

Most materials in this course are based on materials from <http://aima.cs.berkeley.edu/> and <http://ai.berkeley.edu>



Chapter 0

Course Details

COMP 3270
Artificial Intelligence

Dirk Schnieders

Outline

- Syllabus
- Assessment
- Plagiarism
- Staff
- Slides
- Recording

Prerequisite

- COMP2119 Introduction to Data Structures and Algorithms
 - or equivalent

Staff

- Instructor (Exam, Forum, Lecture): Dirk Schnieders
 - Email: sdirk@cs.hku.hk
 - Office: CB324
 - Consultation hours: Friday, 2:00 - 4:00 pm
- TA (Quiz 1, A1, A2, Forum): Wu Yijie
 - Email: wuyj@connect.hku.hk
 - Office: HWG01 [g103]
 - Consultation hours: tbd
- TA (Quiz 2, A3, A4, Forum): Cui Zhiming
 - Email: cuizm.neu.edu@gmail.com
 - Office: CB401
 - Consultation hours: tbd

Syllabus - Schedule*

*subject to change

Week	Tue Session (2h)	Topic*	Fri Session (1h)	Topic*
1	3 Sep, 10:30 - 12:20	0. Course Details, 1. Search (Uninformed)	6 Sep, 11:30 - 12:20	1. Search (Informed)
2	10 Sep, 10:30 - 12:20	1. Search (Informed, Local, CSPs)	13 Sep, 11:30 - 12:20	1. Search (Adversarial)
3	17 Sep, 10:30 - 12:20	1. Search (Adversarial)	20 Sep, 11:30 - 12:20	2. Markov Decision Processes
4	24 Sep, 10:30 - 12:20	2. Markov Decision Processes	27 Sep, 11:30 - 12:20	2. Markov Decision Processes
5	1 Oct, 10:30 - 12:20	Public Holiday	4 Oct, 11:30 - 12:20	3. Reinforcement Learning
6	8 Oct, 10:30 - 12:20	3. Reinforcement Learning	11 Oct, 11:30 - 12:20	3. Reinforcement Learning
7	Reading Week			
8	22 Oct, 10:30 - 12:20	4. Markov Models	25 Oct, 11:30 - 12:20	Quiz 1
9	29 Oct, 10:30 - 12:20	4. Markov Models	1 Nov, 11:30 - 12:20	5. Hidden Markov Models
10	5 Nov, 10:30 - 12:20	5. Hidden Markov Models	8 Nov, 11:30 - 12:20	5. Hidden Markov Models
11	12 Nov, 10:30 - 12:20	5. Hidden Markov Models	15 Nov, 11:30 - 12:20	6. Bayes Nets
12	19 Nov, 10:30 - 12:20	6. Bayes Nets	22 Nov, 11:30 - 12:20	6. Bayes Nets
13	26 Nov, 10:30 - 12:20	<i>Buffer (time permitting: Machine Learning)</i>	29 Nov, 11:30 - 12:20	Quiz 2

Syllabus - Topics*

*subject to change

- Part I: Search and Planning
 - 1. Search (Uninformed, Informed, Local, CSPs, Adversarial)
 - 2. MDPs (Value Iteration, Policy Iteration)
 - 3. RL (Temporal Difference Learning, Q Learning)
- Part II: Probabilistic Reasoning
 - 4. MMs (Probability Review, Markov Chain, Mini-Forward Algorithm)
 - 5. HMMs (Forward Algorithm, Particle Filtering)
 - 6. BNs (Inference, Sampling)
- Part III: Machine Learning
 - COMP3314

Assessment - Weight

- Exam: 50%
- Quiz 1: 8%
- Quiz 2: 8%
- Assignment 0: 0%
- Assignment 1: 9%
- Assignment 2: 9%
- Assignment 3: 8%
- Assignment 4: 8%

mark = quiz 1 + quiz 2 + A1 + A2 + A3 + A4 + exam

grade = m(mark)

the mapping function m() will not be published

Assessment - Quiz 1 & 2

- Written, closed book, calculator allowed
- Cheating cases will receive 0 marks and referred to program director
- Bring your student ID or HKID
- No show
 - 0 marks unless sick leave certificate or official leave from university
- Dates: 25 Oct & 29 Nov
- Time: 11:35 - 12:15 (please arrive by 11:25)
- Quiz 1 will cover
 - Chapter 1 - 3
- Quiz 2 will cover
 - Chapter 4 - 6

Assessment - Assignment 0, 1, 2, 3 & 4

- 5 Programming Assignments
 - Python 3
 - 2-3 weeks time, no deadline extension
 - Late submission, after the deadline, will result in 0 marks
 - Submission of the wrong files will result in 0 marks
 - Some students spend 80+ hours on these
 - Work smart
 - Submission of wrong files will result in 0 marks
 - Please check carefully
 - Release - Deadline (subject to change)
 - A0: ∞
 - A1: 23 Sep
 - A2: 15 Oct
 - A3: 4 Nov
 - A4: 1 Dec

Assessment - Final

- 2 hours
- Closed book
 - Students may bring
 - A4 paper with printed or handwritten notes
 - Calculator
- Date: TBD
- Time: TBD
- Location: TBD

Assessment - Results

- In-class assessment (i.e., quiz & assignment) results will be frozen on the day of the final exam
- You must check for the correctness of your in-class assessment on Moodle
 - If you find any problem, send an email to the appropriate TA immediately but definitely before the final exam
 - We will not be able to make adjustments to your assessment results after the final exam

Plagiarism

- What is Plagiarism ?
 - <https://tl.hku.hk/plagiarism/>
- First attempt
 - Written warning
 - Zero marks
 - Referred to program director
- Second attempt
 - Referred to University Disciplinary Committee
 - Published reprimand
 - Suspension of study
 - Expulsion

Shutterstock ID: 338747057



Prevent Plagiarism

- Never copy/paste anything **without citing**
 - If you copy existing work you must clearly state and cite appropriately
 - Solutions to assignments may be found on the internet
 - To prevent plagiarism, don't search for them
- Never look at existing solutions
 - Work on the problem yourself
- Never ask your friend to see his/her solution
 - Note that the source will also be punished with 0 marks

Slides

- Available on Moodle on the day before the lecture
 - Usually at night
 - Google Slides
 - Install on mobile device to view materials
 - Save paper and don't print
- Use the provided materials responsibly
 - Don't share with others without permission
 - Do not upload to internet without permission
 - I do not hold copyright for most materials, I am using them because I consider it fair use
 - You will be legally responsible to copyright holder if you share

Recording

- Lecture video and audio recording will be uploaded
 - Please remind me to switch on the recorder
 - I apologize in advance shall I forget to switch it on
 - Do not rely on the recording, there could be a technical issue
- Don't skip class and just listen to the recording at home
 - Attend the lecture
 - Ask/answer questions and present solutions
 - Crucial for your learning
 - Talk to me, our TA and your fellow students

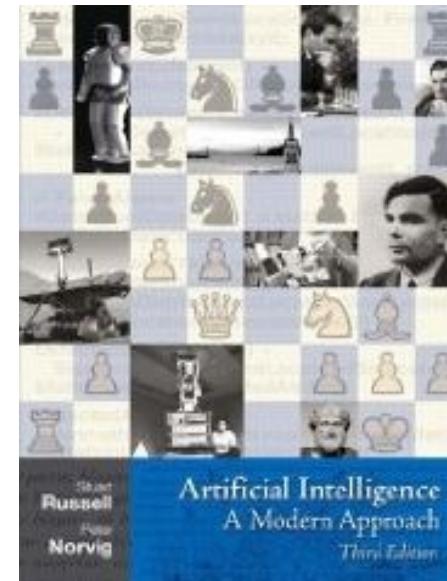


Big Class Management

- **We will not answer emails** unless personal information is involved
- If you have questions about course materials, please **use the forum**
 - We will try to answer within 48 hours

Textbook

- Artificial Intelligence: A Modern Approach
 - <http://aima.cs.berkeley.edu/>



Assignment 0

- Released

COMP3270 - Assignment 0

19/20 Semester 1

This assignment is based on materials by <http://ai.berkeley.edu>. This is an optional assignment, **no need to submit anything**.

Introduction

This is an introductory assignment which you optionally can submit before the deadline published on Moodle. Assignment 0 will mainly cover the following:

- A basic command line tutorial
- A basic Python 3 tutorial
- Some tasks with an autograder that checks for technical correctness

Getting Help: You are not alone! If you find yourself stuck on something, please let us know in the forum. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

This document assumes that you are using Linux. However, all the software used here is platform independent and should run on most other operating systems.

Command Line Basics

Here are basic commands that help you to navigate in Linux using the command line.

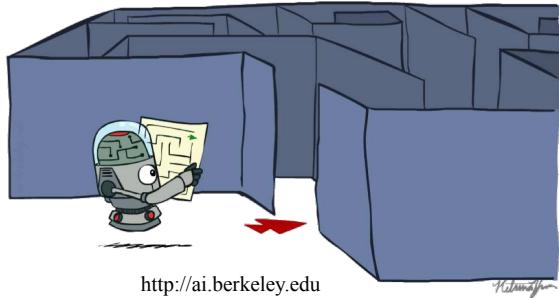
Note: In Windows like operating system, the equivalent command to `ls` is `dir`. The commands of `mkdir` and `cd` are the same.

Q & A



Chapter 1

Search



<http://ai.berkeley.edu>

COMP 3270
Artificial Intelligence

Dirk Schnieders

Outline

- Abstraction
- Types of Search
- Search Problem Definition
- State Space Graph vs. Search Tree
- Search Algorithm (TSA, GSA)
- Uninformed Search
 - BFS, DFS, UCS
- Informed Search
 - Greedy, A*
- Local Search
- Constraint Satisfaction Problems
- Adversarial Search
 - Minimax
 - DLS
 - Horizon Effect
 - α - β Pruning
 - Expectimax
 - Multi-Agent Utilities

Search



<http://ai.berkeley.edu>

Abstraction

- Search problems are models
 - Simplifications of the real world



- Don't deal with the unnecessary complexities of the real world
 - World states vs. search states

Types of Search

- Uninformed search
 - No information about the problem other than its definition is given
- Informed search
 - A heuristic is used that leads to better overall performance in getting to the goal state
- Local Search
 - Evaluate and modify a current state to move closer to a goal state
- Constraint Satisfaction Problems
 - For certain types of problems we can search for solution faster by understanding states better
- Adversarial Search
 - Search in the presence of an adversary

Search Problem Definition

- States: Details of what constitutes a state
- Initial state: The state the agent starts in
- Actions and transition model
 - Description of possible actions available
 - Description of what each action does
- Goal test: Is a given state a goal state?
- Path cost: A function that assigns a zero or positive numeric cost to each path

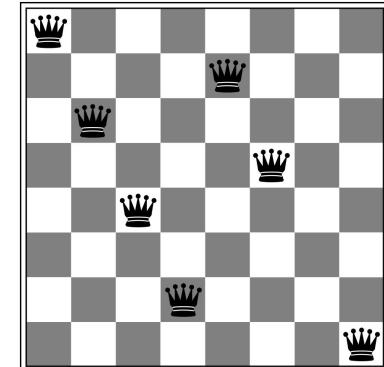
A solution is a sequence of actions (a plan) which transforms the start state to a goal state

N Queens Puzzle - An Example Problem

- Problem Definition A

Eight Queens Puzzle on [Wikipedia](#)

- States
 - Any arrangement of 0 to n queens on the board
- Initial state
 - No queens on the board
- Actions and Transition model
 - Add a queen to an empty square
- Goal test
 - n queens are on the board, none attacked



<http://aima.cs.berkeley.edu/>

N Queens Puzzle - An Example Problem

- Problem Definition B
 - States
 - One queen per column, none attacking another
 - Initial state
 - No queens on the board
 - Actions and Transition model
 - Add a queen to an empty column such that no other queen is under attack
 - Goal test
 - n queens are on the board

State Space

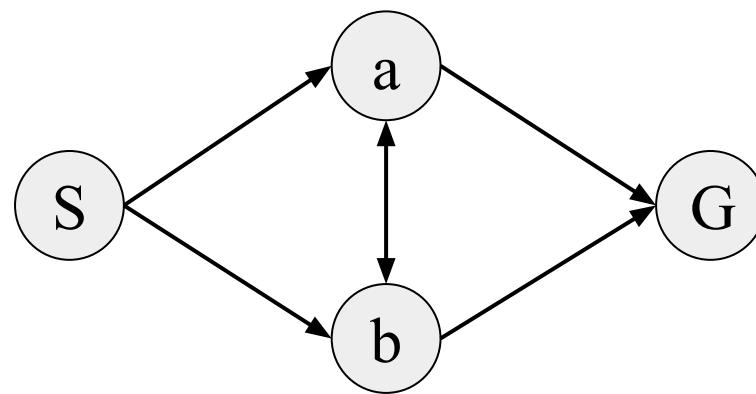
- The set of all states reachable from the initial state by any sequence of actions is the state space
 - Usually a graph
 - The possible action sequences form a search tree
- The nodes are states, and the links between nodes are actions
- A solution is a sequence of actions (i.e., a path) leading from the initial state to a goal state
- Task: Which state space has more states?
 - Problem Definition A
 - Problem Definition B

State Space Graph vs. Search Tree

- State space graph
 - A mathematical representation of a search problem
 - Nodes are (abstracted) world configurations
 - Arcs represent successors (action results)
 - The goal test is a set of goal nodes
 - Each state occurs only once
- Search tree
 - Root of the tree is the start state
 - Nodes represent the possible action sequences
 - States may occur more than once

State Space Graph vs. Search Tree

- Consider the following state space graph
 - Let S be the start state and G be the goal state



- Task: Draw the complete search tree

States vs. State Sequences

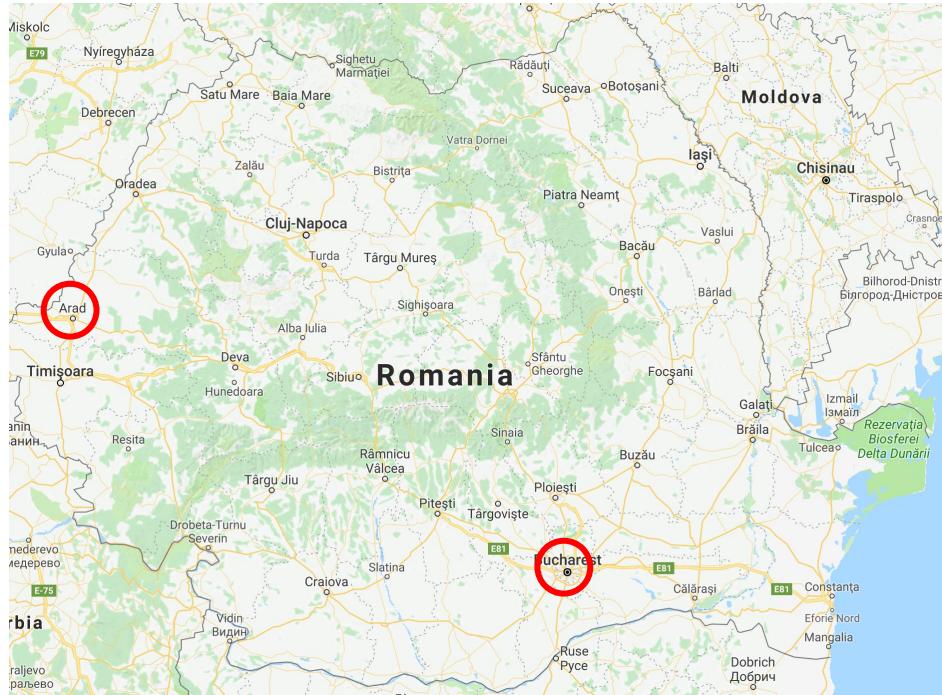
- A large state space results in a huge number of state sequences
 - I.e., a large number of nodes in the search tree
- Example: Chess
 - Claude Shannon estimated 10^{43} possible states and 10^{120} possible state sequences in his 1950 paper “Programming a Computer for Playing Chess”

“There are only 10^{15} total hairs on all the human heads in the world, 10^{23} grains of sand on Earth, and about 10^{81} atoms in the universe. The number of typical chess games is many times as great as all those numbers multiplied together - an impressive feat for 32 wooden pieces lined up on a board.”

Extracted from: Answers to the World's Greatest Questions, By Bjorn Carey

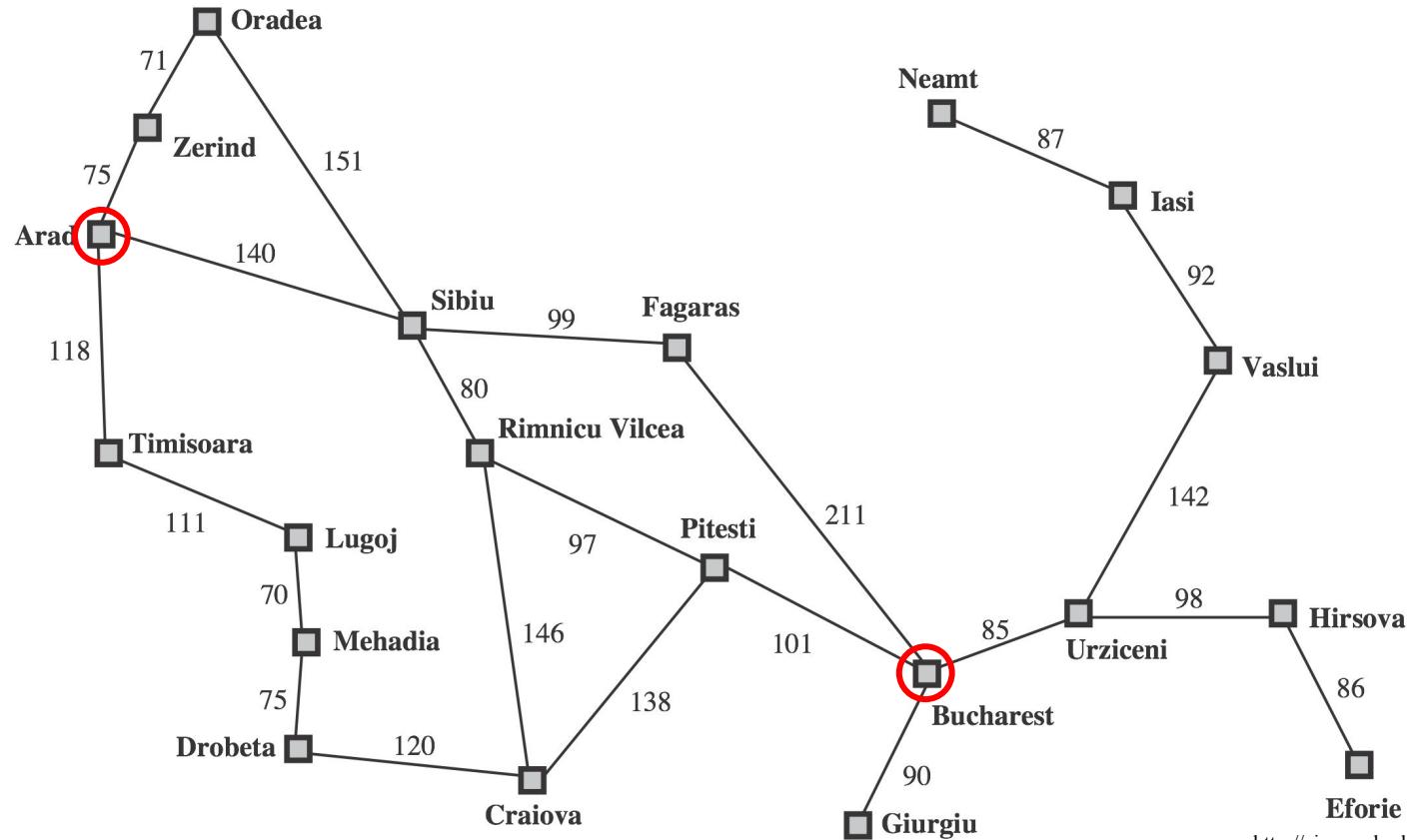
Romania Problem - A Toy Problem

- States
 - The cities
- Initial state
 - Arad
- Actions and Transition model
 - Go to neighboring city
- Goal test
 - In Bucharest?
- Path cost
 - Distance between the cities



www.google.com/maps/place/Romania/

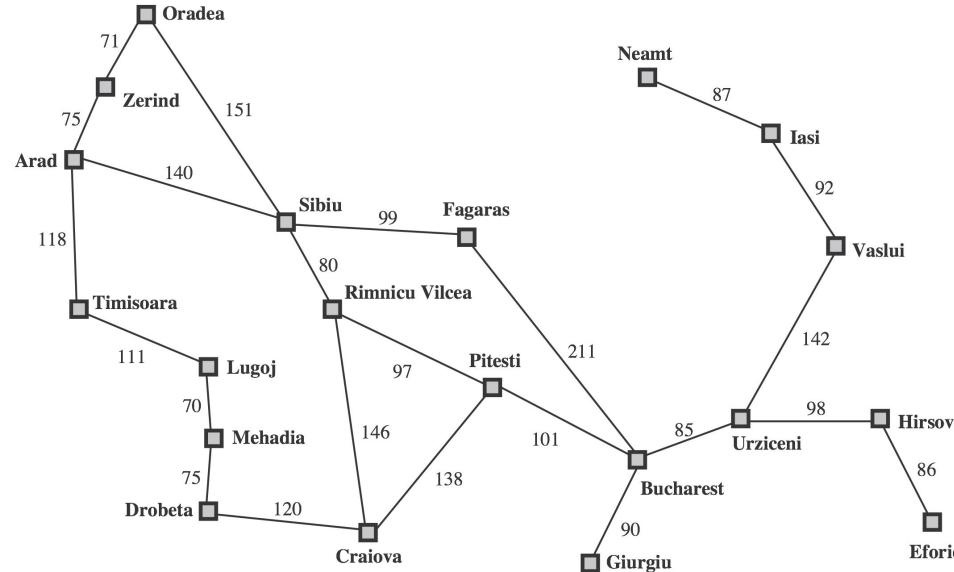
Romania Problem - State Space



Romania Problem Definition

Use dictionary to store neighbors for each cities

```
>>> romania['A']
['S', 'T', 'Z']
```



```
romania = {
    'A':[['S', 'T', 'Z'], 'Z':[['A', 'O'], 'O':[['S', 'Z'], 'T':[['A', 'L'], 'L':[['M', 'T'], 'M':[['D', 'L'], 'D':[['C', 'M'], 'S':[['A', 'F', 'O', 'R'], 'R':[['C', 'P', 'S'], 'C':[['D', 'P', 'R'], 'F':[['B', 'S'], 'P':[['B', 'C', 'R'], 'B':[]}}]]]]]]]]]
```

Search Strategy

- The search strategy defines the order of node expansion
 - I.e., the order in which nodes in the search tree are discovered
- Search strategies are evaluated along the following dimensions
 - Completeness: Always find a solution if one exists?
 - Optimality: Always find a least-cost solution?
 - Time complexity: Number of nodes generated (= edges traversed)
 - Space complexity: Max number of nodes in memory
- Time / space complexity are measured in terms of
 - b : maximum branching factor of the search tree
 - d : distance to root of the shallowest solution
 - m : maximum length of any path in the state space

Uninformed (Blind) Search Strategies

- Uninformed search strategies (also called blind search) use only the information available in the problem definition
- Examples
 - Breadth-first search (BFS)
 - Depth-first search (DFS)
 - Uniform-cost search (UCS)

Search Algorithms

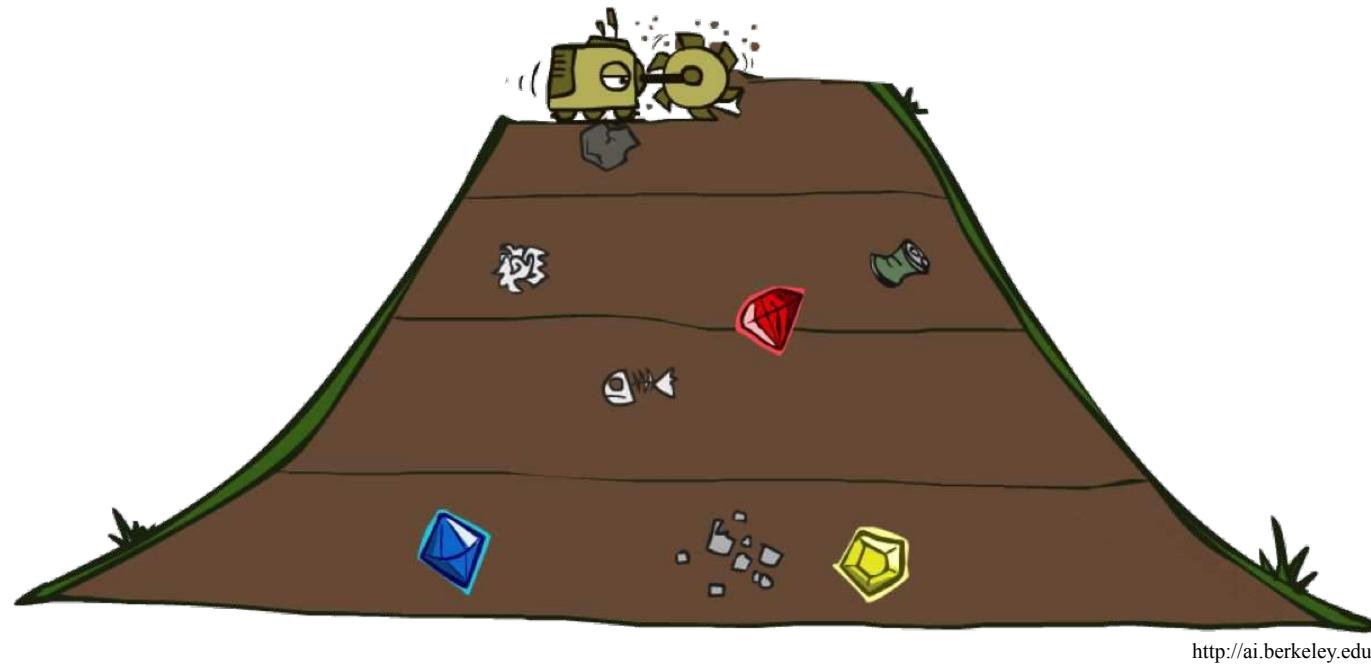
- Search algorithms come in two different flavors
 - Tree search algorithm (TSA)
 - Graph search algorithm (GSA)

Tree Search Algorithm (TSA)

```
function TSA(problem) returns solution
    initialize frontier using initial state of problem
    while frontier is not empty
        choose a node and remove it from frontier
        if node contains a goal state then return corresponding solution
        explore the node, adding the resulting nodes to the frontier
```

BFS

- The BFS search strategy explores the shallowest node in the search tree



<http://ai.berkeley.edu>

frontier is a queue (first in first out data structure)

Queue in Python

- We are going to use the deque in Python
 - Example usage:

```
>>> import collections
>>> queue = collections.deque(['A', 'B', 'C', 'D'])
>>> queue.popleft()
'A'
>>> queue
deque(['B', 'C', 'D'])
>>> queue.popleft()
'B'
>>> queue
deque(['C', 'D'])
```

Examples

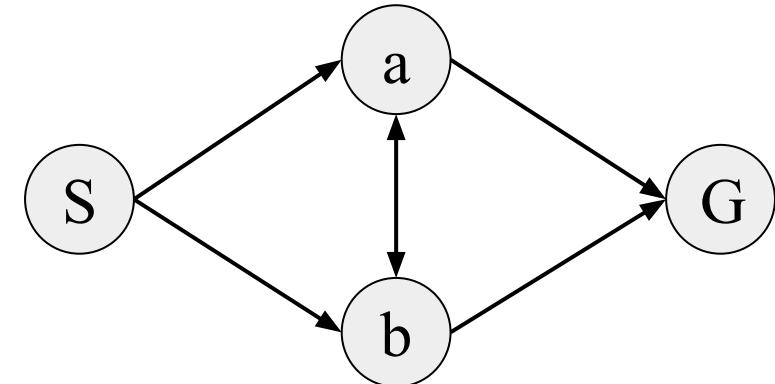
- In the following examples we will use

- characters for states, e.g,

- 'S'
 - 'a'

- strings for state sequences

- 'Sa'
 - 'SabG'



Tree Search Algorithm (TSA) - BFS Version

```
function TSA(problem) returns solution
    initialize frontier using initial state of problem
    while frontier is not empty
        choose a node and remove it from frontier
        if node contains a goal state then return corresponding solution
        explore the node, adding the resulting nodes to the frontier
```

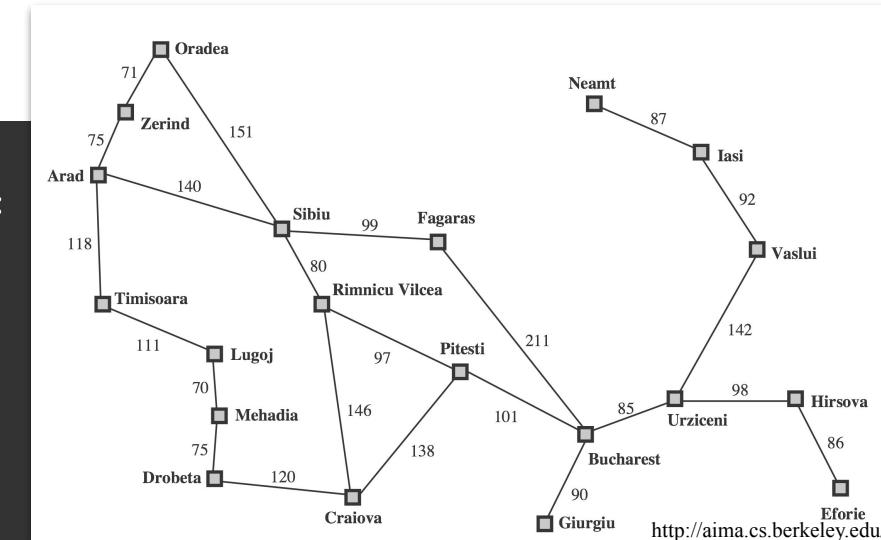
```
import collections
def bfsTsa(stateSpaceGraph, startState, goalState):
    frontier = collections.deque([startState])
    while frontier:
        node = frontier.popleft()
        if (node.endswith(goalState)): return node
        for child in stateSpaceGraph[node[-1]]: frontier.append(node+child)
```

BFS-TSA Romania Code

```

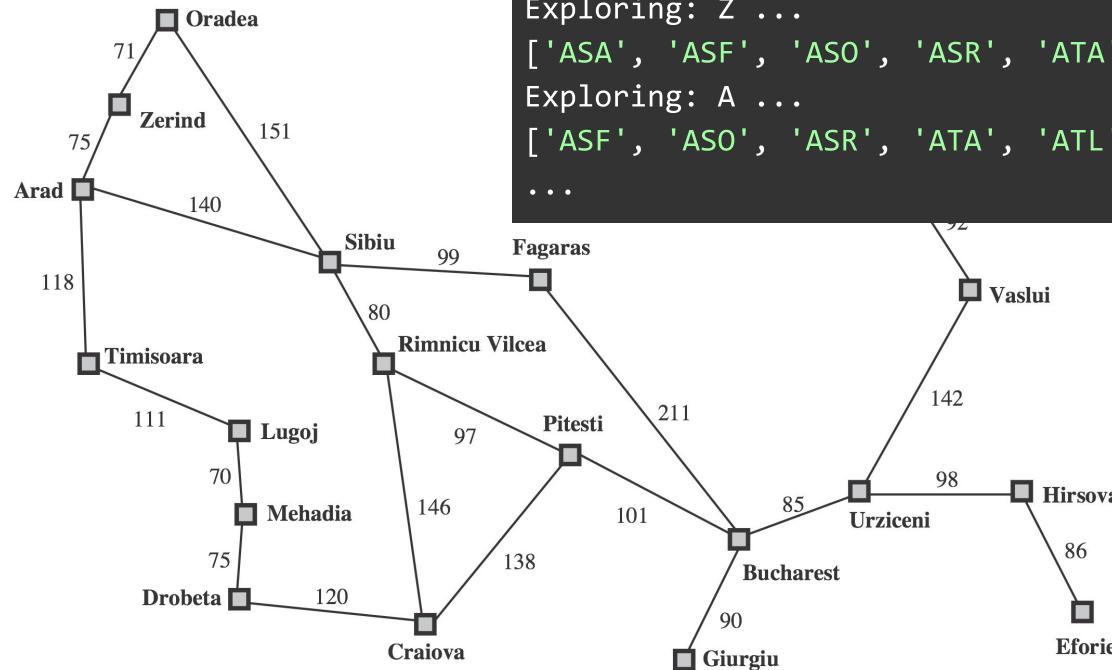
import collections
def bfsTsa(stateSpaceGraph, startState, goalState):
    frontier = collections.deque([startState])
    print('Initial frontier:',list(frontier))
    input()
    while frontier:
        node = frontier.popleft()
        if (node.endswith(goalState)): return node
        print('Exploring:',node[-1], '...')
        for child in stateSpaceGraph[node[-1]]: frontier.append(node+child)
        print(list(frontier))
        input()
    romania = {
        'A':[['S','T','Z'],['A','O'],['S','Z'],['A','L'],['M','T'],['D','L'],
        'D':[['C','M'],['A','F','O','R'],['C','P','S'],['D','P','R'],
        'F':[['B','S'],['B','C','R']],
        'P':[],
        'B':[]
    }
    print('Solution path:',bfsTsa(romania, 'A', 'B'))

```



<http://aima.cs.berkeley.edu/>

BFS-TSA Romania



```
Initial frontier: ['A']
Exploring: A ...
['AS', 'AT', 'AZ']
Exploring: S ...
['AT', 'AZ', 'ASA', 'ASF', 'ASO', 'ASR']
Exploring: T ...
['AZ', 'ASA', 'ASF', 'ASO', 'ASR', 'ATA', 'ATL']
Exploring: Z ...
['ASA', 'ASF', 'ASO', 'ASR', 'ATA', 'ATL', 'AZA', 'AZO']
Exploring: A ...
['ASF', 'ASO', 'ASR', 'ATA', 'ATL', 'AZA', 'AZO', 'ASAS', 'ASAT', 'ASAZ']
...
...
```

Will `bfsTsa(romania, 'A', 'B')` terminate ?

Graph Search Algorithm (GSA) - BFS Version

```
function GSA (problem) returns solution
    initialize frontier using initial state of problem
    initialize explored set to be empty
    while frontier is not empty
        choose a node and remove it from frontier
        if node contains a goal state then return corresponding solution
        If node is not in explored set
            add node to explored set
            explore the node, adding the resulting nodes to the frontier
```

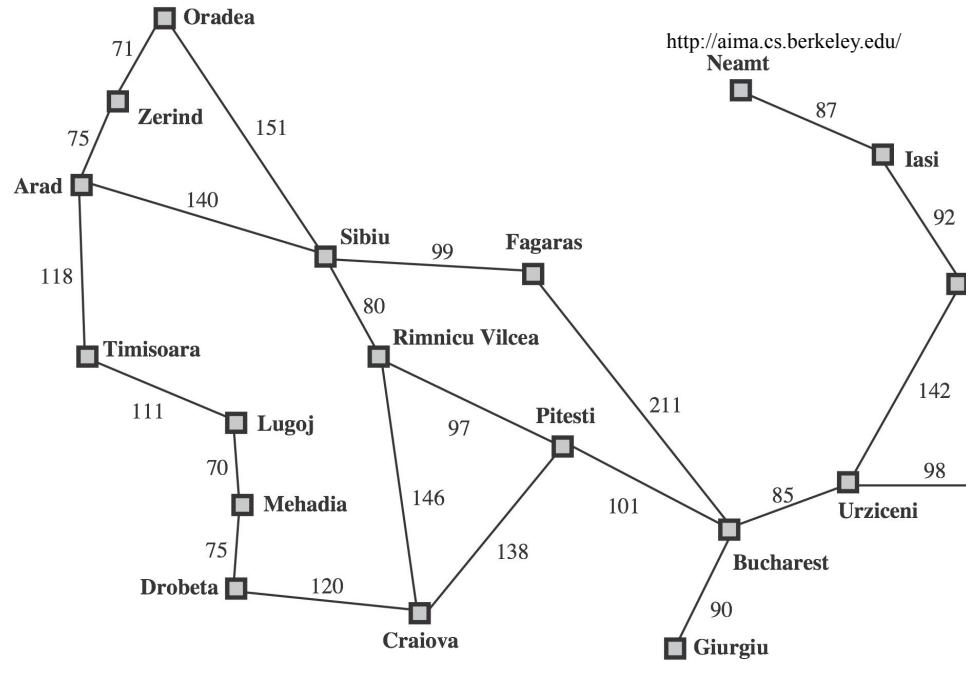
```
import collections
def bfsGsa(stateSpaceGraph, startState, goalState):
    frontier = collections.deque([startState])
    exploredSet = set()
    while frontier:
        node = frontier.popleft()
        if (node.endswith(goalState)): return node
        if node[-1] not in exploredSet:
            exploredSet.add(node[-1])
            for child in stateSpaceGraph[node[-1]]: frontier.append(node+child)
```

```
import collections
def bfsGsa(stateSpaceGraph, startState, goalState):
    frontier = collections.deque([startState])
    exploredSet = set()
    print('Initial frontier:',list(frontier))
    input()
    while frontier:
        node = frontier.popleft()
        if (node.endswith(goalState)): return node
        if node[-1] not in exploredSet:
            print('Exploring:',node[-1],...)
            exploredSet.add(node[-1])
            for child in stateSpaceGraph[node[-1]]: frontier.append(node+child)
            print(list(frontier))
            print(exploredSet)
            input()
    romania = {
        'A':[['S','T','Z'],['Z'],['O'],[],[]],
        'T':[['A','L'],[],[],[],[]],
        'L':[['M','T'],[],[],[],[]],
        'M':[['D','L'],[],[],[],[]],
        'D':[['C','M'],[],[],[],[]],
        'C':[['A','F','O','R'],[],[],[],[]],
        'F':[['B','S'],[],[],[],[]],
        'R':[['C','P','S'],[],[],[],[]],
        'P':[['B','C','R'],[],[],[],[]],
        'B':[]
    }
    print('Solution path:',bfsGsa(romania, 'A', 'B'))
```

```

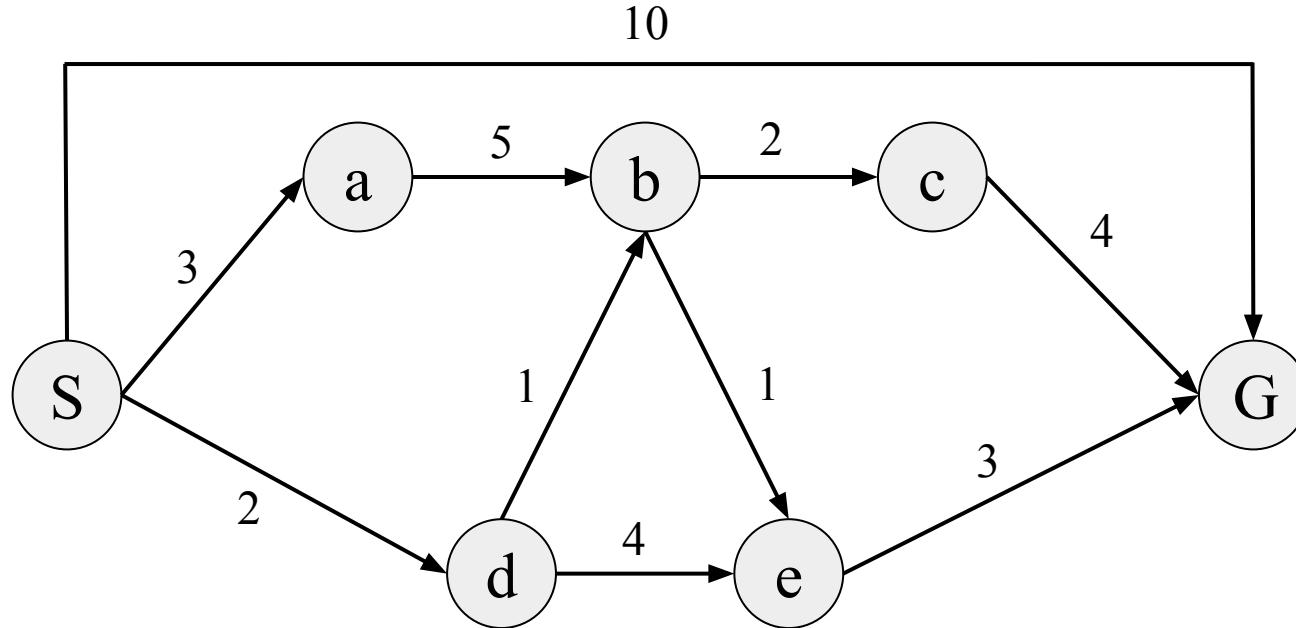
Initial frontier: ['A']
Exploring: A ...
['AS', 'AT', 'AZ']
{'A'}
Exploring: S ...
['AT', 'AZ', 'ASA', 'ASF', 'ASO', 'ASR']
{'S', 'A'}
Exploring: T ...
['AZ', 'ASA', 'ASF', 'ASO', 'ASR', 'ATA', 'ATL']
{'S', 'A', 'T'}
Exploring: Z ...
['ASA', 'ASF', 'ASO', 'ASR', 'ATA', 'ATL', 'AZA', 'AZO']
{'S', 'A', 'Z', 'T'}
Exploring: F ...
['ASO', 'ASR', 'ATA', 'ATL', 'AZA', 'AZO', 'ASFB', 'ASFS']
{'A', 'Z', 'T', 'S', 'F'}
Exploring: O ...
['ASR', 'ATA', 'ATL', 'AZA', 'AZO', 'ASFB', 'ASFS', 'ASOS', 'ASOZ']
{'A', 'Z', 'T', 'S', 'O', 'F'}
Exploring: R ...
['ATA', 'ATL', 'AZA', 'AZO', 'ASFB', 'ASFS', 'ASOS', 'ASOZ', 'ASRC', 'ASRP', 'ASRS']
{'A', 'R', 'Z', 'T', 'S', 'O', 'F'}
Exploring: L ...
['AZA', 'AZO', 'ASFB', 'ASFS', 'ASOS', 'ASOZ', 'ASRC', 'ASRP', 'ASRS', 'ATLM', 'ATLT']
{'A', 'R', 'L', 'Z', 'T', 'S', 'O', 'F'}
Solution path: ASFB

```



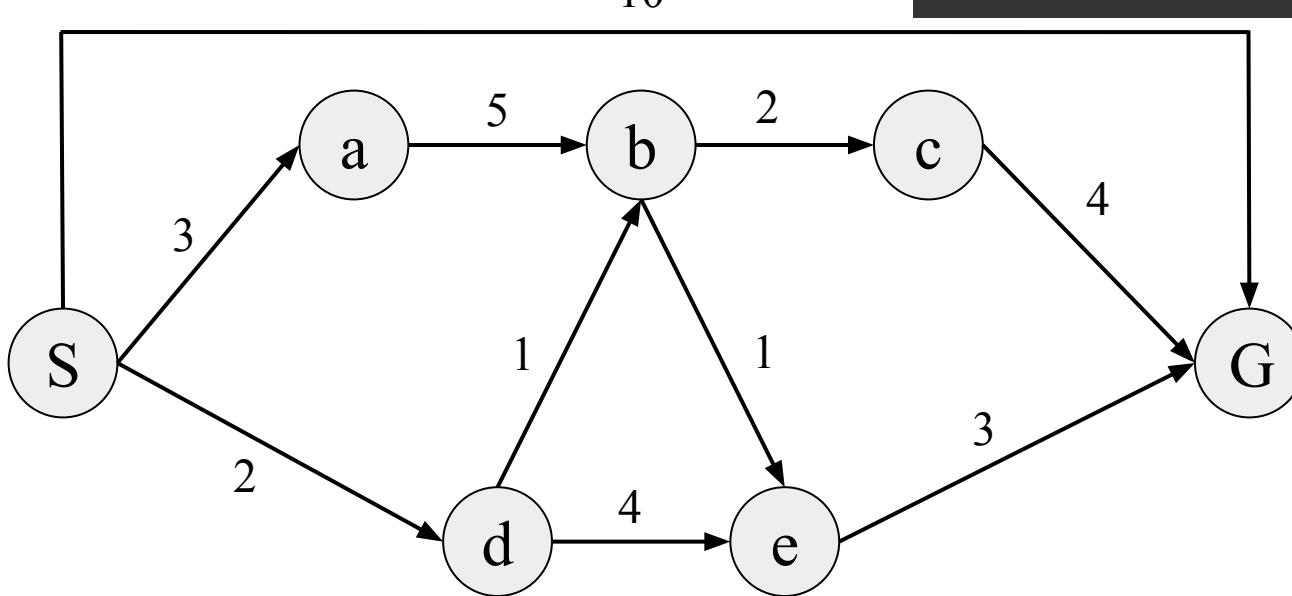
BFS-GSA Practice

- Consider the following state space graph
 - Let S be the start state and G be the goal state
- Perform **BFS-GSA** and write down the order of explored states
 - Assume children are added into the frontier in alphabetical order



BFS-GSA Practice Solution

```
Initial frontier: ['S']
Exploring: S ...
['Sa', 'Sd', 'SG']
{'S'}
Exploring: a ...
['Sd', 'SG', 'Sab']
{'S', 'a'}
Exploring: d ...
['SG', 'Sab', 'Sdb', 'Sde']
{'S', 'a', 'd'}
Solution path: SG
```



Try it [here](#)
[bfsTsaPractice.py](#)
[bfsGsaPractice.py](#)

BFS Properties

- Complete
- Optimal
- Complexity
 - Time
 - Space

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

<http://aima.cs.berkeley.edu/>

b: maximum branching factor of the search tree

d: depth of the shallowest solution

DFS

- DFS explores the deepest node in the search tree



<http://ai.berkeley.edu>

frontier is a stack (last in first out data structure)

Stack in Python

- We are going to use the deque in Python
 - Example usage:

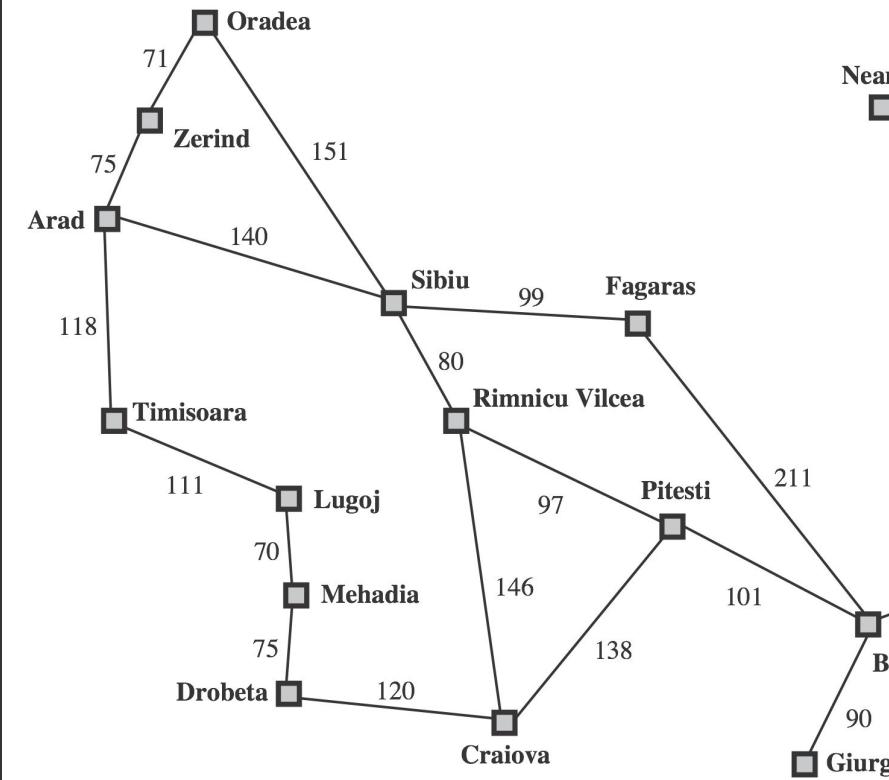
```
>>> import collections
>>> queue = collections.deque(['A', 'B', 'C', 'D'])
>>> queue.pop()
'D'
>>> queue
deque(['A', 'B', 'C'])
>>> queue.pop()
'C'
>>> queue
deque(['A', 'B'])
```

```
import collections
def dfsGsa(stateSpaceGraph, startState, goalState):
    frontier = collections.deque([startState])
    exploredSet = set()
    print('Initial frontier:',list(frontier))
    input()
    while frontier:
        node = frontier.pop()
        if (node.endswith(goalState)): return node
        if node[-1] not in exploredSet:
            print('Exploring:',node[-1],...)
            exploredSet.add(node[-1])
            for child in stateSpaceGraph[node[-1]]: frontier.append(node+child)
            print(list(frontier))
            print(exploredSet)
            input()
romania = {
    'A':['S','T','Z'],'Z':['A','O'],'O':['S','Z'],'T':['A','L'],'L':['M','T'],'M':['D','L'],
    'D':['C','M'],'S':['A','F','O','R'],'R':['C','P','S'],'C':['D','P','R'],
    'F':['B','S'],'P':['B','C','R'],'B':[] }
print('Solution path:',dfsGsa(romania, 'A', 'B'))
```

```

Initial frontier: ['A']
Exploring: A ...
['AS', 'AT', 'AZ']
{'A'}
Exploring: Z ...
['AS', 'AT', 'AZA', 'AZO']
{'Z', 'A'}
Exploring: O ...
['AS', 'AT', 'AZA', 'AZOS', 'AZOZ']
{'Z', 'O', 'A'}
Exploring: S ...
['AS', 'AT', 'AZA', 'AZOSA', 'AZOSF', 'AZOSO', 'AZOSR']
{'Z', 'O', 'S', 'A'}
Exploring: R ...
['AS', 'AT', 'AZA', 'AZOSRC', 'AZOSRP', 'AZOSRS']
{'A', 'Z', 'R', 'O', 'S'}
Exploring: P ...
['AS', 'AT', 'AZA', 'AZOSRPB', 'AZOSRPC', 'AZOSRPR']
{'A', 'P', 'Z', 'R', 'O', 'S'}
Exploring: C ...
['AS', 'AT', 'AZA', 'AZOSRPCD', 'AZOSRPCP', 'AZOSRPCR']
{'C', 'A', 'P', 'Z', 'R', 'O', 'S'}
Exploring: D ...
['AS', 'AT', 'AZA', 'AZOSRPB', 'AZOSRPCDC', 'AZOSRPCDM']
{'D', 'C', 'A', 'P', 'Z', 'R', 'O', 'S'}

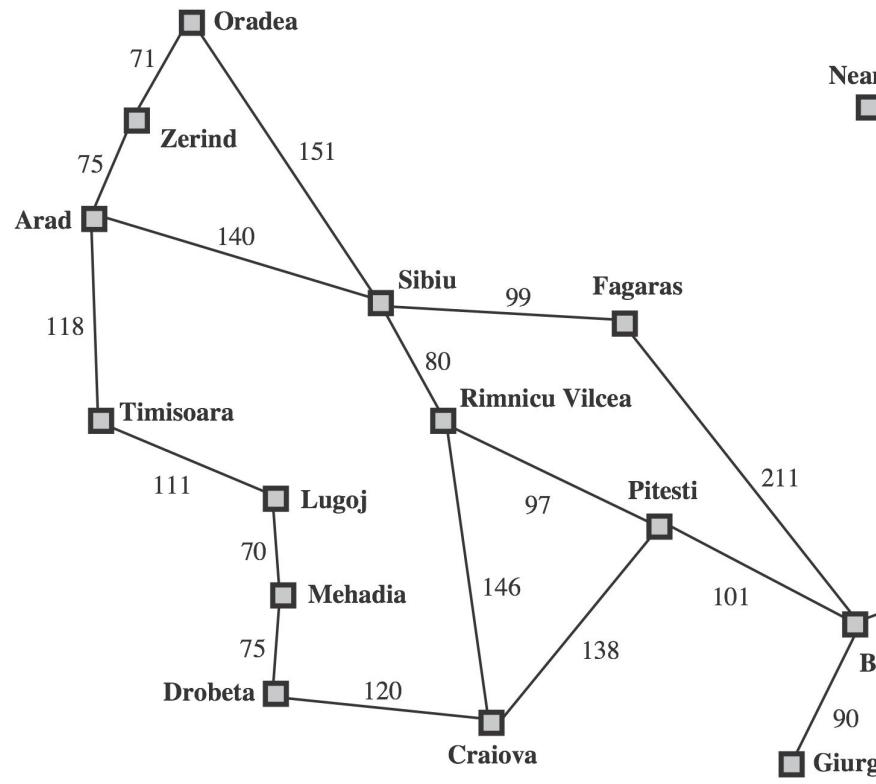
```



```

...
Exploring: M ...
['AS', 'AT', 'AZA', --- 'AZOSRPCDC', 'AZOSRPCDMD', 'AZOSRPCDML'
{'D', 'C', 'A', 'P', 'Z', 'R', 'O', 'S', 'M'}
Exploring: L ...
['AS', 'AT', 'AZA', --- 'AZOSRPCDMD', 'AZOSRPCDMLM', 'AZOSRPCDMLT']
{'D', 'C', 'A', 'P', 'Z', 'R', 'L', 'O', 'S', 'M'}
Exploring: T ...
['AS', 'AT', 'AZA', --- 'AZOSRPCDMLM', 'AZOSRPCDMLTA', 'AZOSRPCDMLTL']
{'D', 'C', 'A', 'P', 'Z', 'T', 'R', 'L', 'O', 'S', 'M'}
Solution path: AZOSRPB

```

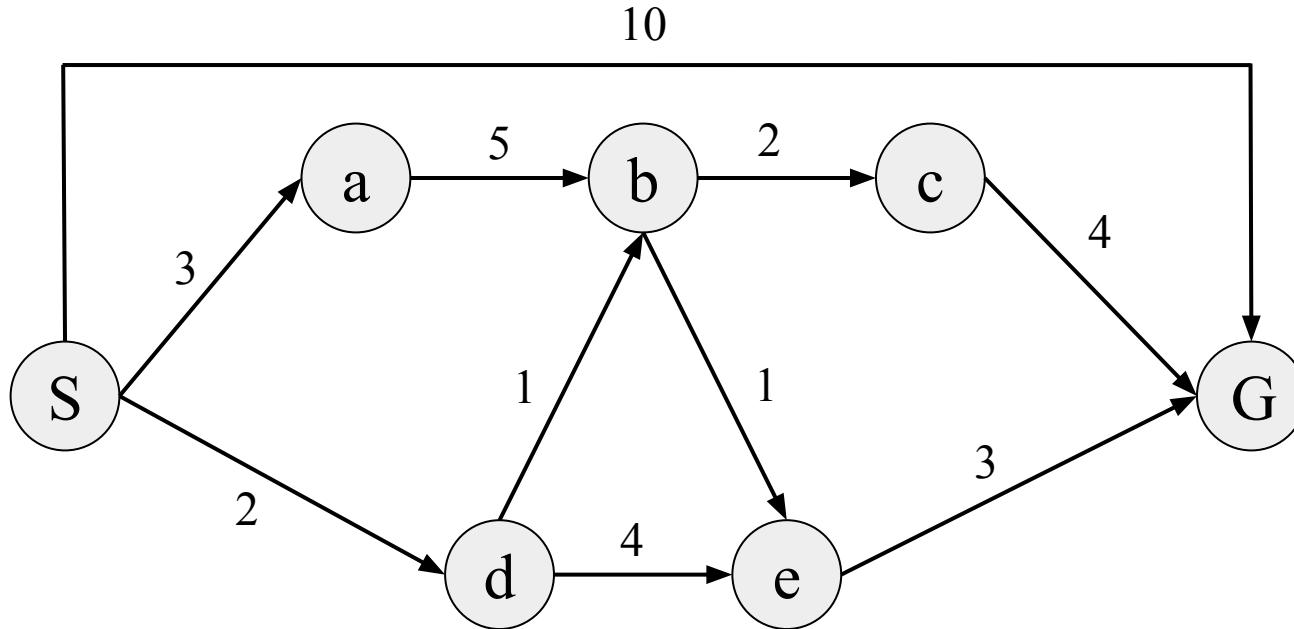


<http://aima.cs.berkeley.edu/>

Try it [here](#)
[dfsTsaRomania.py](#)
[dfsGsaRomania.py](#)

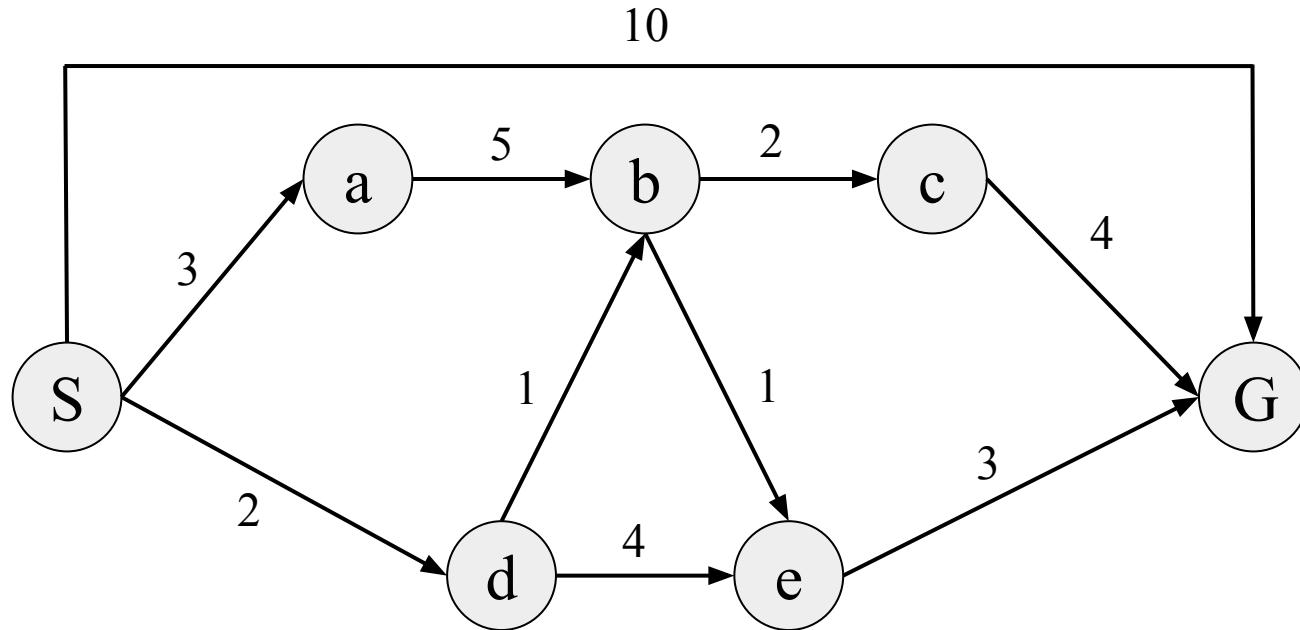
DFS-TSA Practice

- Consider the following state space graph
 - Let S be the start state and G be the goal state
- Perform **DFS-TSA** and write down the order of explored states
 - Assume children are added into the frontier in alphabetical order



DFS-TSA Practice Solution

```
Initial frontier: ['S']
Exploring: S ...
['Sa', 'Sd', 'SG']
Solution path: SG
```



DFS Properties

- Complete
- Optimal
- Complexity
 - Time
 - Space

b: maximum branching factor of the search tree

m: maximum length of any path in the state space

BFS vs. DFS

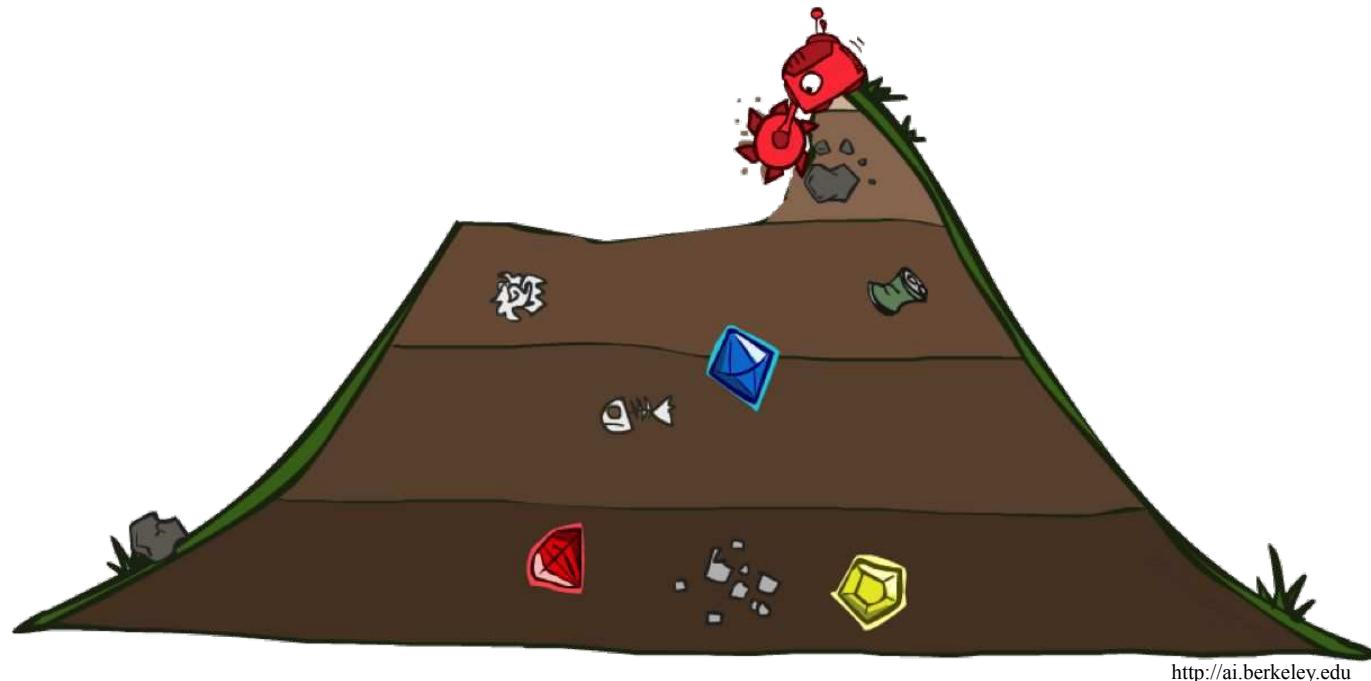
- When will BFS outperform DFS?
 - If solutions are close to the root of the search tree
- When will DFS outperform BFS?
 - If all solutions are deep inside the search tree

GSA vs. TSA

- GSA
 - Avoids infinite loops
 - Eliminates exponentially many redundant paths
 - Requires memory proportional to its runtime
- TSA
 - Could be stuck in infinite loops
 - Explores redundant paths
 - Requires less memory
 - Easier to implement

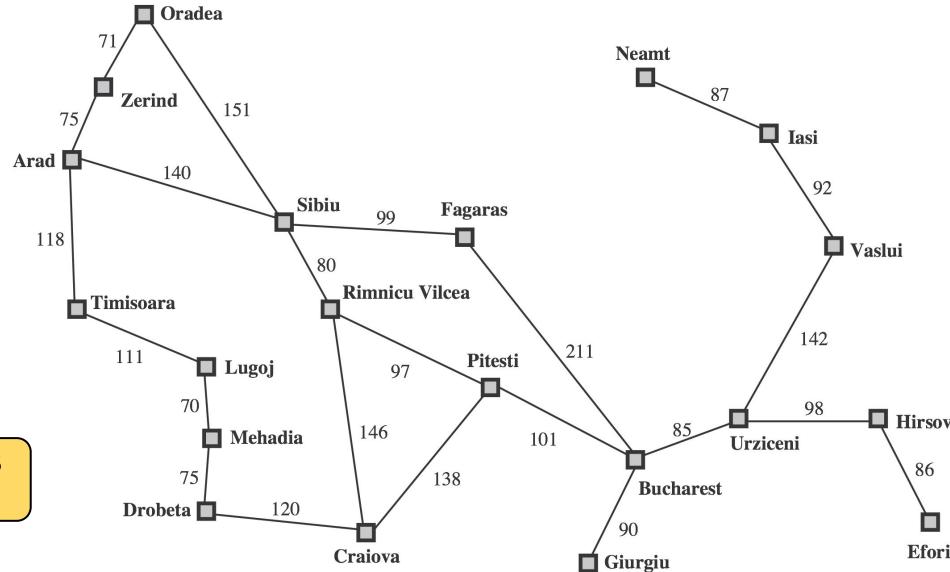
UCS

- UCS explores the cheapest node first



frontier is a priority queue (queue data structure where each element has a priority)

Updated Romania Problem Definition



Updated version includes
the distances

This is the cost
from O to Z

```
romania = {
    'A':[(140, 'S'), (118, 'T'), (75, 'Z')], 'Z':[(75, 'A'), (71, 'O')], 'O':[(151, 'S'), (71, 'Z')],
    'T':[(118, 'A'), (111, 'L')], 'L':[(70, 'M'), (111, 'T')], 'M':[(75, 'D'), (70, 'L')],
    'D':[(120, 'C'), (75, 'M')], 'S':[(140, 'A'), (99, 'F'), (151, 'O'), (80, 'R')], 'F':[(211, 'B'), (99, 'S')],
    'R':[(146, 'C'), (97, 'P'), (80, 'S')], 'C':[(120, 'D'), (138, 'P'), (146, 'R')], 'P':[(101, 'B'), (138, 'C'), (97, 'R')], 'B':[]}
```

Priority Queue in Python

We will use this library
for our Priority Queue

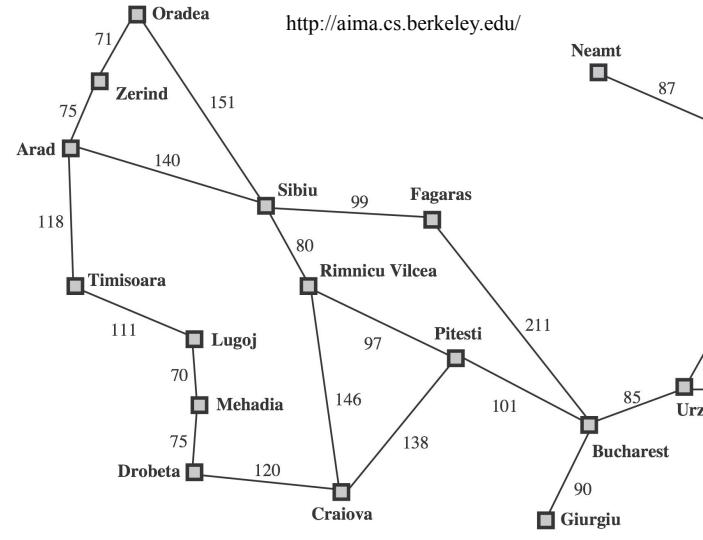
```
frontier = []
>>> from heapq import heappush, heappop
import random
>>> heappush(frontier, (random.randint(1, 10), 'A'))
>>> heappush(frontier, (random.randint(1, 10), 'B'))
>>> heappush(frontier, (random.randint(1, 10), 'C'))
>>> frontier
[(2, 'C'), (7, 'B'), (6, 'A')]
>>> heappop(frontier)
(2, 'C')
>>> heappop(frontier)
(6, 'A')
>>> heappop(frontier)
(7, 'B')
>>> (1, 'B') < (2, 'A')
True
>>> (1, 'B') < (1, 'A')
False
```

This is how tuples are
compared in Python

```

from heapq import heappush, heappop
def ucsGsa(stateSpaceGraph, startState, goalState):
    frontier = []
    heappush(frontier, (0, startState))
    exploredSet = set()
    print('Initial frontier:', list(frontier)); input()
    while frontier:
        node = heappop(frontier)
        if (node[1].endswith(goalState)): return node
        if node[1][-1] not in exploredSet:
            print('Exploring:', node[1][-1], 'at cost', node[0])
            exploredSet.add(node[1][-1])
            for child in stateSpaceGraph[node[1][-1]]:
                heappush(frontier, (node[0]+child[0], node[1]+child[1]))
            print(list(frontier)); print(exploredSet); input()
romania = {
'A':[(140, 'T'),(118, 'T'),(75, 'Z')], 'Z':[(75, 'A'),(71, 'O')], 'O':[(151, 'S'),(71, 'Z')], 
'T':[(118, 'A'),(111, 'L')], 'L':[(70, 'M'),(111, 'T')], 'M':[(75, 'D'),(70, 'L')], 
'D':[(120, 'C'),(75, 'M')], 'S':[(140, 'A'),(99, 'F'),(151, 'O'),(80, 'R')], 
'R':[(146, 'C'),(97, 'P'),(80, 'S')], 'C':[(120, 'D'),(138, 'P'),(146, 'R')], 
'F':[(211, 'B'),(99, 'S')], 'P':[(101, 'B'),(138, 'C'),(97, 'R')], 'B':[]}
print('Solution path:', ucsGsa(romania, 'A', 'B'))

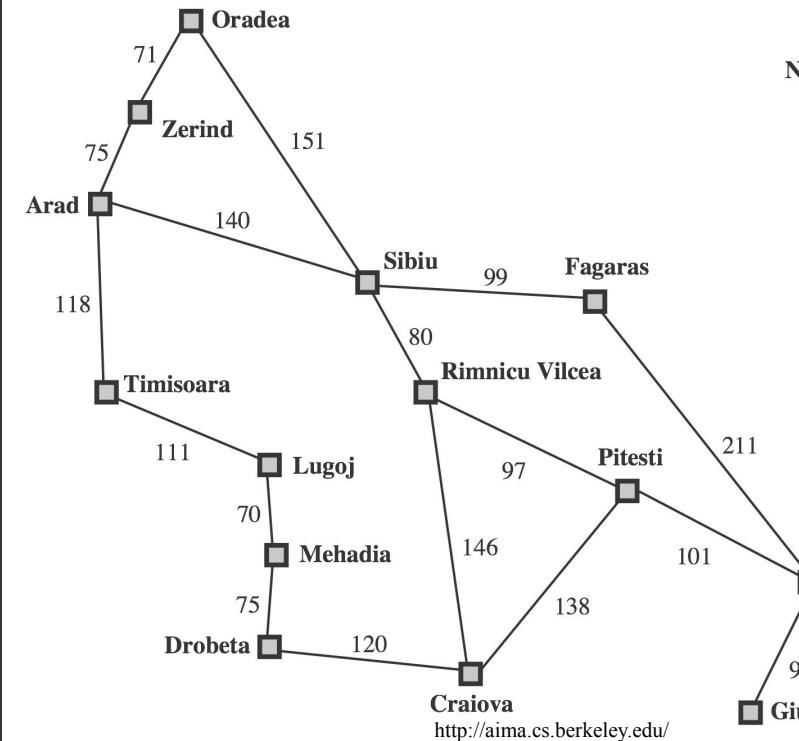
```



```

Initial frontier: [(0, 'A')]
Exploring: A at cost 0
[(75, 'AZ'), (140, 'AS'), (118, 'AT')]
{'A'}
Exploring: Z at cost 75
[(118, 'AT'), (140, 'AS'), (150, 'AZA'), (146, 'AZO')]
{'Z', 'A'}
Exploring: T at cost 118
[(140, 'AS'), (146, 'AZO'), (150, 'AZA'), (236, 'ATA'), (229, 'ATL')]
{'Z', 'T', 'A'}
Exploring: S at cost 140
[(146, 'AZO'), (220, 'ASR'), --- (291, 'ASO'), (236, 'ATA')]
{'Z', 'T', 'A', 'S'}
Exploring: O at cost 146
[(150, 'AZA'), (217, 'AZOZ'), --- (297, 'AZOS'), (229, 'ATL')]
{'A', 'O', 'S', 'Z', 'T'}
Exploring: R at cost 220
[(229, 'ATL'), (280, 'ASA'), --- (317, 'ASRP'), (300, 'ASRS')]
{'R', 'A', 'O', 'S', 'Z', 'T'}
Exploring: L at cost 229
[(236, 'ATA'), (280, 'ASA'), --- (299, 'ATLM'), (340, 'ATLT')]
{'R', 'A', 'O', 'S', 'L', 'Z', 'T'}
Exploring: F at cost 239
[(280, 'ASA'), (291, 'ASO'), --- (450, 'ASFB'), (338, 'ASFS')]
{'R', 'A', 'O', 'S', 'L', 'F', 'Z', 'T'}

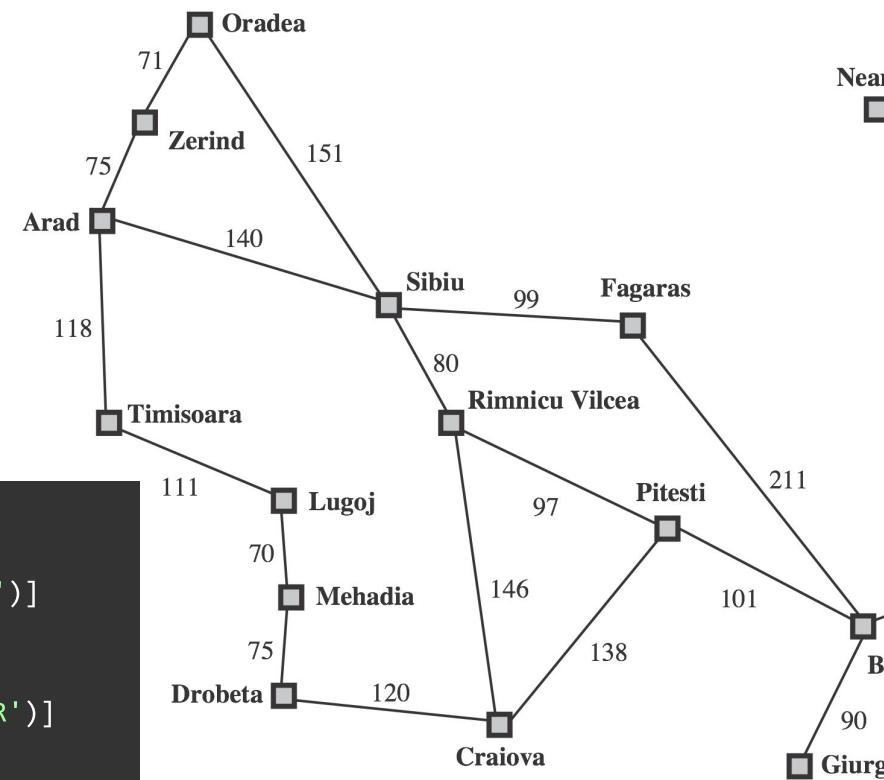
```



```

...
Exploring: M at cost 299
[(300, 'ASRS'), (317, 'ASRP'), --- (374, 'ATLMD'), (369, 'ATMLM')]
{'M', 'R', 'A', 'O', 'S', 'L', 'F', 'Z', 'T'}
Exploring: P at cost 317
[(338, 'ASFS'), (369, 'ATMLM'), --- (455, 'ASRPC'), (414, 'ASPRP')]
{'M', 'P', 'R', 'A', 'O', 'S', 'L', 'F', 'Z', 'T'}
Exploring: C at cost 366
[(369, 'ATMLM'), (374, 'ATLMD'), --- (504, 'ASRCP'), (512, 'ASRCR')]
{'M', 'P', 'R', 'A', 'O', 'S', 'L', 'F', 'Z', 'T', 'C'}
Exploring: D at cost 374
[(414, 'ASPRP'), (418, 'ASRPB'), --- (504, 'ASRCP'), (494, 'ATLMDC')]
{'M', 'P', 'R', 'A', 'D', 'O', 'S', 'L', 'F', 'Z', 'T', 'C'}
Solution path: (418, 'ASRPB')

```

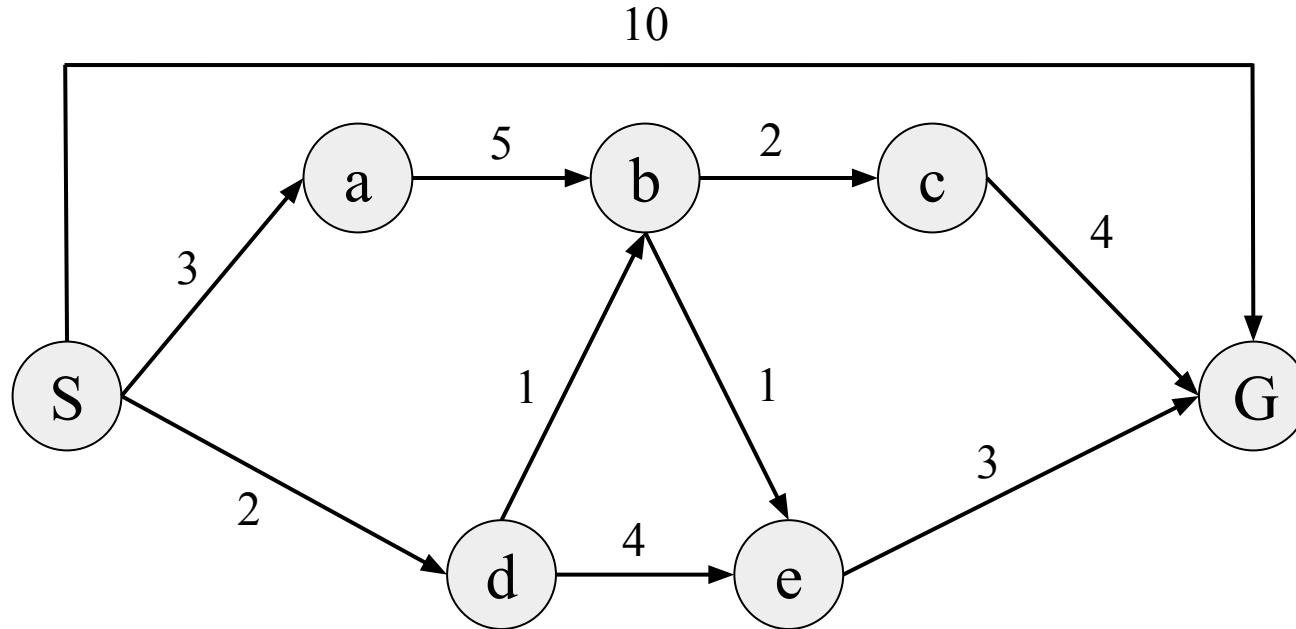


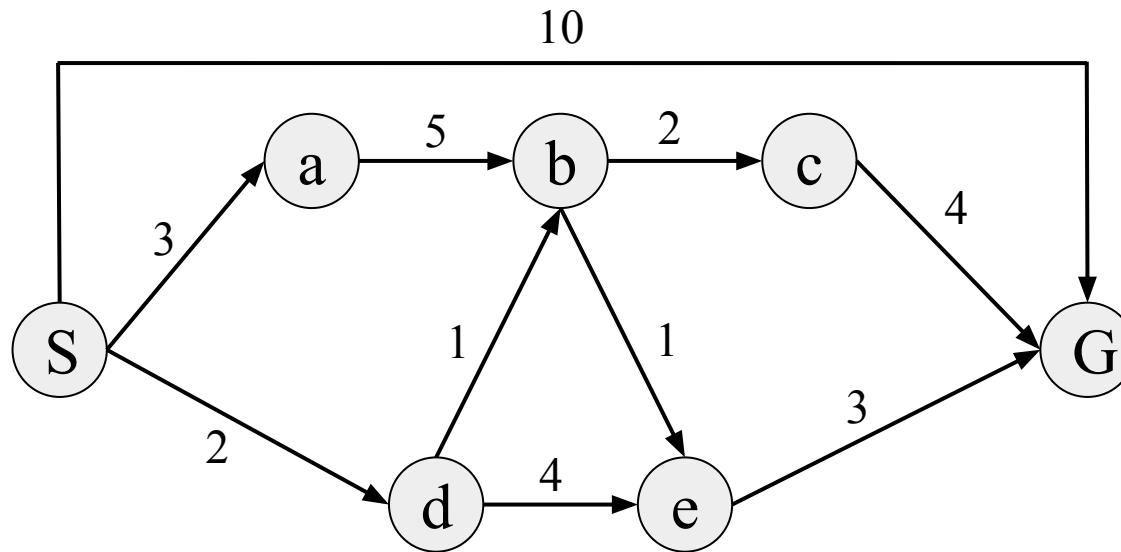
<http://aima.cs.berkeley.edu/>

Try it [here](#)
[ucsTsaRomania.py](#)
[ucsGsaRomania.py](#)

UCS-GSA Practice

- Consider the following state space graph
 - Let S be the start state and G be the goal state
- Perform **UCS-GSA** and write down the order of explored states
 - Assume children are added into the frontier in alphabetical order

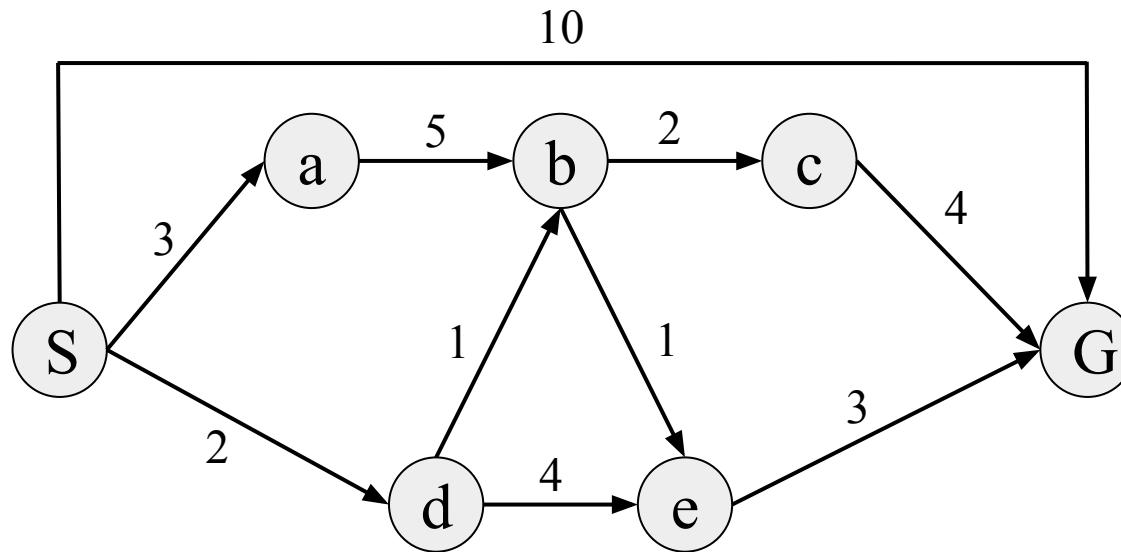




```

Initial frontier: [(0, 'S')]
Exploring: S at cost 0
[(2, 'Sd'), (3, 'Sa'), (10, 'SG')]
{'S'}
Exploring: d at cost 2
[(3, 'Sa'), (6, 'Sde'), (3, 'Sdb'), (10, 'SG')]
{'S', 'd'}
Exploring: a at cost 3
[(3, 'Sdb'), (6, 'Sde'), (10, 'SG'), (8, 'Sab')]
{'S', 'a', 'd'}

```



```

...
Exploring: b at cost 3
[(4, 'Sdbe'), (5, 'Sdbc'), (10, 'SG'), (8, 'Sab'), (6, 'Sde')]
{'b', 'S', 'a', 'd'}
Exploring: e at cost 4
[(5, 'Sdbc'), (6, 'Sde'), (10, 'SG'), (8, 'Sab'), (7, 'SdbeG')]
{'S', 'e', 'd', 'a', 'b'}
Exploring: c at cost 5
[(6, 'Sde'), (7, 'SdbeG'), (10, 'SG'), (8, 'Sab'), (9, 'SdbcG')]
{'S', 'e', 'd', 'c', 'a', 'b'}
Solution path: (7, 'SdbeG')

```

Try it [here](#)
[ucsTsaPractice.py](#)
[ucsGsaPractice.py](#)

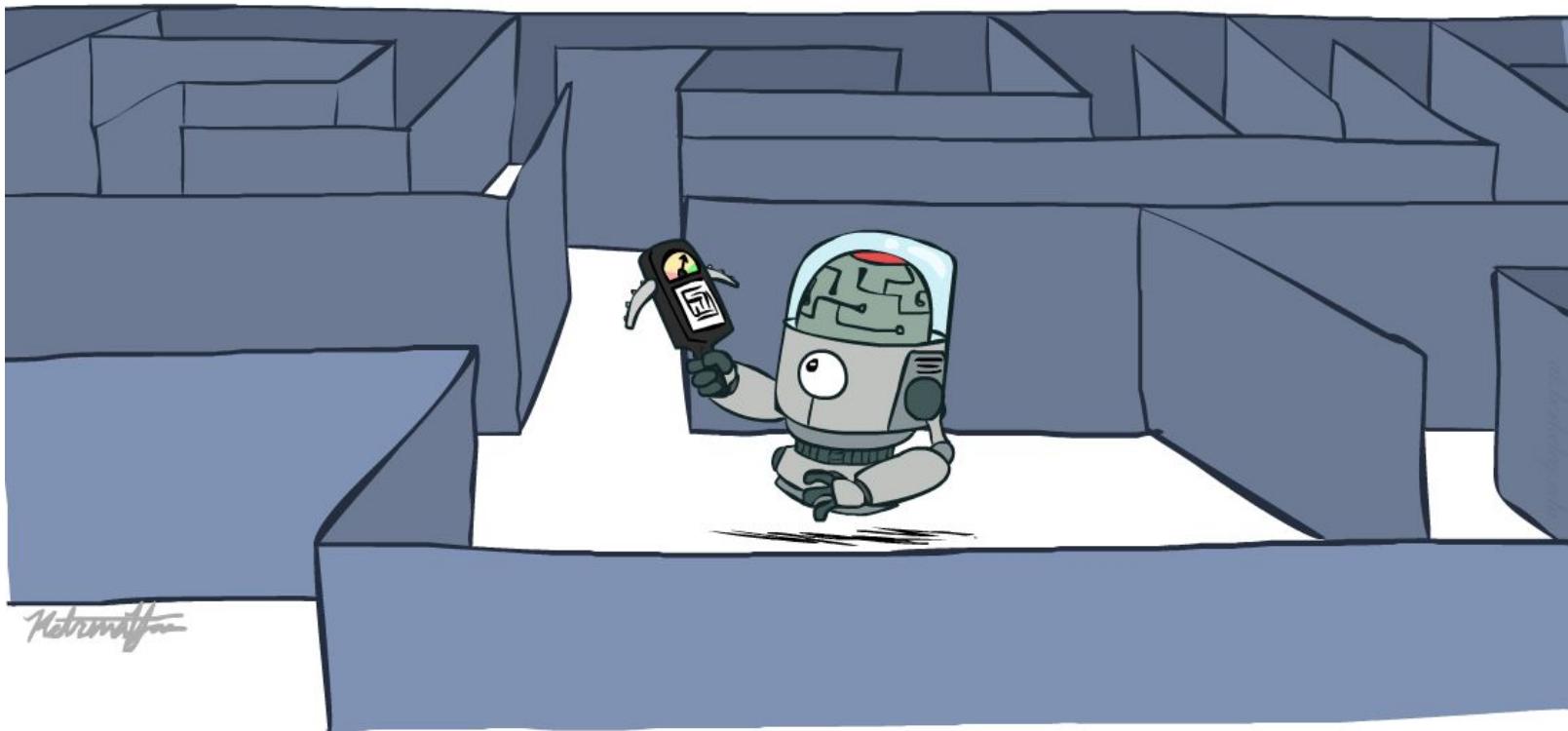
UCS Properties

- Complete
- Optimal
- Complexity
 - Time
 - Space

C^* : cost of optimal solution

epsilon: smallest path cost in search graph

Informed Search

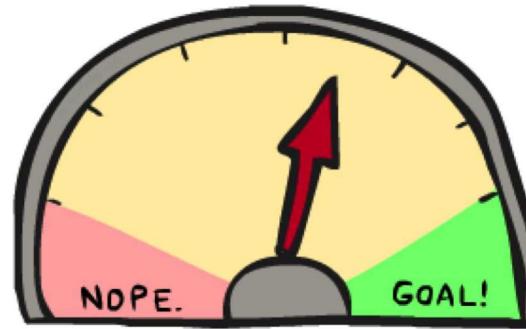


Informed Search

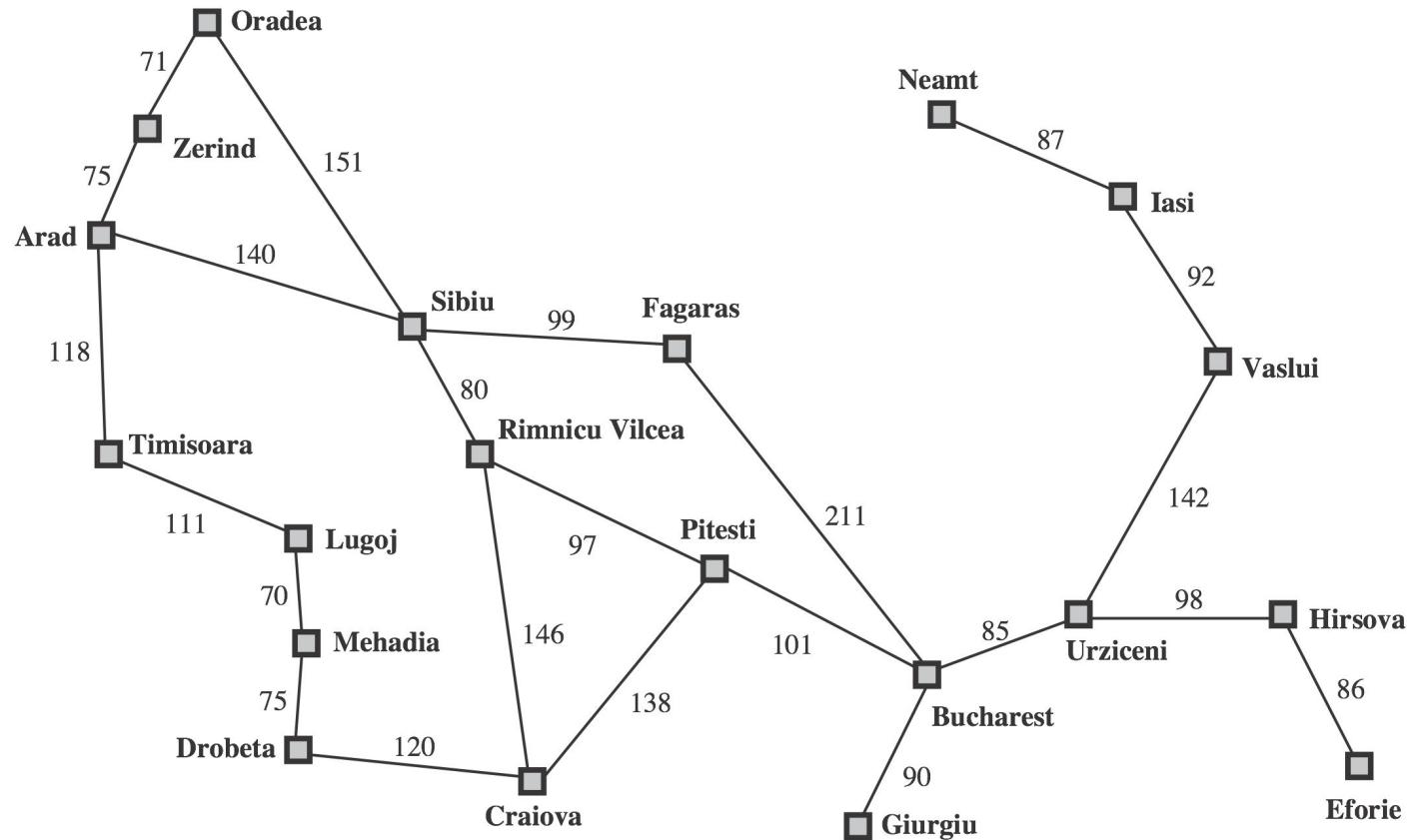
- Informed search strategies find solutions more efficiently than uninformed search strategies
- They employ problem specific knowledge beyond the definition of the problem itself
 - Heuristic function
- Examples
 - Greedy best-first search
 - A* search

Heuristic Function

- A function that estimates how close you are to the goal
- Designed for a particular search problem
- $h(n)$
 - Cost (estimate) of the cheapest path from the state at node n to a goal state
 - If n is a goal node $h(n) = 0$



