

CMSC 471

Artificial Intelligence

Search

KMA Solaiman – ksolaima@umbc.edu

Some material adopted from notes
by Charles R. Dyer, University of
Wisconsin-Madison

Today's topics

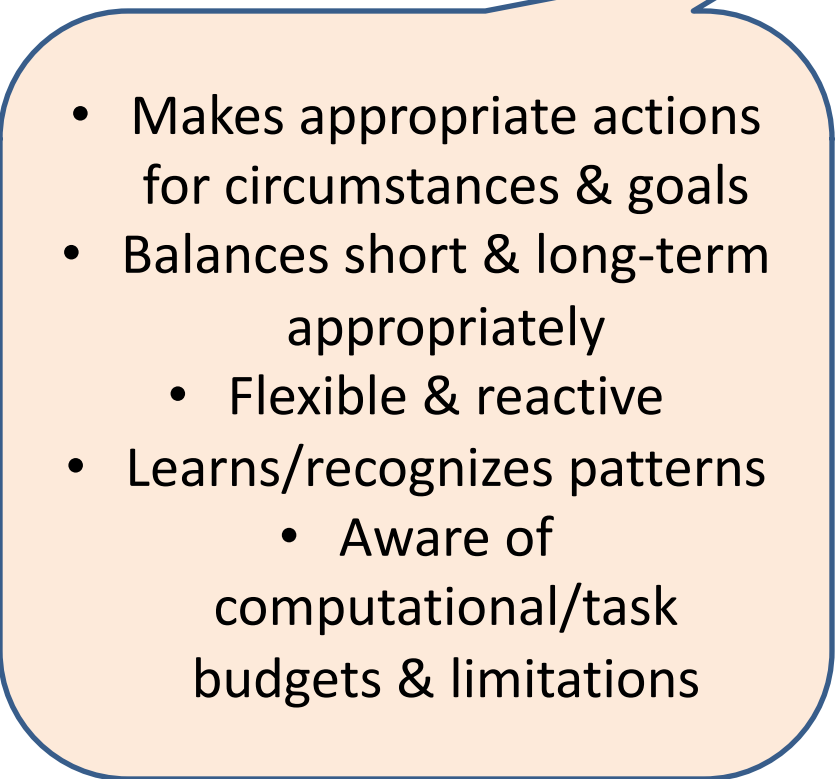
- Goal-based agents
- Representing states and actions
- Example problems
- Generic state-space search algorithm
- Specific algorithms
 - Breadth-first search
 - Depth-first search
 - Uniform cost search
 - Depth-first iterative deepening
- Example problems revisited

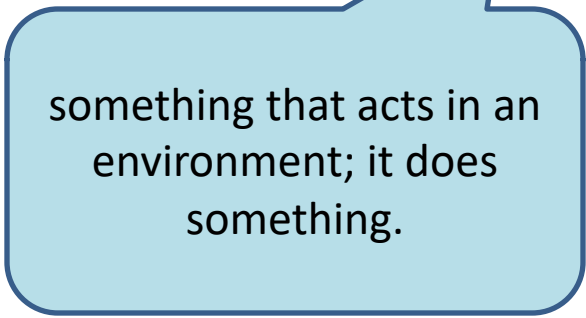


Recap

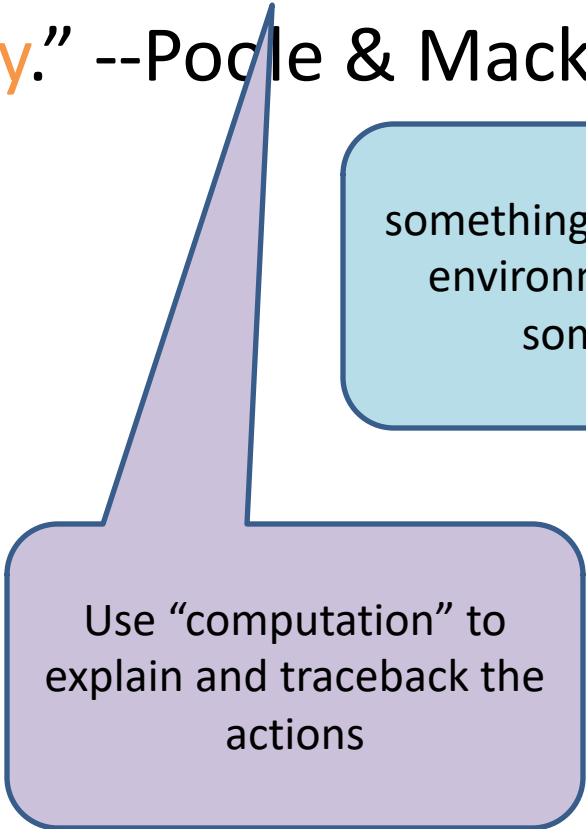
Okay, but really? What is AI?

“Artificial intelligence, or AI, is the field that studies the synthesis and analysis of **computational agents** that act **intelligently**.” --Poole & Mackworth

- 
- Makes appropriate actions for circumstances & goals
 - Balances short & long-term appropriately
 - Flexible & reactive
 - Learns/recognizes patterns
 - Aware of computational/task budgets & limitations



something that acts in an environment; it does something.



Use “computation” to explain and traceback the actions

Characteristics of environments

→ Lots of real-world domains fall into the hardest case!

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
Solitaire	No	Yes	Yes	Yes	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes	No
Taxi driving	No	No	No	No	No	No
Internet shopping	No	No	No	No	Yes	No
Medical diagnosis	No	No	No	No	No	Yes

