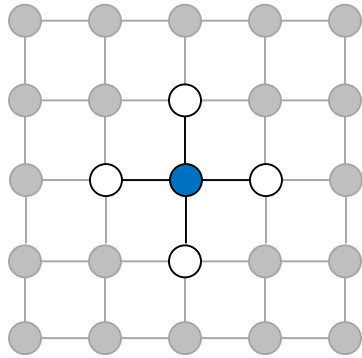
set of frontier nodes contains the start state of **problem**

loop

- **if** there are no frontier nodes **then return** failure
- choose a frontier node for expansion using **strategy**, **and add it to the explored set**
- **if** the node contains a goal **then return** the corresponding solution
- **else** expand the node and add the resulting nodes to the set of frontier nodes, **only if not in the frontier or explored set**



GRAPH SEARCH ILLUSTRATED



Separation property: Every path from initial state to an unexplored state has to pass through the frontier

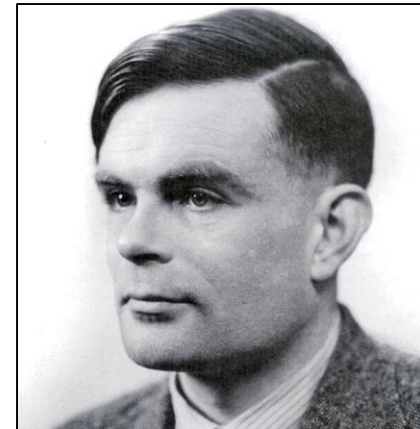


UNINFORMED VS. INFORMED



Uninformed

Can only generate successors and distinguish goals from non-goals



Informed

Strategies that know whether one non-goal is more promising than another

MEASURING PERFORMANCE



Completeness

Guaranteed to find a solution when there is one?



Optimality

Finds the cheapest solution?



Time

How long does it take to find a solution?



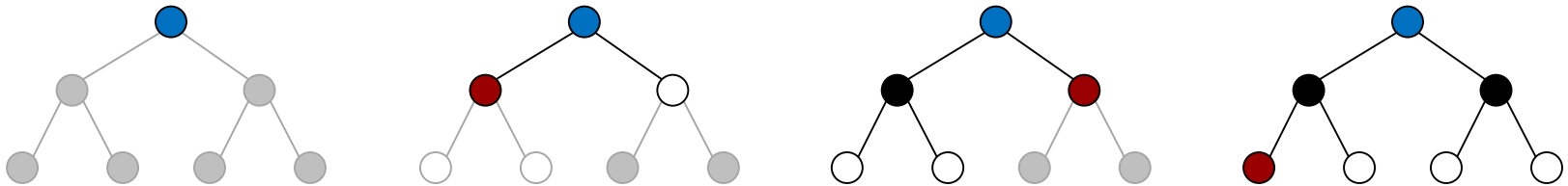
Space

How much memory is needed to perform the search?



BREADTH-FIRST SEARCH

- **Strategy:** Expand **shallowest** unexpanded node
- Can be implemented by using a FIFO queue for the frontier
- Goal test applied when node is generated



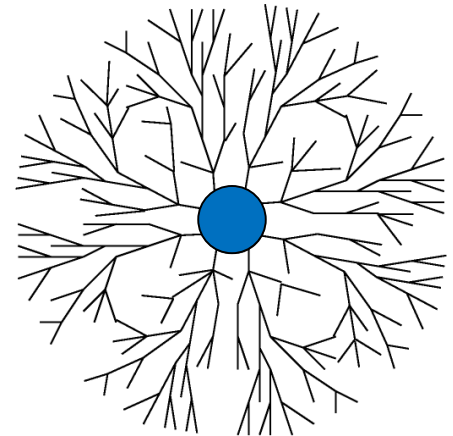
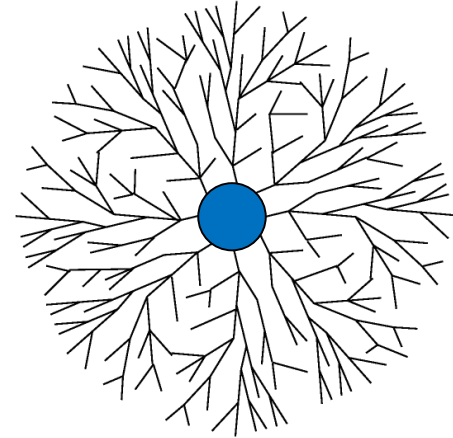
BREADTH-FIRST SEARCH