Romania Problem



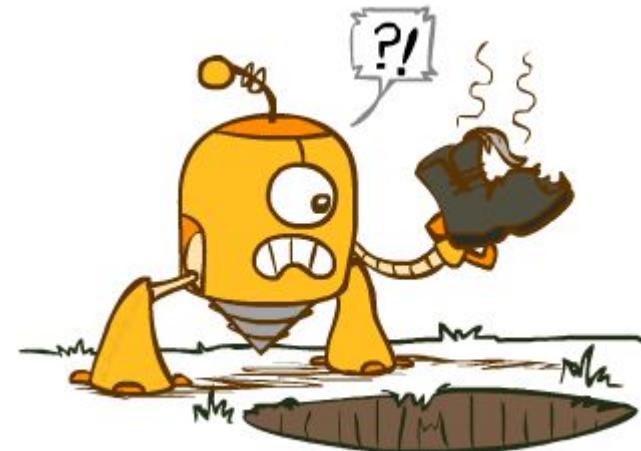
Greedy

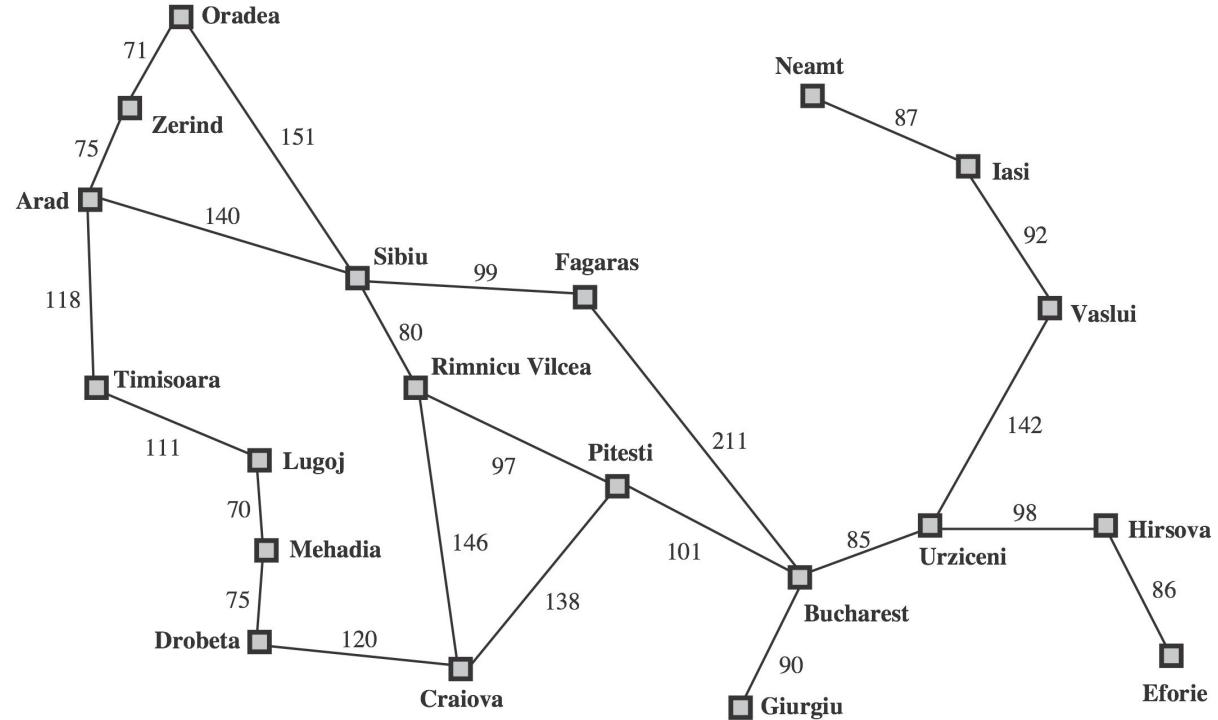


frontier is a priority queue (queue data structure where each element has a priority)

Greedy Search

- Also known as Best-first Search
- Expand the node that has the lowest $h(n)$
 - What can possibly go wrong with this approach?





```
romaniaH = {
    'A':366, 'B':0, 'C':160, 'D':242, 'E':161, 'F':176, 'G':77, 'H':151, 'I':226,
    'L':244, 'M':241, 'N':234, 'O':380, 'P':100, 'R':193, 'S':253, 'T':329, 'U':80,
    'V':199, 'Z':374}
```

Straight-line distance to B

n | h(n)

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374

```
Initial frontier: [(366, 'A')]
```

```
Exploring: A at cost 366
```

```
[(253, 'AS'), (329, 'AT'), (374, 'AZ')]
{'A'}
```

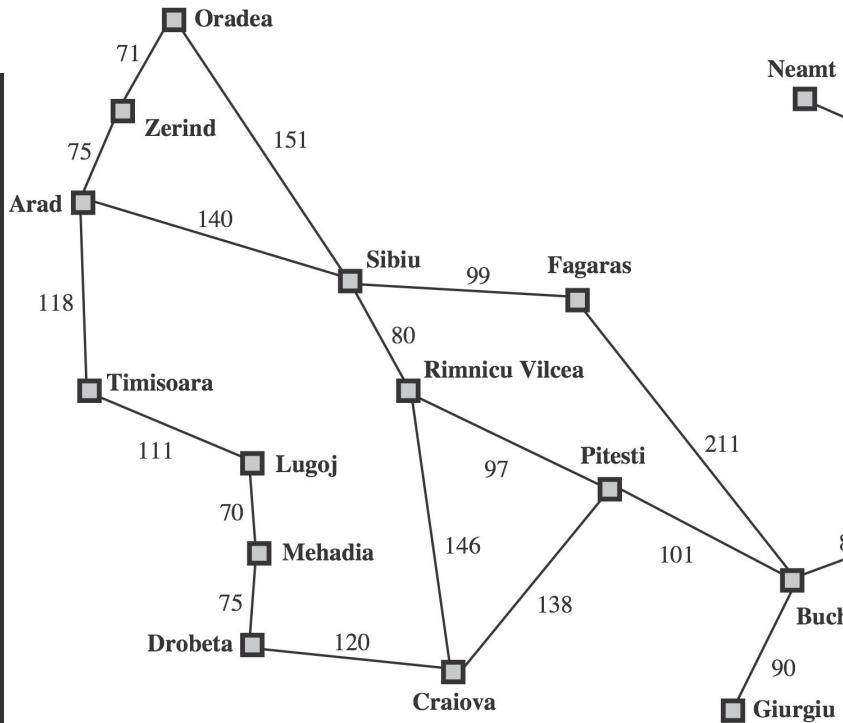
```
Exploring: S at cost 253
```

```
[(176, 'ASF'), (329, 'AT'), (193, 'ASR'),
(374, 'AZ'), (380, 'ASO'), (366, 'ASA')]
{'A', 'S'}
```

```
Exploring: F at cost 176
```

```
[(0, 'ASFB'), (329, 'AT'), (193, 'ASR'),
(374, 'AZ'), (380, 'ASO'), (366, 'ASA'),
(253, 'ASFS')]
{'F', 'A', 'S'}
```

```
Solution path: (0, 'ASFB')
```



Straight-line
distance to B

n	h(n)
---	------

A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374

Try it [here](#)

[greedyTsaRomania.py](#)

[greedyGsaRomania.py](#)

```

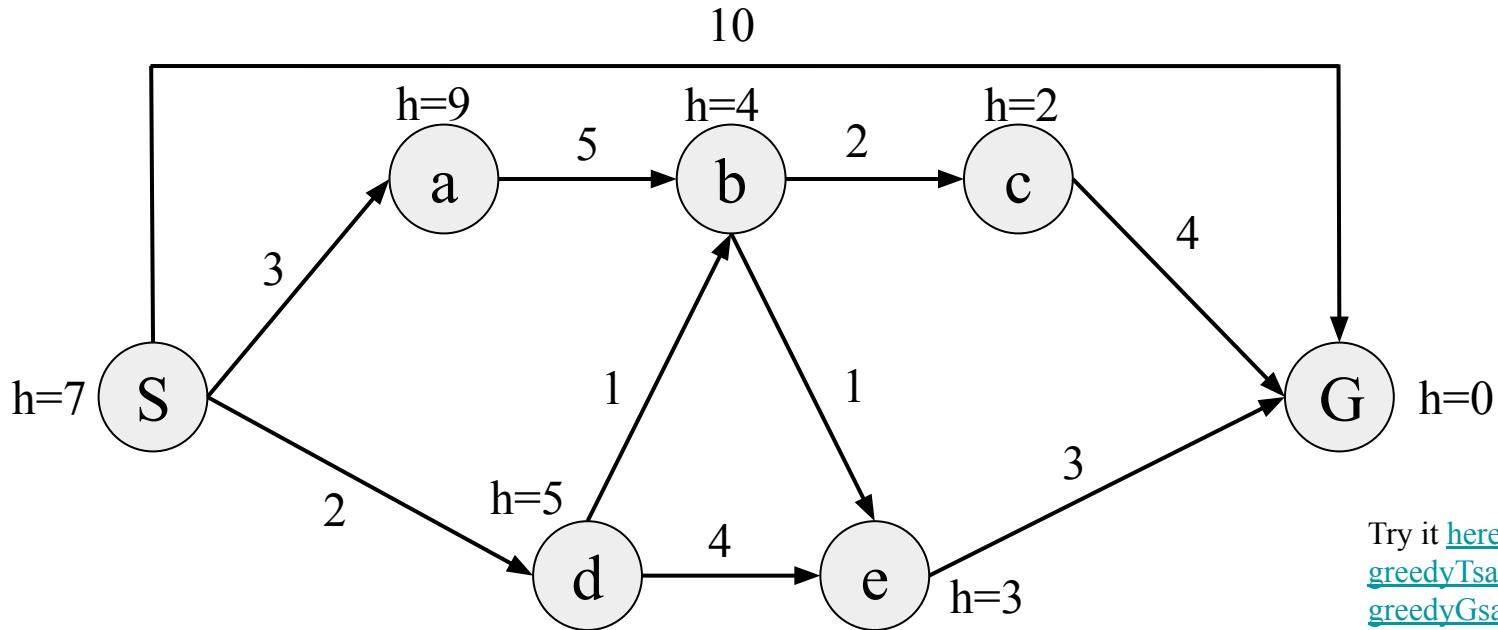
from heapq import heappush, heappop
def greedyTsa(stateSpaceGraph, h, startState, goalState):
    frontier = []
    heappush(frontier, (h[startState], startState))
    print('Initial frontier:',list(frontier)); input()
    while frontier:
        node = heappop(frontier)
        if (node[1].endswith(goalState)): return node
        print('Exploring:',node[1][-1],'at cost',node[0])
        for child in stateSpaceGraph[node[1][-1]]:
            heappush(frontier, (h[child[1]], node[1]+child[1]))
        print(list(frontier)); input()
romania = {
    'A':[(140,'S'),(118,'T'),(75,'Z')], 'Z':[(75,'A'),(71,'O')], 'O':[(151,'S'),(71,'Z')],
    'T':[(118,'A'),(111,'L')], 'L':[(70,'M'),(111,'T')], 'M':[(75,'D'),(70,'L')],
    'D':[(120,'C'),(75,'M')], 'S':[(140,'A'),(99,'F'),(151,'O'),(80,'R')], 
    'R':[(146,'C'),(97,'P'),(80,'S')], 'C':[(120,'D'),(138,'P'),(146,'R')], 
    'F':[(211,'B'),(99,'S')], 'P':[(101,'B'),(138,'C'),(97,'R')], 'B':[]
}
romaniaH = {
    'A':366, 'B':0, 'C':160, 'D':242, 'E':161, 'F':176, 'G':77, 'H':151, 'I':226,
    'L':244, 'M':241, 'N':234, 'O':380, 'P':100, 'R':193, 'S':253, 'T':329, 'U':80,
    'V':199, 'Z':374}
print('Solution path:',greedyTsa(romania, romaniaH, 'A', 'B'))

```

Greedy-TSA Practice

- Consider the following state space graph
 - Let S be the start state and G be the goal state
- Perform **Greedy-TSA** and write down the order of explored states

```
Initial frontier: [(7, 'S')]
Exploring: S at cost 7
[(0, 'SG'), (9, 'Sa'), (5, 'Sd')]
Solution path: (0, 'SG')
```



Try it [here](#)
[greedyTsaPractice.py](#)
[greedyGsaPractice.py](#)

```
from heapq import heappush, heappop
def greedyTsa(stateSpaceGraph, h, startState, goalState):
    frontier = []
    heappush(frontier, (h[startState], startState))
    print('Initial frontier:',list(frontier)); input()
    while frontier:
        node = heappop(frontier)
        if (node[1].endswith(goalState)): return node
        print('Exploring:',node[1][-1],'at cost',node[0])
        for child in stateSpaceGraph[node[1][-1]]:
            heappush(frontier, (h[child], node[1]+child))
        print(list(frontier)); input()
practice = {
    'S':[(3, 'a'),(2, 'd'),(10, 'G')], 'a':[(5, 'b')],
    'd':[(1, 'b'),(4, 'e')], 'G':[], 'b':[(1, 'e'),(2, 'c')],
    'e':[(3, 'G')], 'c':[(4, 'G')]}
practiceH = {'S':7, 'a':9, 'b':4, 'c':2, 'd':5, 'e':3, 'G':0}
print('Solution path:',greedyTsa(practice, practiceH, 'S', 'G'))
```

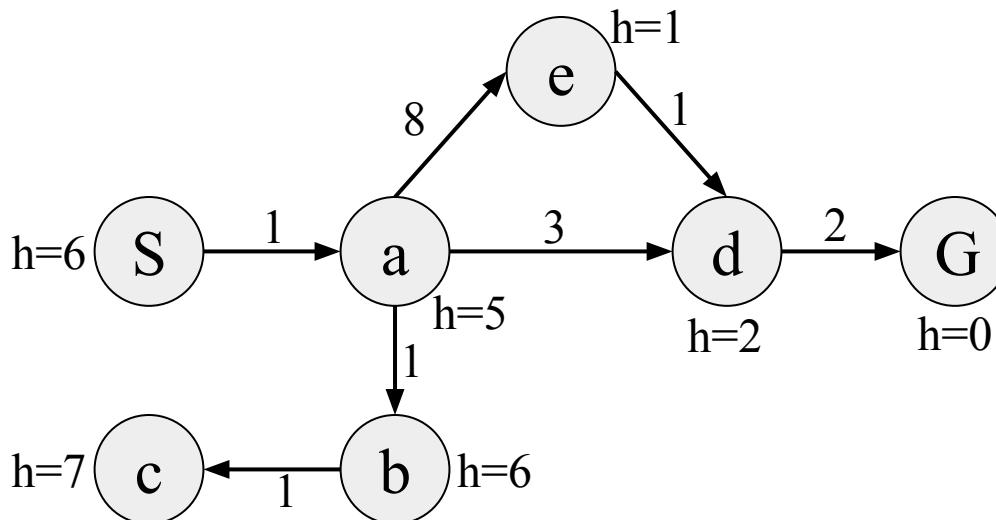
A*



frontier is a priority queue (queue data structure where each element has a priority)

A* Motivation UCS-TSA

- UCS orders by **backward cost**
 - $g(n)$



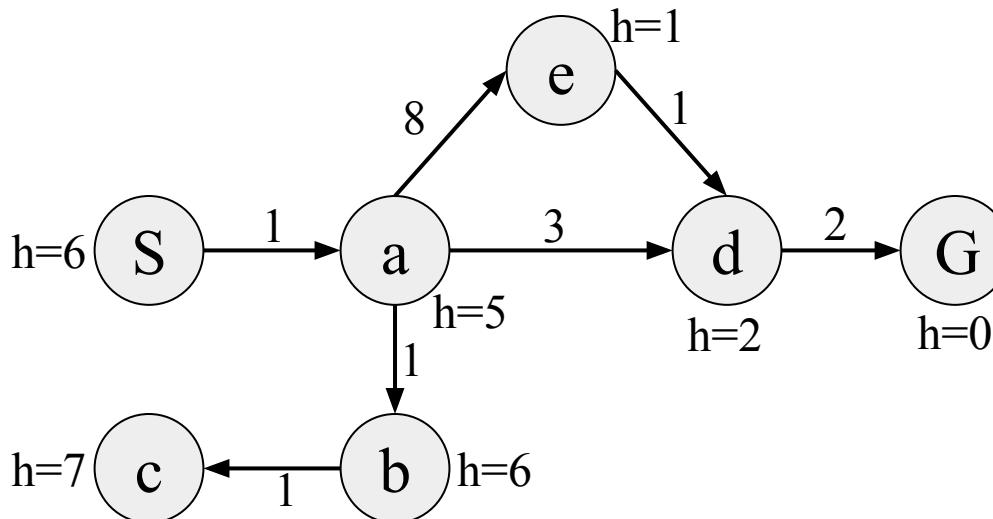
```

Initial frontier: [(0, 'S')]
Exploring: S at cost 0
[(1, 'Sa')]
Exploring: a at cost 1
[(2, 'Sab'), (4, 'Sad'), (9, 'Sae')]
Exploring: b at cost 2
[(3, 'Sabc'), (9, 'Sae'), (4, 'Sad')]
Exploring: c at cost 3
[(4, 'Sad'), (9, 'Sae')]
Exploring: d at cost 4
[(6, 'SadG'), (9, 'Sae')]
Solution path: (6, 'SadG')

```

A* Motivation Greedy-TSA

- Greedy orders by **forward cost**
 - $h(n)$



```

Initial frontier: [(6, 'S')]
Exploring: S at cost 6
[(5, 'Sa')]
Exploring: a at cost 5
[(1, 'Sae'), (6, 'Sab'), (2, 'Sad')]
Exploring: e at cost 1
[(2, 'Sad'), (6, 'Sab'), (2, 'Saed')]
Exploring: d at cost 2
[(0, 'SadG'), (6, 'Sab'), (2, 'Saed')]
Solution path: (0, 'SadG')
  
```

The Tortoise and the Hare



<http://ai.berkeley.edu>

UCS



<http://ai.berkeley.edu>

Greedy

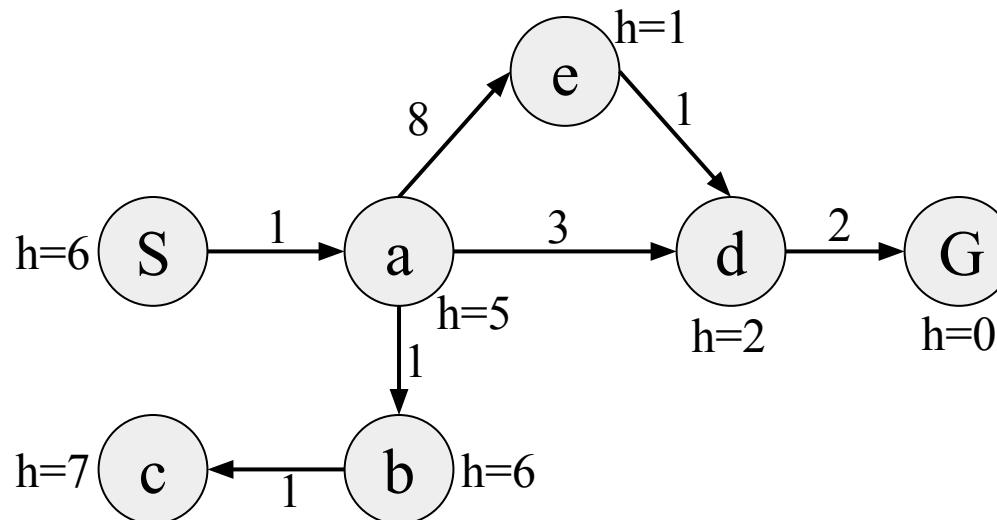


<http://ai.berkeley.edu>

A*

A* Motivation A*-TSA

- A* orders by **backward cost + forward cost**
 - $f(n) = g(n) + h(n)$



```

Exploring: S at cost 6
[(6, 'Sa')]
Exploring: a at cost 6
[(6, 'Sad'), (8, 'Sab'), (10, 'Sae')]
Exploring: d at cost 6
[(6, 'SadG'), (10, 'Sae'), (8, 'Sab')]
Solution path: (6, 'SadG')
  
```

Try it [here](#)
[aStarTsaAStarMotivation.py](#)

```

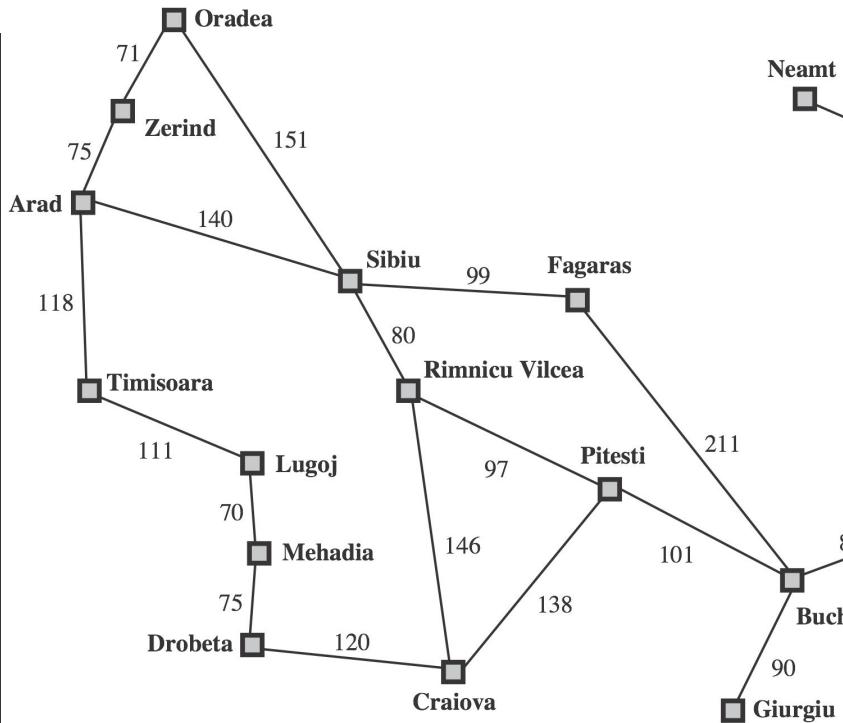
from heapq import heappush, heappop
def aStarTsa(stateSpaceGraph, h, startState, goalState):
    frontier = []
    heappush(frontier, (h[startState], startState))
    print('Initial frontier:',list(frontier)); input()
    while frontier:
        node = heappop(frontier)
        if (node[1].endswith(goalState)): return node
        print('Exploring:',node[1][-1],'at cost',node[0])
        for child in stateSpaceGraph[node[1][-1]]:
            heappush(frontier, (node[0]+child[0]-h[node[1][-1]]+h[child[1]], node[1]+child[1]))
        print(list(frontier)); input()
aStarMotivation = {
'S':[(1,'a')], 'a':[(1,'b'),(3,'d'),(8,'e')], 'b':[(1,'c')], 'c':[], 'd':[(2,'G')], 'e':[(1,'d')]}
aStarMotivationH = { 'S':6, 'a':5, 'b':6, 'c':7, 'd':2, 'e':1, 'G':0}
print('Solution path:',aStarTsa(aStarMotivation, aStarMotivationH, 'S', 'G'))

```

```

Initial frontier: [(366, 'A')]
Exploring: A at cost 366
[(393, 'AS'), (447, 'AT'), (449, 'AZ')]
Exploring: S at cost 393
[(413, 'ASR'), (447, 'AT'), (415, 'ASF'),
(449, 'AZ'), (671, 'ASO'), (646, 'ASA')]
Exploring: R at cost 413
[(415, 'ASF'), (447, 'AT'), (417, 'ASRP'),
(449, 'AZ'), (671, 'ASO'), (646, 'ASA'),
(526, 'ASRC'), (553, 'ASRS')]
Exploring: F at cost 415
[(417, 'ASRP'), (447, 'AT'), (526, 'ASRC'),
(449, 'AZ'), (671, 'ASO'), (646, 'ASA'),
(553, 'ASRS'), (450, 'ASFB'), (591, 'ASFS')]
Exploring: P at cost 417
[(418, 'ASRPB'), (447, 'AT'), (526, 'ASRC'),
(449, 'AZ'), (607, 'ASRPR'), (646, 'ASA'),
(553, 'ASRS'), (591, 'ASFS'), (450, 'ASFB'),
(671, 'ASO'), (615, 'ASRPC')]
Solution path: (418, 'ASRPB')

```



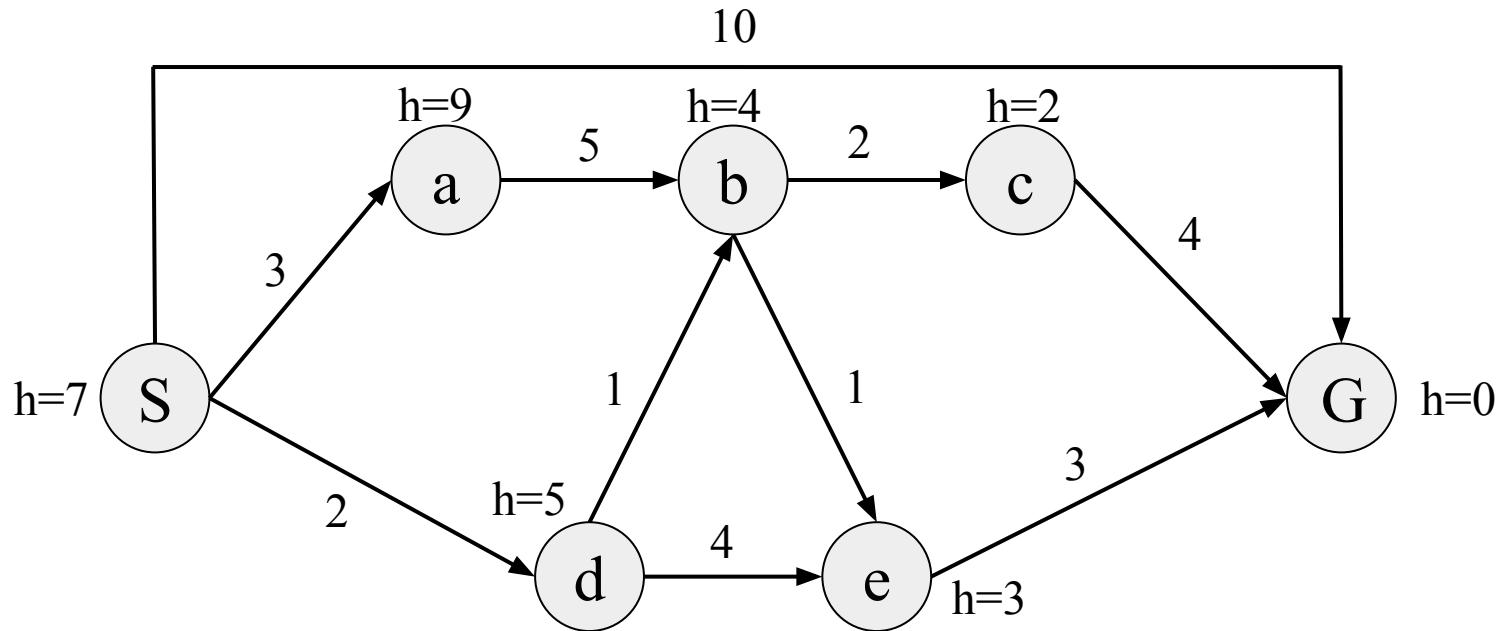
Straight-line
distance to B

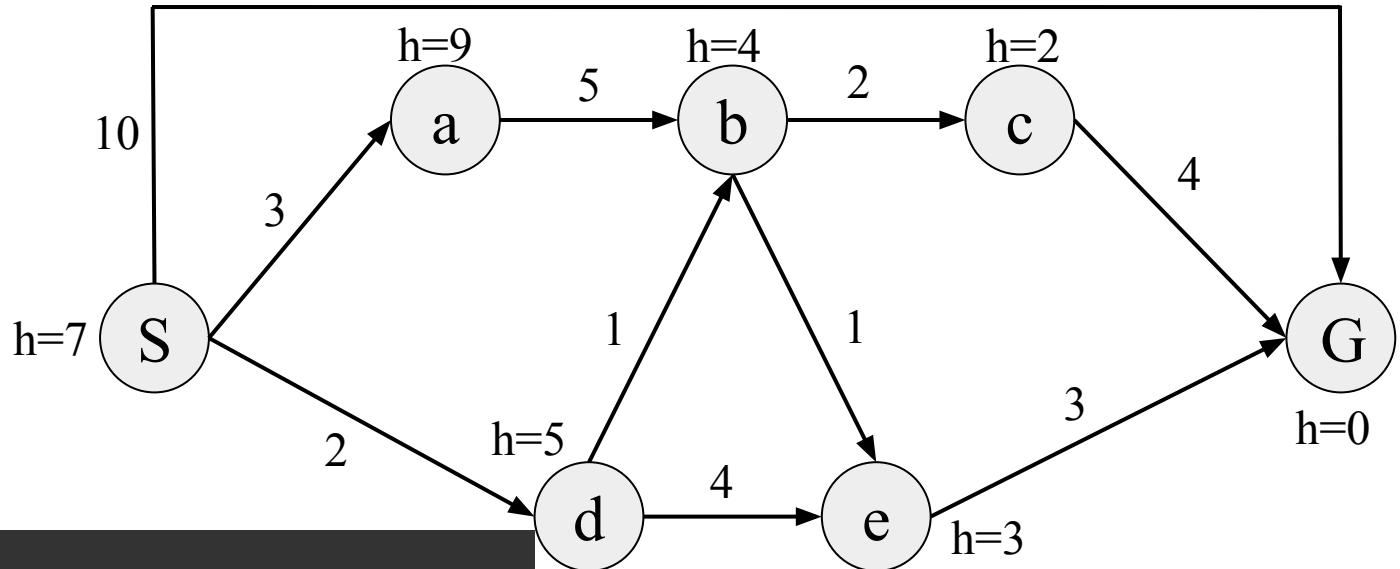
n	h(n)
A	366
B	0
C	160
D	242
E	161
F	176
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	100
R	193
S	253
T	329
U	80
V	199
Z	374

Try it [here](#)
[aStarTsaRomania.py](#)
[aStarGsaRomania.py](#)

A*-TSA Practice

- Consider the following state space graph
 - Let S be the start state and G be the goal state
- Perform **A*-TSA** and write down the order of explored states





```

Initial frontier: [(7, 'S')]
Exploring: S at cost 7
[(7, 'Sd'), (12, 'Sa'), (10, 'SG')]
Exploring: d at cost 7
[(7, 'Sdb'), (9, 'Sde'), (10, 'SG'), (12, 'Sa')]
Exploring: b at cost 7
[(7, 'Sdbc'), (7, 'Sdbe'), (10, 'SG'), (12, 'Sa'), (9, 'Sde')]
Exploring: c at cost 7
[(7, 'Sdbe'), (9, 'SdbcG'), (10, 'SG'), (12, 'Sa'), (9, 'Sde')]
Exploring: e at cost 7
[(7, 'SdbeG'), (9, 'SdbcG'), (10, 'SG'), (12, 'Sa'),
(9, 'Sde')]
Solution path: (7, 'SdbeG')

```

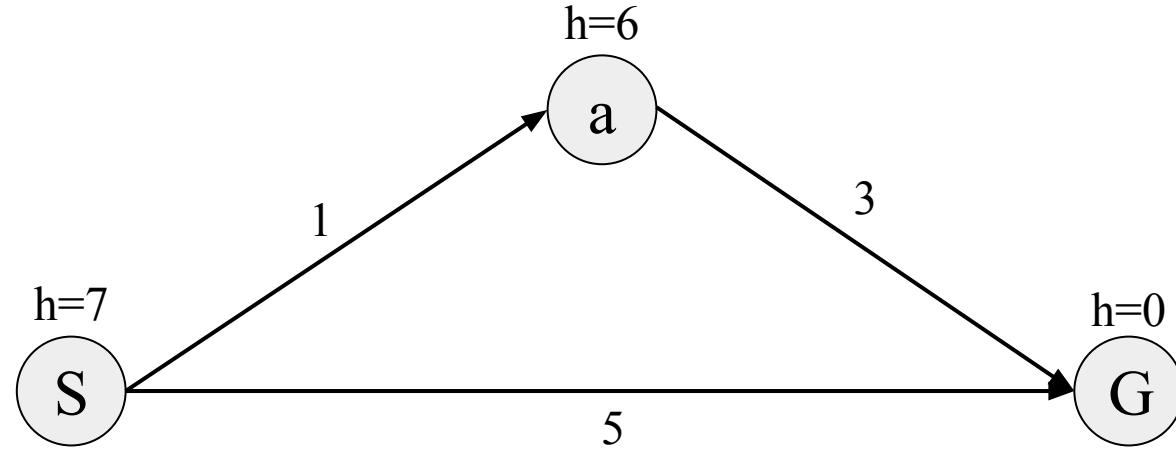
Try it [here](#)
[aStarTsaPractice.py](#)
[aStarGsaPractice.py](#)

Optimality of A*



<http://ai.berkeley.edu>

Is A* Optimal ?



```
Initial frontier: [(7, 'S')]  
Exploring: S at cost 7  
[(5, 'SG'), (7, 'Sa')]  
Solution path: (5, 'SG')
```

What went wrong?

Try it [here](#)
[aStarTsaInadmissible.py](#)
[aStarGsaInadmissible.py](#)

Admissibility of Heuristic

A heuristic $h(n)$ is admissible (optimistic)

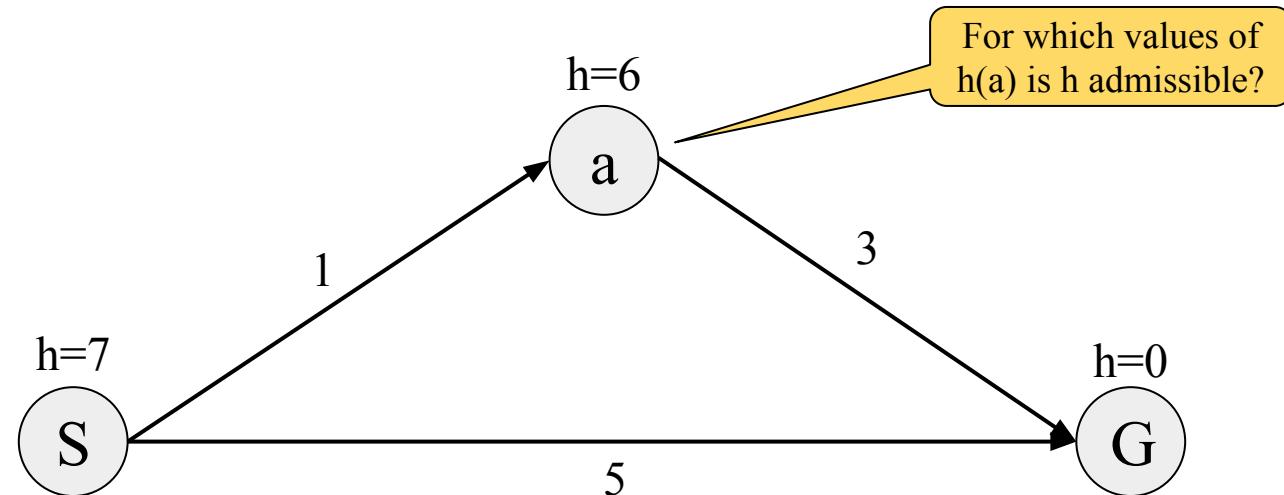
$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to the nearest goal

Admissibility of Heuristic

A heuristic $h(n)$ is admissible (optimistic) if
 $0 \leq h(n) \leq h^*(n)$,

where $h^*(n)$ is the true cost to the nearest goal from n

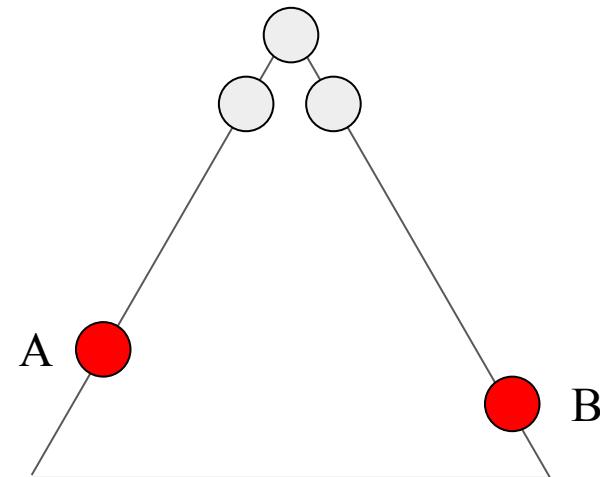


Optimality of A*

- Claim
 - A* is optimal if an admissible heuristic is used

Optimality of A*

- Let
 - A be an optimal goal node
 - B be a suboptimal goal node
 - h be admissible
- Claim
 - A will exit the frontier before B

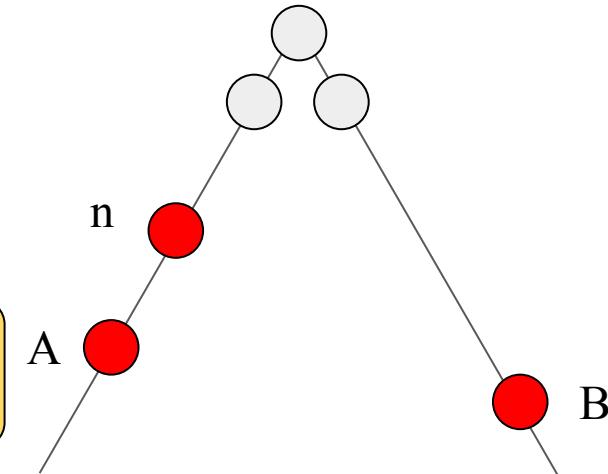


Optimality of A*

- Let n be an ancestor of A
- Let both B and n be on the frontier
- Claim
 - n will exit frontier before B
- Proof
 - $f(n) \leq f(A)$
 - $f(A) < f(B)$
 - n exits frontier before B

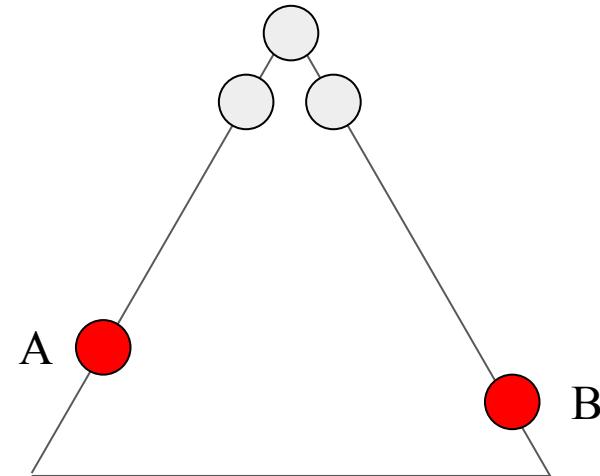
Follows from definition of
admissibility and the fact that n is
an ancestor of A

B is suboptimal

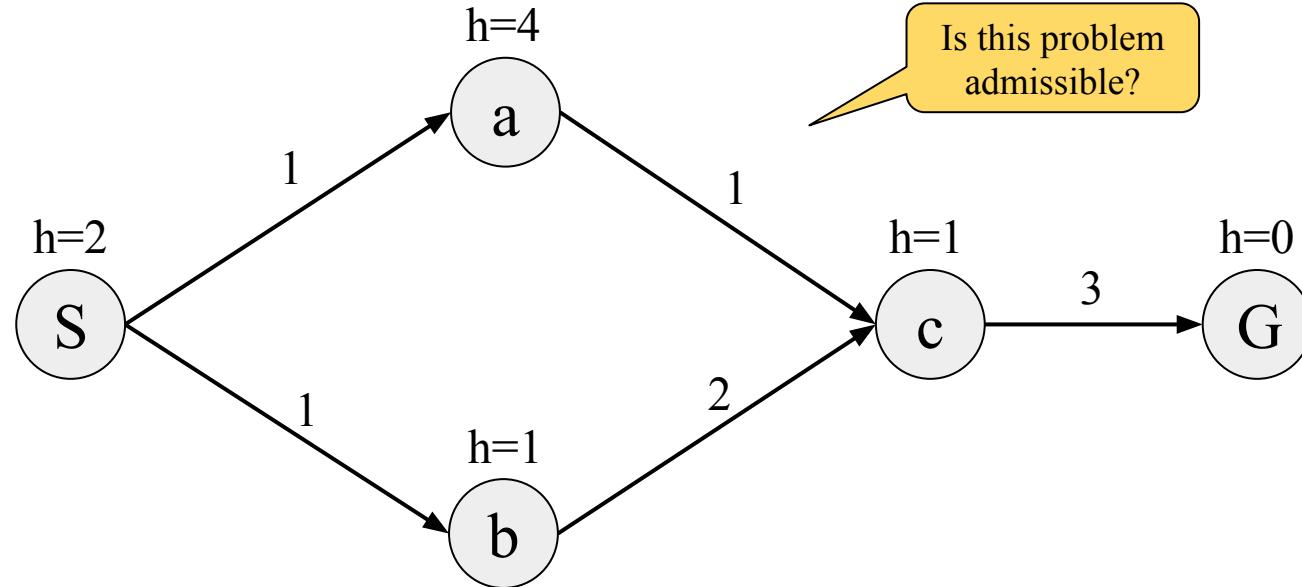


Optimality of A*

- All ancestors of A will exit frontier before B
- A will exit frontier before B
- A* TSA is optimal



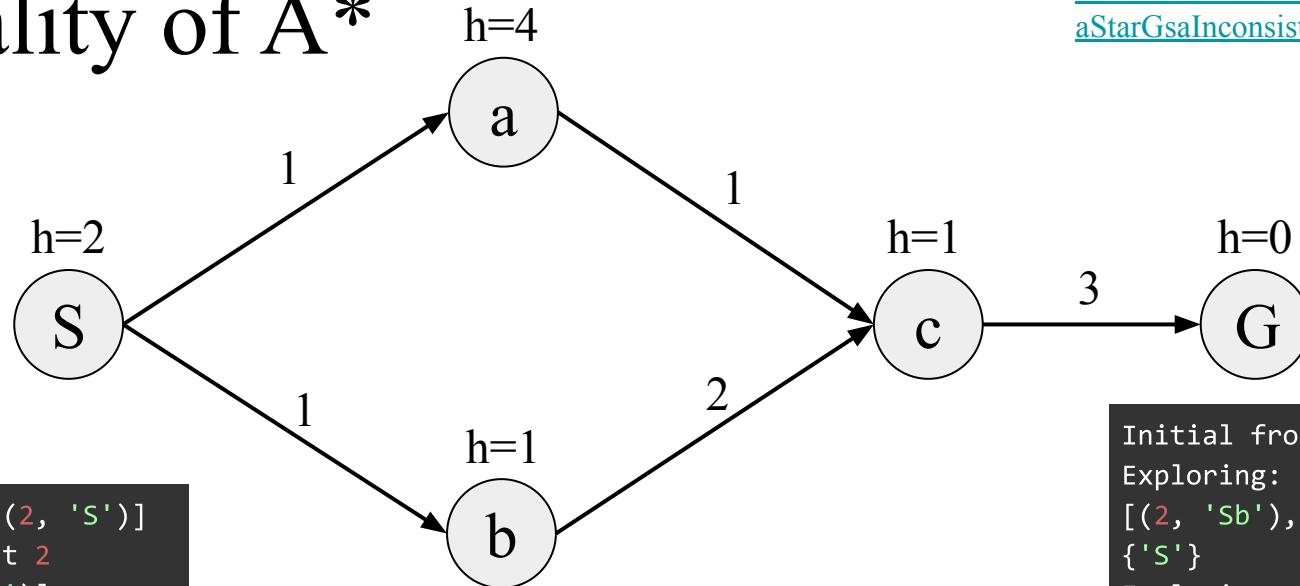
Optimality of A*



A heuristic $h(n)$ is admissible (optimistic) if
 $0 \leq h(n) \leq h^*(n)$,
where $h^*(n)$ is the true cost to the nearest goal from n

Try it [here](#)
[aStarTsaInconsistent.py](#)
[aStarGsaInconsistent.py](#)

Optimality of A*



```

Initial frontier: [(2, 'S')]
Exploring: S at cost 2
[(2, 'Sb'), (5, 'Sa')]
Exploring: b at cost 2
[(4, 'Sbc'), (5, 'Sa')]
Exploring: c at cost 4
[(5, 'Sa'), (6, 'SbcG')]
Exploring: a at cost 5
[(3, 'Sac'), (6, 'SbcG')]
Exploring: c at cost 3
[(5, 'SacG'), (6, 'SbcG')]
Solution path: (5, 'SacG')
  
```

Looks good, this is
the optimal
solution

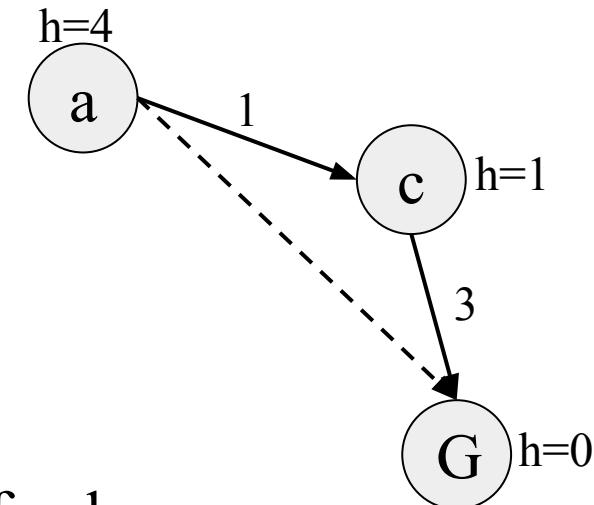
What went
wrong ?

```

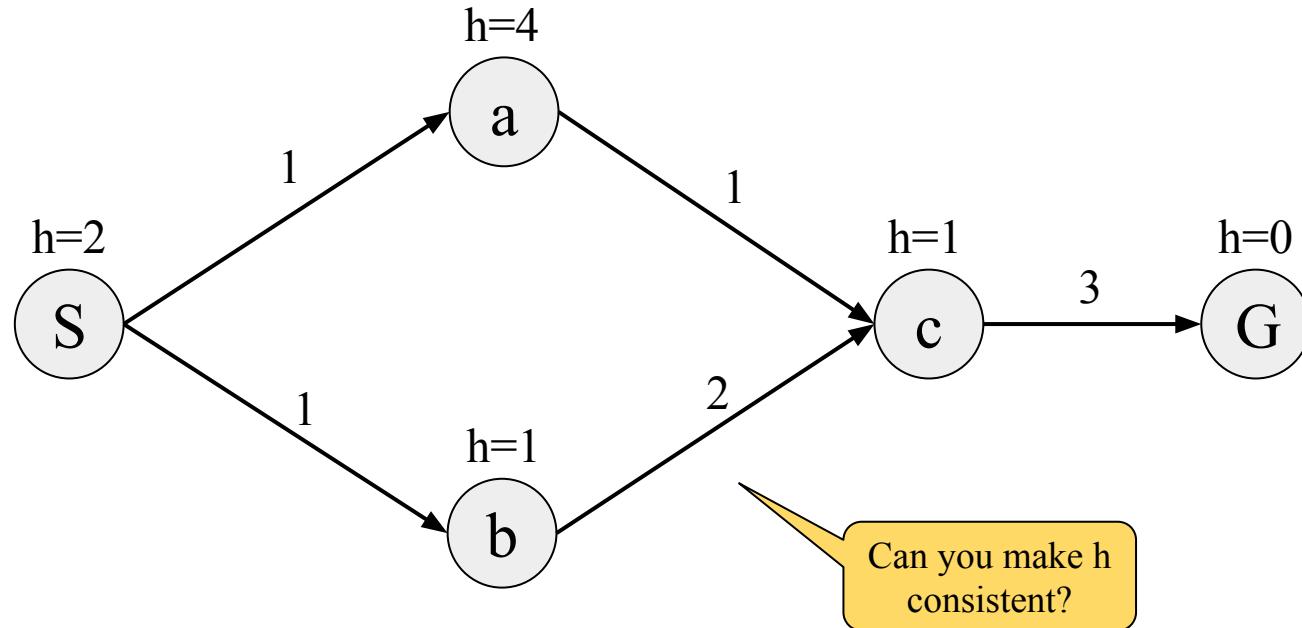
Initial frontier: [(2, 'S')]
Exploring: S at cost 2
[(2, 'Sb'), (5, 'Sa')]
{'S'}
Exploring: b at cost 2
[(4, 'Sbc'), (5, 'Sa')]
{'S', 'b'}
Exploring: c at cost 4
[(5, 'Sa'), (6, 'SbcG')]
{'S', 'c', 'b'}
Exploring: a at cost 5
[(3, 'Sac'), (6, 'SbcG')]
{'S', 'c', 'a', 'b'}
Solution path: (6, 'SbcG')
  
```

Consistency of Heuristic

- Definition
 - Heuristic cost \leq actual cost for each arc
 - $h(a) - h(c) \leq \text{cost}(a \text{ to } c)$
- Consequence of consistency
 - The f value along a path never decreases
 - $h(a) \leq \text{cost}(a \text{ to } c) + h(c)$
- Can you prove that A* GSA is optimal if the f value never decreases?



Consistency of Heuristic



Local Search

Planning vs. Identification

- Planning: sequences of actions
 - The path to the goal is the important thing
 - Paths have various costs, depths
 - Heuristics to guide, frontier to keep backups
- Identification: assignments to variables
 - The goal itself is important, not the path
- Local Search can find solutions faster for specific types of identification problems

Local Search

- Evaluate and modify one current state rather than systematically explore paths from an initial state
- Suitable for problems where all that matters is the solution state and not the path cost to reach it
- Although local search algorithms are not systematic they have two advantages
 - Require very little memory
 - Often find reasonable solutions in large spaces

Example: 8-Queens

- States
 - Each state has 8 queens on the board, one per column
- Successors
 - All possible states generated by moving a single queen to another square in the same column
- Cost function
 - Number of pairs of queens that are attacking each other, either directly or indirectly

Example: 8-Queens

- What is the cost of the following state?

Q ₁			Q ₄				
				Q ₅			
Q ₂					Q ₆		Q ₈
		Q ₃				Q ₇	

- Answer:
- Consider unique attacks
 - Q₁ is attacking Q₂, Q₃, Q₅
 - Q₂ is attacking Q₃, Q₄, Q₆, Q₈
 - Q₃ is attacking Q₅, Q₇
 - Q₄ is attacking Q₅, Q₆, Q₇
 - Q₅ is attacking Q₆, Q₇
 - Q₆ is attacking Q₇, Q₈
 - Q₇ is attacking Q₈
- The cost of the state is 17

Example: 8-Queens

- How many neighbors does the following state have?

Q_1				Q_5			
	Q_2				Q_6		Q_8
		Q_3				Q_7	

- Answer:
- Every queen can move to 7 locations
 - There are $7*8 = 56$ neighbors

Example: 8-Queens

- Costs of all neighbors with minimas shown in red

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	Q_4	13	16	13	16
Q_1	14	17	15	Q_5	14	16	16
17	Q_2	16	18	15	Q_6	15	Q_8
18	14	Q_3	15	15	14	Q_7	16
14	14	13	17	12	14	12	18

Local Search - Algorithm

1. Randomly initialize currentState
2. If cost of currentState == 0 return currentState
3. If $\min(\text{cost}(\text{getNeighbors}(\text{currentState}))) > \text{cost}(\text{currentState})$
 goto step 1 (we have reached a local minimum)
4. Select cheapest neighbor as currentState and goto step 2

Example: 8-Queens

- Starting from a randomly generated 8-queens state, local search gets stuck 86% of the time, solving only 14% of problem instances
- It just takes 4 steps on average when it succeeds and 3 when it gets stuck
 - This is remarkable considering the huge state space

Constraint Satisfaction Problems

Constraint Satisfaction Problems

- CSPs use a factored representation for states
 - A set of variables, each of which has a value
- A problem is solved when each variable has a value that satisfies certain constraints
- CSPs can often be solved more efficiently
 - They eliminate large portions of the search space by identifying variable/value combinations that violate constraints

Defining CSPs

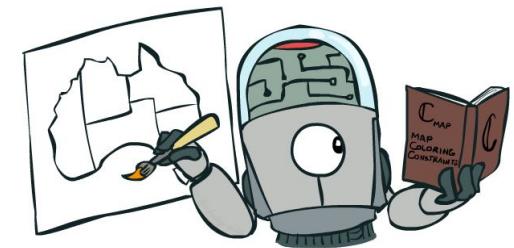
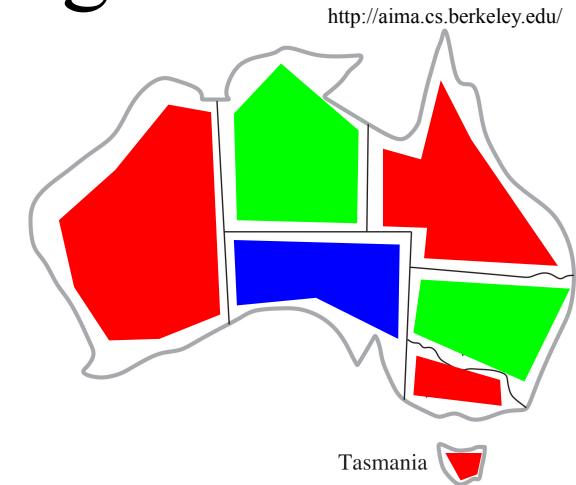
- A CSP consists of three components
 - A set of variables, $X = \{X_1, \dots, X_n\}$
 - A set of domains, $D = \{D_1, \dots, D_n\}$,
where $D_i = \{v_1, \dots, v_k\}$ for each variable X_i
 - A set of constraints C which specify allowable combinations of values
- To solve a CSP we need to define a state space
 - Each state is defined by an assignment of values to some or all variables $\{X_i = v_i, X_j = v_j, \dots\}$

Solutions to CSPs

- If no constraints are violated we call the assignment consistent
- If every variable is assigned we call the assignment complete
- A solution is an assignment that is both consistent and complete

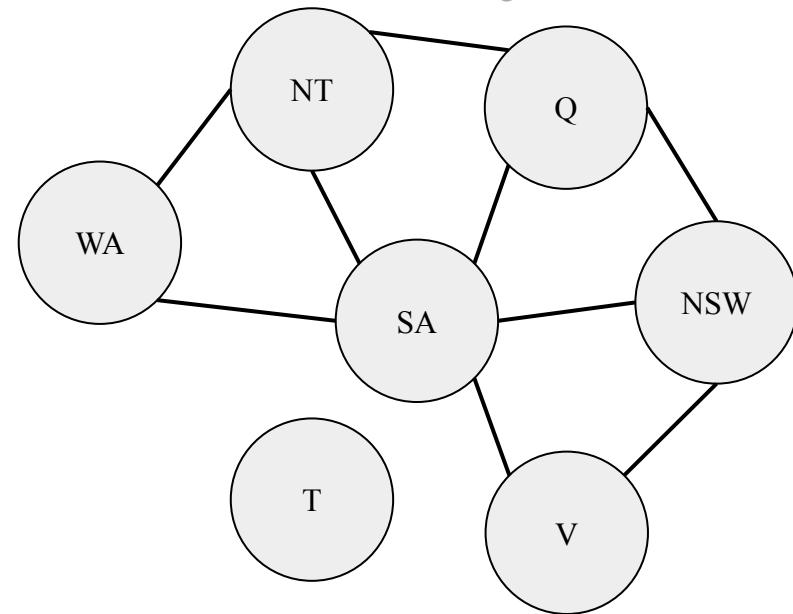
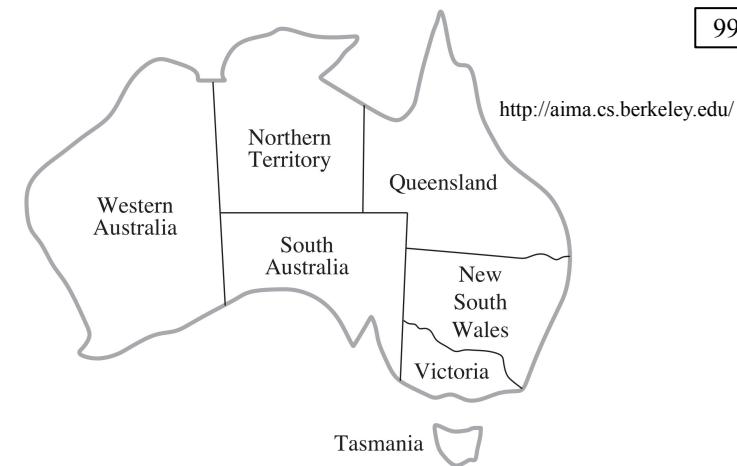
Example: Australia - Map Coloring

- Variables
 - $X = \{\text{WA, NT, Q, NSW, V, SA, T}\}$
- Domains
 - $D = \{\text{red, green, blue}\}$
- Constraints (different colors for neighbors)
 - Implicit
 - $C = \{\text{SA} \neq \text{WA}, \dots\}$
 - Explicit
 - $C = \{(\text{SA, WA}) \in \{(\text{red, green}), \dots\}\}$



Constraint Graph

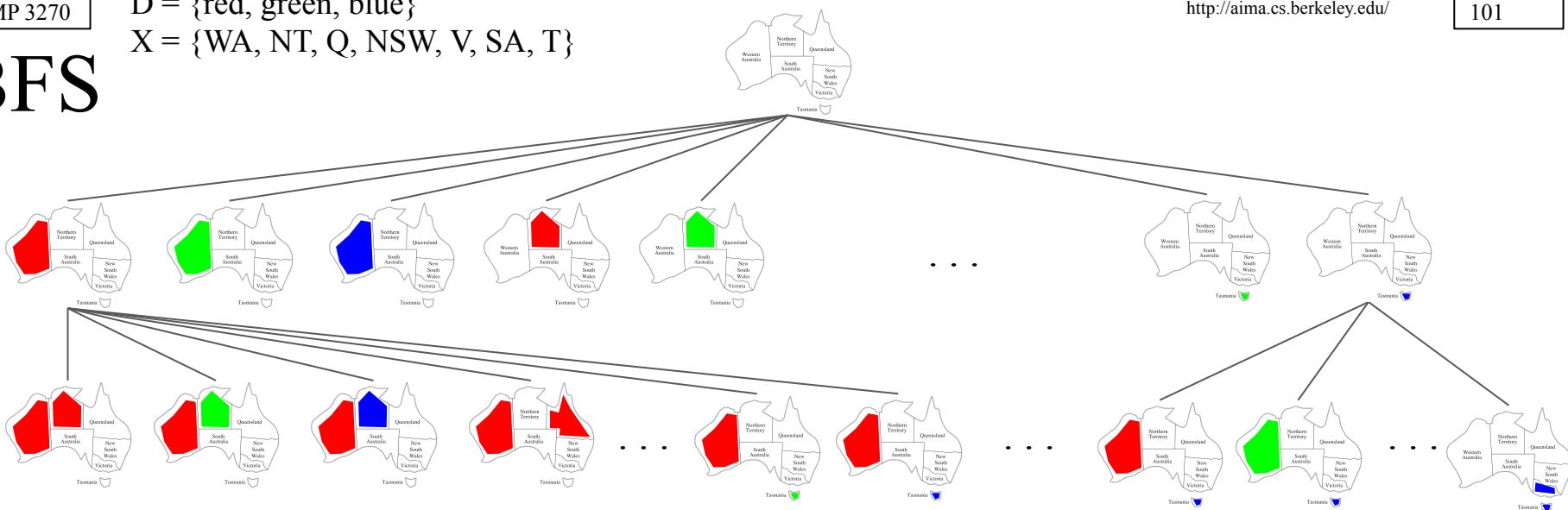
- It can be helpful to visualize a CSP as a constraint graph
 - Nodes correspond to variables
 - Arcs show existence of constraints



Solving CSPs

- Let's start with a naive approach
 - States are defined by the values assigned so far
 - Initial state
 - Empty assignment { }
 - Successor function
 - Assign a value to an unassigned variable
 - Goal test
 - Current assignment is complete and consistent

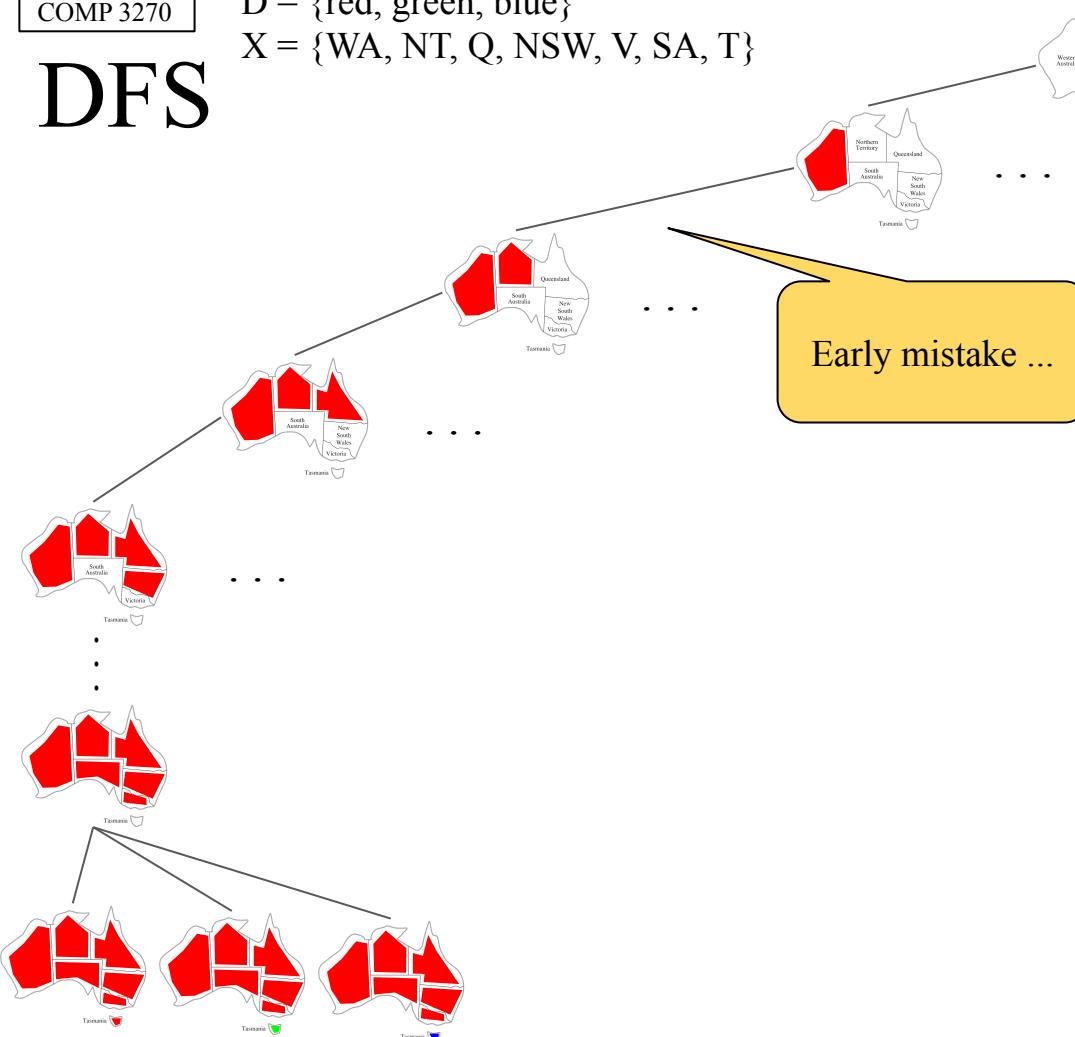
BFS



Note that all solutions will be at the bottom of this search tree :-(

This calls
for DFS
:-)

DFS

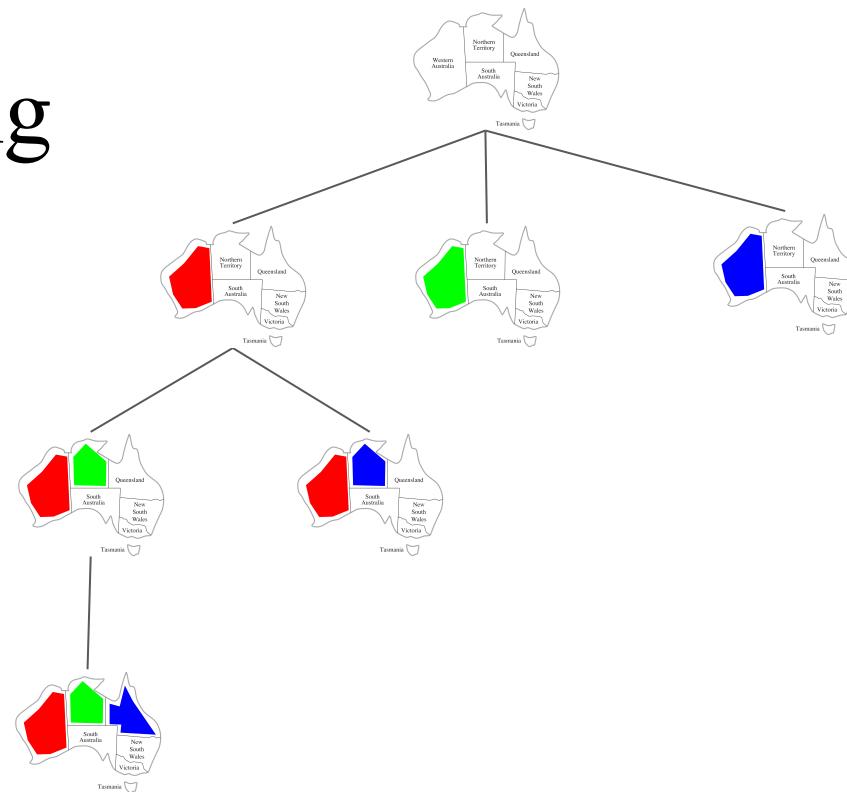


Backtracking Search

- The basic algorithm for solving CSPs
- Idea
 - a. Only consider assignments to a single variable at each point
 - Note that variable assignments are commutative
 - Therefore, we need only consider a single variable at each node
 - b. Only allow legal assignments at each point
 - Consider only values which do not conflict previous assignments
 - Incremental goal test
- DFS with these two ideas is called backtracking search

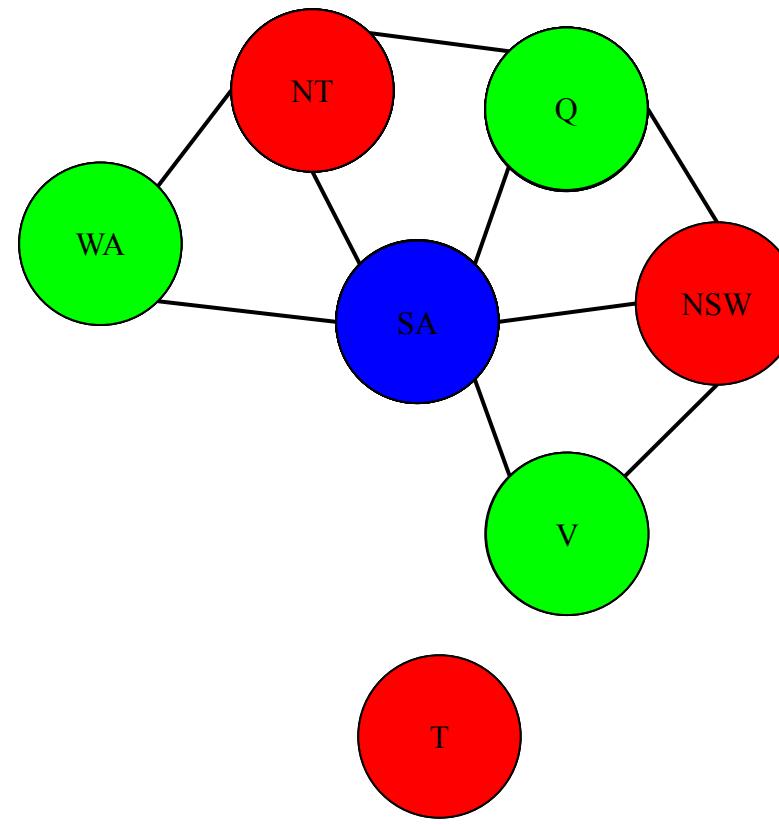


Backtracking



$D = \{\text{red, green, blue}\}$
 $X = \{\text{WA, NT, Q, NSW, V, SA, T}\}$

Example: Backtracking



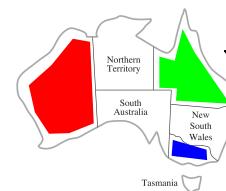
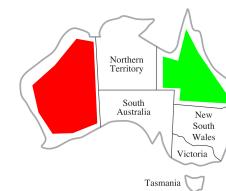
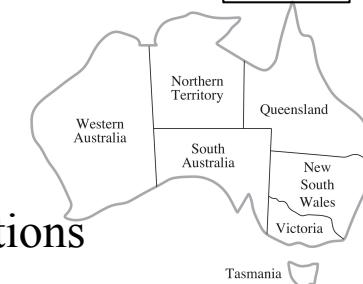
$D = \{\text{red, green, blue}\}$
 $X = \{\text{NSW, WA, NT, Q, SA, V, T}\}$

Improving Backtracking

- Idea
 - Can we detect inevitable failure early?
 - Forward checking (FC)
 - Constraint propagation (AC-3)

Filtering: Forward Checking

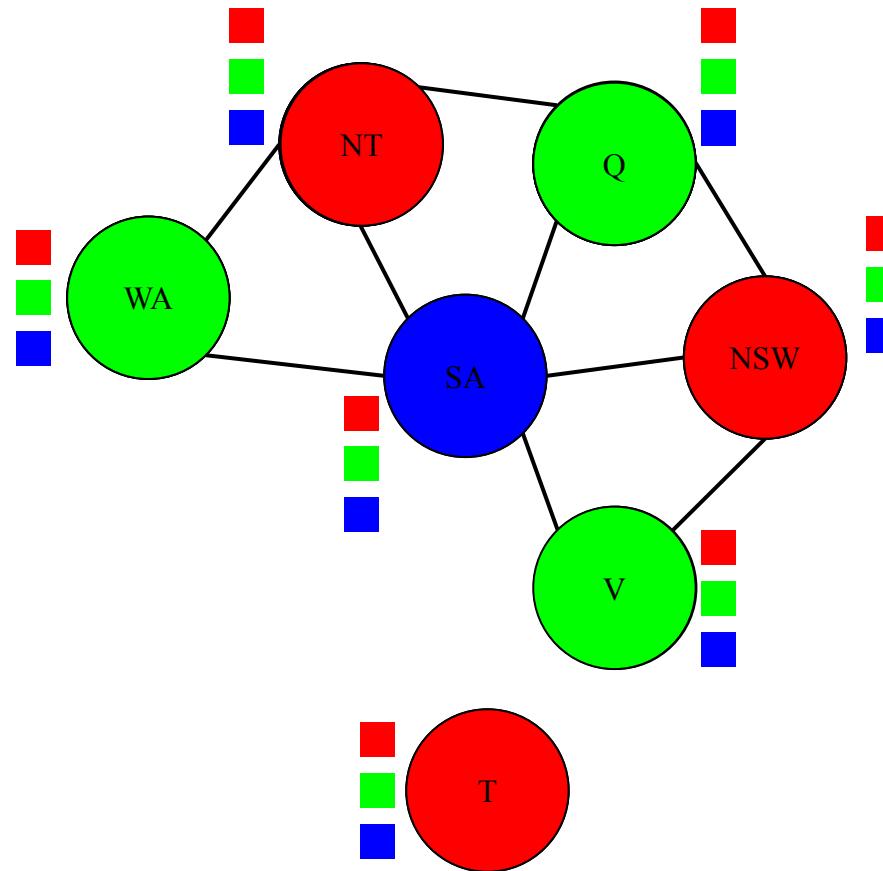
- Filtering
 - Keep track of domains for unassigned variables and cross off bad options
- Forward checking
 - Cross off values that violate a constraint when added to the existing assignment



Detect failure earlier

WA	NT	Q	NSW	V	SA	T
Red Green Blue						
Red	Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Green Blue	Red Green Blue
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue
Red	Blue	Green	Red	Blue		Red Green Blue

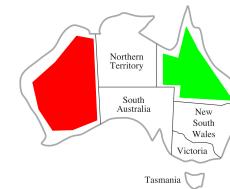
Example: Backtracking + Forward Checking



$D = \{\text{red, green, blue}\}$
 $X = \{\text{NSW, WA, NT, Q, SA, V, T}\}$

Filtering: Forward Checking

- Filtering
 - Keep track of domains for unassigned variables and cross off bad options
- Forward checking
 - Cross off values that violate a constraint when added to the existing assignment



Doomed
to fail

Can we
detect this
earlier?

WA	NT	Q	NSW	V	SA	T
Red Green Blue						
Red	Green Blue	Red Green Blue	Red Green Blue	Red Green Blue	Green Blue	Red Green Blue
Red	Blue	Green	Red Blue	Red Green Blue	Blue	Red Green Blue



Consistency of an Arc

- An arc $X \rightarrow Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint

WA	NT	Q	NSW	V	SA	T
Red	Red Green Blue					

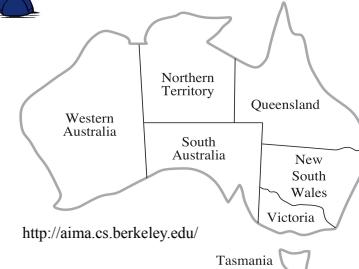
A curved arrow points from the 'WA' column to the 'NT' column.

- Arc consistency explained by Prof. Alan Mackworth

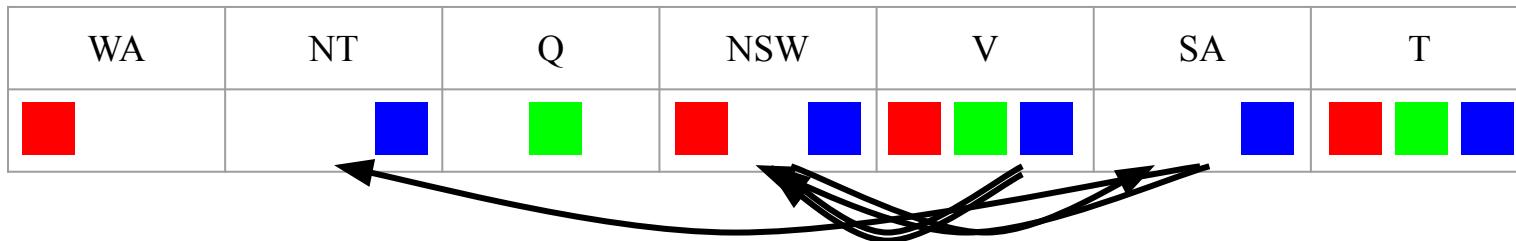
Constraint propagation



Delete from tail!



- Constraint propagation repeatedly enforces constraints
- Note
 - Arcs can become inconsistent
 - If X loses a value, neighbors of X need to be rechecked
 - Arc consistency detects failure earlier than forward checking
 - Can be run as a preprocessor or after each assignment



An arc $X \rightarrow Y$ is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint

AC-3 to Enforce Arc Consistency

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with components $\{X, D, C\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if REVISE(*csp*, X_i , X_j) **then**

if size of $D_i = 0$ **then return** *false*

for each X_k **in** $X_i.\text{NEIGHBORS} - \{X_j\}$ **do**

 add (X_k, X_i) to *queue*

return *true*

Why - { X_j } ?

function REVISE(*csp*, X_i , X_j) **returns** true iff revise domain of X_i

revised \leftarrow *false*

for each x **in** D_i **do**

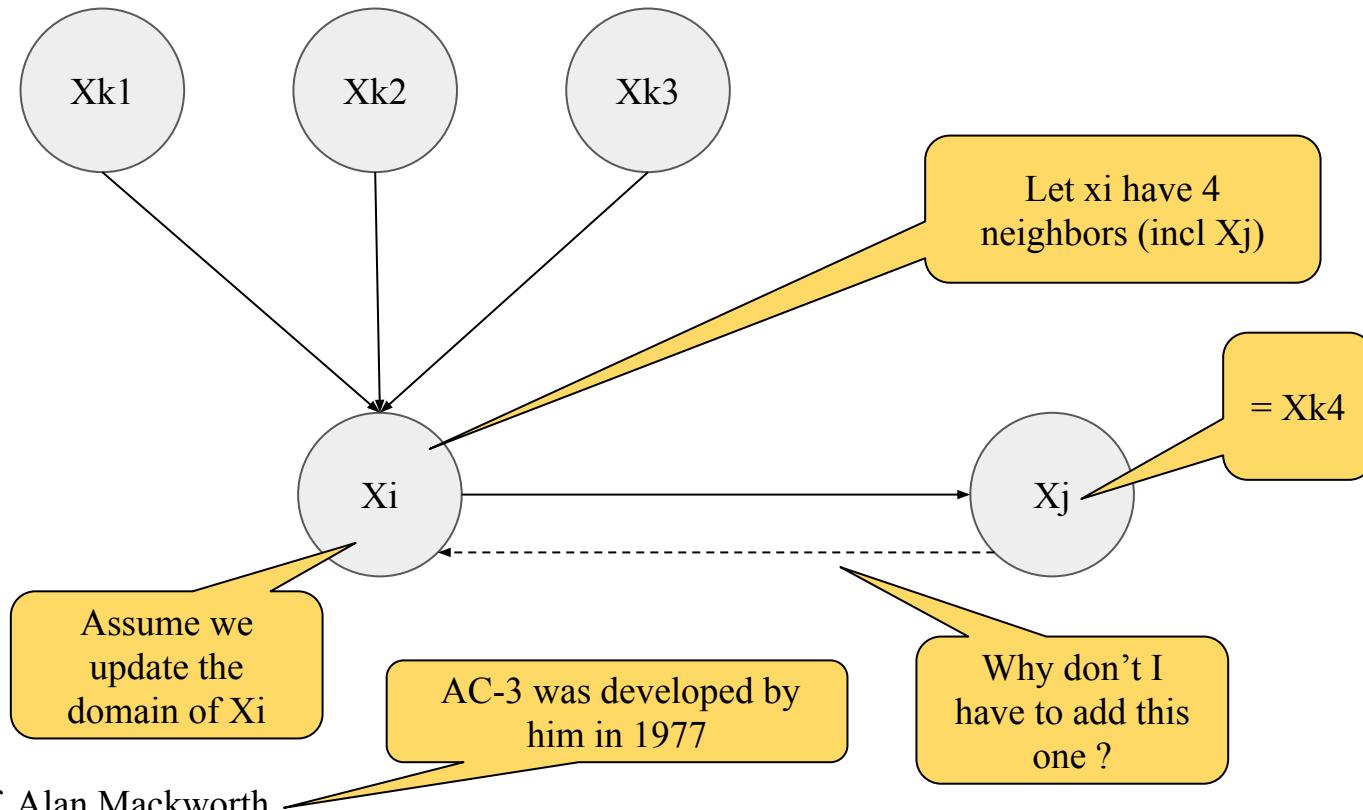
if no value y in D_j allows (x,y) to satisfy constraint between X_i and X_j

 delete x from D_i

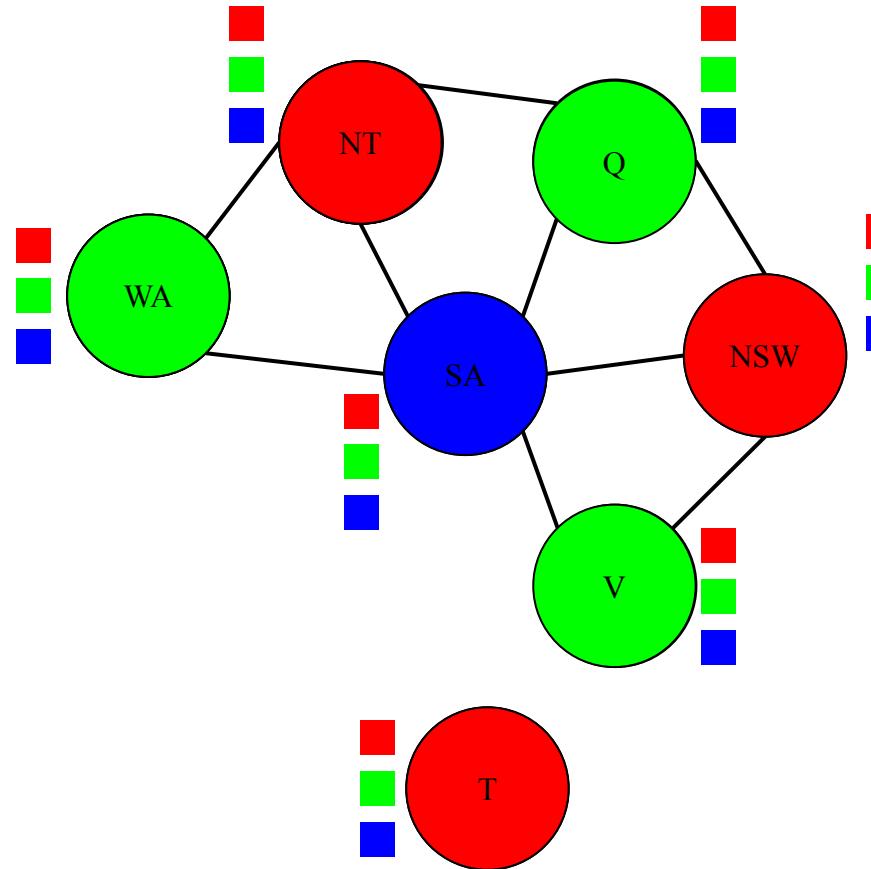
revised \leftarrow *true*

return *revised*

Why - { X_j } ?



Example: Backtracking + AC-3

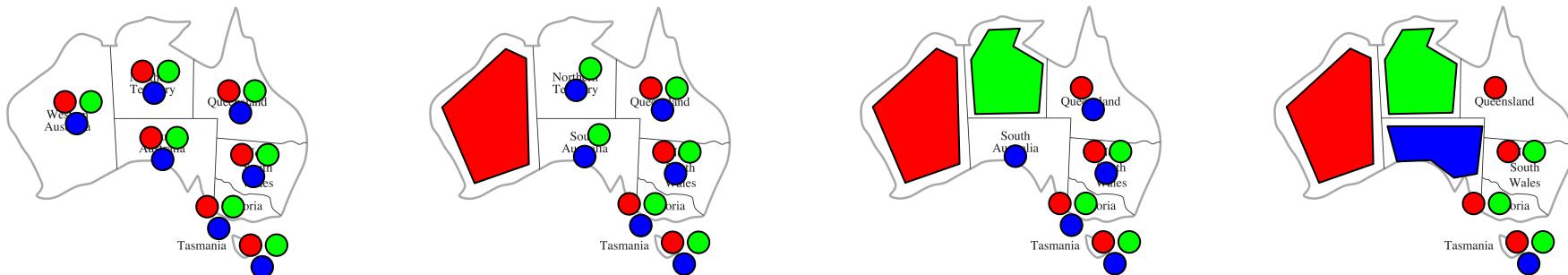


Improving Backtracking Further

- Variable Ordering
 - Minimum remaining values (MRV)
 - Choose the variable with the fewest legal left values in its domain
 - Most constrained variable
 - Fail-first heuristic
 - Tie-breaker among MRV variables
 - Degree Heuristic (Deg)
 - Choose the variable with the most constraints on remaining variables
 - Value Ordering
 - Least constraining value (LCV)
 - Choose the value that rules out the fewest values in the remaining variables

Variable Ordering (MRV)

- Minimum remaining values
 - Choose the variable with the fewest legal left values in its domain
 - Most constrained variable
 - Fail-first heuristic

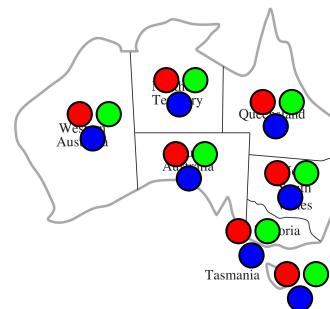


using forward checking

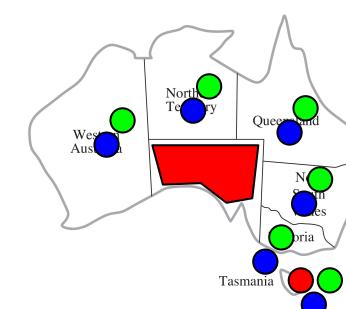
$$\begin{aligned} D &= \{\text{red, green, blue}\} \\ X &= \{\text{WA, NSW, NT, Q, SA, V, T}\} \end{aligned}$$

Variable Ordering (MRV + Deg)

- Degree Heuristic (Deg)
 - Choose the variable with the most constraints on remaining variables



using forward checking

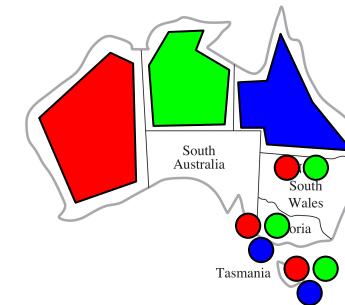
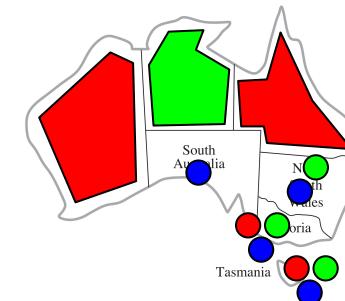
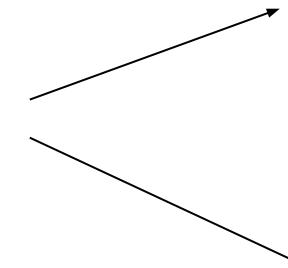
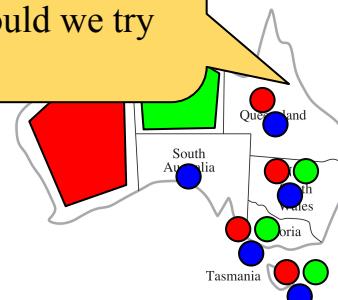


$$\begin{aligned} D &= \{\text{red, green, blue}\} \\ X &= \{\text{NSW, WA, NT, Q, SA, V, T}\} \end{aligned}$$

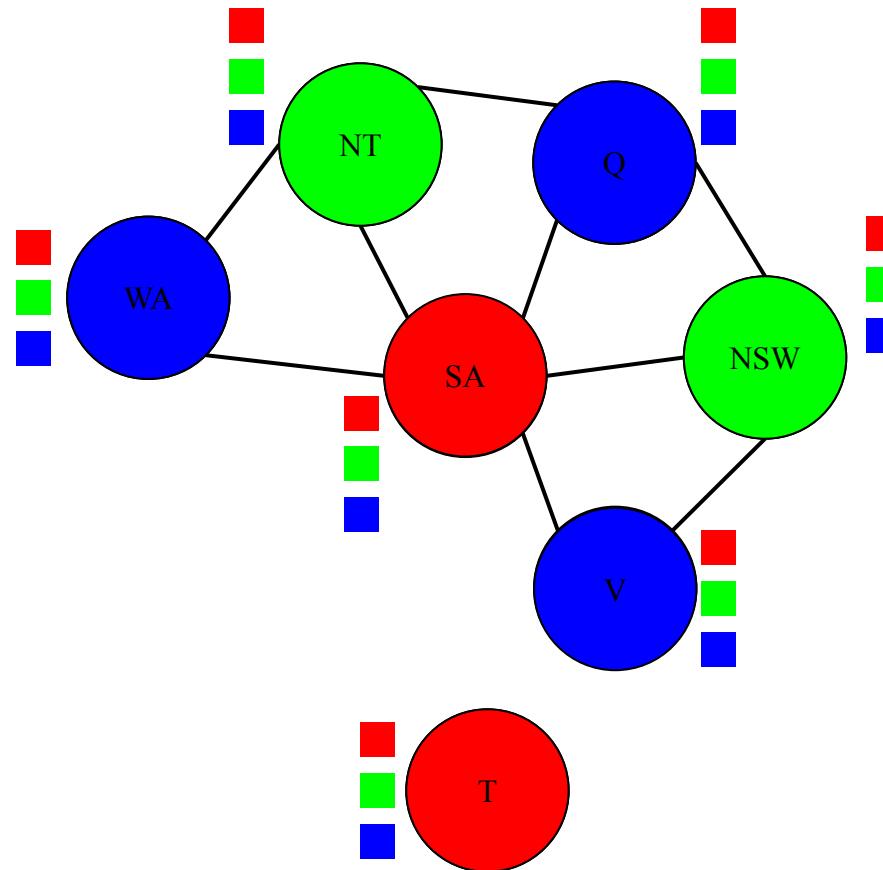
Value Ordering (LCV)

- Choose the value that rules out the fewest values in the remaining variables
 - Least constraining value (LCV)

Suppose we have decided to assign this variable next.
What value should we try first?