A **Yes** in a cell means that aspect is simpler; a **No** more complex

Recap

Agents

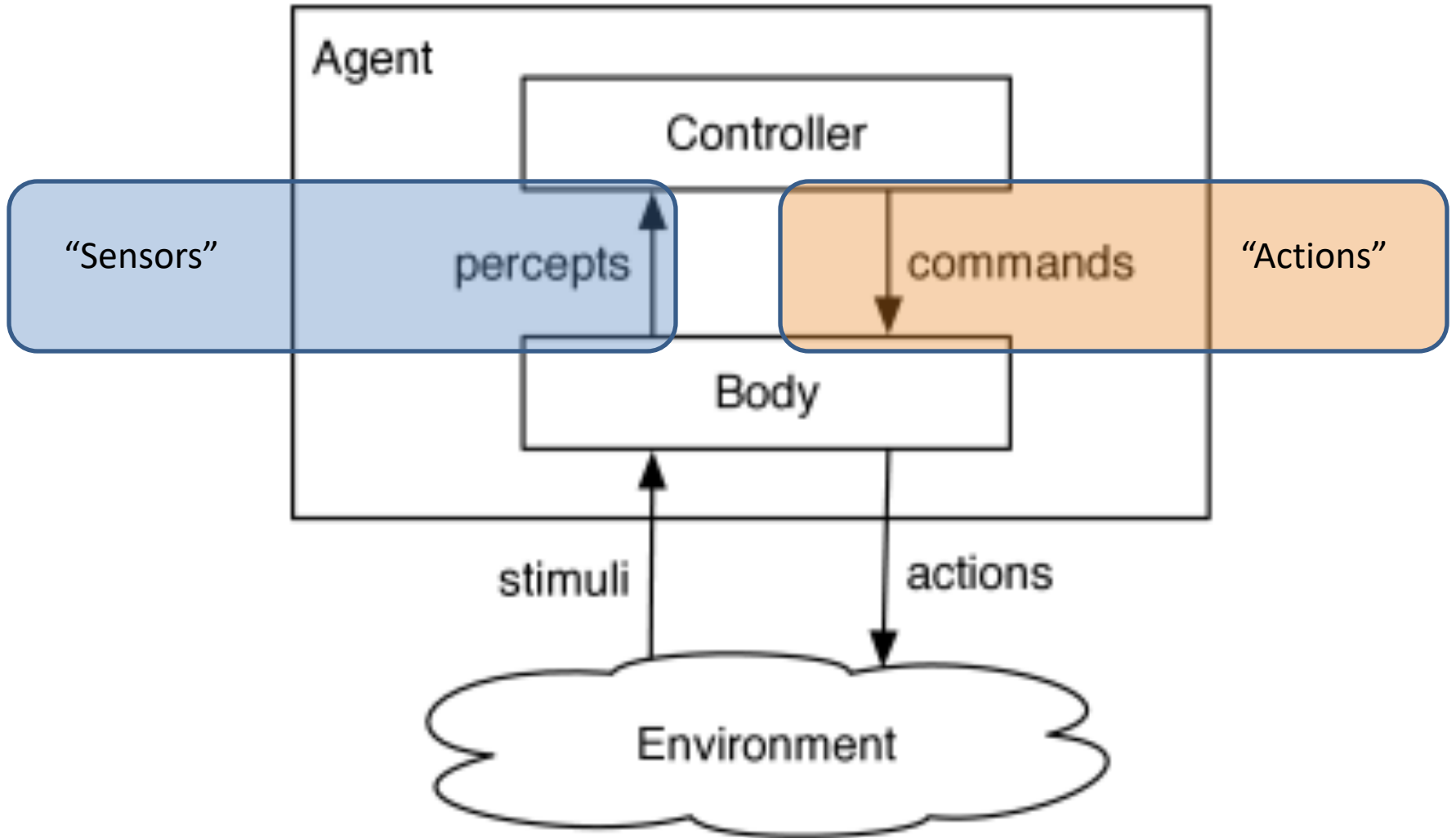


Fig. 2.1

(0) Table-driven agents

Use percept sequence/action table to find next action. Implemented by a **lookup table**

(1) Simple reflex agents

Based on **condition-action rules**, stateless devices with no memory of past world states

(2) Agents with memory

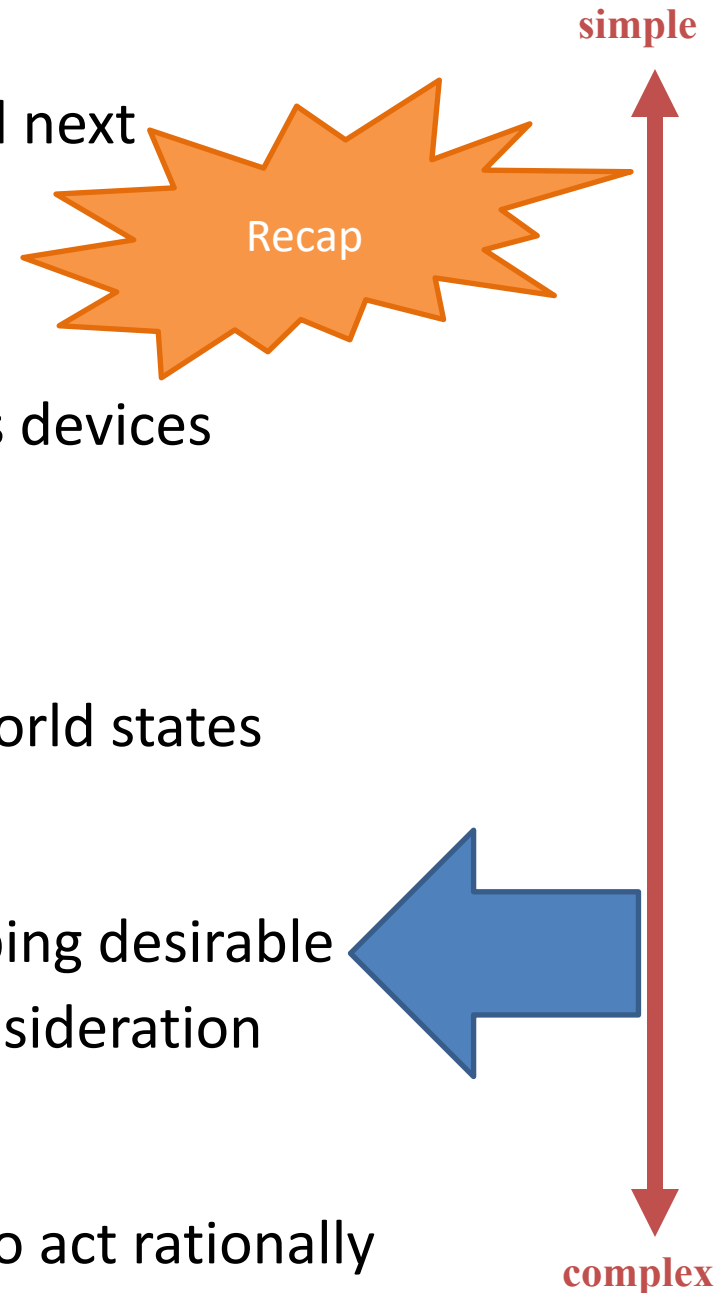
represent states and keep track of past world states

(3) Agents with goals

Have a state and **goal information** describing desirable situations; can take future events into consideration