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Not really	$\Theta(b^d)$	$\Theta(b^d)$

- **Optimality:** If the path cost is a nondecreasing function of the depth (e.g., all actions have the same cost)
- **Time complexity:** Imagine each node has b successors, and solution is at depth d , then generate $\sum_{i=1}^d b^i = \Theta(b^d)$ nodes
- **Space complexity:** Essentially the same as time complexity

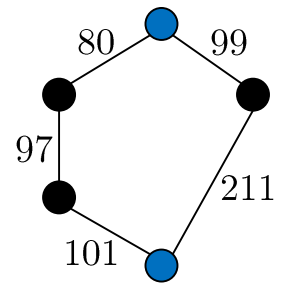
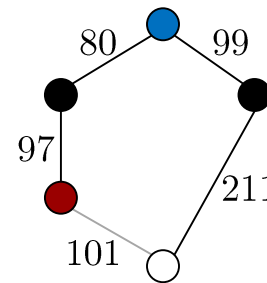
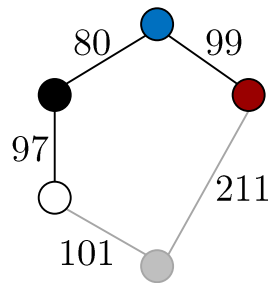
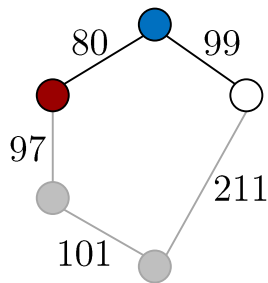
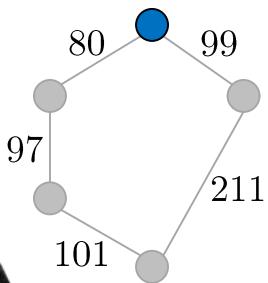
BIDIRECTIONAL SEARCH

- **Idea:** Possibly improve the running time of BFS by running two simultaneous searches, forward from the initial state and backward from the goal
- **Poll 1:** What is the worst-case running time of BIDIRECTIONAL SEARCH?
 1. $\Theta(b \cdot d)$
 2. $\Theta((b/2)^d)$
 3. $\Theta(b^{d/2})$
 4. $\Theta(b^d)$



UNIFORM-COST SEARCH

- **Strategy:** Expand unexpanded node with lowest path cost $g(n)$
- Can be implemented by using a priority queue ordered by $g(n)$ for the frontier
- Other changes from BFS:
 - Goal test applied when node is selected for expansion
 - Need to update cost of nodes on frontier



UNIFORM-COST SEARCH

Algorithm	Complete?	Optimal?	Time	Space
UCS	Sorta	Yes	$\Theta(b^{1+\lceil C^*/\epsilon \rceil})$	$\Theta(b^{1+\lceil C^*/\epsilon \rceil})$