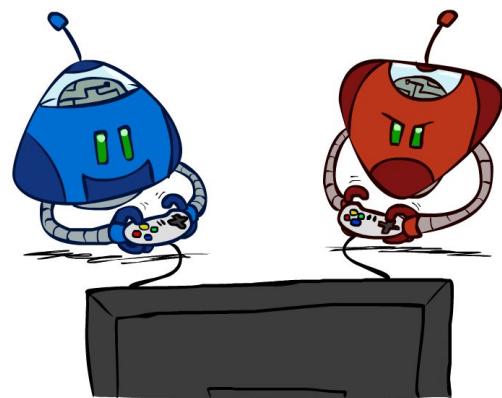


Example: Backtracking +AC-3 + MRV & Deg + LCV



$D = \{red, green, blue\}$
 $X = \{NSW, WA, NT, Q, SA, V, T\}$

Adversarial Search

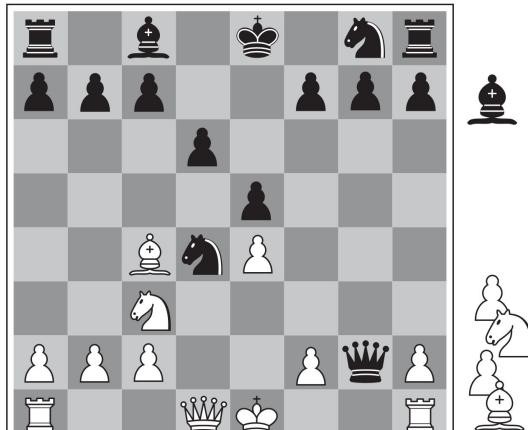


Adversarial Search

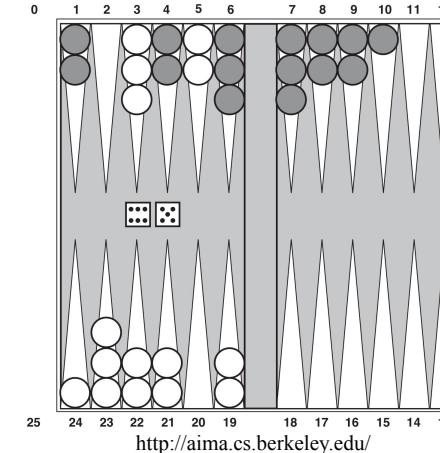
- A multi-agent competitive environment
 - We try to plan ahead in a world where other agents are planning against us
 - Goals are in conflict (not necessarily)



<http://ai.berkeley.edu>



<http://aima.cs.berkeley.edu/>

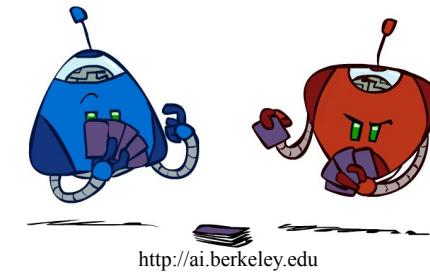


<http://aima.cs.berkeley.edu/>

X	O
X	
O	

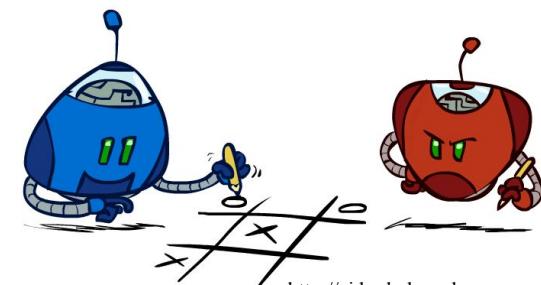
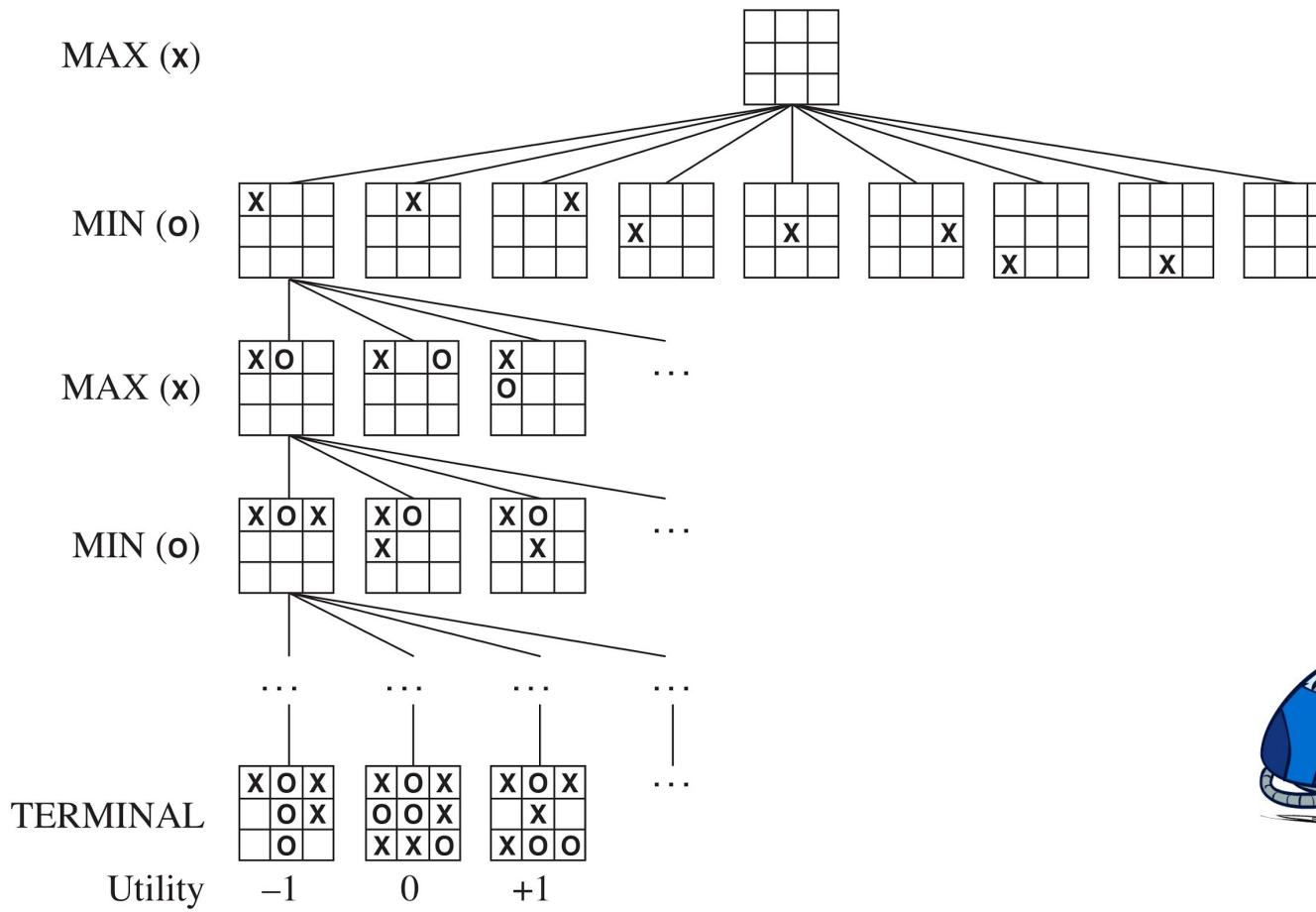
Game Definition

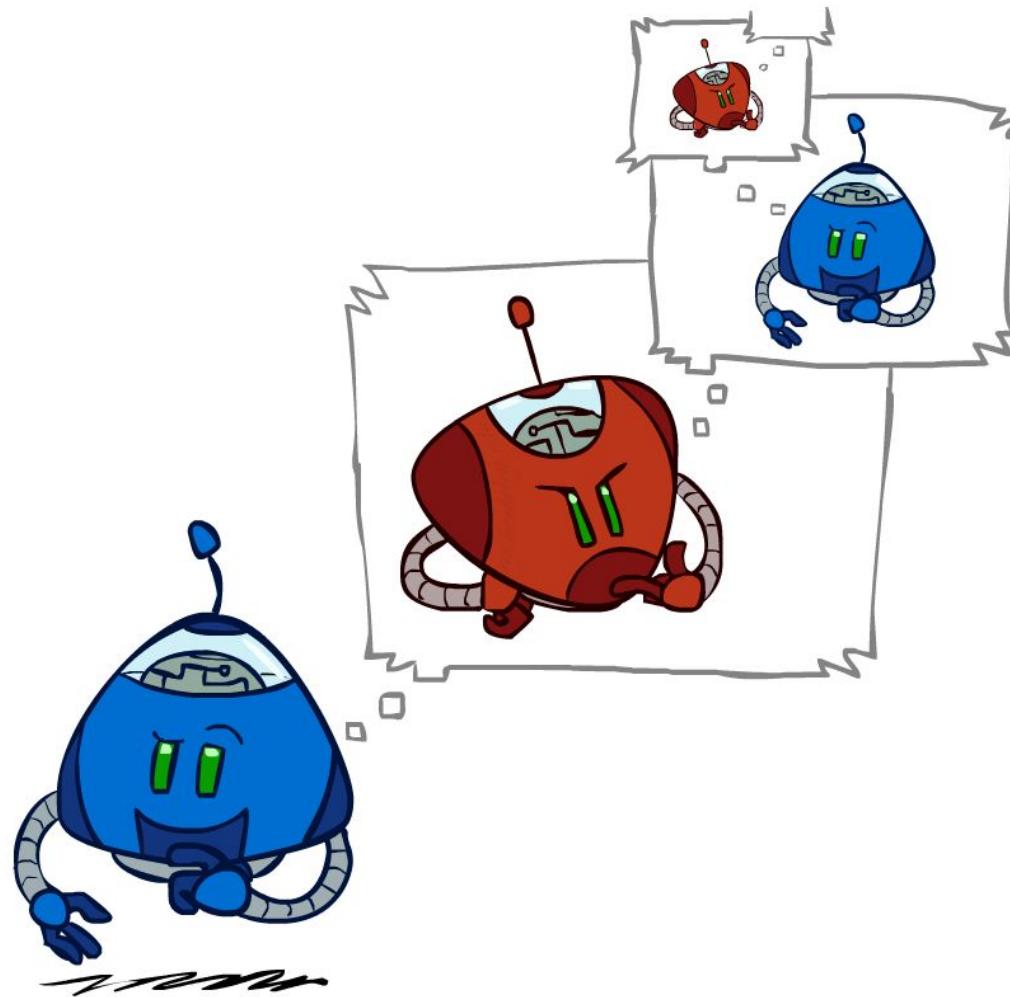
- A game can be defined as
 - s : States
 - s_0 : Initial state
 - Player(s) : Defines which player has the move
 - Actions(s) : Returns a set of legal moves
 - Result(s,a) : Defines the result of a move
 - TerminalTest(s) : True when game is over, false otherwise
 - Utility(s,p) : Defines the final numeric value for a game that ends in terminal state s for player p
- A game tree can be constructed
 - Nodes are game states and edges are moves



<http://ai.berkeley.edu>

Tic-Tac-Toe Game Tree





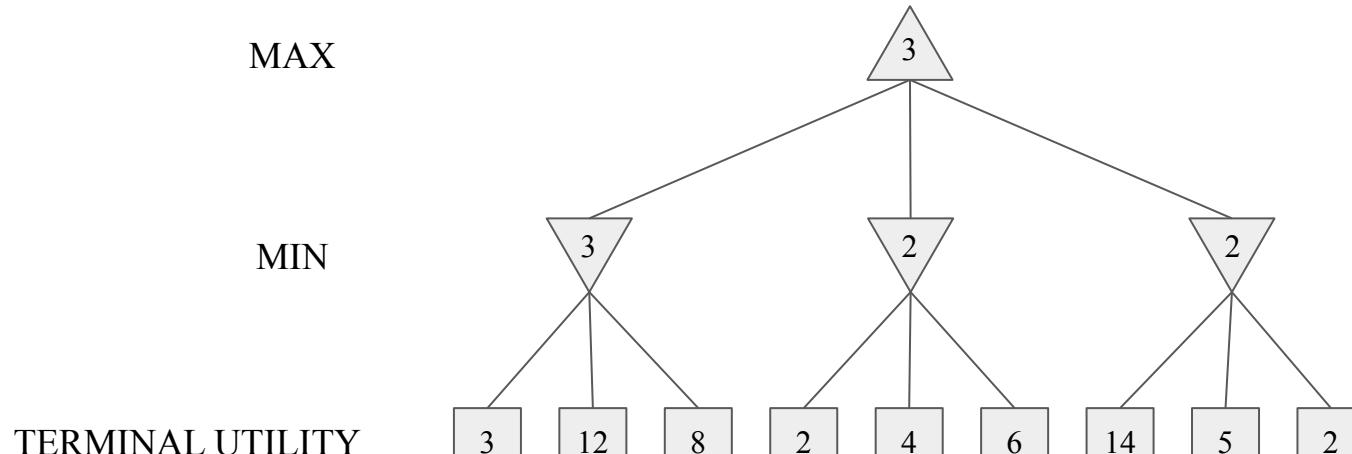
Minimax Search

- A state-space search tree
- Players alternate turns
- Compute each node's minimax value
 - the best achievable utility against a rational (optimal) adversary