(4) Utility-based agents

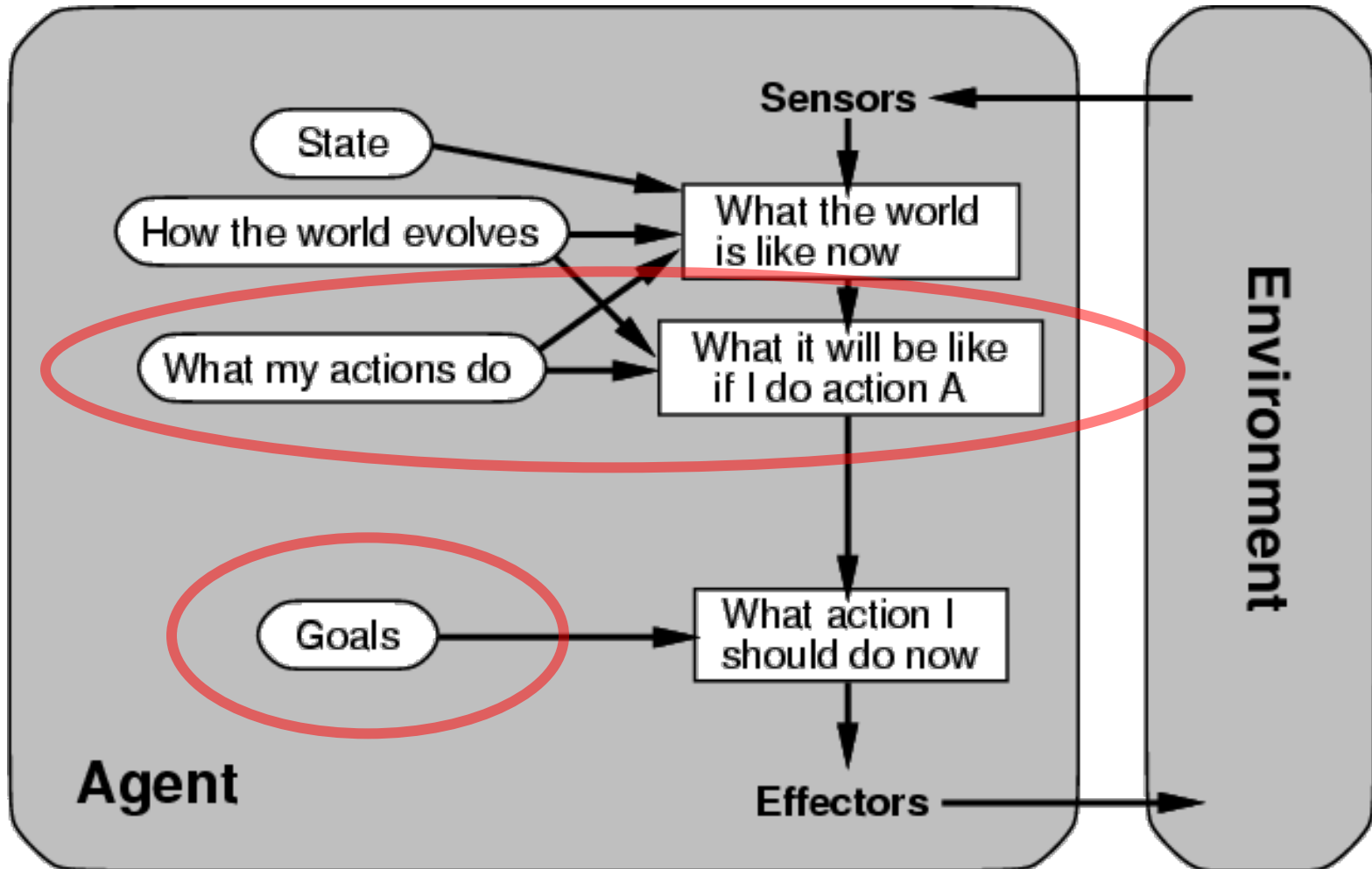
base decisions on [utility theory](#) in order to act rationally



Recap

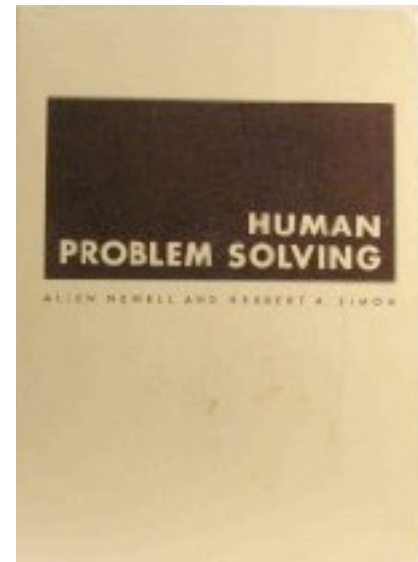
(3) Architecture for goal-based agent

state and **goal information** describe desirable situations allowing agent to take future events into consideration



Big Idea

[Allen Newell](#) and [Herb Simon](#) developed the *problem space principle* as an AI approach in the late 60s/early 70s

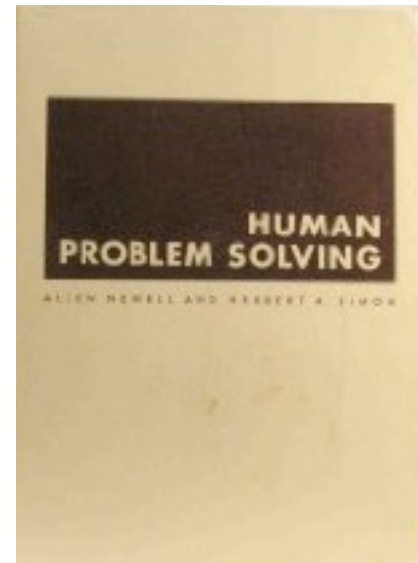


"The rational activity in which people engage to solve a problem can be described in terms of (1) a set of **states** of knowledge, (2) **operators** for changing one state into another, (3) **constraints** on applying operators and (4) **control** knowledge for deciding which operator to apply next."

Newell A & Simon H A. Human problem solving.
Englewood Cliffs, NJ: Prentice-Hall. 1972.

Big Idea

[Allen Newell](#) and [Herb Simon](#) developed the *problem space principle* as an AI approach in the late 60s/early 70s

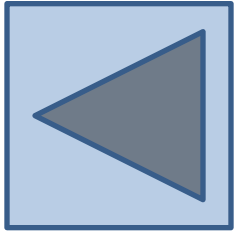


"The rat
problem
of know
another
control
next."

We'll achieve this by
formulating an appropriate
graph and then applying
graph search algorithms to it

e a
tes
nto
4)
ply

Newell A & Simon H A. Human problem solving.
Englewood Cliffs, NJ: Prentice-Hall. 1972.

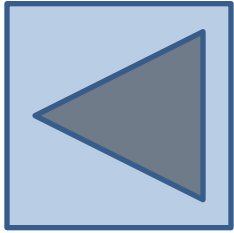


Remember: Graphs

- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes, (x, y)

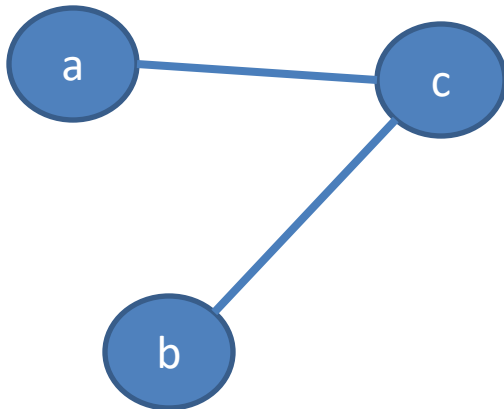
G can be:

- Undirected: order of (x, y) doesn't matter
 - These are symmetric
- Directed: order of (x, y) does matter
- Weighted: cost function $g(x, y)$
- (among other qualities)



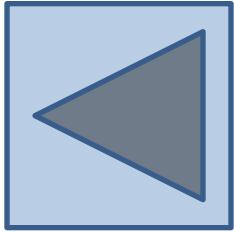
Remember: Graphs

- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



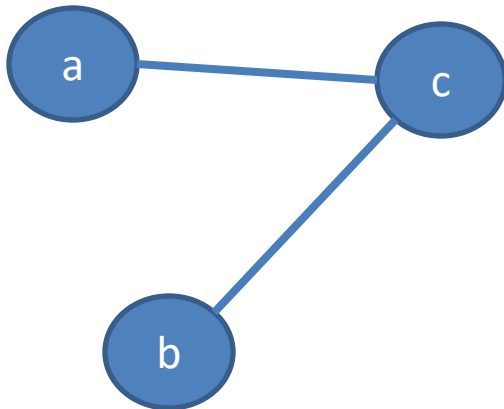
$V = \{ ??? \}$

$E = \{ ??? \}$



Remember: Graphs

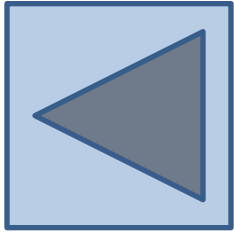
- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