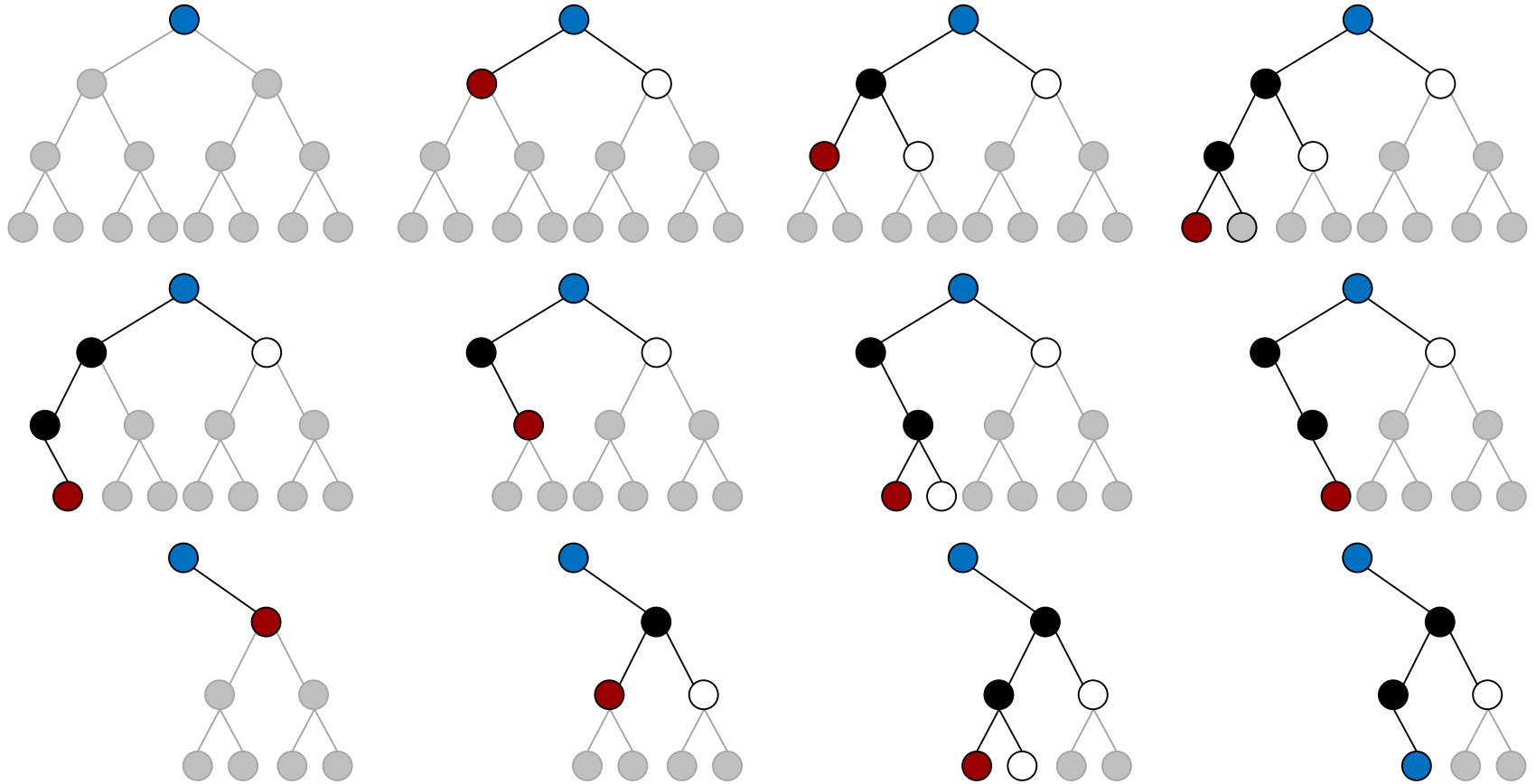
- **Optimality:** Easy to see
- **Completeness:** If the cost of every step exceeds $\epsilon > 0$
- **Time complexity:** If C^* is the cost of the optimal solution and ϵ is a lower bound on the step size, the depth of the search tree is $1 + \lceil C^*/\epsilon \rceil$
- The complexity is $\Theta(b^{d+1})$ when step costs are uniform

DEPTH-FIRST SEARCH

- **Strategy:** Expand **deepest** unexpanded node
- Can be implemented by using a stack for the frontier
- Recursive implementation is also common



DEPTH-FIRST SEARCH



DEPTH-FIRST SEARCH

Algorithm	Complete?	Optimal?	Time	Space
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$

- **Completeness:** Clearly not in general
- **Poll 2:** In a finite state space, which version of DFS is complete?
 1. TREE SEARCH
 2. GRAPH SEARCH
 3. Both
 4. Neither

DEPTH-FIRST SEARCH

Algorithm	Complete?	Optimal?	Time	Space
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$

- **Time complexity:** $\Theta(b^m)$, where m is the maximum depth of any solution
- **Space complexity:** DFS tree search needs to store only a single path from the root to a leaf, along with unexpanded sibling nodes for each node on the path
- Consequently, depth-first tree search is the workhorse of many areas of AI (including CSPs and SAT solving)