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

- Will lead to optimal strategy
 - Best achievable payoff against best play

Minimax Search - Example



$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax Implementation

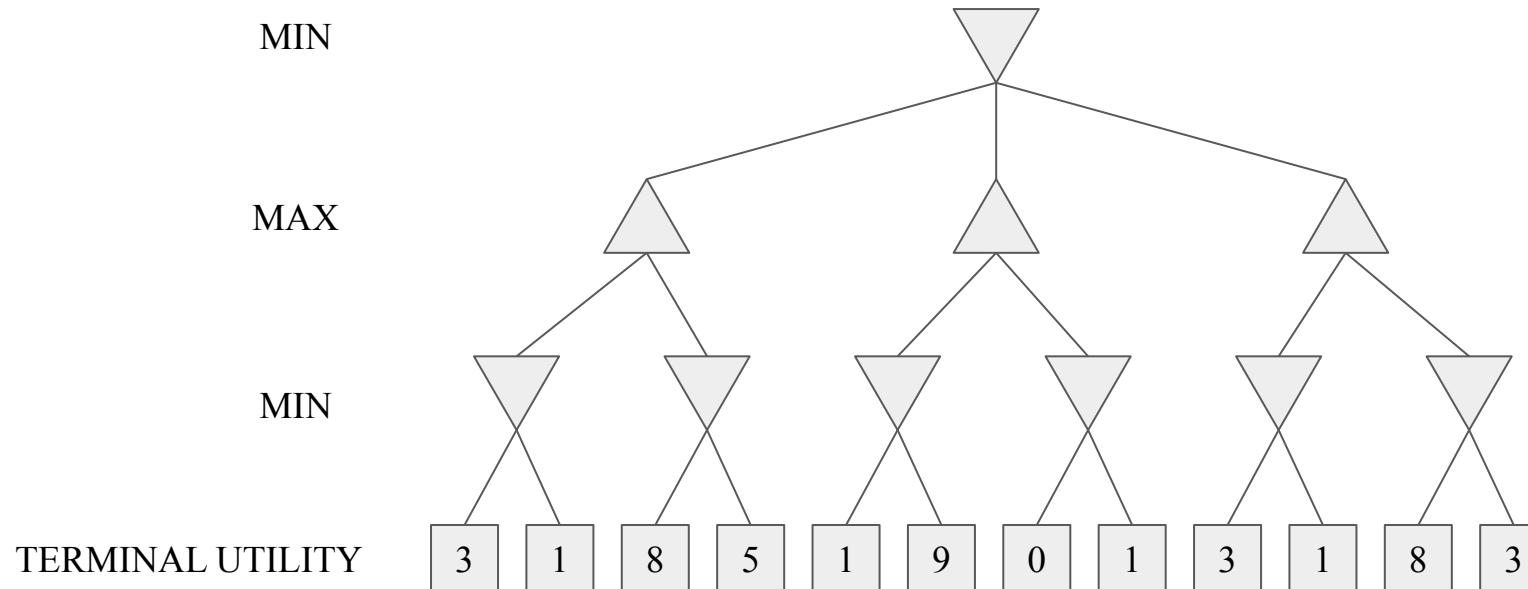
```
function MINIMAX-DECISION(state) returns an action
    return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow -\infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow \infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

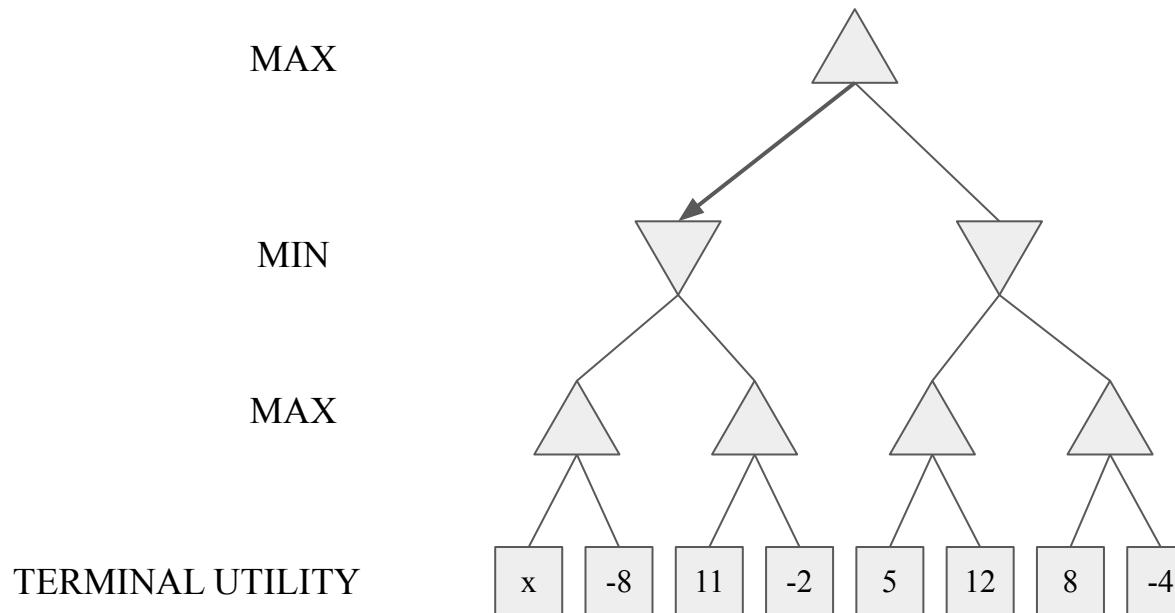
Quiz

- Consider the simple game tree shown below
 - What is the minimax value of the game tree?
 - Which action will the minimizer take when playing according to the minimax strategy?



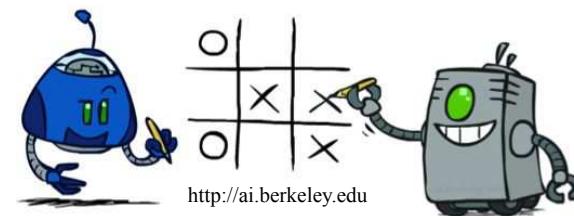
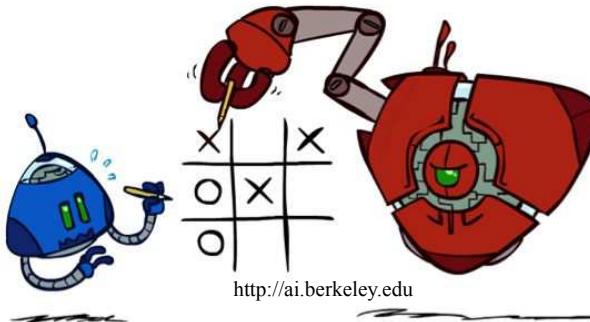
Quiz

- Consider the following game tree, where one of the leaves has an unknown payoff x
- For what values of x is MAX guaranteed to choose the initial left action?



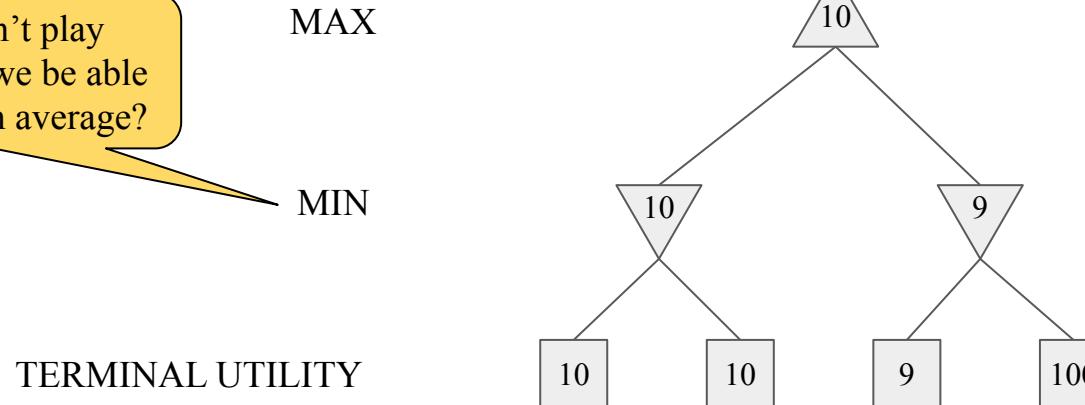
Minimax Properties

- Will minimax lead to optimal play?
 - No
 - It will lead to an optimal strategy
 - I.e., it will be optimal against perfect play



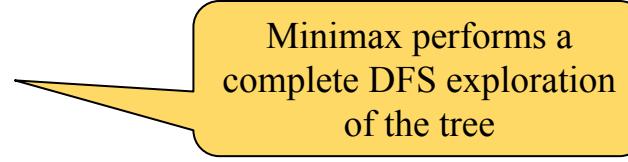
What if MIN does not play optimally?

If MIN doesn't play
optimally, will we be able
to play better on average?



Minimax Properties

- Complete?
 - Yes, if tree is finite
- Optimal?
 - In general no, yes against an optimal opponent
- Time complexity?
 - $O(b^m)$
- Space complexity
 - $O(bm)$



Minimax performs a complete DFS exploration of the tree

Example: Chess

- $b \approx 35, m \approx 100$
 - Cannot search to leaves !



Resource Limits

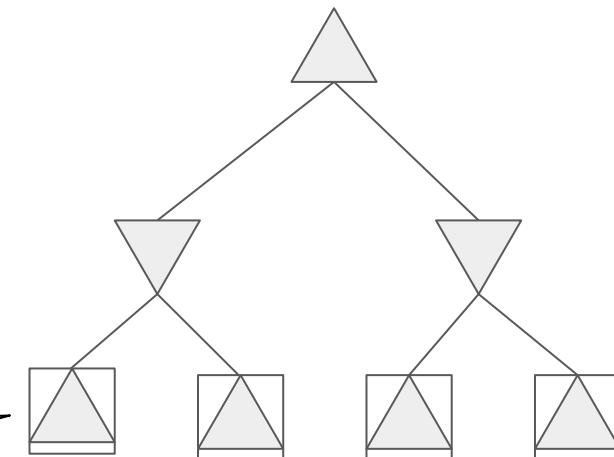


<http://ai.berkeley.edu>

DLS

- Depth-Limit Search
 - DFS-TSA with a depth limit and an evaluation function
 - i.e., nodes at depth l have no successors and use value of evaluation function
 - Properties
 - Complete ?
 - No, if $l < d$; Yes, if $l \geq d$
 - Optimal ?
 - No
 - Time ?
 - $O(b^l)$
 - Space ?
 - $O(bl)$

Cut the tree here

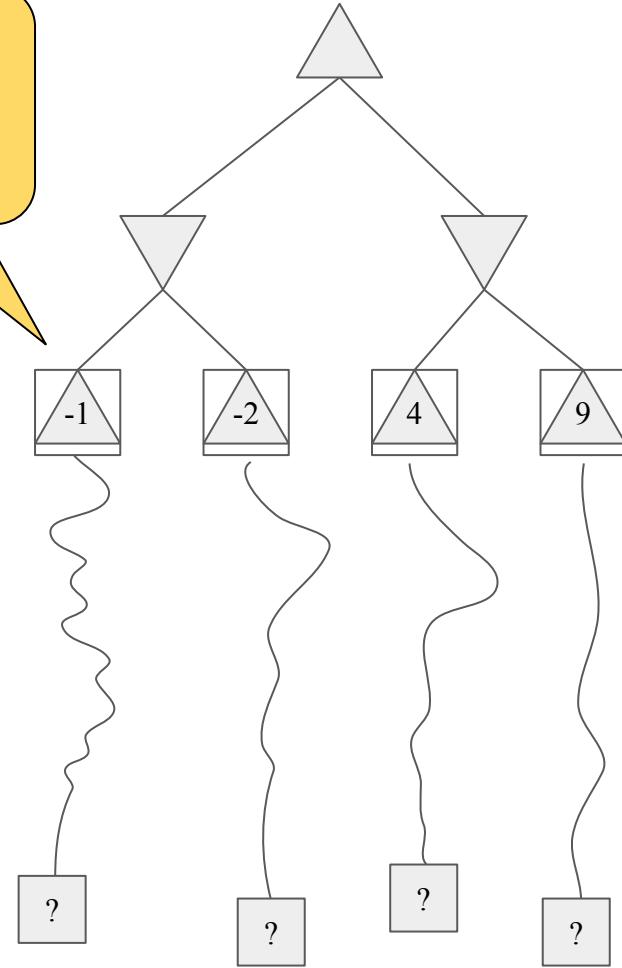


d : depth of the shallowest solution

DLS

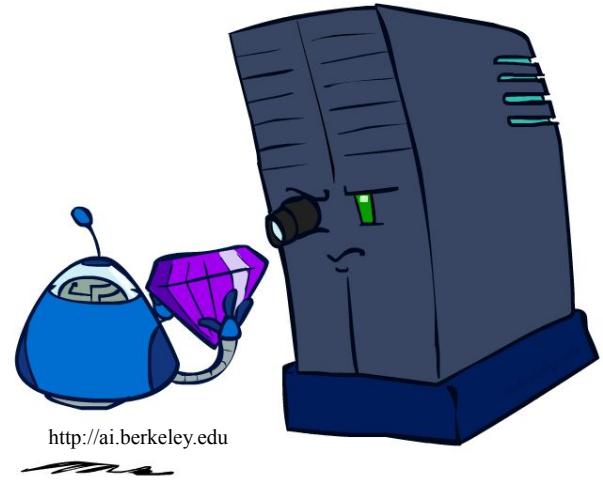
- Depth limited search
 - Search only to a limited depth in the tree
 - Replace terminal utilities with an evaluation function for non-terminal positions
- Problems:
 - Guarantee of optimal play is gone
 - Need to design evaluation function

Cut the tree here
and plug in these
numbers from the
evaluation
function



Evaluation Function

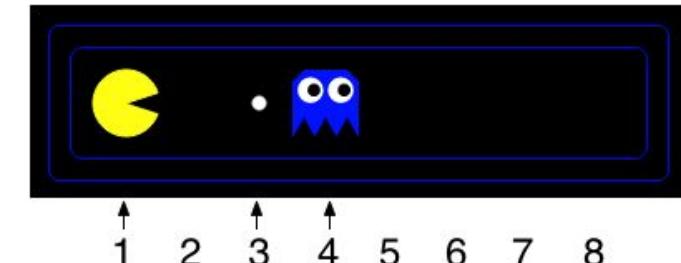
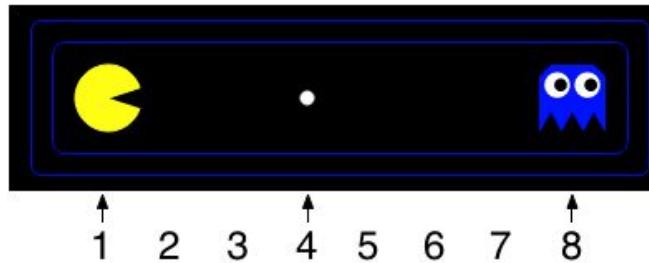
- An evaluation function $\text{Eval}(s)$ scores non-terminals in depth-limited search
 - An estimate of the expected utility of the game from a given position
- Ideal function
 - The actual minimax value of the position
- The performance of a game-playing program depends strongly on the quality of its evaluation function



<http://ai.berkeley.edu>

Quiz

- For the two situations shown below, which evaluation functions will give the situation on the left a higher score than the situation on the right?

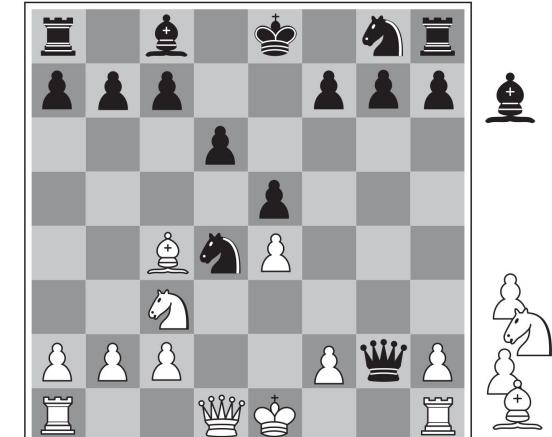


- $1 / (\text{Pacman's distance to the nearest food pellet})$
- Pacman's distance to the nearest ghost
- Pacman's distance to the nearest ghost + $1 / (\text{Pacman's distance to the nearest food pellet})$
- Pacman's distance to the nearest ghost + $1000 / (\text{Pacman's distance to the nearest food pellet})$

Evaluation Function

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

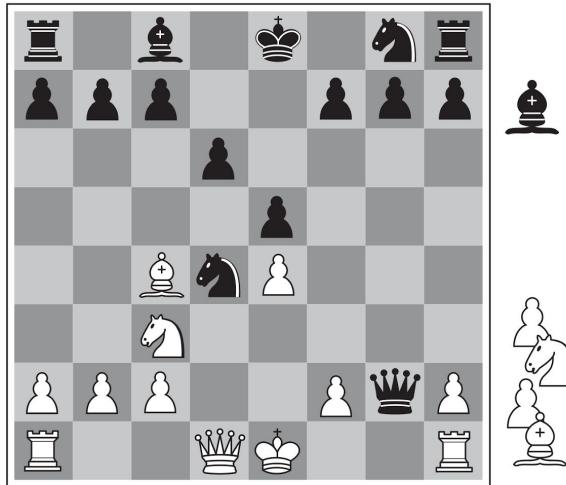
- Example: Chess
 - Successful evaluation functions for chess compute numerical contributions from each feature
 - E.g., each pawn is worth 1, a knight or bishop is worth 3, a rook 5 and a queen 9
 - Other features such as “king safety” or “good pawn structure” might be worth half a pawn
 - The features and their weights are then combined



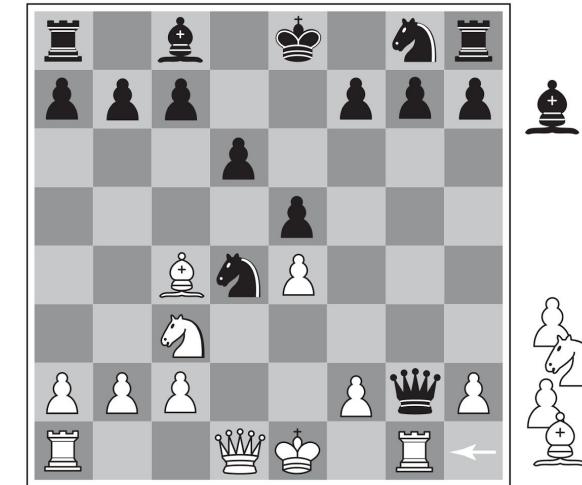
<http://aima.cs.berkeley.edu/>

Quiz

- Which player has an advantage for (a) and for (b) ?



(a) White to move



(b) White to move

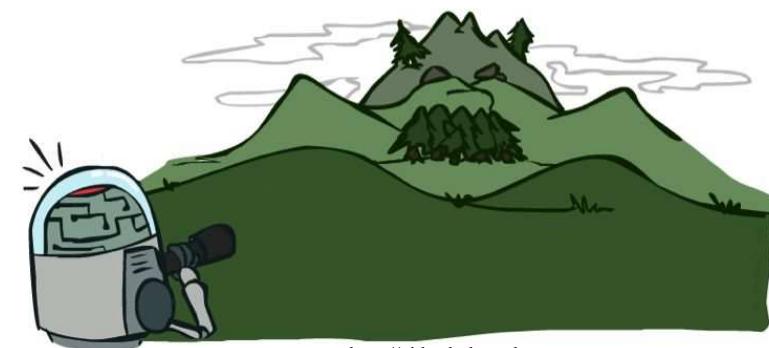
Figure 5.8 Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the trade off between complexity of features and complexity of computation



<http://ai.berkeley.edu>



<http://ai.berkeley.edu>

Horizon Effect

- Consider an opponent's move that causes serious damage and is ultimately unavoidable
- With a low depth limit we don't know that the damage is ultimately unavoidable
- Employing delaying tactics to temporality avoided the damage may cause even more damage



Horizon Effect - Example

- Consider the following example
 - Can black save its bishop?

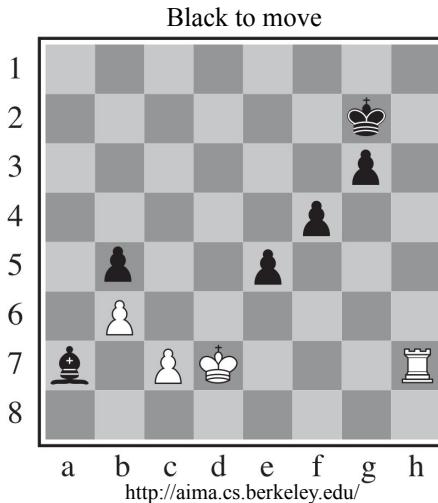
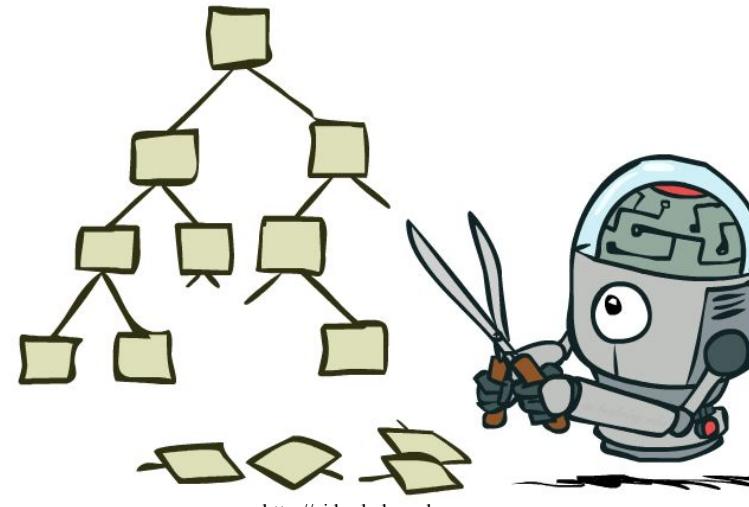


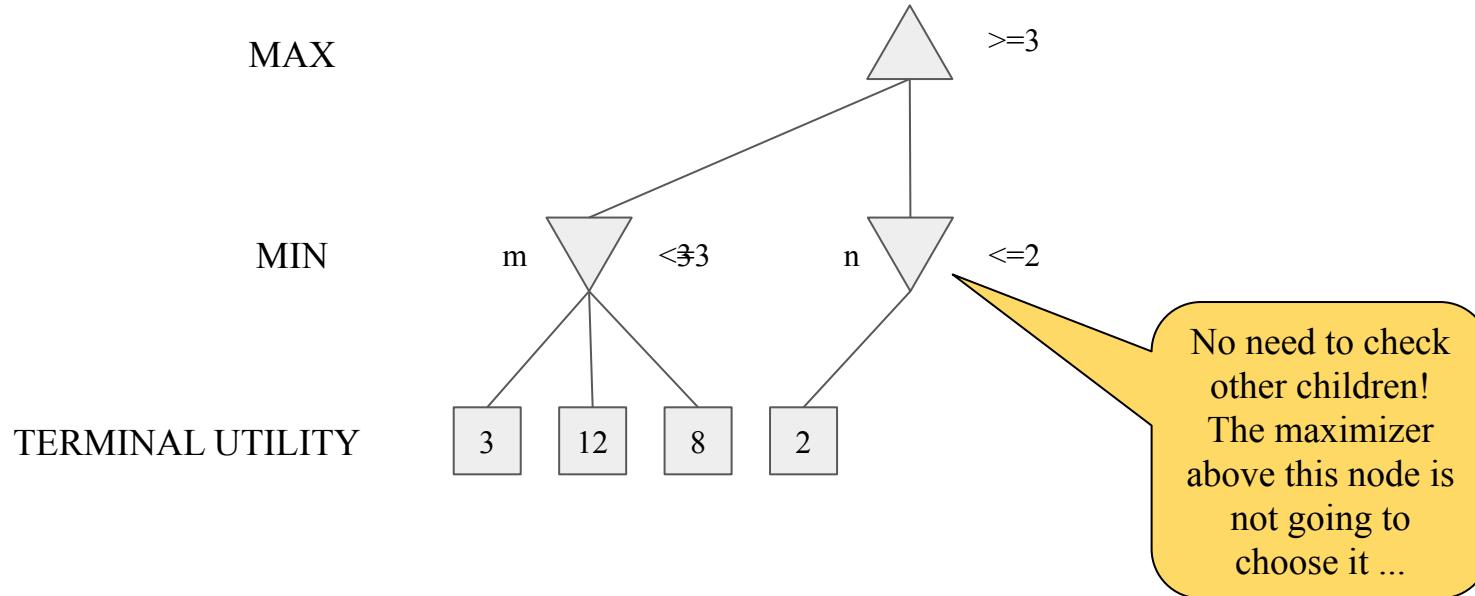
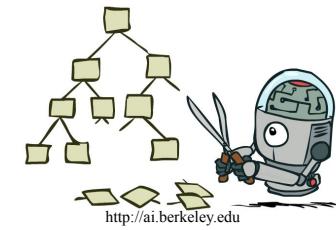
Figure 5.9 The horizon effect. With Black to move, the black bishop is surely doomed. But Black can forestall that event by checking the white king with its pawns, forcing the king to capture the pawns. This pushes the inevitable loss of the bishop over the horizon, and thus the pawn sacrifices are seen by the search algorithm as good moves rather than bad ones.

Game Tree Pruning

- Is it possible to compute the correct minimax value without looking at every node?

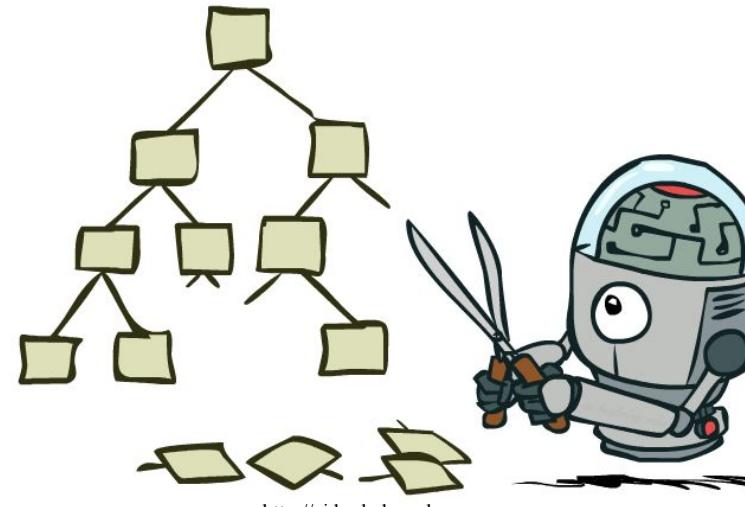


Pruning - Motivation



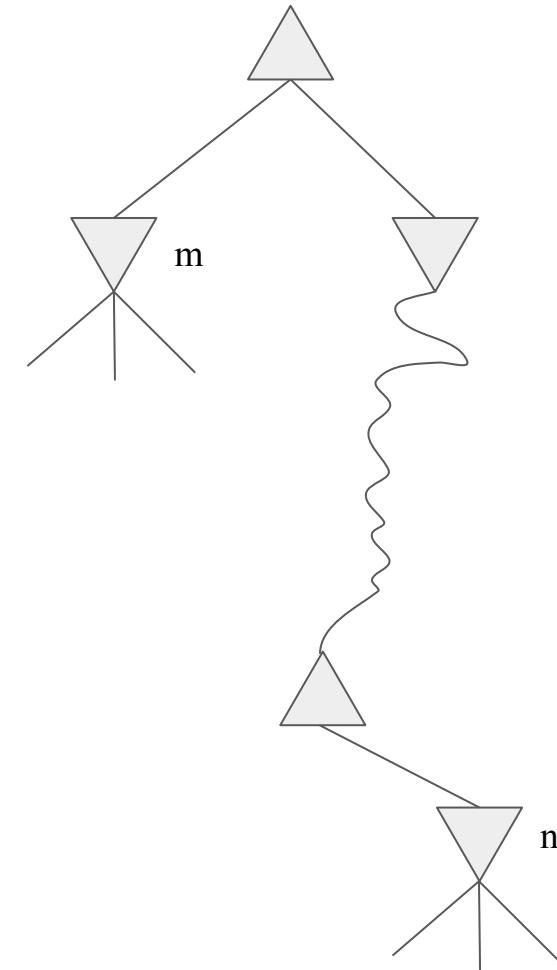
Pruning

- The α - β pruning algorithm can determine the minimax value without looking at all the nodes



α - β Pruning Algorithm

- Min version
 - Consider Min's value at some node n
 - n will decrease (or stay constant) while the descendants of n are examined
 - Let m be the best value that Max can get at any choice point along the current path from the root
 - If n becomes worse (<) than m
 - Max will avoid it
 - Stop considering n's other children
- Max version is symmetric



α - β Pruning Algorithm

function MINIMAX-DECISION(*state*) **returns** an action
return $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$

function MAX-VALUE(*state*) **returns** a utility value
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$
return *v*

function MIN-VALUE(*state*) **returns** a utility value
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow \infty$
for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$
return *v*

α = best explored option along path to root for max
 β = best explored option along path to root for min

function ALPHA-BETA-SEARCH(*state*) **returns** an action
 $v \leftarrow \text{MAX-VALUE}(s, -\infty, +\infty)$
return the action in ACTIONS(*state*) with value *v*

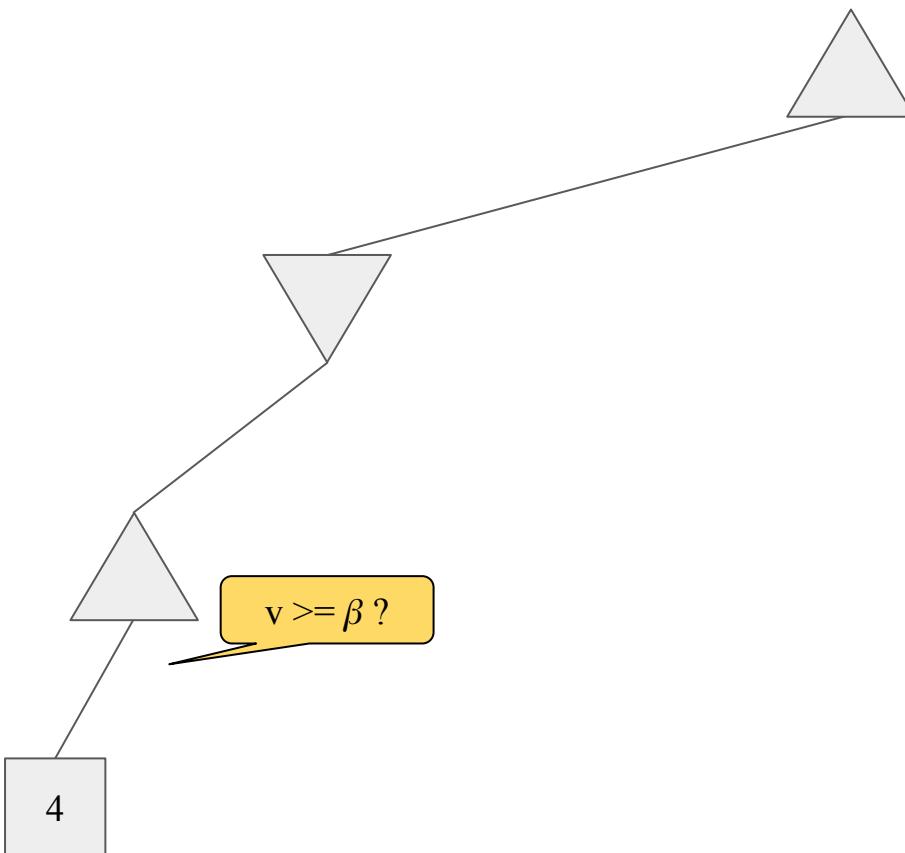
function MAX-VALUE(*state*, α , β) **returns** a utility value
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow -\infty$
for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$
if *v* $\geq \beta$ **then return** *v*
 $\alpha \leftarrow \text{MAX}(\alpha, v)$
return *v*

function MIN-VALUE(*state*, α , β) **returns** a utility value
if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
 $v \leftarrow +\infty$
for each *a* **in** ACTIONS(*state*) **do**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$
if *v* $\leq \alpha$ **then return** *v*
 $\beta \leftarrow \text{MIN}(\beta, v)$
return *v*

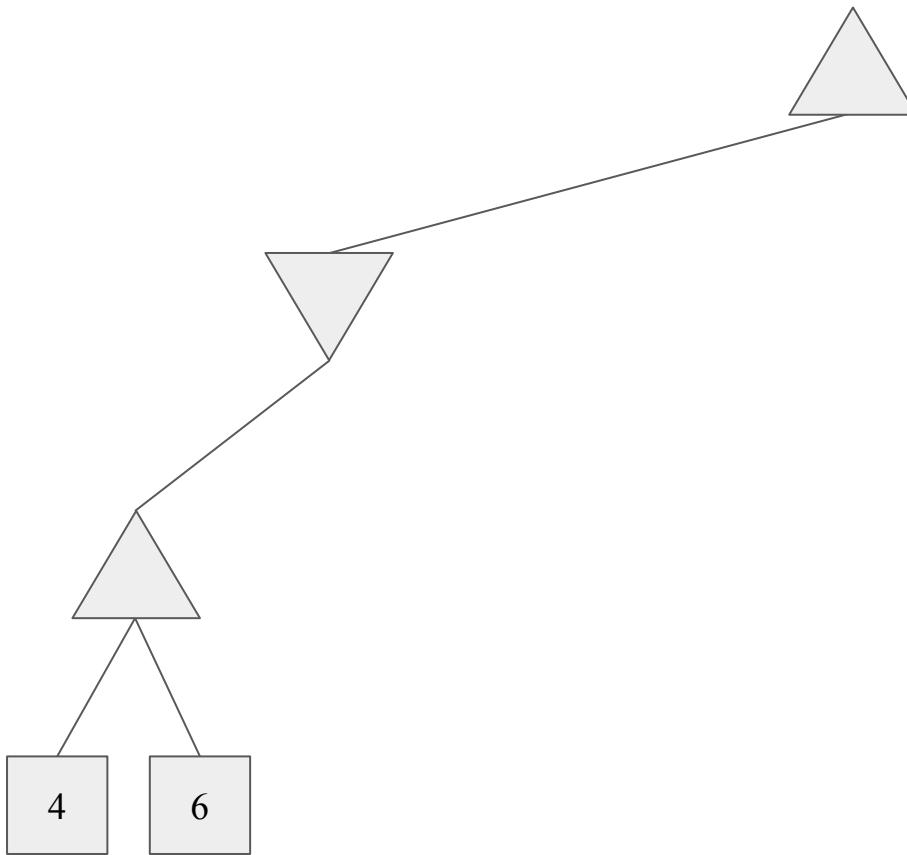
α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example



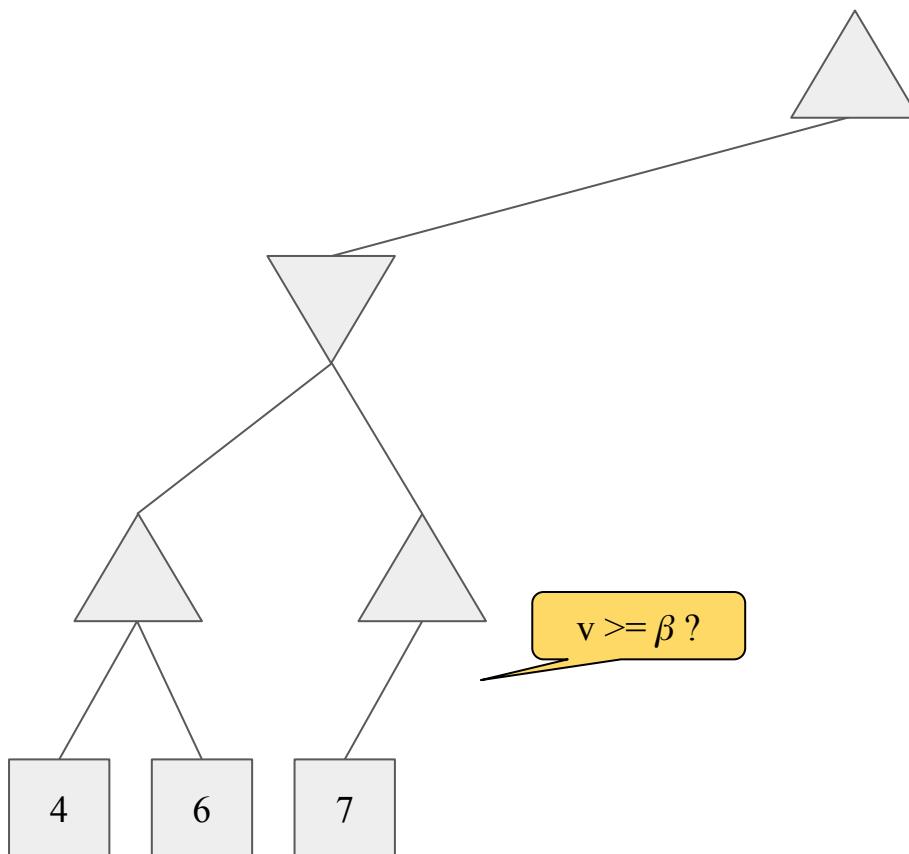
α = best explored option along path to root for max
 β = best explored option along path to root for min

Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)

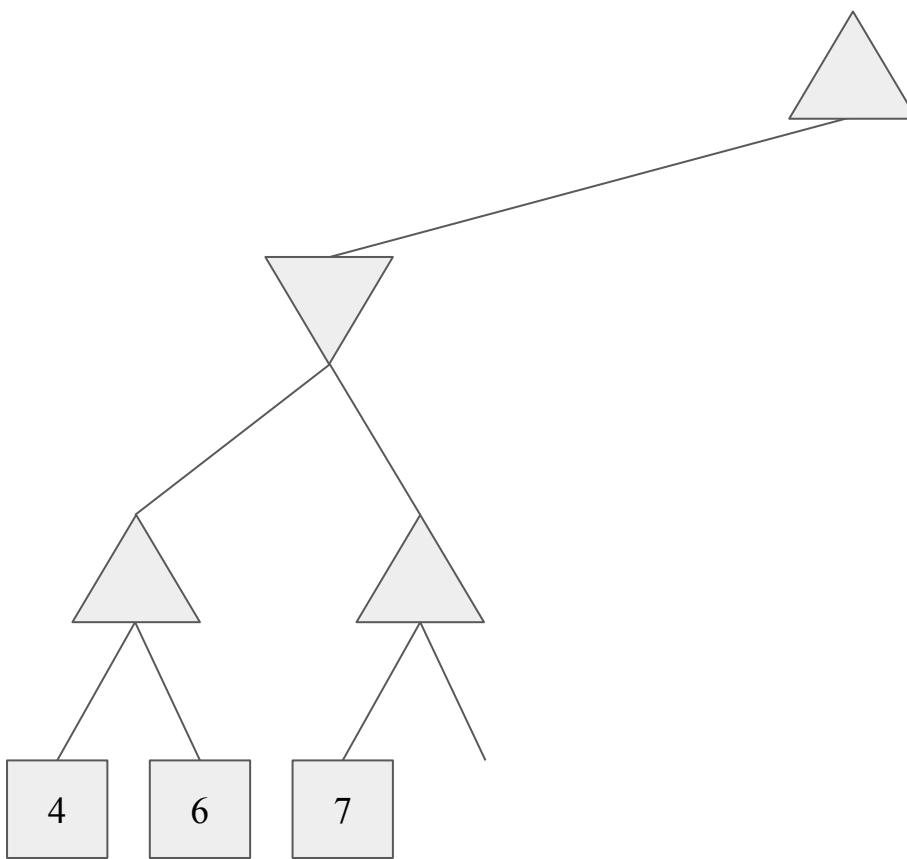
α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example



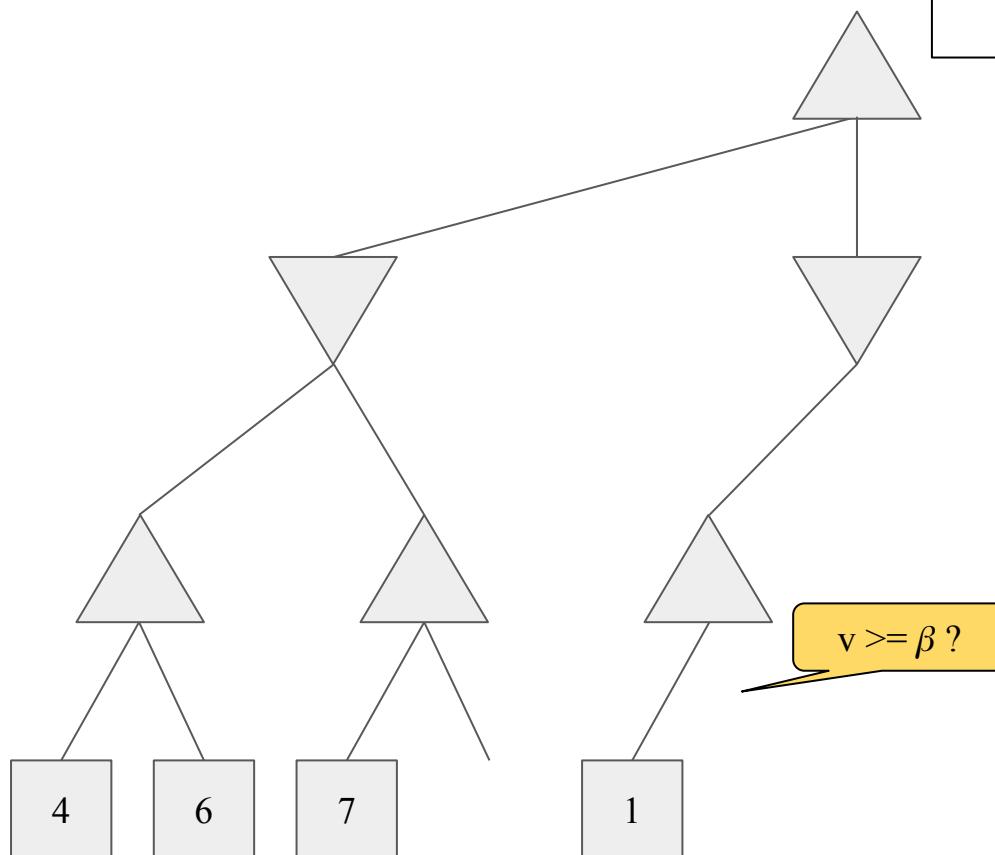
α = best explored option along path to root for max
 β = best explored option along path to root for min

Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)

α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

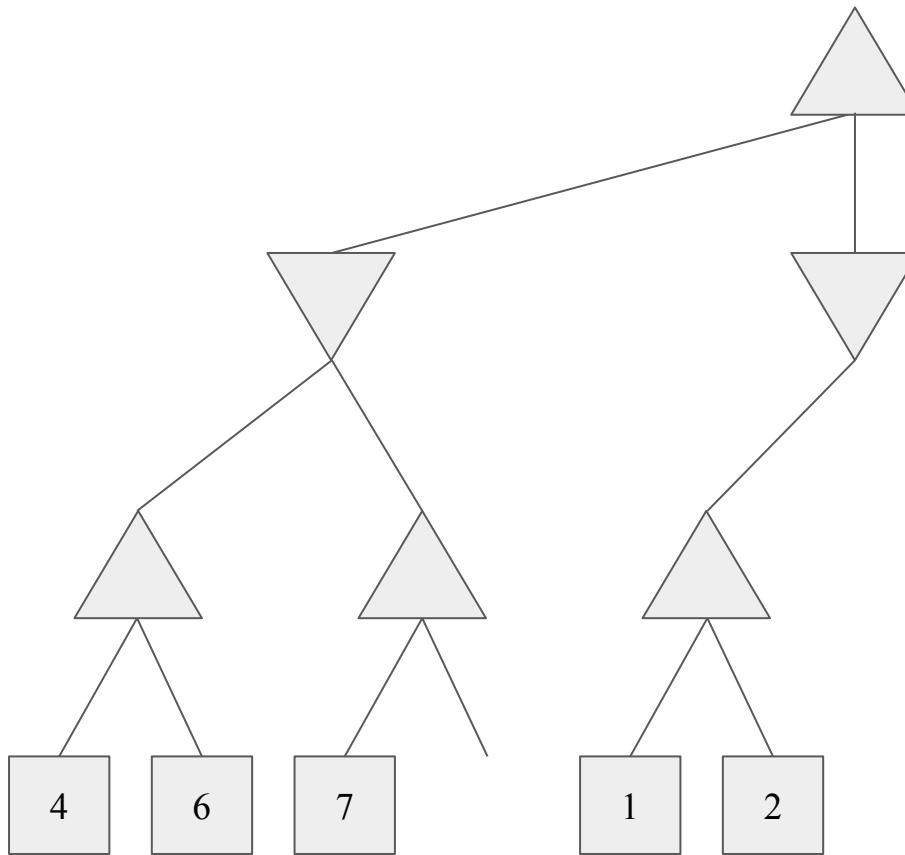
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

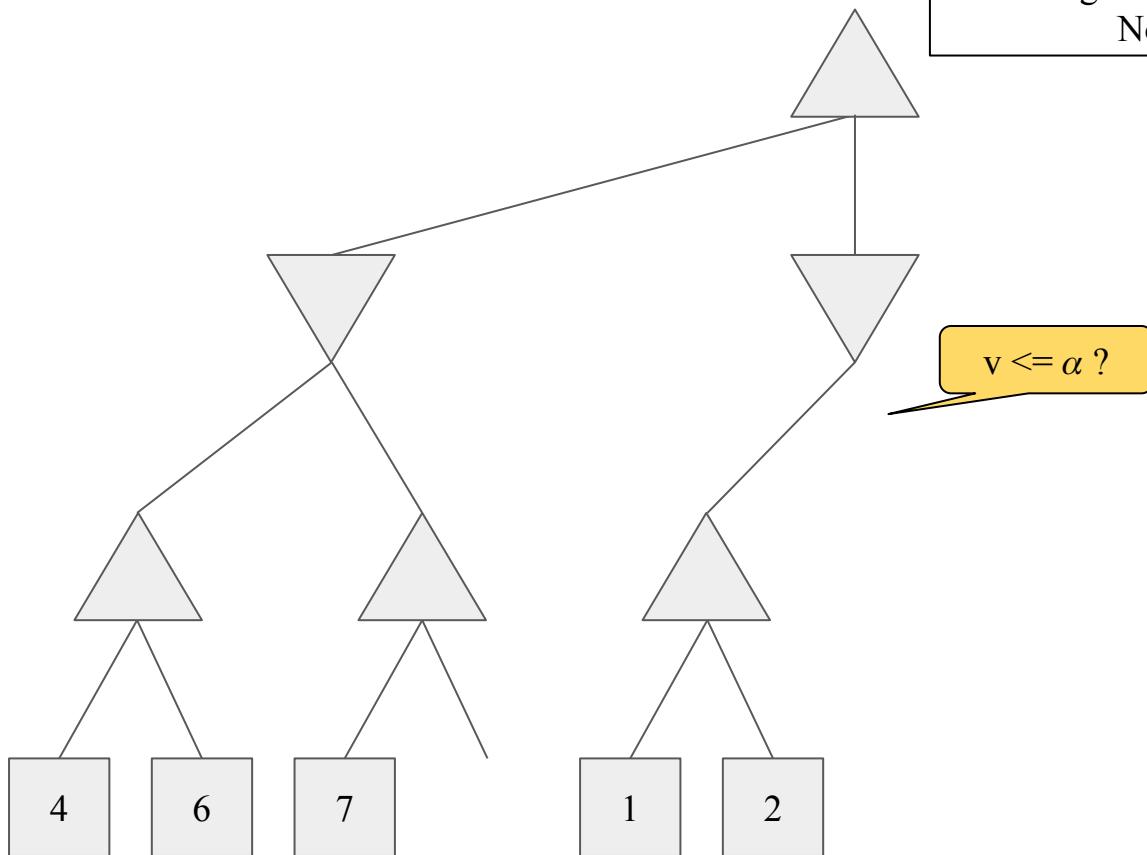
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

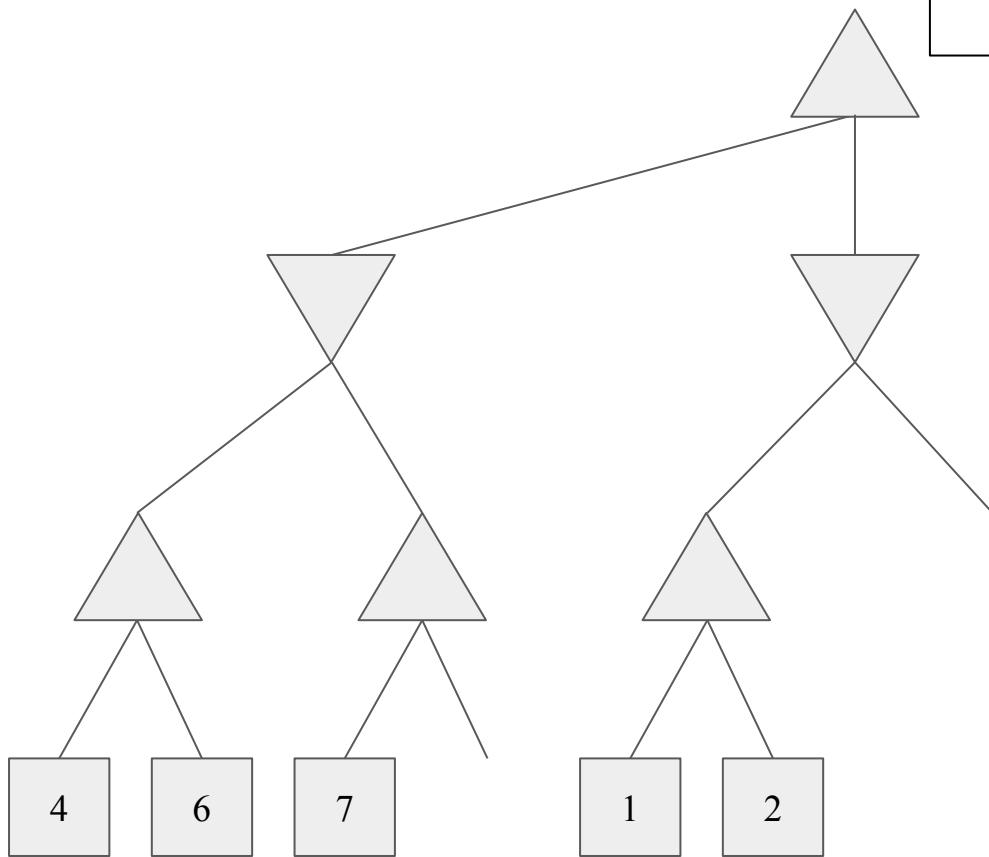
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

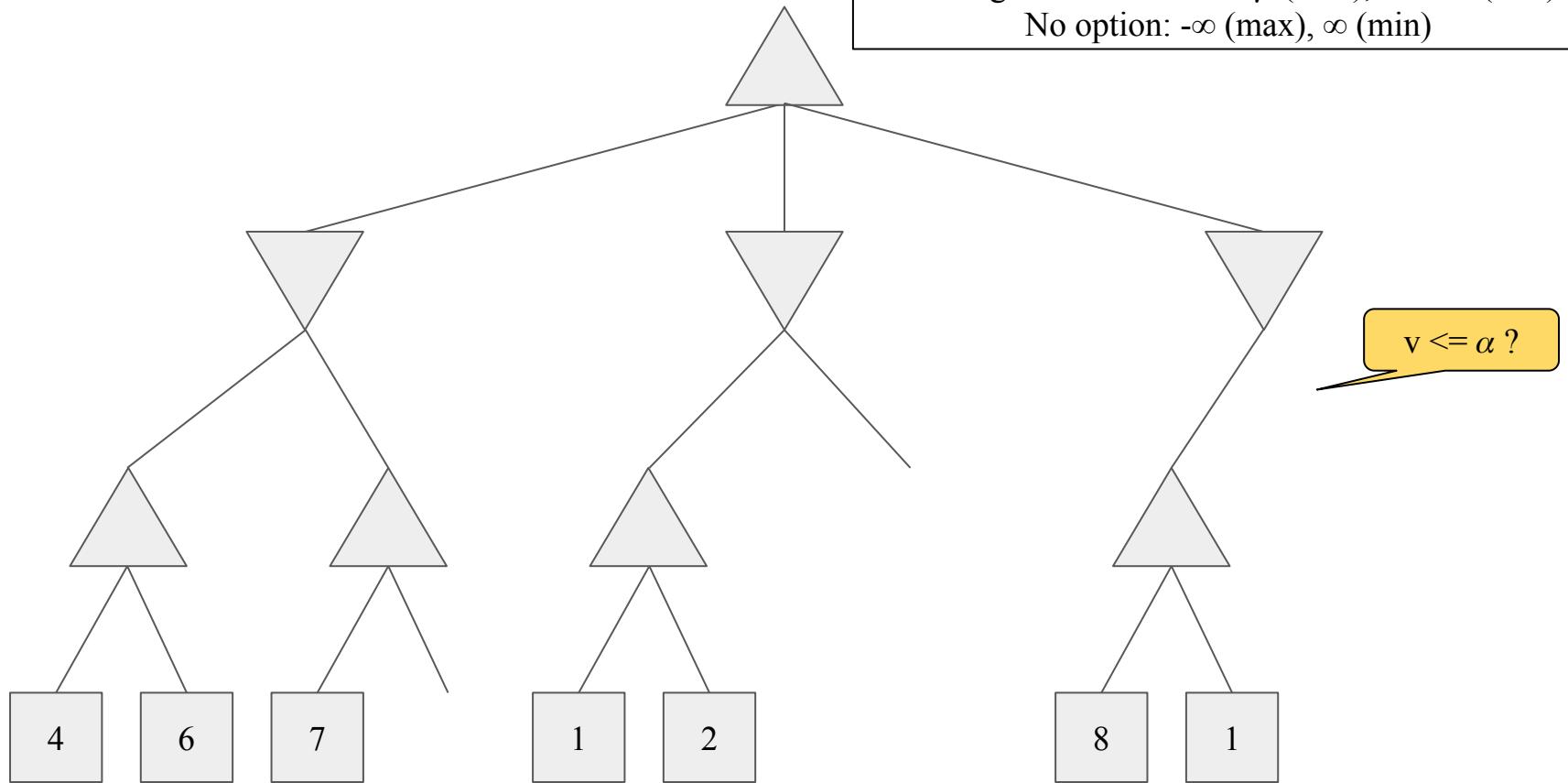
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

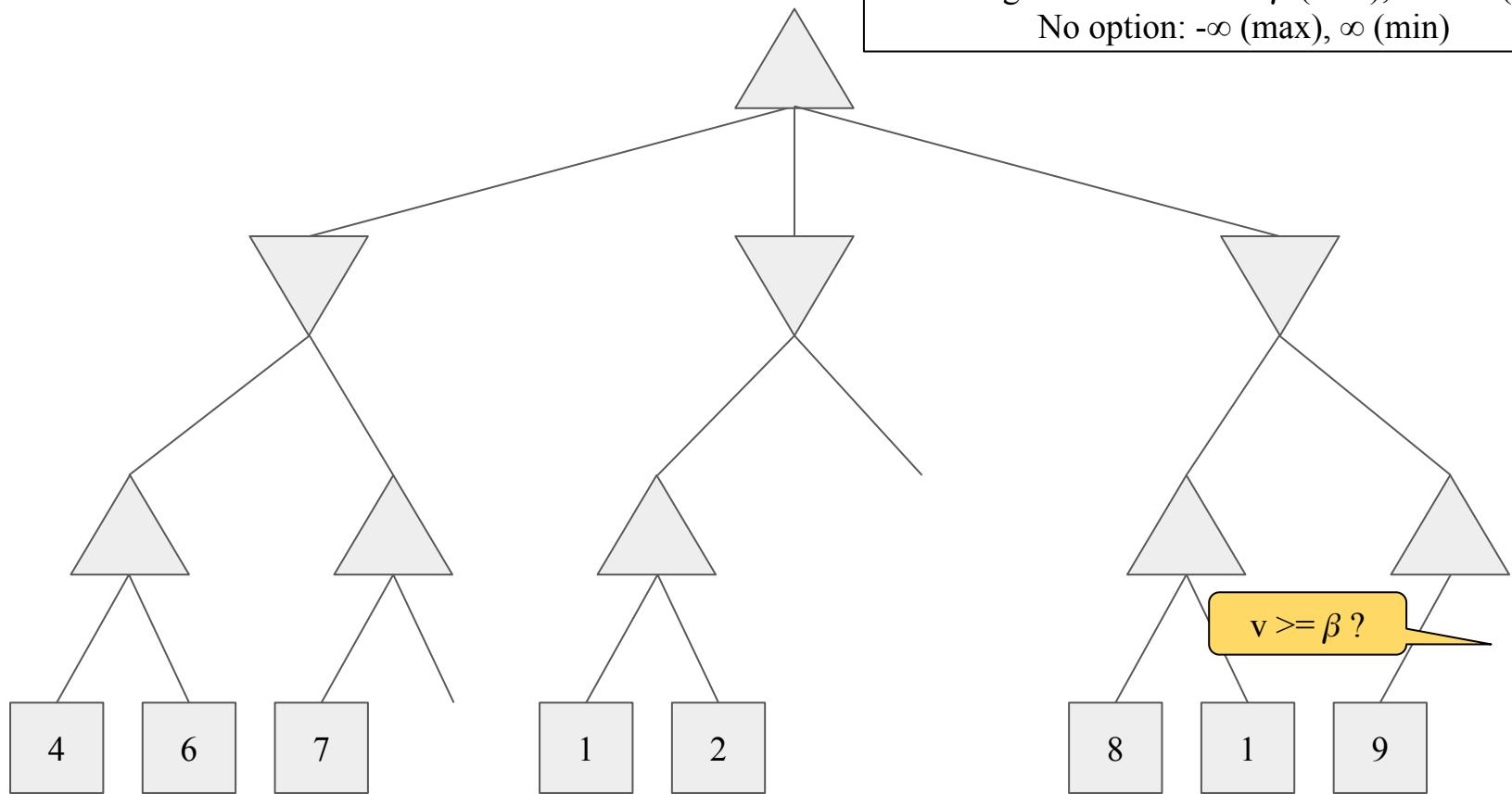
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

α = best explored option along path to root for max
 β = best explored option along path to root for min

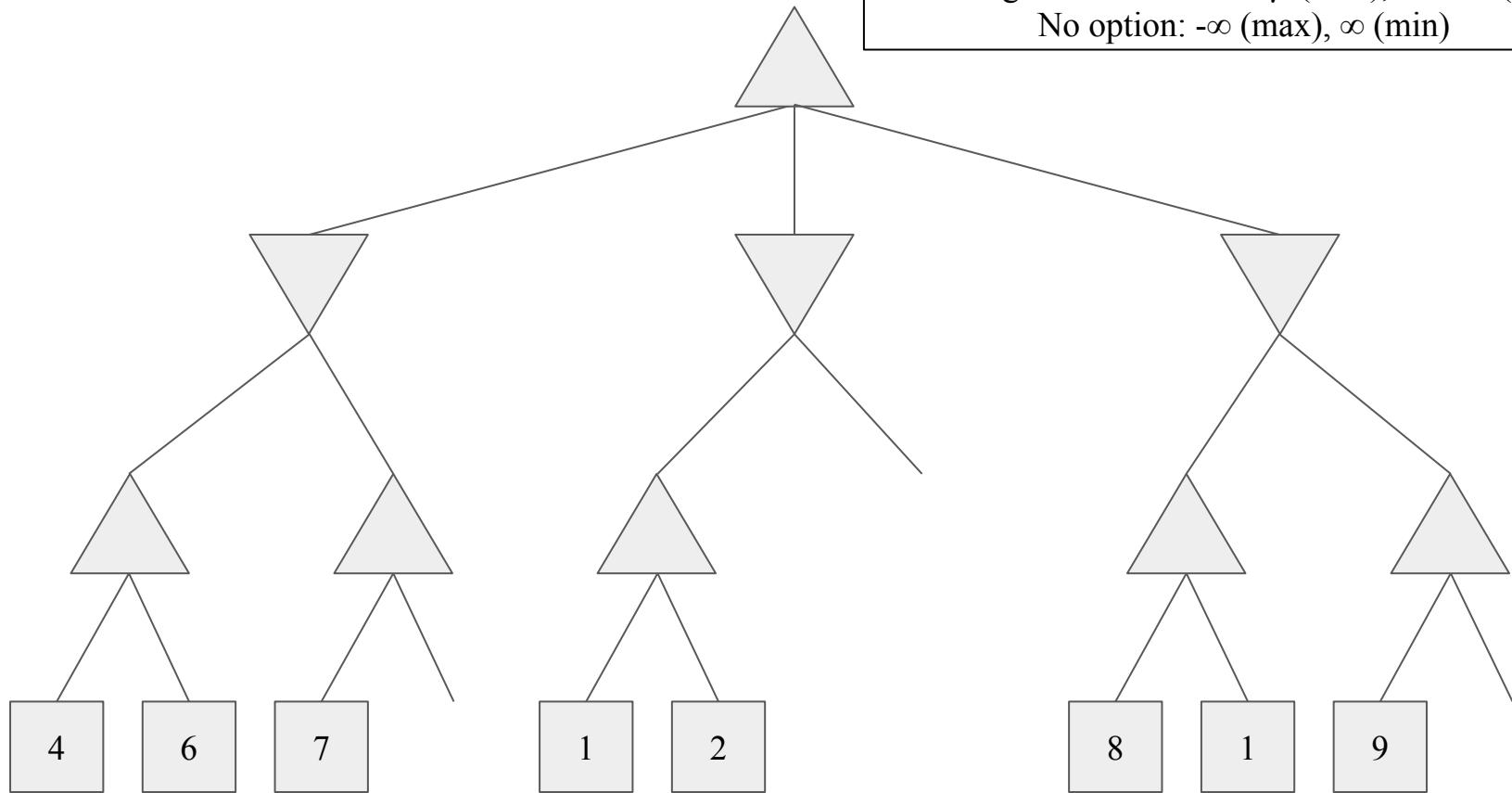
Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



α - β Pruning Example

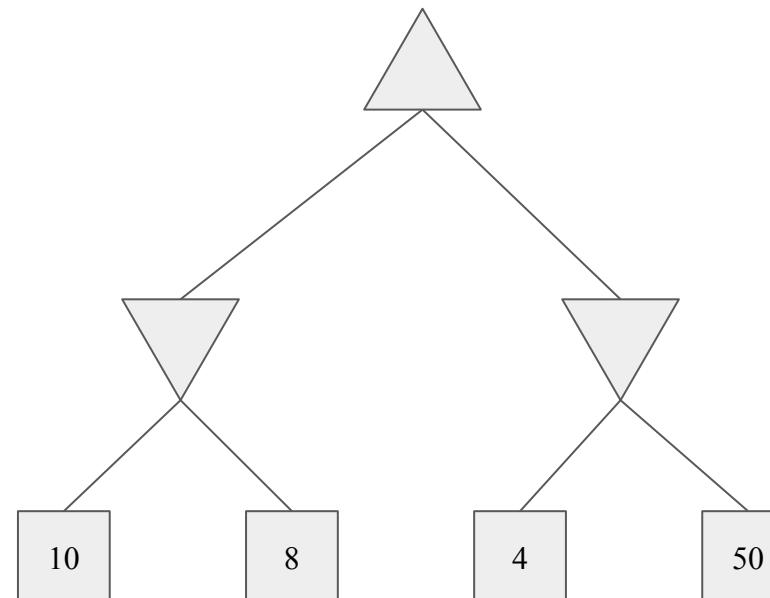
α = best explored option along path to root for max
 β = best explored option along path to root for min

Pruning Conditions: $v \geq \beta$ (max), $v \leq \alpha$ (min)
No option: $-\infty$ (max), ∞ (min)



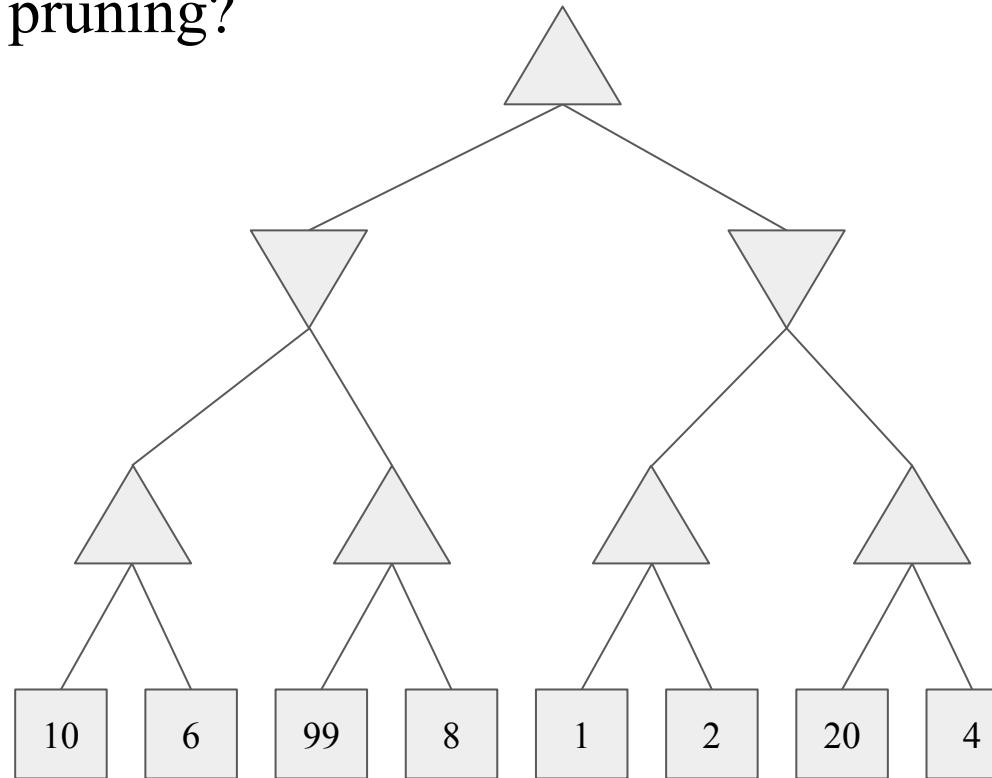
Quiz

- For the game tree shown below, which branches will be pruned by alpha-beta pruning?



Quiz

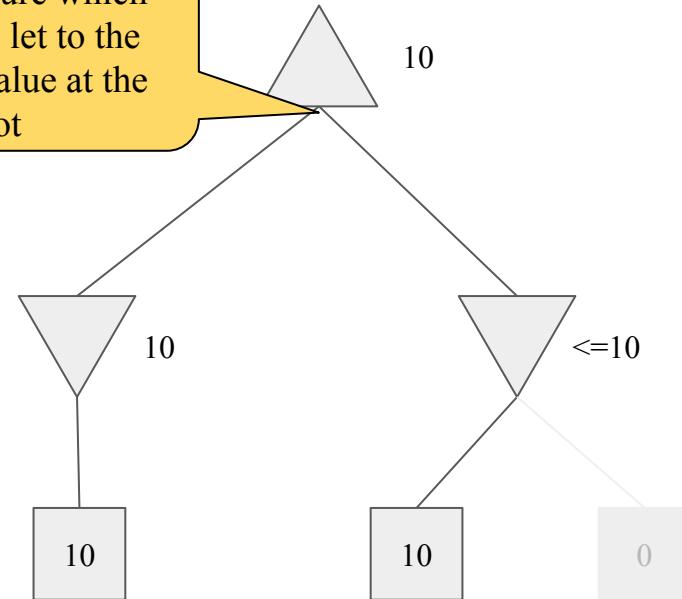
- For the game tree shown below, which branches will be pruned by alpha-beta pruning?



α - β Pruning Properties

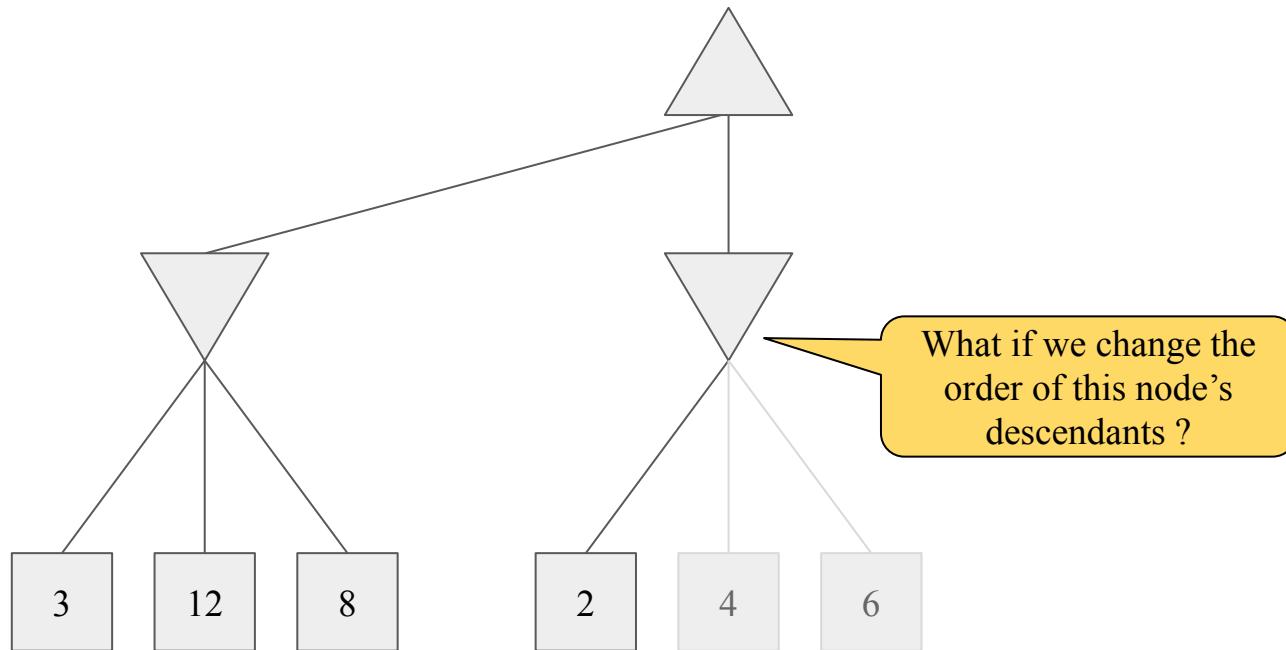
- Pruning has no effect on the minimax value at the root
- However, values of intermediate nodes might be wrong
 - Action selection not appropriate for this simple version of alpha-beta pruning

We are unsure which action will lead to the minimax value at the root



Move Ordering

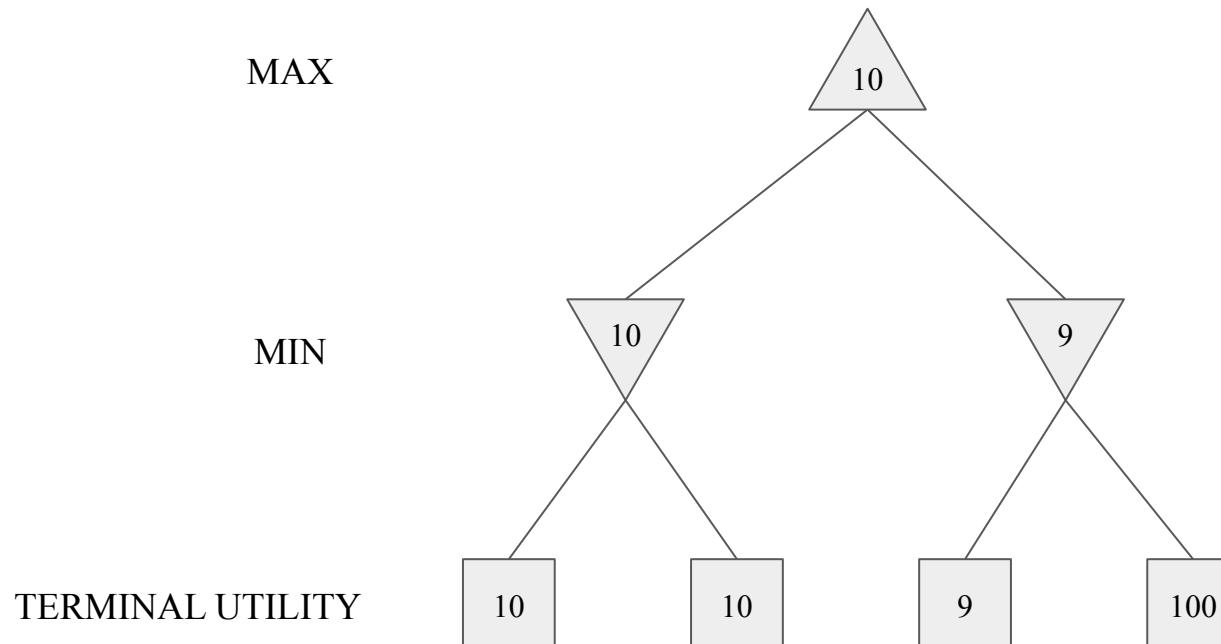
- The effectiveness of alpha-beta pruning is highly dependent on the order in which states are examined



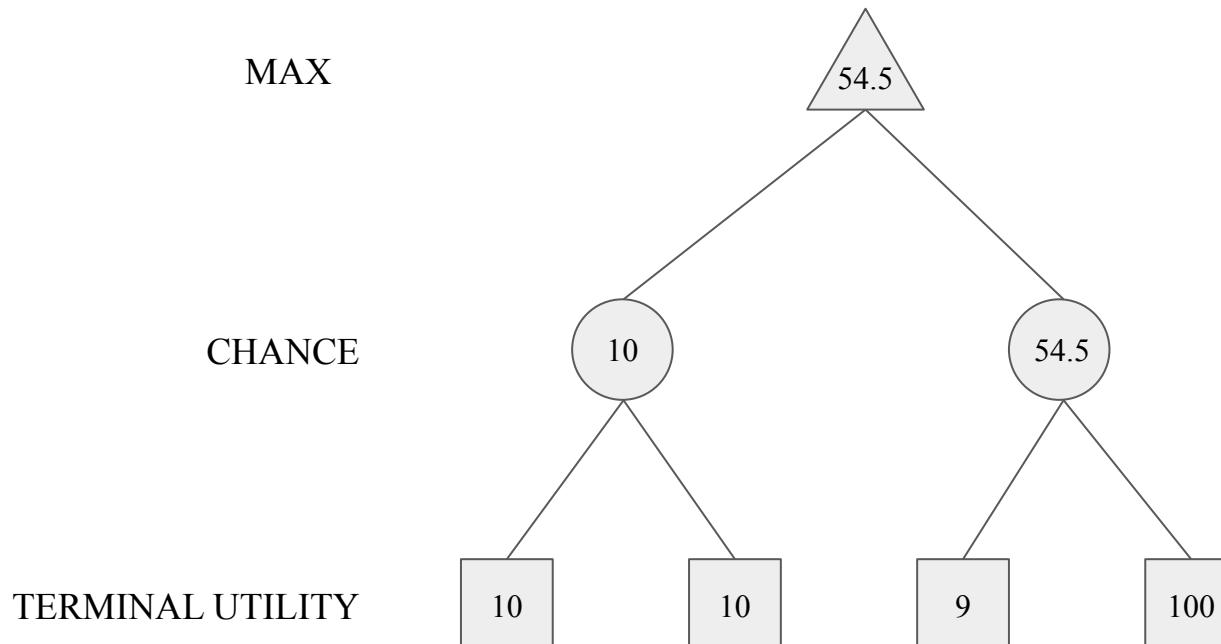
Alpha-Beta Pruning Properties

- It is worthwhile to try to examine first the successors that are likely best
- If this can be done, then it turns out that alpha-beta needs to examine only $O(b^{m/2})$ nodes to pick the best move, instead of $O(b^m)$ for minimax
 - Knuth, D. E., and Moore, R. W. [An analysis of Alpha-Beta pruning](#), 1975

Worst Case vs. Average Case



Worst Case vs. Average Case

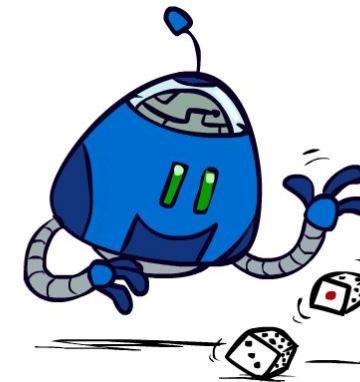


Chance Note

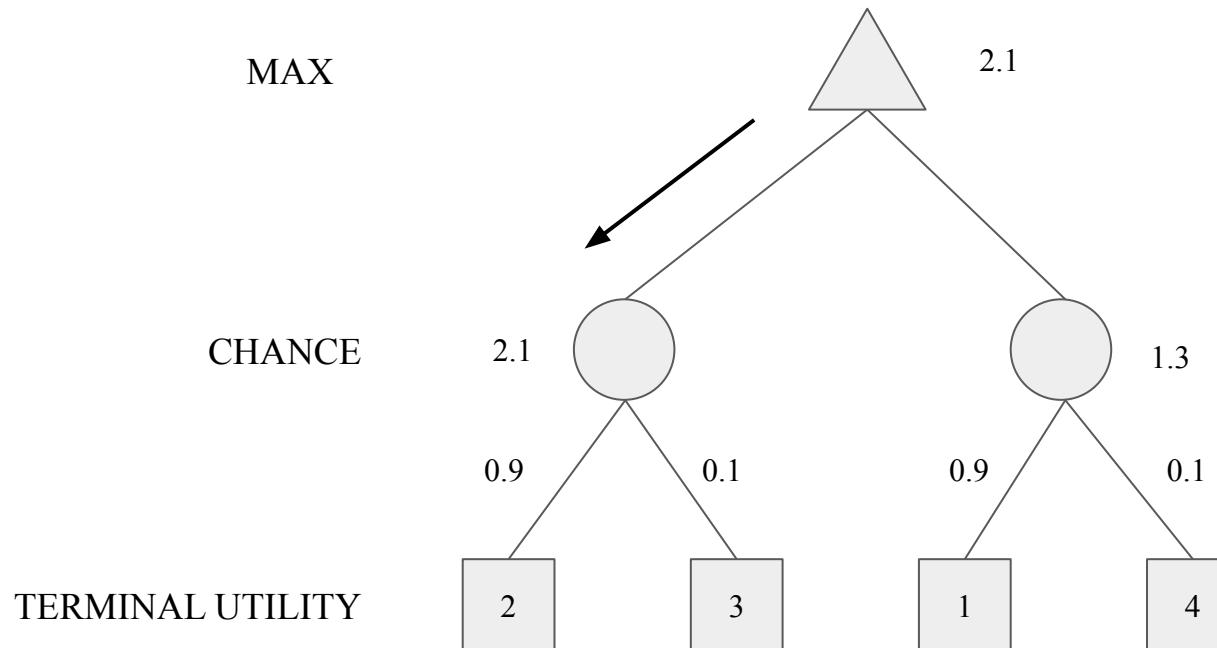
- Why wouldn't we know what the result of an action will be?
 - Explicit randomness: rolling dice
 - Unpredictable opponents: the ghosts respond randomly
 - Actions can fail: when moving a robot, wheels might slip

Expectimax Search

- Values reflect average case outcomes, not worst case outcomes
- Expectimax search computes the expected score under optimal play
 - Max nodes as in minimax search
 - Chance nodes are like min nodes but the outcome is uncertain
 - Calculate their expected utilities
 - I.e., take weighted average of children

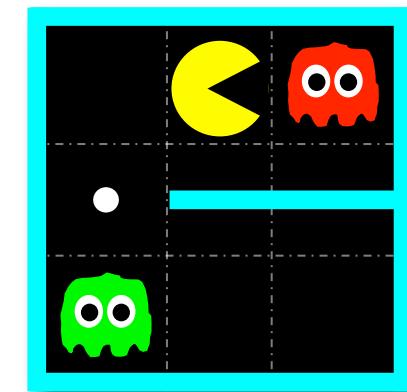


Expectimax Search Example



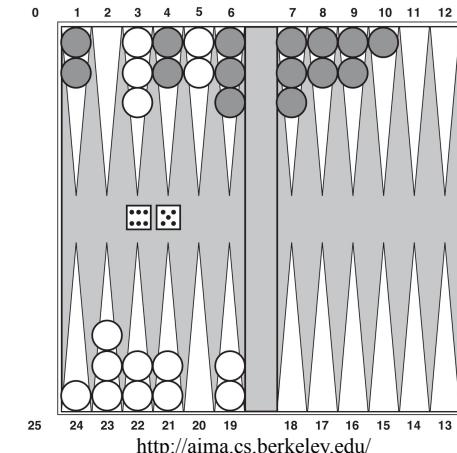
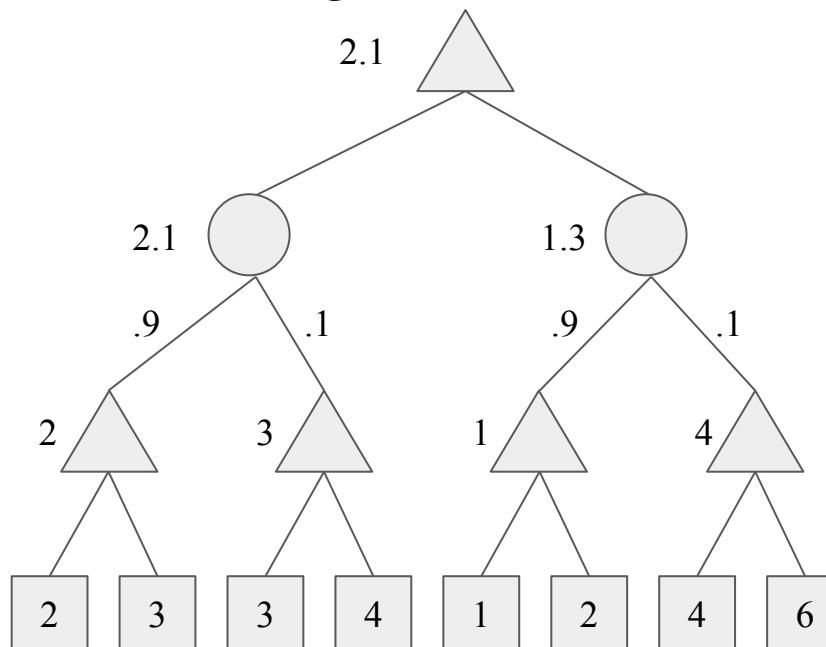
Quiz

- Given the maze below, build the expectimax tree. Assume that Pacman moves first, followed by the lower left ghost, then the top right ghost
- Rules:
 - Ghosts cannot change direction unless they are facing a wall. The possible actions are east, west, south, and north (not stop). Initially, they have no direction and can move to any adjacent square
 - We use random ghosts which choose uniformly between legal moves
 - Assume that Pacman cannot stop
 - The game is scored as follows
 - -1 for each action Pacman takes
 - 10 for each food dot eaten
 - -500 for losing (if Pacman is eaten)
 - 500 for winning (all food dots eaten)

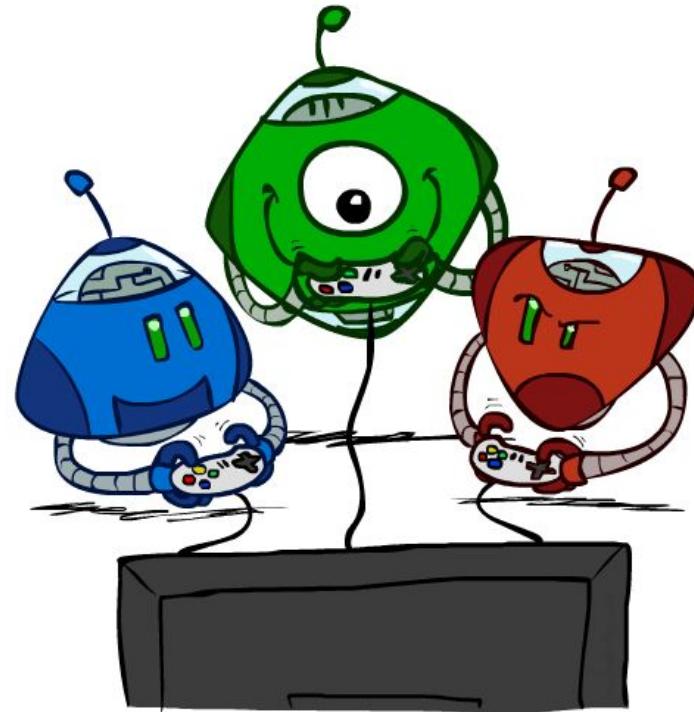


Expectiminimax Search

- Expectiminimax
 - Environment is an extra “random agent” player that moves after each min/max agent

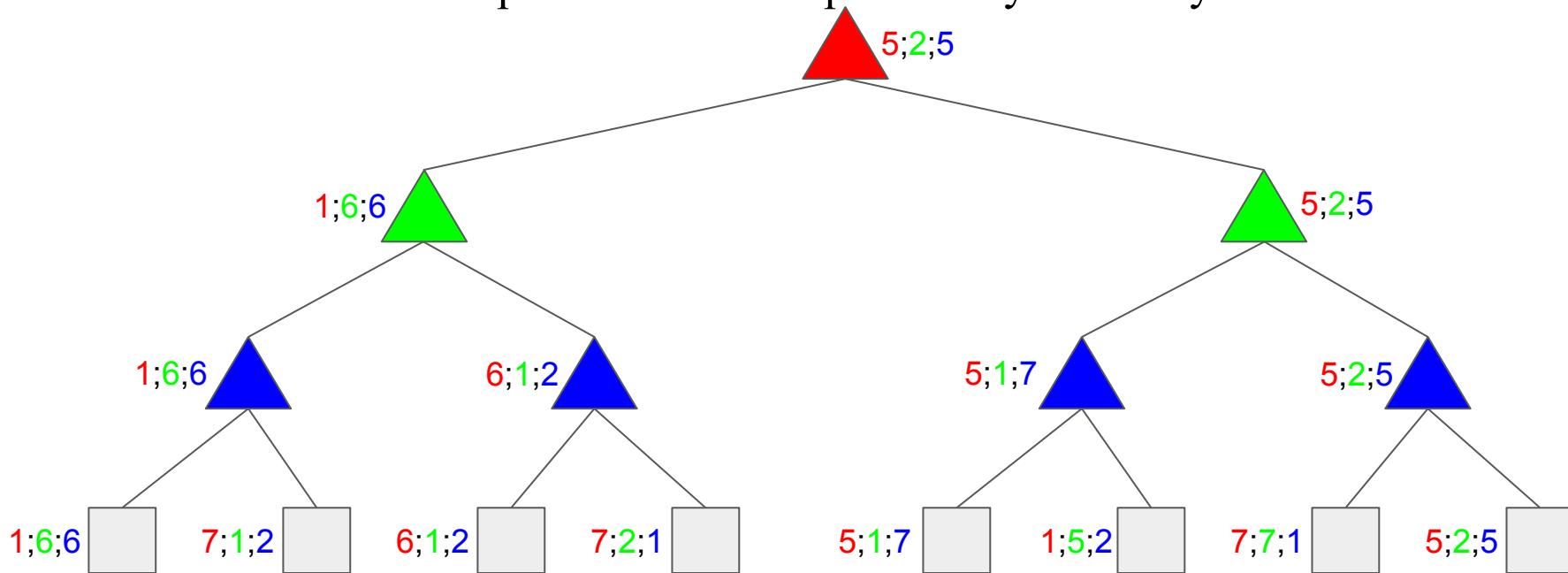


Multi-Agent Utilities

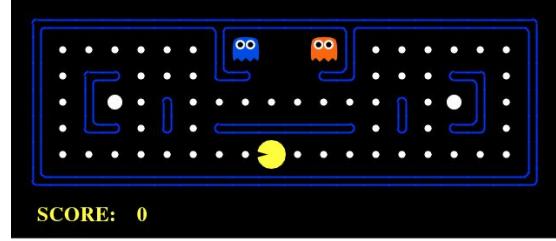


Multi-Agent Utilities

- Generalisation of minimax
 - Terminals and nodes have utility vectors
 - Each player maximizes its own component
 - Gives rise to cooperation and competition dynamically



<http://ai.berkeley.edu>



Assignment 1



Chapter 2

Markov Decision Processes

COMP 3270
Artificial Intelligence

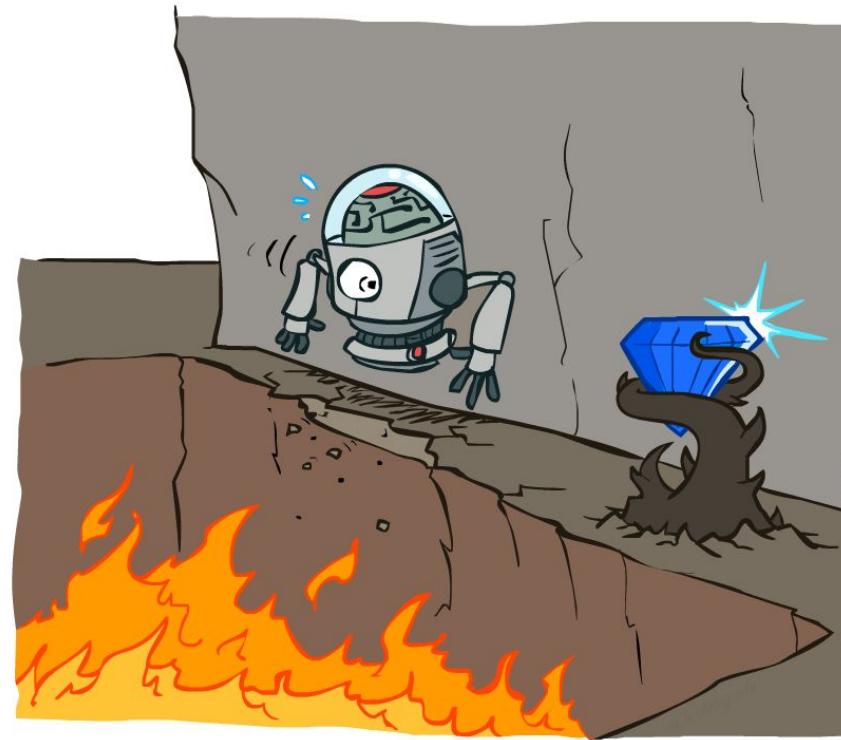
Dirk Schnieders

Sequential Decision Problem

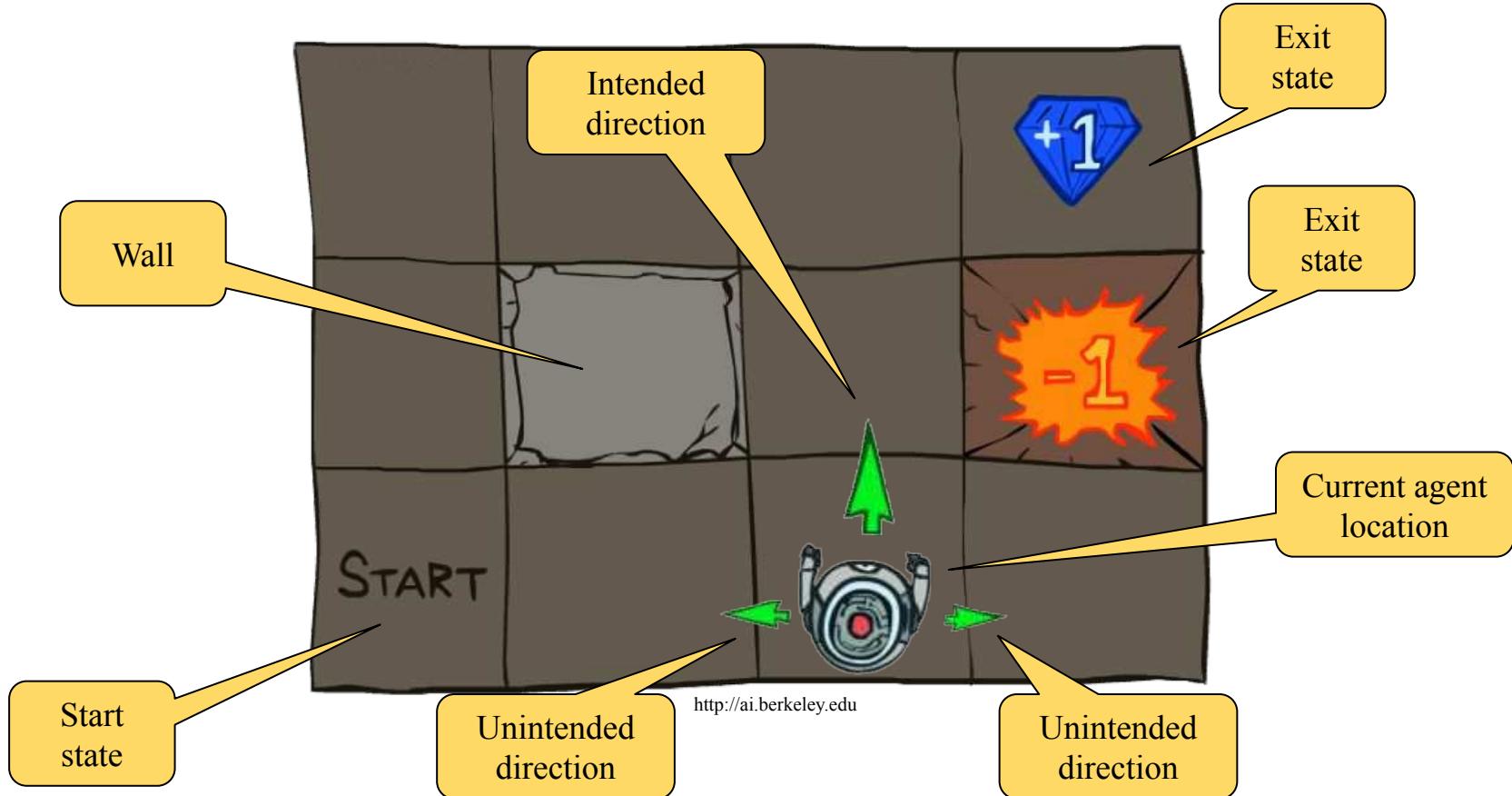
- In a sequential decision problem the agent's utility depends on a sequence of decisions
- Sequential decision problems incorporate utilities, uncertainty and sensing
- Optimal behavior balances the risks and rewards of acting in an uncertain environment



Non-Deterministic Search



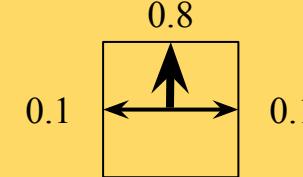
Running Example: Grid World



Grid World

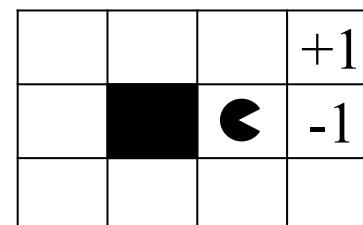
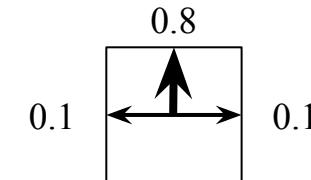
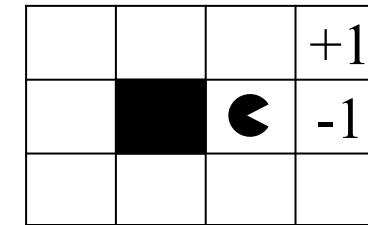
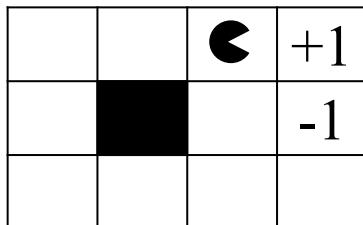
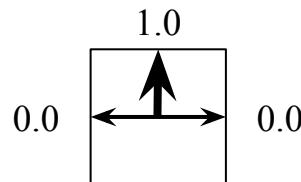
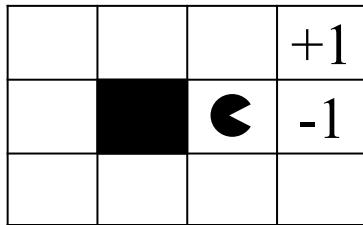
- Maze-like problem
 - Agent lives in a grid
 - Walls block the agent's path
- Unreliable actions
 - Each action achieves the intended effect 80% of the time
 - 10% of the time the action moves the agent at right angles (i.e., 90°) to the intended direction
 - If the agent bumps into a wall it stays in the same square
- The agent receives rewards for each action
 - Small “living” rewards (can be negative)
 - Big rewards come at the end (good or bad)
- Goal states
 - +1 and -1 are goal states
 - Only action available: exit action
- Goal: maximize sum of rewards

Example of stochastic motion:
Intended direction is north

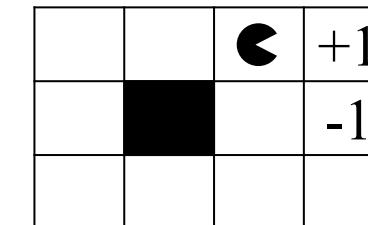


			+1
			-1

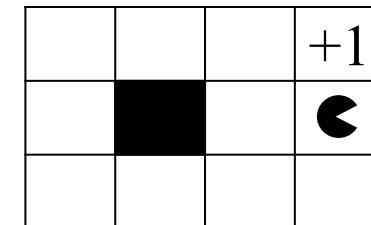
Deterministic vs. stochastic motion



0.1

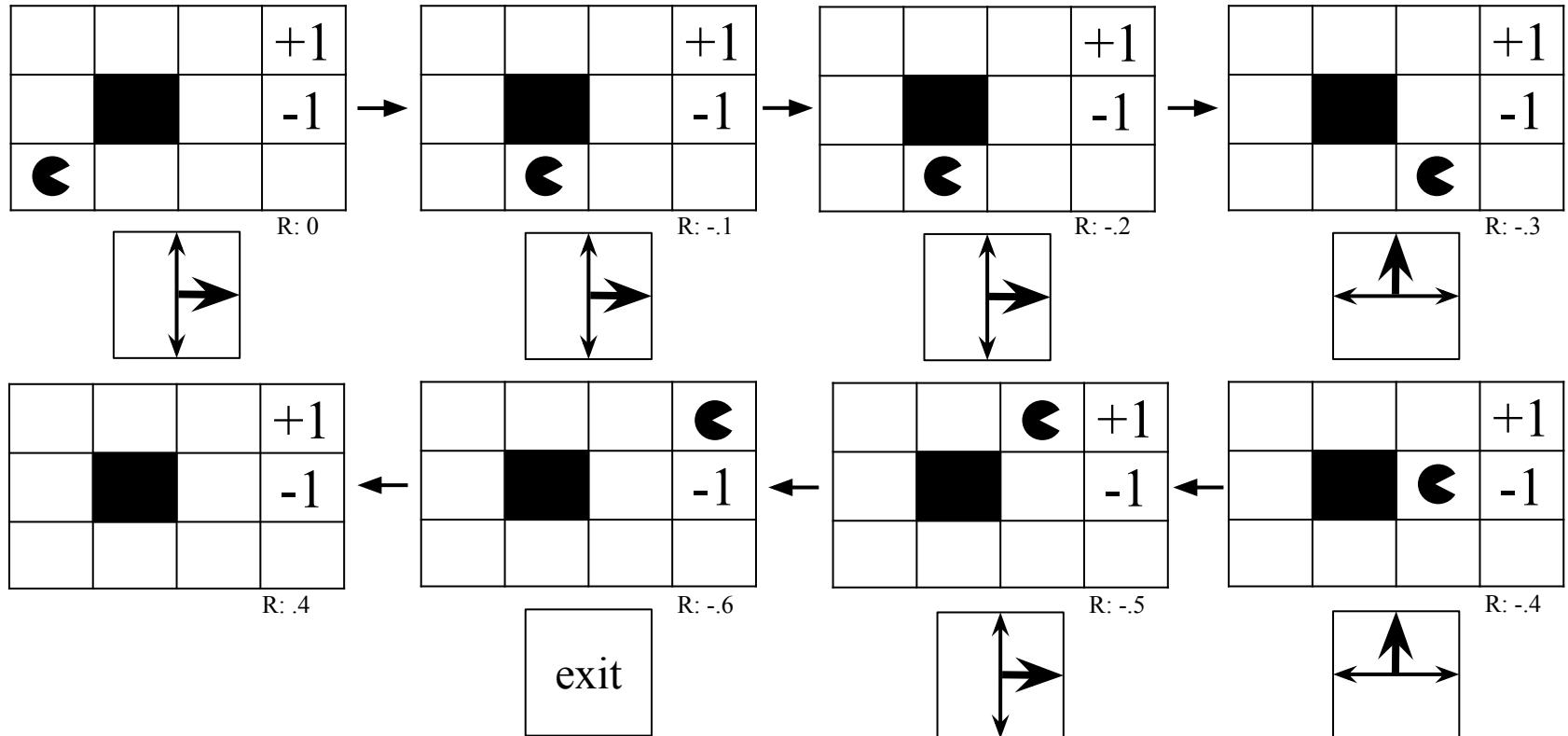


0.8



0.1

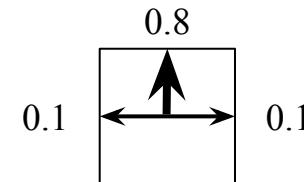
Example episode in grid world



Quiz - North North

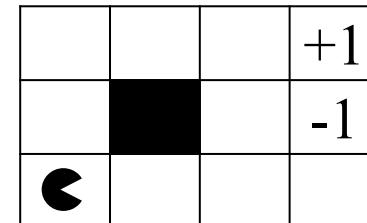
- Consider the fixed action sequence [North, North]
 - What is the probability of reaching the highlighted state from the start state with this action sequence?

			+1
	■	■	-1
●			



Quiz - East East

- Which squares can be reached from the start state by the action sequence [East, East] and with what probabilities?



Transition model

- The transition model $T(s,a,s')$ describes the outcome of each action in each state
- The outcome is stochastic and we write $P(s'|s,a)$ to denote the probability of reaching state s' if action a is done in state s
- We assume that transitions are Markovian
 - I.e., the probability of reaching s' from s depends only on s and not on the history of earlier states
 - The Markov property is named after the Russian mathematician Andrey Markov

[source](#)



Andrey Markov
(1856-1922)

Markov Decision Processes

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process (MDP)
- It is defined by
 - A set of states $s \in S$
 - A set of actions $a \in A$
 - A transition function $T(s,a,s')$
 - Probability that a from s leads to s' , i.e., $P(s'|s,a)$
 - A reward function $R(s,a,s')$
 - A start state s_0
 - A terminal state (optional)

How to solve MDPs?

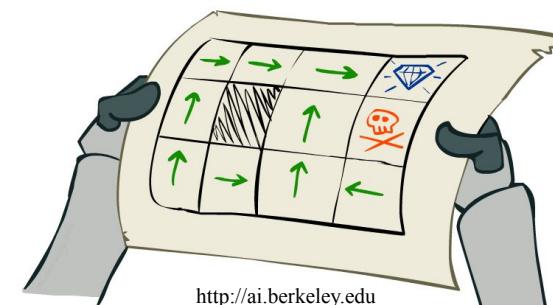
A fixed action sequence?

Policy

- How to solve MDPs?
 - A fixed action sequence won't solve the problem because the agent might end up in a state other than the goal
 - A solution must specify what the agent should do for any state that the agent might reach
 - A solution of this kind is called a policy π
 - $\pi(s)$ is the action recommended by the policy π for the state s



<http://ai.berkeley.edu>



<http://ai.berkeley.edu>

Optimal Policy

- Each time a given policy is executed, the stochastic nature of the environment may lead to a different environment history
- The quality of a policy is therefore measured by the expected utility of the possible environment histories generated by that policy
- An optimal policy π^* is a policy that yields the highest expected utility
- Example: π^* for Grid World with $R(s) = -0.04$ (\forall nonterminal s)

→	→	→	+1
↑		↑	-1
↑	←	←	↓

Optimal Policy - Example

→	→	→	+1
↑		←	-1
↑	←	←	↓

$$R(s) = -0.01$$

→	→	→	+1
↑		↑	-1
↑	←	←	←

$$R(s) = -0.03$$

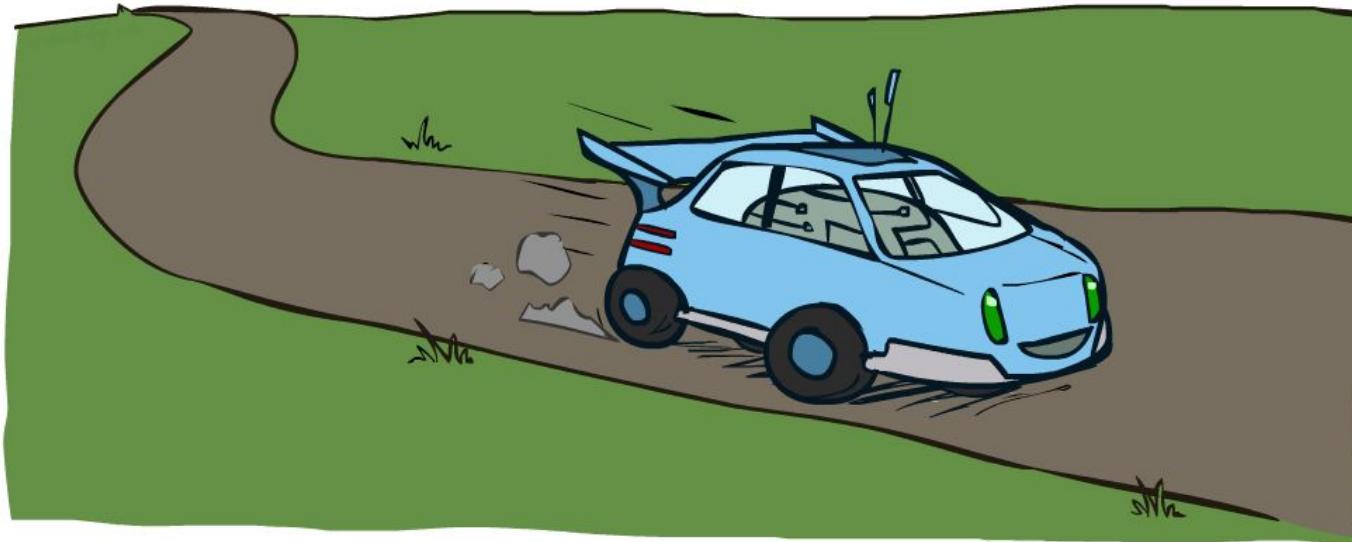
→	→	→	+1
↑		↑	-1
↑	→	↑	←

$$R(s) = -0.4$$

→	→	→	+1
↑		→	-1
→	→	→	↑

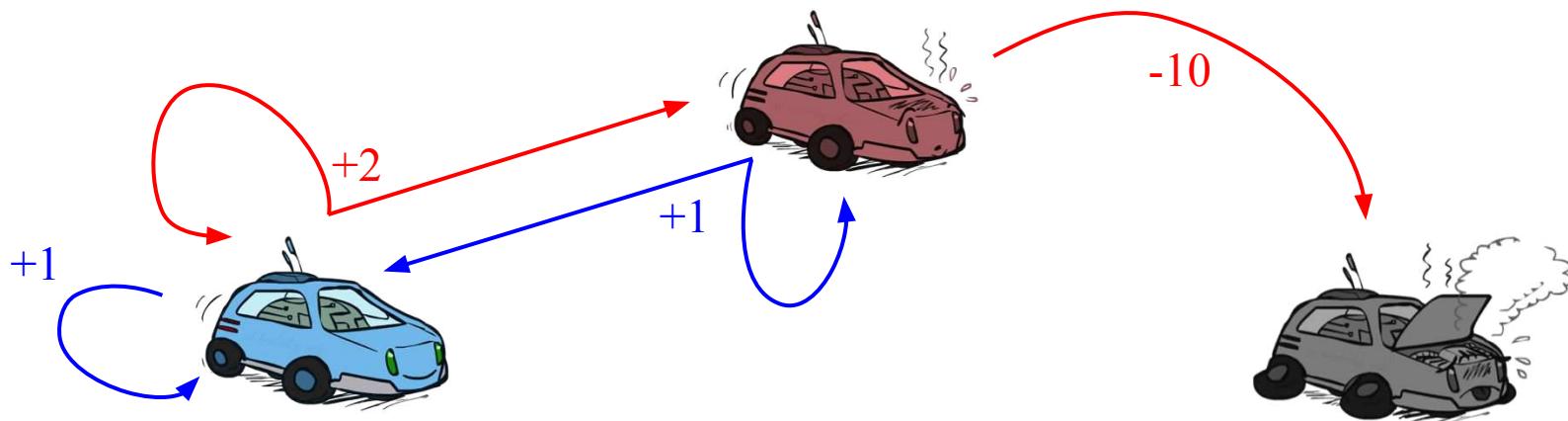
$$R(s) = -2.0$$

Example: Racing Car

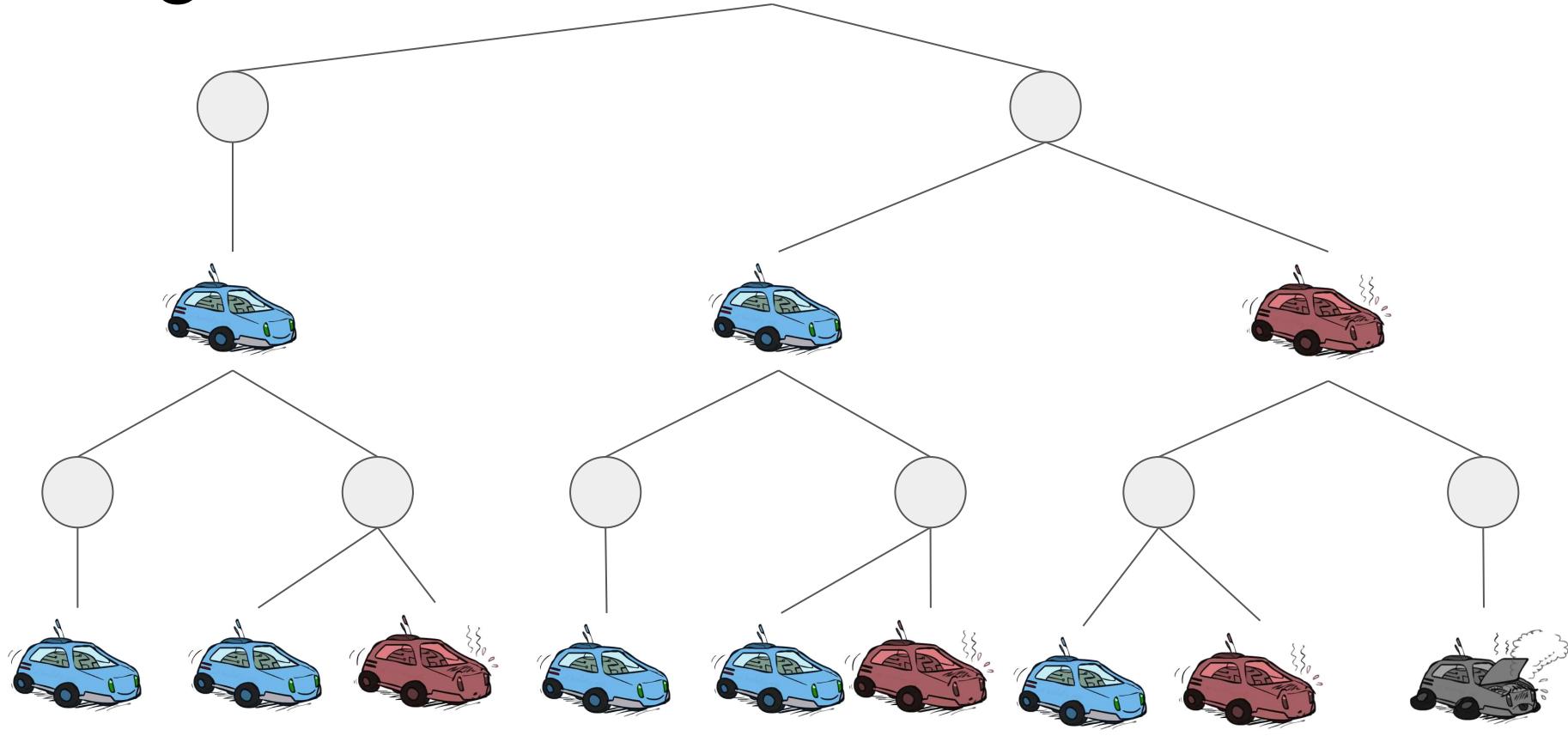


Example: Racing Car

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: Slow (+1), Fast (+2)

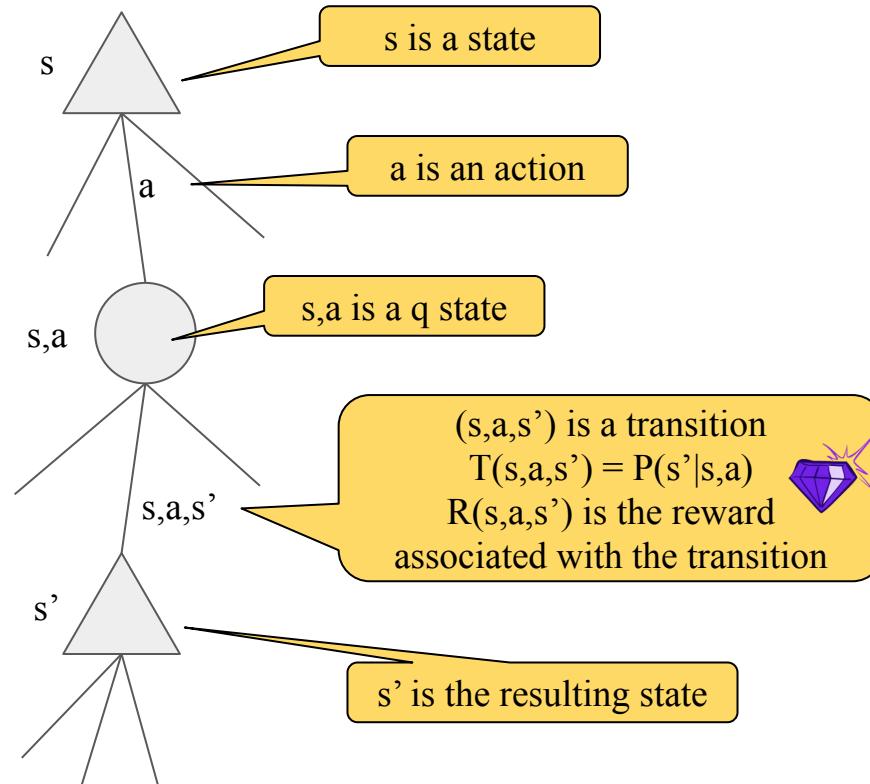


Racing Search Tree



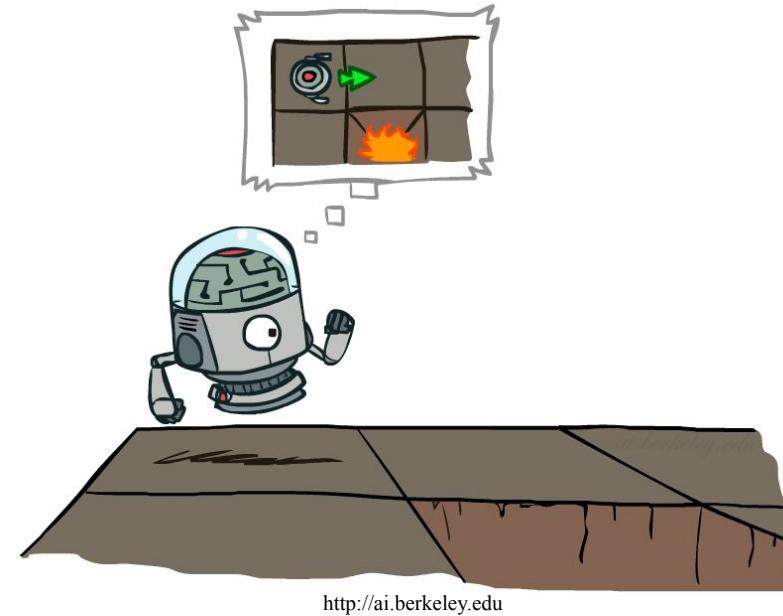
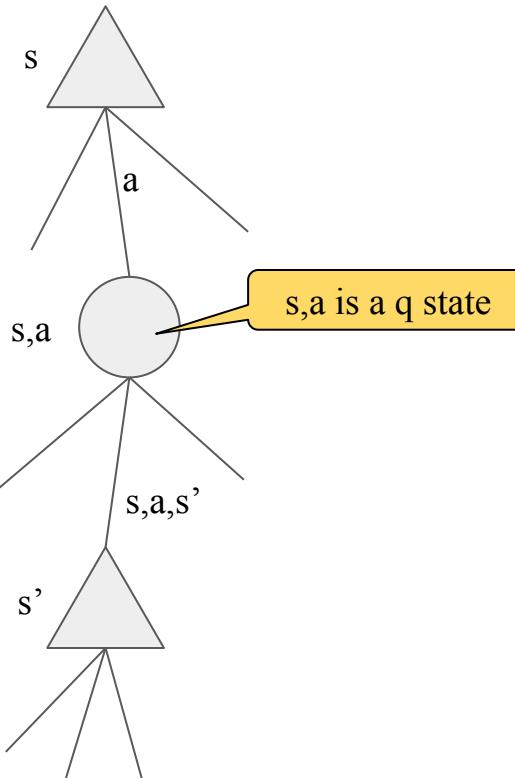
MDP Search Tree

- Each MDP state projects an expectimax-like search tree



Q state

- The agent has committed to the action but has not done it yet



10				1
a	b	c	d	e

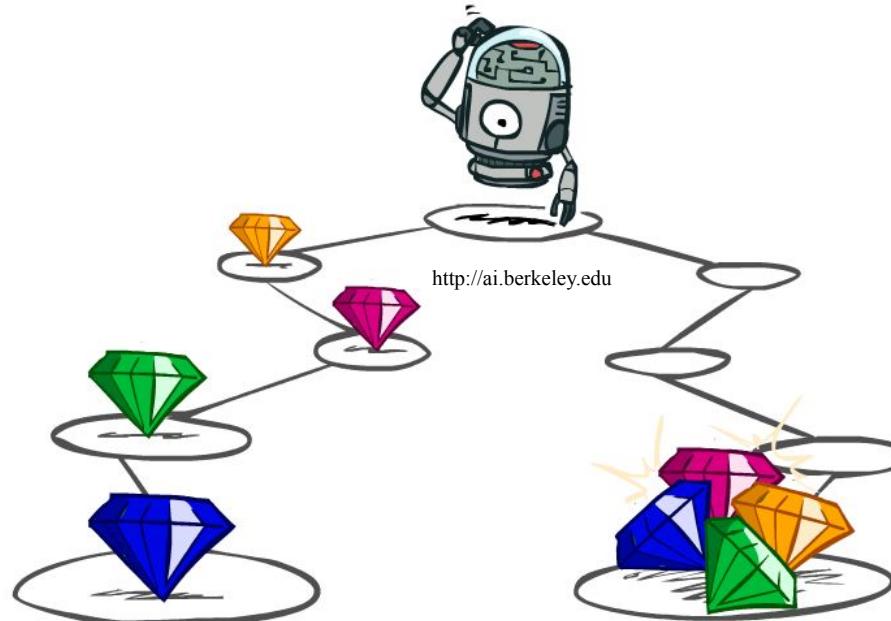
Quiz - Notation

- Consider a grid world MDP as shown above
- The available actions in each state are to move to the neighboring grid squares
- East and west actions are successful 80% of the time
- When not successful, the agent stays in place
- From state a, there is also an exit action available, which results in going to the terminal state and collecting a reward of 10
 - Similarly, in state e, the reward for the exit action is 1
- Exit actions are successful 100% of the time

Find the following quantities

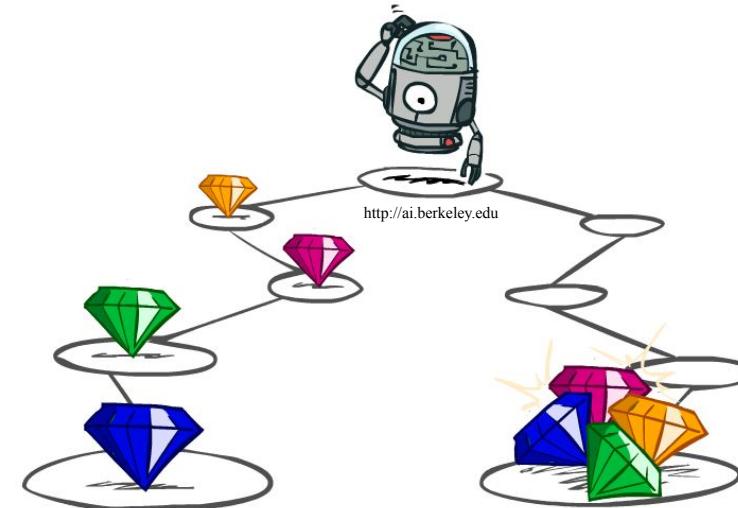
1. $T(c, \text{East}, d)$
2. $T(c, \text{East}, e)$
3. $T(c, \text{East}, c)$
4. $T(c, \text{East}, b)$
5. $T(c, \text{Exit}, a)$
6. $T(c, \text{East}, \text{terminal state})$
7. $T(a, \text{Exit}, \text{terminal state})$
8. $T(a, \text{East}, b)$
9. $T(a, \text{East}, a)$
10. $R(a, \text{East}, a)$
11. $R(a, \text{Exit}, \text{terminal state})$
12. $R(c, \text{East}, d)$

Utility of State Sequences



Utility of State Sequences

- How to calculate the utility of state sequences?
 - What preferences should an agent have over reward sequences?
- More or less?
 - [1, 2, 2] or [2, 3, 4]
- Now or later?
 - [0, 0, 1] or [1, 0, 0]



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution
 - Values of rewards decay exponentially



1

Worth Now



γ

Worth one step
from now



γ^2

Worth two step
from now

$$1 < \gamma < 0$$

Quiz - [1,2,3] vs. [3,2,1]

- Consider the sequences [1,2,3] and [3,2,1]
- Let $\gamma = 0.5$, which sequence has a higher utility?

Quiz - Optimal Action 1

- Consider the same grid world MDP as in one of the previous quiz
- Let actions always be successful
- Let the discount factor be $\gamma = 0.1$
 - Q1: What is the optimal action in the state d ?
- Now let the discount factor be $\gamma = 0.9999$
 - Q2: What is the optimal action in the state d ?

10				1
a	b	c	d	e

Quiz - Optimal Action 2

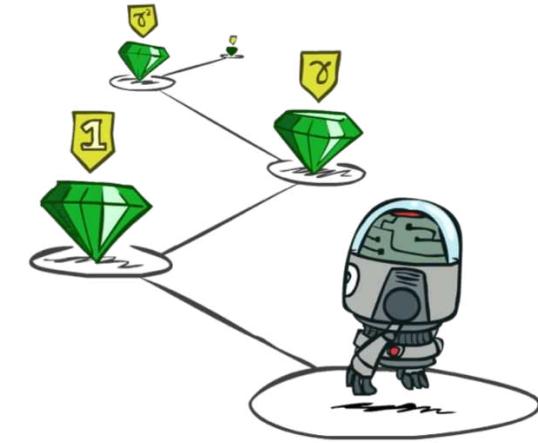
- Consider the following 101×3 world

+50	-1	-1	-1	...	-1	-1	-1	-1	-1
<i>Start</i>					...				
-50	+1	+1	+1	...	+1	+1	+1	+1	+1

- In the start state, the agent has a choice of two deterministic actions up or down but in the other states the agent has one deterministic action
- Assuming a discounted reward function, for what values of the discount γ should the agent choose up and for which down?
- Compute the utility of each action as a function of γ

Stationary Preferences

- We assume an agent's preferences between state sequences are stationary
- If two state sequences begin with the same state r , then the two sequences should be preference-ordered the same way as the sequences without r



<http://ai.berkeley.edu>

$$\begin{array}{c} [r, s_1, s_2, \dots] \succ [r, s'_1, s'_2, \dots] \\ \Updownarrow \\ [s_1, s_2, \dots] \succ [s'_1, s'_2, \dots] \end{array}$$

Stationary Preferences

- Stationarity has strong consequences
- It turns out that there are just two coherent ways to assign utilities to sequences
- Additive rewards

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

- Discounted rewards

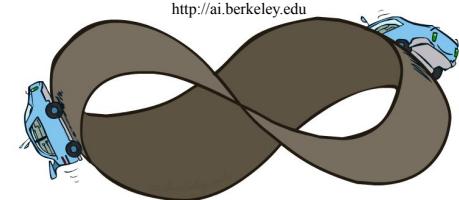
$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

Quiz - Stationary Preference

- Suppose that we define the utility of a state sequence to be the maximum reward obtained in any state in the sequence
- Does this utility function result in stationary preference between state sequences?

Infinity Utilities?

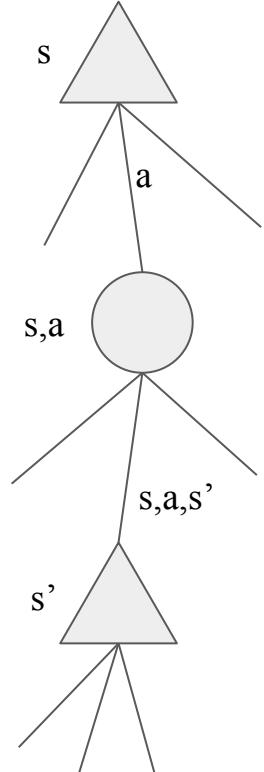
- What if the environment does not contain a terminal state or if the agent never reaches one?
- Utilities with additive undiscounted rewards will be infinite
 - E.g., Race game
 - The smart race car will never overheat
- With discounted rewards, the utility of an infinite sequence is finite



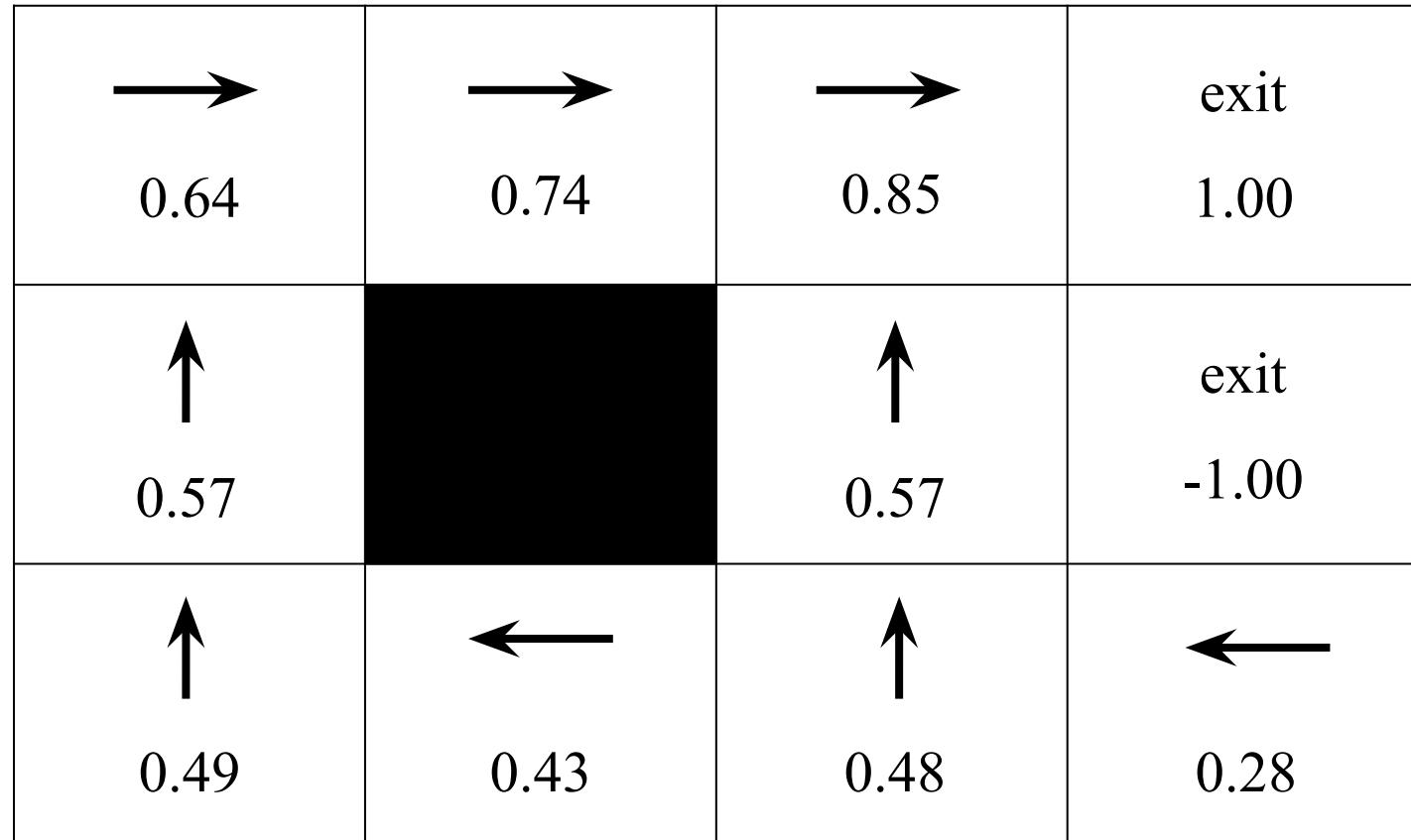
$$U([s_0, s_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1 - \gamma)$$

Optimal Quantities

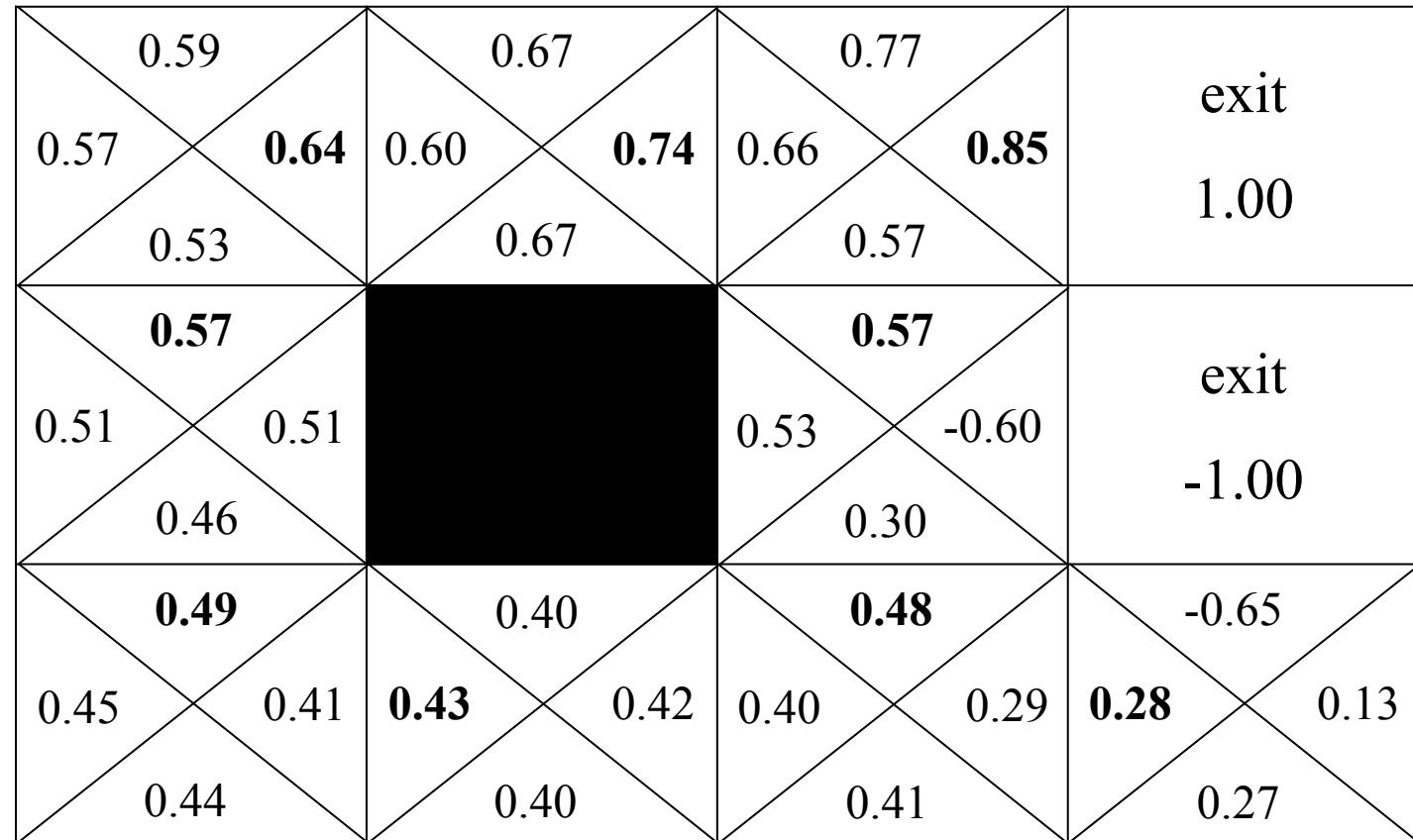
- $V^*(s)$ is the value (utility) of a state s
 - Expected utility starting in s and acting optimally
- $Q^*(s,a)$ is the value (utility) of a q-state (s,a)
 - Expected utility for having taken action a from state s and thereafter acting optimally
- $\pi^*(s)$ is the optimal policy for state s
 - I.e., optimal action from state s



Example - π^* and V^*



Example - Q^*



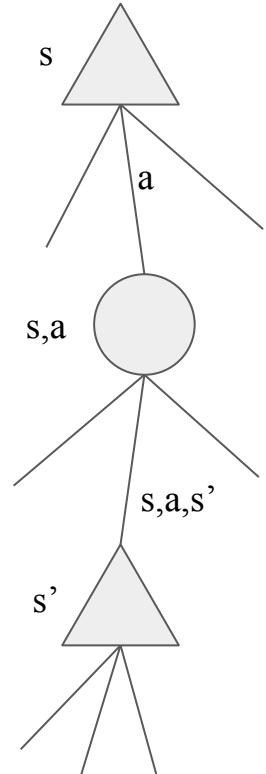
Bellman Equation

- How to compute the value of a state?
 - Average sum of discounted action
 - Very similar to expectimax
 - Recursive definition

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$



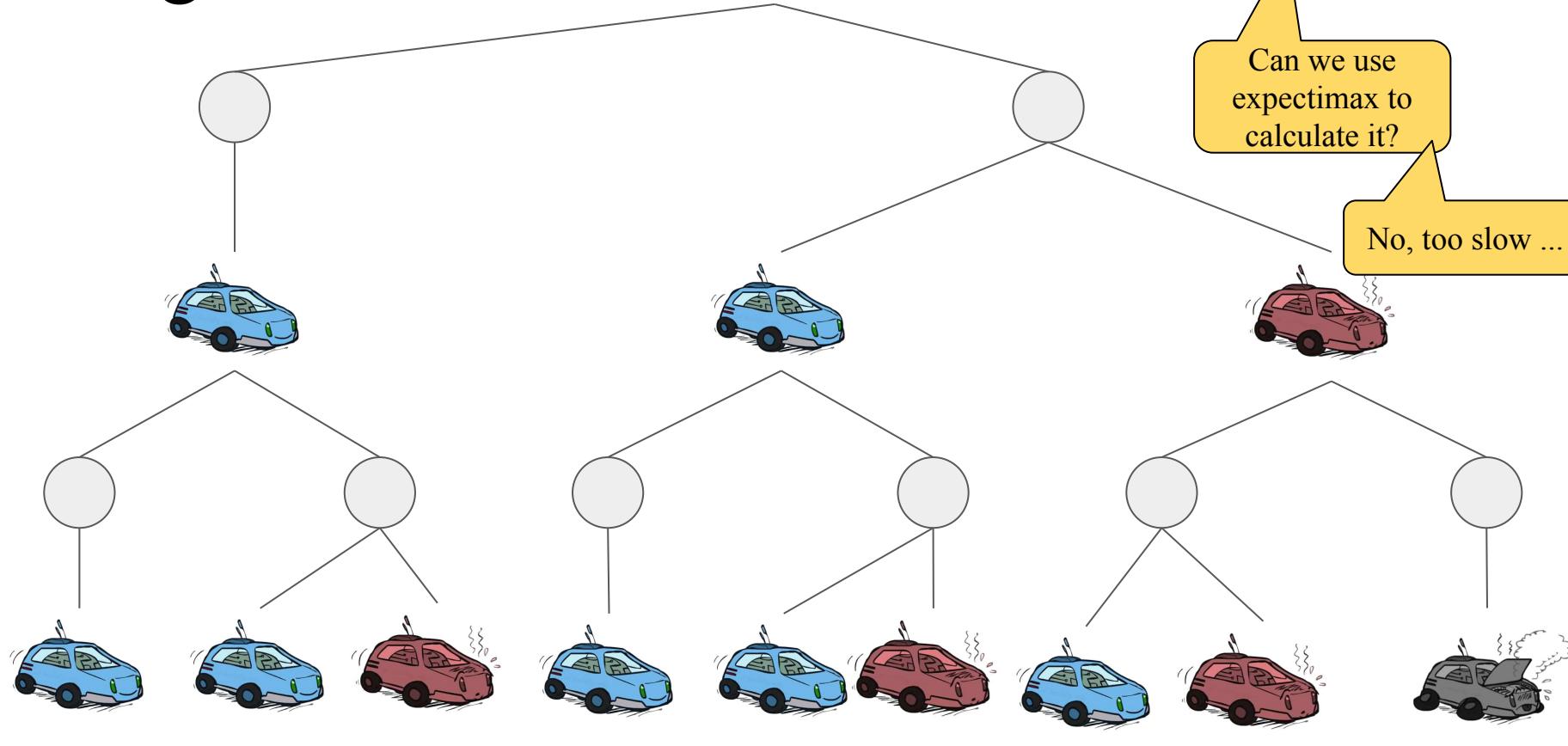
Racing Search Tree



What is the value of
this state? ↗

Can we use expectimax to calculate it?

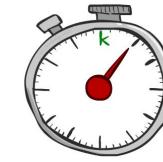
No, too slow ...



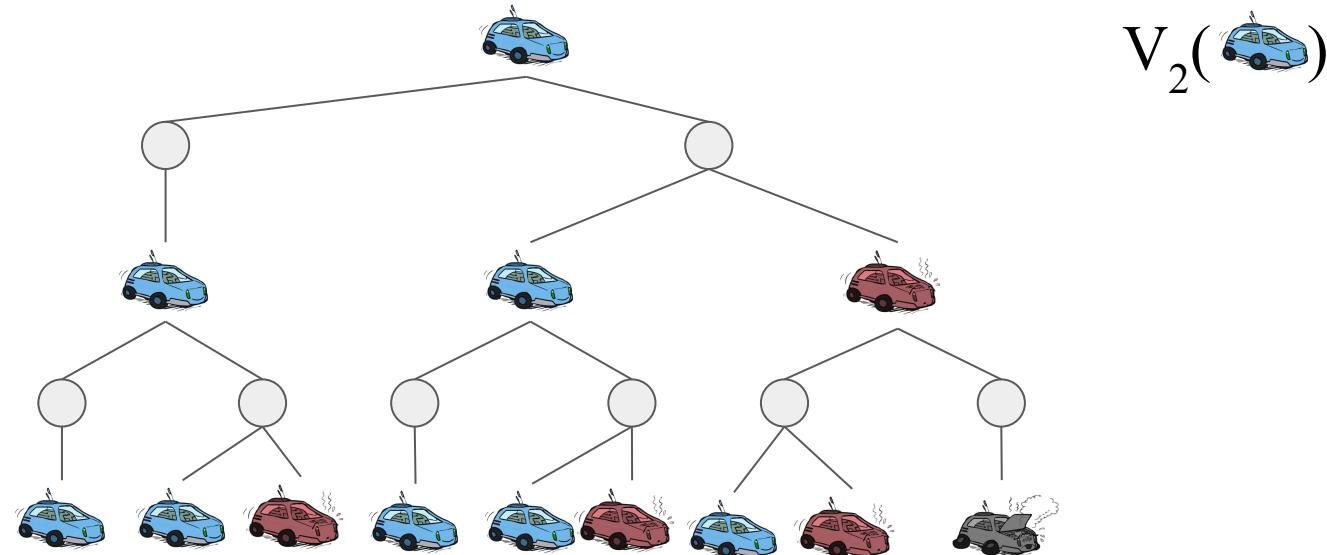
Racing Search Tree

- Problems
 - States are repeated (even at same depth)
 - Tree goes on forever
- Idea
 - Do a depth-limited computation, but with increasing depths until change is small
 - Note: Deep parts of the tree eventually don't matter if $\gamma < 1$
- Solution
 - Only compute needed quantities once
 - Do a depth-limited computation, but with increasing depths until change is small

Time-Limited Values



- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps (k more rewards)
- Equivalently, it's what a depth- k expectimax would give from s



Example - V_0

0.00	0.00	0.00	0.00
0.00		0.00	0.00
$R(s) = 0.00$ $\gamma = 0.99$	0.00	0.00	0.00

Example - V_1

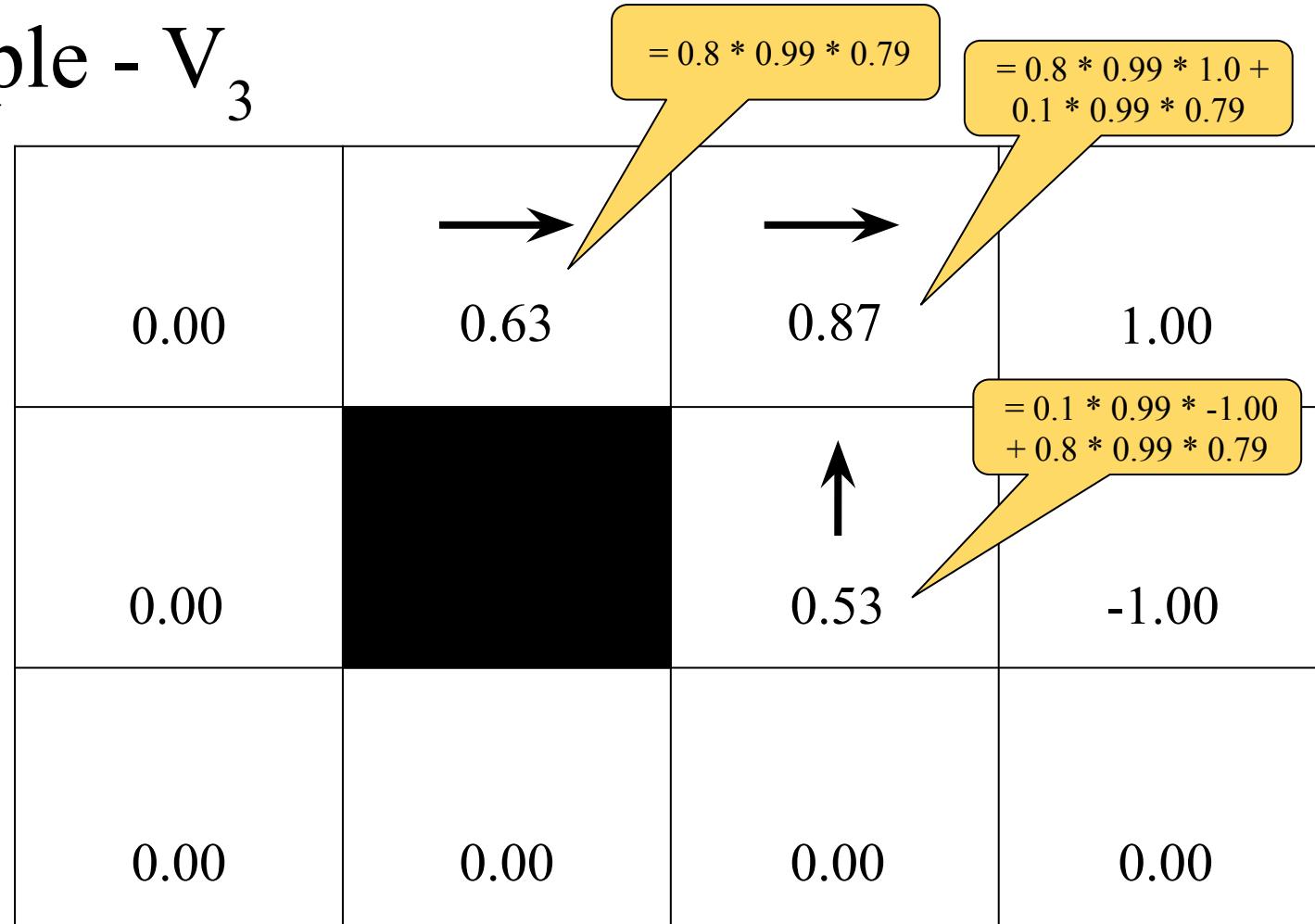
0.00	0.00	0.00	1.00
0.00		0.00	-1.00
$R(s) = 0.00$ $\gamma = 0.99$	0.00	0.00	0.00

Example - V_2

$$= 0.8 * 0.99 * 1.00$$

0.00	0.00	0.79	1.00
0.00	0.00	0.00	-1.00
R(s) = 0.00 $\gamma = 0.99$	0.00	0.00	0.00

Example - V_3



Example - V₄

\rightarrow 0.50	\rightarrow 0.81	\rightarrow 0.93	
		\uparrow 0.64	-1.00
$R(s) = 0.00$ $\gamma = 0.99$	0.00	0.00	0.42 0.00

Example - V₅

\rightarrow 0.69	\rightarrow 0.90	\rightarrow 0.95	1.00
\uparrow 0.39		\uparrow 0.70	-1.00
$R(s) = 0.00$ $\gamma = 0.99$	0.00	0.33	0.23

Example - V₆

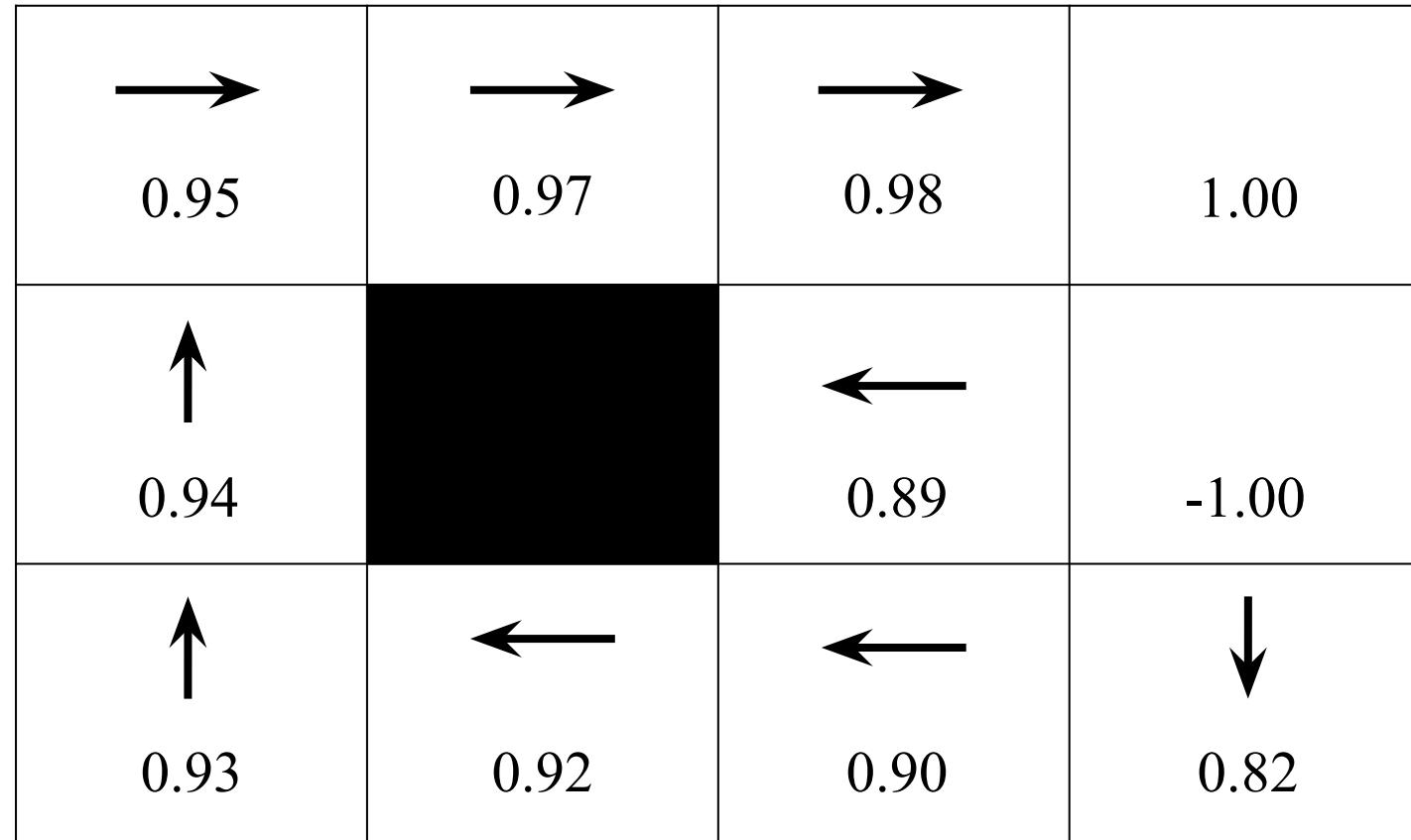
\rightarrow 0.82	\rightarrow 0.93	\rightarrow 0.96	1.00
\uparrow 0.63		\uparrow 0.72	-1.00
\uparrow 0.34	\rightarrow 0.47	\uparrow 0.61	\leftarrow 0.33

$$\begin{aligned} R(s) &= 0.00 \\ \gamma &= 0.99 \end{aligned}$$

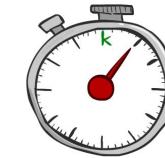
Example - V₇

\rightarrow 0.88	\rightarrow 0.94	\rightarrow 0.96	1.00
\uparrow 0.77		\uparrow 0.73	-1.00
\uparrow 0.58 $R(s) = 0.00$ $\gamma = 0.99$	\rightarrow 0.58	\uparrow 0.65	\leftarrow 0.42

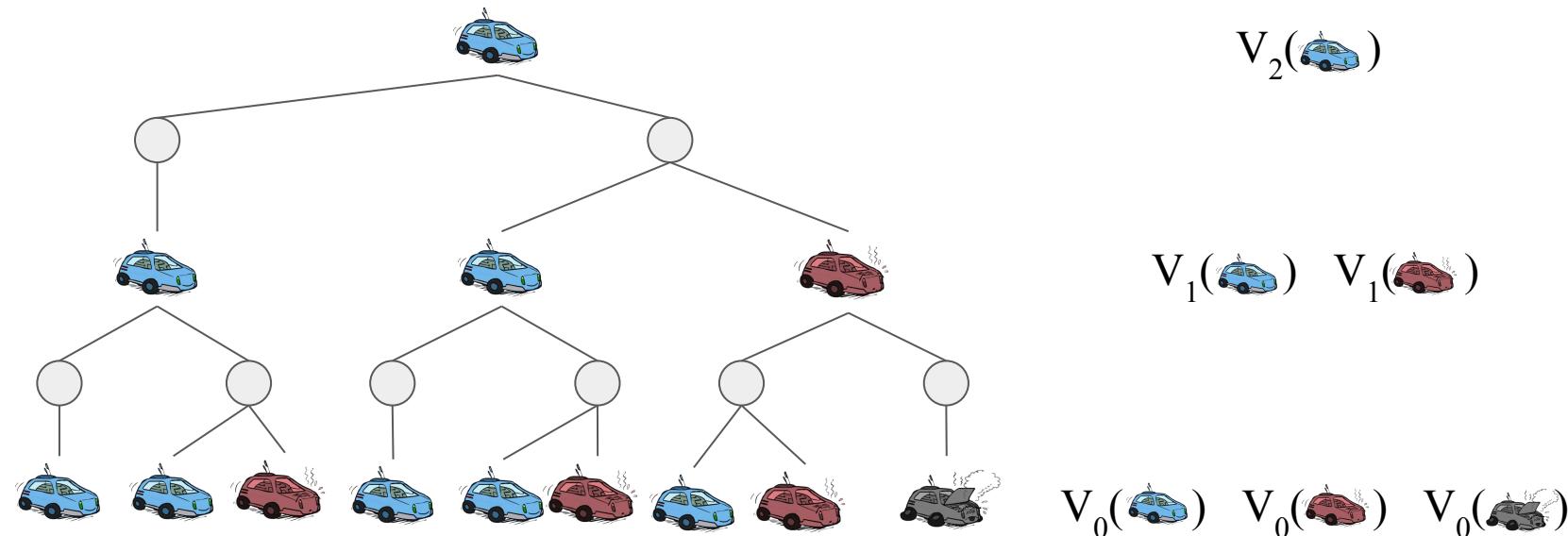
Example - V₁₀₀



Computing Time-Limited Values

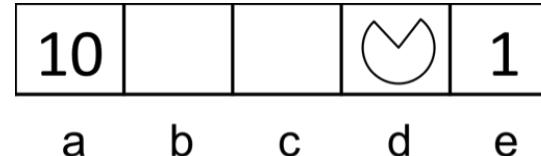


- We can save a lot of computation
- Example:
 - At every layer we have to compute at most 3 time limited values

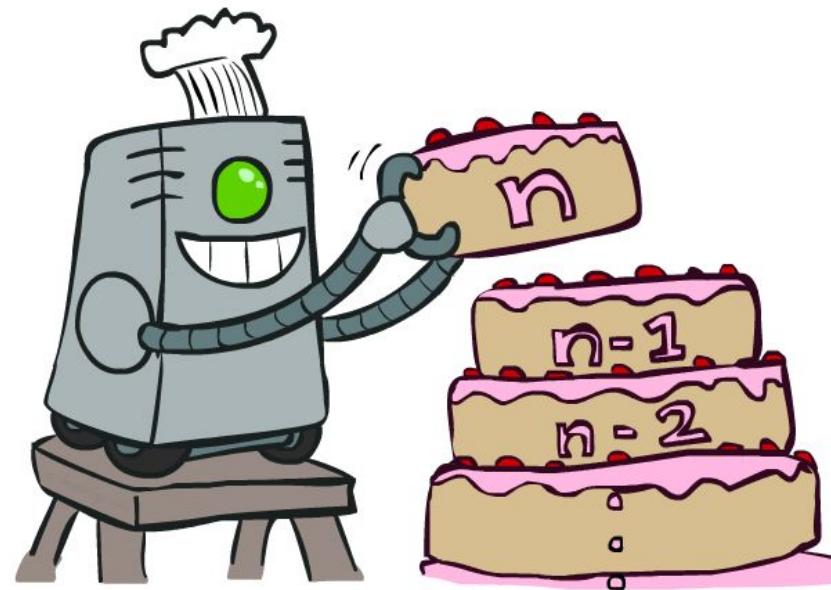


Quiz - Time-Limited Values

- Consider the same grid world MDP as in the previous quiz
 - Actions are successful 100% of the time
 - $\gamma = 1$
- Determine in the following quantities
 - $V_0(d)$
 - $V_1(d)$
 - $V_2(d)$
 - $V_3(d)$
 - $V_4(d)$
 - $V_5(d)$



Value Iteration



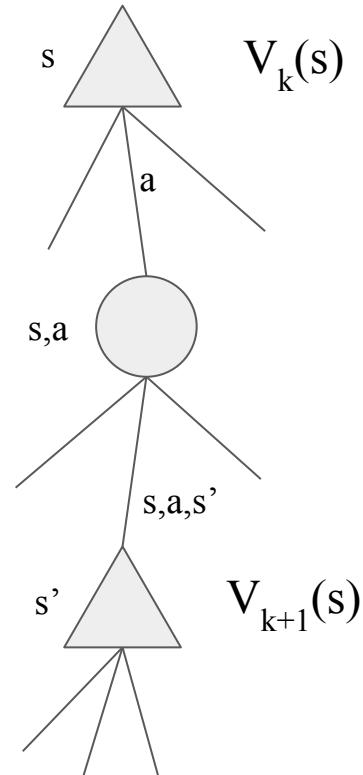
<http://ai.berkeley.edu>

Value Iteration

- Start with $V_0 = 0$
 - No time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one round of expectimax

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

- Repeat until convergence
- Complexity
 - $O(S^2A)$
- Theorem
 - Converges to unique optimal values



Quiz - Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$
$$\gamma = 1.0$$

$$V_0(\text{blue car}) =$$