$$V = \{ a, b, c \}$$

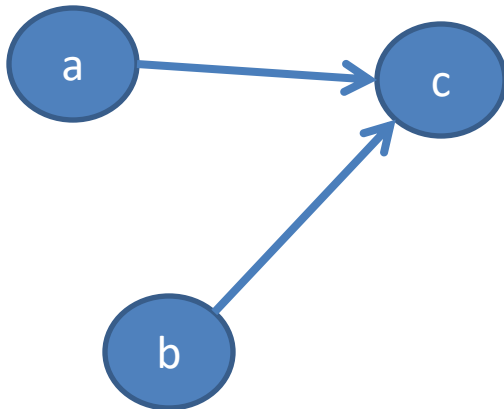
$$E = \{ (a, c), (b, c) \}$$

undirected



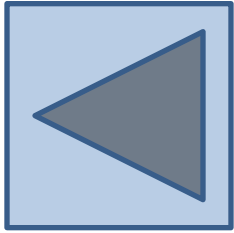
Remember: Graphs

- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



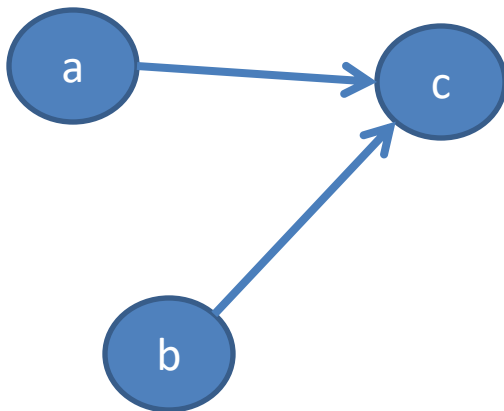
$V = \{ ??? \}$

$E = \{ ??? \}$



Remember: Graphs

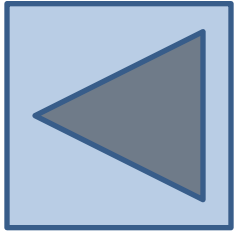
- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



$$V = \{ a, b, c \}$$

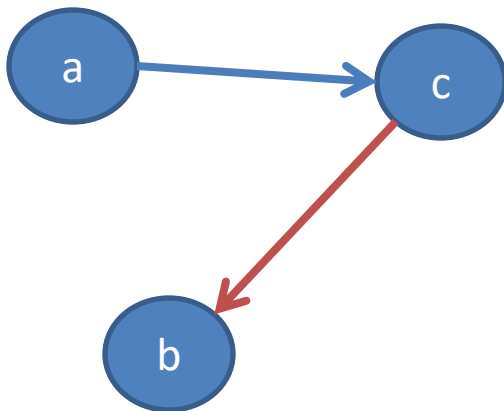
$$E = \{ (a, c), (b, c) \}$$

directed



Remember: Graphs

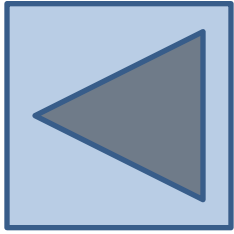
- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



$$V = \{ a, b, c \}$$

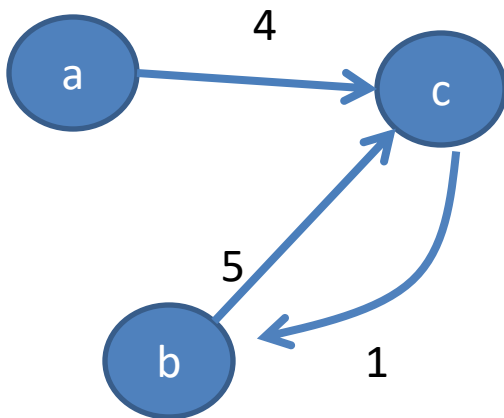
$$E = \{ (a, c), (c, b) \}$$

directed



Remember: Graphs

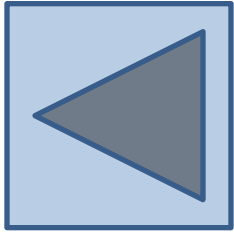
- A graph $G = (E, V)$
- V = set of vertices (nodes)
- E = set of edges between pairs of nodes



$V = \{ ??? \}$

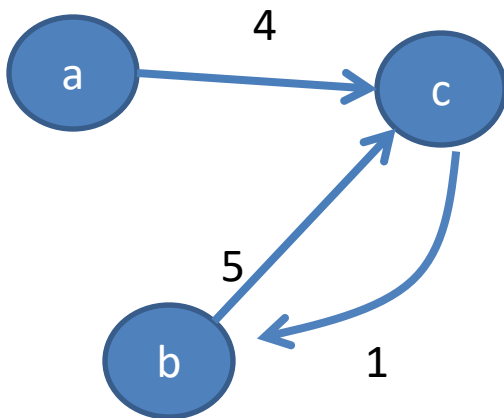
$E = \{ ??? \}$

$g = ???$



Remember: Graphs

- A graph $G = (E, V)$
- $V =$ set of vertices (nodes)
- $E =$ set of edges between pairs of nodes



$$V = \{ a, b, c \}$$

$$E = \{ (a, c), (b, c), (c, b) \}$$

$$g = \{ (a, c): 4, (b, c): 5, (c, b): 1 \}$$

weighted, directed

Some Key Terms: States, Goal, and Solution

State: a representation of the current world/environment (as needed for the agent)

Initial State: The state the agent/problem starts in

Goal State: The desired state

Actions, Transition Model: what is allowed, with what cost;
(state, action) = next state

Solution: a sequence of actions that operate sequentially on states and allow the agent to achieve its goal

Example: 8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles to produce a desired goal configuration

5	4	
6	1	8
7	3	2

Start State

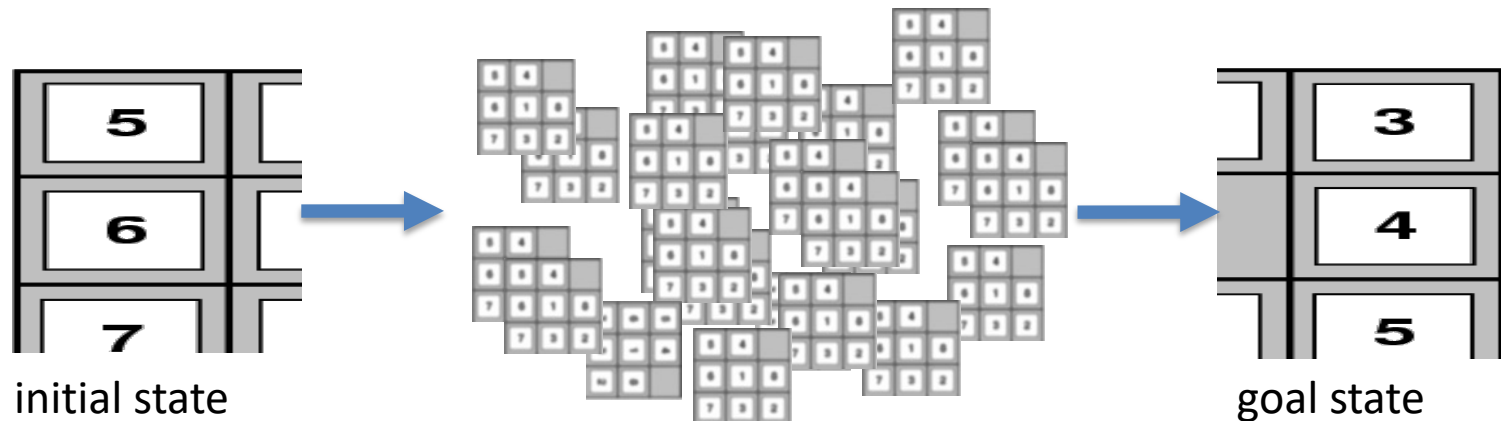
1	2	3
8		4
7	6	5

Goal State

Building goal-based agents

We must answer the following questions

- How do we represent the **state** of the “world”?
- What is the **goal** and how can we recognize it?
- What are the possible **actions**?
- What *relevant* information do we encoded to describe states, actions and their effects and thereby solve the problem?