ITERATIVE DEEPENING SEARCH

Algorithm	Complete?	Optimal?	Time	Space
IDS	Yes	No	$\Theta(b^d)$	$\Theta(b \cdot d)$

- Run DFS with depth limit $\ell = 1, 2, \dots$
- Combines the best properties of BFS and DFS
- **Completeness:** Yes, for the same reason BFS is complete
- **Time complexity:** Seems wasteful but most of the nodes are at the bottom level; total

$$d \cdot b + (d - 1)b^2 + \dots + 1 \cdot b^d = \Theta(b^d)$$

SUMMARY OF ALGORITHMS

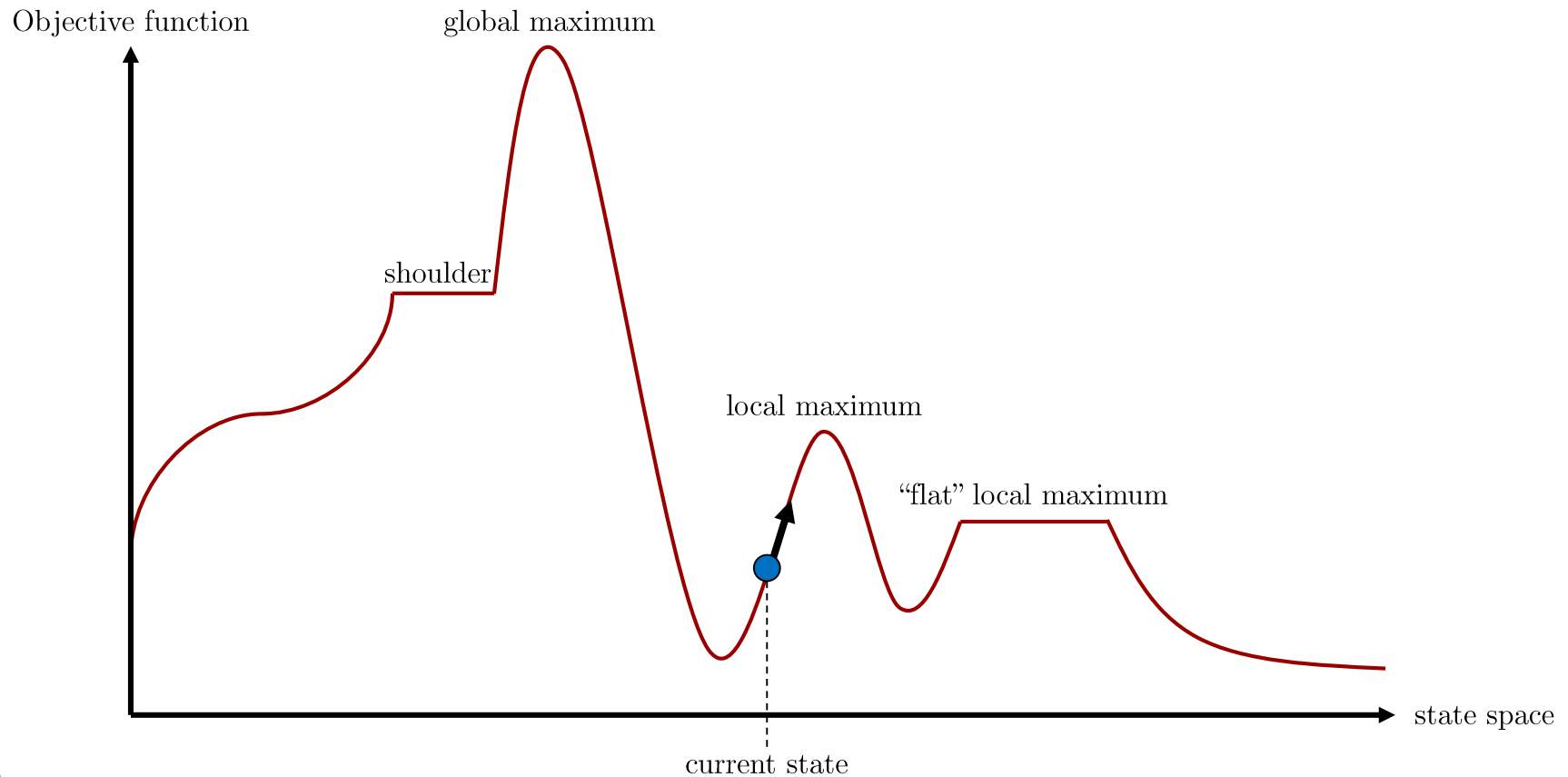
Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Not really	$\Theta(b^d)$	$\Theta(b^d)$
UCS	Sorta	Yes	$\Theta(b^{1+\lceil C^*/\epsilon \rceil})$	$\Theta(b^{1+\lceil C^*/\epsilon \rceil})$
DFS	No	No	$\Theta(b^m)$	$\Theta(b \cdot m)$
IDS	Yes	No	$\Theta(b^d)$	$\Theta(b \cdot d)$