$$V_1(\text{blue car}) =$$

$$V_2(\text{blue car}) =$$

$$V_0(\text{red car}) =$$

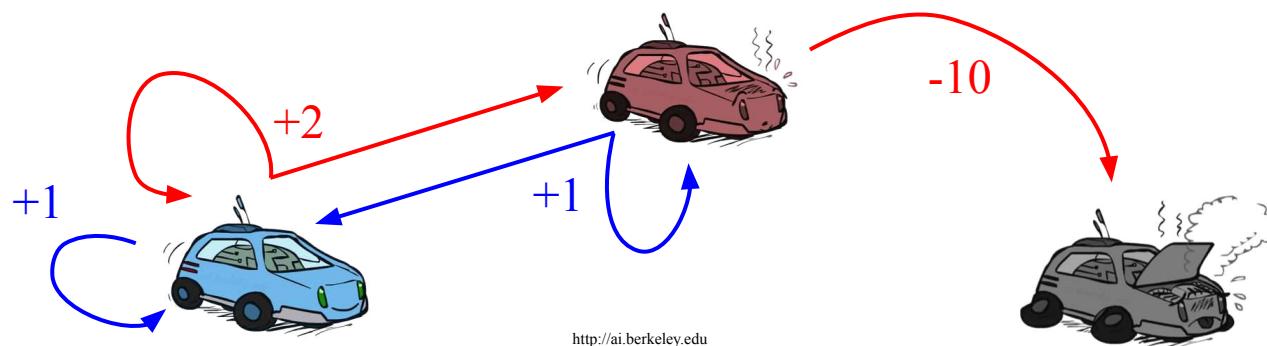
$$V_1(\text{red car}) =$$

$$V_2(\text{red car}) =$$

$$V_0(\text{crashed car}) =$$

$$V_1(\text{crashed car}) =$$

$$V_2(\text{crashed car}) =$$



Quiz - V_∞

- Consider the same grid world as in the previous quiz (where east and west actions are successful 100% of the time)
- $\gamma = 0.2$
- Determine the following quantities
 - $V^*(a) = V_\infty(a) = ?$
 - $V^*(b) = V_\infty(b) = ?$
 - $V^*(c) = V_\infty(c) = ?$
 - $V^*(d) = V_\infty(d) = ?$
 - $V^*(e) = V_\infty(e) = ?$

10				1
a	b	c	d	e

Quiz - Convergence

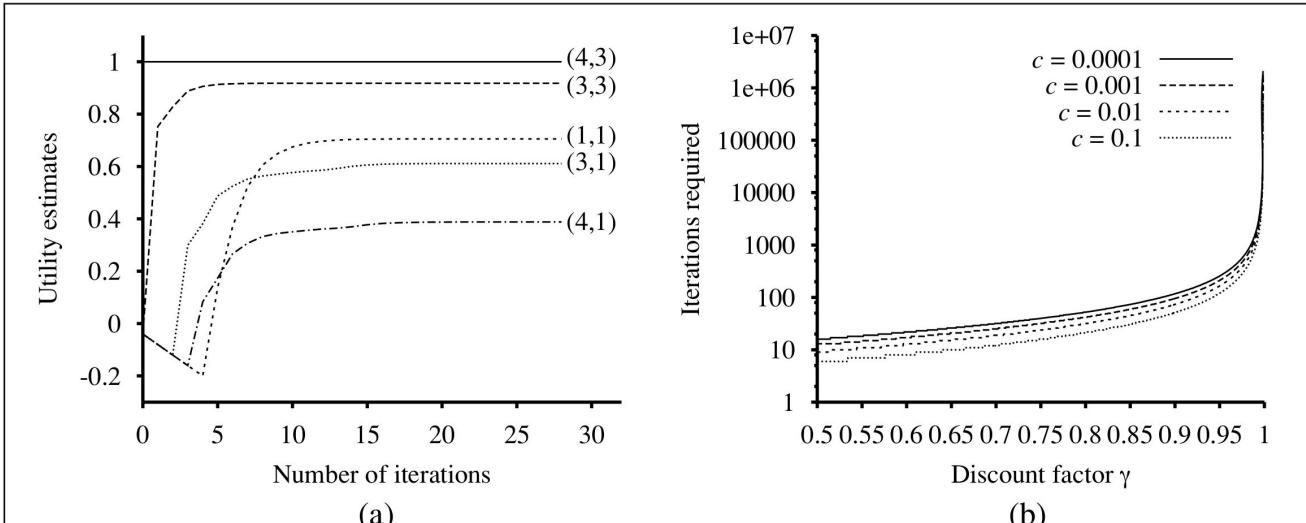
- Consider the following example where $\gamma = 1.0$ and $R(s) = -0.04$
 - Show that the value at (3,3) has converged

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

1 2 3 4

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

Evolution of Utilities / # Iterations vs. Gamma



3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c , as a function of the discount factor γ .

Quiz - $V_{k+1}(B)$

- Consider the following transition and reward functions for an MDP with $\gamma = 0.5$
- Suppose that after iteration k of value iteration we end up with the following values for V_k
 - $V_k(A) = 1.7, V_k(B) = 1.82, V_k(C) = 1.22$
 - What is $V_{k+1}(B)$?
- Now, suppose that we ran value iteration to completion and found the following value function
 - $V^*(A) = 2.208, V^*(B) = 2.416, V^*(C) = 1.766$
- What is $Q^*(B, \text{CW})$?
- What is $Q^*(B, \text{CCW})$?
- What is the optimal action from state B ?

s	a	s'	T(s,a,s')	R(s,a,s')
A	CW	B	1.0	1.0
A	CCW	B	0.4	-2.0
A	CCW	C	0.6	-1.0
B	CW	A	0.6	1.0
B	CW	C	0.4	2.0
B	CCW	A	0.2	1.0
B	CCW	C	0.8	-1.0
C	CW	A	0.6	-2.0
C	CW	B	0.4	1.0
C	CCW	A	0.4	0.0
C	CCW	B	0.6	1.0

Quiz

- Consider the transition and reward function shown in the table for an MDP that has two states and two actions
- Let $\gamma = 1.0$
- Determine the following
 - $V_0(A), V_0(B)$
 - $Q_1(A,0), Q_1(A,1), Q_1(B,0), Q_1(B,1)$
 - $V_1(A), V_1(B)$
 - $Q_2(A,0), Q_2(A,1), Q_2(B,0), Q_2(B,1)$

s	a	s'	T(s,a,s')	R(s,a,s')
A	0	A	0.5	2
A	0	B	0.5	-1
A	1	A	0.5	1
A	1	B	0.5	2
B	0	A	0.0	-2
B	0	B	1.0	-1
B	1	A	0.1	-3
B	1	B	0.9	-1

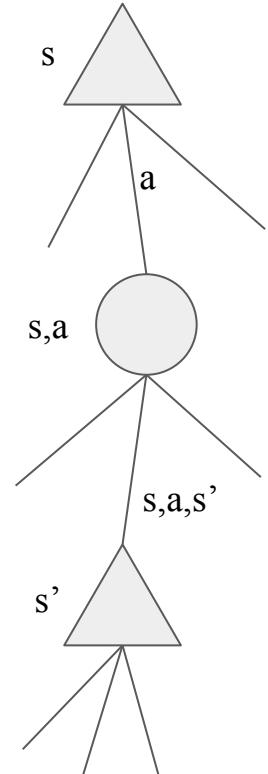
Bellman Equation vs. Value Iteration

- Bellman equations characterize the optimal values

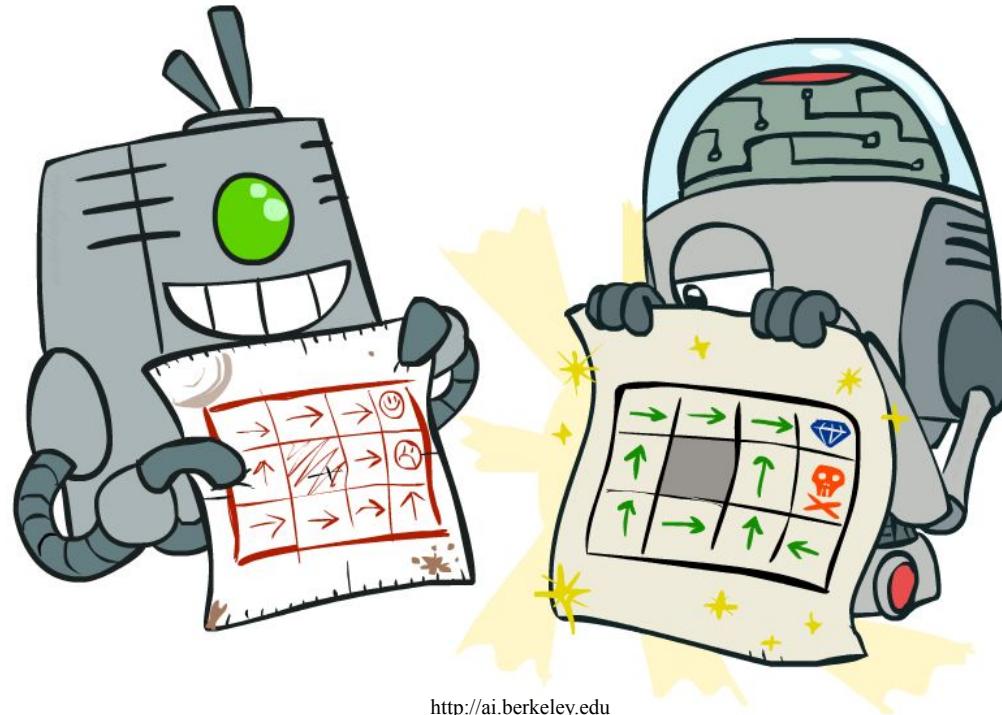
$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

- Value iteration computes them

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

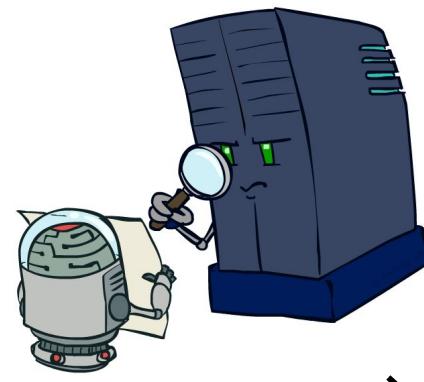


Policy Methods

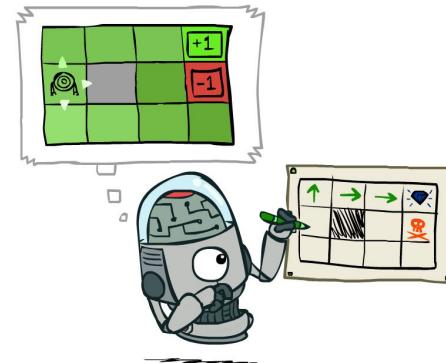


<http://ai.berkeley.edu>

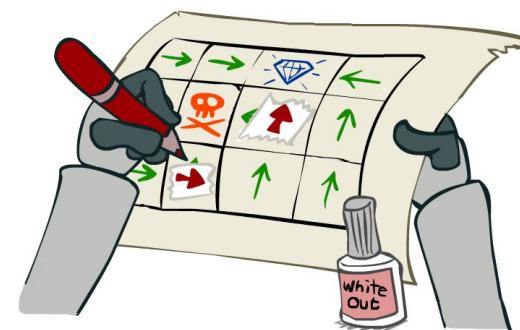
Policy Evaluation



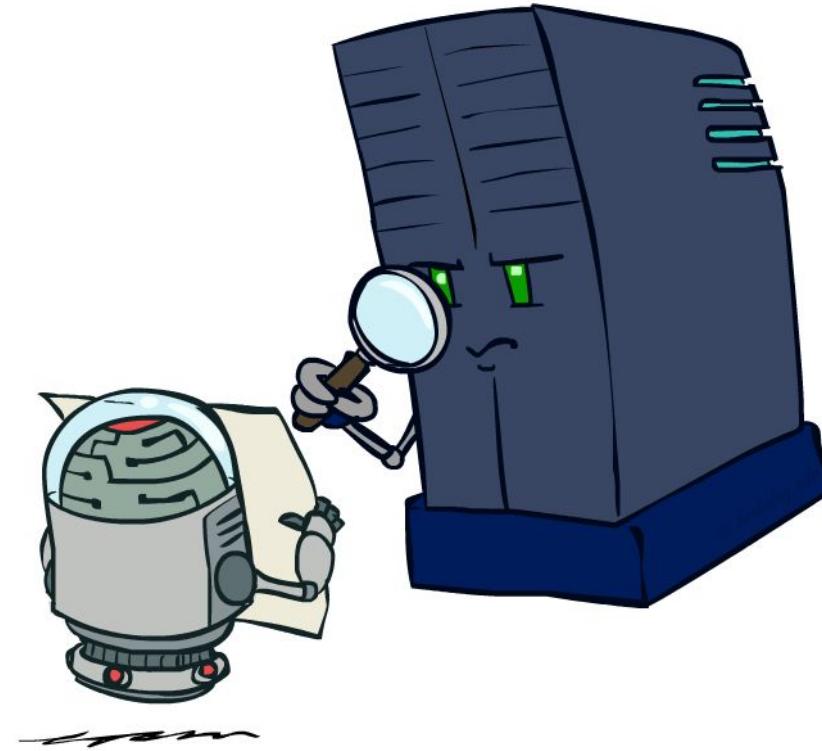
Policy Extraction



Policy Iteration



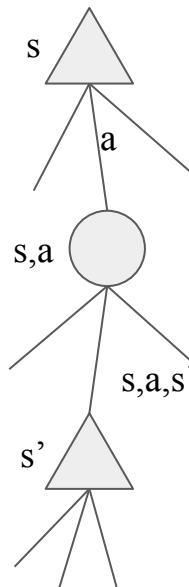
Policy Evaluation



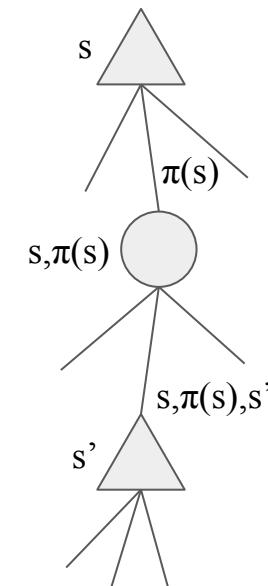
Policy Evaluation

- We would like to determine how good a given policy π is
 - How well will I perform if I follow π ?

Do the optimal action



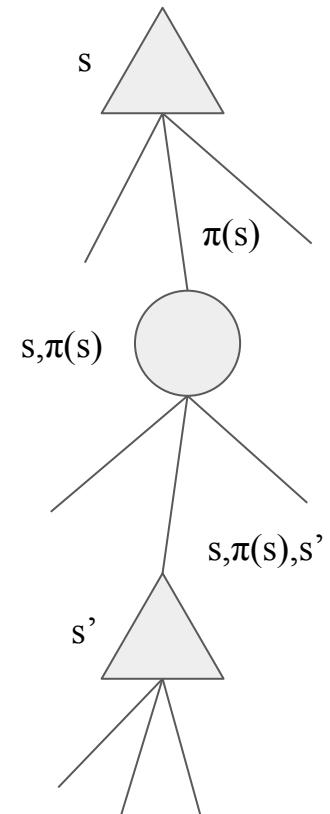
Do what π says



Utilities for a Fixed Policy

- We compute the utility of a state s under a fixed (generally non-optimal) policy
 - $V^\pi(s)$ = expected total discounted rewards starting in s and following π

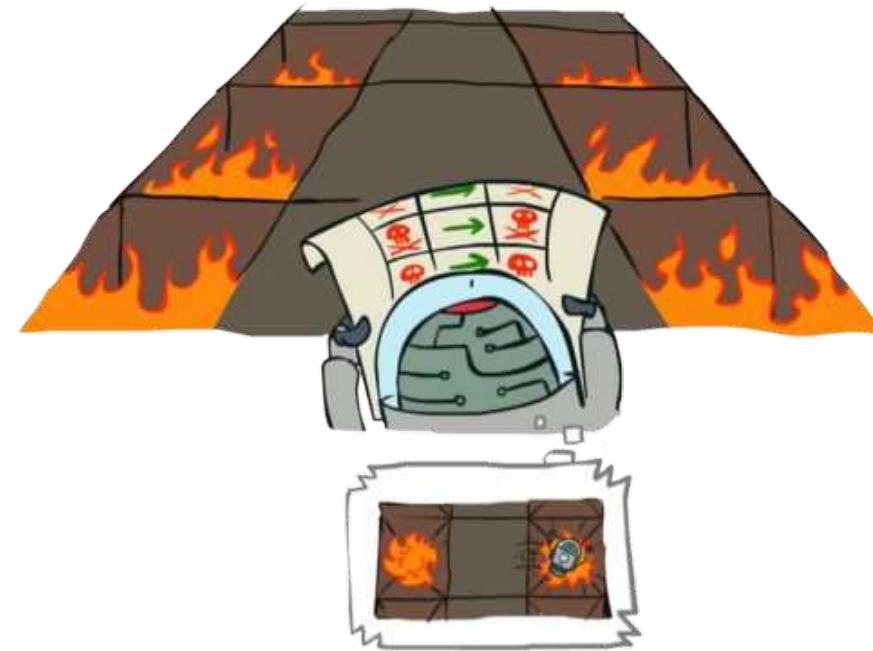
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$



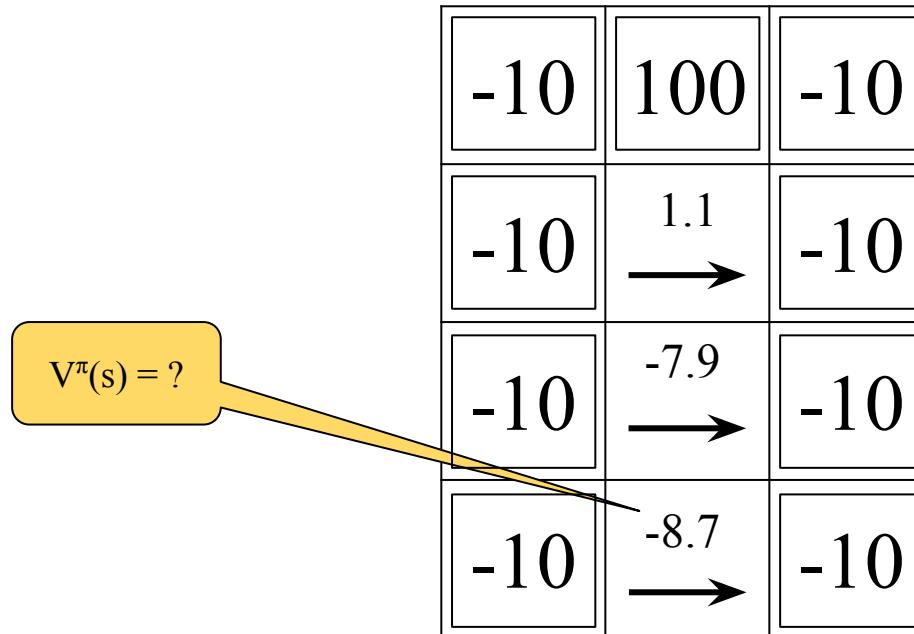
New Example

-10	100	-10
-10		-10
-10		-10
-10		-10

Example: Policy Evaluation

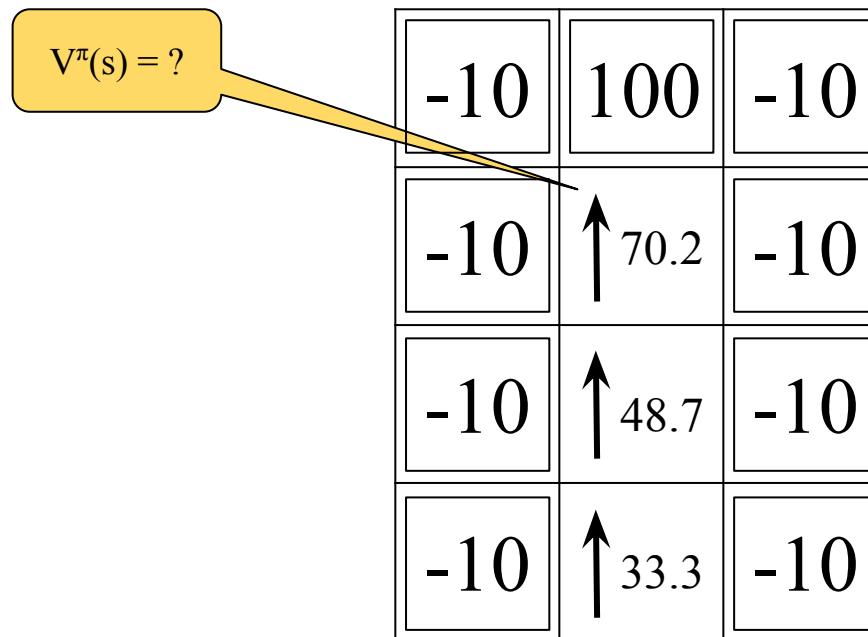


Quiz - Go East



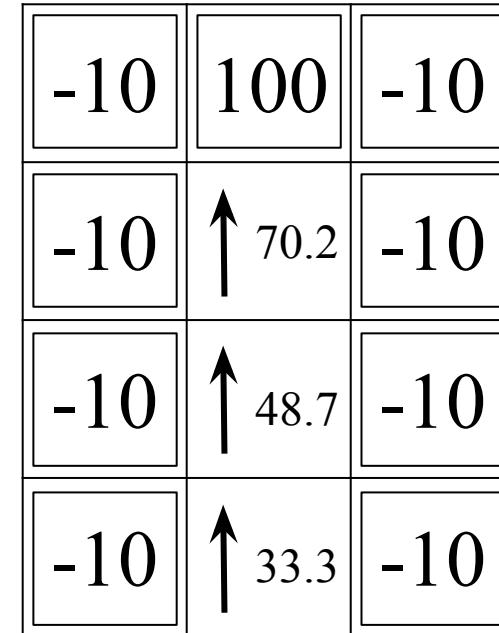
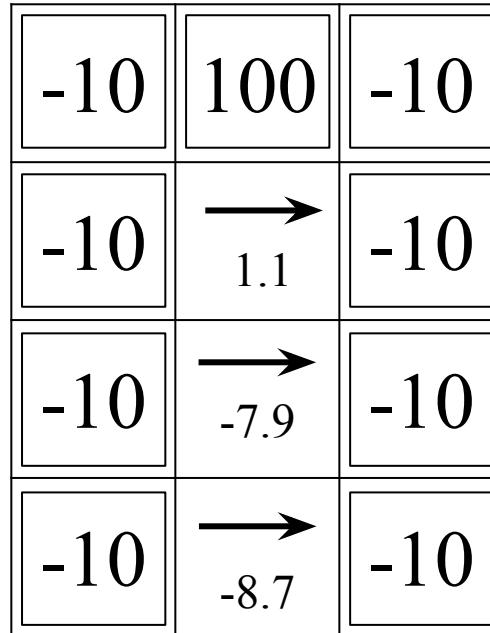
$$\begin{aligned} R(s) &= 0.00 \\ \gamma &= 0.90 \end{aligned}$$

Quiz - Go North



$$R(s) = 0.00$$
$$\gamma = 0.90$$

Example: Policy Evaluation



$$\begin{aligned} R(s) &= 0.00 \\ \gamma &= 0.90 \end{aligned}$$

Policy Evaluation

- How do we calculate the V 's for a fixed policy π ?
 - Idea 1: Turn recursive Bellman equations into updates (like value iteration)

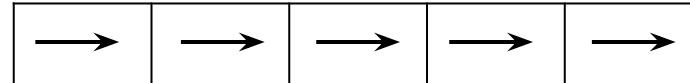
$$\begin{aligned} V_0^\pi(s) &= 0 \\ V_{k+1}^\pi(s) &\leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')] \end{aligned}$$

- Efficiency: $O(S^2)$ per iteration
- Idea 2: Without the max, the Bellman equations are just a linear system
 - Use a linear system solver

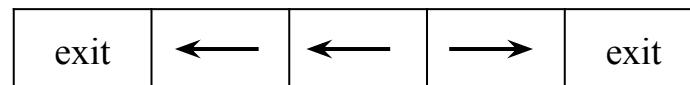
Quiz - Policy Evaluation

10					1
a	b	c	d	e	

- Consider the same grid world as in the previous quiz, where east and west actions are successful 100% of the time and $\gamma = 1$
- Consider the policy π

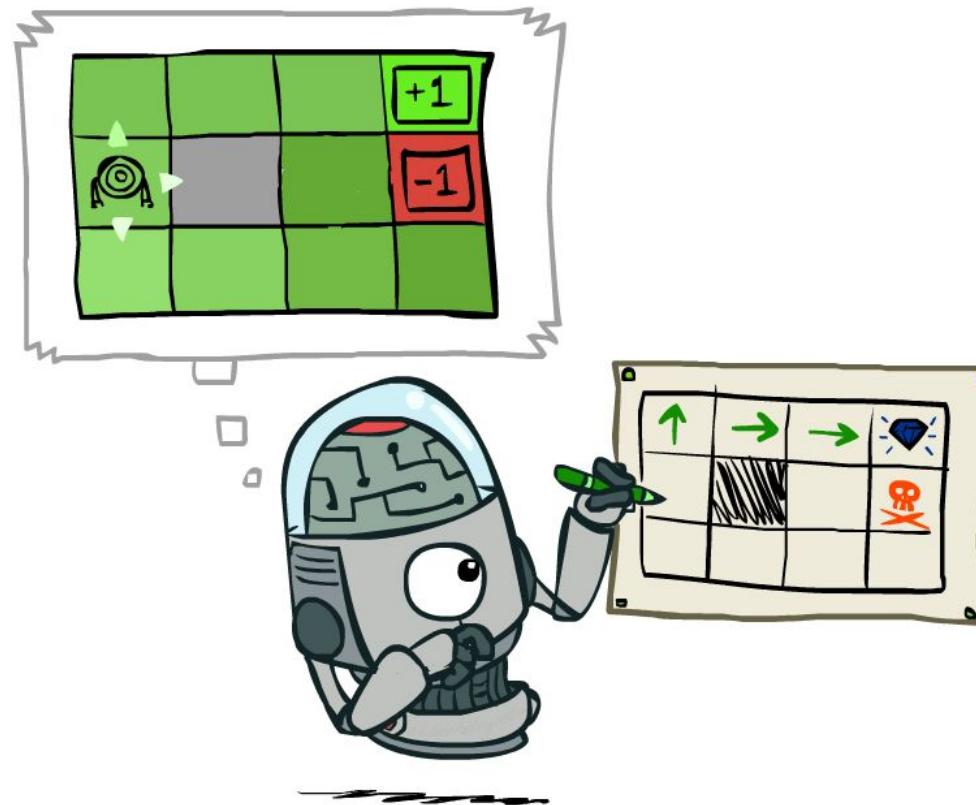


- Evaluate the values for all states
- Consider the policy π'



- Evaluate the values for all states

Policy Extraction



Policy Extraction

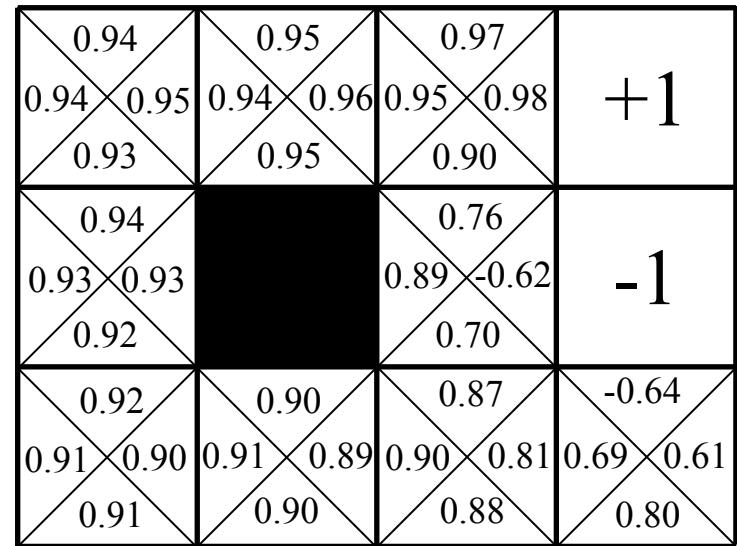
- Let's imagine we have the optimal values $V^*(s)$

0.95	0.97	0.98	+1
0.94		0.89	-1
0.93	0.92	0.90	0.82

- How should we act?
- We need to do a mini-expectimax (one step)
$$\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$
- This is called policy extraction, since it gets the policy implied by the values

Policy Extraction

- Let's imagine we have the optimal q-values
- How should we act?
- Super easy:
$$\pi^*(s) = \arg \max_a Q^*(s, a)$$
- > Actions are easier to select from q-values than values



Can we improve the runtime of value iteration?

Value Iteration - V_0

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00		0.00	0.00
$R(s) = -0.10$ $\gamma = 1.0$	0.00	0.00	0.00

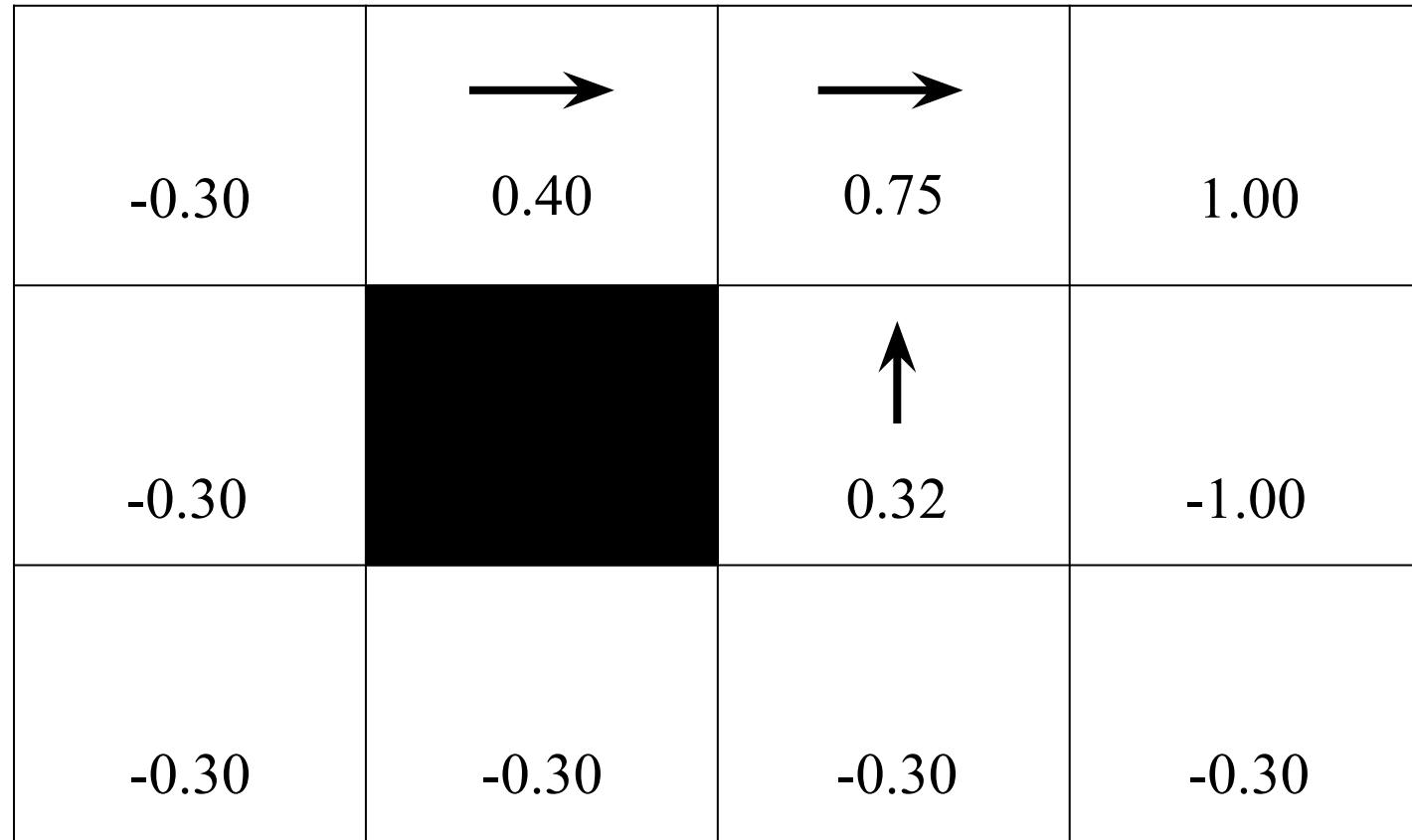
Value Iteration - V_1

-0.10	-0.10	-0.10	1.00
-0.10		-0.10	-1.00
R(s) = -0.10 $\gamma = 1.0$	-0.10	-0.10	-0.10

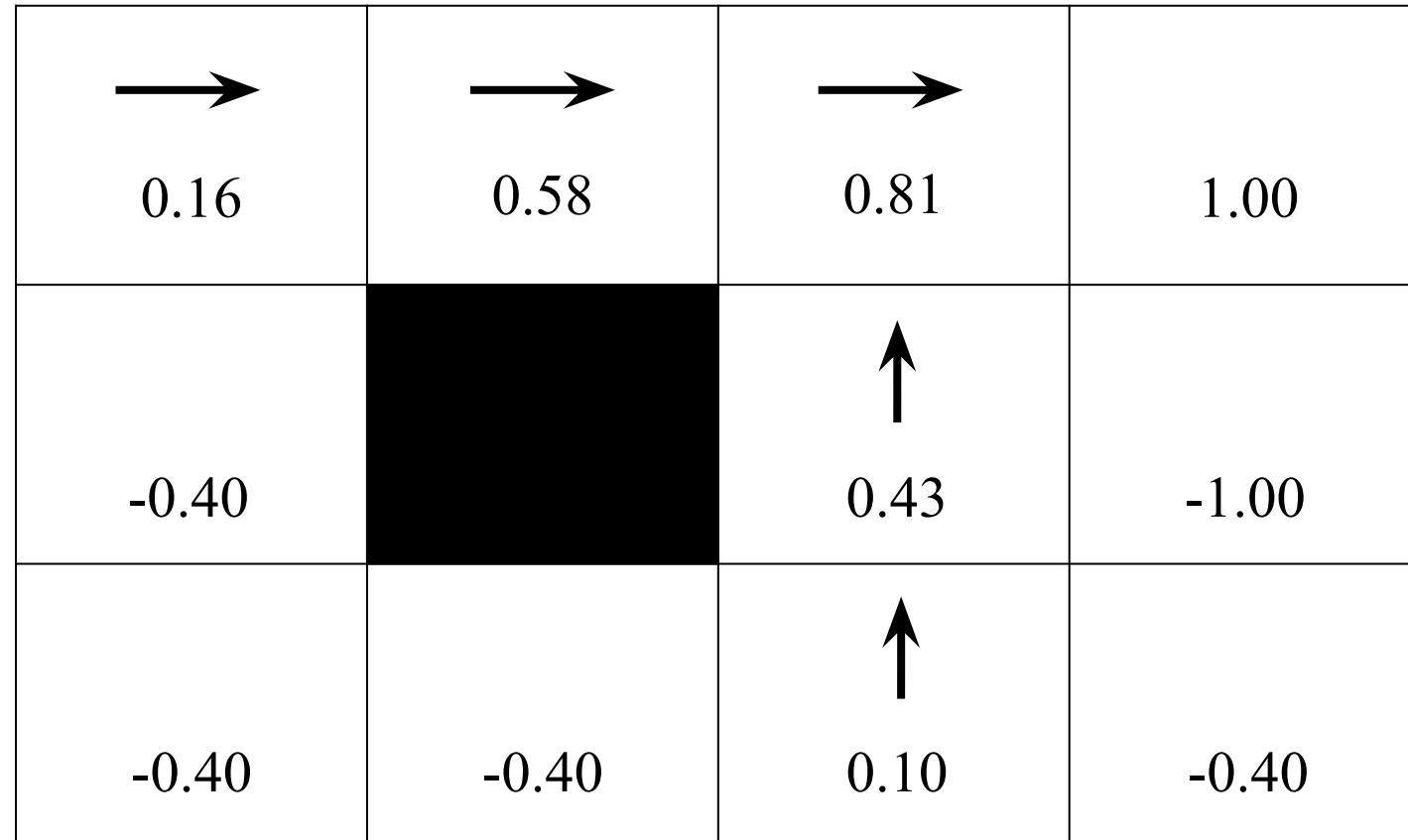
Value Iteration - V_2

-0.20	-0.20		0.68	1.00
-0.20			-0.20	-1.00
R(s) = -0.10 $\gamma = 1.0$	-0.20	-0.20	-0.20	-0.20

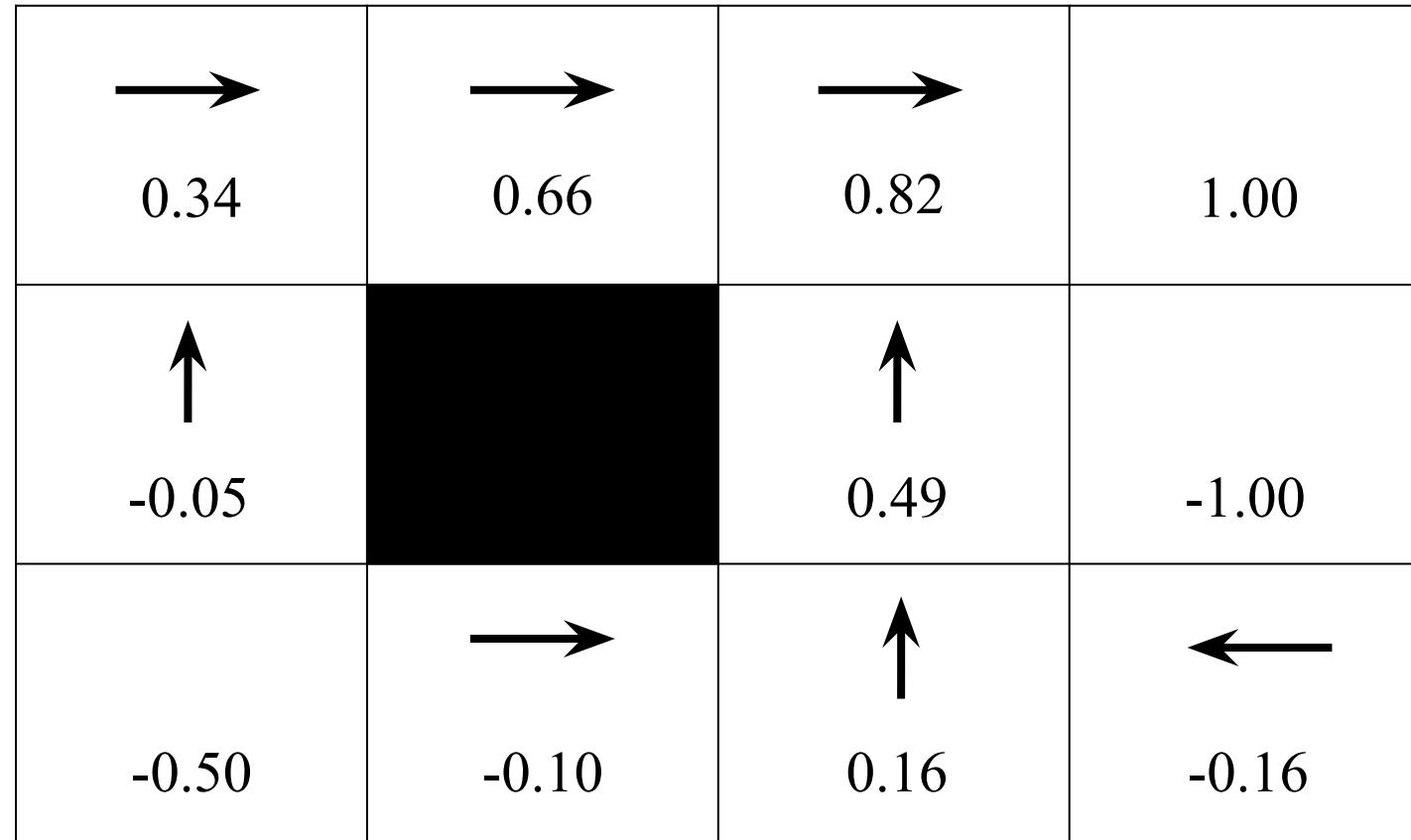
Value Iteration - V_3



Value Iteration - V_4



Value Iteration - V_5

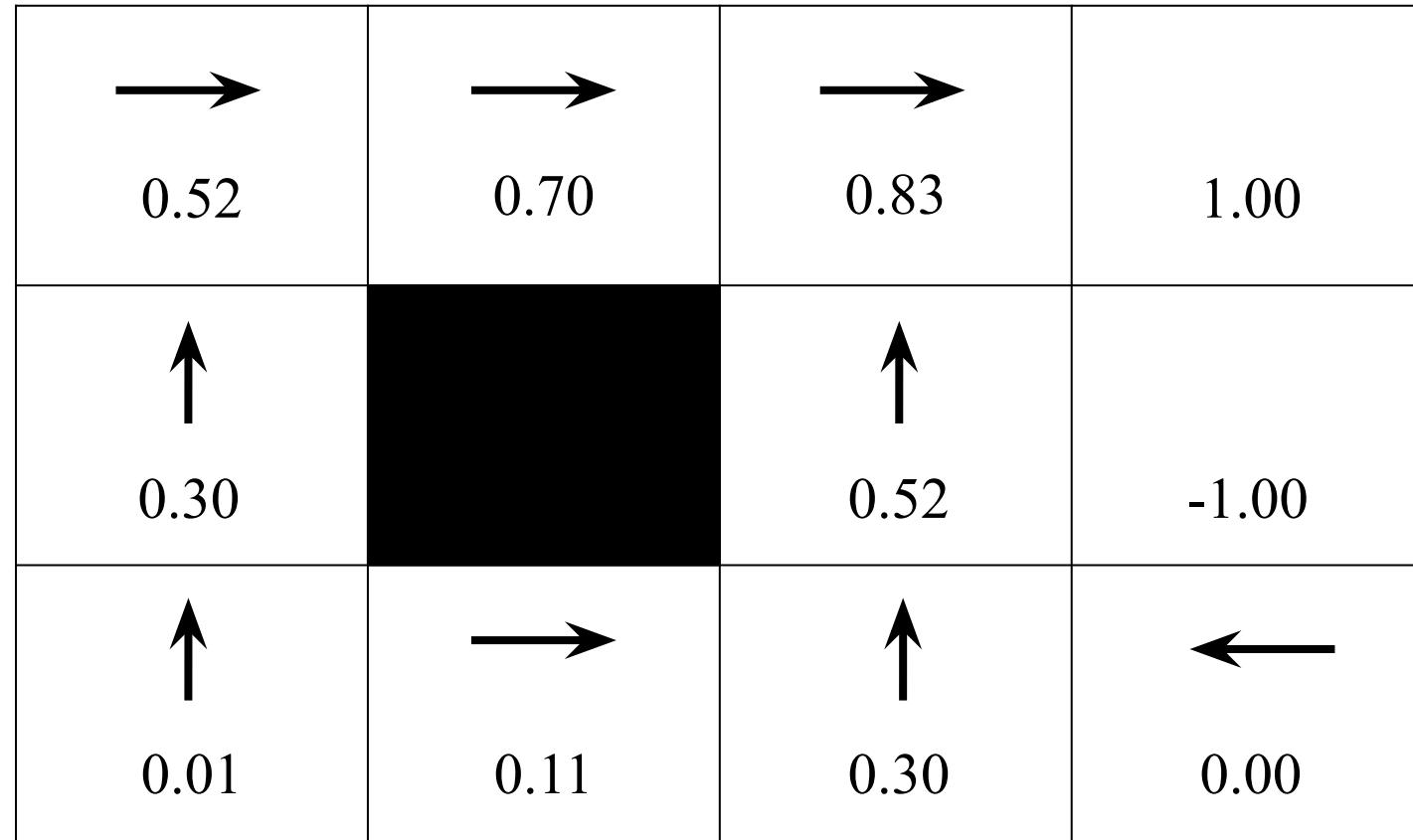


Value Iteration - V_6

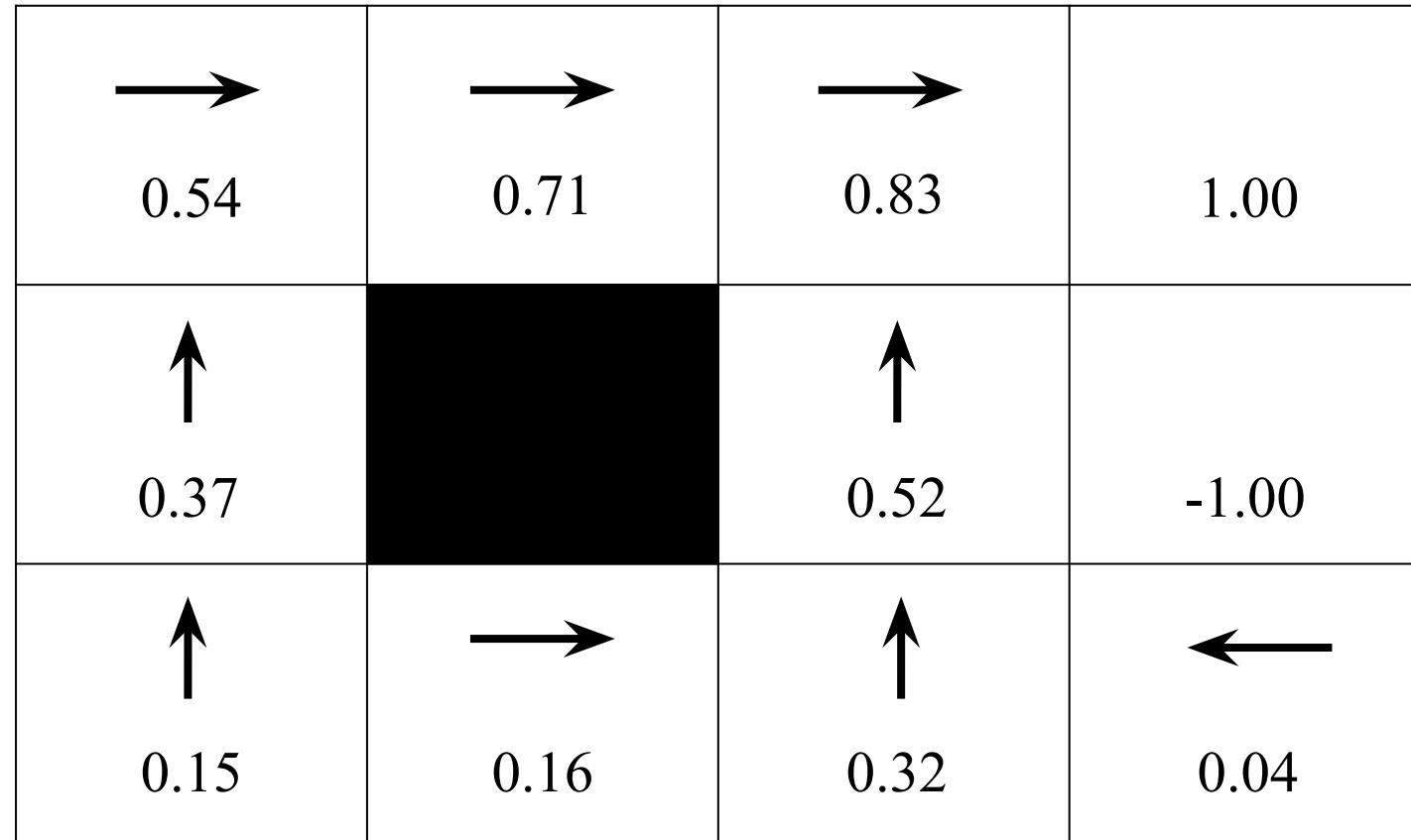
\rightarrow 0.46	\rightarrow 0.69	\rightarrow 0.83	1.00
\uparrow 0.16		\uparrow 0.51	-1.00
\uparrow -0.20	\rightarrow 0.01	\uparrow 0.26	\leftarrow -0.08

$$R(s) = -0.10 \\ \gamma = 1.0$$

Value Iteration - V_7



Value Iteration - V_8

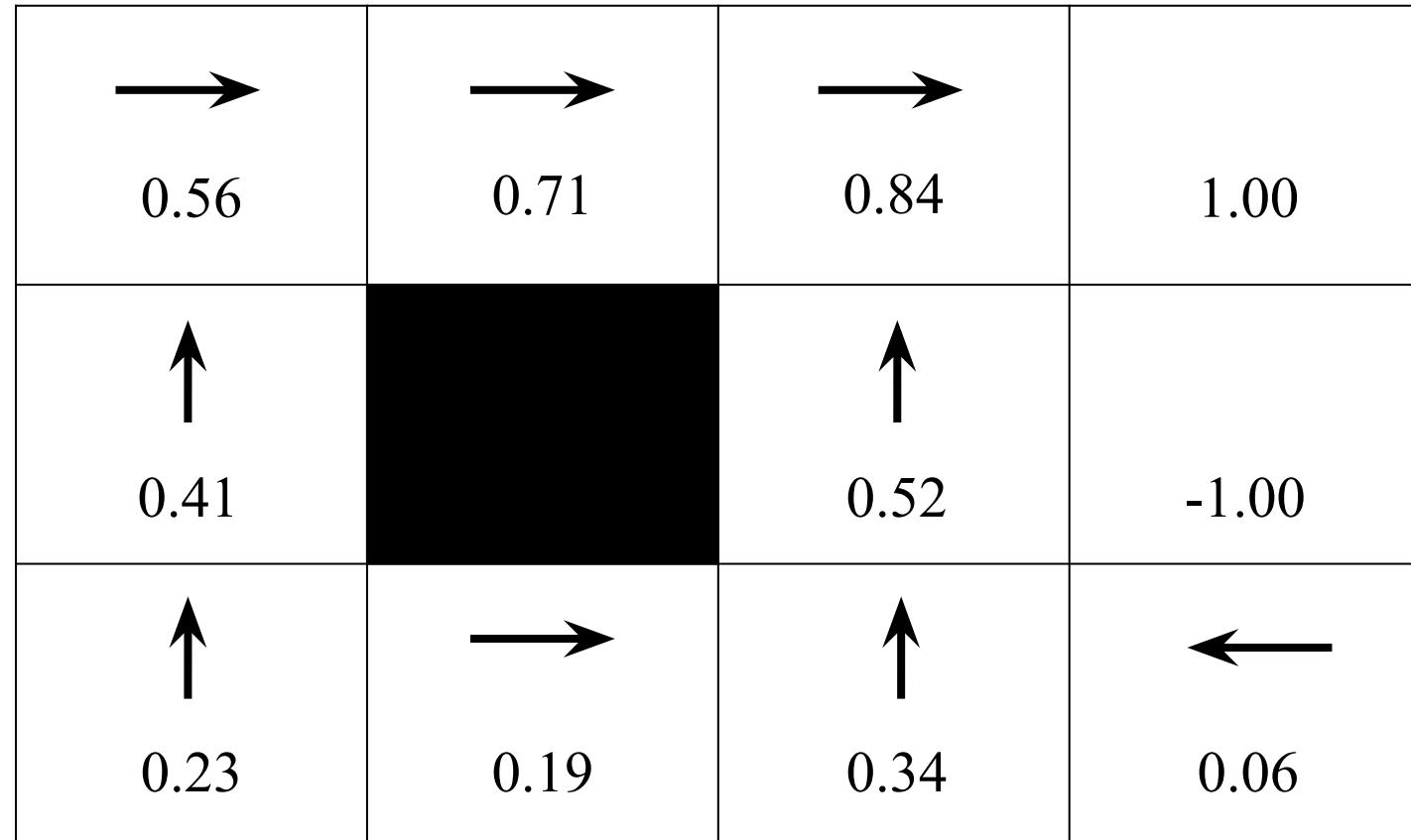


Value Iteration - V_9

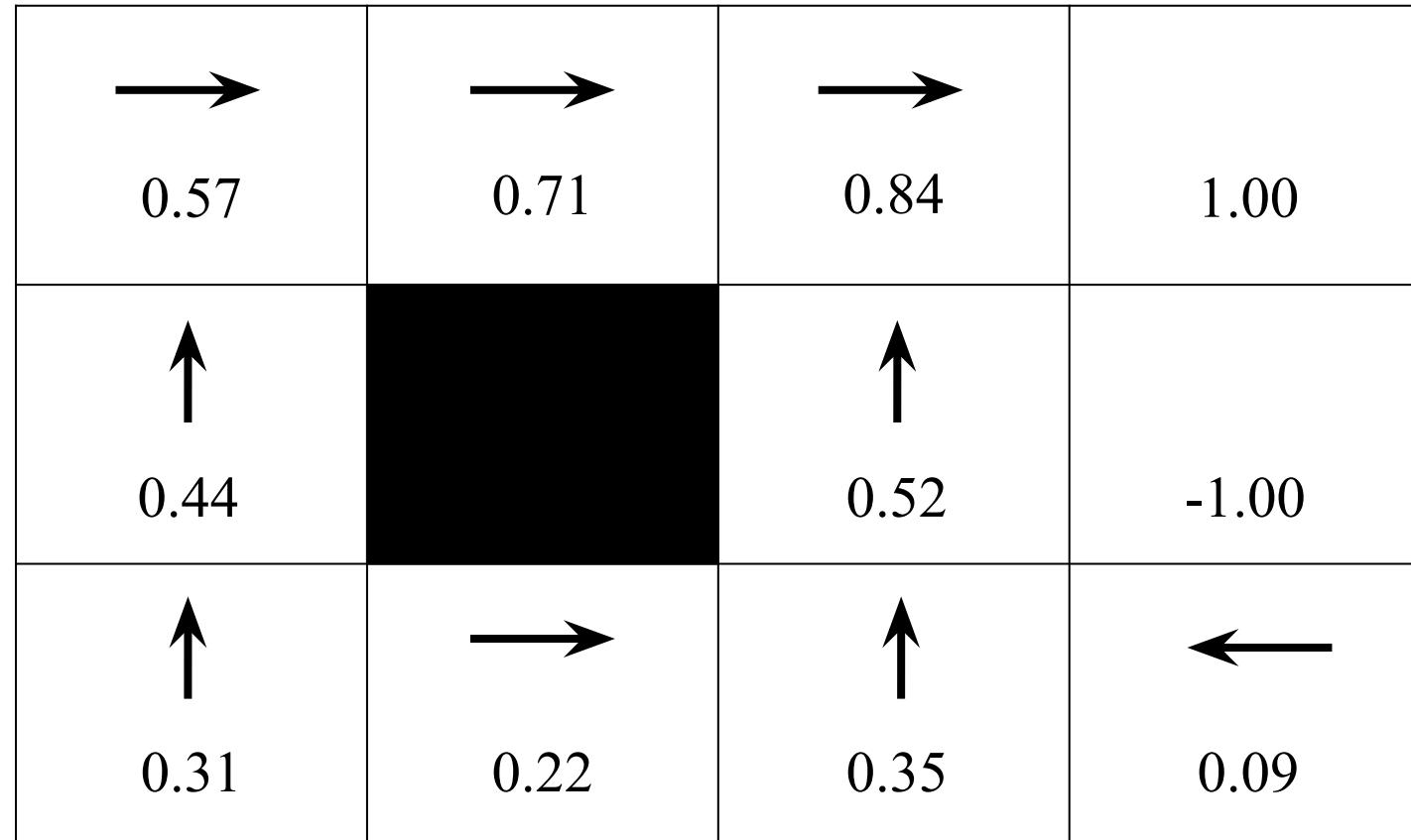
\rightarrow 0.56	\rightarrow 0.71	\rightarrow 0.84	
\uparrow 0.41		\uparrow 0.52	-1.00
\uparrow 0.23	\rightarrow 0.19	\uparrow 0.34	\leftarrow 0.06

$$R(s) = -0.10$$
$$\gamma = 1.0$$

Value Iteration - V_9



Value Iteration - V_{100}

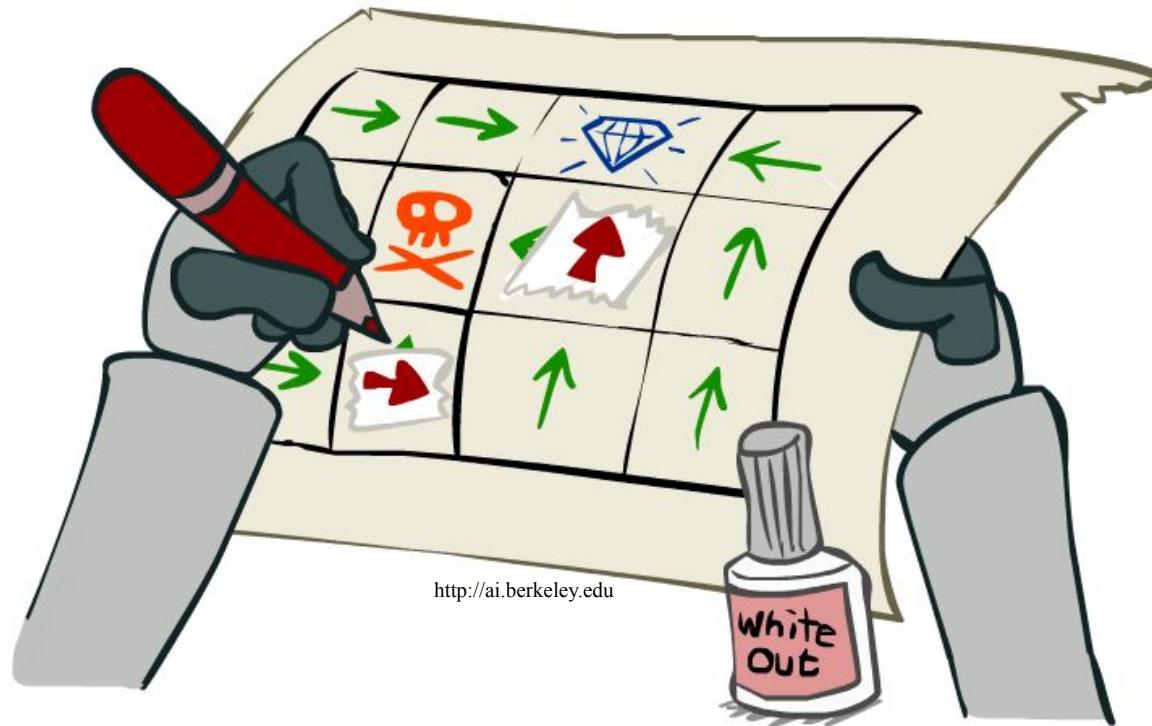


Properties of Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

- Slow
 - $O(S^2A)$ per iteration
- Max at each state rarely changes
 - The policy often converges long before the values

Policy Iteration



<http://ai.berkeley.edu>

Policy Iteration

- Policy iteration is an alternative approach for optimal values
 - Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
 - Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
 - Repeat steps until policy converges
- Still optimal
- Can converge (much) faster under some conditions

Policy Iteration

- Evaluation: For fixed current policy π , find values with policy evaluation
 - Iterate until values converge

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

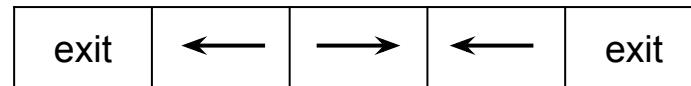
- Improvement: For fixed values, get a better policy using policy extraction
 - One-step look-ahead

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Quiz - Policy Iteration

10				1
a	b	c	d	e

- Consider the same grid world as in the previous quiz, where east and west actions are successful 100% of the time
- $\gamma = 0.9$
- We will execute one round of policy iteration
- Consider the policy π_i shown below



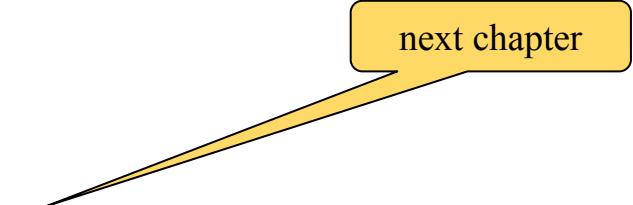
- Evaluate the following quantities
 - Policy evaluation: $V^{\pi_i}(a), V^{\pi_i}(b), V^{\pi_i}(c), V^{\pi_i}(d), V^{\pi_i}(e)$
 - Policy improvement: $\pi_{i+1}(a), \pi_{i+1}(b), \pi_{i+1}(c), \pi_{i+1}(d), \pi_{i+1}(e)$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration
 - Every iteration updates both the values and (implicitly) the policy
 - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration
 - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
 - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
 - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to....
 - Compute optimal values: use value iteration or policy iteration
 - Compute values for a particular policy: use policy evaluation
 - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
 - They basically are – they are all variations of Bellman updates
 - They all use one-step lookahead expectimax fragments
 - They differ only in whether we plug in a fixed policy or max over actions



next chapter

MDP vs. RL

Double-Bandits



Double-Bandits

- An agent can play two slot machines
 - Blue, or Red



You receive \$1

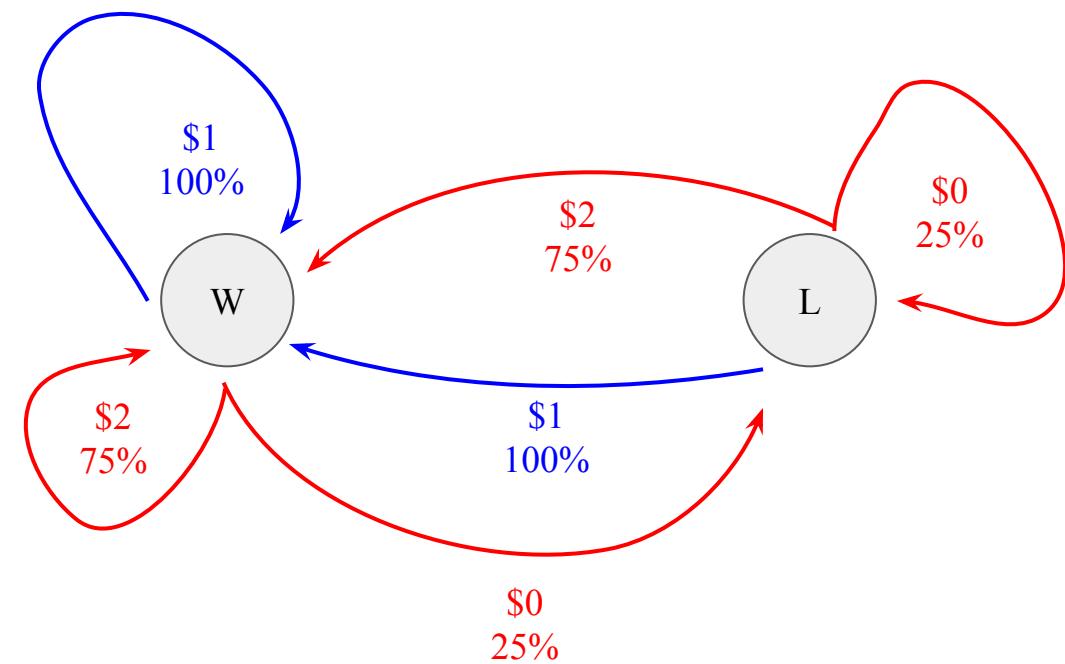


You receive \$0 or \$2,
25% and 75% of the
time, respectively

- What should the agent do?

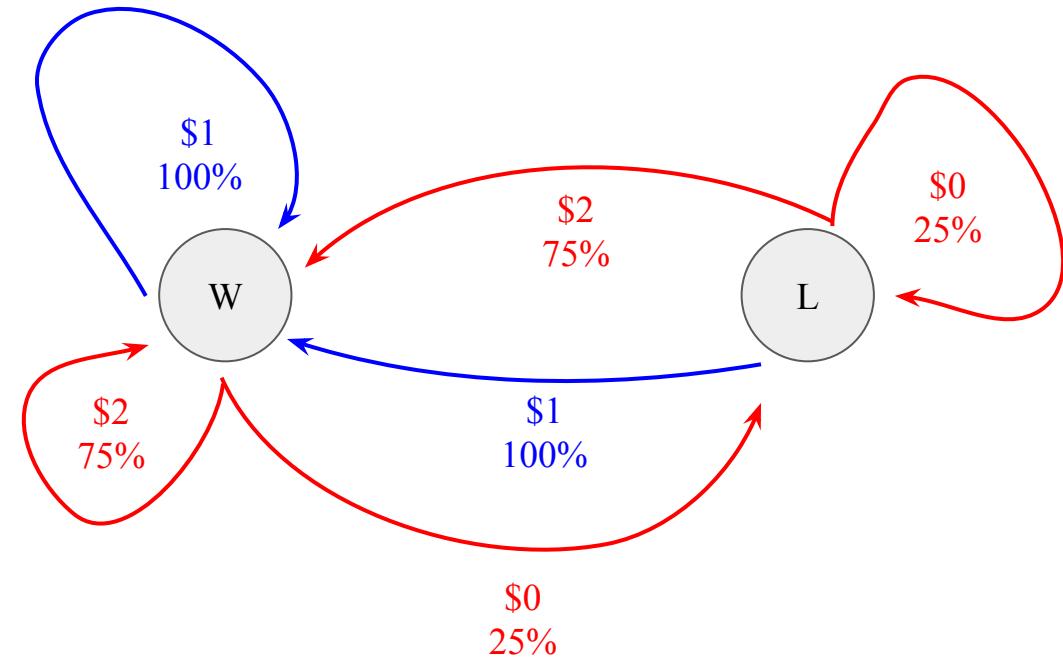
Double-Bandits MDP

- Actions: Blue and Red
- States: Win, Lose
- Assumption:
 - No discount
 - 100 time steps



Offline Planning

- Solving MDPs is offline planning
 - We determine all quantities through computation
 - We need to know the details of the MDP
 - We do not actually play the game

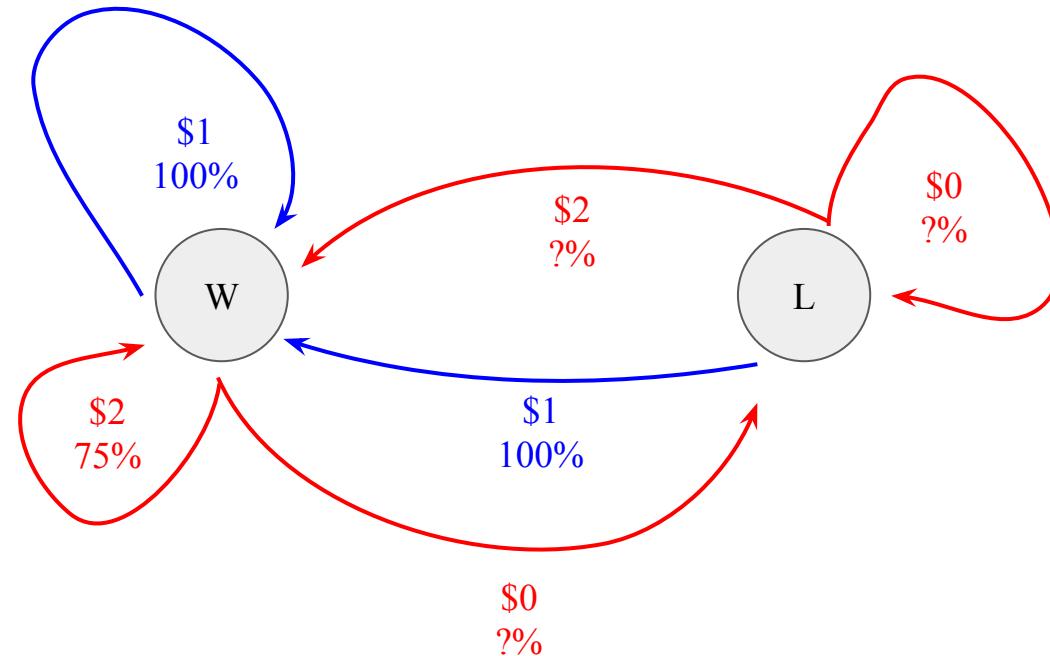


Let's Play!



\$2 \$0 \$2 \$2 \$0 \$0

Rules Changed!



Let's Play!



\$1



\$0 \$0 \$0 \$0 \$0 \$2

What Just Happened ?

- That wasn't planning, it was learning
 - Specifically, reinforcement learning
 - There was an MDP, but you couldn't solve it with just computation !
 - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
 - Exploration: you have to try unknown actions to get information
 - Exploitation: eventually, you have to use what you know
 - Regret: even if you learn intelligently, you make mistakes
 - Sampling: because of chance, you have to try things repeatedly
 - Difficulty: learning can be much harder than solving a known MDPs

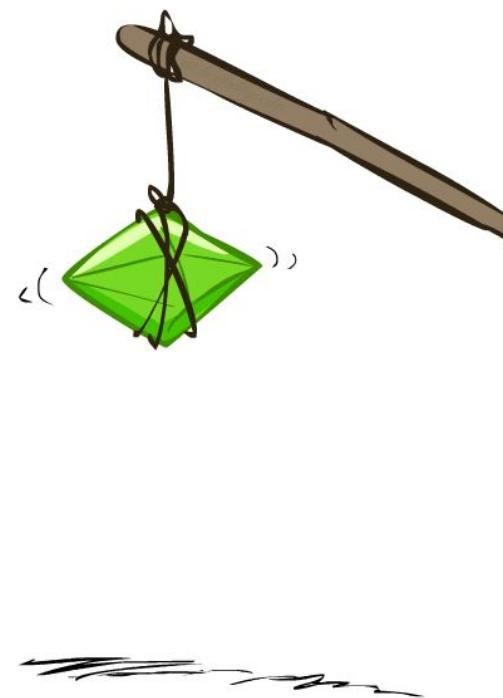
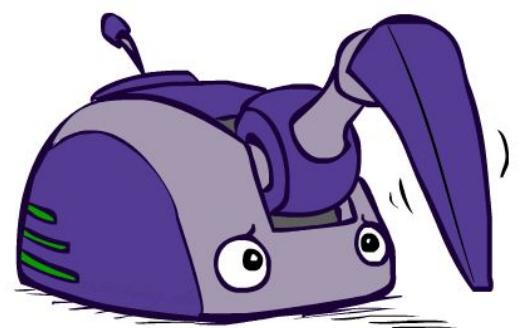


Chapter 3

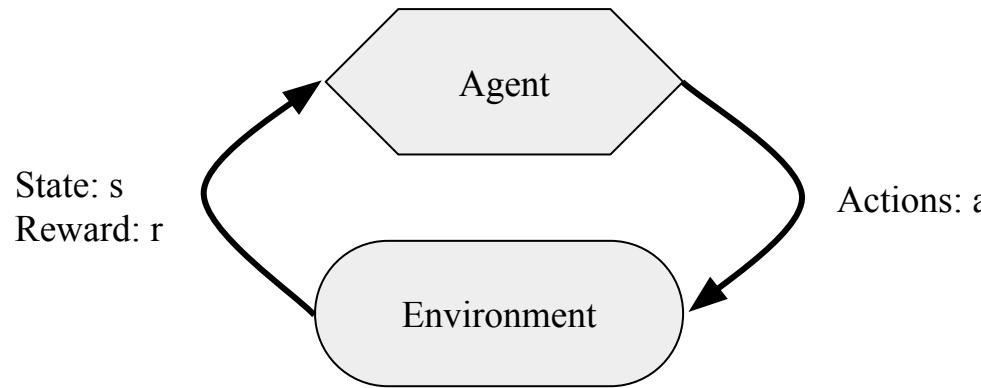
Reinforcement Learning

COMP 3270
Artificial Intelligence

Dirk Schnieders

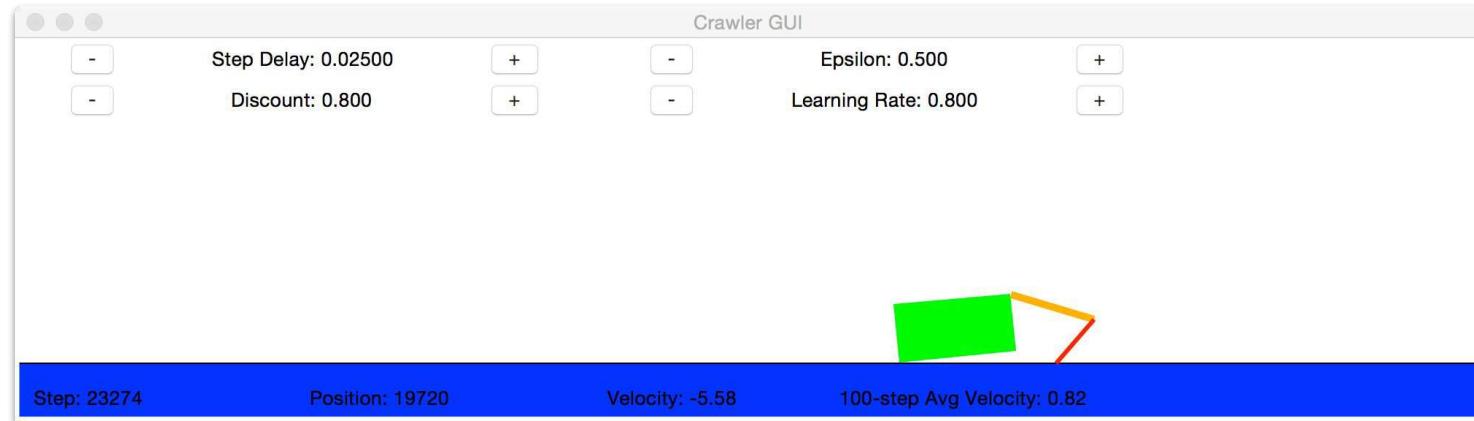


Reinforcement Learning - Idea



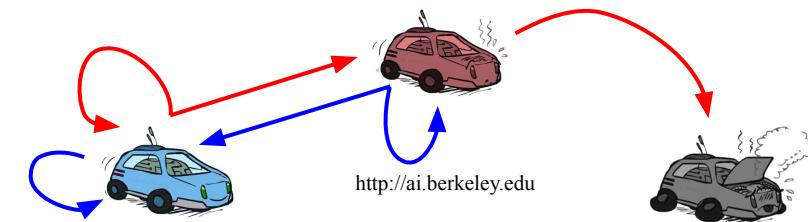
- Basic idea:
 - Receive feedback in the form of rewards
 - Agent's utility is defined by the reward function
 - Must (learn to) act so as to maximize expected rewards
 - All learning is based on observed samples of outcomes!

Crawler in Assignment 3

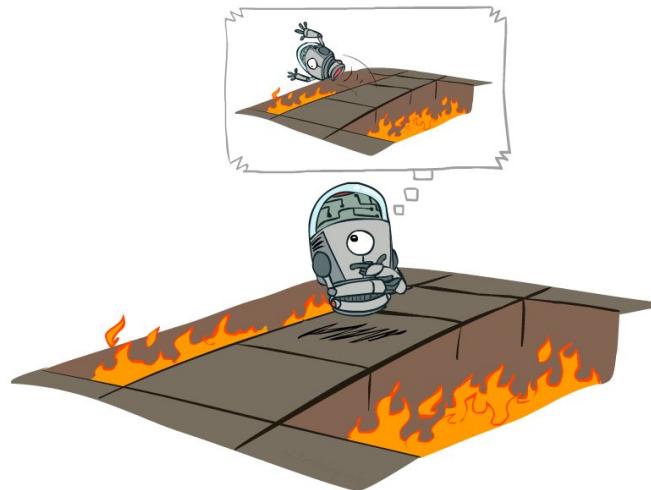


Reinforcement Learning

- It's just the same ...
 - Still assume a Markov decision process (MDP):
 - A set of states $s \in S$
 - A set of actions (per state) A
 - A model $T(s,a,s')$
 - A reward function $R(s,a,s')$
 - Still looking for a policy $\pi(s)$
- New
 - Don't know T or R
 - I.e., we don't know what the actions do
 - Must actually try actions and states out to learn



Offline (MDPs) vs. Online (RL)

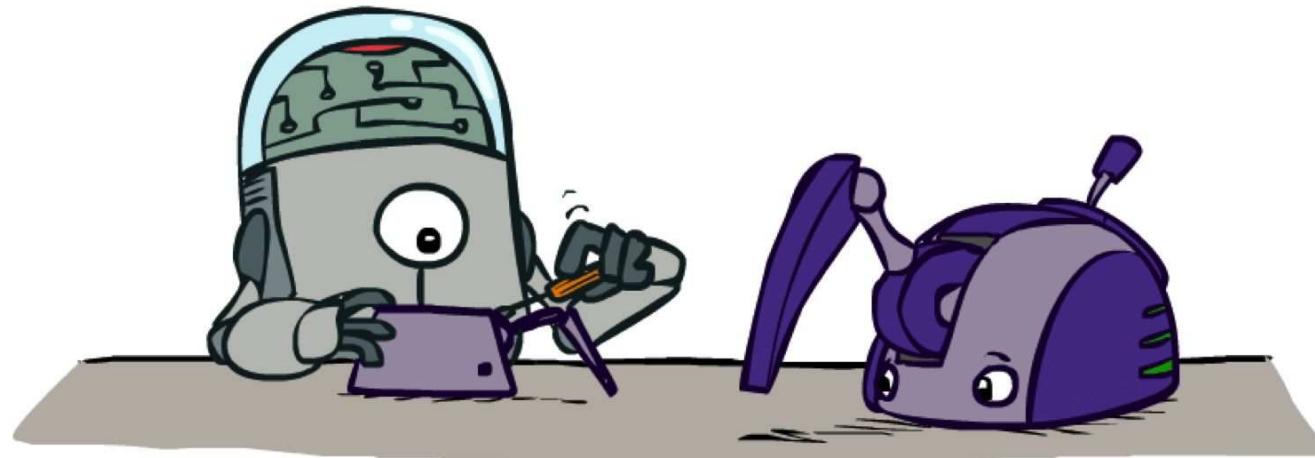


Offline



Online

Model-Based Learning



Model-Based Learning

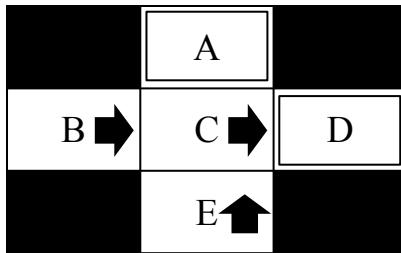
- Model-Based Idea:
 - Learn an approximate model based on experiences
 - Solve for values as if the learned model were correct
 - Step 1: Learn empirical MDP model
 - Count outcomes s' for each s, a
 - Normalize to give an estimate of $\hat{T}(s, a, s')$
 - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
 - Step 2: Solve the learned MDP
 - For example, use policy iteration, as before

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} \hat{T}(s, \pi_i(s), s')[\hat{R}(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$



Example: Model-Based Learning

Given: π



$$\gamma = 1.0$$

Episode 1:
 B, east, C, -1
 C, east, D, -1
 D, exit, , +10

Episode 2:
 B, east, C, -1
 C, east, D, -1
 D, exit, , +10

Episode 3:
 B, east, C, -1
 C, east, D, -1
 D, exit, , +10

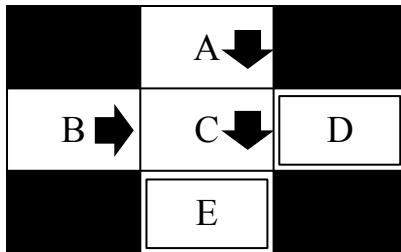
Episode 4:
 E, north, C, -1
 C, east, A, -1
 A, exit, , -10

$T^\wedge(s, a, s')$:
 $T^\wedge(B, \text{east}, C) = 1.00$
 $T^\wedge(C, \text{east}, D) = 0.75$
 $T^\wedge(C, \text{east}, A) = 0.25$
 ...

$R^\wedge(s, a, s')$:
 $R^\wedge(B, \text{east}, C) = -1$
 $R^\wedge(C, \text{east}, D) = -1$
 $R^\wedge(C, \text{east}, A) = -1$
 ...

Quiz - Model-Based Learning

Given: π



$$\gamma = 1.0$$

Episode 1:
A, south, C, -1
C, south, E, -1
E, exit, , 10

Episode 2:
B, east, C, -1
C, south, D, -1
D, exit, , -10

Episode 3:
B, east, C, -1
C, south, E, -1
E, exit, , 10

Episode 4:
A, south, C, -1
C, south, E, -1
E, exit, , 10

$T^{\wedge}(s, a, s')$:
 $T^{\wedge}(A, \text{south}, C) = ?$
 $T^{\wedge}(B, \text{east}, C) = ?$
 $T^{\wedge}(C, \text{south}, E) = ?$
 $T^{\wedge}(C, \text{south}, D) = ?$

Example: Expected Age

- Goal: Compute expected age of comp3270 students

Known P(a)
$E[A] = \sum_a P(a) \cdot a = 0.35 \times 20 + \dots$

Unknown P(a): Model based

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown P(a): Model free

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Quiz - Model-Based Learning

- Consider an MDP with 3 states, A, B and C; and 2 actions CW and CCW
- We are given samples of what an agent experiences
- In this problem, we will first estimate the model
 - i.e., the transition function and the reward function
- Consider the samples (shown on the next slide) that the agent encountered

Quiz (continued) - Model-Based Learning

s	a	s'	r
A	CW	C	-9.0
A	CW	C	-9.0
A	CW	B	0.0
A	CW	B	0.0
A	CW	B	0.0
A	CCW	B	0.0
A	CCW	B	0.0
A	CCW	B	0.0
A	CCW	C	-3.0
A	CCW	B	0.0

s	a	s'	r
B	CW	A	2.0
B	CW	A	2.0
B	CW	A	2.0
B	CW	A	2.0
B	CW	A	2.0
B	CW	A	2.0
B	CCW	A	-6.0
B	CCW	A	-6.0
B	CCW	C	5.0
B	CCW	C	5.0
B	CCW	C	5.0

s	a	s'	r
C	CW	A	3.0
C	CW	A	3.0
C	CW	A	3.0
C	CW	A	3.0
C	CW	A	3.0
C	CW	B	-4.0
C	CCW	A	0.0
C	CCW	A	0.0
C	CCW	A	0.0
C	CCW	A	0.0
C	CCW	A	0.0

Quiz (continued) - Model-Based Learning

- Estimating the transition function, $T(s,a,s')$ and reward function $R(s,a,s')$ for this MDP
- Fill in the missing values in the table for $T(s,a,s')$ and $R(s,a,s')$

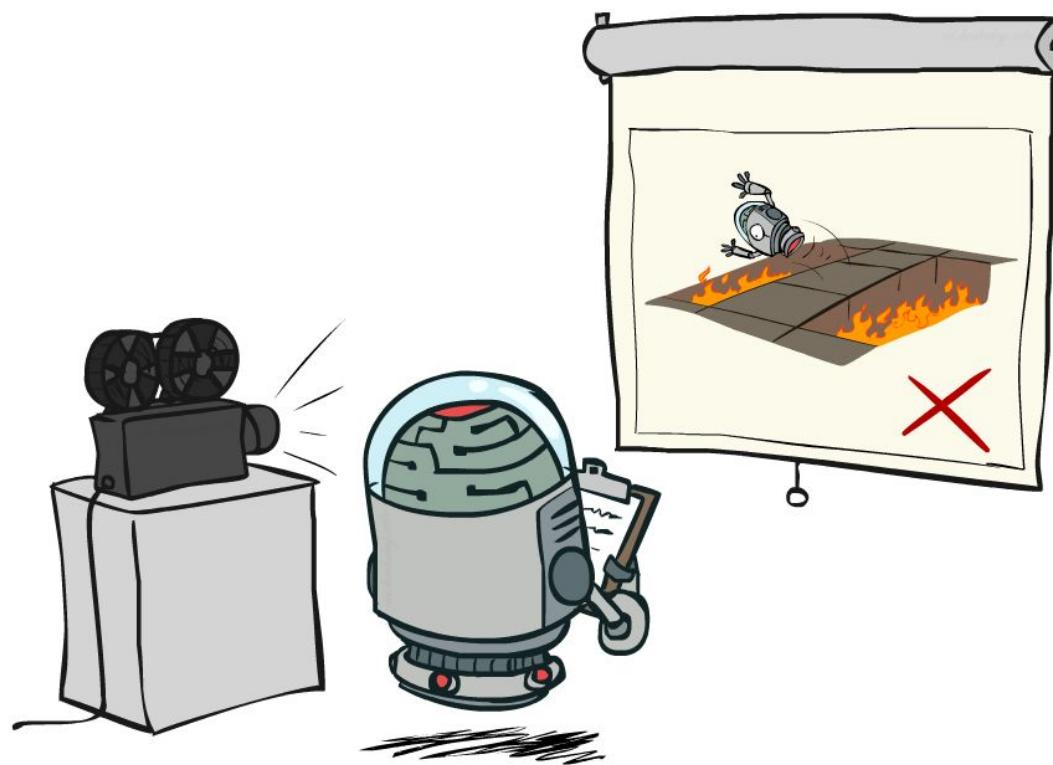
s	a	s'	$T(s,a,s')$	$R(s,a,s')$
A	CW	B		
A	CW	C		
A	CCW	B	0.8	0.0
A	CCW	C	0.2	-3.0
B	CW	A	1.0	2.0
B	CCW	A	0.4	-6.0
B	CCW	C	0.6	5.0
C	CW	A	0.8	3.0
C	CW	B	0.2	-4.0
C	CCW	A	1.0	0.0

Model-Free Learning

In model-free learning
you compare what you
want vs. what you got



Passive Reinforcement Learning



Passive Reinforcement Learning

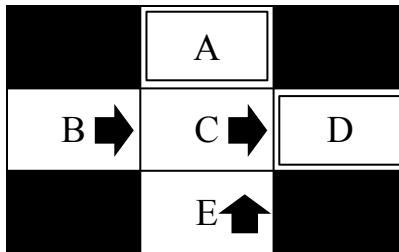
- Simplified task: policy evaluation
 - Input: a fixed policy $\pi(s)$
 - You don't know the transitions $T(s,a,s')$
 - You don't know the rewards $R(s,a,s')$
 - Goal: learn the state values
- In this case:
 - Learner is “along for the ride”
 - No choice about what actions to take
 - Just execute the policy and learn from experience
 - NOT offline planning! You actually take actions in the world

Direct Evaluation

- Goal: Compute values for each state under π
- Idea: Average together observed sample values
 - Act according to π
 - Every time you visit a state, write down what the sum of discounted rewards turned out to be
 - Average those samples
- This is called direct evaluation

Example: Direct Evaluation

Given: π

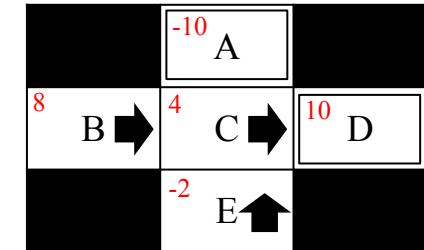


$$\gamma = 1.0$$

Episode 1:
 B, east, C, -1
 C, east, D, -1
 D, exit, , +10

Episode 2:
 B, east, C, -1
 C, east, D, -1
 D, exit, , +10

Output: Values

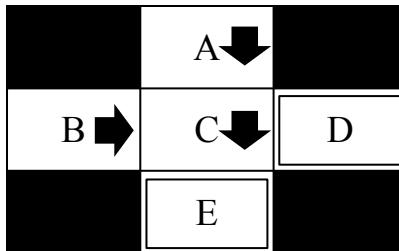


Episode 3:
 E, north, C, -1
 C, east, D, -1
 D, exit, , +10

Episode 4:
 E, north, C, -1
 C, east, A, -1
 A, exit, , -10

Quiz - Direct Evaluation

Given: π



$$\gamma = 1.0$$

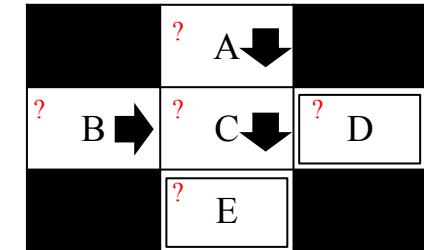
Episode 1:

- A, south, C, -1
- C, south, E, -1
- E, exit, , +10

Episode 2:

- B, east, C, -1
- C, south, D, -1
- D, exit, , -10

Output: Values



Episode 3:

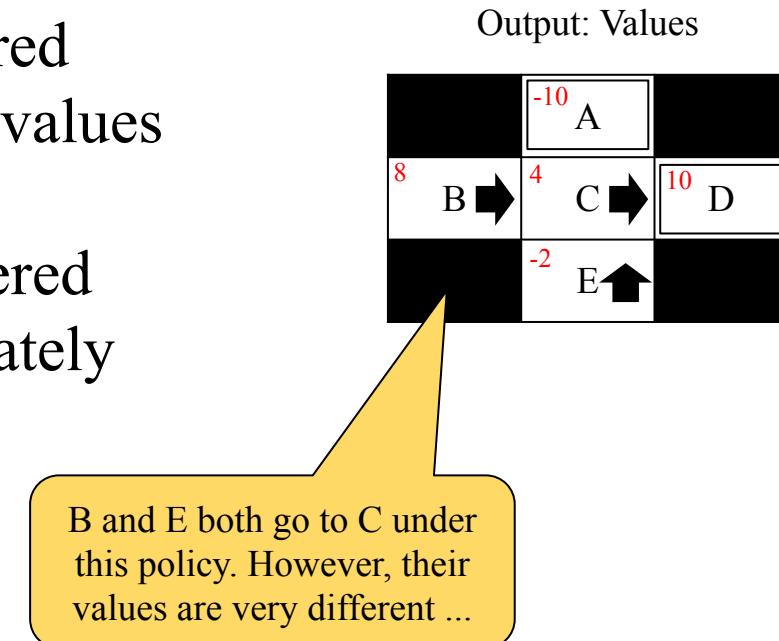
- B, east, C, -1
- C, south, E, -1
- E, exit, , +10

Episode 4:

- A, south, C, -1
- C, south, E, -1
- E, exit, , +10

Problems with Direct Evaluation

- Positives
 - Easy to understand
 - Knowledge of T and R not required
 - Eventually computes the correct values
- Negatives
 - State connections are not considered
 - Each state must be learned separately
 - Takes a long time to learn



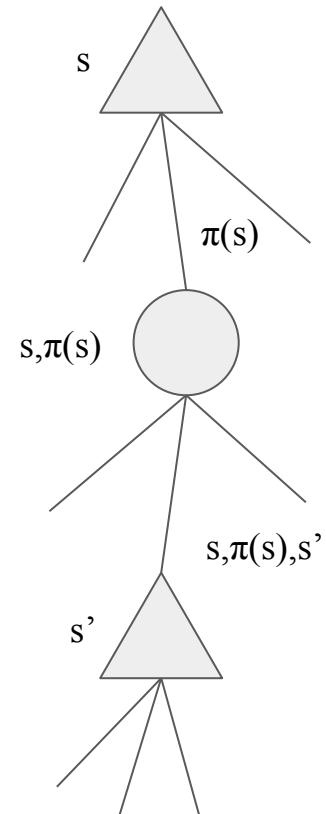
Why Not Use Policy Evaluation?

- Can we use policy evaluation?
 - Each round, replace V with a one-step-look-ahead layer over V

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- This approach fully exploits the connections
 - Unfortunately, we need T and R to do it



Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

...

$$\text{sample}_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

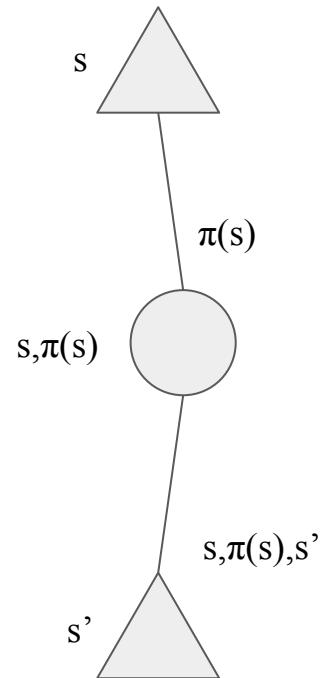
Almost, however we cannot
rewind time to get sample after
sample from state s

Will this work?

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i \text{sample}_i$$

Temporal Difference Learning

- Big idea: learn from every experience!
 - Update $V(s)$ each time we experience a transition (s, a, s', r)
 - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
 - Policy still fixed, still doing evaluation!
 - Move values toward value of whatever successor occurs: running average



Update to $V(s)$: $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)\text{sample}$

Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(\text{sample} - V^\pi(s))$

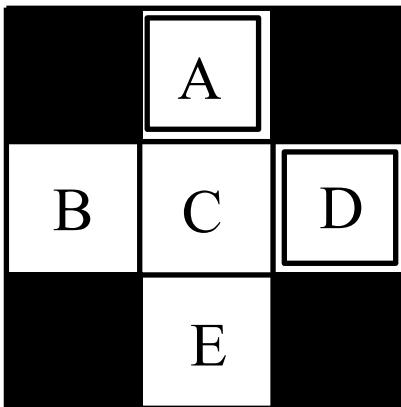
Temporal Difference Learning

- Exponential Moving Average
 - The running interpolation update makes recent samples more important
 - Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (α) can give converging averages

Demo

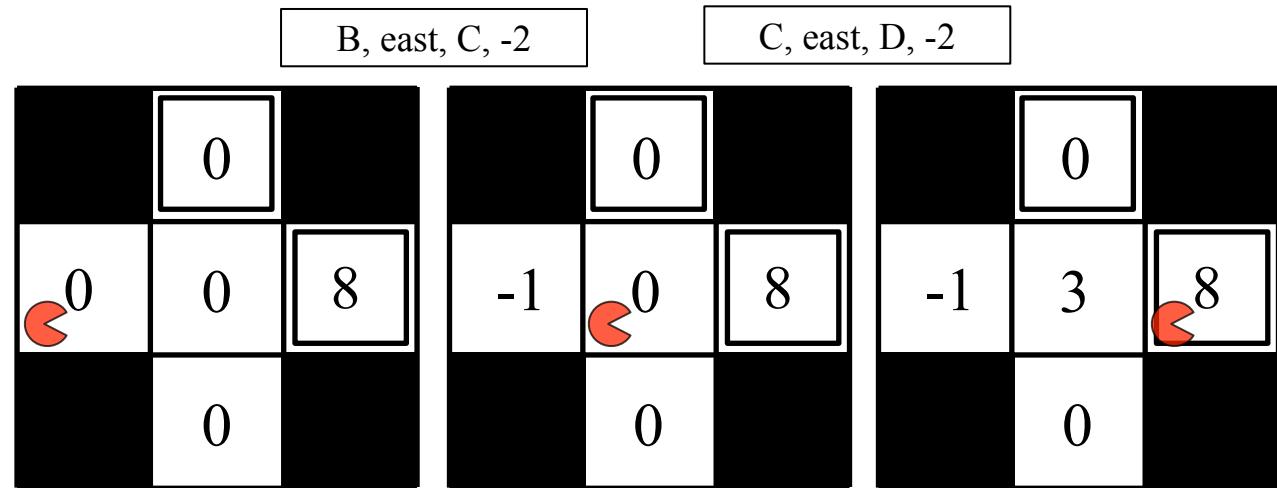
Another Example

States



$$\alpha = 0.5, \gamma = 1.0$$

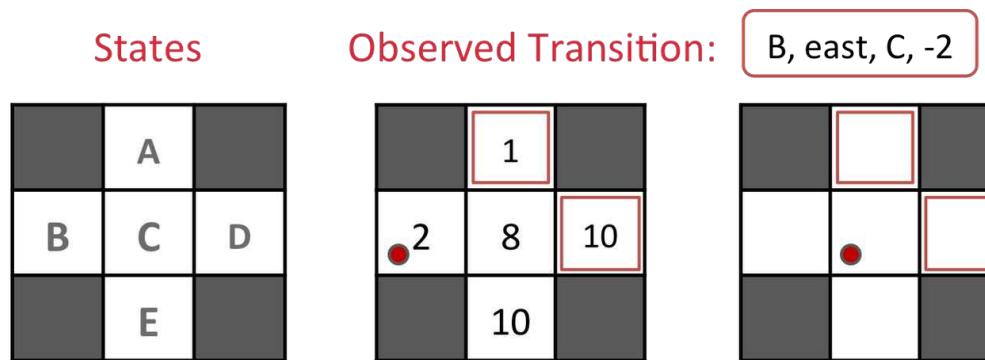
Observed Transitions



$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

Quiz - Temporal Difference Learning

- Consider the grid world shown below
- The left panel shows the name of each state A through E. The middle panel shows the current estimate of the value function V^π for each state
- A transition is observed, that takes the agent from state B through taking action east into state C, and the agent receives a reward of -2.
- What are the value estimates after the TD learning update?



Assume: $\gamma = 1$, $\alpha = 1/2$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

TD Value Learning

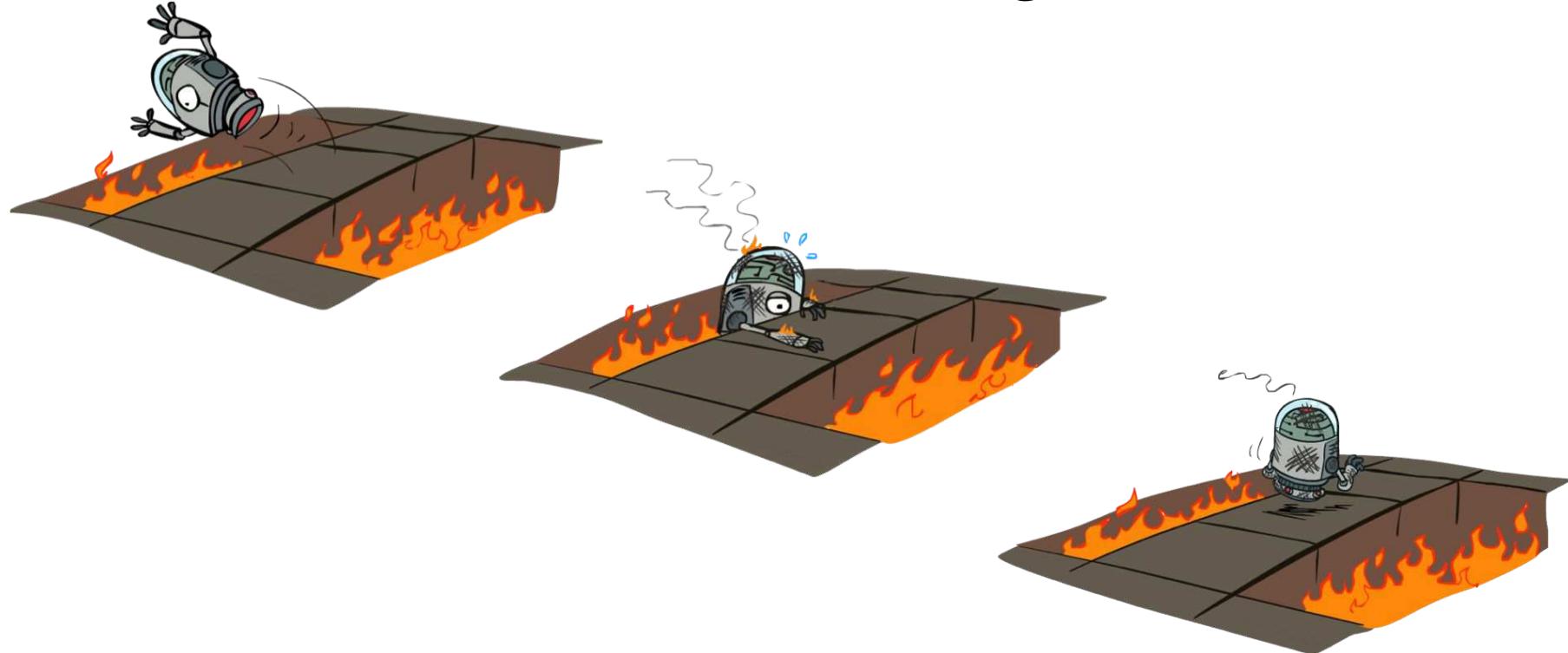
- Model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, we cannot turn values into a new policy

$$\pi(s) = \arg \max_a Q(s, a)$$

$$Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')]$$

- Idea
 - Learn Q-values, not values
 - Makes action selection model-free

Active Reinforcement Learning



Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
 - Start with $V_0(s) = 0$, which we know is right
 - Given V_k , calculate the depth $k+1$ values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- But Q-values are more useful, so compute them instead
 - Start with $Q_0(s, a) = 0$, which we know is right
 - Given Q_k , calculate the depth $k+1$ q-values for all q-states:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

- Learn $Q(s, a)$ values as you go

- Receive a sample (s, a, s', r)
 - Consider your old estimate: $Q(s, a)$
 - Consider your new sample estimate:
 $sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
 - Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [sample]$$

Demo

Quiz - Q-Learning

- Consider an MDP with 3 states, A, B and C; and 2 actions CW and CCW
- In this quiz, instead of first estimating the transition and reward functions, we will directly estimate the Q function using Q-learning
- Assume, the discount factor is 0.5 and the step size for Q-learning is 0.5
- Let the current Q function, $Q(s,a)$, be

	A	B	C
CW	-2	2	1
CCW	0.5	2	10

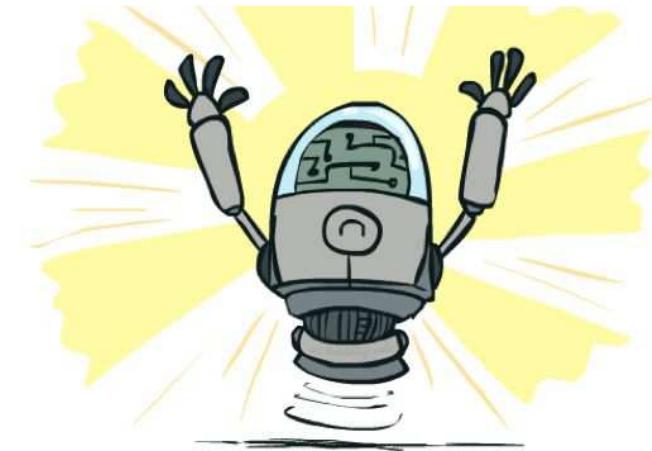
- The agent encounters the following samples
- Process the samples given above and fill in the Q-values after both samples have been accounted for

s	a	s'	r
A	CW	B	0.0
B	CW	A	2.0

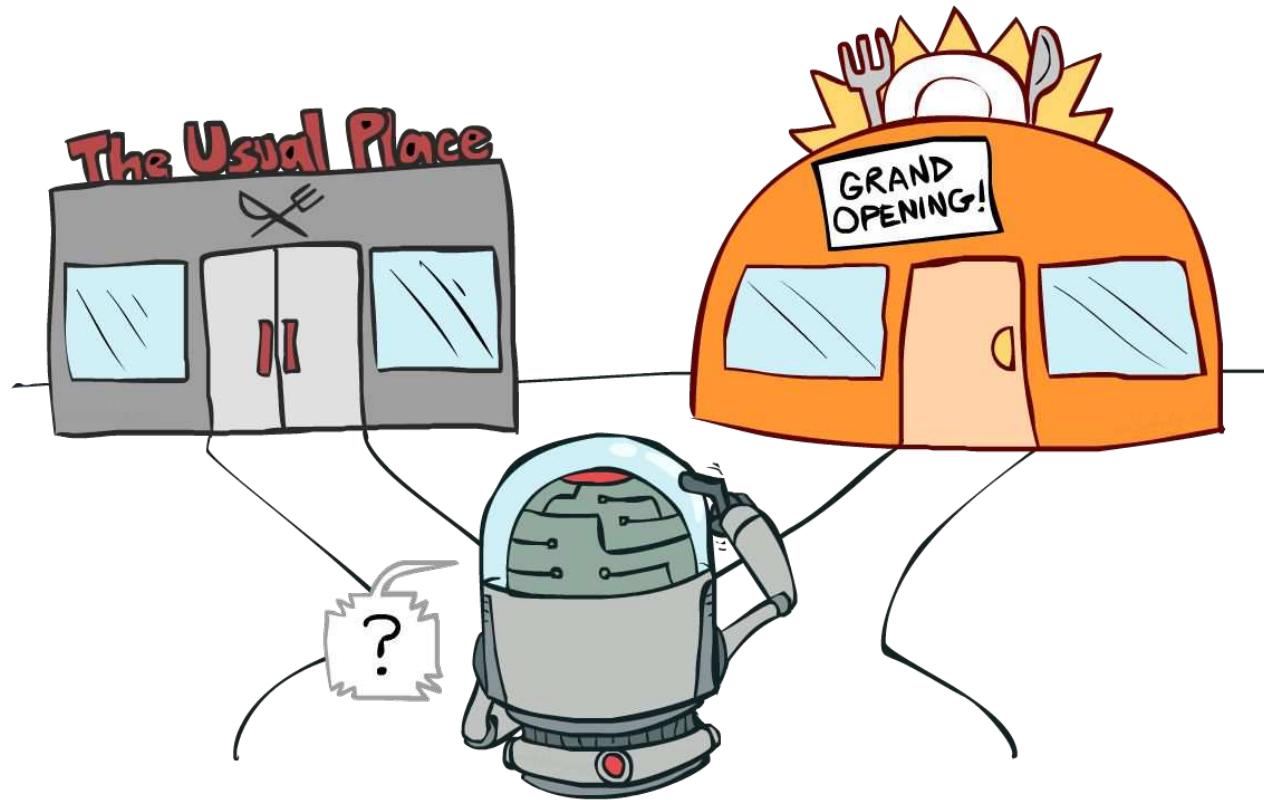
	A	B	C
CW			
CCW			

Q-Learning Properties

- Amazing result
 - Q-learning converges to optimal policy -- even if you're acting suboptimally!
 - This is called off-policy learning
- Limitations:
 - You have to explore enough
 - You have to eventually make the learning rate small enough
 - ... but not decrease it too quickly
 - Basically, in the limit, it doesn't matter how you select actions



Exploration vs. Exploitation

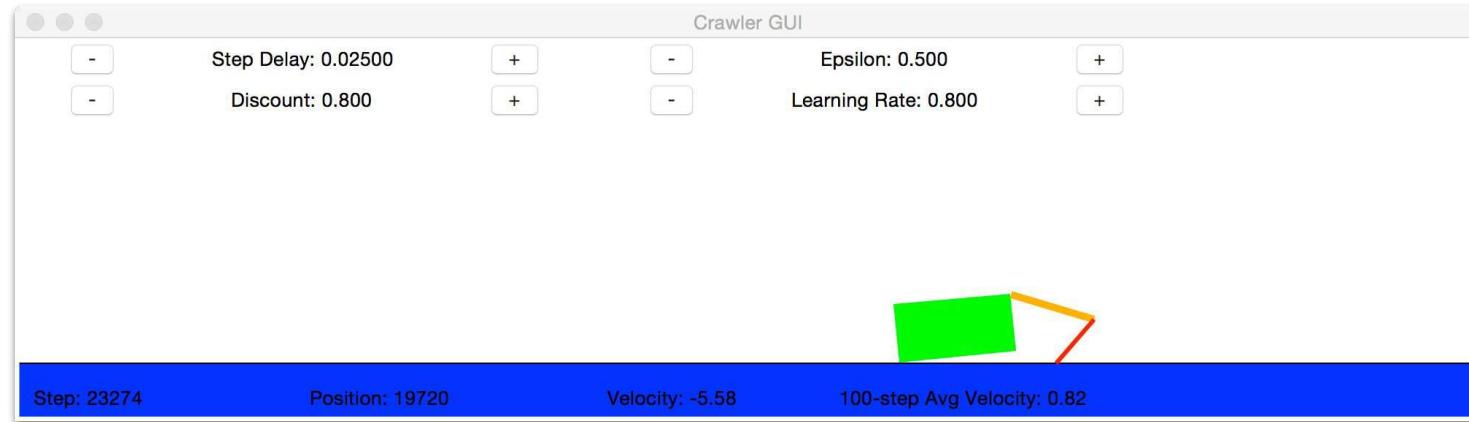


How to Explore?

- Several schemes for forcing exploration
- Simplest: random actions (ϵ -greedy)
 - Every time step, flip a coin
 - With (small) probability ϵ , act randomly
 - With (large) probability $1-\epsilon$, act on current policy
 - Problems with random actions?
 - You do eventually explore the space, but keep thrashing around once learning is done
 - Solution
 - Lower ϵ over time
 - Exploration functions



Crawler in Assignment 3



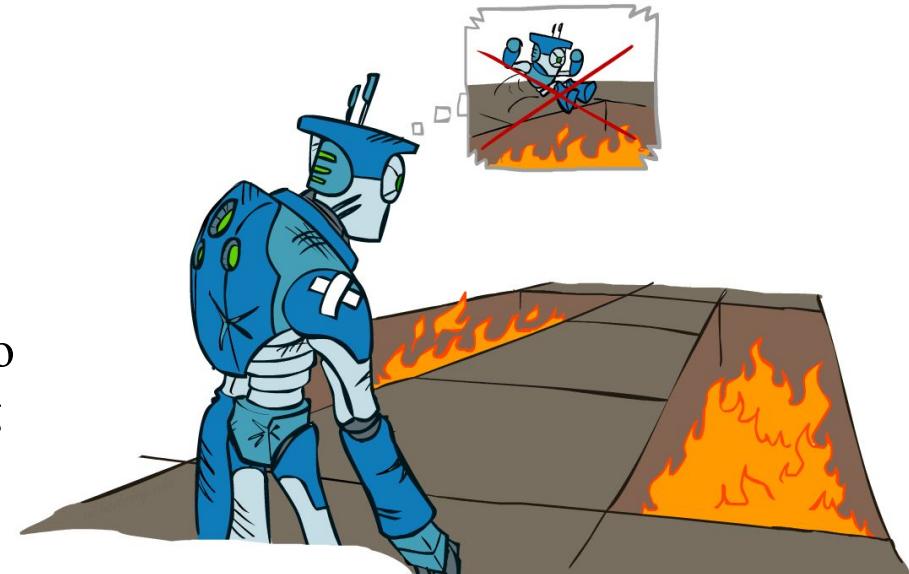
Exploration Functions

- When to explore?
 - Random actions: explore a fixed amount
 - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring
- Exploration function
 - Takes a value estimate u and a visit count n , and returns an optimistic utility, e.g. $f(u, n) = u + k/n$
$$Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} Q(s', a')$$
$$Q(s, a) \leftarrow_{\alpha} R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$$
- Note: this propagates the “bonus” (k) back to states that lead to unknown states as well!

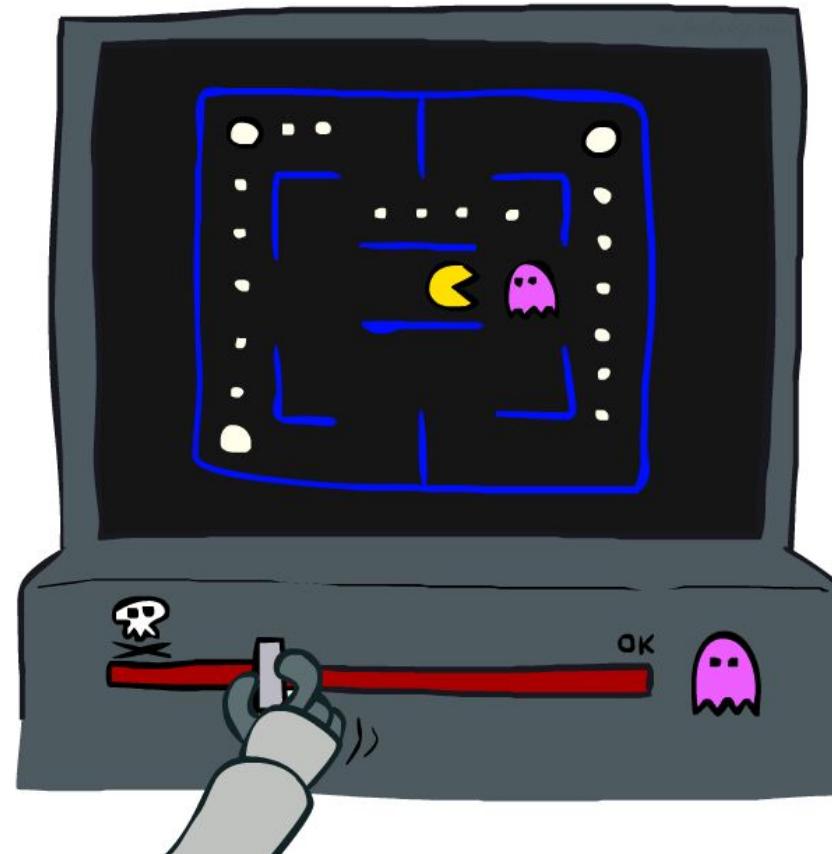


Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret

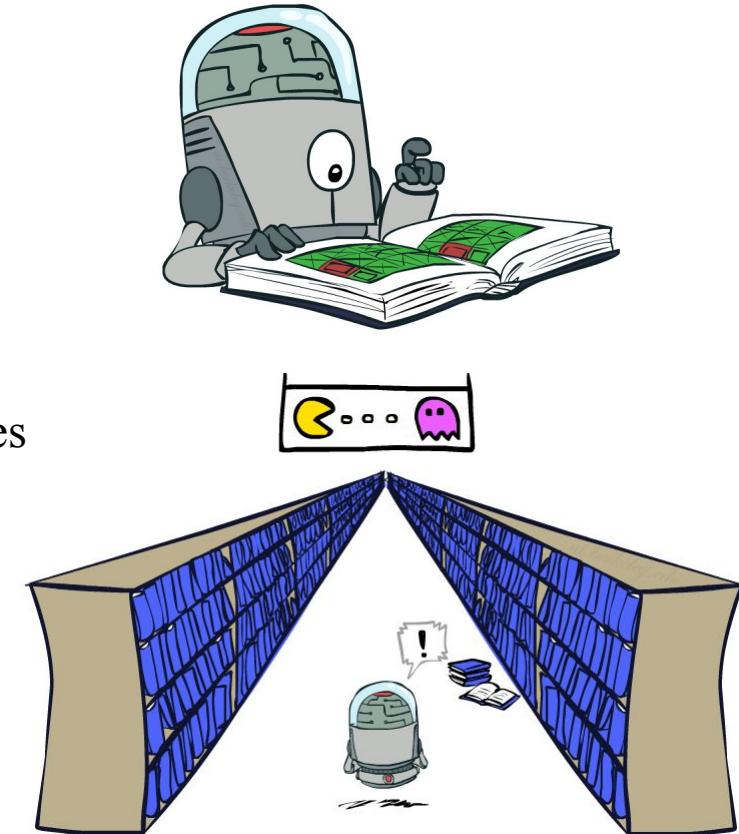


Approximate Q-Learning



Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
 - Too many states to visit them all in training
 - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
 - Learn about some small number of training states from experience
 - Generalize that experience to new, similar situations
 - This is a fundamental idea in machine learning

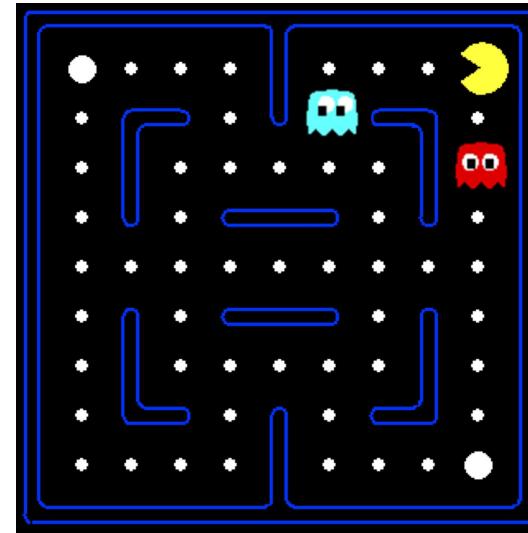


Example: Pacman

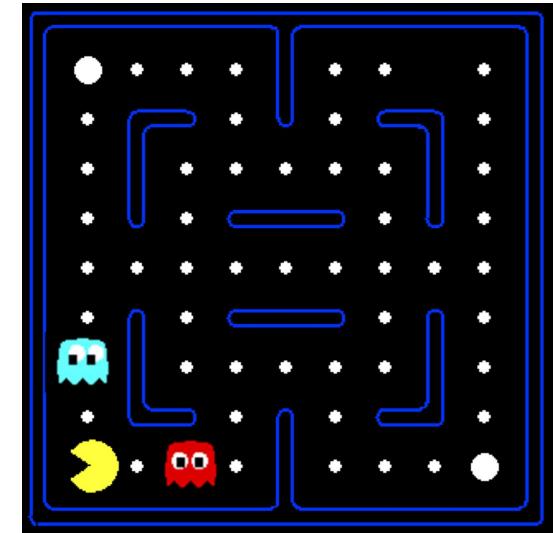
Let's say we discover through experience that this state is bad



In naive q-learning, we know nothing about this state

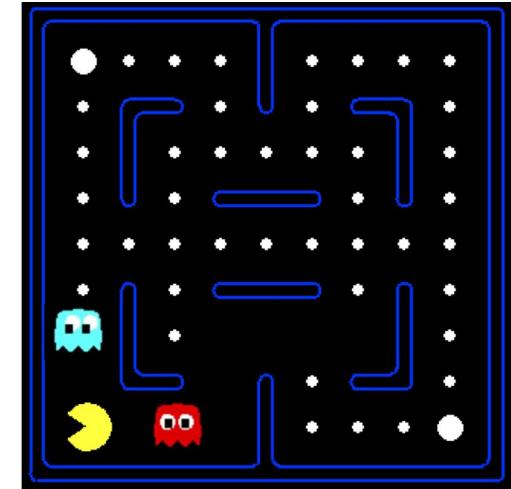


Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
 - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
 - Example features:
 - Distance to closest ghost
 - Distance to closest dot
 - Number of ghosts
 - $1 / (\text{dist to dot})^2$
 - Is Pacman in a tunnel? (0/1)
 - etc.
 - Is it the exact state on this slide?
- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value / Q Functions

- Using a feature representation, we can write a Q-function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Advantage
 - Our experience is summed up in a few powerful numbers
- Disadvantage
 - States may share features but actually be very different in value

Approximate Q-Learning

$$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \dots + w_n f_n(s, a)$$

- Q-learning with linear Q-functions

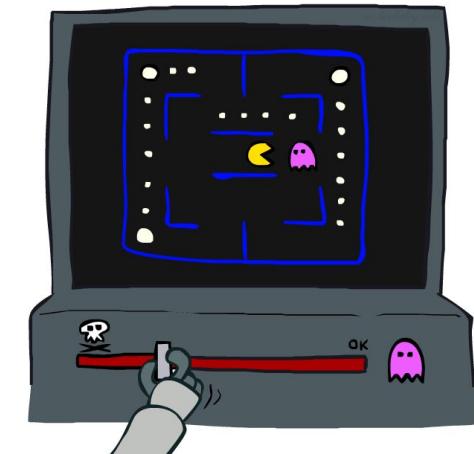
transition = (s, a, r, s')

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a') \right] - Q(s, a)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \text{ [difference]} \quad \text{Exact Q's}$$

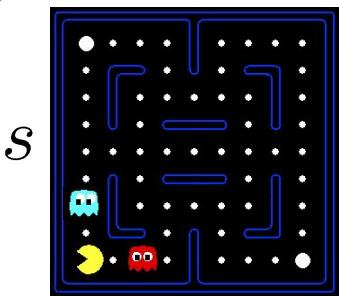
$$w_i \leftarrow w_i + \alpha \text{ [difference]} f_i(s, a) \quad \text{Approximate Q's}$$

- Intuitive interpretation:
 - Adjust weights of active features
 - E.g., if something unexpectedly bad happens, blame the features that were on
 - disprefer all states with that state's features



Example: Q-Pacman

$$Q(s, a) = 4.0f_{DOT}(s, a) - 1.0f_{GST}(s, a)$$

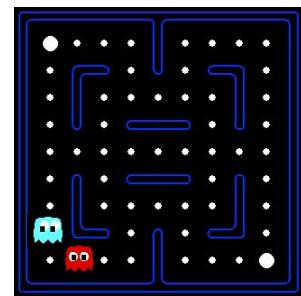


$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

$$\begin{aligned} a &= \text{NORTH} \\ r &= -500 \end{aligned}$$

s'



$$Q(s, \text{NORTH}) = +1$$

$$Q(s', \cdot) = 0$$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference = -501