Characteristics of 8-puzzle ?

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
8-puzzle						

Characteristics of 8-puzzle

	Fully observable?	Deterministic?	Episodic?	Static?	Discrete?	Single agent?
8-puzzle	Yes	Yes	Yes	Yes	Yes	Yes

- All the Yes's mean it may be relatively easy!
- This is typical of the problems worked on in the 60s and 70s
- And the algorithms for solving them a state-space search approach

Representing states

- State of an 8-puzzle?

5	4	
6	1	8
7	3	2

Representing states

5	4	
6	1	8
7	3	2

- State of an 8-puzzle?
- A 3x3 array of integer in {0..8}
- No integer appears twice
- 0 represents the empty space

- In Python, we might implement this using a nine-character string: "540681732"
- And write functions to map the 2D coordinates to an index

What's the goal to be achieved?



- Describe situation we want to achieve, a set of properties that we want to hold, etc.
- Defining a **goal test** function that when applied to a state returns True or False
- For our problem:

```
def isGoal(state):  
    return state == "123405678"
```

What are the actions?



- **Primitive actions** for changing the state
 - In a **deterministic** world: no uncertainty in an action's effects (simple model)
- Given action and description of **current world state**, action completely specifies
 - Whether action **can** be applied to the current world (i.e., is it applicable and legal?) and
 - What state **results** after action is performed in the current world (i.e., no need for *history* information to compute the next state)

Representing actions



- Actions ideally considered as **discrete events** that occur at an **instant of time**
- Example, in a planning context
 - If `state:inClass` and perform `action:goHome`, then next state is `state:atHome`
 - There's no time where you're neither in class nor at home (i.e., in the state of “going home”)

Representing actions

- Actions for 8-puzzle?

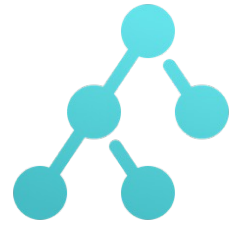
5	4	
6	1	8
7	3	2

Representing actions

5	4	
6	1	8
7	3	2

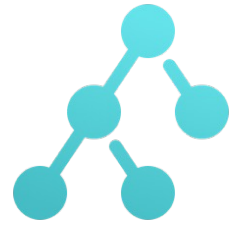
- Actions for 8-puzzle?
- Number of actions/operators depends on the **representation** used in describing a state
 - Specify 4 possible moves for each of the 8 tiles, resulting in a total of **$4*8=32$ operators**
 - Or: Specify four moves for “blank” square and we only need **4 operators**
- **Representational shift can simplify a problem!**

Representing states



- **Size of a problem** usually described in terms of possible **number of states**
 - Tic-Tac-Toe has about 3^9 states ($19,683 \approx 2 * 10^4$)
 - Checkers has about 10^{40} states
 - Rubik's Cube has about 10^{19} states
 - Chess has about 10^{120} states in a typical game
 - Go has $2 * 10^{170}$
 - Theorem provers may deal with an infinite space
- State space size \approx solution difficulty

Representing states

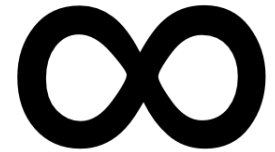


- Our estimates were loose upper bounds
- How many **possible, legal** states does tic-tac-toe really have?
- Simple upper bound: nine board cells, each of which can be empty, O or X, so 3^9
- Only 593 states after eliminating

– impossible states 

– Rotations and reflections 

Can Problem spaces be infinite?



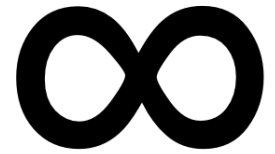
Yes! examples include theorem proving and this simple example from [Knuth](#) (1964)

- Starting with the number 4, a sequence of square root, floor, and factorial operations can reach any desired positive integer
- To get to 5 from 4, do

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \rfloor = 5.$$

- `floor(sqrt (sqrt (sqrt (sqrt (sqrt (fact (fact 4)))))))`

Infinitely hard to solve?



- No
- But you must be more careful in searching a problem space that may be infinite
- Some approaches (e.g., breadth first search) may be better than others (e.g., depth first search)
 - Depth first search can get lost exploring an infinite subspace

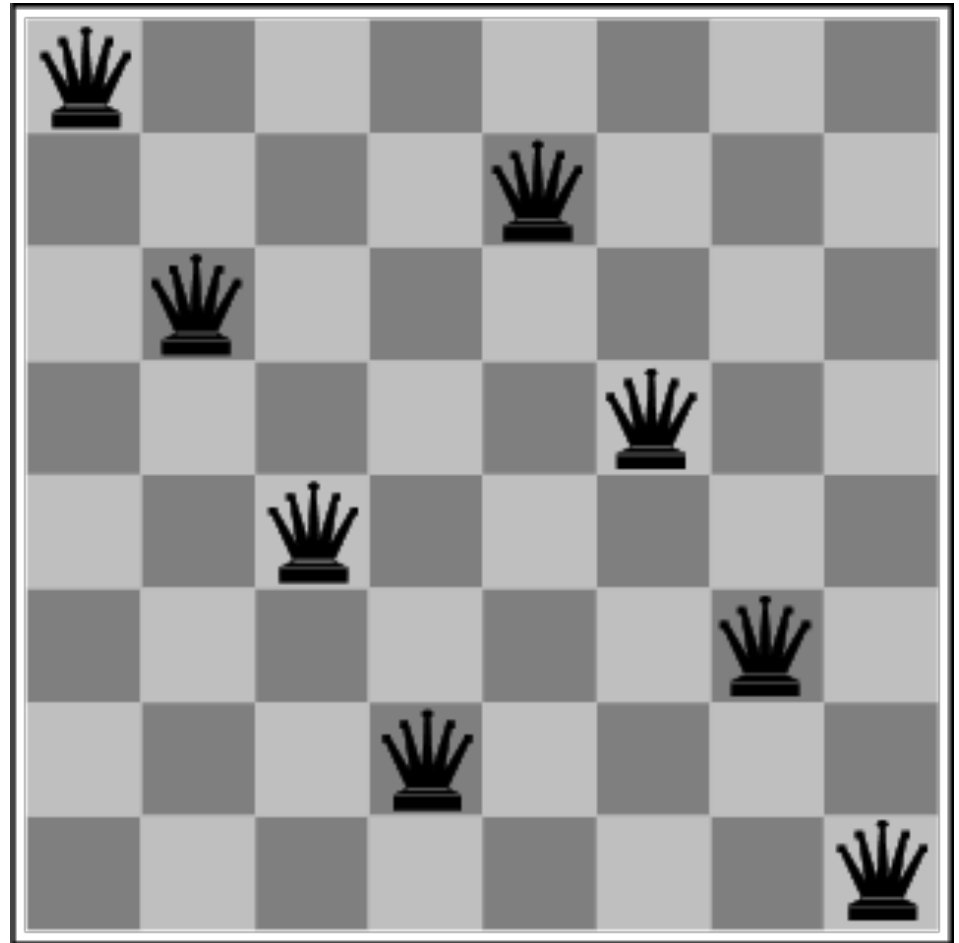
Some example problems

- Toy problems and micro-worlds
 - 8-Puzzle
 - Missionaries and Cannibals
 - Cryptarithmic
 - 8-Queens Puzzle
 - Remove 5 Sticks
 - Water Jug Problem
- Real-world problems