OPTIMIZATION AND LOCAL SEARCH

- The algorithms we discussed so far are designed to find a path to the solution
- If the path doesn't matter, can use **local search algorithms** that consider a single current node, and move to one of its neighbors in the next step
- Local search algorithms are useful for **optimization problems**, where the goal is to find the best state according to an **objective function**



STATE SPACE LANDSCAPE











HILL-CLIMBING SEARCH









- Move in the direction of increasing value (up the hill)
- Terminate when no neighbor has higher value



HILL-CLIMBING SEARCH

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

State with 17 conflicts, showing the #conflicts by moving a queen within its column, with best moves in red

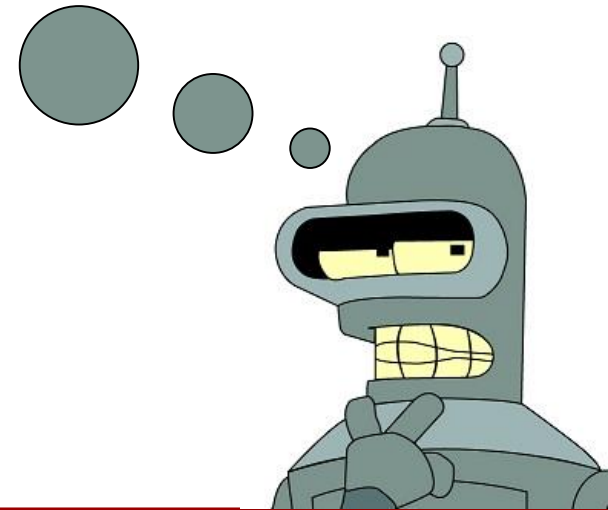
						
						
						
						
						
						
						
						

Local optimum: state that has only one conflict, but every move leads to larger #conflicts

HILL-CLIMBING SEARCH

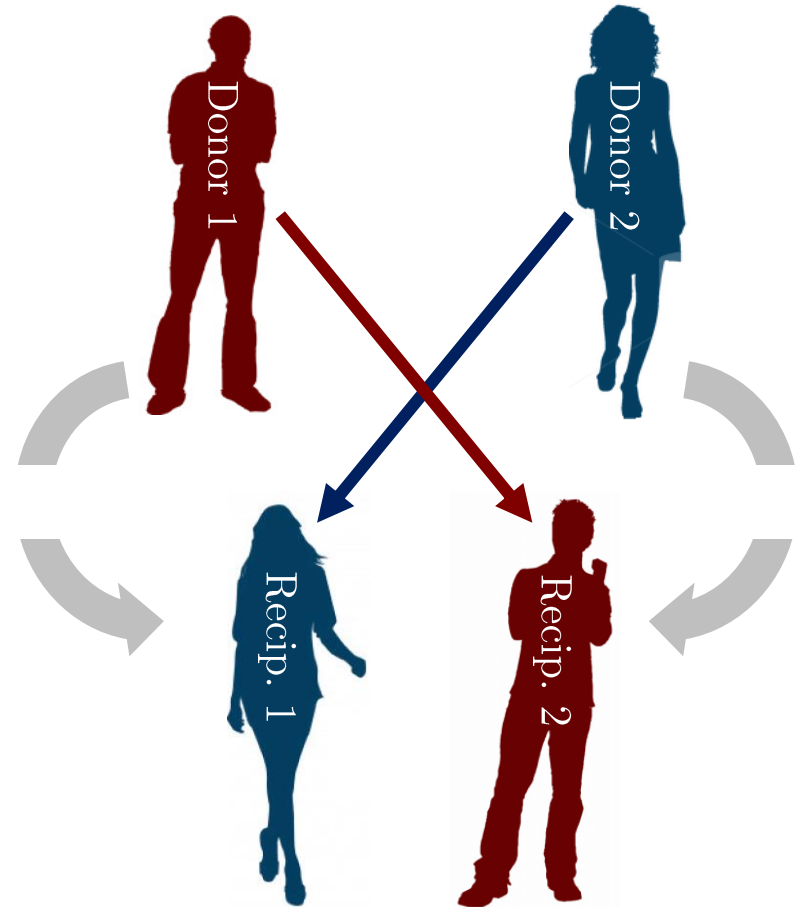
- 8 queens statistics:
 - State space of size ≈ 17 million
 - Starting from random state, steepest-ascent hill climbing solves 14% of problem instances
 - It takes 4 steps on average when it succeeds and 3 when it gets stuck
 - When “sideways” moves are allowed, solves 94% of instances, but with 21 steps for success and 64 for failure
- Variants:
 - **Stochastic hill climbing:** Chooses at random among uphill moves, with the probability depending on the improvement
 - **Random-restart hill climbing:** Conducts a series of hill-climbing searches from random states; obviously complete, and expected number of iterations is roughly 7, with roughly 22 steps overall