Example: The 8-Queens Puzzle

Place eight queens on a chessboard such that no queen attacks any other

We can generalize the problem to a $N \times N$ chessboard



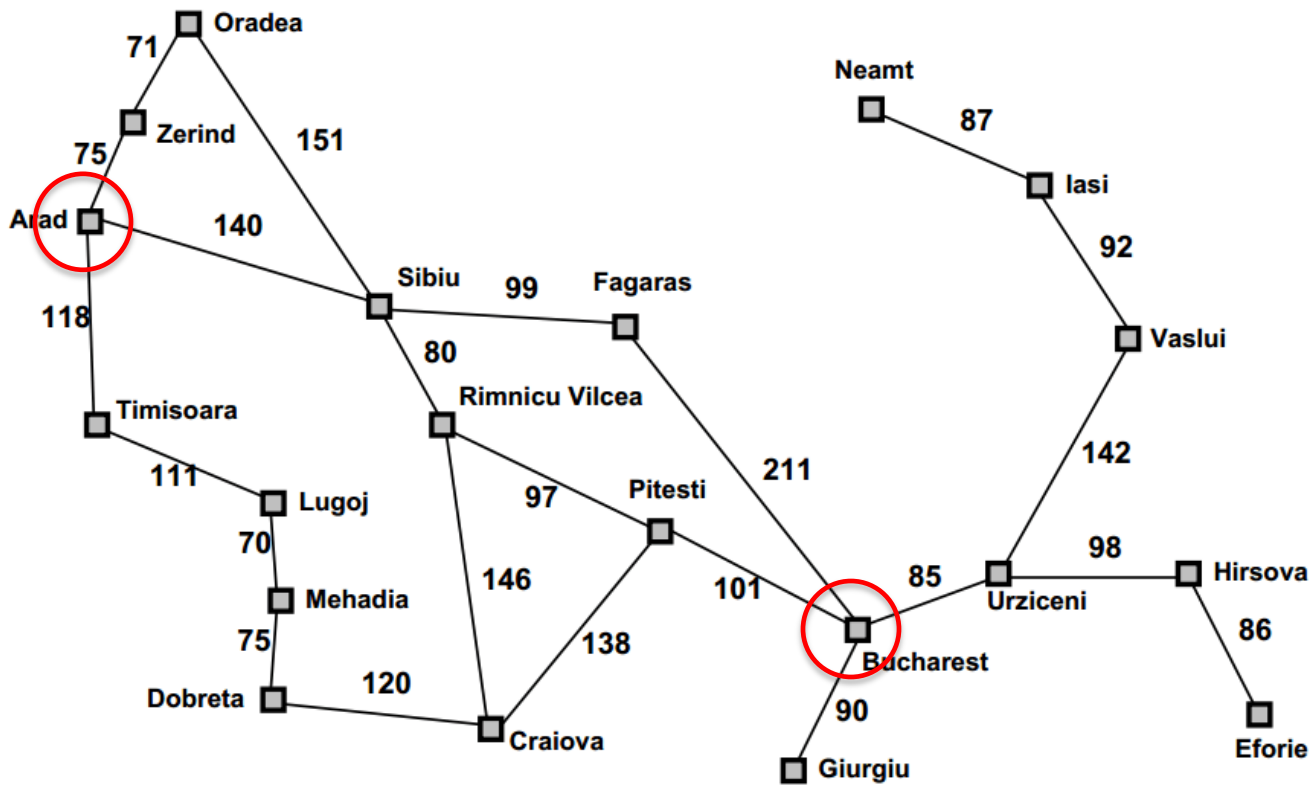
What are the states, goal test, actions?

Some more real-world problems

- Route finding
- Touring (traveling salesman)
- Logistics
- VLSI layout
- Robot navigation
- Theorem proving
- Learning

Route Planning

Find a route from Arad to Bucharest



A simplified map of major roads in Romania used in our text

Water Jug Problem



- Two jugs J1 & J2 with capacity C1 & C2
- Initially J1 has W1 water and J2 has W2 water
 - e.g.: full 5 gallon jug and empty 2 gallon jug
- Possible actions:
 - Pour from jug X to jug Y until X empty or Y full
 - Empty jug X onto the floor
- Goal: J1 has G1 water and J2 G2
 - G1 or G2 can be -1 to represent any amount
- E.g.: initially full jugs with capacities 3 and 1 liters, goal is to have 1 liter in each

Example: Water Jug Problem



- Two jugs J1 and J2 with capacity C1 and C2
- Initially J1 has W1 water and J2 has W2 water
 - e.g.: a full 5-gallon jug and an empty 2-gallon jug
- Possible actions:
 - Pour from jug X to jug Y until X empty or Y full
 - Empty jug X onto the floor
- Goal: J1 has G1 water and J2 G2
 - G1 or G2 can be -1 to represent any amount

Example: Water Jug Problem



Given full 5-gal. jug and empty 2-gal. jug, fill 2-gal jug with one gallon

- State representation?
 - General state?
 - Initial state?
 - Goal state?
- Possible actions?
 - Condition?
 - Resulting state?

Action table

Name	Cond.	Transition	Effect
dump1	$x > 0$	$(x, y) \rightarrow (0, y)$	Empty Jug 1
dump2	$y > 0$		Empty Jug 2
pour_1_2	$x > 0$ & $y < C_2$		Pour from Jug 1 to Jug 2
pour_2_1	$y > 0$ & $X < C_1$		Pour from Jug 2 to Jug 1

Example: Water Jug Problem



Given full 5-gal. jug
and empty 2-gal. jug,
fill 2-gal jug with one
gallon

- State = (x,y) , where x is water in jug 1; y is water in jug 2
- Initial State = $(5,0)$
- Goal State = $(-1,1)$, where -1 means any amount

Action table

Name	Cond.	Transition	Effect
dump1	$x > 0$	$(x,y) \rightarrow (0,y)$	Empty Jug 1
dump2	$y > 0$	$(x,y) \rightarrow (x,0)$	Empty Jug 2
pour_1_2	$x > 0$ & $y < C2$	$(x,y) \rightarrow (x-D, y+D)$ $D = \min(x, C2-y)$	Pour from Jug 1 to Jug 2
pour_2_1	$y > 0$ & $X < C1$	$(x,y) \rightarrow (x+D, y-D)$ $D = \min(y, C1-x)$	Pour from Jug 2 to Jug 1

So...

- How can we represent the states?
- What's an initial state
- How do we recognize a goal state
- What are the actions; how can we tell which ones can be performed in a given state; what is the resulting state
- How do we search for a solution from an initial state given a goal state
- What is a solution? The goal state achieved or a path to it?

Search in a state space

- Basic idea:
 - Create representation of initial state
 - Try all possible actions & connect states that result
 - Recursively apply process to the new states until we find a solution or dead ends
- We need to keep track of the connections between states and might use a
 - Tree data structure or
 - Graph data structure
- A graph structure is best in general...

Formalizing state space search

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node
- **Nodes:** data structures with state description and other info, e.g., node's parent, name of action that generated it from parent, etc.
- **Arcs:** instances of actions, head is a state, tail is the state that results from action

Formalizing search in a state space

- Each arc has fixed, positive **cost** associated with it corresponding to the action cost
 - Simple case: all costs are 1
- Each node has a set of **successor nodes** corresponding to all legal actions that can be applied at node's state
 - **Expanding** a node = generating its successor nodes and adding them and their associated arcs to the graph
- One or more nodes are marked as **start nodes**
- A **goal test** predicate is applied to a state to determine if its associated node is a goal node

Formalizing search

- **Solution:** sequence of actions associated with a path from a start node to a goal node
- **Solution cost:** sum of the arc costs on the solution path
 - If all arcs have same (unit) cost, then solution cost is length of solution (number of steps)
 - Algorithms generally require that arc costs cannot be negative (why?)

Formalizing search

- **State-space search:** searching through state space for solution by **making explicit** a portion of an **implicit** state-space graph to find a goal node
 - Can't materialize whole space for large problems
 - Initially $V=\{S\}$, where S is the start node, $E=\{\}$
 - On expanding S , its *successor nodes* are generated and added to V and associated *arcs added to E*
 - Process continues until a goal node is found
- Nodes represent a *partial solution* path (+ cost of partial solution path) from S to the node
 - From a node there may be many possible paths (and thus solutions) with this partial path as a prefix

A General Searching Algorithm

Core ideas:

1. Maintain a list of **frontier (fringe)** nodes
 1. Nodes coming *into* the frontier have been explored
 2. Nodes *going out of the frontier* have not been explored
2. Iteratively select nodes from the frontier and explore unexplored nodes from the frontier
3. Stop when you reach your **goal**

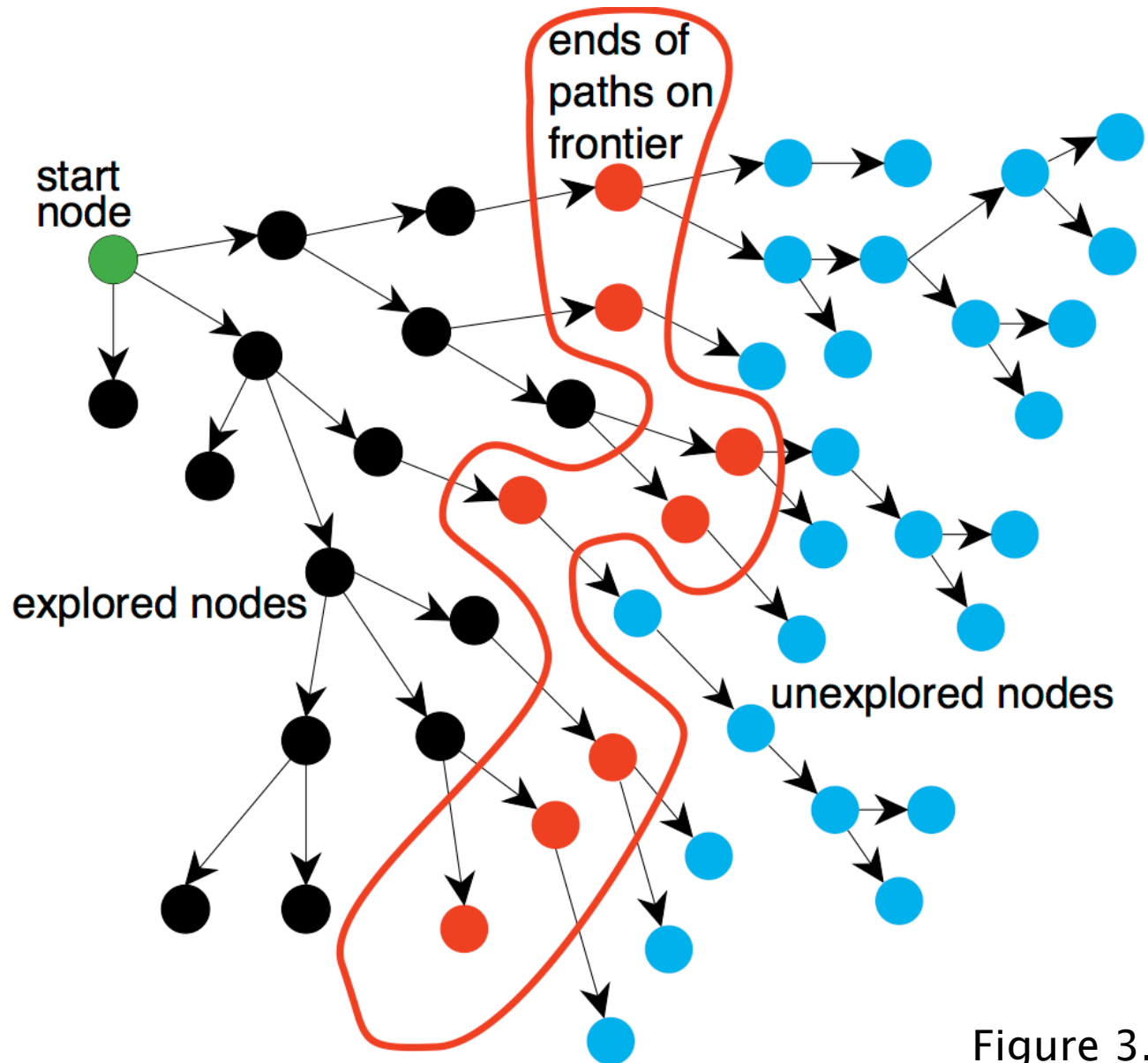


Figure 3.3

State-space search algorithm

;; problem describes the start state, operators, goal test, and operator costs

;; queueing-function is a comparator function that ranks two states

;; general-search returns either a goal node or failure

```
function general-search (problem, QUEUEING-FUNCTION)
  nodes = MAKE-QUEUE (MAKE-NODE (problem.INITIAL-STATE) )
  loop
    if EMPTY(nodes) then return "failure"
    node = REMOVE-FRONT(nodes)
    if problem.GOAL-TEST (node.STATE) succeeds
      then return node
    nodes = QUEUEING-FUNCTION (nodes, EXPAND (node,
      problem.OPERATORS) )
  end
```

;; Note: The goal test is NOT done when nodes are generated

;; Note: This algorithm does not detect loops

Key procedures to be defined

- EXPAND
 - Generate a node's successor nodes, adding them to the graph if not already there
- GOAL-TEST
 - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
 - Maintain ranked list of nodes that are candidates for expansion
 - Changing definition of the QUEUEING-FUNCTION leads to different search strategies

What does “search”
look like for a
particular problem?

start

1	2	3
4	8	
7	6	5

goal

1	2	3
4	5	6
7	8	

start

1	2	3
4	8	
7	6	5

Expanding a node on the *fringe*
(taking a certain action)