If at first you
don't succeed,
try, try again!



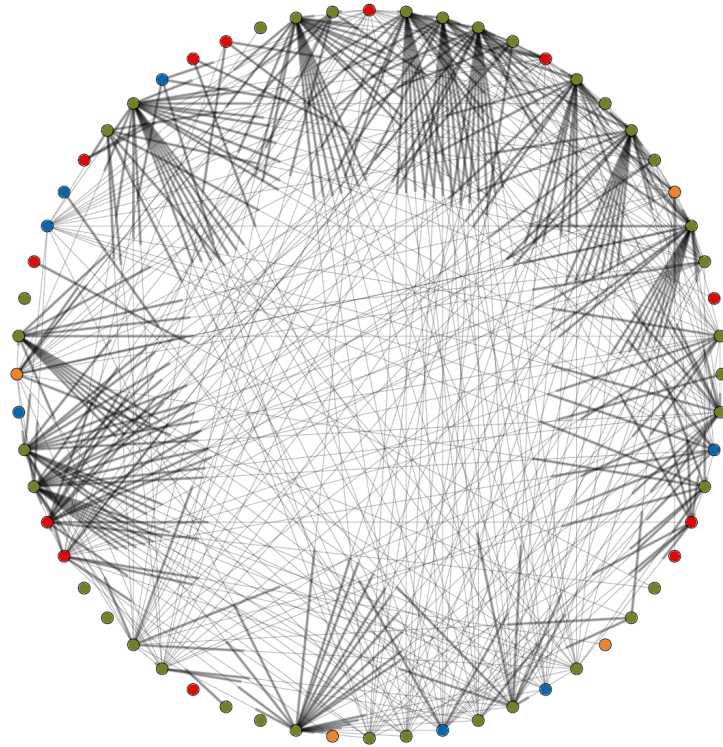
APPLICATION: KIDNEY EXCHANGE*

- Kidney donations from live donors are common
- But some donors are incompatible with their patients
- Kidney exchange enables swaps