1	2	
4	8	3
7	6	5

1	2	3
4	8	5
7	6	

goal

1	2	3
4	5	6
7	8	

start

1	2	3
4	8	
7	6	5

1	2	
4	8	3
7	6	5

1	2	3
4	8	5
7	6	

1	2	3
4	8	5
7		6

*Expanding a node on the **fringe** (taking a certain action). Not all actions shown.*

goal

1	2	3
4	5	6
7	8	

start

1	2	3
4	8	
7	6	5

1	2	
4	8	3
7	6	5

1	2	3
4	8	5
7	6	

1	2	3
4	8	5
7		6

1	2	3
4	8	5
	7	6

1	2	3
4		5
7	8	6

*Expanding a node on the **fringe** (taking a certain action). Not all actions shown.*

goal

1	2	3
4	5	6
7	8	

start

1	2	3
4	8	
7	6	5

1	2	
4	8	3
7	6	5

1	2	3
4	8	5
7	6	

1	2	3
4	8	5
7		6

1	2	3
4	8	5
	7	6

1	2	3
4		5
7	8	6

1	2	3
4	5	
7	8	6

1	2	3
	4	5
7	8	6

1		3
4	2	5
7	8	6

Expanding a node on the *fringe* (taking a certain action). Not all actions shown.

goal

1	2	3
4	5	6
7	8	

start

1	2	3
4	8	
7	6	5

1	2	
4	8	3
7	6	5

1	2	3
4	8	5
7	6	

1	2	3
4	8	5
7		6

1	2	3
4	8	5
	7	6

1	2	3
4		5
7	8	6

1	2	3
4	5	
7	8	6

1	2	3
	4	5
7	8	6

1		3
4	2	5
7	8	6

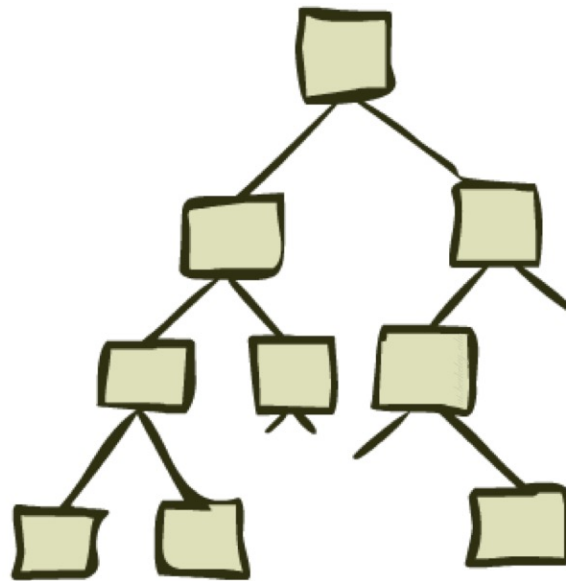
1	2	3
4	5	6
7	8	

1	2	
4	5	3
7	8	6

goal

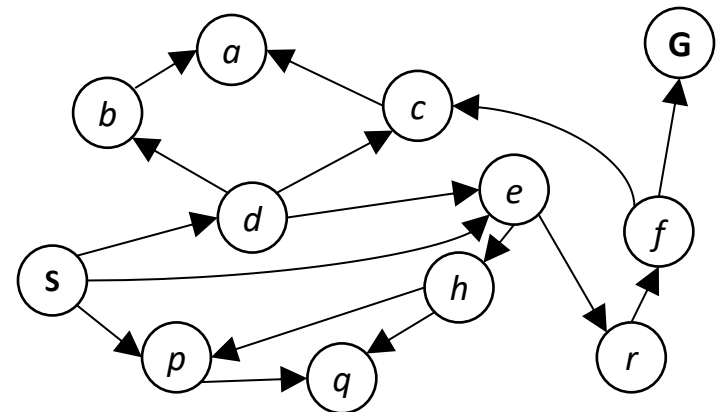


State Space Graphs and Search Trees



State Space Graphs

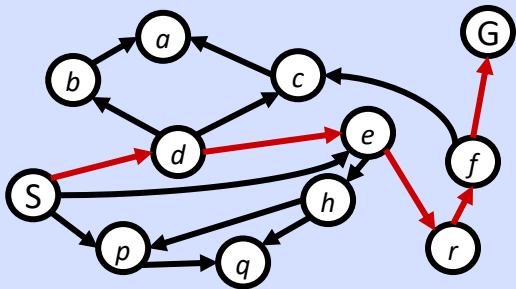
- State space graph: A mathematical representation of a search problem
 - Nodes are (abstracted) world configurations
 - Arcs represent transitions/ successors (action results)
 - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



Tiny state space graph for a tiny search problem

State Space Graphs vs. Search Trees

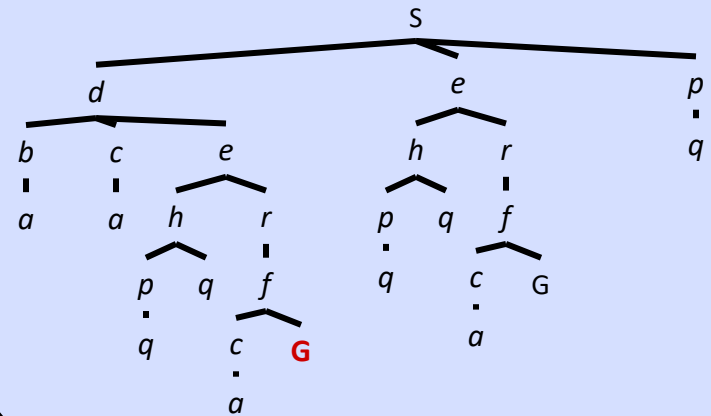
State Space Graph



Each NODE in in the search tree is an entire PATH in the state space graph.

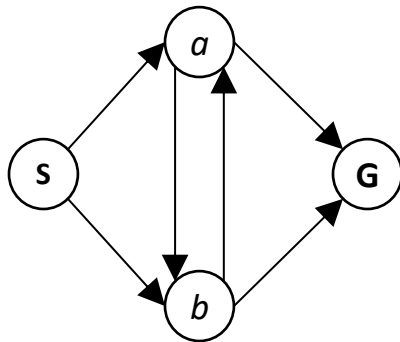
We construct the tree on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

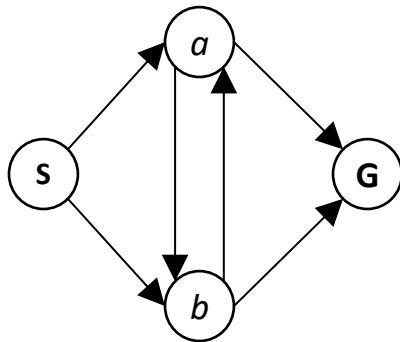


How big is its search tree (from S)?

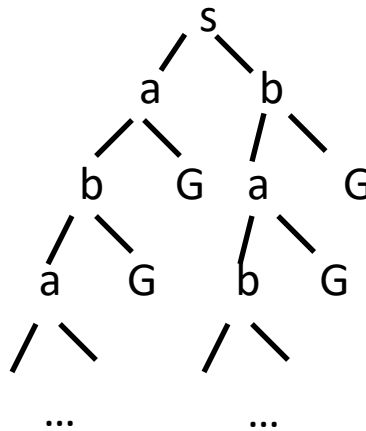


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



Important: Those who don't know history are doomed to repeat it!