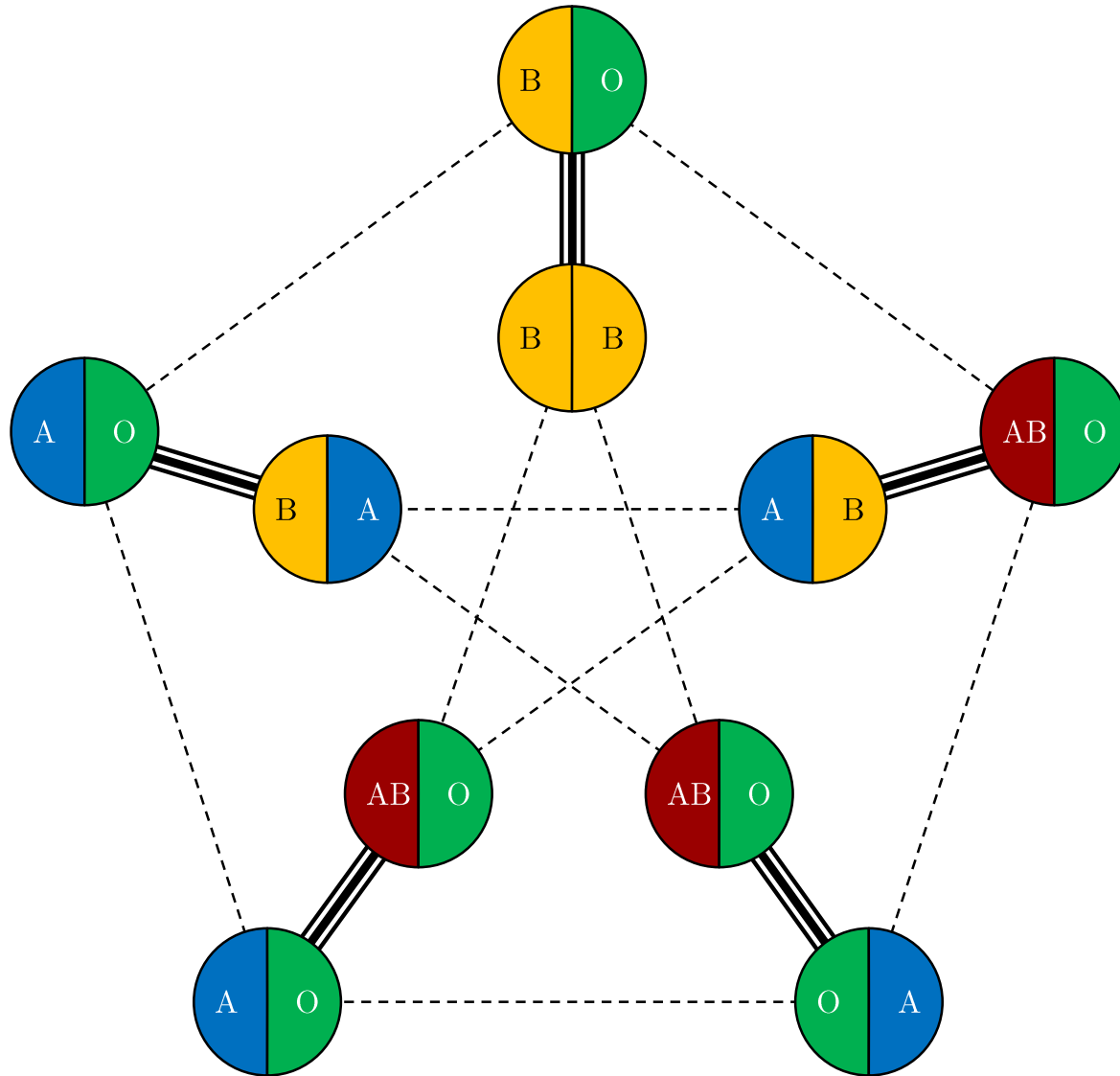
* Just for fun

APPLICATION: KIDNEY EXCHANGE*



UNOS compatibility graph, December 2010

* Just for fun



* Just for fun

APPLICATION: KIDNEY EXCHANGE*

Graph 3

Graph 2

Graph 1

Input



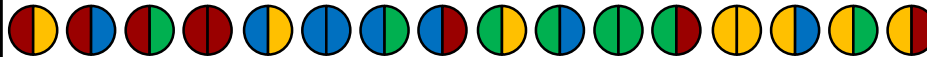
Matching 3

Matching 2

Matching 1

Output

At each step find matching that maximizes
(matching size) - (potential lost)



Local
search!

* Just for fun

SUMMARY

- Terminology:
 - Search problems
 - Local search
- Algorithms:
 - Generic search algorithms:
tree search vs. graph search
 - Strategies: BFS, Bidirectional,
UCS, DFS, Iterative Deepening
 - Local search algorithms: Hill Climbing