$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

$$Q(s, a) = 3.0f_{DOT}(s, a) - 3.0f_{GST}(s, a)$$



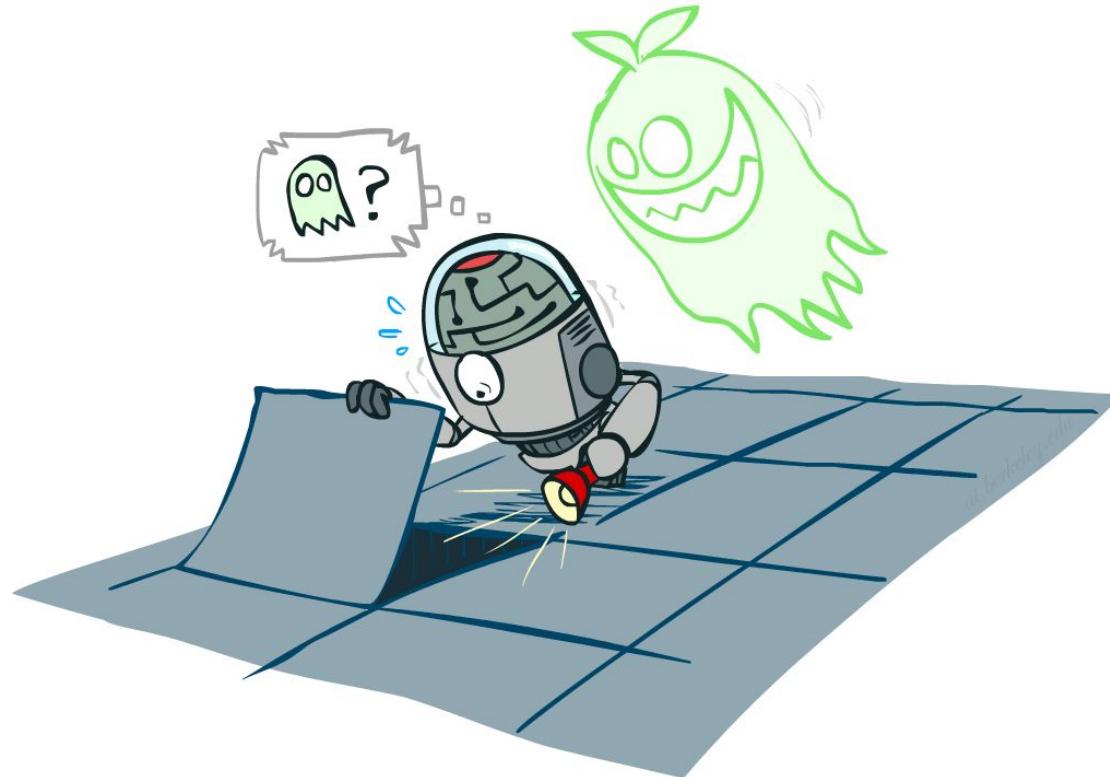
Chapter 4

Markov Models

COMP 3270
Artificial Intelligence

Dirk Schnieders

Uncertainty

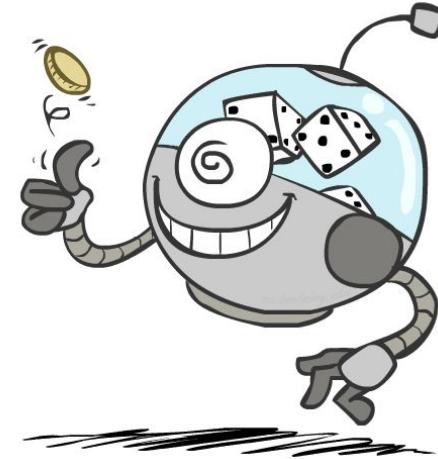


Probabilistic Reasoning

- This lecture is a bit dry
- Motivation:
 - AI History
 - Applications
 - Diagnosis
 - Physician measures symptoms and infer disease
 - IT experts diagnose printer problem
 - Speech recognition: Noisy signal, people have different pronunciations of words
 - Tracking objects: Combine noisy measurements from multiple sensors to estimate position
 - Genetics: Reproduction
 - Error correcting codes: Electrical signals are noisy

Refresher: Probability

- Random Variables
- Joint and Marginal Distributions
- Conditional Distribution
- Product Rule, Chain Rule, Bayes' Rule
- Inference
- Independence



The content in this refresher is important. In this and some of the following chapters we assume that you have the above background.

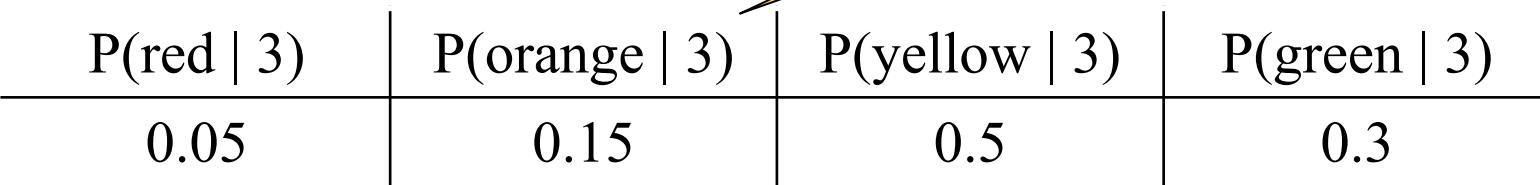
Running Example: Ghostbusters



Running Example: Ghostbusters

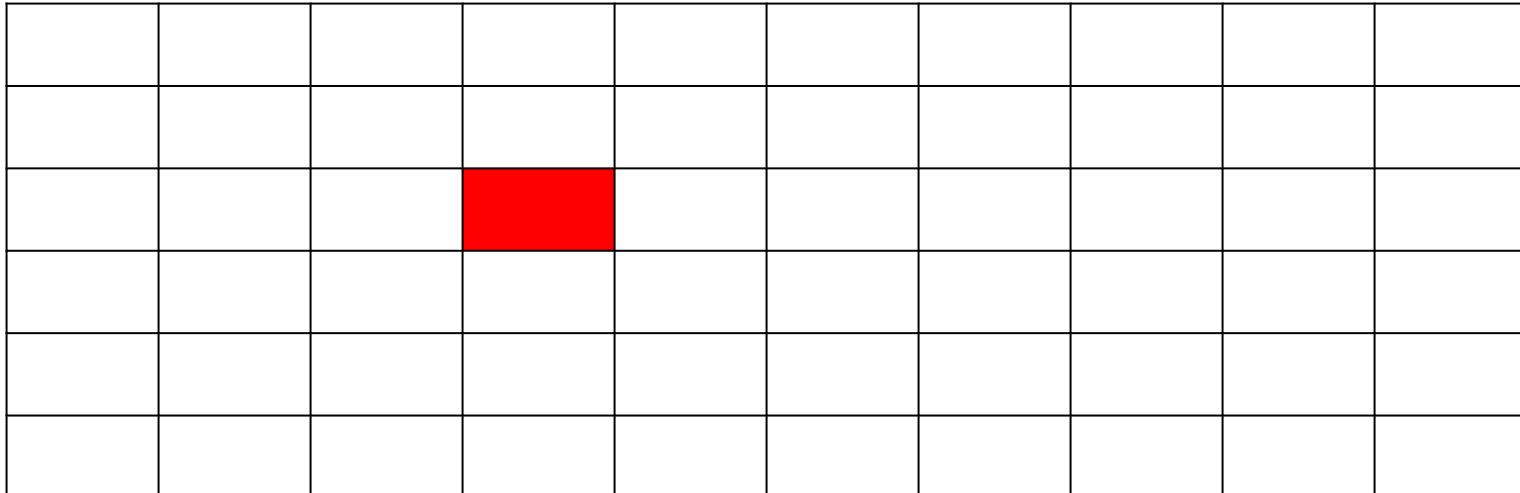
- A single ghost is hiding in the grid somewhere
- Noisy sensor readings tell how close a certain square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green
- We know $P(\text{color} \mid \text{distance})$
 - Example:

Each of the distances has a distribution similar to this one



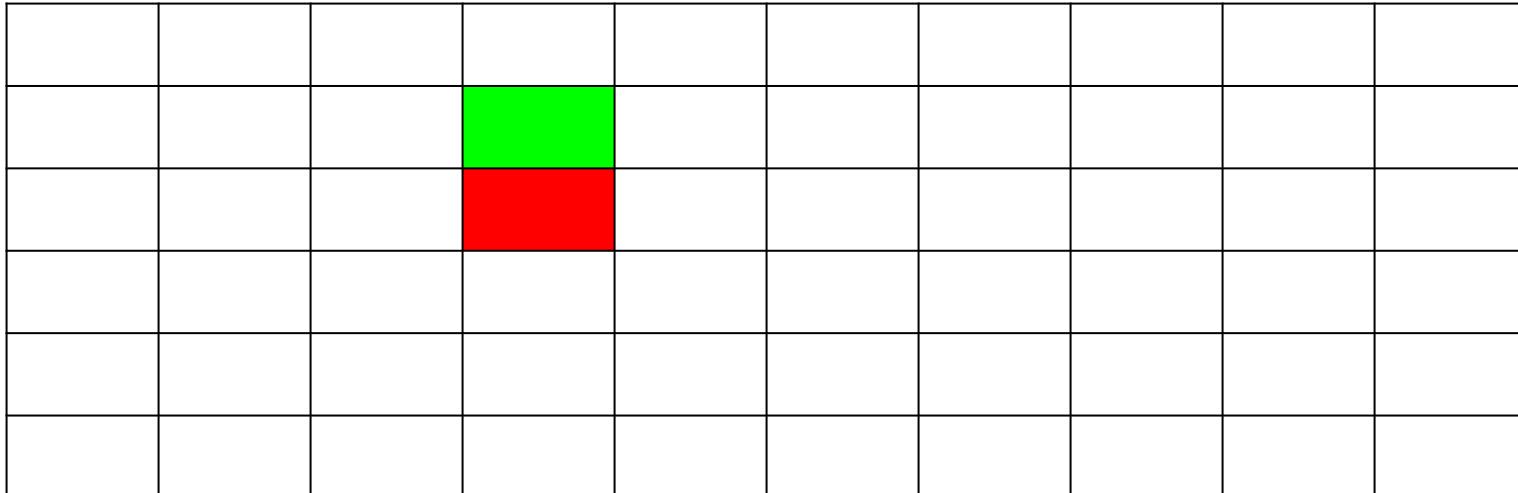
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



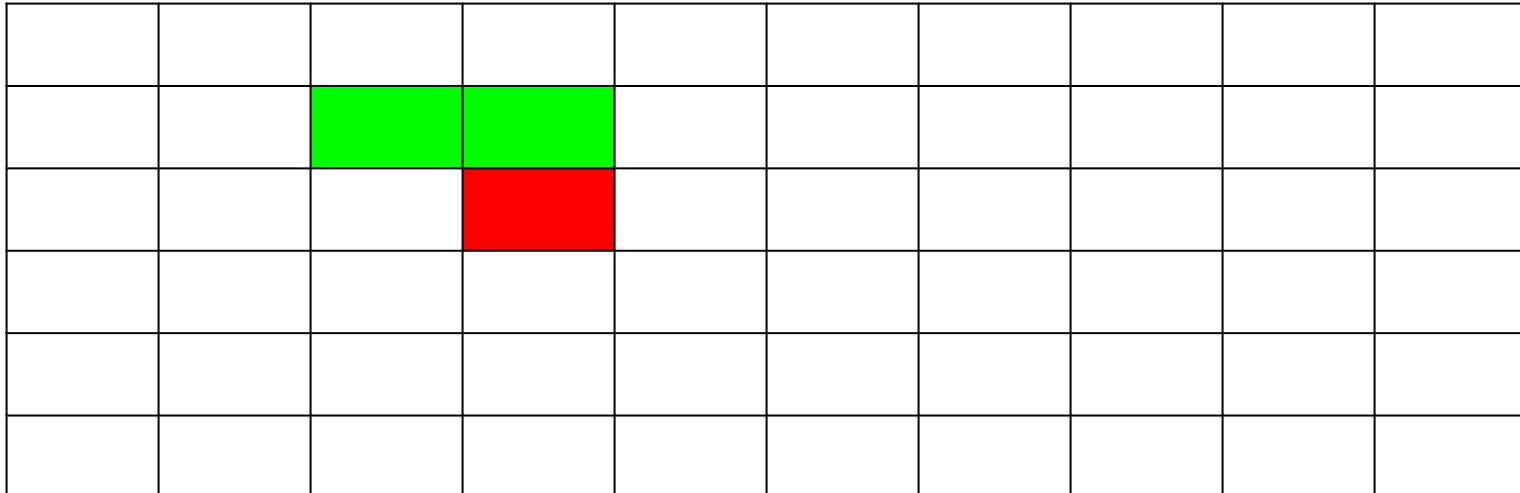
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

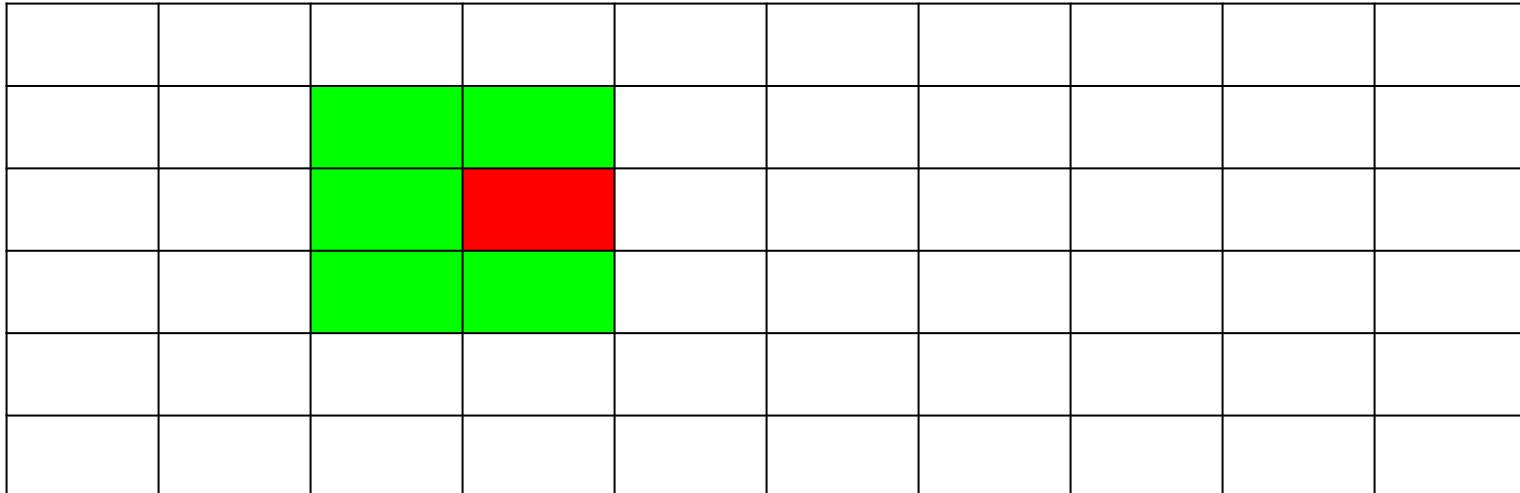
- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

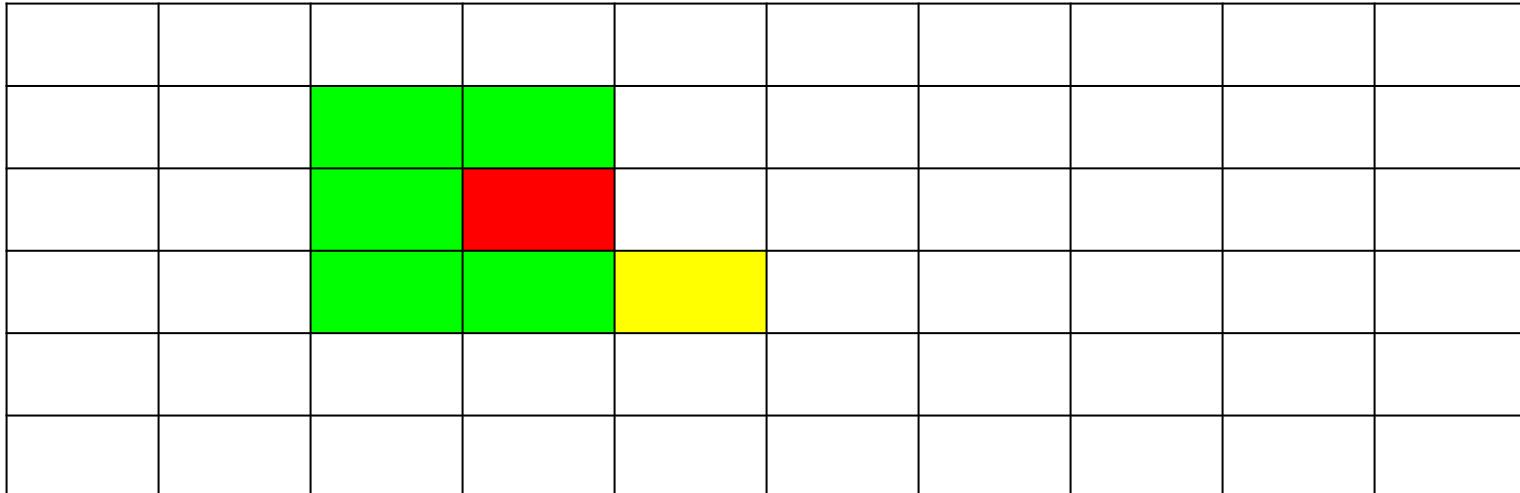
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



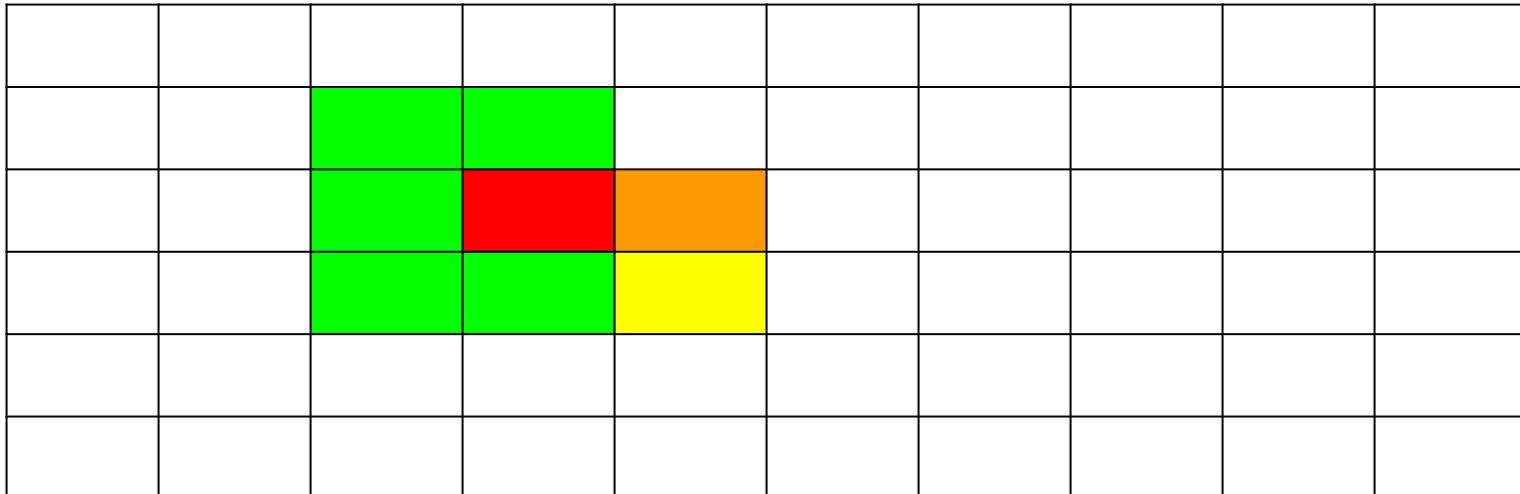
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



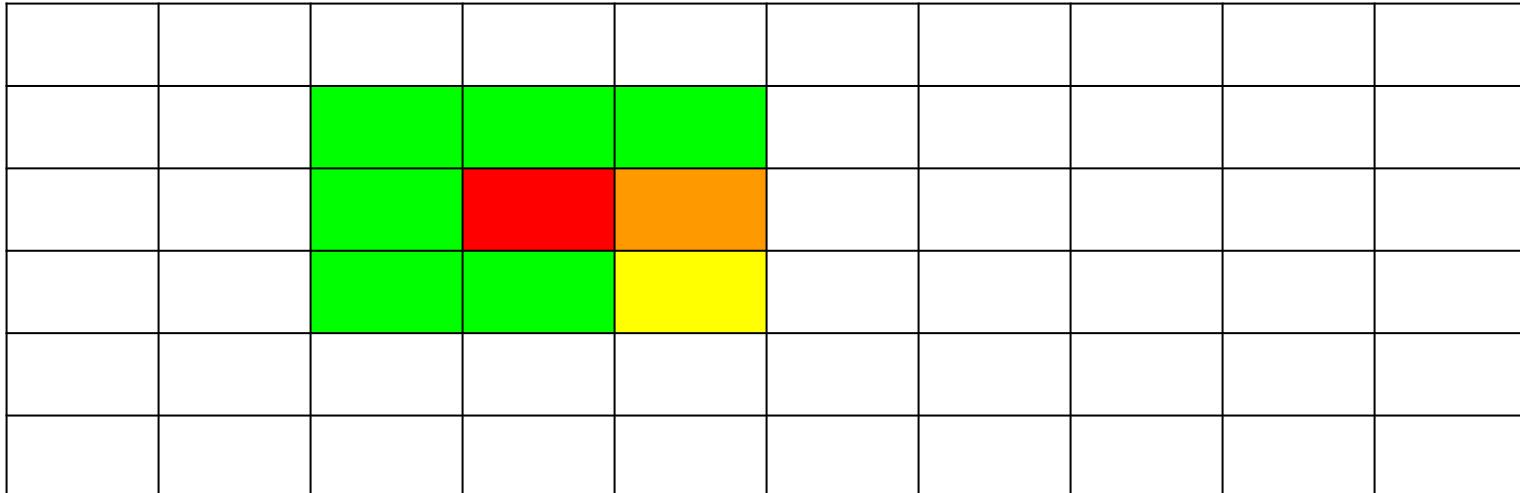
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



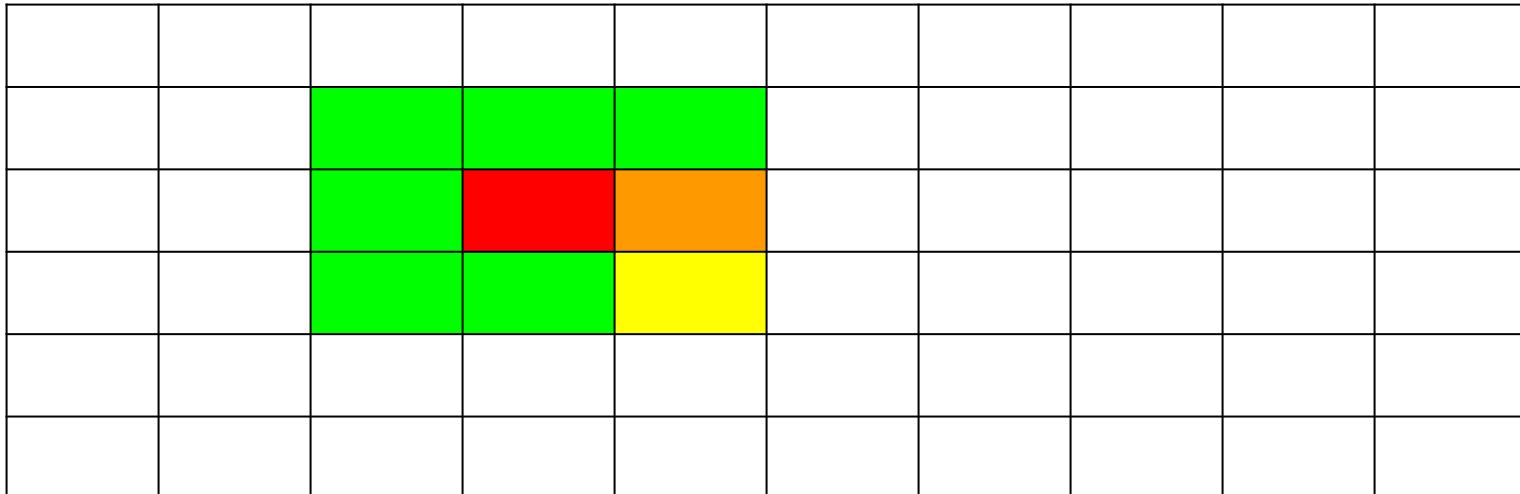
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



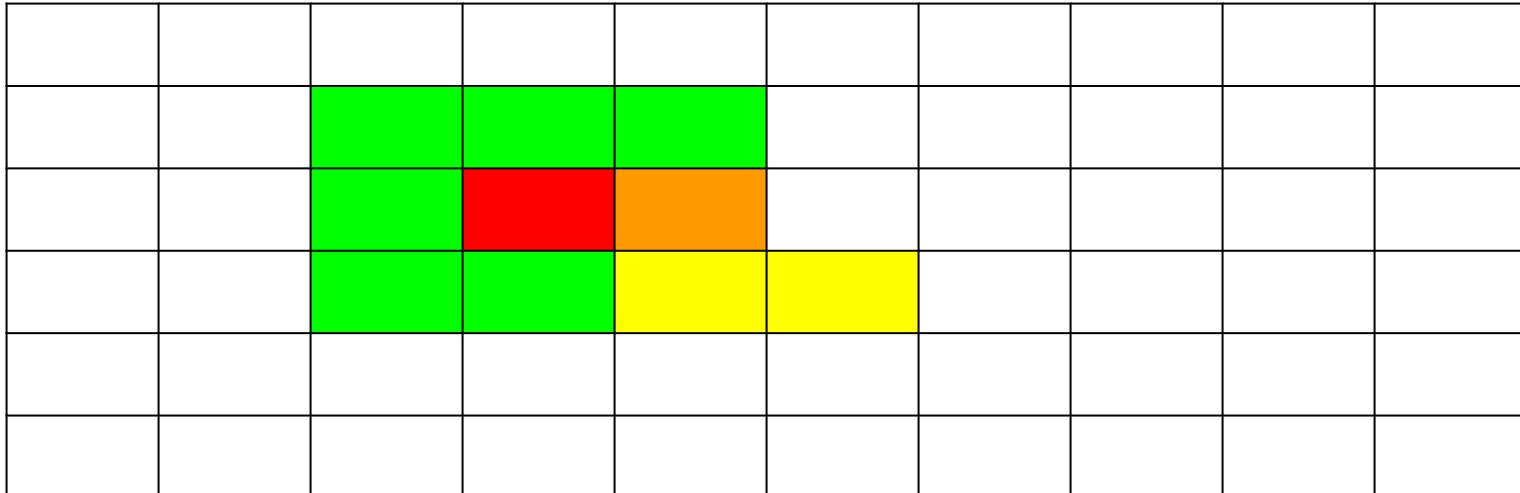
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



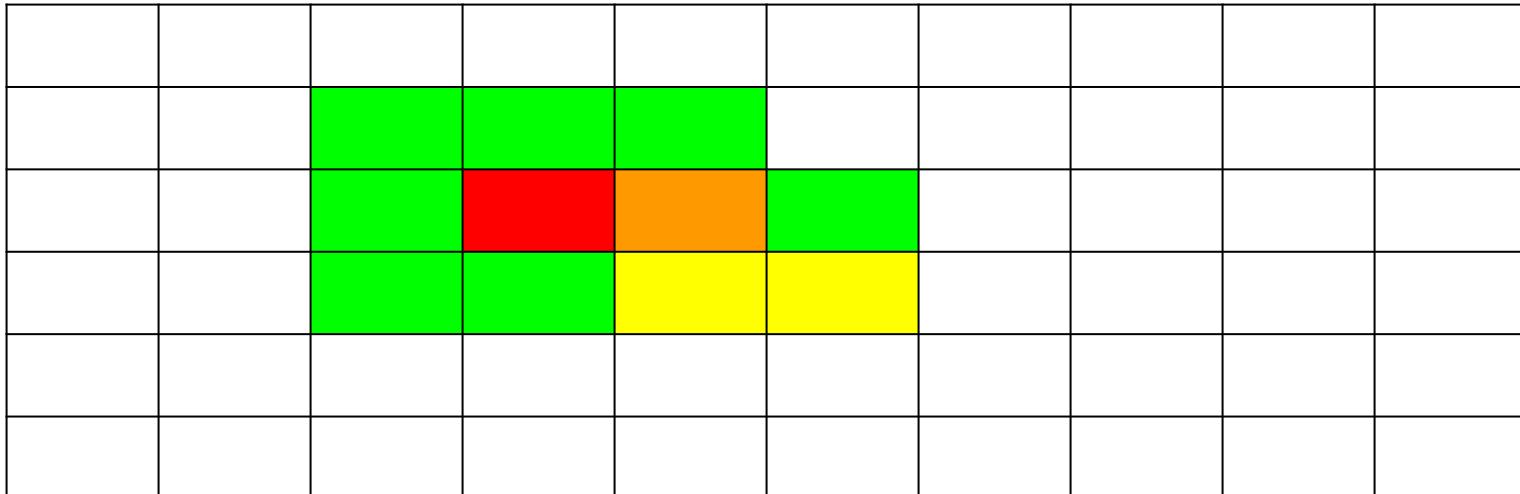
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



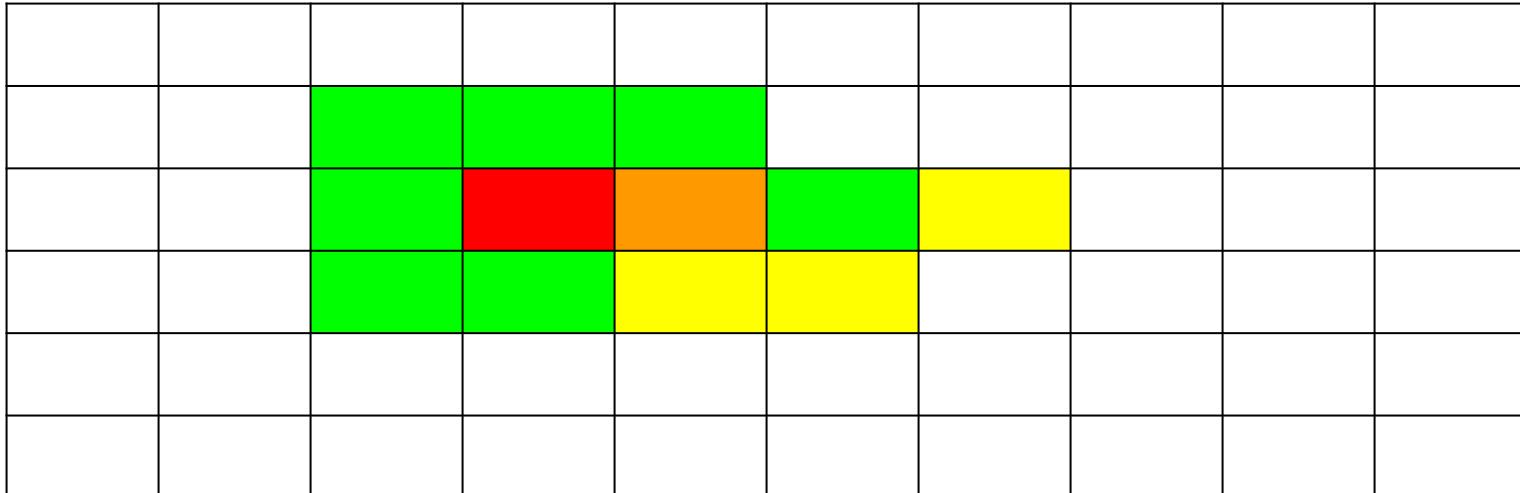
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



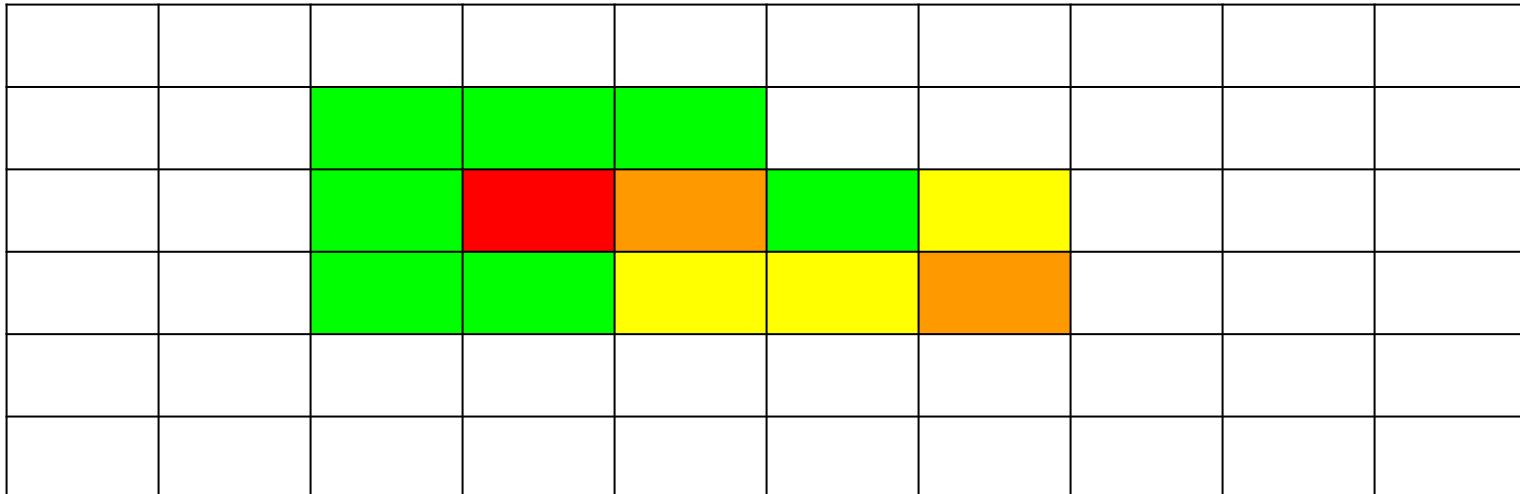
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost

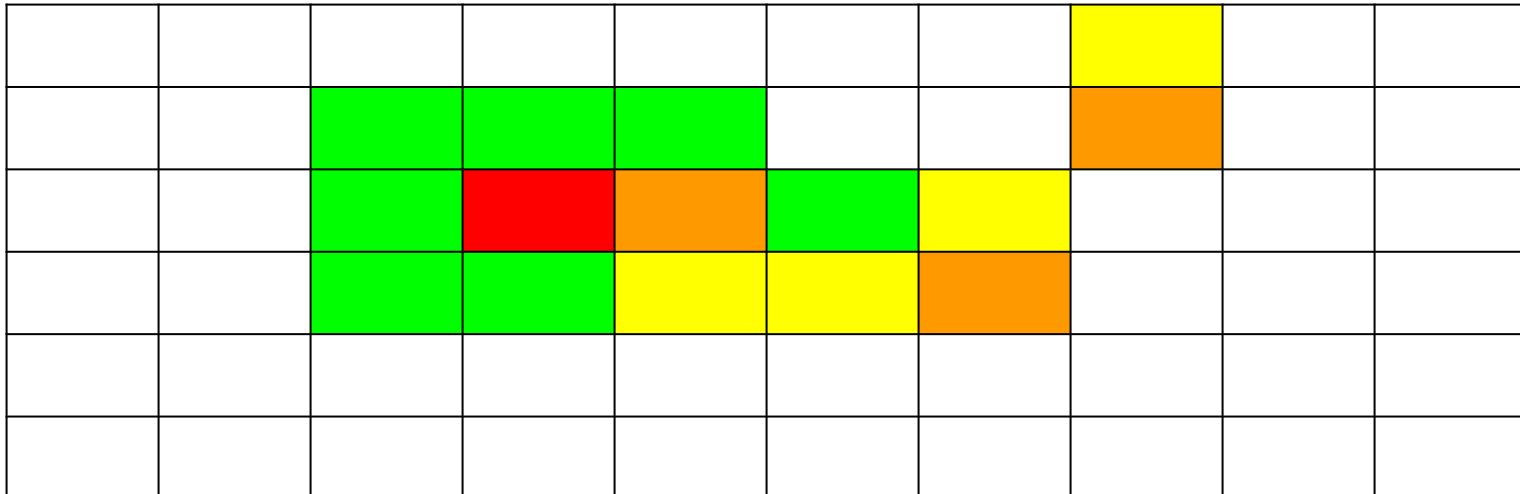
- On the ghost: red
- 1 or 2 away: orange
- 3 or 4 away: yellow
- 5+ away: green

The grid shows sensor readings for a 10x10 environment. The ghost is located at position (3, 3). The sensor readings are as follows:

- (3, 3) is red (On the ghost).
- (2, 3) is orange (1 or 2 away).
- (4, 3) is orange (1 or 2 away).
- (3, 2) is orange (1 or 2 away).
- (3, 4) is orange (1 or 2 away).
- (2, 2) is green (5+ away).
- (2, 4) is green (5+ away).
- (4, 2) is green (5+ away).
- (4, 4) is green (5+ away).
- All other squares are white (green, 5+ away).

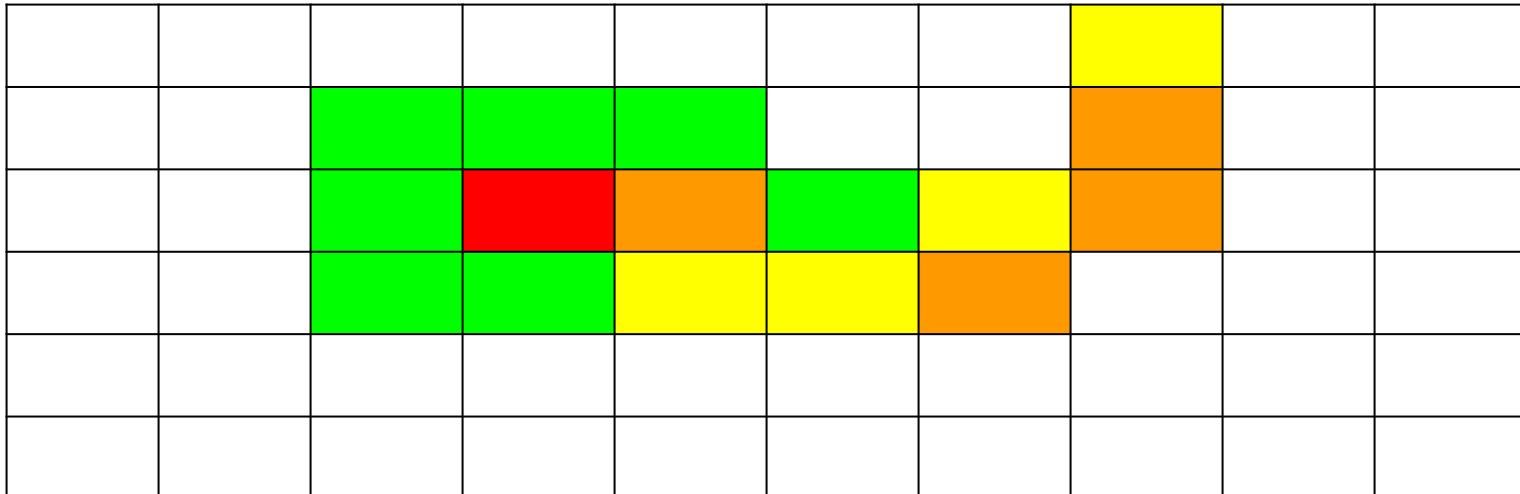
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



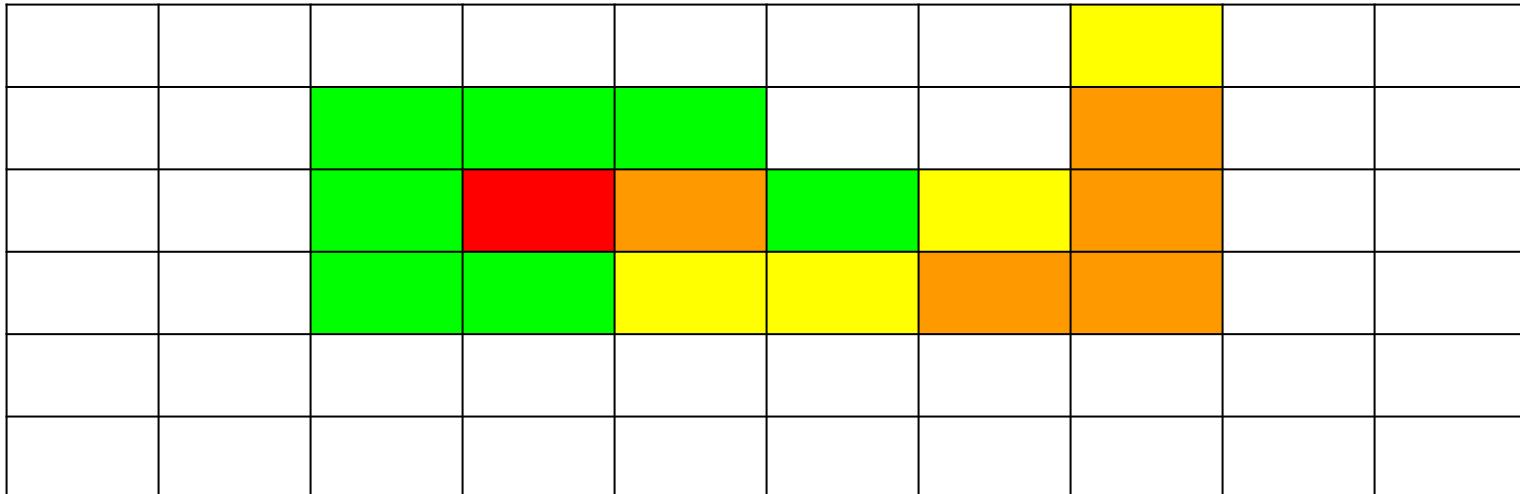
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

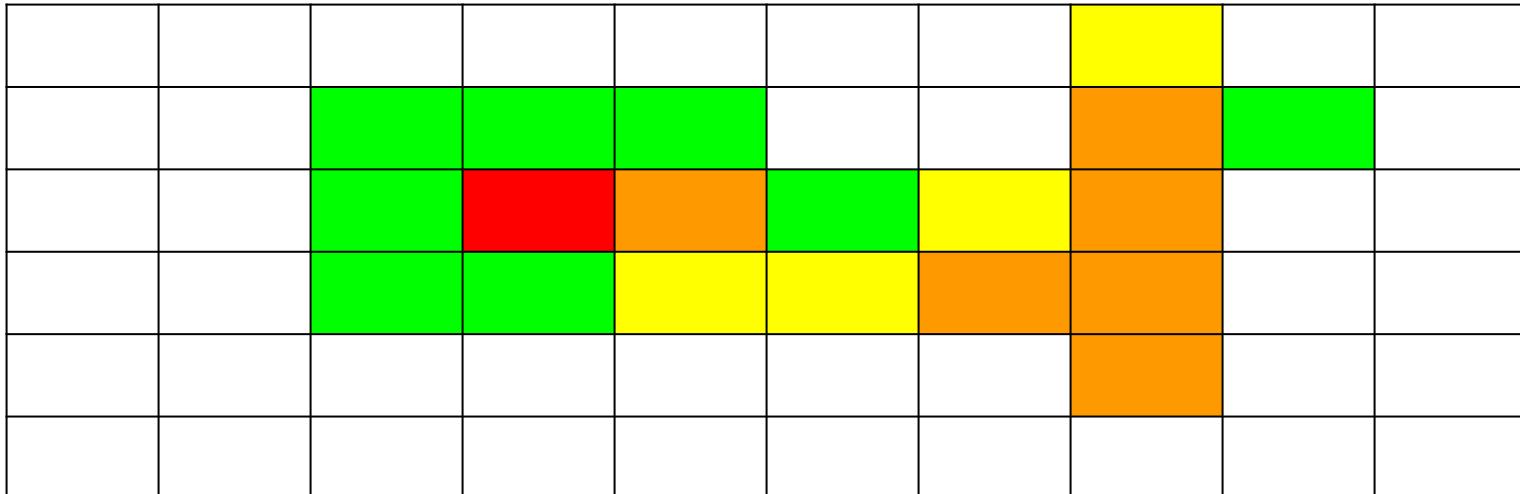
- Sensor readings tell how close a square is to the ghost

- On the ghost: red
- 1 or 2 away: orange
- 3 or 4 away: yellow
- 5+ away: green

A 10x10 grid representing sensor readings for a ghost. The grid shows a pattern of colors: green, red, orange, yellow, and white. The red square is at (3,3), orange squares are at (3,4), (4,3), (4,4), and (6,3), yellow squares are at (4,5), (5,4), (5,5), and (6,4), and green squares are at (2,2), (2,3), (2,4), (3,2), (3,5), (4,2), (4,5), (5,2), (5,3), (5,4), (6,2), (6,3), and (6,4). White squares represent 5+ away from the ghost.

Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

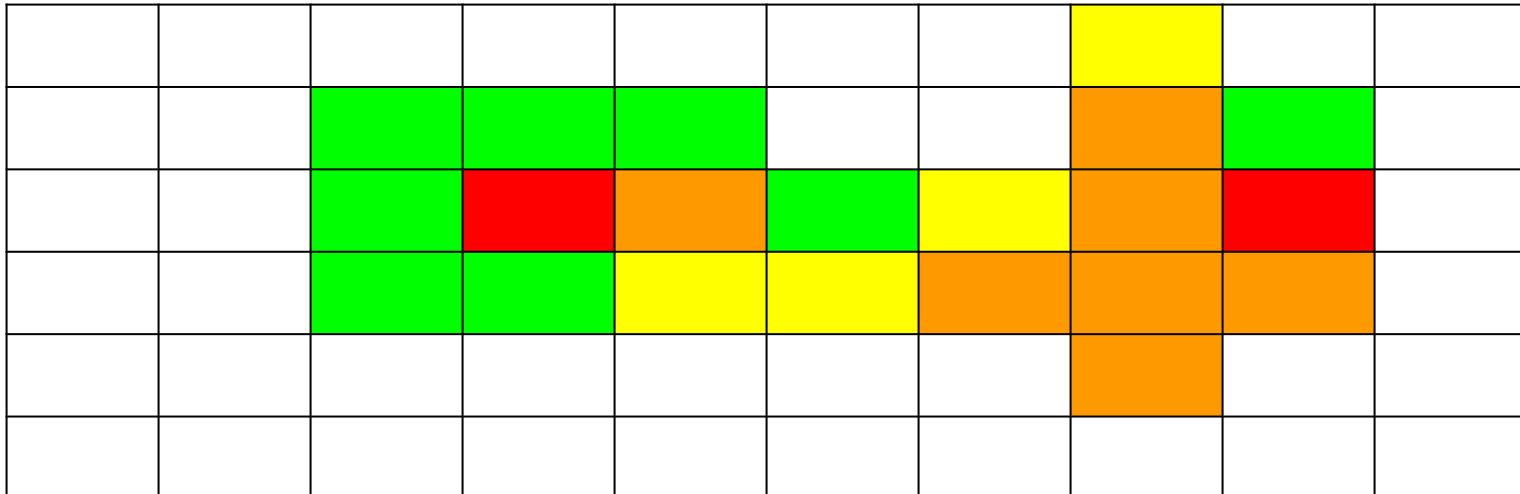


Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

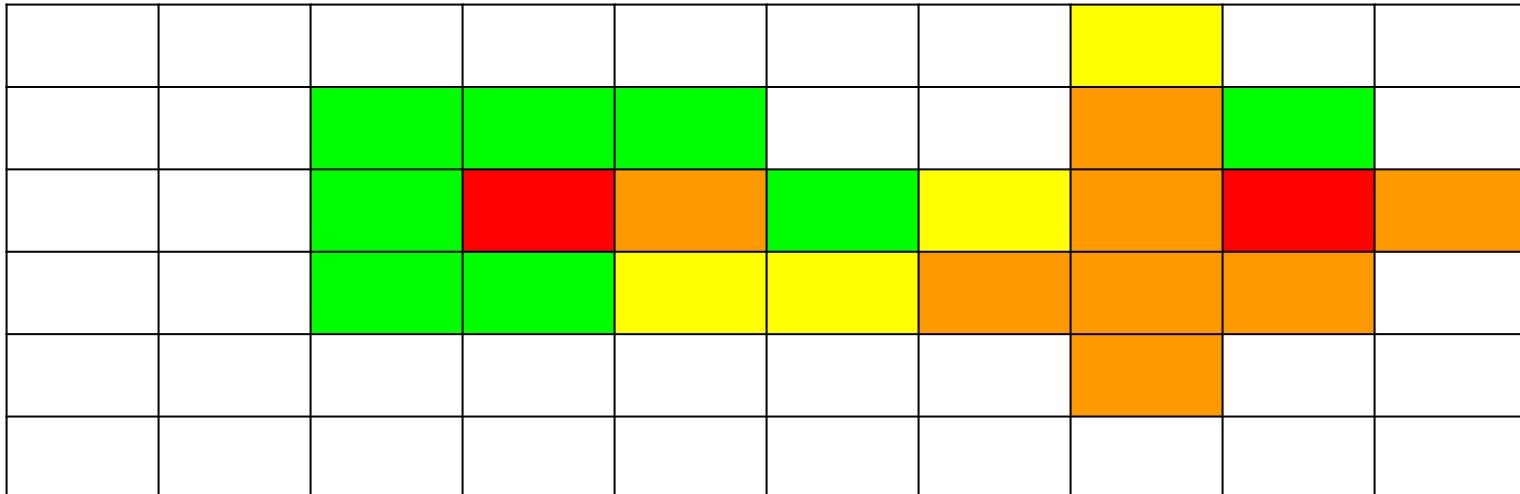
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



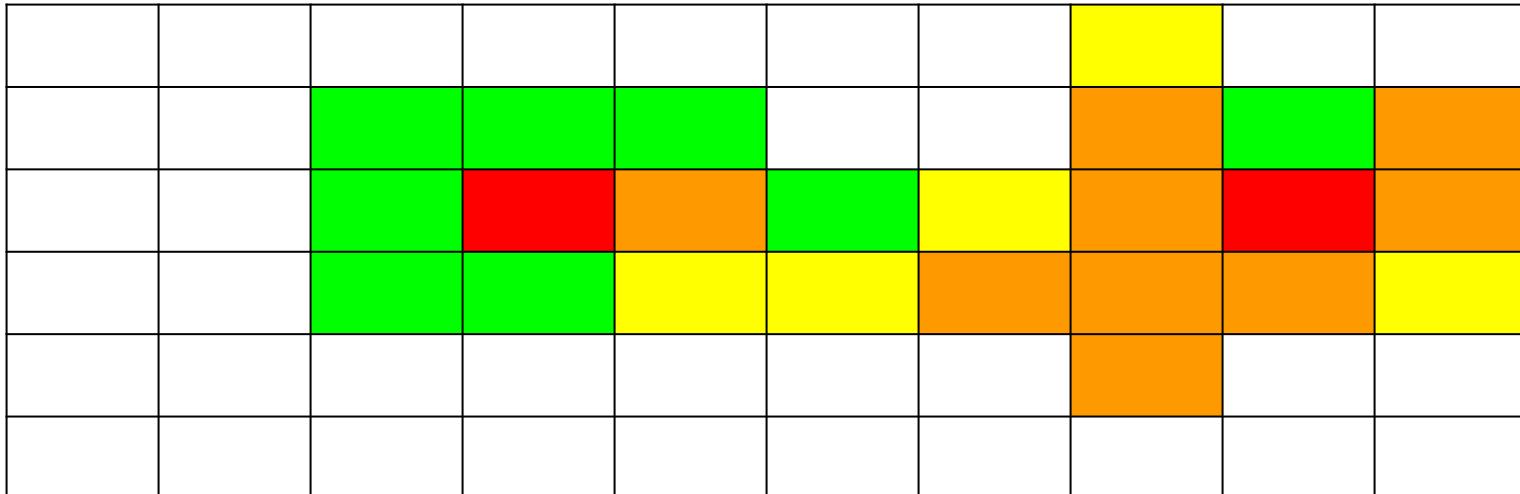
Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green



Running Example: Ghostbusters

- A single ghost is hiding in the grid somewhere
- Noisy sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green
- We know $P(\text{color} \mid \text{distance})$
 - Example:

$P(\text{red} \mid 3)$	$P(\text{orange} \mid 3)$	$P(\text{yellow} \mid 3)$	$P(\text{green} \mid 3)$
0.05	0.15	0.5	0.3

Uncertainty

- General situation:
 - Observed variables (evidence)
Agent knows certain things about the state of the world
(e.g., sensor readings or symptoms)
 - Unobserved variables
Agent needs to reason about other aspects
(e.g. where an object is or what disease is present)
 - Model
Agent knows something about how the known variables relate to the unknown variables
- Probabilistic reasoning gives us a framework for managing our beliefs and knowledge
- Assume model is given (in this chapter)

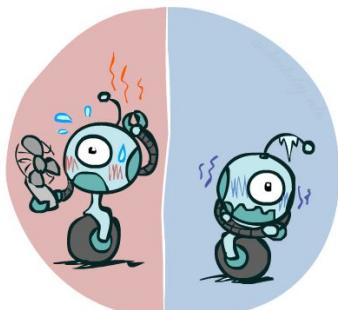
Random Variables

- A random variable is some aspect of the world about which we (may) have uncertainty
 - R = Is it raining?
 - T = Is it hot or cold?
 - D = How long will it take to drive to work?
 - L = Where is the ghost?
- We denote random variables with capital letters
- Like variables in a CSP, random variables have domains
 - R in {true, false} (often write as $\{+r, -r\}$)
 - T in {hot, cold}
 - D in $[0, \infty)$
 - L in possible locations, maybe $\{(0,0), (0,1), \dots\}$

Probability Distributions

- Associate a probability with each value

Temperature



T	P(T)
hot	0.5
cold	0.5

Weather



W	P(W)
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

Probability Distributions

- Unobserved random variables have distributions
 - A distribution is a table of probabilities of values
- A probability (lower case value) is a single number
 - $P(W = \text{rain}) = 0.1$
- The following must be true

$$\forall x \ P(X = x) \geq 0 \quad \sum_x P(X = x) = 1$$

- Shorthand notation (ok if all domain entries are unique)
 - $P(\text{hot}) = P(T = \text{hot})$
 - $P(\text{cold}) = P(T = \text{cold})$
 - $P(\text{rain}) = P(W = \text{rain})$

W	P(W)
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

T	P(T)
hot	0.5
cold	0.5

Joint Distributions

- A joint distribution over a set of random variables: X_1, X_2, \dots, X_n specifies a real number for each assignment (or outcome)

- $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$
- $P(x_1, x_2, \dots, x_n)$

- Must obey:

$$P(x_1, x_2, \dots, x_n) \geq 0$$

$$\sum_{(x_1, x_2, \dots, x_n)} P(x_1, x_2, \dots, x_n) = 1$$

T	W	$P(T,W)$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

- Size of distribution if n variables with domain sizes d ?
 - For all but the smallest distributions, impractical to write out!

Probabilistic Models

- A probabilistic model is a joint distribution over a set of random variables
- Probabilistic models:
 - (Random) variables with domains
 - Assignments are called outcomes
 - Joint distributions: say whether assignments (outcomes) are likely
 - Normalized: sum to 1.0
 - Ideally: only certain variables directly interact
- Constraint satisfaction problems:
 - Variables with domains
 - Constraints: state whether assignments are possible
 - Ideally: only certain variables directly interact

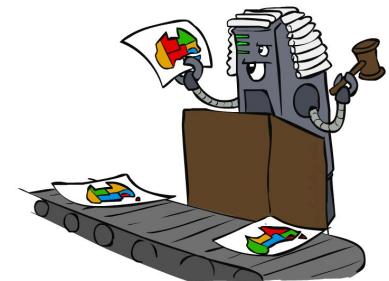
Distribution over T,W

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



Constraint over T,W

T	W	
hot	sun	T
hot	rain	F
cold	sun	F
cold	rain	T



Events

- An event is a set E of outcomes

$$P(E) = \sum_{(x_1 \dots x_n) \in E} P(x_1 \dots x_n)$$

- From a joint distribution, we can calculate the probability of any event
 - Probability that it's hot and sunny
 - Probability that it's hot
 - Probability that it's hot or sunny
- Typically, the events we care about are partial assignments, like $P(T = \text{hot})$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

Quiz: Events

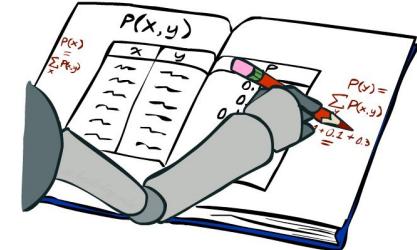
- $P(+x, +y) =$
- $P(+x) =$
- $P(-y \text{ OR } +x) =$

X	Y	$P(X, Y)$
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

Marginal Distributions

- Marginal distributions are sub-tables which eliminate variables
- Marginalization (summing out): Combine collapsed rows by adding

$$P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)$$



T	W	P(T, W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

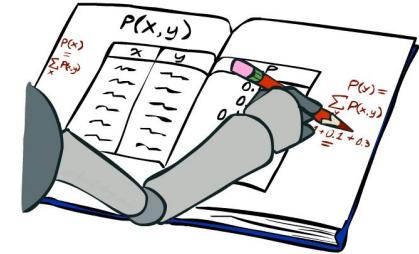
$$\rightarrow P(t) = \sum_s P(t, s) \rightarrow$$

T	P(T)
hot	0.5
cold	0.5

$$\rightarrow P(s) = \sum_t P(t, s) \rightarrow$$

W	P(W)
sun	0.6
rain	0.4

Quiz: Marginal Distributions



X	Y	P(X, Y)
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

→ $P(x) = \sum_y P(x, y)$ →

X	P(X)
+x	
-x	

→ $P(y) = \sum_x P(x, y)$ →

Y	P(Y)
+y	
-y	

Conditional Probabilities

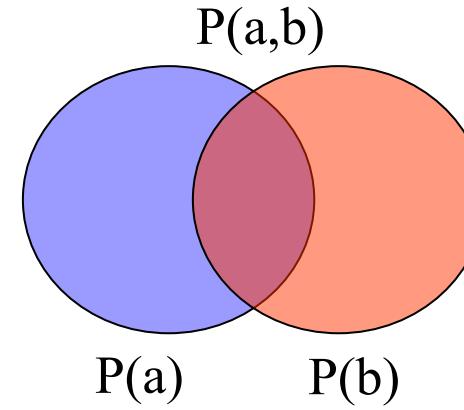
- A simple relation between joint and conditional probabilities
 - Important!
 - If we measure something, what does that tell us about the things we don't measure?

The probability
of having $A=a$
given $B=b$

The probability
of having $A=a$
and $B=b$

$$P(a|b) = \frac{P(a, b)}{P(b)}$$

The probability
of having $B=b$



Conditional Probabilities - Example

$$P(a|b) = \frac{P(a,b)}{P(b)}$$

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(W = s|T = c) = ???$$

$$P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{0.2}{0.5} = 0.4$$

$$\begin{aligned} &= P(W = s, T = c) + P(W = r, T = c) \\ &= 0.2 + 0.3 = 0.5 \end{aligned}$$

Quiz: Conditional Probabilities

$$P(a|b) = \frac{P(a,b)}{P(b)}$$

X	Y	P(X,Y)
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

$$P(+x | +y) =$$

$$P(-x | +y) =$$

$$P(-y | +x) =$$

Conditional Distributions

$$P(a|b) = \frac{P(a,b)}{P(b)}$$

- Conditional distributions are probability distributions over some variables given fixed values of others

Conditional Distributions

Set of tables!

		$P(W T)$
		$P(W T = \text{hot})$
T	W	
	sun	0.8
		$P(W T = \text{cold})$
T	W	
	sun	0.4
		$P(W T = \text{hot})$
T	W	
	rain	0.6

Joint Distribution

T	W	$P(T,W)$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



Normalization Trick

- What we have done so far:

$$P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)}$$

Joint Distribution

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$\begin{aligned} &= \frac{P(W = s, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.2}{0.2 + 0.3} = 0.4 \end{aligned}$$

Conditional Distribution

W	P(W T = c)
sun	0.4
rain	0.6

$$\begin{aligned} P(W = r|T = c) &= \frac{P(W = r, T = c)}{P(T = c)} \\ &= \frac{P(W = r, T = c)}{P(W = s, T = c) + P(W = r, T = c)} \\ &= \frac{0.3}{0.2 + 0.3} = 0.6 \end{aligned}$$

Normalization Trick

Joint Distribution

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

SELECT the joint probabilities matching the evidence

$$P(W = s|T = c) =$$

$$P(W = r|T = c) =$$

T	W	P(c, W)
cold	sun	0.2
cold	rain	0.3

NORMALIZE the selection (make it sum to one)



W	P(W T = c)
sun	0.4
rain	0.6

Conditional Distribution

Quiz: Normalization Trick

X	Y	P(X, Y)
+x	+y	0.2
+x	-y	0.3
-x	+y	0.4
-x	-y	0.1

$P(X| -y) =$

SELECT the joint
probabilities
matching the
evidence



NORMALIZE the selection
(make it sum to one)



To Normalize

Sum to one

- (Dictionary) To bring or restore to a normal condition
- Procedure:
 - Step 1: Compute $Z = \text{sum over all entries}$
 - Step 2: Divide every entry by Z

W	P
sun	0.2
rain	0.3

Normalize

$Z = 0.5$

T	W	P
hot	sun	20
hot	rain	5
cold	sun	10
cold	rain	15

Normalize

$Z = 50$

Probabilistic Inference

- Probabilistic inference: compute a desired probability from other known probabilities (e.g. conditional from joint)
- We generally compute conditional probabilities
 - $P(\text{on time} \mid \text{no reported accidents}) = 0.90$
 - These represent the agent's beliefs given the evidence
- Probabilities change with new evidence:
 - $P(\text{on time} \mid \text{no accidents, 5 a.m.}) = 0.95$
 - $P(\text{on time} \mid \text{no accidents, 5 a.m., raining}) = 0.80$
 - Observing new evidence causes beliefs to be updated



Inference by Enumeration

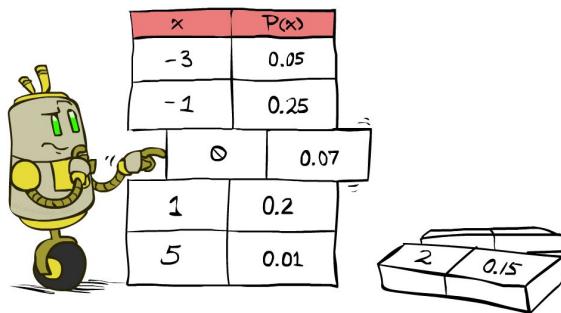
- General case

- Evidence variables: $E_1 \dots E_k = e_1 \dots e_k$
- Query variable: Q
- Hidden variables: $H_1 \dots H_r$

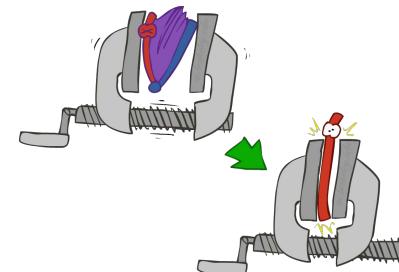
$X_1, X_2, \dots X_n$
All variables

We want: $P(Q|e_1 \dots e_k)$

Step 1: Select the entries
consistent with the evidence



Step 2: Sum out H to get joint of
Query and evidence



$$P(Q, e_1 \dots e_k) = \sum_{h_1 \dots h_r} P(Q, h_1 \dots h_r, e_1 \dots e_k)$$

$X_1, X_2, \dots X_n$

$$\times \frac{1}{Z}$$

$$Z = \sum_q P(Q, e_1 \dots e_k)$$

$$P(Q|e_1 \dots e_k) = \frac{1}{Z} P(Q, e_1 \dots e_k)$$

Inference by Enumeration - Example

- $P(W) = ?$
 - $Q = W, E = \emptyset, H = \{S, T\}$

2	W	P(W)
sun		0.65
rain		0.35

3	W	P(W)
sun		0.65
rain		0.35

	S	T	W	P(S, T, W)
1	summer	hot	sun	0.30
	summer	hot	rain	0.05
	summer	cold	sun	0.10
	summer	cold	rain	0.05
	winter	hot	sun	0.10
	winter	hot	rain	0.05
	winter	cold	sun	0.15
	winter	cold	rain	0.20

Inference by Enumeration - Example

- $P(W | \text{winter}) = ?$
- $Q = W, E = S, H = T$

2	W	P(W)
	sun	0.25
	rain	0.25

3	W	P(W)
	sun	0.5
	rain	0.5

	S	T	W	P(S, T, W)
1	summer	hot	sun	0.30
	summer	hot	rain	0.05
	summer	cold	sun	0.10
	summer	cold	rain	0.05
	winter	hot	sun	0.10
	winter	hot	rain	0.05
	winter	cold	sun	0.15
	winter	cold	rain	0.20

Inference by Enumeration - Example

- $P(W | \text{winter, hot}) = ?$
- $Q = W, E = S, T, H = \emptyset$

2	W	P(W)
	sun	0.10
	rain	0.05

3	W	P(W)
	sun	0.67
	rain	0.33

	S	T	W	P(S, T, W)
1	summer	hot	sun	0.30
	summer	hot	rain	0.05
	summer	cold	sun	0.10
	summer	cold	rain	0.05
	winter	hot	sun	0.10
	winter	hot	rain	0.05
	winter	cold	sun	0.15
	winter	cold	rain	0.20

Inference by Enumeration

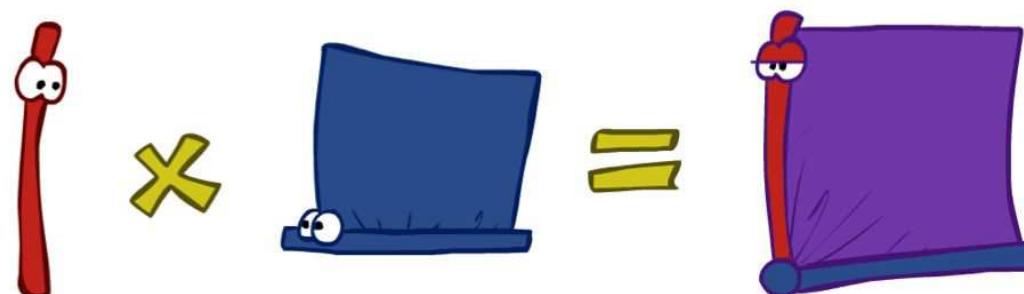
- Worst-case time complexity $O(d^n)$
- Space complexity $O(d^n)$ to store the joint distribution

The Product Rule

- Sometimes have conditional distributions but want the joint

$$P(x|y) = \frac{P(x,y)}{P(y)} \quad \longleftrightarrow \quad P(y)P(x|y) = P(x,y)$$

Definition of
Conditional
Probability



The Product Rule

$$P(y)P(x|y) = P(x, y)$$

- Example

T	P(W)
hot	0.5
cold	0.5

W	P(W hot)
sun	0.8
rain	0.2

W	P(W cold)
sun	0.4
rain	0.6

W	T	P(W,T)
hot	sun	.5 * .8 = .4
hot	rain	.5 * .2 = .1
cold	sun	.5 * .4 = .2
cold	rain	.5 * .6 = .3

Quiz: Product Rule

$$P(y)P(x|y) = P(x,y)$$

	W	P(W)
sun	wet	0.8
rain	dry	0.2

D	P(D sun)
wet	0.1
dry	0.9

D	P(D rain)
wet	0.7
dry	0.3

D	W	P(D,W)
wet	sun	
dry	sun	
wet	rain	
dry	rain	

The Chain Rule

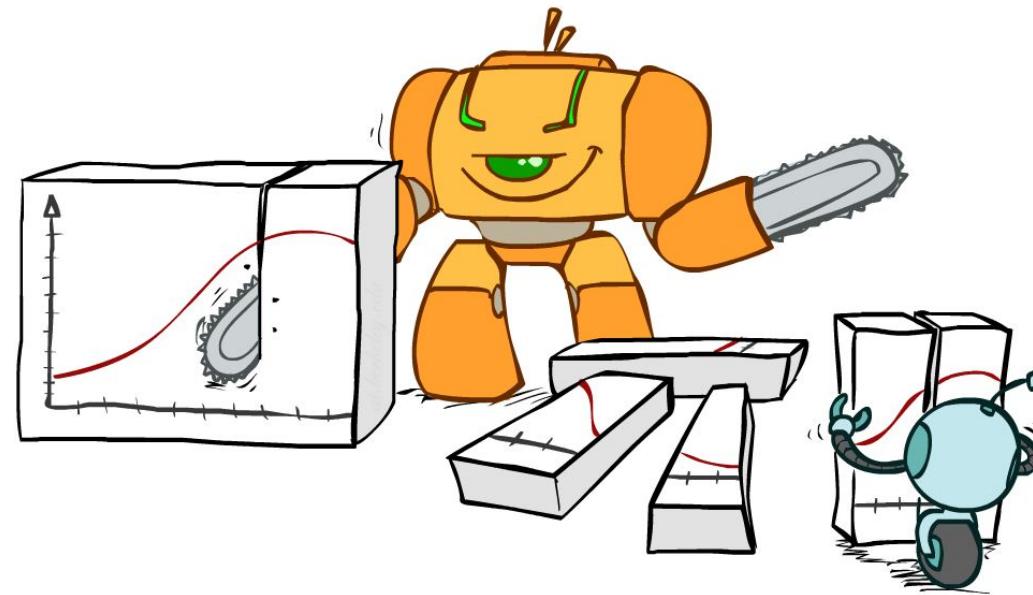
- More generally, can always write any joint distribution as an incremental product of conditional distributions

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

$$P(x_1, x_2, \dots, x_n) = \prod_i P(x_i|x_1 \dots x_{i-1})$$

- Why is this always true?

Bayes Rule



Bayes' Rule

$$P(y)P(x|y) = P(x,y)$$

- Two ways to factor a joint distribution over two variables:

$$P(x,y) = P(x|y)P(y) = P(y|x)P(x)$$

- Dividing, we get:

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

- Why is this at all helpful?
 - Lets us build one conditional from its reverse
 - Often one conditional is tricky but the other one is simple
 - Foundation of many systems we'll see later

Example: Inference with Bayes' Rule

$$P(x|y) = \frac{P(y|x)}{P(y)} P(x)$$

- Diagnostic probability from causal probability

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- M: meningitis, S: stiff neck
- Given:
 - $P(+m) = 0.0001$
 - $P(+s | +m) = 0.8$
 - $P(+s) = 0.01$
- $P(+m | +s) = ?$

Quiz: Bayes' Rule

$$P(x|y) = \frac{P(y|x)}{P(y)} P(x)$$

W	P(W)
sun	0.8
rain	0.2

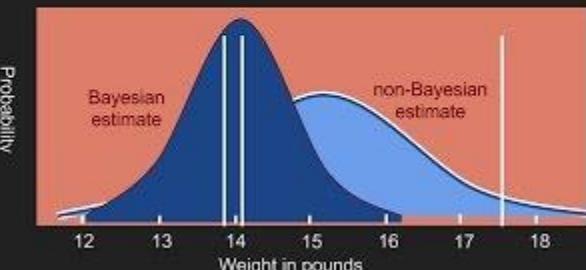
D	W	P(D W)
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3

Distribution!

$$P(W | \text{dry}) = ?$$

More on Bayes' Rule

How Bayesian Inference Works



by Brandon Rohrer

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$$

Ghostbusters, Revisited

- Let's say we have two distributions:
 - Prior distribution over ghost location: $P(G)$
 - Let this be uniform
 - Sensor reading model: $P(R|G)$
 - Given: we know what our sensors do
 - R = reading color measured at $(1,1)$
 - E.g. $P(R = \text{yellow} | G=(1,1)) = 0.1$
- We can calculate the posterior distribution $P(G|r)$ over ghost locations given a reading using Bayes' rule:

$$P(g|r) \propto P(r|g)P(g)$$

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

Ghostbusters with Probability

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

Ghostbusters with Probability

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

0.04	0.04	0.04	<0.01	<0.01	<0.01	0.04	0.04	0.04	0.04
0.04	0.04	<0.01	<0.01	<0.01	<0.01	<0.01	0.04	0.04	0.04
0.04	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.04	0.04
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.04
0.04	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.04	0.04
0.04	0.04	<0.01	<0.01	<0.01	<0.01	<0.01	0.04	0.04	0.04

Ghostbusters with Probability

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

0.02	0.02	0.02	<0.01	<0.01	0.01	0.13	0.01	<0.01	<0.01
0.02	0.02	<0.01	<0.01	<0.01	<0.01	0.01	0.13	0.01	<0.01
0.02	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.01	0.13	0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.01	0.13
0.02	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.02	0.13
0.02	0.02	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.02	0.02

Ghostbusters with Probability

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

Ghostbusters with Probability

- Sensor readings tell how close a square is to the ghost
 - On the ghost: red
 - 1 or 2 away: orange
 - 3 or 4 away: yellow
 - 5+ away: green

Independence

- Two variables are independent in a joint distribution if

$$P(X, Y) = P(X)P(Y)$$

$$X \perp\!\!\!\perp Y$$

$$\forall x, y P(x, y) = P(x)P(y)$$

- Says the joint distribution factors into a product of two simple ones
- Note: Usually, in practice, variables are not independent
- Can use independence as a modeling assumption
 - Simplifying assumption
 - Empirical joint distributions: at best “close” to independent
 - What could we assume for {Weather, Traffic, Cavity}?
- Independence is like something from CSPs: what?



Example: Independence check

- Given: Distribution
- Task: Check for independence

T	W	$P(T,W)$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

T	$P(T)$
hot	0.5
cold	0.5

W	$P(W)$
sun	0.6
rain	0.4

T	W	$P = P(T) P(W)$
hot	sun	0.3
hot	rain	0.2
cold	sun	0.3
cold	rain	0.2

Example: Independence

- N fair, independent coin flips

$$P(X_1)$$

H	0.5
T	0.5

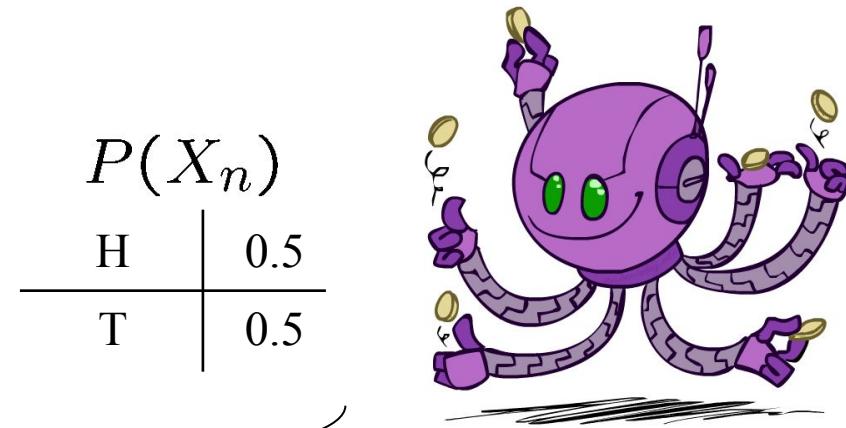
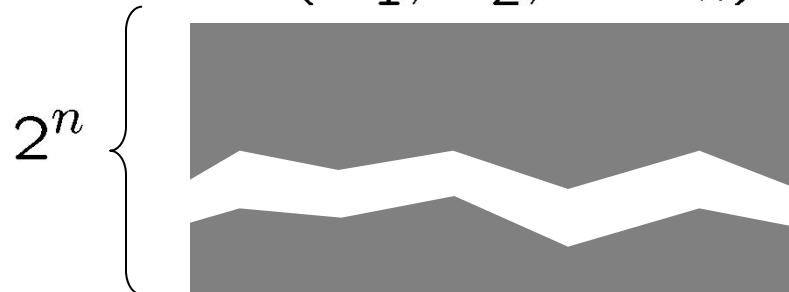
$$P(X_2)$$

H	0.5
T	0.5

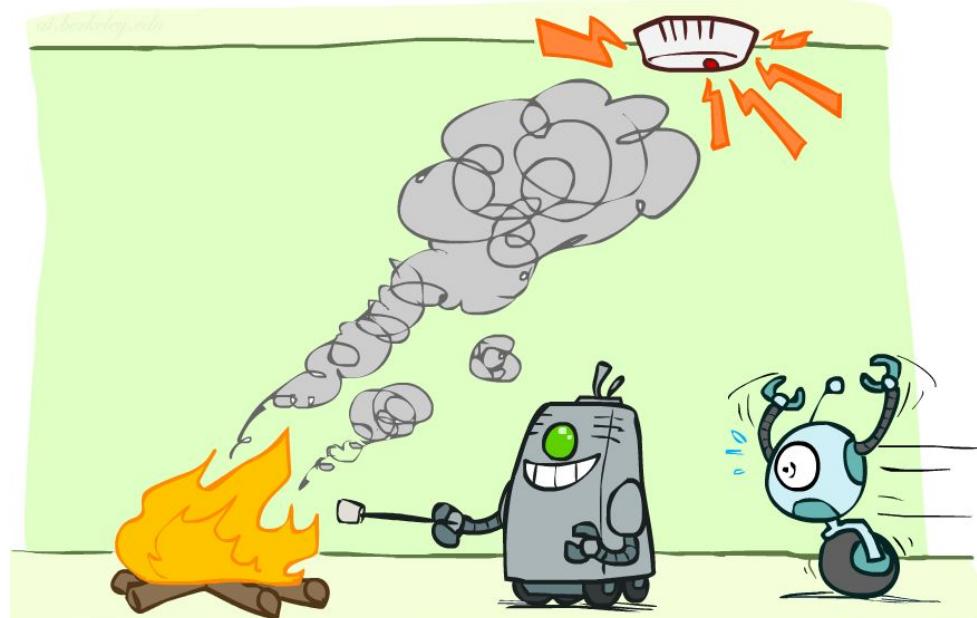
...

$$P(X_n)$$

H	0.5
T	0.5

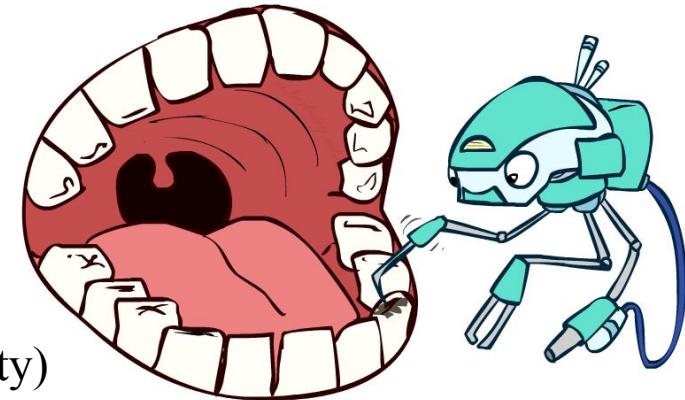


Conditional Independence



Conditional Independence

- $P(\text{Toothache}, \text{Cavity}, \text{Catch})$
- If I have a cavity, the probability that the probe catches in it doesn't depend on whether I have a toothache:
 - $P(+\text{catch} | +\text{toothache}, +\text{cavity}) = P(+\text{catch} | +\text{cavity})$
- The same independence holds if I don't have a cavity:
 - $P(+\text{catch} | +\text{toothache}, -\text{cavity}) = P(+\text{catch} | -\text{cavity})$
- Catch is conditionally independent of Toothache given Cavity:
 - $P(\text{Catch} | \text{Toothache}, \text{Cavity}) = P(\text{Catch} | \text{Cavity})$
- Equivalent statements:
 - $P(\text{Toothache} | \text{Catch}, \text{Cavity}) = P(\text{Toothache} | \text{Cavity})$
 - $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) P(\text{Catch} | \text{Cavity})$
 - One can be derived from the other easily



Conditional Independence

- Unconditional (absolute) independence very rare (why?)
- Conditional independence is our most basic and robust form of knowledge about uncertain environments

$$X \perp\!\!\!\perp Y | Z$$

- X is conditionally independent of Y given Z iff

$$\forall x, y, z : P(x, y | z) = P(x | z)P(y | z)$$

- or, equivalently, iff

$$\forall x, y, z : P(x | z, y) = P(x | z)$$

Quiz - Conditional Independence Proof

- Show that the statement

$$\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)$$

is equivalent to

$$\forall x, y, z : P(x|z, y) = P(x|z)$$

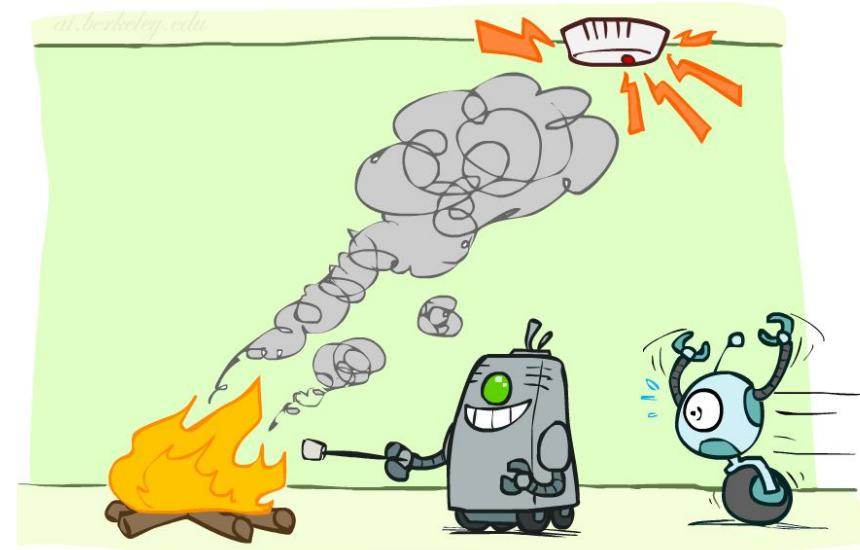
Quiz: Conditional Independence

- What about this domain
 - Traffic (T)
 - Umbrella (U)
 - Raining (R)



Quiz: Conditional Independence

- What about this domain
 - Fire (F)
 - Smoke (S)
 - Alarm (A)



Probability Recap

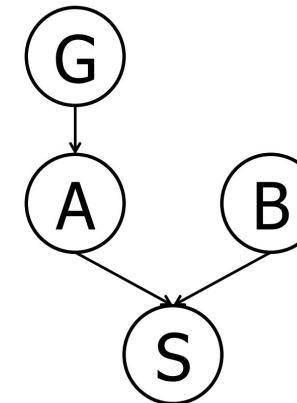
- Conditional probability $P(x|y) = \frac{P(x,y)}{P(y)}$
- Product rule $P(x,y) = P(x|y)P(y)$
- Chain rule
$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots \\ &= \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1}) \end{aligned}$$
- X, Y independent if and only if: $\forall x, y : P(x,y) = P(x)P(y)$
- X and Y are conditionally independent given Z if and only if:
$$\begin{aligned} \forall x, y, z : P(x, y|z) &= P(x|z)P(y|z) \\ \forall x, y, z : P(x|z, y) &= P(x|z) \end{aligned} \qquad \qquad \qquad X \perp\!\!\!\perp Y | Z$$

Quiz - G, S, A, B

- Suppose that a patient can have a symptom (S) that can be caused by two different diseases (A and B)
- It is known that the variation of gene G plays a big role in the manifestation of disease A
- The conditional probability tables for this situation are shown below
- Calculate the following
 - 1. $P(+g, +a, +b, +s)$
 - 2. $P(+a)$
 - 3. $P(+a| + b)$
 - 4. $P(+a| + s, + g)$
 - 5. $P(+g| + a)$
 - 6. $P(+g| + b)$

$\mathbb{P}(G)$	
$+g$	0.1
$-g$	0.9

$\mathbb{P}(A G)$		
$+g$	$+a$	1.0
$+g$	$-a$	0.0
$-g$	$+a$	0.1
$-g$	$-a$	0.9



$\mathbb{P}(B)$	
$+b$	0.4
$-b$	0.6

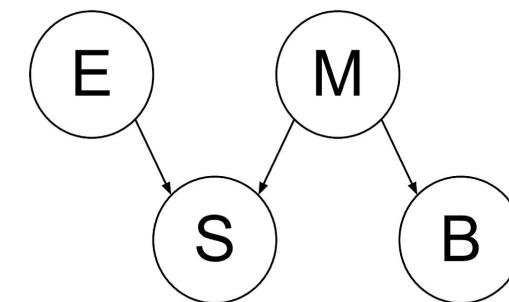
$\mathbb{P}(S A, B)$			
$+a$	$+b$	$+s$	1.0
$+a$	$+b$	$-s$	0.0
$+a$	$-b$	$+s$	0.9
$+a$	$-b$	$-s$	0.1
$-a$	$+b$	$+s$	0.8
$-a$	$+b$	$-s$	0.2
$-a$	$-b$	$+s$	0.1
$-a$	$-b$	$-s$	0.9

Quiz - E, S, M, B

- A smell of sulphur (S) can be caused either by rotten eggs (E) or as a sign of the doom brought by the Mayan Apocalypse (M)
- The Mayan Apocalypse also causes the oceans to boil (B)
- The conditional probability tables for this situation are shown below
- Calculate the following
 - $P(-e, -s, -m, -b)$
 - $P(+b)$
 - $P(+m| + b)$
 - $P(+m| + s, +b, +e)$
 - $P(+e| + m)$

$P(E)$	
$+e$	0.4
$-e$	0.6

$P(S E, M)$			
$+e$	$+m$	$+s$	1.0
$+e$	$+m$	$-s$	0.0
$+e$	$-m$	$+s$	0.8
$+e$	$-m$	$-s$	0.2
$-e$	$+m$	$+s$	0.3
$-e$	$+m$	$-s$	0.7
$-e$	$-m$	$+s$	0.1
$-e$	$-m$	$-s$	0.9

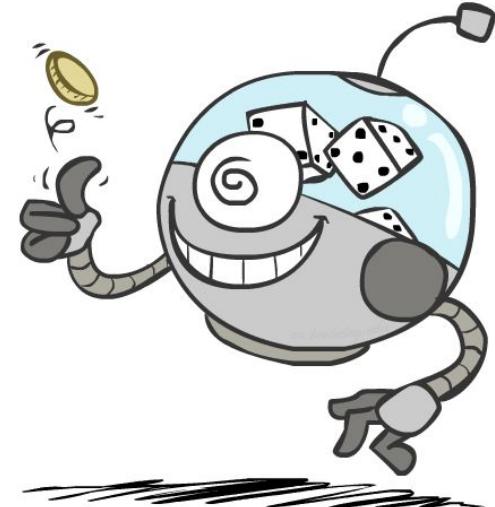


$P(M)$	
$+m$	0.1
$-m$	0.9

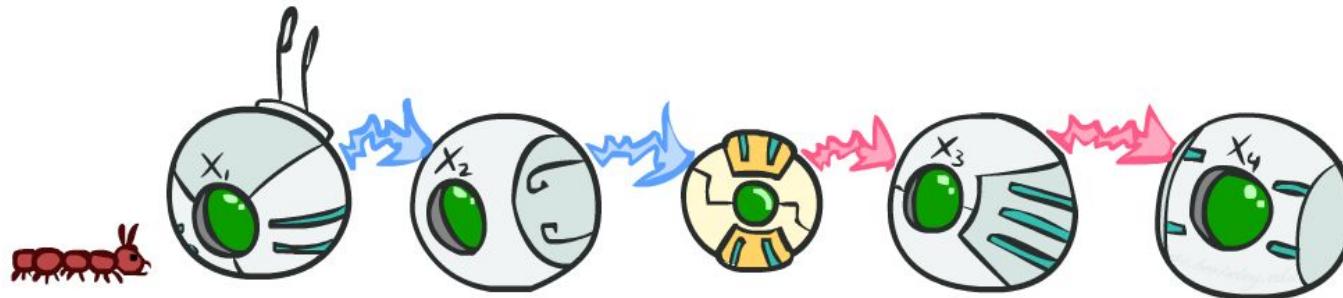
$P(B M)$		
$+m$	$+b$	1.0
$+m$	$-b$	0.0
$-m$	$+b$	0.1
$-m$	$-b$	0.9

Refresher: Probability

- Random Variables
- Joint and Marginal Distributions
- Conditional Distribution
- Product Rule, Chain Rule, Bayes' Rule
- Inference
- Independence



Markov Model



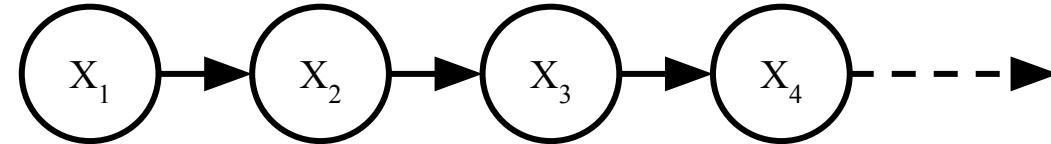
Introducing time (or space) into our models

Reasoning over Time or Space

- Often, we want to reason about a sequence of observations
 - Robot localization
 - From noisy readings of sensors
 - User attention
 - When is a good time to interrupt (popup-ad) a user?
 - Speech recognition
 - Inference on the acoustic sequence of words
 - Medical monitoring
 - Is there a medical emergency occurring ?
- Need to introduce time (or space) into our models

Markov Model

- Chain structure

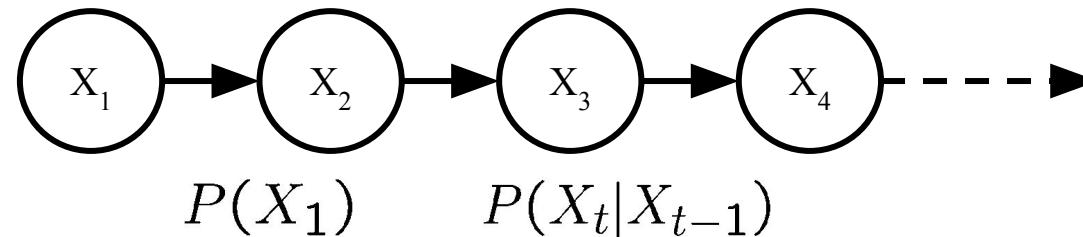


- Value of X at a given time is called the state
- We assume that future states depend only on the current state
 - not on previous states

$$P(X_1) \quad P(X_t | X_{t-1})$$

- Parameters: called transition probabilities or dynamics, specify how the state evolves over time (also, initial state probabilities)
 - Stationarity assumption
 - transition probabilities **the same at all times**
 - Same as MDP transition model, but no choice of action

Joint Distribution of a Markov Model



- Joint distribution:

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

- More generally:

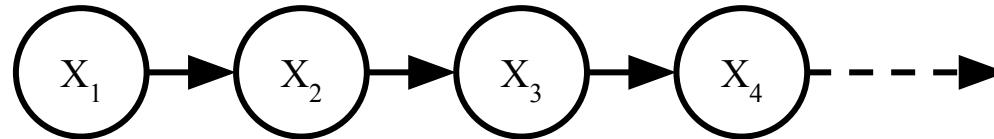
$$P(X_1, X_2, \dots, X_T) = P(X_1)P(X_2|X_1)P(X_3|X_2)\dots P(X_T|X_{T-1})$$

$$= P(X_1) \prod_{t=2}^T P(X_t|X_{t-1})$$

- Questions to be resolved:

- Does this indeed define a joint distribution?
- Can every joint distribution be factored this way, or are we making some assumptions about the joint distribution by using this factorization?

Chain Rule and Markov Models



- From the chain rule, every joint distribution over X_1, X_2, X_3, X_4 can be written as:

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)P(X_4|X_1, X_2, X_3)$$

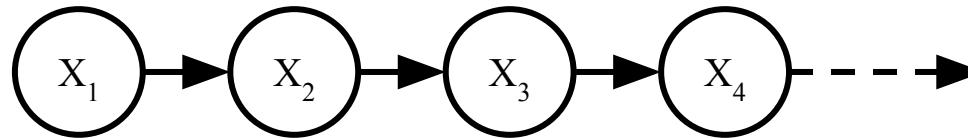
- Assuming that

$$X_3 \perp\!\!\!\perp X_1 \mid X_2 \quad \text{and} \quad X_4 \perp\!\!\!\perp X_1, X_2 \mid X_3$$

results in the expression

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

Chain Rule and Markov Models



- From the chain rule, every joint distribution over X_1, X_2, \dots, X_T can be written as:

$$P(X_1, X_2, \dots, X_T) = P(X_1) \prod_{t=2}^T P(X_t | X_1, X_2, \dots, X_{t-1})$$

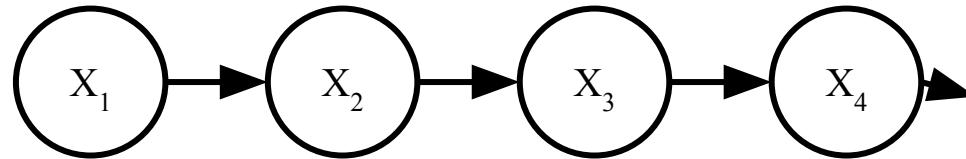
- Assuming that

$$X_t \perp\!\!\!\perp X_1, \dots, X_{t-2} \mid X_{t-1}$$

results in the expression

$$P(X_1, X_2, \dots, X_T) = P(X_1) \prod_{t=2}^T P(X_t | X_{t-1})$$

Quiz - Chain Rule and Markov Models



- We assumed $X_3 \perp\!\!\!\perp X_1 \mid X_2$ and $X_4 \perp\!\!\!\perp X_1, X_2 \mid X_3$
- Do we also have $X_1 \perp\!\!\!\perp X_3, X_4 \mid X_2$?

Markov Models Recap

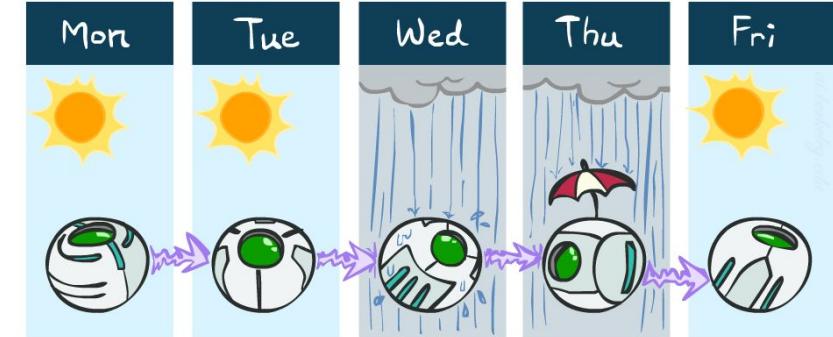
- Explicit assumption for all t: $X_t \perp\!\!\!\perp X_1, \dots, X_{t-2} \mid X_{t-1}$
- Consequence, joint distribution can be written as:

$$\begin{aligned} P(X_1, X_2, \dots, X_T) &= P(X_1)P(X_2|X_1)P(X_3|X_2)\dots P(X_T|X_{T-1}) \\ &= P(X_1) \prod_{t=2}^T P(X_t|X_{t-1}) \end{aligned}$$

- Implied conditional independencies:
 - Past variables independent of future variables given the present
 - i.e., if $t_1 < t_2 < t_3$ or $t_1 > t_2 > t_3$ then: $X_{t_1} \perp\!\!\!\perp X_{t_3} \mid X_{t_2}$
- Additional explicit assumption: $P(X_t \mid X_{t-1})$ is the same for all t

Example Markov Chain: Weather

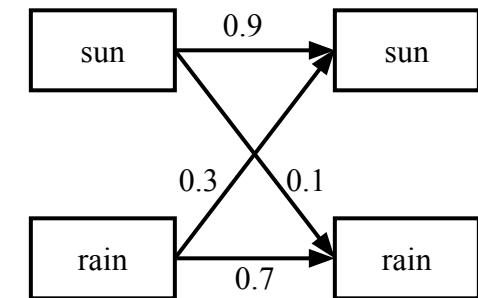
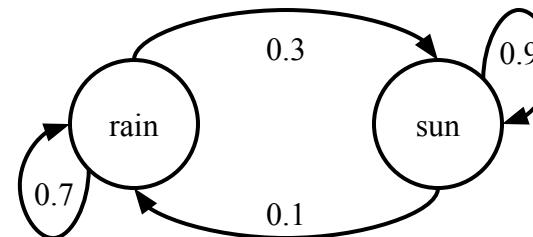
- States: $X = \{\text{rain, sun}\}$
- Initial distribution: 1.0 sun



Conditional Probability Table (CPT):

X_{t-1}	X_t	$P(X_t X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

Two new ways of representing the same CPT



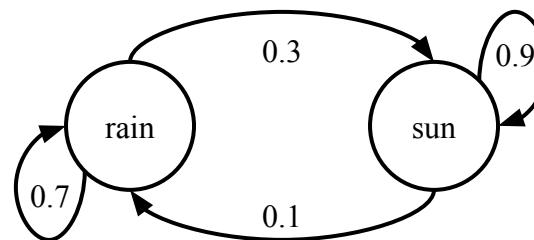
Example Markov Chain: Weather

- Initial distribution: 1.0 sun
- What is the probability distribution after one step?

Initial distribution: 1.0 sun

$$P(X_2 = \text{sun}) = ?$$

$$P(X_2 = \text{rain}) = ?$$



$$P(X_2 = \text{sun}) = P(X_2 = \text{sun} | X_1 = \text{sun})P(X_1 = \text{sun}) + P(X_2 = \text{sun} | X_1 = \text{rain})P(X_1 = \text{rain})$$

$$P(X_2 = \text{sun}) = .9 * 1.0 + 0.3 * 0.0 = 0.9$$

$$P(X_2 = \text{rain}) = P(X_2 = \text{rain} | X_1 = \text{sun})P(X_1 = \text{sun}) + P(X_2 = \text{rain} | X_1 = \text{rain})P(X_1 = \text{rain})$$

$$P(X_2 = \text{rain}) = .1 * 1.0 + 0.7 * 0.0 = 0.1$$

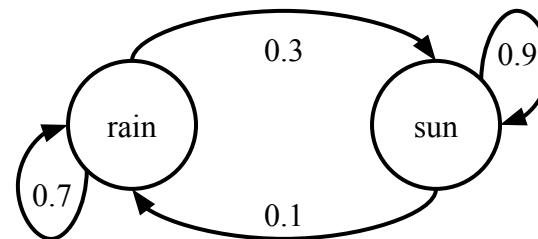
Quiz - Markov Chain: Weather

- Initial distribution: 1.0 sun
- What is the probability distribution after two steps?

Distribution after one step:

0.9 sun

0.1 rain

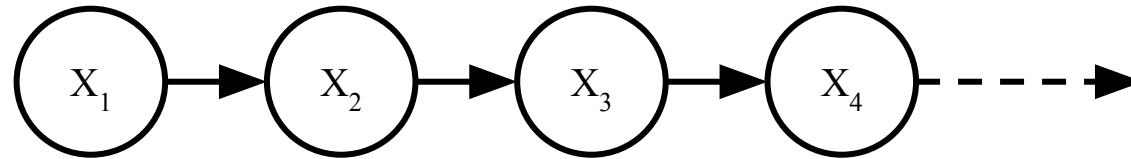


$$P(X_3 = \text{sun}) = ?$$

$$P(X_3 = \text{rain}) = ?$$

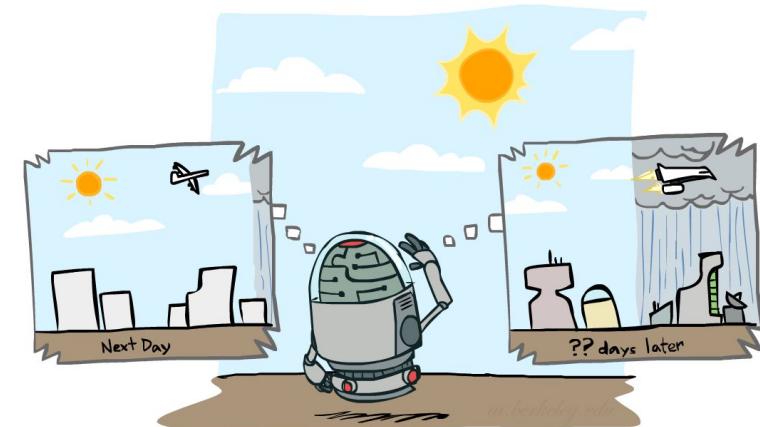
Mini-Forward Algorithm

- Question: What's $P(X)$ on some day t ?



$P(x_1) = \text{known}$

$$\begin{aligned} P(x_t) &= \sum_{x_{t-1}} P(x_{t-1}, x_t) \\ &= \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1}) \end{aligned}$$



Example Run of Mini-Forward Algorithm

- From initial observation of sun

$$\begin{array}{c} \left\langle \begin{array}{c} 1.0 \\ 0.0 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.9 \\ 0.1 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.84 \\ 0.16 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.804 \\ 0.196 \end{array} \right\rangle \longrightarrow \left\langle \begin{array}{c} 0.75 \\ 0.25 \end{array} \right\rangle \\ P(X_1) \qquad P(X_2) \qquad P(X_3) \qquad P(X_4) \qquad \qquad \qquad P(X_\infty) \end{array}$$

- From initial observation of rain

$$\begin{array}{c} \left\langle \begin{array}{c} 0.0 \\ 1.0 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.3 \\ 0.7 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.48 \\ 0.52 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.588 \\ 0.412 \end{array} \right\rangle \longrightarrow \left\langle \begin{array}{c} 0.75 \\ 0.25 \end{array} \right\rangle \\ P(X_1) \qquad P(X_2) \qquad P(X_3) \qquad P(X_4) \qquad \qquad \qquad P(X_\infty) \end{array}$$

- From initial observation of rain

$$\begin{array}{c} \left\langle \begin{array}{c} p \\ 1-p \end{array} \right\rangle \qquad \dots \qquad \longrightarrow \left\langle \begin{array}{c} 0.75 \\ 0.25 \end{array} \right\rangle \\ P(X_1) \qquad \qquad \qquad \qquad \qquad P(X_\infty) \end{array}$$

Stationary Distributions

- For most chains:
 - Influence of the initial distribution gets less and less over time
 - The distribution we end up in is independent of the initial distribution
- Stationary distribution:
 - The distribution we end up with is called the stationary distribution of the chain
 - It satisfies

$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$



Example: Stationary Distributions

- Question: What's $P(X)$ at time $t = \text{infinity}$?

$$P_\infty(\text{sun}) = P(\text{sun}|\text{sun})P_\infty(\text{sun}) + P(\text{sun}|\text{rain})P_\infty(\text{rain})$$

$$P_\infty(\text{rain}) = P(\text{rain}|\text{sun})P_\infty(\text{sun}) + P(\text{rain}|\text{rain})P_\infty(\text{rain})$$

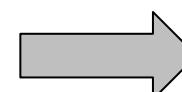
$$P_\infty(\text{sun}) = 0.9P_\infty(\text{sun}) + 0.3P_\infty(\text{rain})$$

$$P_\infty(\text{rain}) = 0.1P_\infty(\text{sun}) + 0.7P_\infty(\text{rain})$$

$$P_\infty(\text{sun}) = 3P_\infty(\text{rain})$$

$$P_\infty(\text{rain}) = 1/3P_\infty(\text{sun})$$

$$P_\infty(\text{sun}) + P_\infty(\text{rain}) = 1$$

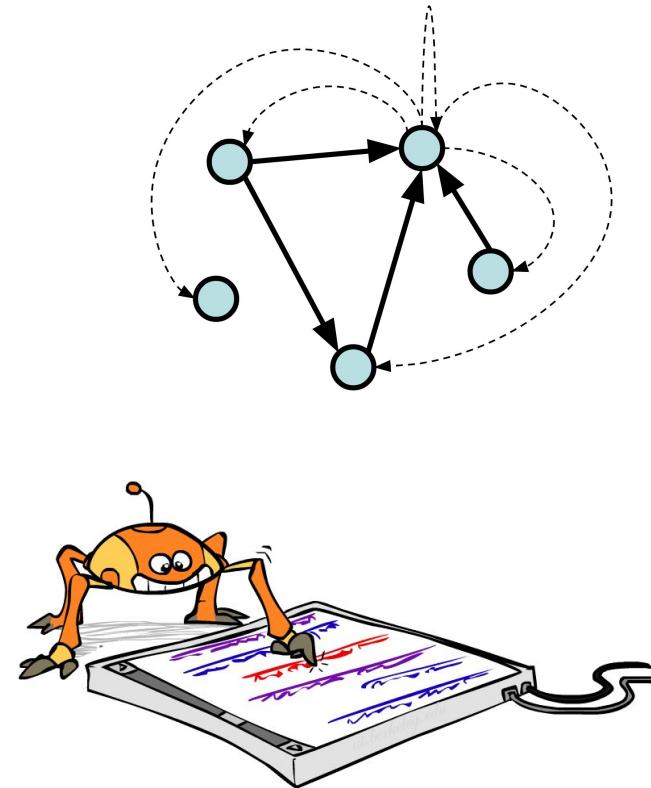


$$\begin{aligned} P_\infty(\text{sun}) &= 3/4 \\ P_\infty(\text{rain}) &= 1/4 \end{aligned}$$

X_{t-1}	X_t	$P(X_t X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

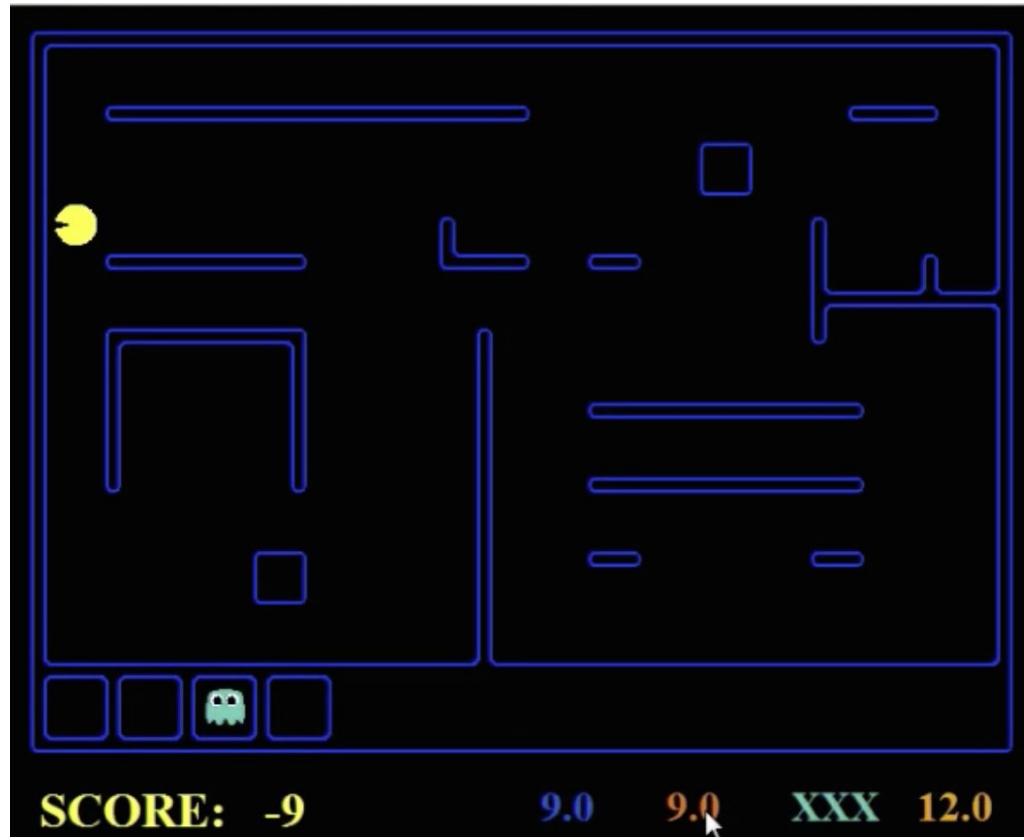
Application of Stationary Distribution: Web Link Analysis

- PageRank over a web graph
 - Each web page is a state
 - Initial distribution: uniform over pages
- Transitions:
 - With prob. c , uniform jump to a random page (dotted lines, not all shown)
 - With prob. $1-c$, follow a random outlink (solid lines)
- Stationary distribution
 - Will spend more time on highly reachable pages
 - E.g. many ways to get to the hku.hk page
 - Somewhat robust to link spam
 - Google 1.0 returned the set of pages containing all your keywords in decreasing rank, now all search engines use link analysis along with many other factors (rank actually getting less important over time)



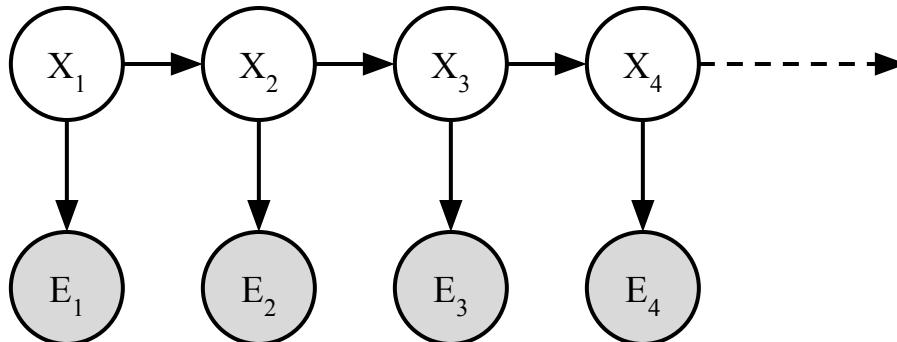
Next time: Hidden Markov Models



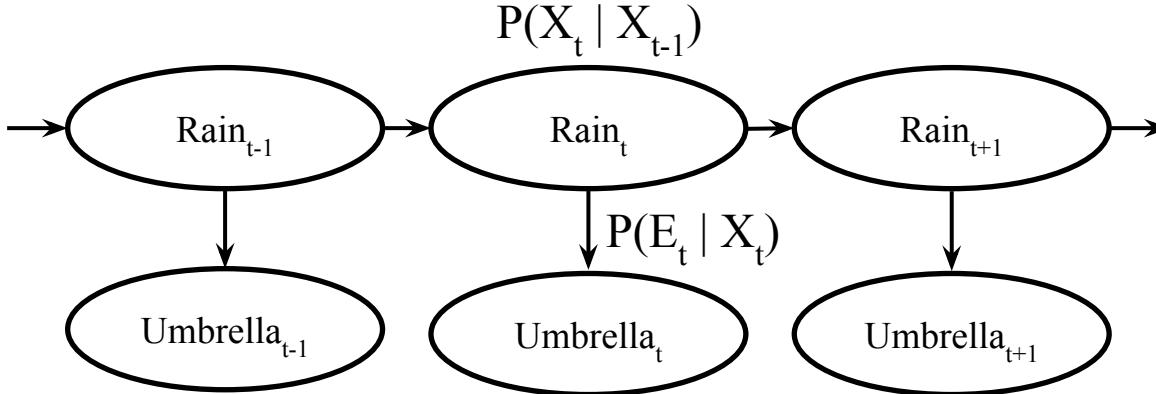


Hidden Markov Models

- Markov chains not so useful for most agents
 - Need observations to update your beliefs
- Hidden Markov models (HMMs)
 - Underlying Markov chain over states X
 - You observe outputs (effects) at each time step



Example: Weather HMM



R _t	R _{t+1}	P(R _{t+1} R _t)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

- An HMM is defined by
 - Initial distribution P(X₁)
 - Transitions P(X_t | X_{t-1})
 - Emissions P(E_t | X_t)

R _t	U _t	P(U _t R _t)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8



Chapter 5

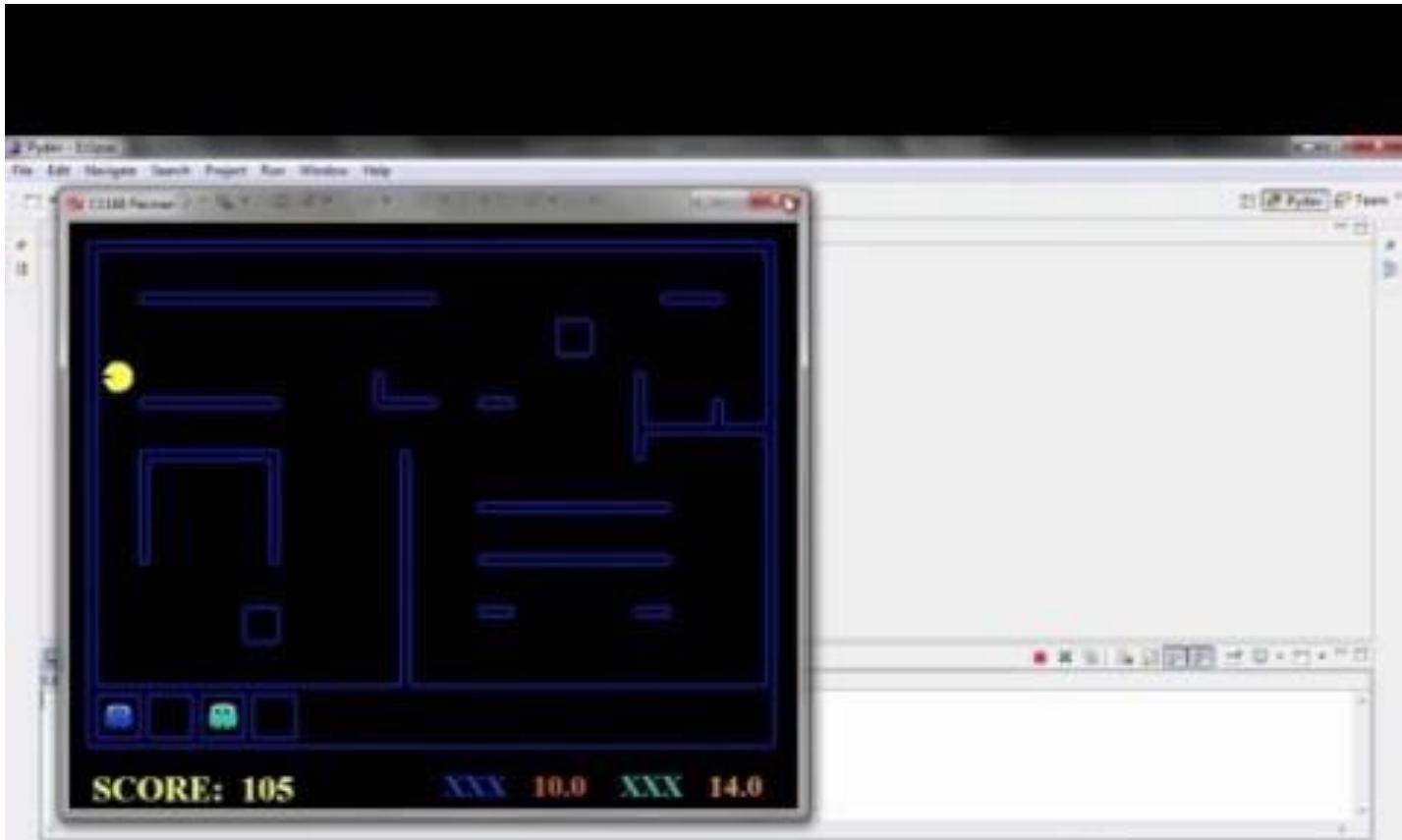
Hidden Markov Models

COMP 3270
Artificial Intelligence

Dirk Schnieders

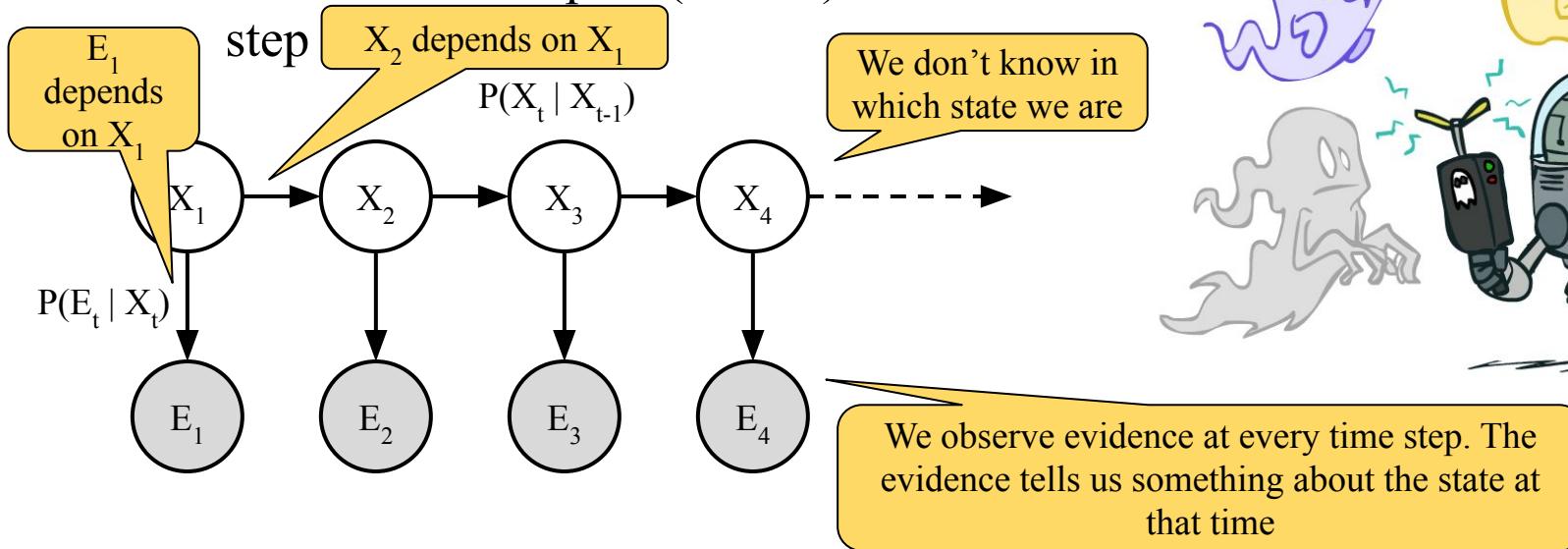
Hidden Markov Models



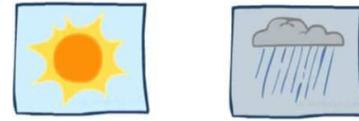
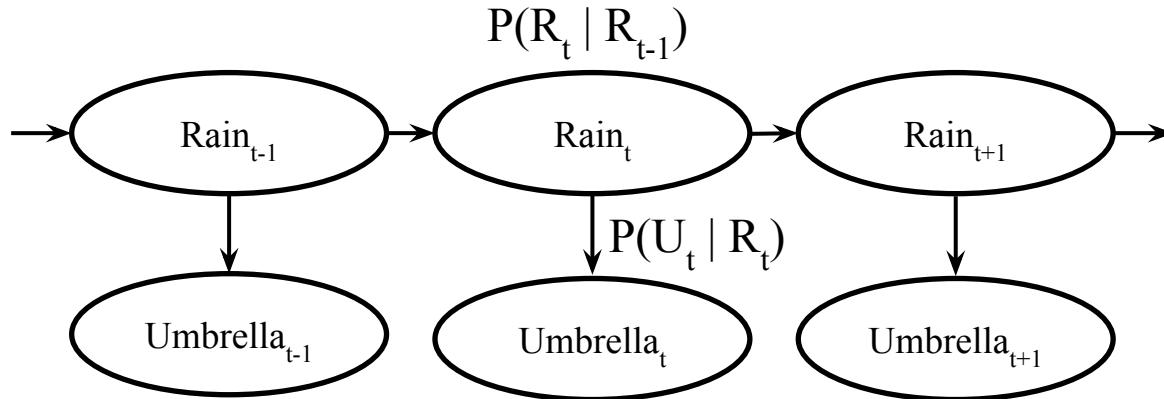


Hidden Markov Models

- Markov chains not so useful for most agents
 - Need observations to update your beliefs
- Hidden Markov models (HMMs)
 - Underlying Markov chain over states X
 - You observe outputs (effects) at each time step



Example: Weather HMM



R_t	R_{t+1}	$P(R_{t+1} R_t)$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

R_t	U_t	$P(U_t R_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

- An HMM is defined by
 - Initial distribution $P(R_1)$
 - Transitions $P(R_t | R_{t-1})$
 - Emissions $P(U_t | R_t)$

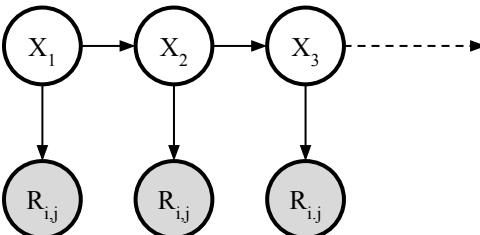
Example: Ghostbusters HMM

- $P(X_1)$ = uniform

Where is the ghost?

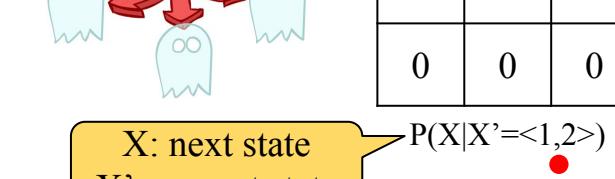
- $P(X|X')$ = usually move clockwise, but sometimes move in a random direction or stay in place

- $P(R_{ij}|X)$ = same sensor model as before
 - red means close, green means far away

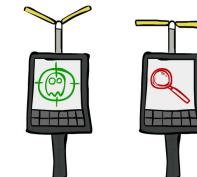


$P(X_1)$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

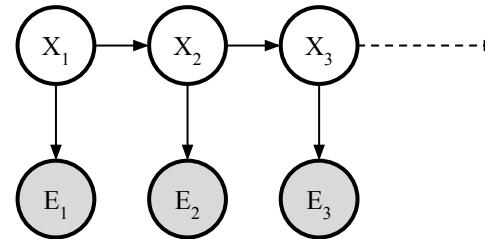


$P(X|X'=<1,2>)$



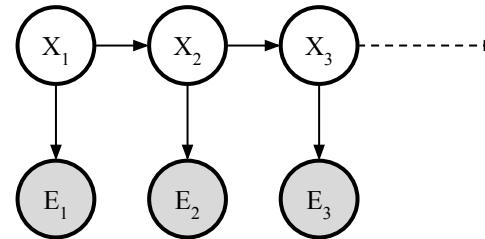


Joint Distribution of an HMM



- Joint distribution:
 - $P(X_1, E_1, X_2, E_2, X_3, E_3) = ?$

Joint Distribution of an HMM



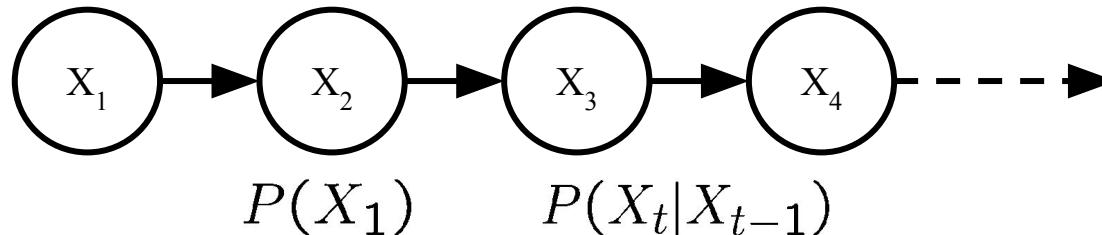
- Joint distribution:

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)$$

- More generally:

$$P(X_1, E_1, \dots, X_T, E_T) = P(X_1)P(E_1|X_1) \prod_{t=2}^T P(X_t|X_{t-1})P(E_t|X_t)$$

Joint Distribution of a Markov Model



- Joint distribution:

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

- More generally:

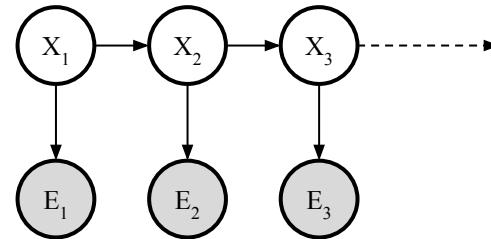
$$P(X_1, X_2, \dots, X_T) = P(X_1)P(X_2|X_1)P(X_3|X_2)\dots P(X_T|X_{T-1})$$

$$= P(X_1) \prod_{t=2}^T P(X_t|X_{t-1})$$

- Questions to be resolved:

- Does this indeed define a joint distribution?
- Can every joint distribution be factored this way, or are we making some assumptions about the joint distribution by using this factorization?

Quiz 1: Joint Distribution of an HMM



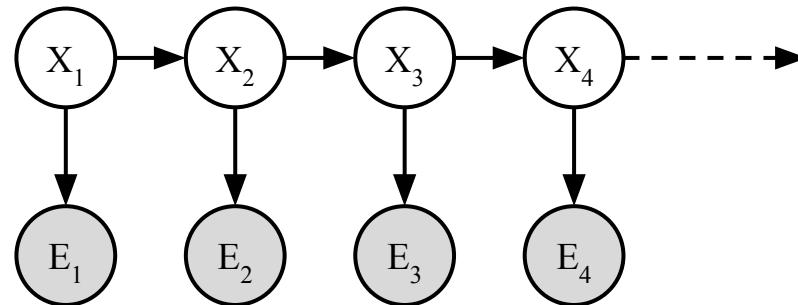
- Joint distribution:

Why?

$$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1)P(E_1|X_1)P(X_2|X_1)P(E_2|X_2)P(X_3|X_2)P(E_3|X_3)$$

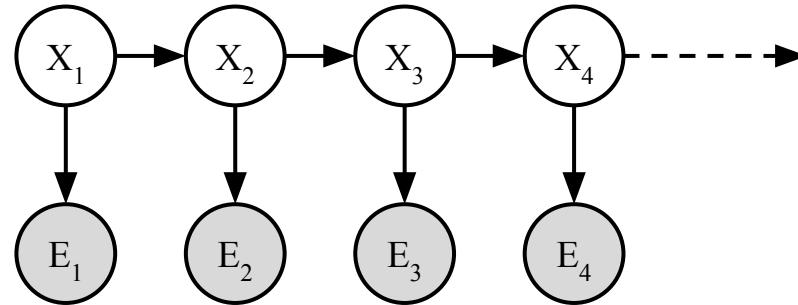
Conditional Independence

- HMMs have two important independence properties
 - Hidden Markov process: future depends on past via present
 - Current observation independent of all else given current state



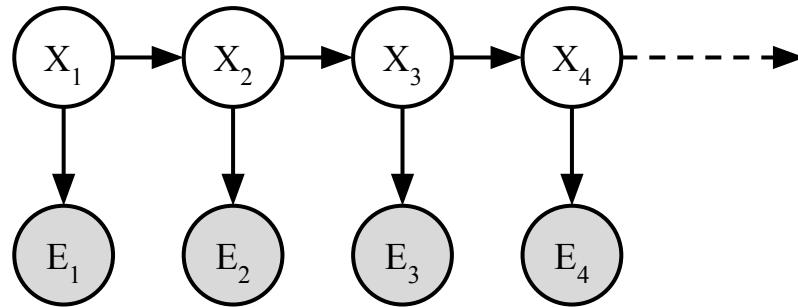
Quiz 2: Conditional Independence

- HMMs have two important independence properties
 - Markov hidden process: future depends on past via present
 - Current observation independent of all else given current state



- Quiz
 - Does this mean that evidence variables are guaranteed to be independent?

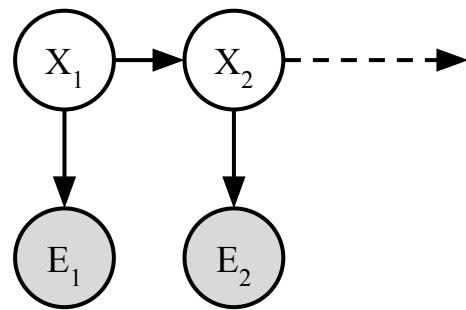
Implied Conditional Independencies



- Many implied conditional independencies, e.g.,

$$E_1 \perp\!\!\!\perp X_2, E_2, X_3, E_3 \mid X_1$$

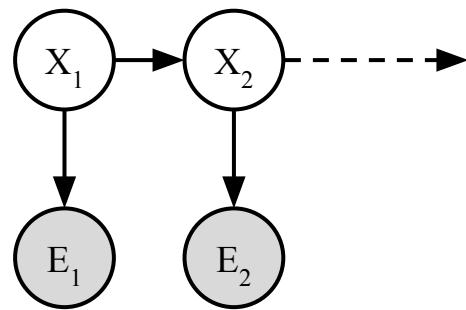
Quiz 3: Joint Distribution of an HMM



		X_t	X_{t+1}	$P(X_{t+1} X_t)$	X_t	E_t	$P(E_t X_t)$
		$+x_t$	$+x_{t+1}$	0.2	$+x_t$	$+e_t$	0.5
$P(X_1)$	$+x_1$	0.7	$+x_t$	0.8	$+x_t$	$-e_t$	0.5
	$-x_1$	0.3	$-x_t$	0.6	$-x_t$	$+e_t$	0.9
	$-x_t$		$-x_{t+1}$	0.4	$-x_t$	$-e_t$	0.1

- $P(X_1, X_2, E_1=+e_1, E_2=-e_2) = ?$

Quiz 4: Joint Distribution of an HMM



		X_t	X_{t+1}	$P(X_{t+1} X_t)$	X_t	E_t	$P(E_t X_t)$
		$+x_t$	$+x_{t+1}$	0.2	$+x_t$	$+e_t$	0.5
X_1	$P(X_1)$	$+x_t$	$-x_{t+1}$	0.8	$+x_t$	$-e_t$	0.5
$+x_1$	0.7	$-x_t$	$+x_{t+1}$	0.6	$-x_t$	$+e_t$	0.9
$-x_1$	0.3	$-x_t$	$-x_{t+1}$	0.4	$-x_t$	$-e_t$	0.1

- $P(X_2, E_1=+e_1, E_2=-e_2) = ?$

HMM Examples

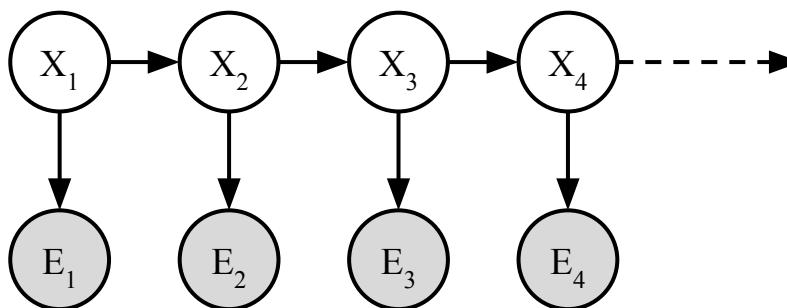
- Speech recognition
 - Observations are acoustic signals (continuous valued)
 - States are specific positions in specific words (so, tens of thousands)
- Machine translation
 - Observations are words (tens of thousands)
 - States are translation options
- Robot tracking
 - Observations are range readings (continuous)
 - States are positions on a map (continuous)

Filtering / Monitoring

- Filtering, or monitoring, is the task of tracking the distribution $B_t(X) = P(X_t | e_1, \dots, e_t)$ (the belief state) over time

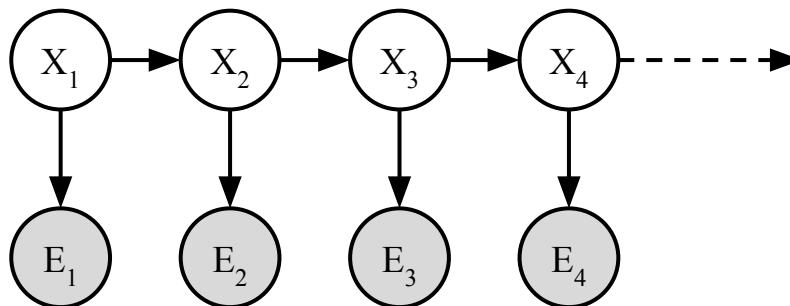
New notation!

Keep track of what you believe about a variable X as evidence comes in and time passes

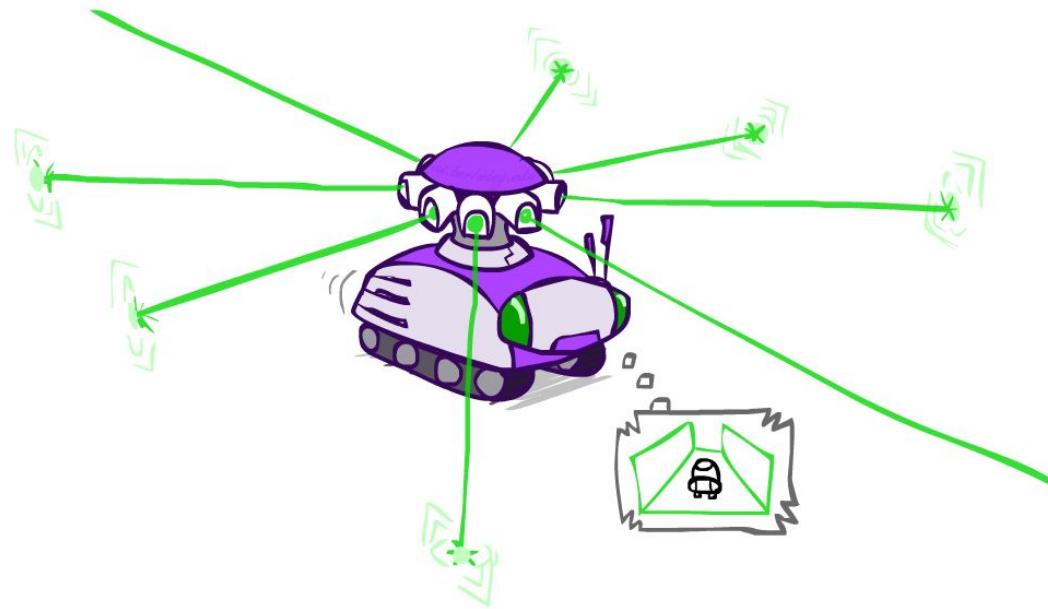


Filtering / Monitoring

- Filtering, or monitoring, is the task of tracking the distribution $B_t(X) = P(X_t | e_1, \dots, e_t)$ (the belief state) over time
- We start with $B_1(X)$ in an initial setting, usually uniform
- As time passes, or we get observations, we update $B(X)$
- The Kalman filter was invented in the 60's and first implemented as a method of trajectory estimation for the Apollo program

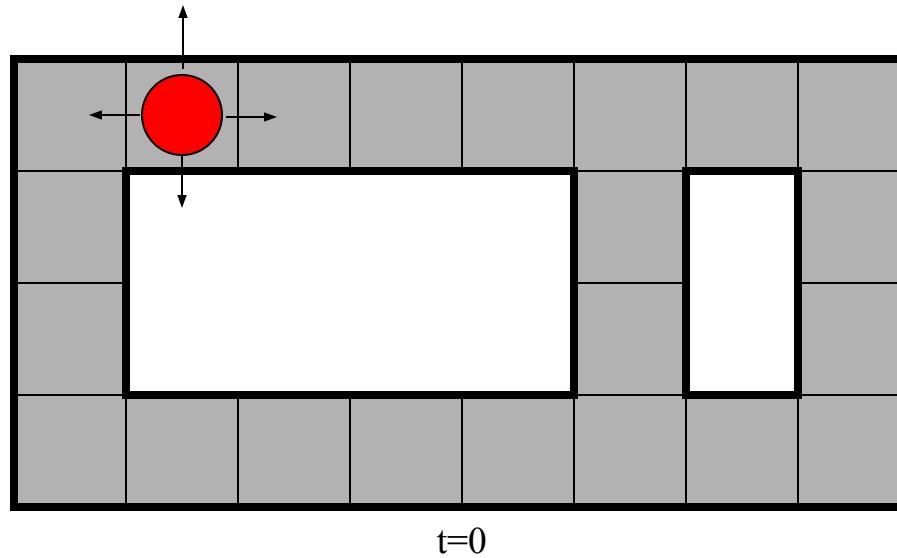


Example: Robot Localization



Sensor model: can read in which directions there is a wall, never more than 1 mistake
Motion model: may not execute action with small prob.

Example: Robot Localization



$t=0$

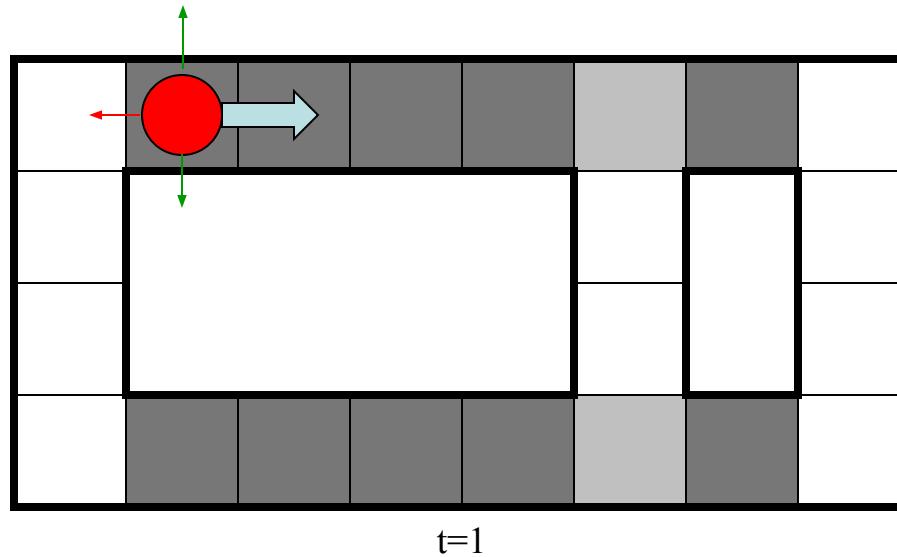
Prob

0

1

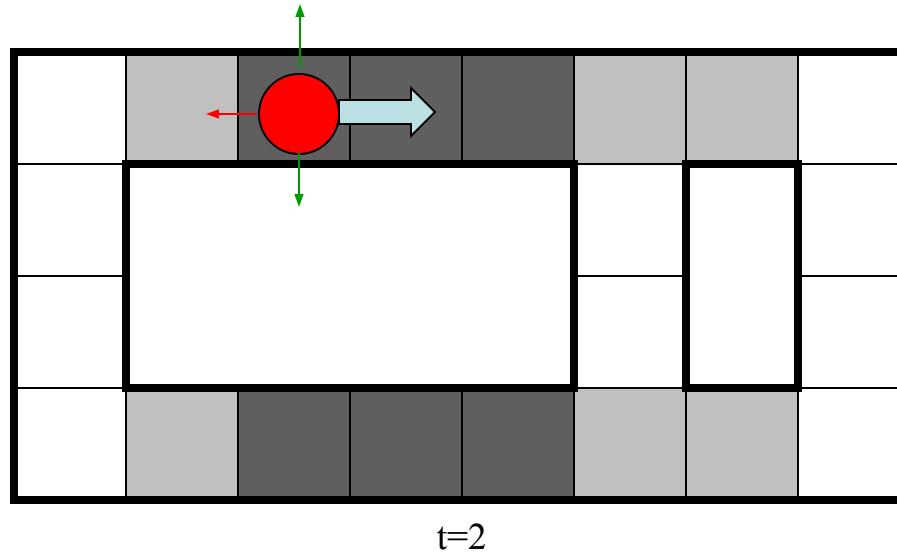


Example: Robot Localization



Prob 0 1

Example: Robot Localization

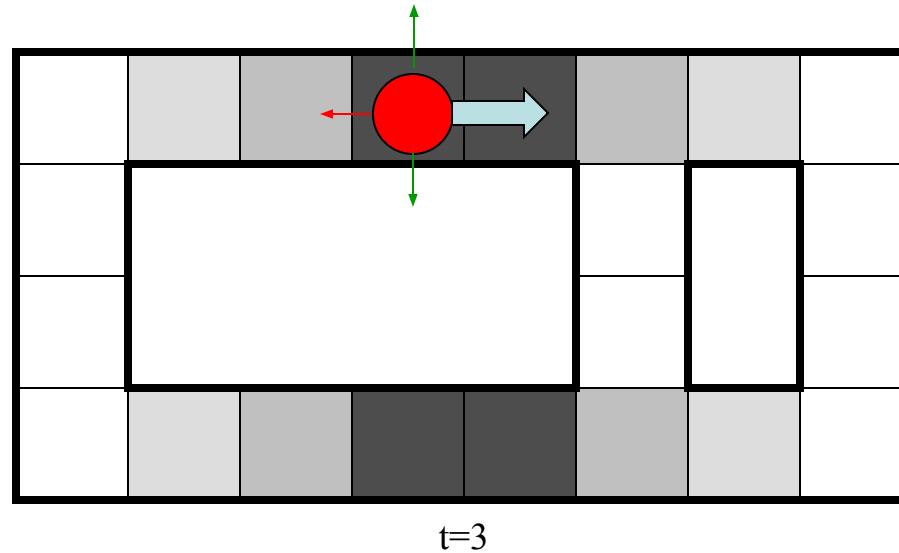


Prob

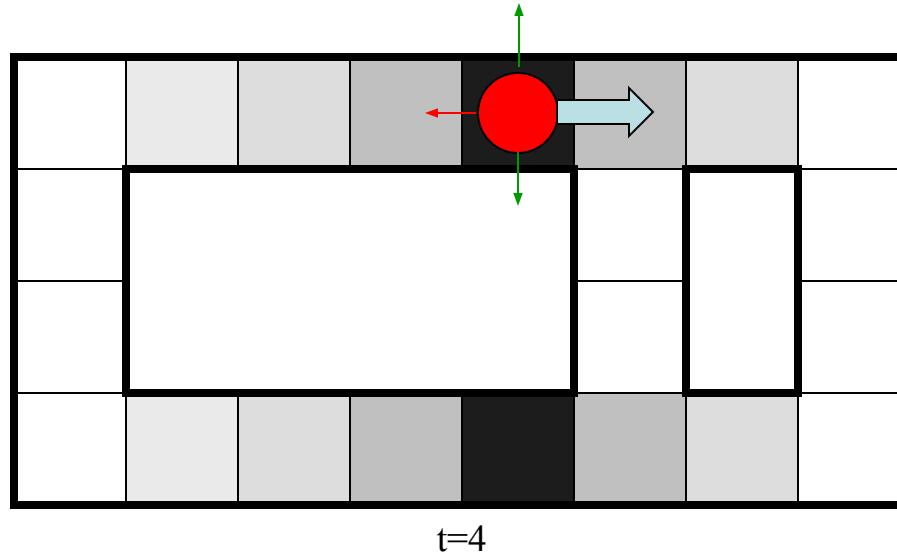
0

1

Example: Robot Localization



Example: Robot Localization

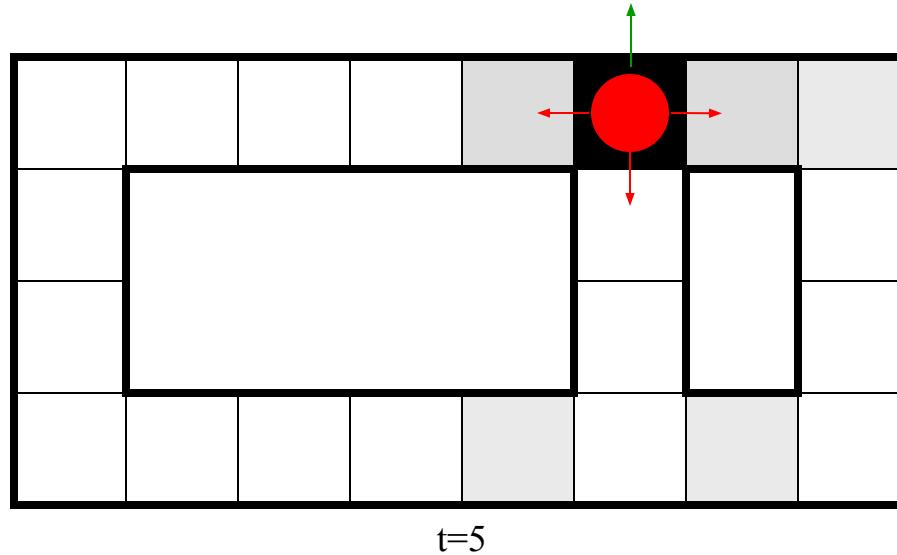


Prob

0

1

Example: Robot Localization



Prob

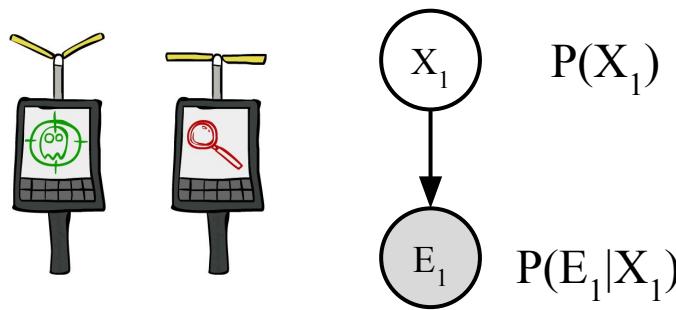
0

1



Inference: Base Cases

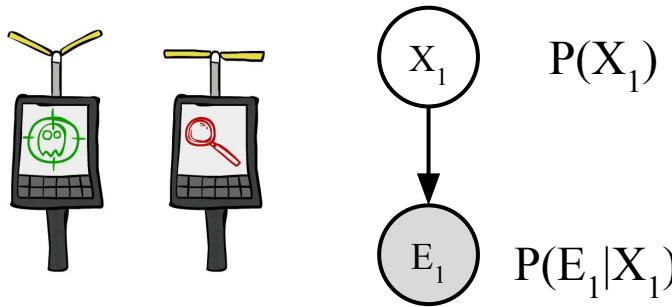
Seeing evidence:



$$P(X_1 | e_1) = ?$$

Inference: Base Cases

Seeing evidence:



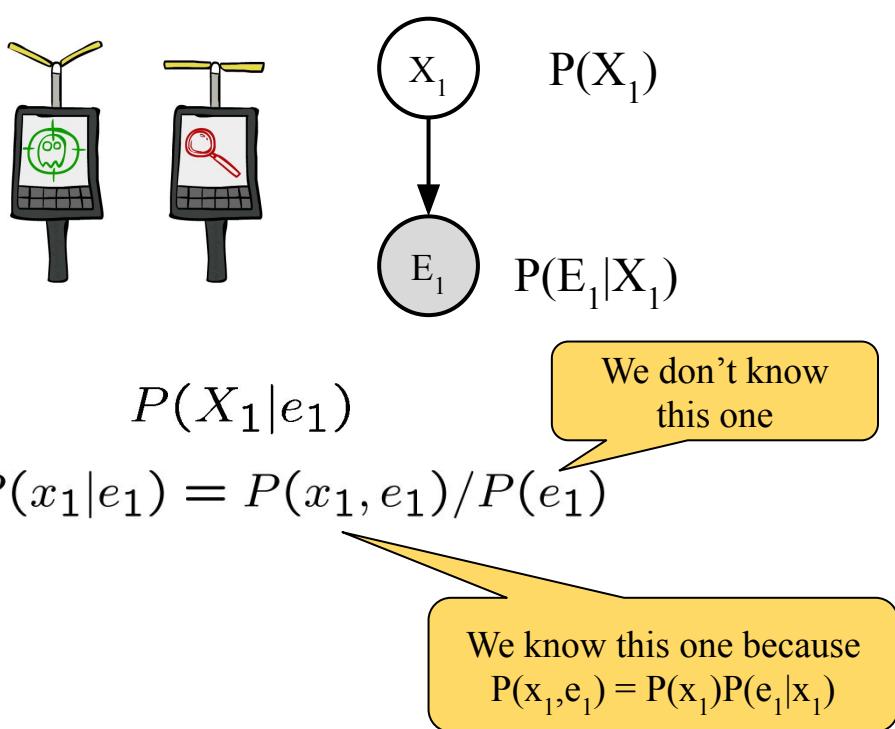
$$P(X_1|e_1)$$

$$P(x_1|e_1) = P(x_1, e_1)/P(e_1)$$

Definition of conditional probability

Inference: Base Cases

Seeing evidence:



Ghostbusters, Revisited

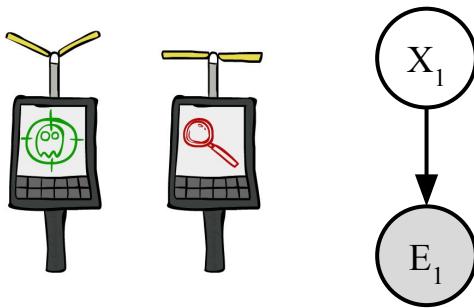
- Let's say we have two distributions:
 - Prior distribution over ghost location: $P(G)$
 - Let this be uniform
 - Sensor reading model: $P(R|G)$
 - Given: we know what our sensors do
 - R = reading color measured at $(1,1)$
 - E.g. $P(R = \text{yellow} | G=(1,1)) = 0.1$
- We can calculate the posterior distribution $P(G|r)$ over ghost locations given a reading using Bayes' rule:

$$P(g|r) \propto P(r|g)P(g)$$

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

Inference: Base Cases

Seeing evidence:



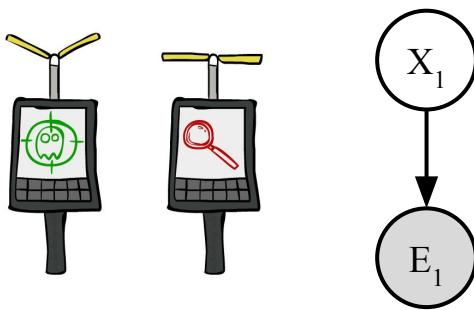
$$P(X_1|e_1)$$

$$\begin{aligned} P(x_1|e_1) &= P(x_1, e_1)/P(e_1) \\ &\propto_{X_1} P(x_1, e_1) \\ &= P(x_1)P(e_1|x_1) \end{aligned}$$

We take our current probabilities and multiply with the evidence probability. Then renormalize

Inference: Base Cases

Seeing evidence:



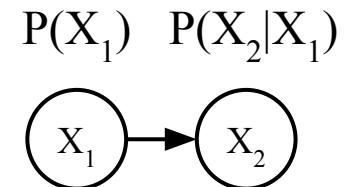
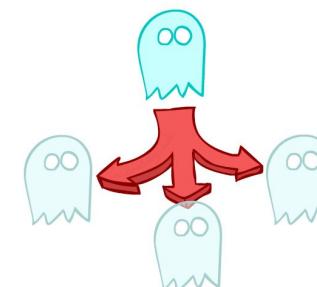
$$P(X_1|e_1)$$

$$P(x_1|e_1) = P(x_1, e_1)/P(e_1)$$

$$\propto_{X_1} P(x_1, e_1)$$

$$= P(x_1)P(e_1|x_1)$$

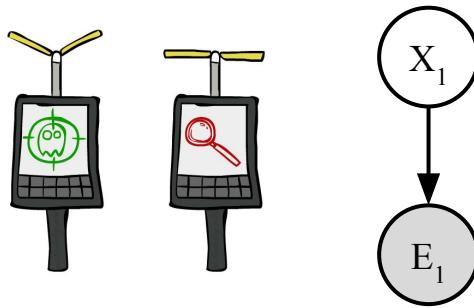
Time passes:



$$P(X_2)$$

Inference: Base Cases

Seeing evidence:



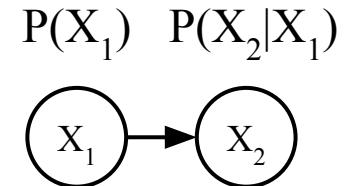
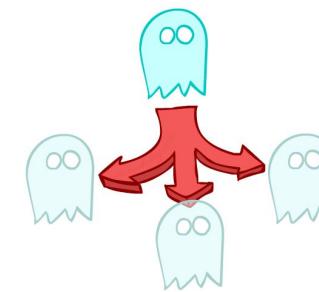
$$P(X_1|e_1)$$

$$P(x_1|e_1) = P(x_1, e_1)/P(e_1)$$

$$\propto_{X_1} P(x_1, e_1)$$

$$= P(x_1)P(e_1|x_1)$$

Time passes:



$$P(X_2)$$

$$P(x_2) = \sum_{x_1} P(x_1, x_2)$$

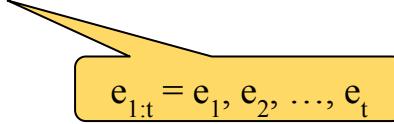
$$= \sum_{x_1} P(x_1)P(x_2|x_1)$$

Same as Markov Model

Passage of Time

- Assume we have current belief $P(X | \text{evidence to date})$

$$B(X_t) = P(X_t | e_{1:t})$$


$$e_{1:t} = e_1, e_2, \dots, e_t$$

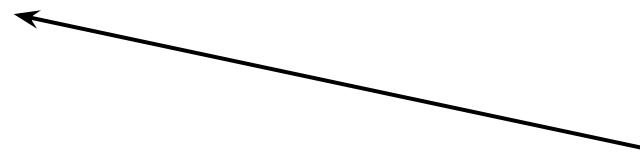
Passage of Time

- Assume we have current belief $P(X \mid \text{evidence to date})$

$$B(X_t) = P(X_t | e_{1:t})$$

- Then, after one time step passes:

$$P(X_{t+1} | e_{1:t})$$



$$\begin{aligned} P(x_2) &= \sum_{x_1} P(x_1, x_2) \\ &= \sum_{x_1} P(x_1)P(x_2|x_1) \end{aligned}$$

We can still apply the formula, even with all the $| e_{1:t}$ in the back

We are going to apply this formula

Passage of Time

- Assume we have current belief $P(X \mid \text{evidence to date})$

$$B(X_t) = P(X_t | e_{1:t})$$

- Then, after one time step passes:

$$\begin{aligned} P(X_{t+1} | e_{1:t}) &= \sum_{x_t} P(X_{t+1}, x_t | e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t}) \end{aligned}$$

$$\begin{aligned} P(x_2) &= \sum_{x_1} P(x_1, x_2) \\ &= \sum_{x_1} P(x_1) P(x_2 | x_1) \end{aligned}$$

Use conditional
independence assumption
to get rid of this

Passage of Time

- Assume we have current belief $P(X \mid \text{evidence to date})$

$$B(X_t) = P(X_t | e_{1:t})$$

- Then, after one time step passes:

$$\begin{aligned} P(X_{t+1} | e_{1:t}) &= \sum_{x_t} P(X_{t+1}, x_t | e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1} | x_t) P(x_t | e_{1:t}) \end{aligned}$$

- Or, compactly

$$B'(X_{t+1}) = \sum_{x_t} P(X' | x_t) B(x_t)$$

We have not seen e_{t+1} yet

New notation! Note that
 $B'(X_{t+1}) = P(X_{t+1} | e_{1:t})$
is different from
 $B(X_{t+1}) = P(X_{t+1} | e_{1:t+1})$

Passage of Time

- Assume we have current belief $P(X \mid \text{evidence to date})$

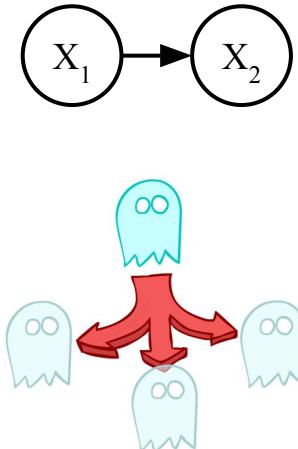
$$B(X_t) = P(X_t | e_{1:t})$$

- Then, after one time step passes:

$$\begin{aligned} P(X_{t+1} | e_{1:t}) &= \sum_{x_t} P(X_{t+1}, x_t | e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t}) \\ &= \sum_{x_t} P(X_{t+1} | x_t) P(x_t | e_{1:t}) \end{aligned}$$

- Or, compactly

$$B'(X_{t+1}) = \sum_{x_t} P(X' | x_t) B(x_t)$$



Basic idea

Beliefs get “pushed” through the transitions

Example: Passage of Time

- As time passes, uncertainty “accumulates”

(Transition model: ghosts usually go clockwise)

<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	1.00	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01

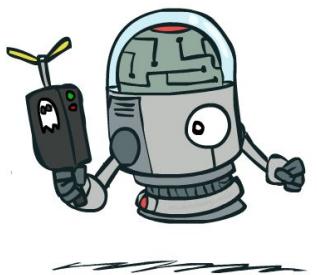
T = 1

<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01
<0.01	0.76	0.06	0.06	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01

T = 2

0.05	0.01	0.05	<0.01	<0.01	<0.01
0.02	0.14	0.11	0.35	<0.01	<0.01
0.07	0.03	0.05	<0.01	0.03	<0.01
0.03	0.03	<0.01	<0.01	<0.01	<0.01

T = 5



Observation

- Assume we have current belief $P(X \mid \text{previous evidence})$:

$$B'(X_{t+1}) = P(X_{t+1} | e_{1:t})$$

- Then, after evidence comes in

$$P(X_{t+1} | e_{1:t+1}) = P(X_{t+1}, e_{t+1} | e_{1:t}) / P(e_{t+1} | e_{1:t})$$

We are going to apply this formula

$$P(x_1 | e_1) = P(x_1, e_1) / P(e_1)$$

$$\propto_{X_1} P(x_1, e_1)$$

$$= P(x_1)P(e_1 | x_1)$$

Observation

- Assume we have current belief $P(X \mid \text{previous evidence})$:

$$B'(X_{t+1}) = P(X_{t+1} | e_{1:t})$$

- Then, after evidence comes in

$$P(X_{t+1} | e_{1:t+1}) = P(X_{t+1}, e_{t+1} | e_{1:t}) / P(e_{t+1} | e_{1:t})$$

$$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1} | e_{1:t})$$

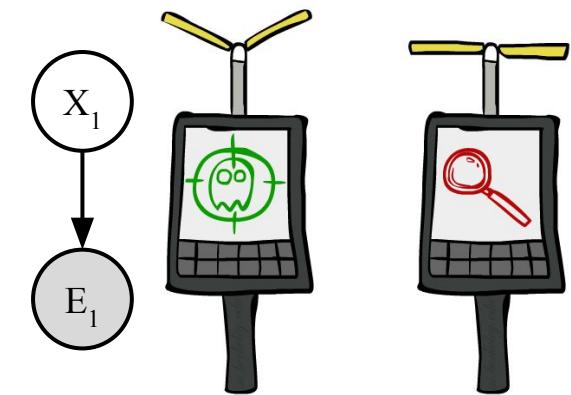
$$= P(e_{t+1} | e_{1:t}, X_{t+1}) P(X_{t+1} | e_{1:t})$$

$$= P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t})$$

- Or, compactly

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1} | X_{t+1}) B'(X_{t+1})$$

- Basic idea: beliefs “reweighted” by likelihood of evidence
 - Unlike passage of time, we have to renormalize



Example: Observation

- As we get observations, beliefs get reweighted, uncertainty “decreases”

0.05	0.01	0.05	<0.01	<0.01	<0.01
0.02	0.14	0.11	0.35	<0.01	<0.01
0.07	0.03	0.05	<0.01	0.03	<0.01
0.03	0.03	<0.01	<0.01	<0.01	<0.01

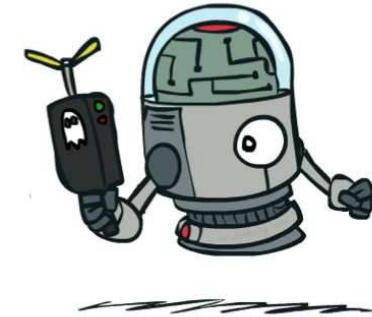
Before observation

<0.01	<0.01	<0.01	<0.01	0.02	<0.01
<0.01	<0.01	<0.01	0.83	0.02	<0.01
<0.01	<0.01	0.11	<0.01	<0.01	<0.01
<0.01	<0.01	<0.01	<0.01	<0.01	<0.01

After observation



$$B(X) \propto P(e|X)B'(X)$$



Example: Weather

$$\begin{array}{l} B'(+r) = ? \\ B'(-r) = ? \end{array}$$

\downarrow

$$\begin{array}{l} B(+r) = ? \\ B(-r) = ? \end{array}$$

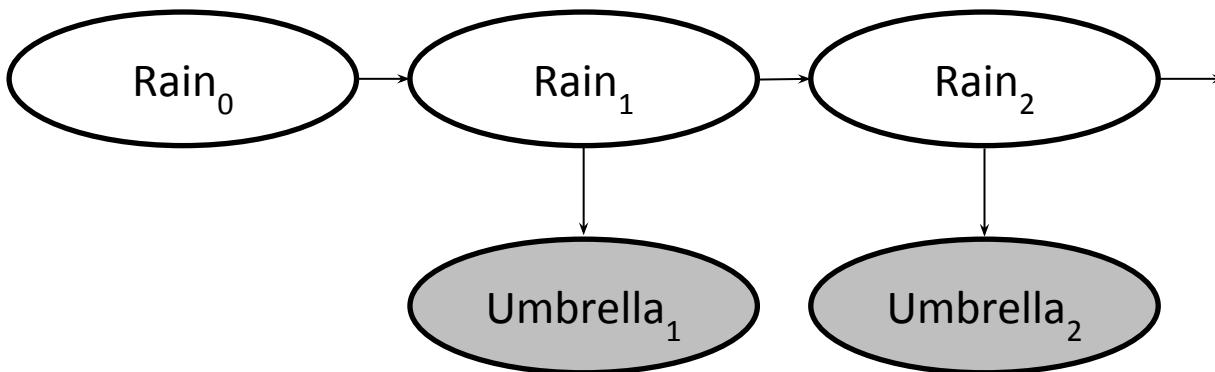
$$\begin{array}{l} B'(+r) = ? \\ B'(-r) = ? \end{array}$$

\downarrow

$$\begin{array}{l} B(+r) = ? \\ B(-r) = ? \end{array}$$

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

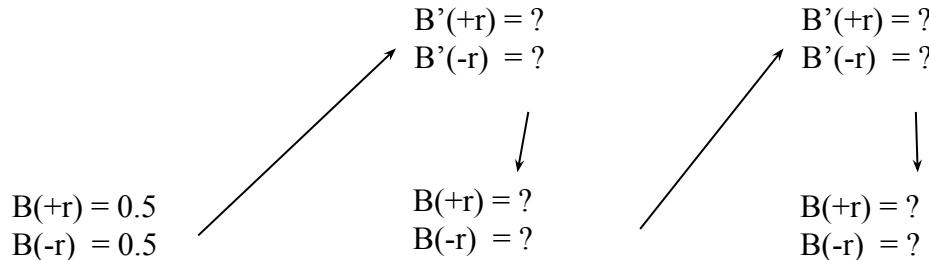
$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$



X_t	X_{t+1}	$P(X_{t+1} X_t)$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X_t	E_t	$P(E_t X_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Example: Weather



$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$

X_t	X_{t+1}	$P(X_{t+1} X_t)$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X_t	E_t	$P(E_t X_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Example: Weather

$$\begin{array}{c}
 \text{B}'(+r) = 0.5 \\
 \text{B}'(-r) = 0.5
 \end{array}
 \quad \downarrow \quad
 \begin{array}{c}
 \text{B}'(+r) = ? \\
 \text{B}'(-r) = ?
 \end{array}$$

$$\begin{array}{l}
 \text{B}(+r) = 0.5 \\
 \text{B}(-r) = 0.5
 \end{array}$$

$$\begin{array}{c}
 \text{B}'(+r) = ? \\
 \text{B}'(-r) = ?
 \end{array}
 \quad \downarrow \quad
 \begin{array}{c}
 \text{B}(+r) = ? \\
 \text{B}(-r) = ?
 \end{array}$$

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$

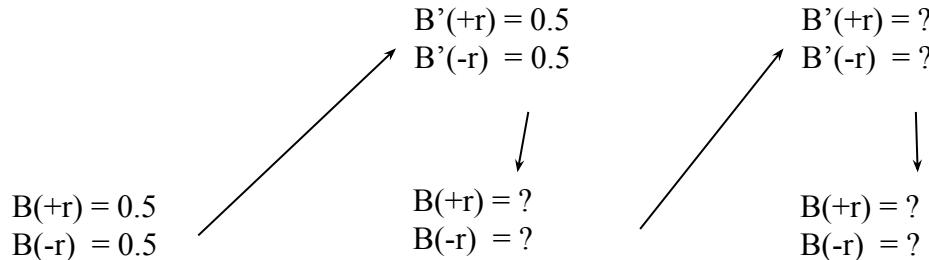
$$\begin{aligned}
 B'(X_{t+1} = +r) &= P(X_{t+1} = +r | x_t = +r) B(x_t = +r) + P(X_{t+1} = +r | x_t = -r) B(x_t = -r) \\
 &= 0.7 * 0.5 + 0.3
 \end{aligned}$$

$$\begin{aligned}
 B'(X_{t+1} = -r) &= P(X_{t+1} = -r | x_t = +r) B(x_t = +r) + P(X_{t+1} = -r | x_t = -r) B(x_t = -r) \\
 &= 0.3 * 0.5 + 0.7
 \end{aligned}$$

X _t	X _{t+1}	P(X _{t+1} X _t)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X _t	E _t	P(E _t X _t)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Example: Weather



$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$

X_t	X_{t+1}	$P(X_{t+1} X_t)$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X_t	E_t	$P(E_t X_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Example: Weather

$$\begin{array}{c}
 \text{B}(+r) = 0.5 \\
 \text{B}(-r) = 0.5
 \end{array}
 \xrightarrow{\quad}
 \begin{array}{c}
 \text{B}'(+r) = 0.5 \\
 \text{B}'(-r) = 0.5
 \end{array}
 \xrightarrow{\quad}
 \begin{array}{c}
 \text{B}(+r) = 0.8182 \\
 \text{B}(-r) = 0.1818
 \end{array}$$

$$\begin{array}{c}
 \text{B}(+r) = ? \\
 \text{B}(-r) = ?
 \end{array}
 \xrightarrow{\quad}
 \begin{array}{c}
 \text{B}'(+r) = ? \\
 \text{B}'(-r) = ?
 \end{array}
 \xrightarrow{\quad}
 \begin{array}{c}
 \text{B}(+r) = ? \\
 \text{B}(-r) = ?
 \end{array}$$

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$

X _t	X _{t+1}	P(X _{t+1} X _t)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

$$B(X_{t+1} = +r) \propto P(E_{t+1} = +u | X_{t+1} = +r) B'(X_{t+1} = +r) = 0.9 * 0.5 = 0.45$$

$$B(X_{t+1} = -r) \propto P(E_{t+1} = +u | X_{t+1} = -r) B'(X_{t+1} = -r) = 0.2 * 0.5 = 0.1$$

$$z = 0.55$$

$$B(X_{t+1} = +r) = 0.45 / z = 0.8182$$

$$B(X_{t+1} = -r) = 0.1 / z = 0.1818$$

X _t	E _t	P(E _t X _t)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Quiz 5: Weather

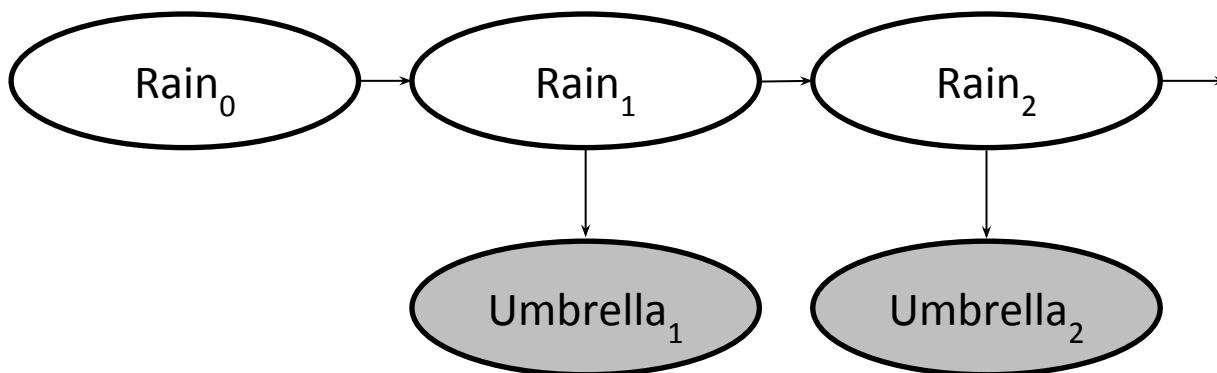
Calculate these values!

$$\begin{aligned} & \text{B}'(+r) = 0.5 \\ & \text{B}'(-r) = 0.5 \\ \downarrow & \\ & \text{B}(+r) = 0.8182 \\ & \text{B}(-r) = 0.1818 \end{aligned}$$

$$\begin{aligned} & \text{B}'(+r) = ? \\ & \text{B}'(-r) = ? \\ \downarrow & \\ & \text{B}(+r) = ? \\ & \text{B}(-r) = ? \end{aligned}$$

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$



X _t	X _{t+1}	P(X _{t+1} X _t)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X _t	E _t	P(E _t X _t)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

Quiz 6: Weather

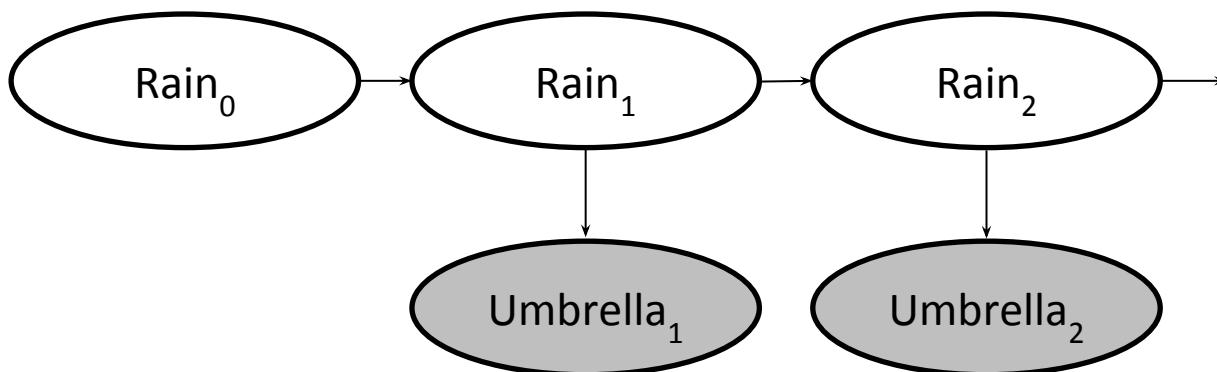
$$\begin{aligned}
 & B(+r) = 0.5 & B'(+r) = 0.5 \\
 & B(-r) = 0.5 & B'(-r) = 0.5 \\
 & & \downarrow \\
 & B(+r) = 0.8182 & \\
 & B(-r) = 0.1818 &
 \end{aligned}$$

$$\begin{aligned}
 & B'(+r) = ? & \\
 & B'(-r) = ? & \\
 & & \downarrow \\
 & B(+r) = ? & \\
 & B(-r) = ? &
 \end{aligned}$$

Calculate these values!

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$



X _t	X _{t+1}	P(X _{t+1} X _t)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

X _t	E _t	P(E _t X _t)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

The Forward Algorithm

- Alternatively, we can also just do a single update
- We are given evidence at each time and want to know

$$B_t(X) = P(X_t | e_{1:t})$$

- We can derive the following updates

$$P(x_t | e_{1:t}) \propto_X P(x_t, e_{1:t})$$

We can normalize as we go if we want to have $P(x|e)$ at each time step, or just once at the end...

$$= \sum_{x_{t-1}} P(x_{t-1}, x_t, e_{1:t})$$

$$= \sum_{x_{t-1}} P(x_{t-1}, e_{1:t-1}) P(x_t | x_{t-1}) P(e_t | x_t)$$

$$= P(e_t | x_t) \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1}, e_{1:t-1})$$

observation update

Time update

Online Belief Updates

- Every time step, we start with current $P(X \mid \text{evidence})$
- We update for time:

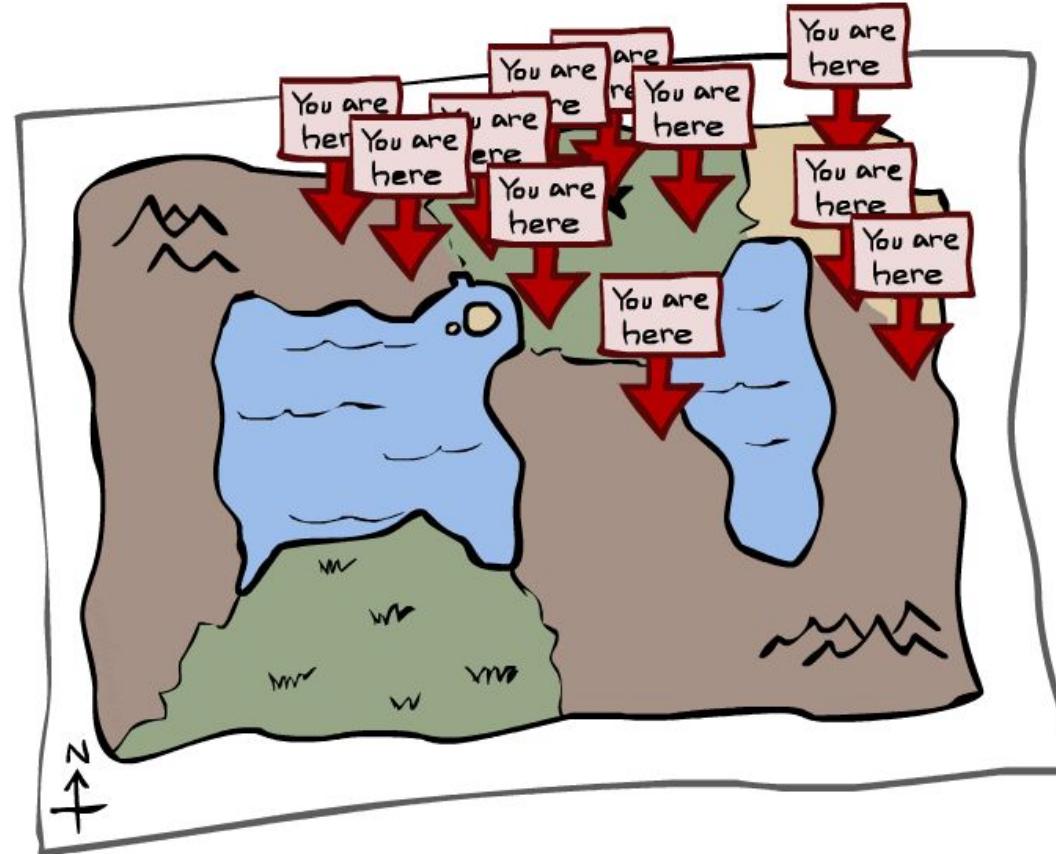
$$P(x_t | e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1} | e_{1:t-1}) \cdot P(x_t | x_{t-1})$$

- We update for evidence:

$$P(x_t | e_{1:t}) \propto_X P(x_t | e_{1:t-1}) \cdot P(e_t | x_t)$$

- The forward algorithm does both at once (and doesn't normalize)

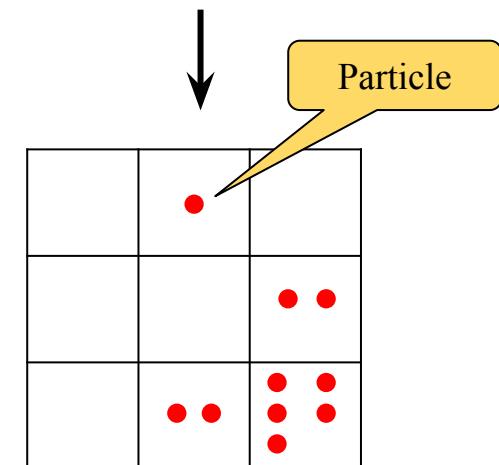
Particle Filtering



Particle Filtering

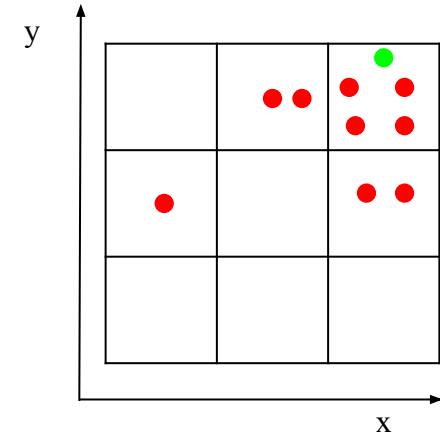
- Filtering: approximate solution
- Sometimes $|X|$ is too big to use exact inference
 - $|X|$ may be too big to even store $B(X)$
 - E.g. X is continuous
- Solution: approximate inference
 - Track samples of X , not all values
 - Samples are called particles
 - Time per step is linear in the number of samples
 - But: number needed may be large
 - In memory: list of particles, not states
- This is how robot localization works in practice
 - Particle is just new name for sample

0.0	0.1	0.0
0.0	0.0	0.2
0.0	0.2	0.5



Representation: Particles

- Our representation of $P(X)$ is now a list of N particles (samples)
 - Generally, $N \ll |X|$
 - Storing map from X to counts would defeat the point
- $P(x)$ approximated by number of particles with value x
 - So, many x may have $P(x) = 0$!
 - More particles, more accuracy
- For now, all particles have a weight of 1



Particles (x,y) :

(3,3)

(2,3)

(3,3)

(3,2)

(3,3)

(3,2)

(1,2)

(3,3)

(3,3)

(2,3)

Particle Filtering: Elapse of Time

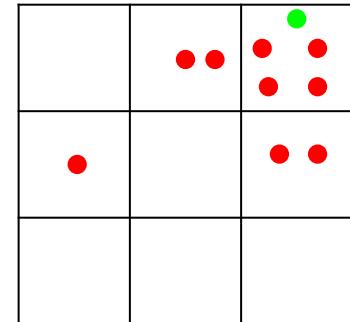
- Each particle is moved by sampling its next position from the transition model

$$x' = \text{sample}(P(X'|x))$$

- This is like prior sampling – samples' frequencies reflect the transition probabilities
- Here, most samples move clockwise, but some move in another direction or stay in place
- This captures the passage of time
 - If enough samples, close to exact values before and after (consistent)

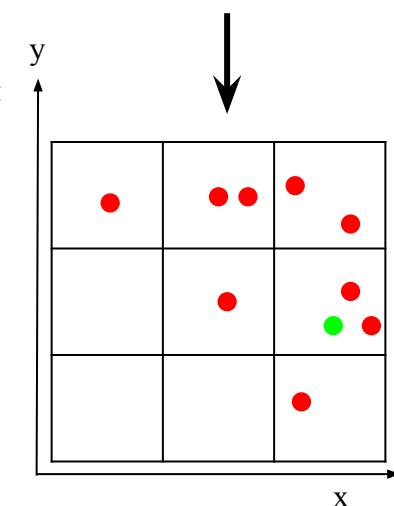
Particles:

(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)



Particles:

(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,3)
(1,3)
(2,3)
(3,2)
(2,2)



Particle Filtering: Observe

- Slightly trickier:
 - Don't sample observation, fix it
 - Similar to likelihood weighting, downweight samples based on the evidence

$$w(x) = P(e|x)$$

$$B(X) \propto P(e|X)B'(X)$$

- As before, the probabilities don't sum to one, since all have been downweighted (in fact they now sum to (N times) an approximation of $P(e)$)

Particles:

(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,3)
(1,2)
(3,3)
(3,3)
(2,3)

•	• •	• •
	•	• • •
		•



Particles:

(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,3) w=.4
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4

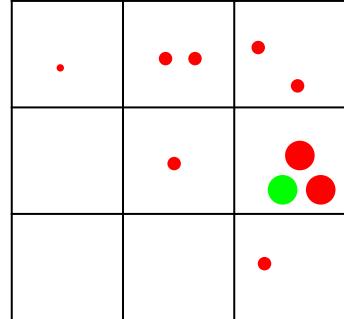
.	• •	• •
	•	• • •
		•

Particle Filtering: Resample

- Rather than tracking weighted samples, we resample
 - N times, we choose from our weighted sample distribution (i.e. draw with replacement)
 - This is equivalent to renormalizing the distribution
 - Now the update is complete for this time step, continue with the next one

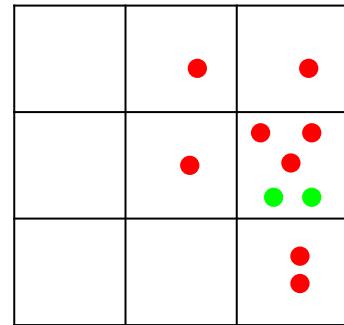
Particles:

(3,2) w=.9
 (2,3) w=.2
 (3,2) w=.9
 (3,1) w=.4
 (3,3) w=.4
 (3,3) w=.4
 (1,3) w=.1
 (2,3) w=.2
 (3,2) w=.9
 (2,2) w=.4



(New) Particles:

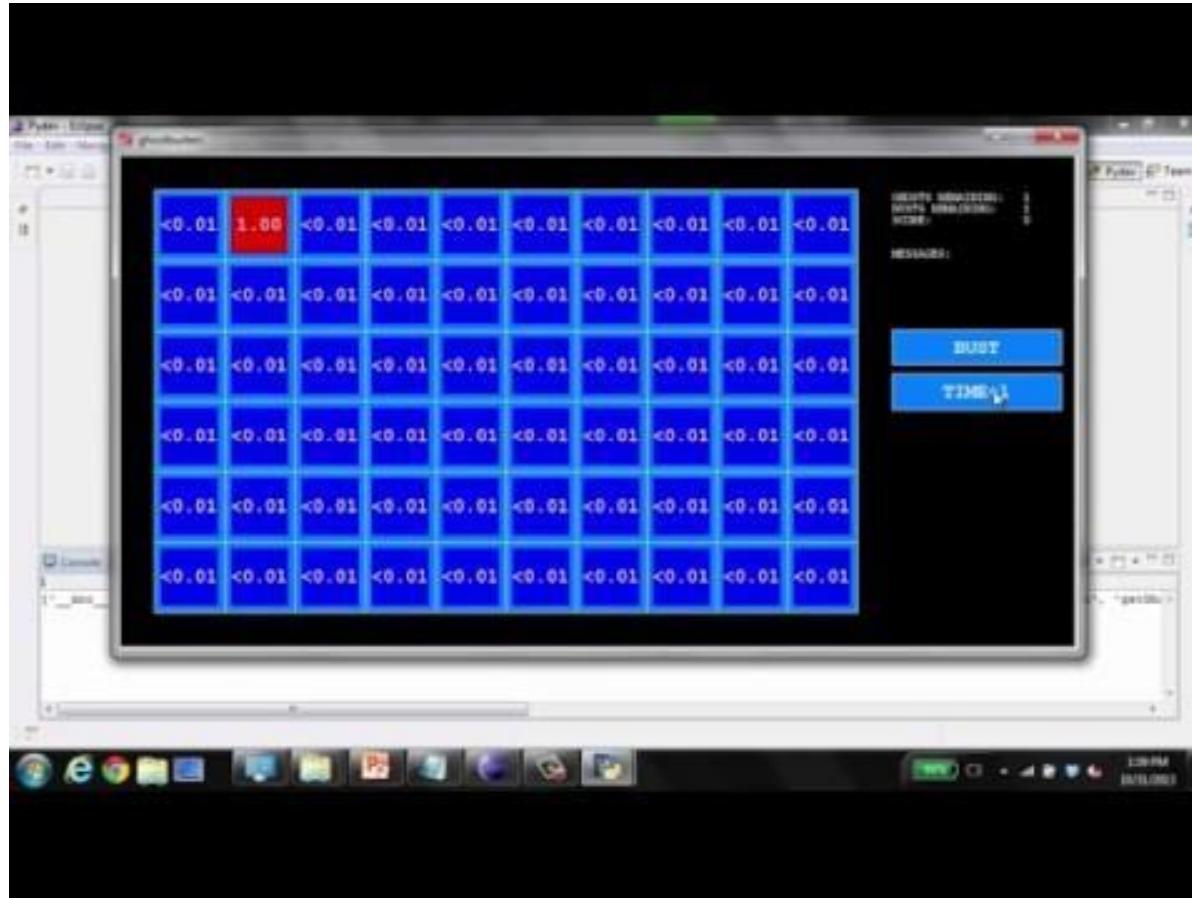
(3,2)
 (2,2)
 (3,2)
 (3,1)
 (3,3)
 (3,2)
 (3,1)
 (2,3)
 (3,2)
 (3,2)



Particle Filtering in Ghostbusters - Few Particles



Particle Filtering in Ghostbusters - One Particle

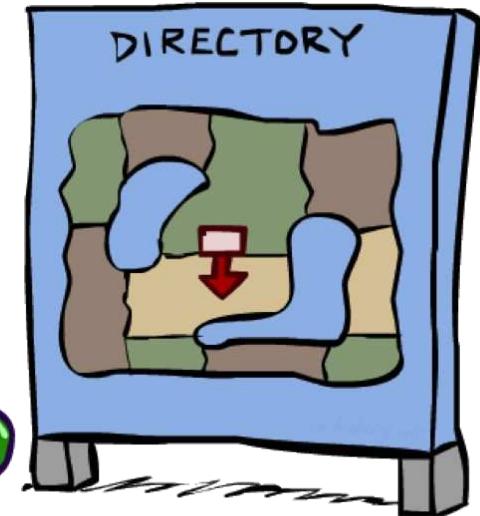
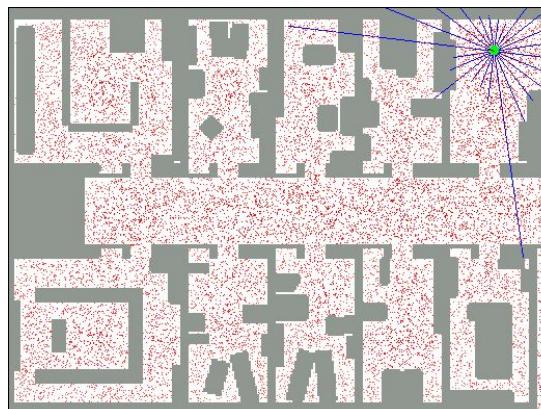


Particle Filtering in Ghostbusters - Many Particles

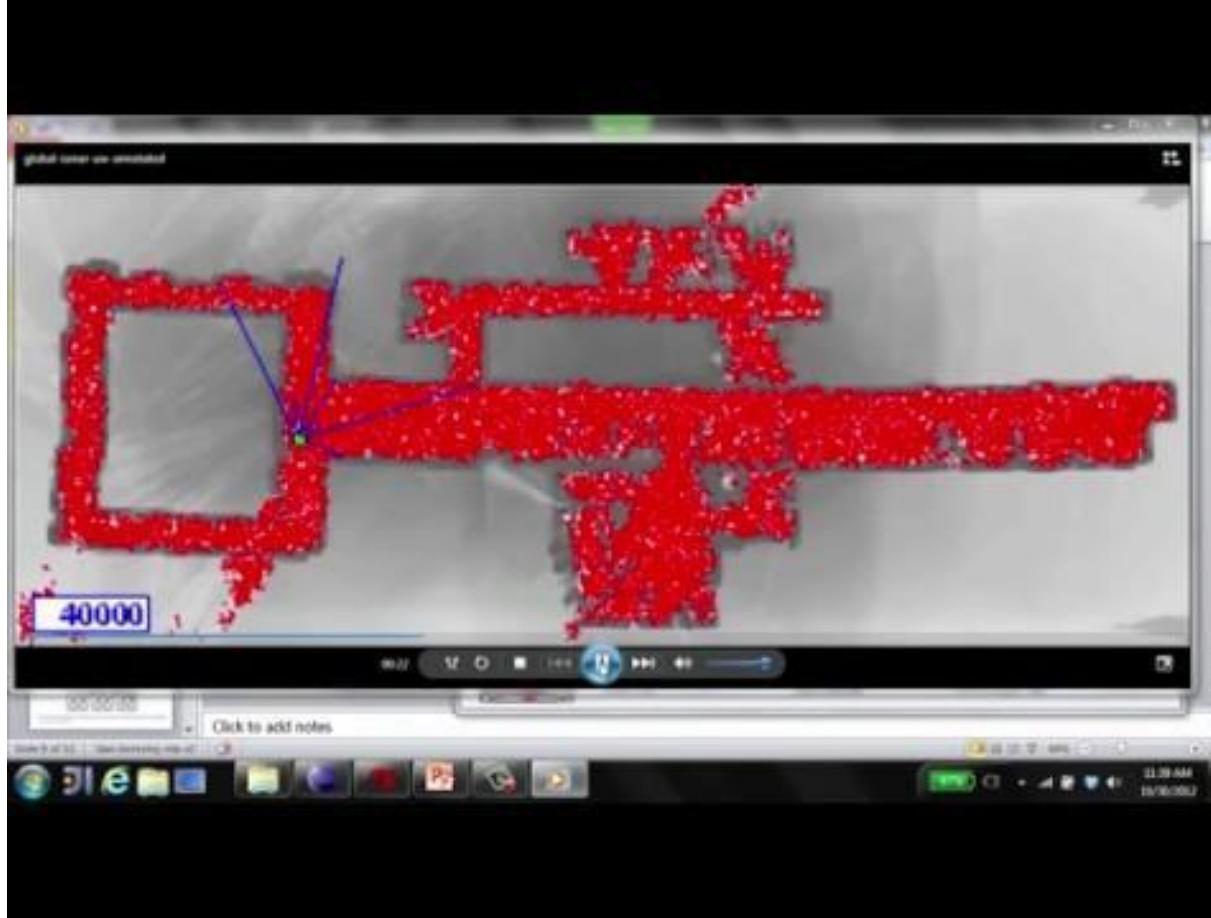


Robot Localization

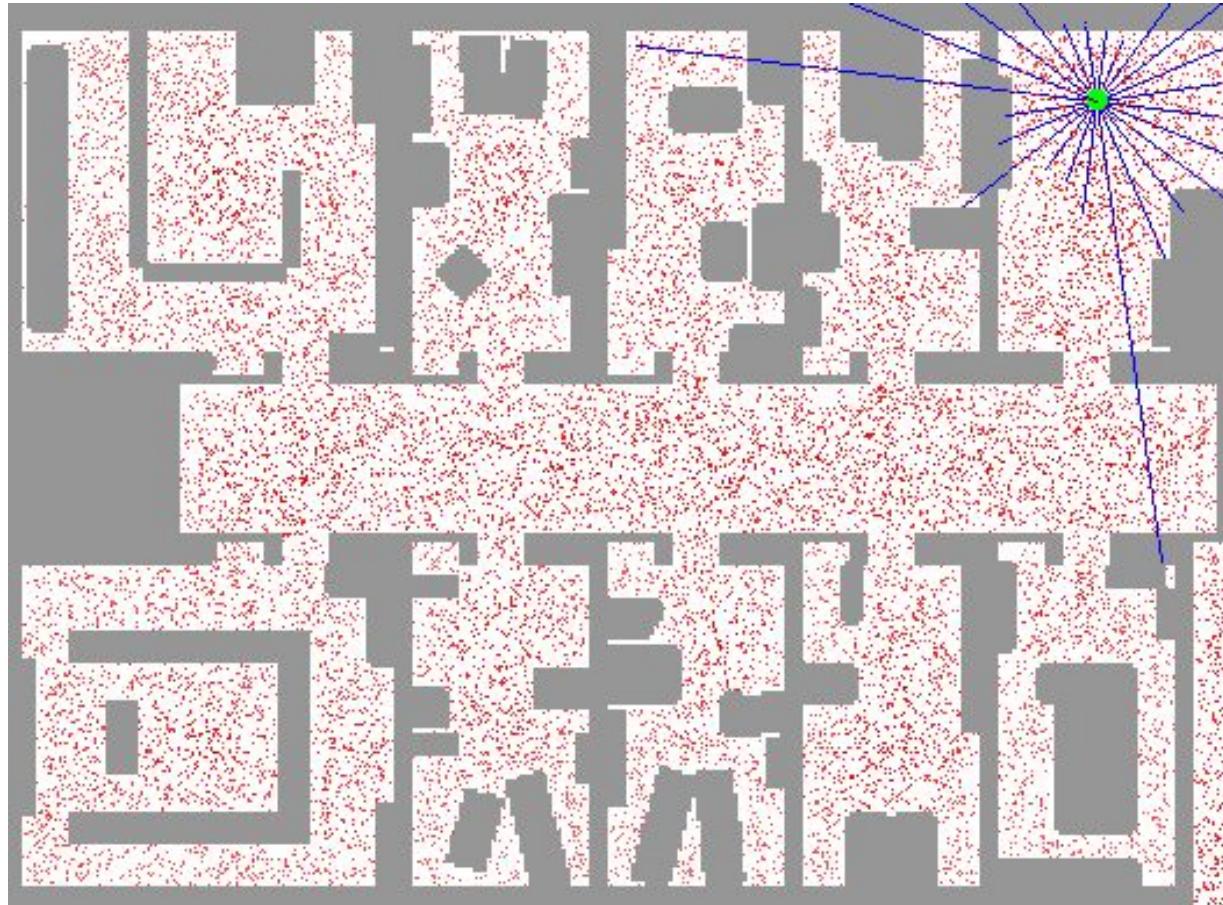
- In robot localization:
 - We know the map, but not the robot's position
 - Observations may be vectors of range finder readings
 - State space and readings are typically continuous (works basically like a very fine grid) and so we cannot store $B(X)$
 - Particle filtering is a main technique



Particle Filter Localization (Sonar)

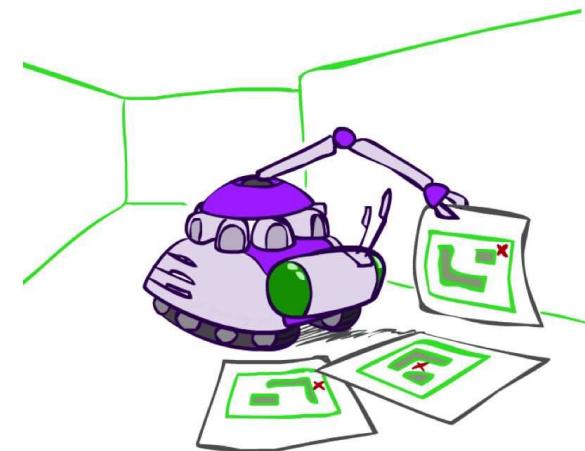


Particle Filter Localization (Laser)

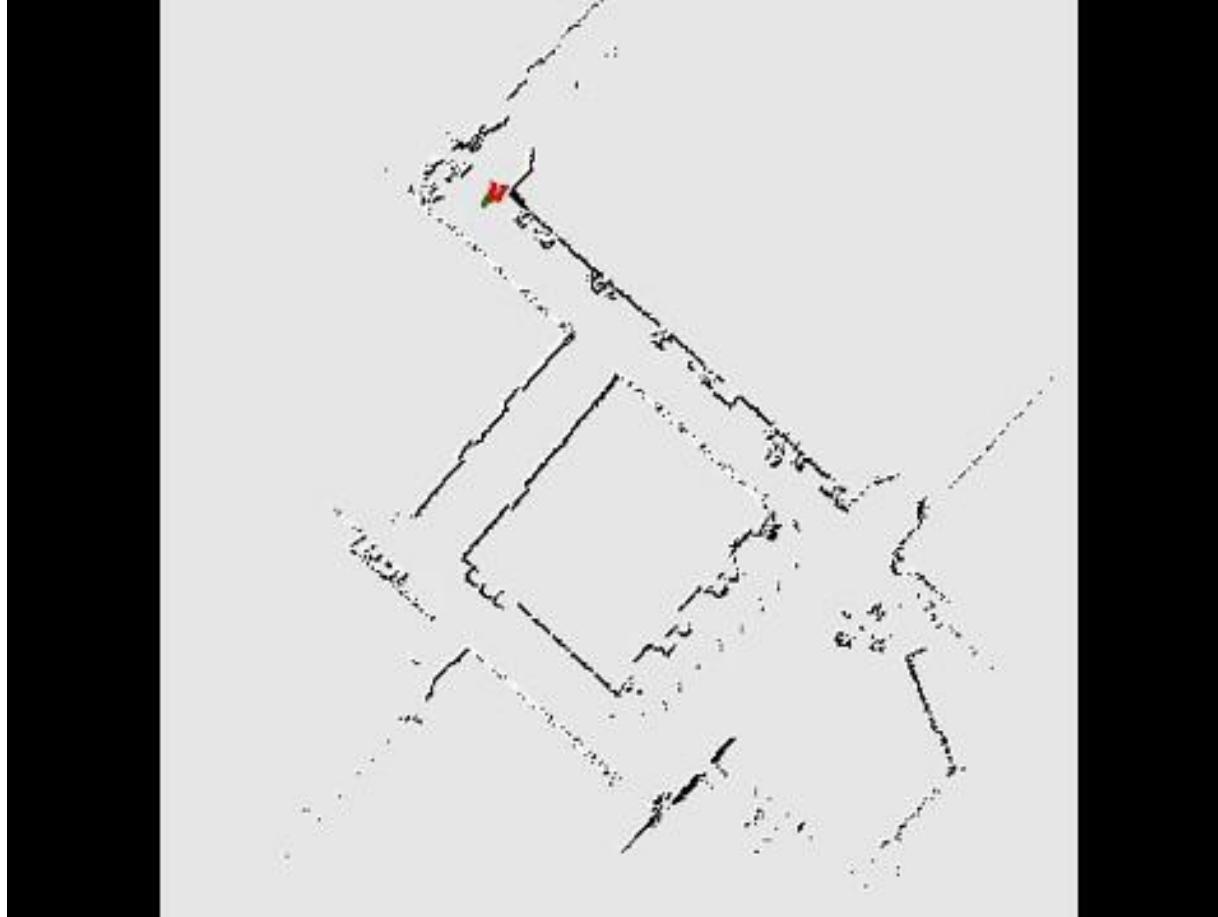


Robot Mapping

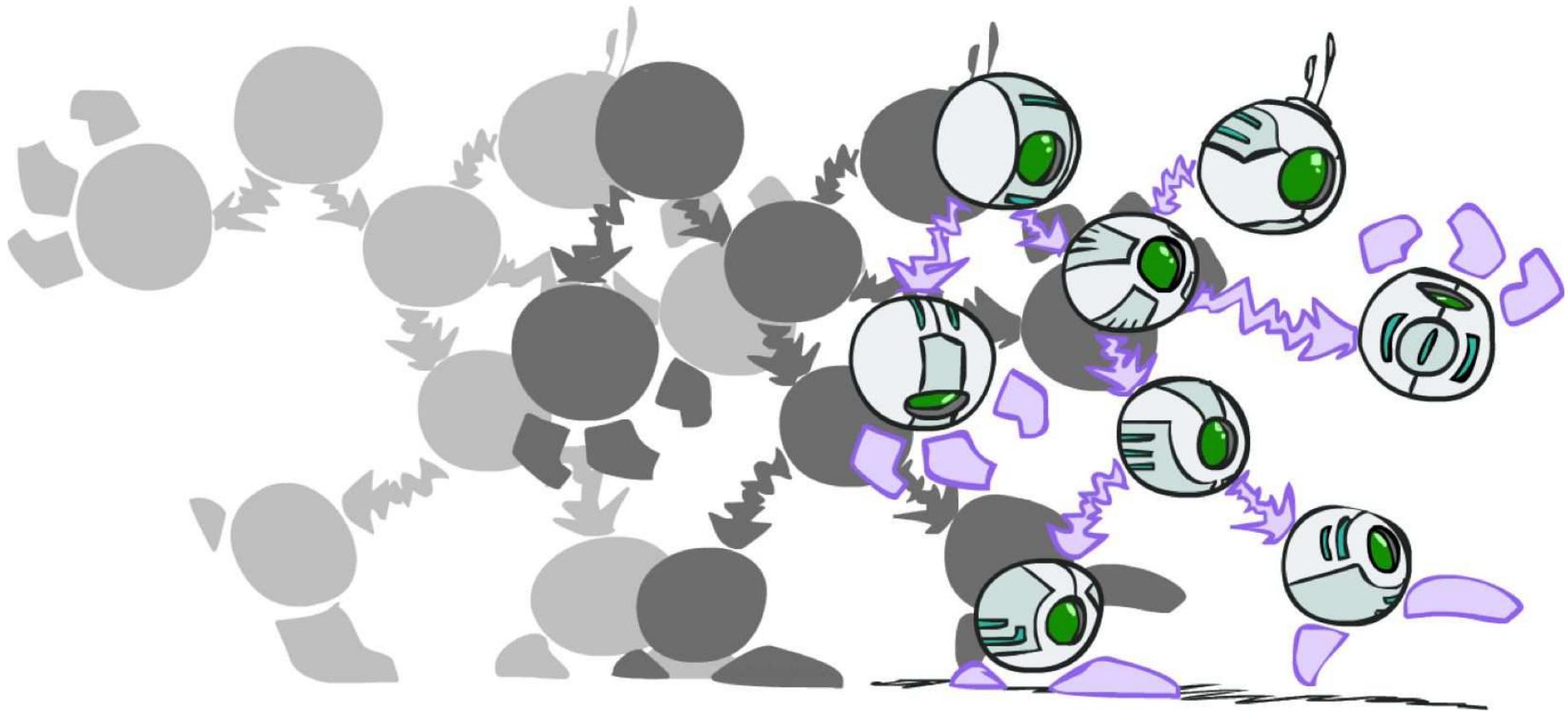
- SLAM: Simultaneous Localization And Mapping
 - We do not know the map or our location
 - State consists of position AND map!
 - Main techniques: Kalman filtering (Gaussian HMMs) and particle methods



SLAM Example

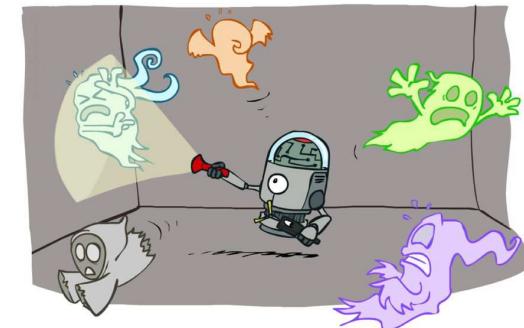
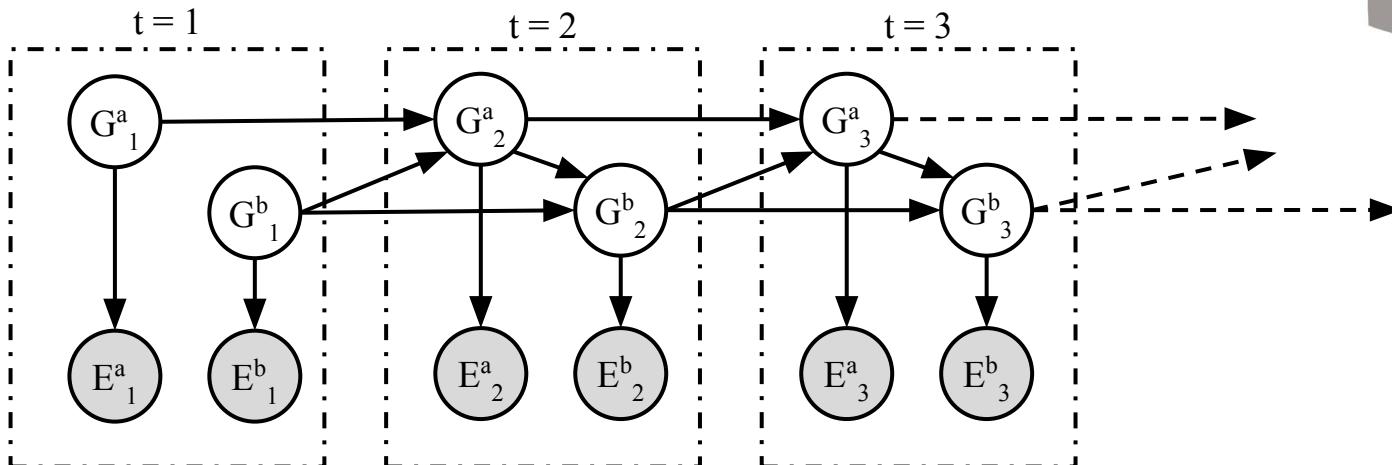


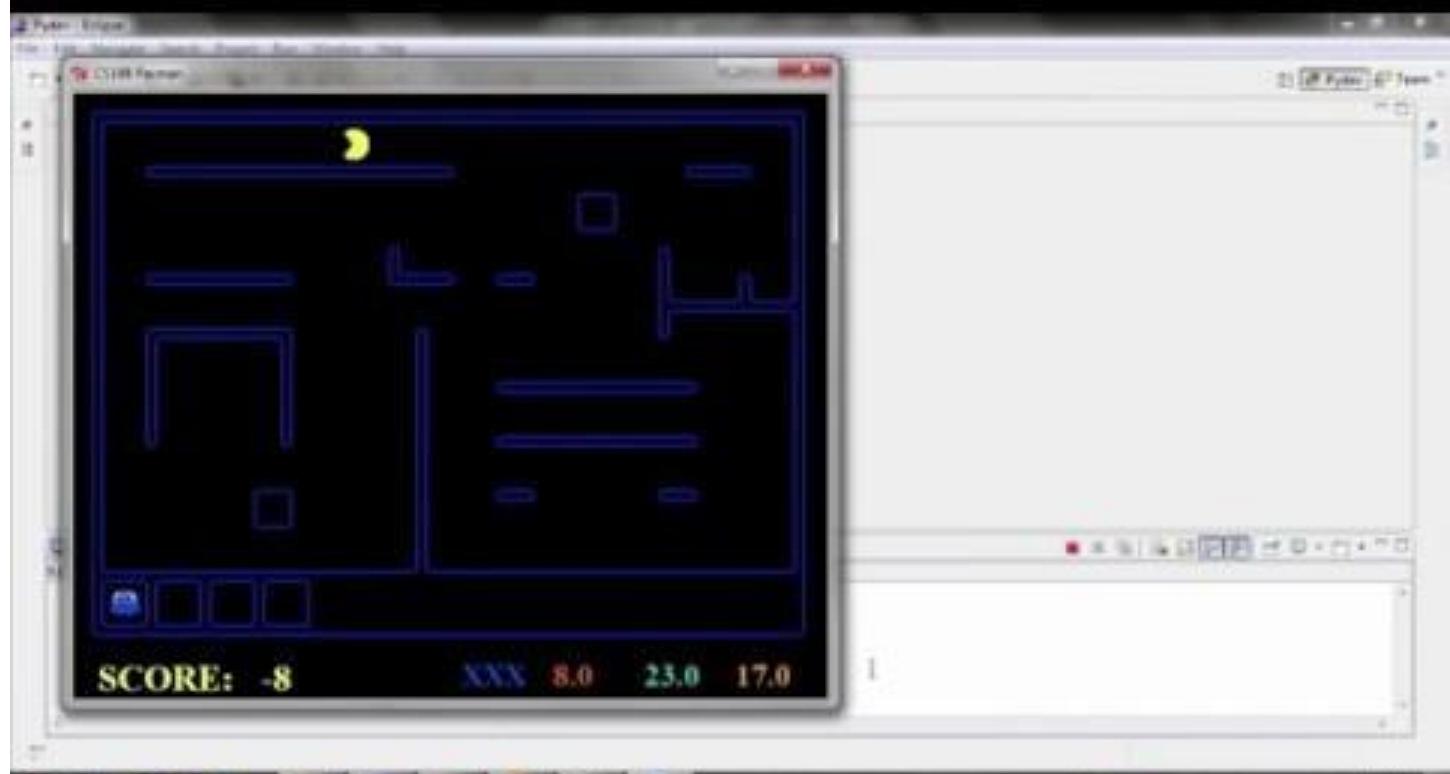
Dynamic Bayes Nets



Dynamic Bayes Nets (DBNs)

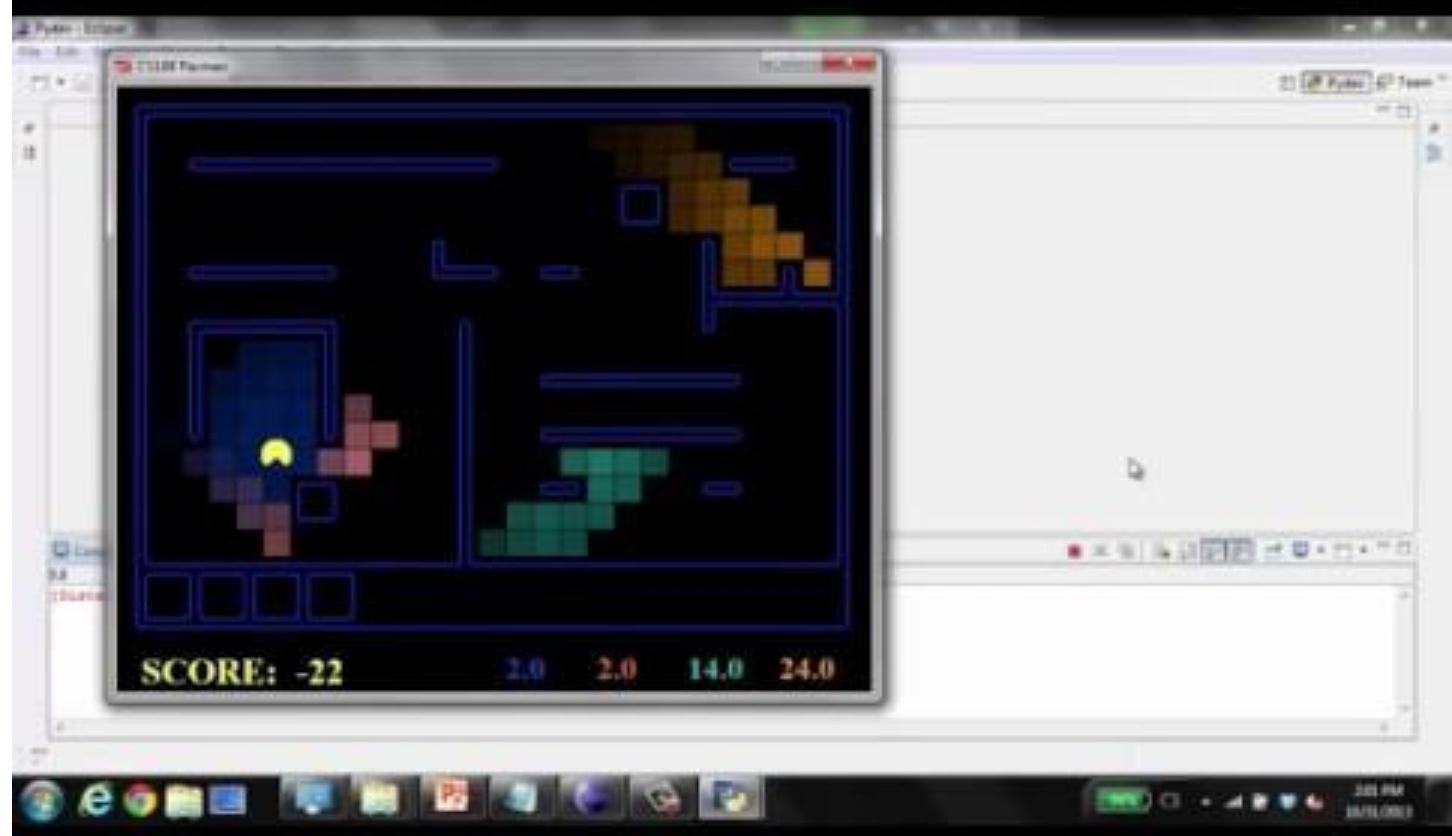
- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net (BN) structure at each time
 - More on BNs later in the course
- Variables from time t can condition on those from $t-1$
- Dynamic Bayes nets are a generalization of HMMs





DBN Particle Filter

- A particle is a complete sample for a time step
- Initialize: Generate prior samples for the t=1 Bayes net
 - Example particle: $G^a_1 = (3,3)$ $G^b_1 = (5,3)$
- Elapse time: Sample a successor for each particle
 - Example successor: $G^a_2 = (2,3)$ $G^b_2 = (6,3)$
- Observe: Weight each entire sample by the likelihood of the evidence conditioned on the sample
 - Likelihood: $P(E^a_1 | G^a_1) * P(E^b_1 | G^b_1)$
- Resample: Select prior samples (tuples of values) in proportion to their likelihood

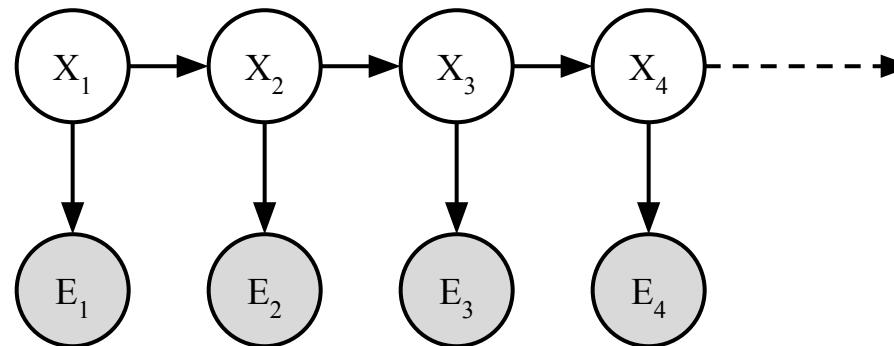


Most Likely Explanation

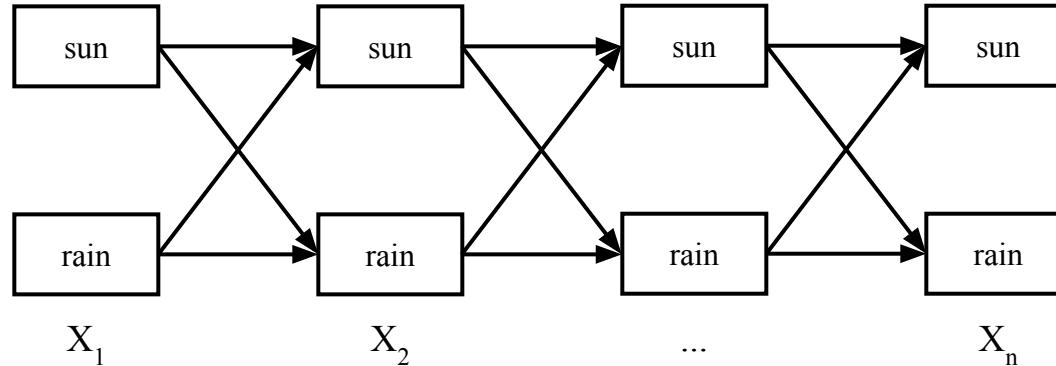


HMMs: MLE Queries

- HMMs defined by
 - States X
 - Observations E
 - Initial distribution: $P(X_1)$
 - Transitions: $P(X|X_{-1})$
 - Emissions: $P(E|X)$
- New query: most likely explanation: $\arg \max_{x_{1:t}} P(x_{1:t}|e_{1:t})$
- New method: the Viterbi algorithm

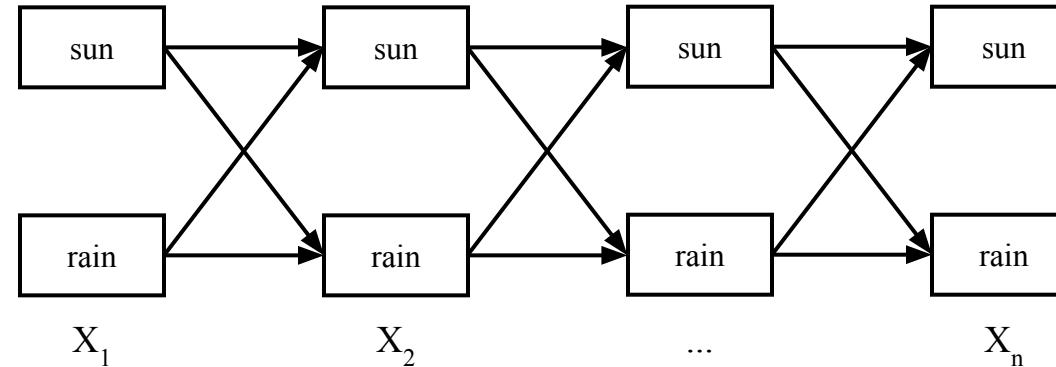


State Trellis



- State trellis: graph of states and transitions over time
- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t|x_{t-1})P(e_t|x_t)$
- Each path is a sequence of states
- The product of weights on a path is that sequence's probability along with the evidence
- Forward algorithm computes sums of paths, Viterbi computes best paths

Forward / Viterbi Algorithms



Forward Algorithm (Sum)

$$f_t[x_t] = P(x_t, e_{1:t})$$

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) f_{t-1}[x_{t-1}]$$

Viterbi Algorithm (Max)

$$m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$$

$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

Quiz 7

- Solve the problem on the handout sheet 1

Quiz 8

- Solve the problem on the handout sheet 2



Chapter 6

Bayes' Nets

COMP 3270
Artificial Intelligence

Dirk Schnieders

Outline

- Part A: Representation
- Part B: Independence
- Part C: Inference

Probabilistic Models

- Models describe how (a portion of) the world works
- Models are always simplifications
 - May not account for every variable
 - May not account for all interactions between variables
 - “All models are wrong; but some are useful.”
– George E. P. Box
- What do we do with probabilistic models?
 - We (or our agents) need to reason about unknown variables, given evidence
 - Example: explanation (diagnostic reasoning)
 - Example: prediction (causal reasoning)
 - Example: value of information



Independence

- Two variables are independent in a joint distribution if

$$P(X, Y) = P(X)P(Y)$$

$$X \perp\!\!\!\perp Y$$

$$\forall x, y P(x, y) = P(x)P(y)$$



- Says the joint distribution factors into a product of two simple ones
- Note: Usually, in practice, variables are not independent
- Can use independence as a modeling assumption
 - Simplifying assumption
 - Empirical joint distributions: at best “close” to independent
 - What could we assume for {Weather, Traffic, Cavity}?

Example: Independence check

- Given: Distribution
- Task: Check for independence

T and W are not independent

T	W	$P_1(T, W)$
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3



T	P(T)
hot	0.5
cold	0.5

W	P(W)
sun	0.6
rain	0.4

T and W are Independent

T	W	$P_2(T, W)$
hot	sun	0.3
hot	rain	0.2
cold	sun	0.3
cold	rain	0.2

Example: Independence

- N fair, independent coin flips

$$P(X_1)$$

H	0.5
T	0.5

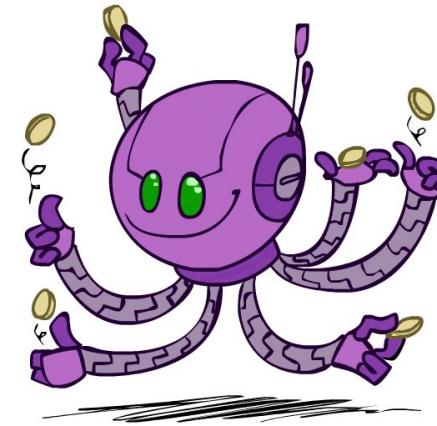
$$P(X_2)$$

H	0.5
T	0.5

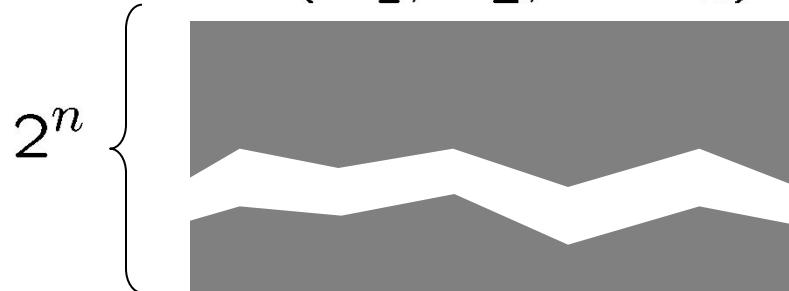
...

$$P(X_n)$$

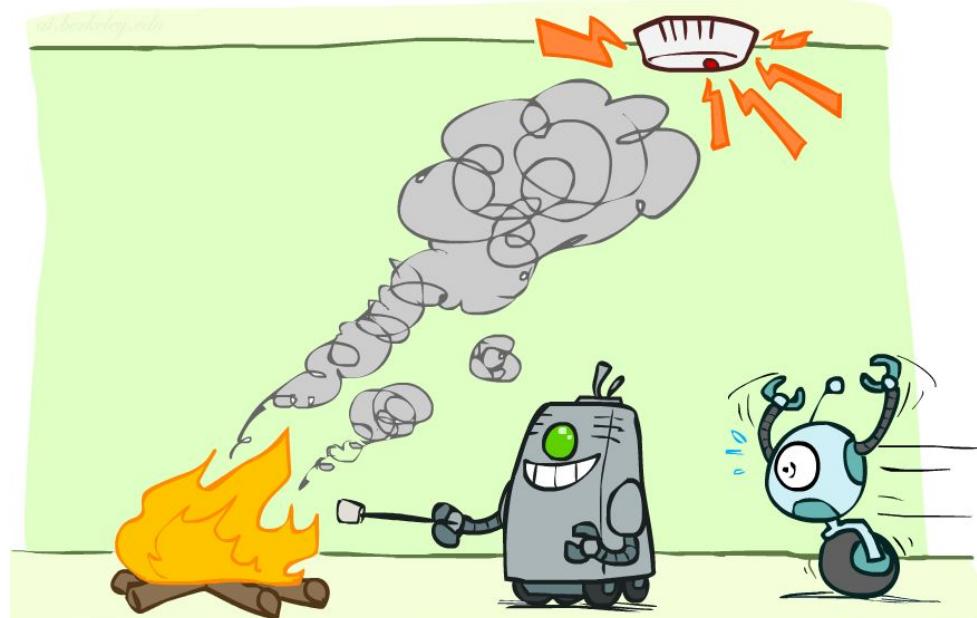
H	0.5
T	0.5



$$P(X_1, X_2, \dots, X_n)$$

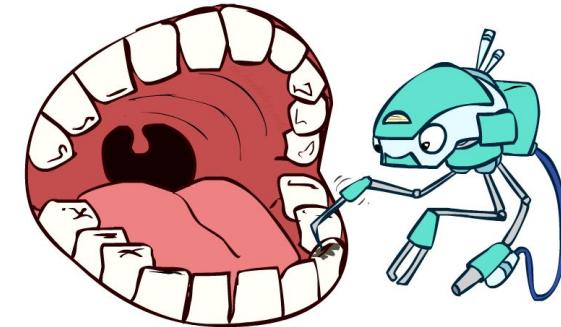


Conditional Independence



Conditional Independence

- $P(\text{Toothache}, \text{Cavity}, \text{Catch})$
- If I have a cavity, the probability that the probe catches in it doesn't depend on whether I have a toothache:
 - $P(+\text{catch} | +\text{toothache}, +\text{cavity}) = P(+\text{catch} | +\text{cavity})$
- The same independence holds if I don't have a cavity:
 - $P(+\text{catch} | +\text{toothache}, -\text{cavity}) = P(+\text{catch} | -\text{cavity})$
- Catch is conditionally independent of Toothache given Cavity:
 - $P(\text{Catch} | \text{Toothache}, \text{Cavity}) = P(\text{Catch} | \text{Cavity})$
- Equivalent statements:
 - $P(\text{Toothache} | \text{Catch}, \text{Cavity}) = P(\text{Toothache} | \text{Cavity})$
 - $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) P(\text{Catch} | \text{Cavity})$
 - One can be derived from the other easily



Conditional Independence

- Unconditional (absolute) independence very rare (why?)
- Conditional independence is our most basic and robust form of knowledge about uncertain environments

$$X \perp\!\!\!\perp Y | Z$$

- X is conditionally independent of Y given Z iff

$$\forall x, y, z : P(x, y | z) = P(x | z)P(y | z)$$

- or, equivalently, iff

$$\forall x, y, z : P(x | z, y) = P(x | z)$$

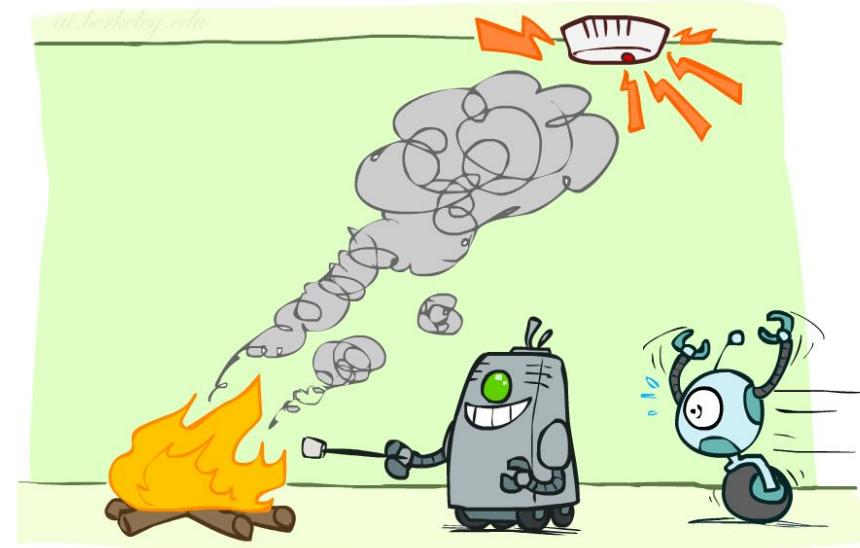
Quiz: Conditional Independence

- What about this domain
 - Traffic (T)
 - Umbrella (U)
 - Raining (R)



Quiz: Conditional Independence

- What about this domain
 - Fire (F)
 - Smoke (S)
 - Alarm (A)



Conditional Independence and the Chain Rule

- Chain rule: $P(X_1, X_2, \dots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \dots$
- Trivial decomposition:
$$\begin{aligned} P(\text{Traffic, Rain, Umbrella}) &= \\ P(\text{Rain})P(\text{Traffic}|\text{Rain})P(\text{Umbrella}|\text{Rain, Traffic}) \end{aligned}$$
- With assumption of conditional independence:
$$\begin{aligned} P(\text{Traffic, Rain, Umbrella}) &= \\ P(\text{Rain})P(\text{Traffic}|\text{Rain})P(\text{Umbrella}|\text{Rain}) \end{aligned}$$
- Bayes' nets / graphical models help us express conditional independence assumptions

Ghostbusters Chain Rule

- Each sensor depends only on where the ghost is
- That means, the two sensors are conditionally independent, given the ghost position

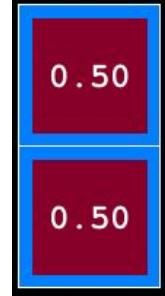
- T: Top square is red
- B: Bottom square is red
- G: Ghost is in the top

- Given:

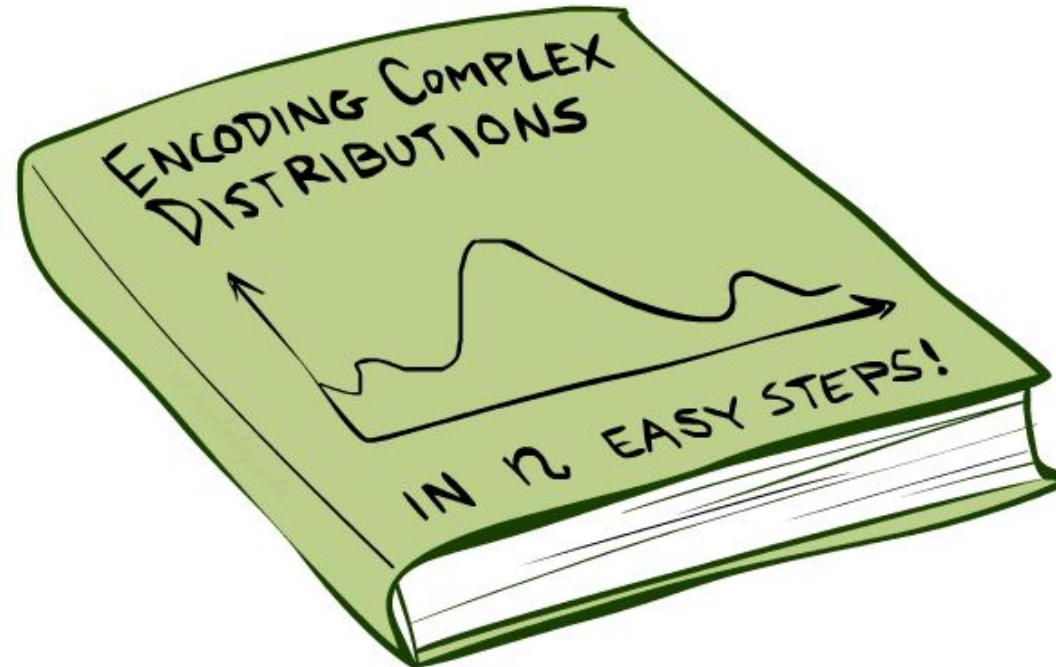
- $P(+g) = 0.5$
- $P(-g) = 0.5$
- $P(+t | +g) = 0.8$
- $P(+t | -g) = 0.4$
- $P(+b | +g) = 0.4$
- $P(+b | -g) = 0.8$

$$P(T, B, G) = P(G) P(T|G) P(B|G)$$

T	B	G	$P(T, B, G)$
+t	+b	+g	0.16
+t	+b	-g	0.16
+t	-b	+g	0.24
+t	-b	-g	0.04
-t	+b	+g	0.04
-t	+b	-g	0.24
-t	-b	+g	0.06
-t	-b	-g	0.06

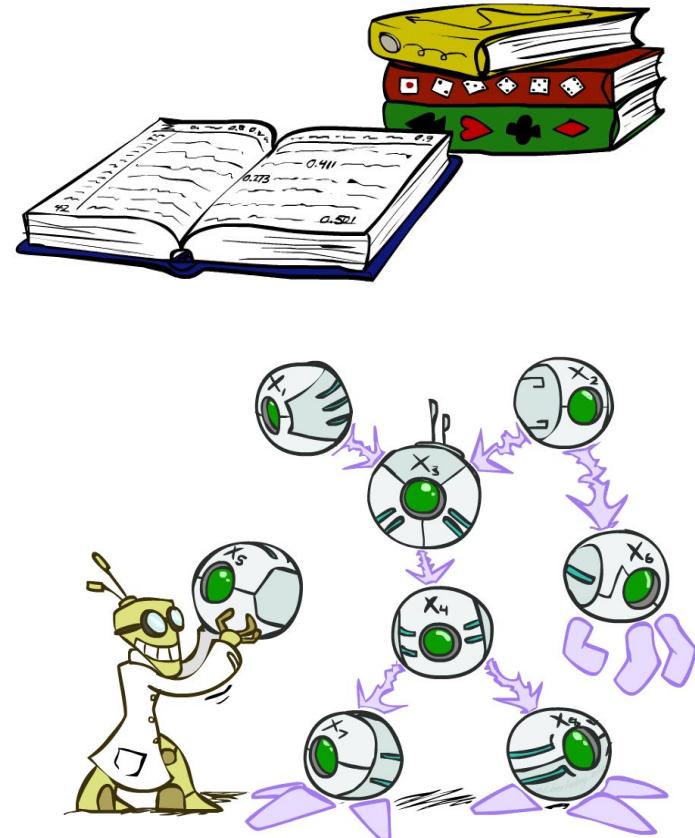


Bayes'Nets: Big Picture

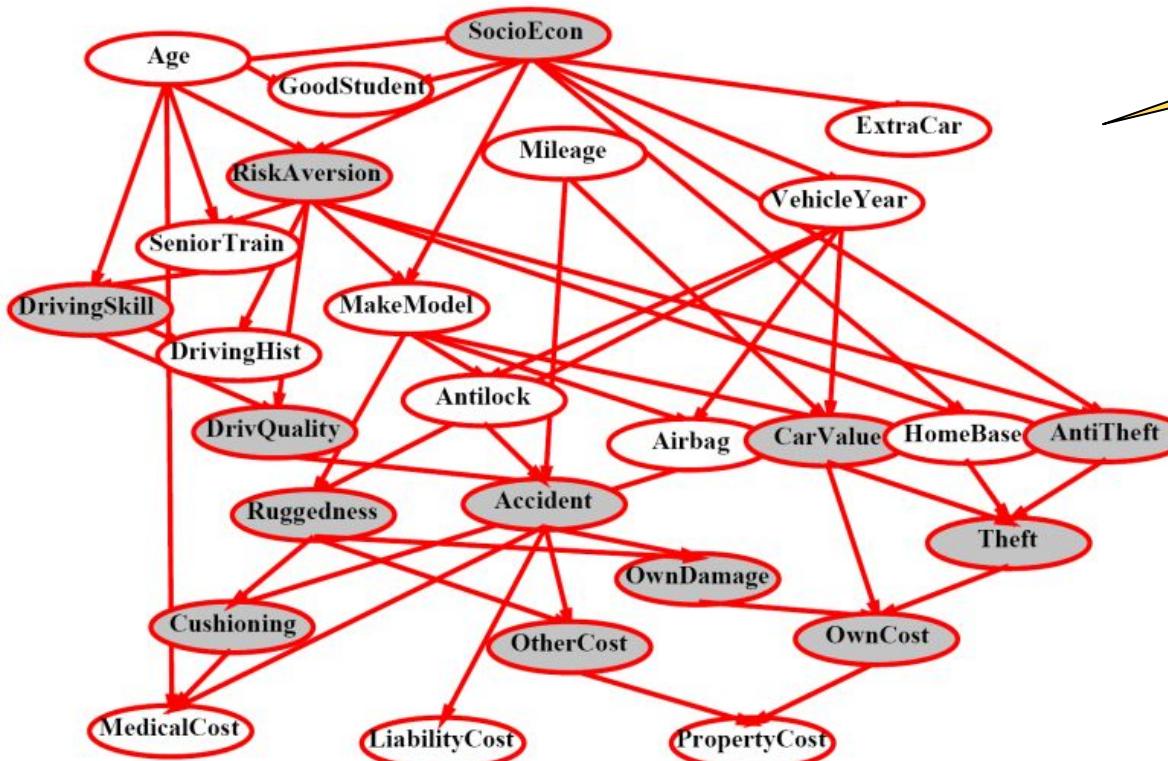


Bayes' Nets: Big Picture

- Two problems with using full joint distribution tables as our probabilistic models:
 - Unless there are only a few variables, the joint is WAY too big to represent explicitly
 - Hard to learn (estimate) anything empirically about more than a few variables at a time
- Bayes' nets: a technique for describing complex joint distributions (models) using simple, local distributions (conditional probabilities)
 - More properly called graphical models
 - We describe how variables locally interact
 - Local interactions chain together to give global, indirect interactions



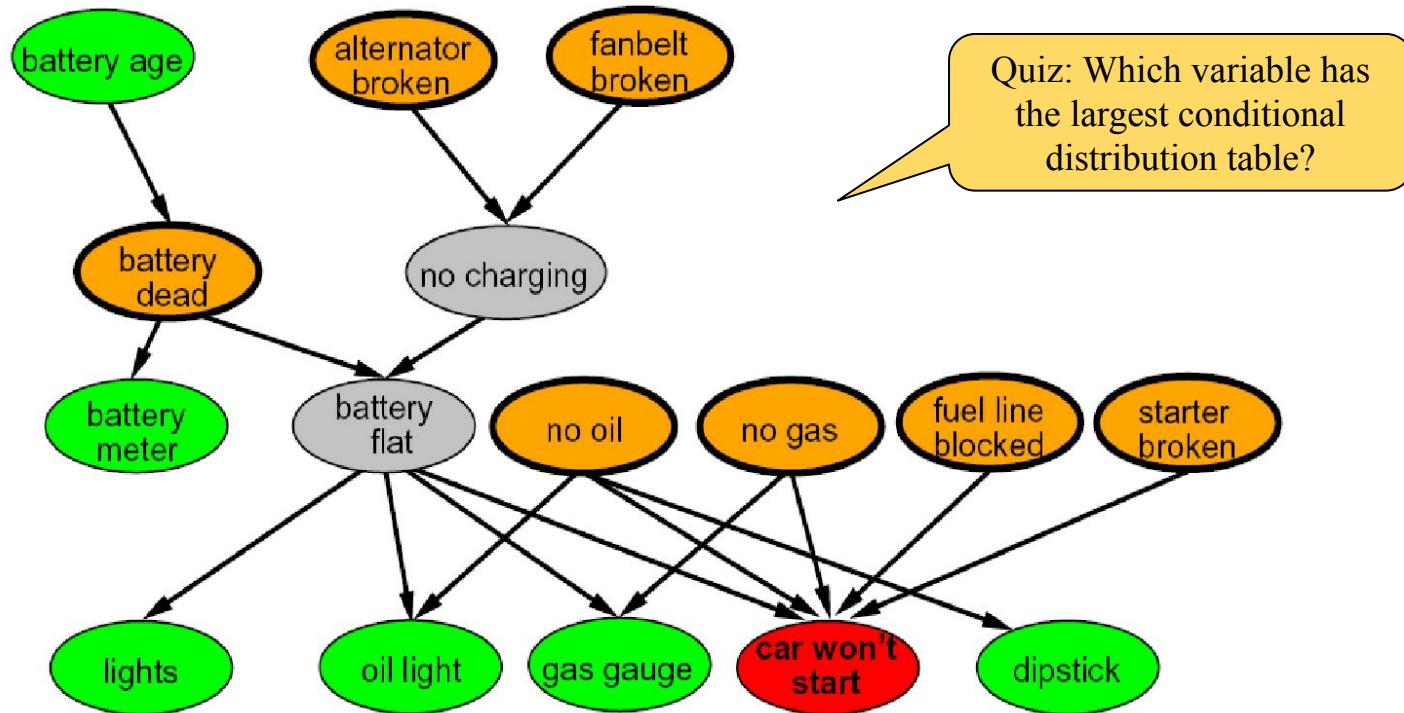
Example Bayes' Net: Insurance



27 variables

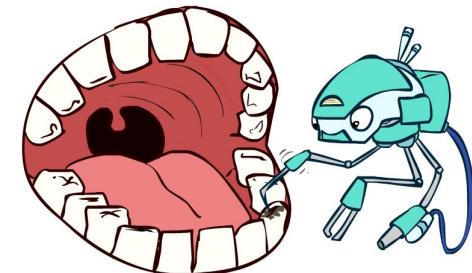
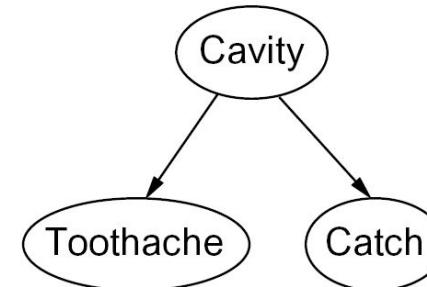
Quiz: How many entries
in the joint distribution?

Example Bayes' Net: Car



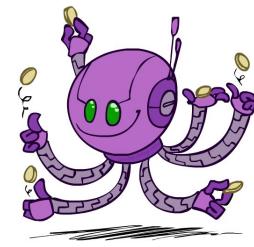
Graphical Model Notation

- Nodes: variables (with domains)
 - Can be assigned (observed) or unassigned (unobserved)
- Arcs: interactions
 - Similar to CSP constraints
 - Indicate “direct influence” between variables
 - Formally: encode conditional independence (more later)
- For now: imagine that arrows mean direct causation (in general, they don’t!)



Example: Coin Flips

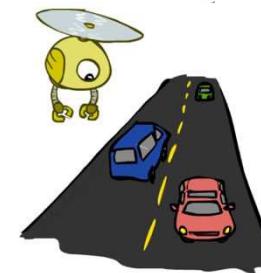
- N independent coin flips


$$X_1$$
$$X_2$$
$$\dots$$
$$X_n$$

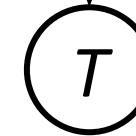
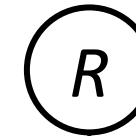
- No interactions between variables: absolute independence

Example: Traffic

- Variables:
 - R: It rains
 - T: There is traffic
- Model 1: independence



- Model 2: rain causes traffic



- Quiz: Why is an agent using model 2 better?

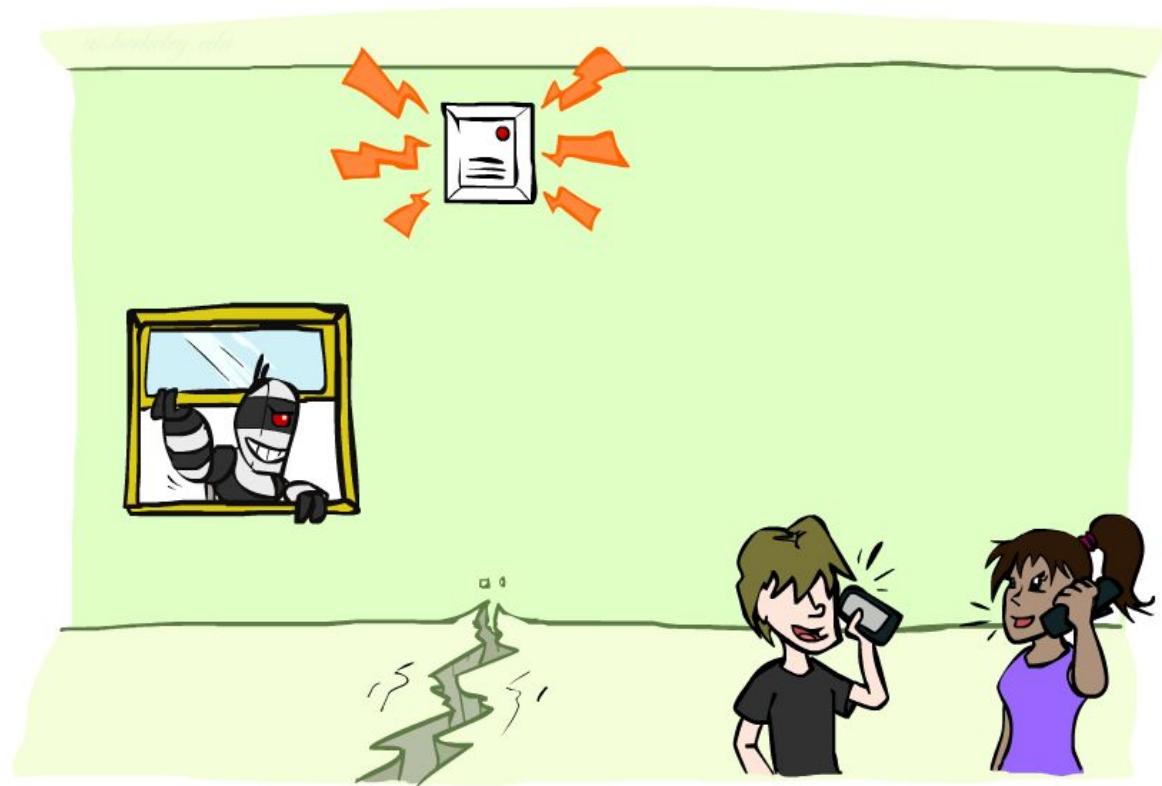
Quiz: Traffic II

- Build a causal graphical model!
- Variables
 - T: Traffic
 - R: It rains
 - L: Low pressure
 - D: Roof drips
 - B: Ballgame
 - C: Cavity



Quiz: Alarm Network

- Build a causal graphical model!
- Variables
 - B: Burglary
 - A: Alarm goes off
 - M: Mary calls
 - J: John calls
 - E: Earthquake!



Bayes' Net Semantics

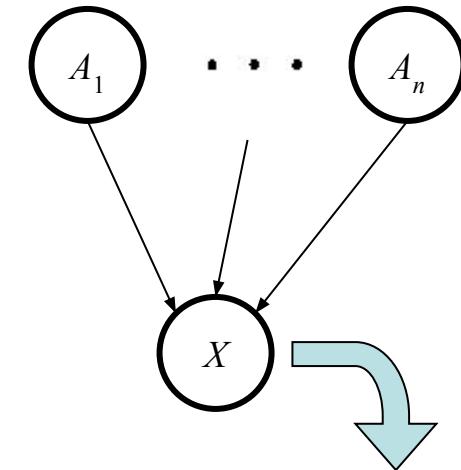


Bayes' Net Semantics

- A set of nodes, one per variable X
- A directed, acyclic graph
- A conditional distribution for each node
 - A collection of distributions over X , one for each combination of parents' values

$$P(X|a_1 \dots a_n)$$

- CPT: conditional probability table



$$P(X|A_1 \dots A_n)$$

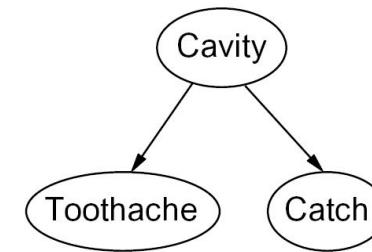
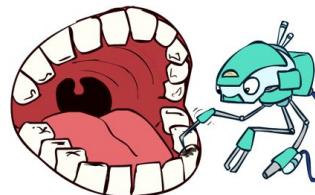
A Bayes net = Topology (graph) + Local Conditional Probabilities

Probabilities in BNs

- Bayes' nets implicitly encode joint distributions
 - As a product of local conditional distributions
 - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

- Quiz:



$$P(+\text{cavity}, +\text{catch}, -\text{toothache})$$

Probabilities in BNs

- Why are we guaranteed that setting

$$P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$$

results in a proper joint distribution?

- Chain rule (valid for all distributions): $P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i | x_1 \dots x_{i-1})$
- Assume conditional independencies: $P(x_i | x_1, \dots, x_{i-1}) = P(x_i | \text{parents}(X_i))$
 - Consequence: $P(x_1, x_2, \dots, x_n) = \prod_{i=1}^n P(x_i | \text{parents}(X_i))$
- Not every BN can represent every joint distribution
 - The topology enforces certain conditional independencies

Quiz: Coin Flips

 X_1 $P(X_1)$

h	0.5
t	0.5

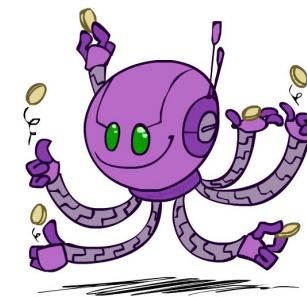
 X_2 $P(X_2)$

h	0.5
t	0.5

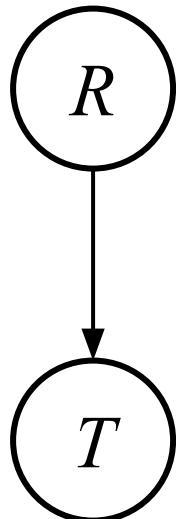
 \dots X_n $P(X_n)$

h	0.5
t	0.5

$$P(h, h, t, h) =$$



Quiz: Traffic



$$P(R)$$

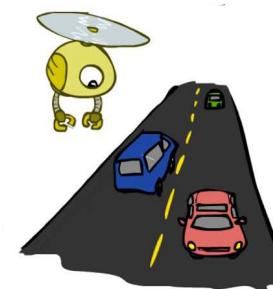
+r	1/4
-r	3/4

$$P(T|R)$$

+r	+t	3/4
	-t	1/4

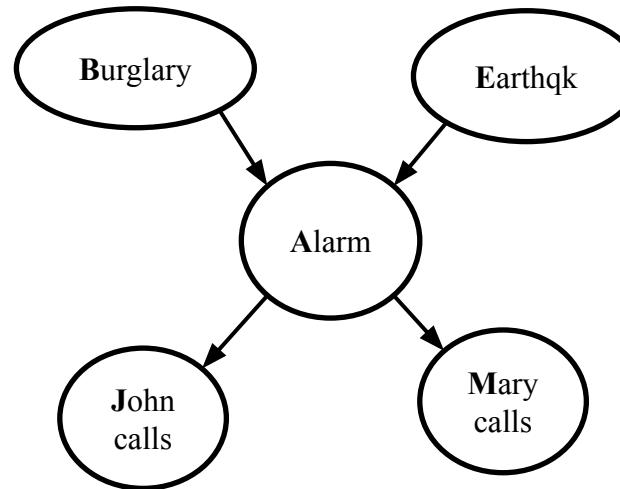
-r	+t	1/2
	-t	1/2

$$P(+r, -t) =$$



Example: Alarm Network

B	P(B)
+b	0.001
-b	0.999

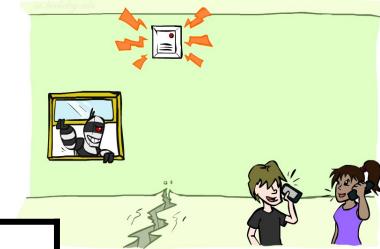


A	J	P(J A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

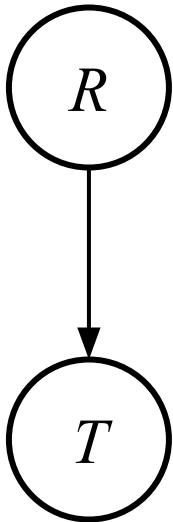
E	P(E)
+e	0.002
-e	0.998

B	E	A	P(A B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999



Example: Traffic

- Causal direction



$P(R)$

+r	1/4
-r	3/4

$P(T|R)$

+r	+t	3/4
	-t	1/4
-r	+t	1/2
	-t	1/2

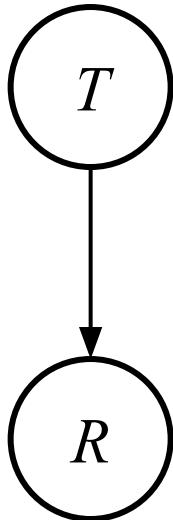


$P(T, R)$

+r	+t	3/16
+r	-t	1/16
-r	+t	6/16
-r	-t	6/16

Example: Reverse Traffic

- Reverse causality?



$P(T)$

+t	9/16
-t	7/16

$P(R|T)$

+t	+r	1/3
	-r	2/3
-t	+r	1/7
	-r	6/7

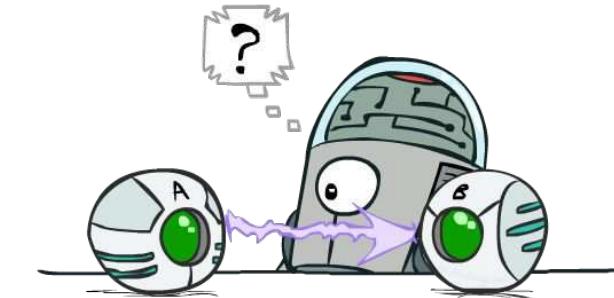


$P(T, R)$

+r	+t	3/16
+r	-t	1/16
-r	+t	6/16
-r	-t	6/16

Causality?

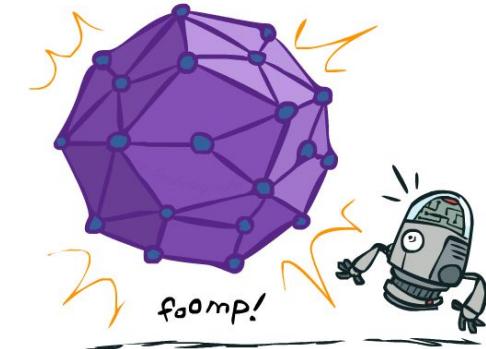
- When Bayes' nets reflect the true causal patterns
 - Often simpler (nodes have fewer parents)
 - Often easier to think about
 - Often easier to elicit from experts
- BNs need not actually be causal
 - Sometimes no causal net exists over the domain (especially if variables are missing)
 - E.g. consider the variables Traffic and Drips
 - End up with arrows that reflect correlation, not causation
- What do the arrows really mean?
 - Topology may happen to encode causal structure
 - Topology really encodes conditional independence



$$P(x_i|x_1, \dots x_{i-1}) = P(x_i|\text{parents}(X_i))$$

Size of a Bayes' Net

- A: How many entries (rows) are in a joint distribution table for a BN with N Boolean variables?
 - 2^N
- B: How many entries (rows) in all (C)PT tables for a BN with N Boolean variables where each variable has at most k parents?
 - $< N2^{k+1}$
- Both A and B give you the power to calculate $P(X_1, X_2, \dots, X_n)$
- BNs
 - Huge space savings!
 - Easier to elicit local CPTs
 - Faster to answer queries



Quiz: Suppose we have 30 nodes each with 5 parents.

- How many rows for A?
- How many rows for B?

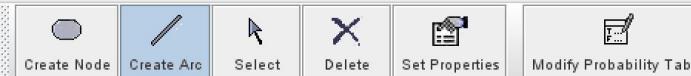
Demo

Belief and Decision Networks
<http://aispace.org/bayes/>



Create Solve

Click the canvas to create a node.

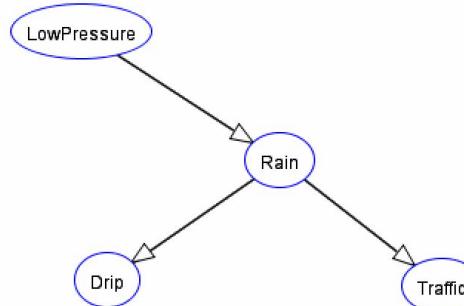


Create Solve

Click on a node to start creating an arc.

Click on another node to finish.

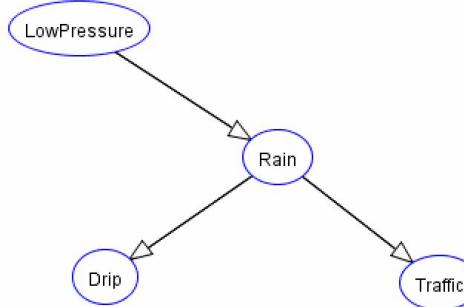
You can cancel arc creation by clicking on the canvas.





Create Solve

Click on an entity to select or drag the mouse to select multiple entities.



Outline

- Part A: Representation
- Part B: Independence
- Part C: Inference

Probability Recap

- Conditional probability $P(x|y) = \frac{P(x,y)}{P(y)}$
- Product rule $P(x,y) = P(x|y)P(y)$
- Chain rule
$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots \\ &= \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1}) \end{aligned}$$
- X, Y independent if and only if: $\forall x, y : P(x,y) = P(x)P(y)$
- X and Y are conditionally independent given Z if and only if:
$$\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)$$
 $X \perp\!\!\!\perp Y | Z$

Conditional Independence Recap

- X and Y are independent if

$$\forall x, y \ P(x, y) = P(x)P(y) \longrightarrow X \perp\!\!\!\perp Y$$

- X and Y are conditionally independent given Z

$$\forall x, y, z \ P(x, y|z) = P(x|z)P(y|z) \longrightarrow X \perp\!\!\!\perp Y|Z$$

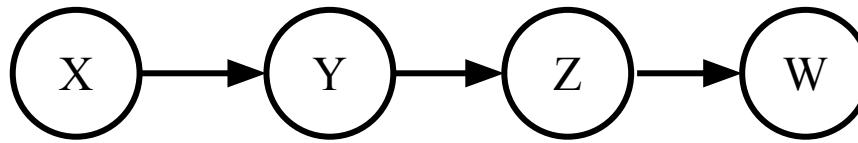
- (Conditional) independence is a property of a distribution

Bayes Nets: Assumptions

- Assumptions we are required to make to define the Bayes net when given the graph
$$P(x_i|x_1 \cdots x_{i-1}) = P(x_i|\text{parents}(X_i))$$
- Beyond above “chain rule → Bayes net” conditional independence assumptions
 - Often additional conditional independencies
 - They can be read off the graph
- Important for modeling: understand assumptions made when choosing a Bayes net graph



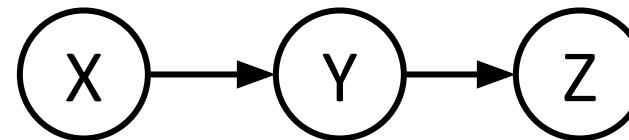
Quiz - Conditional Independence Assumptions



- a) Write down the conditional independence assumptions directly from simplifications in chain rule
- b) Write down additional implied conditional independence assumptions, if any

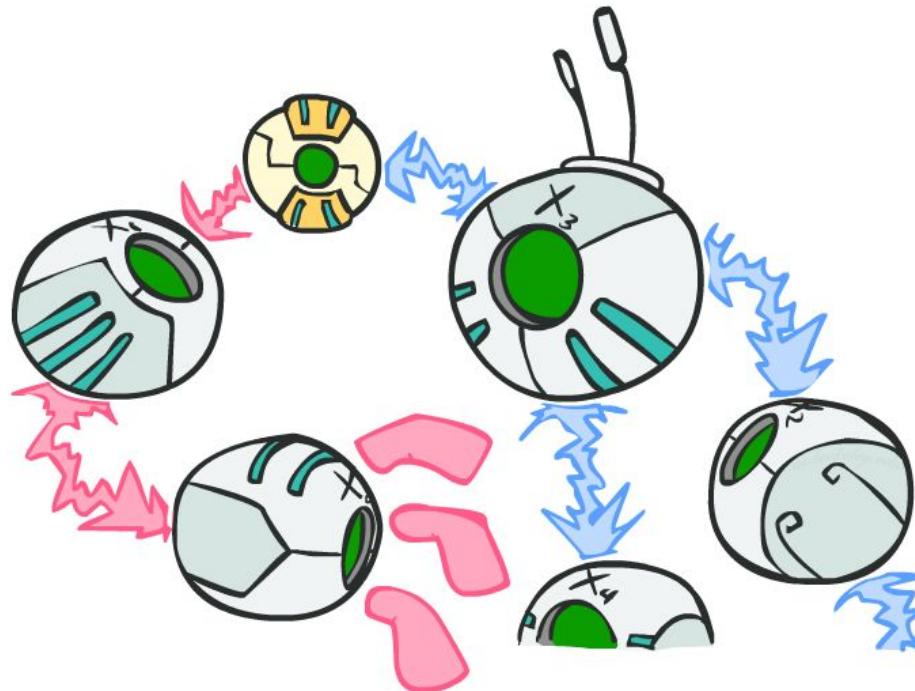
Independence in a BN

- Important question about a BN:
 - Are two nodes independent given certain evidence?
 - If yes, can prove using algebra (tedious in general)
 - If no, can prove with a counter example
 - Example:



- Question: are X and Z necessarily independent?
 - Answer: no. Example: low pressure causes rain, which causes traffic.
 - X can influence Z, Z can influence X (via Y)
 - Quiz: Could they be independent?

D-separation: Outline



D-separation: Outline

- Study independence properties for triples
- Analyze complex cases in terms of member triples
- D-separation: a condition / algorithm for answering such queries

Causal Chain

- This configuration is a “causal chain”



X: Low pressure

Y: Rain

Z: Traffic

$$P(x, y, z) = P(x)P(y|x)P(z|y)$$

- Guaranteed X independent of Z ?
 - No!
 - One example set of CPTs for which X is not independent of Z is sufficient to show this independence is not guaranteed.
 - Example:
 - Low pressure causes rain causes traffic, high pressure causes no rain causes no traffic
 - In numbers:
 - $P(+y | +x) = 1, P(-y | -x) = 1,$
 - $P(+z | +y) = 1, P(-z | -y) = 1$

Causal Chain

- This configuration is a “causal chain”



- Guaranteed X independent of Z given Y?

$$\begin{aligned} P(z|x,y) &= \frac{P(x,y,z)}{P(x,y)} \\ &= \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)} \\ &= P(z|y) \end{aligned}$$

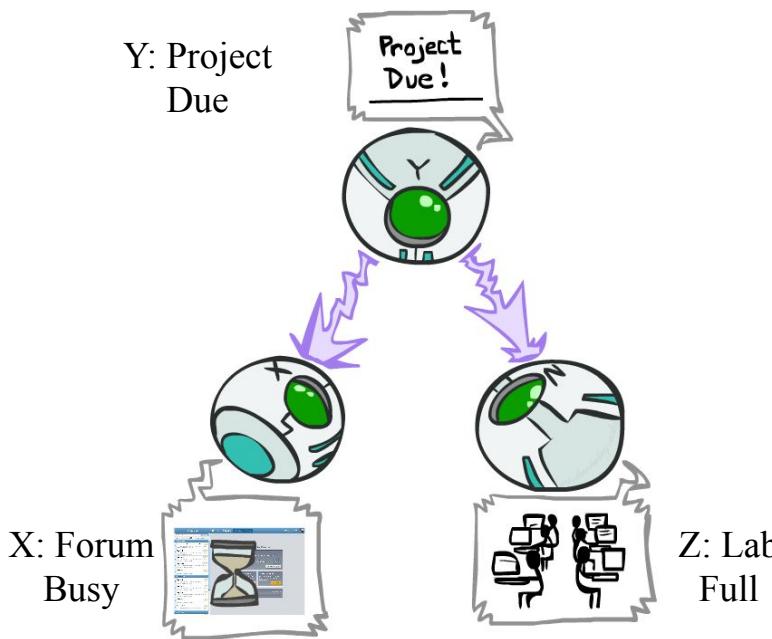
Yes!

Evidence along the chain “blocks” the influence

$$P(x,y,z) = P(x)P(y|x)P(z|y)$$

Common Cause

- This configuration is a “common cause”

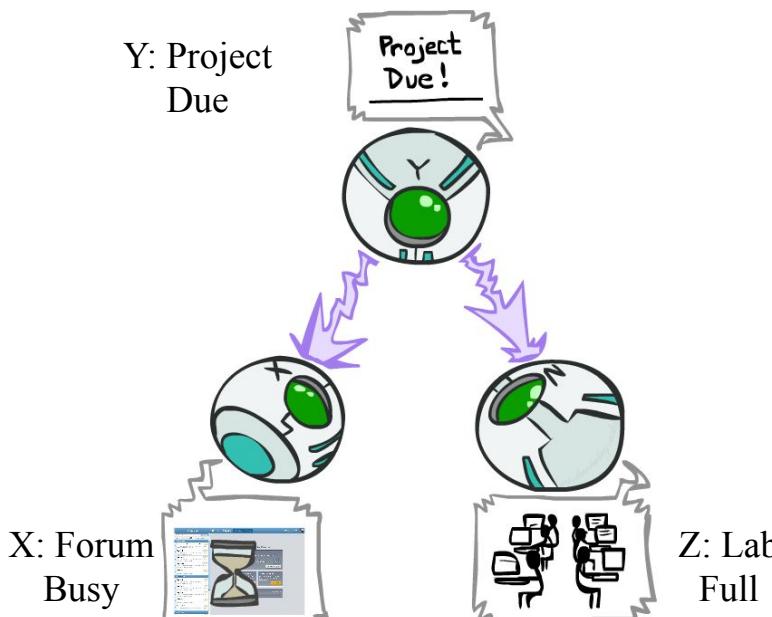


$$P(x, y, z) = P(y)P(x|y)P(z|y)$$

- Guaranteed X independent of Z ?
 - No!
 - One example set of CPTs for which X is not independent of Z is sufficient to show this independence is not guaranteed
 - Example:
 - Project due causes both forums busy and lab full
 - In numbers:
 - $P(+x | +y) = 1, P(-x | -y) = 1,$
 - $P(+z | +y) = 1, P(-z | -y) = 1$

Common Cause

- This configuration is a “common cause”



$$P(x, y, z) = P(y)P(x|y)P(z|y)$$

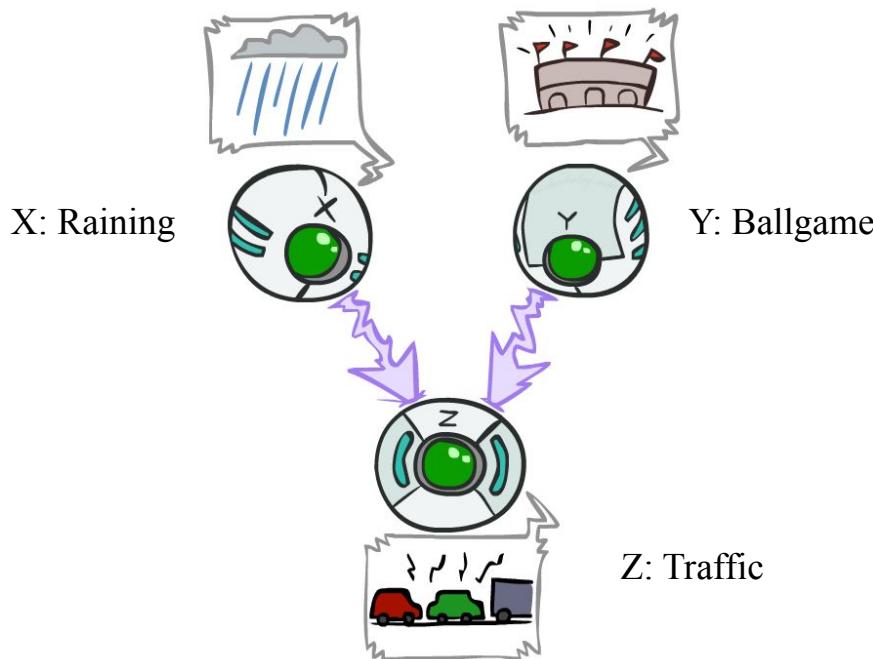
- Guaranteed X and Z independent given Y?

$$\begin{aligned} P(z|x, y) &= \frac{P(x, y, z)}{P(x, y)} \\ &= \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)} \\ &= P(z|y) \end{aligned}$$

Yes!
Observing the cause blocks influence between effects

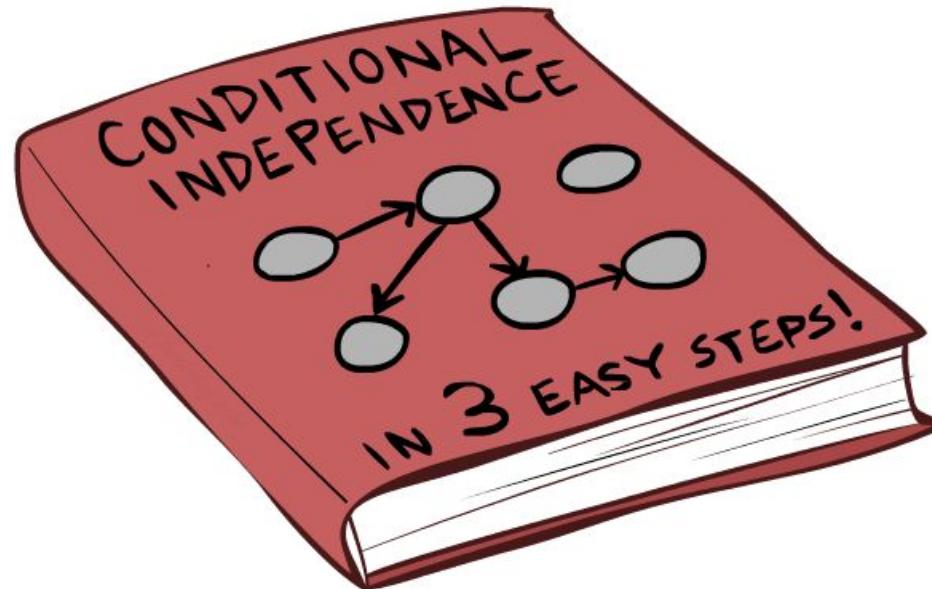
Common Effect

- Last configuration: two causes of one effect (v-structures)



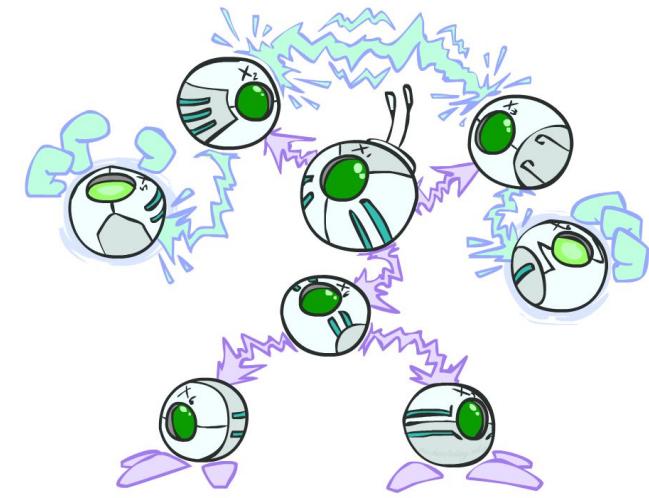
- Are X and Y independent?
 - Yes: the ballgame and the rain cause traffic, but they are not correlated
 - Quiz: Prove they must be
- Are X and Y independent given Z?
 - No: seeing traffic puts the rain and the ballgame in competition
- This is backwards from the other cases
 - Observing an effect activates influence between possible causes

The General Case



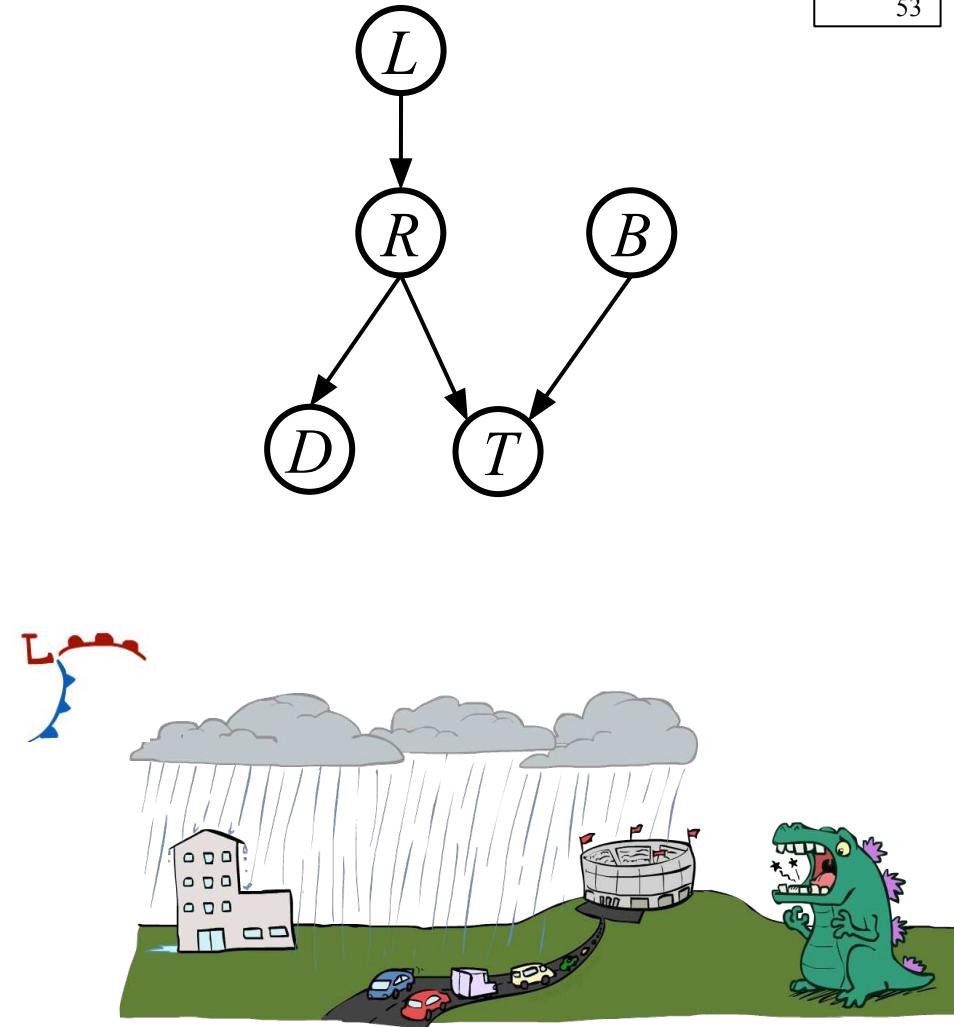
The General Case

- General question: in a given BN, are two variables independent (given evidence)?
- Solution: analyze the graph
- Any complex example can be broken into repetitions of the three canonical cases



Reachability

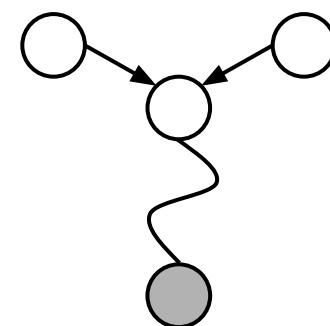
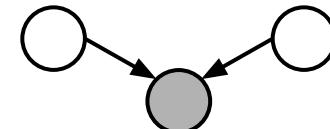
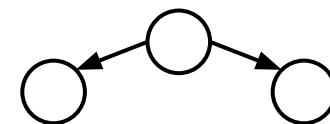
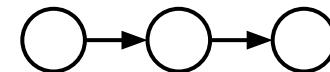
- Recipe
 - Shade evidence nodes, look for paths in the resulting graph
- Attempt 1
 - If two nodes are connected by an undirected path not blocked by a shaded node, they are conditionally independent
- Almost works, but not quite



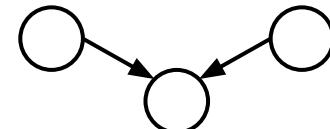
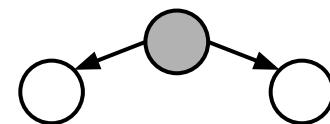
Active / Inactive Paths

- Question: Are X and Y conditionally independent given evidence variables $\{Z\}$?
 - Yes, if X and Y “d-separated” by Z
 - Consider all (undirected) paths from X to Y
 - No active paths = independence!
- A path is active if each triple is active
 - Causal chain $A \rightarrow B \rightarrow C$ where B is unobserved (either direction)
 - Common cause $A \leftarrow B \rightarrow C$ where B is unobserved
 - Common effect (aka v-structure)
 $A \rightarrow B \leftarrow C$ where B or one of its descendants is observed
- All it takes to block a path is a single inactive segment

Active Triples

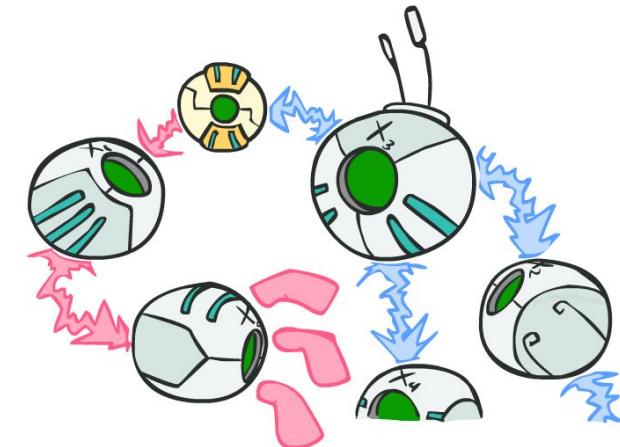


Inactive Triples



D-Separation

- Query: $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$
- Check all (undirected!) paths between X_i and X_j
- If one or more active, then independence not guaranteed
- Otherwise (i.e. if all paths are inactive), then independence is guaranteed



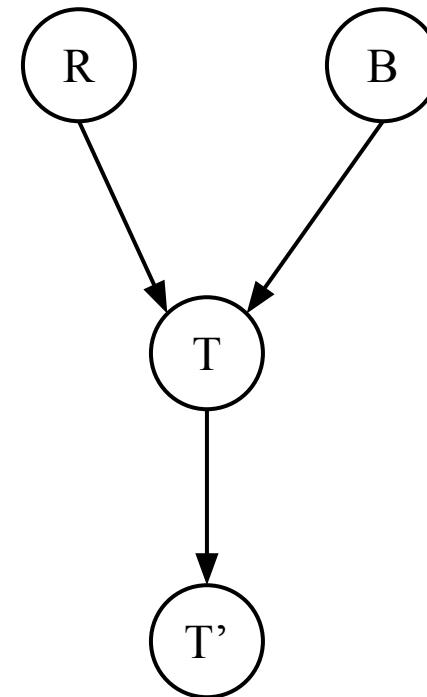
Example 1

$R \perp\!\!\!\perp B$

Yes

$R \perp\!\!\!\perp B | T$

$R \perp\!\!\!\perp B | T'$



Example 2

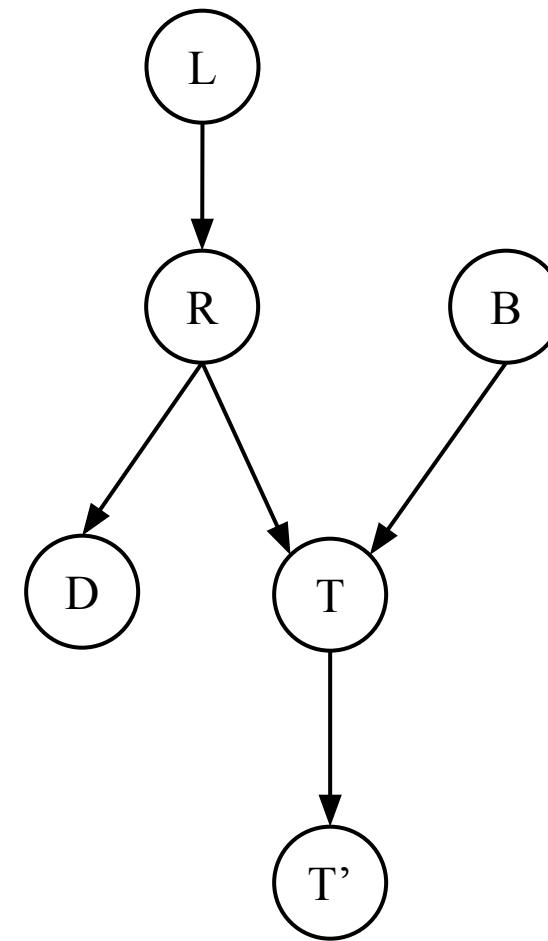
$L \perp\!\!\!\perp T' | T$ Yes

$L \perp\!\!\!\perp B$ Yes

$L \perp\!\!\!\perp B | T$

$L \perp\!\!\!\perp B | T'$

$L \perp\!\!\!\perp B | T, R$ Yes

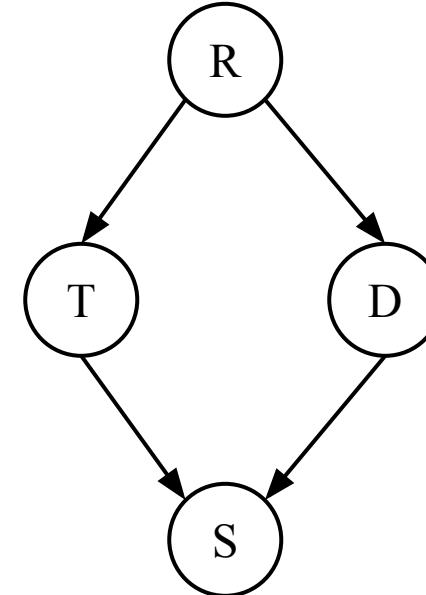


Example 3

$$T \perp\!\!\!\perp D$$

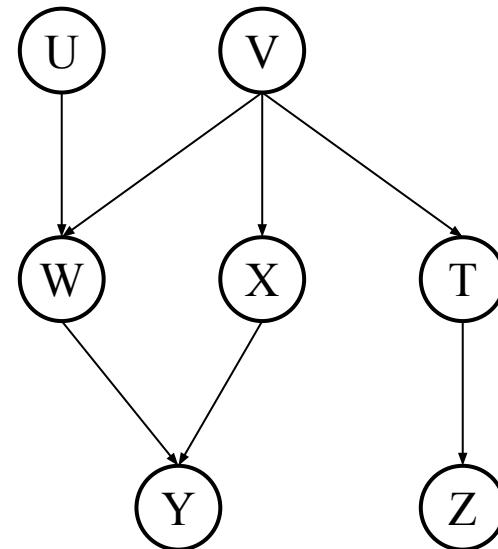
$$T \perp\!\!\!\perp D | R \quad \text{Yes}$$

$$T \perp\!\!\!\perp D | R, S$$



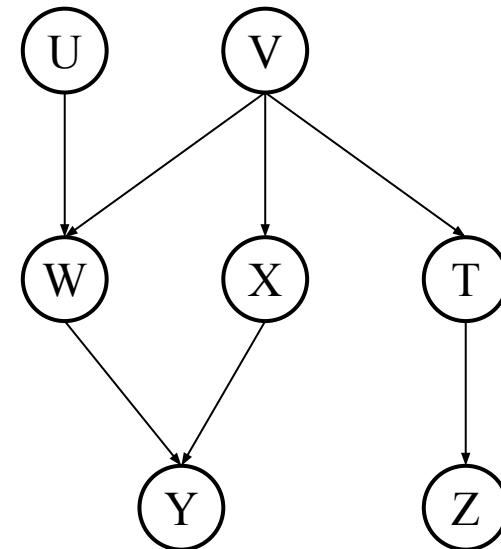
Quiz - 1

$V \perp Z ?$



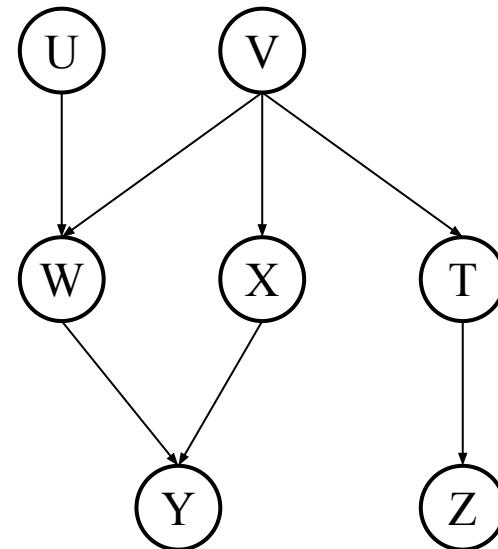
Quiz - 2

$V \perp Z | T ?$



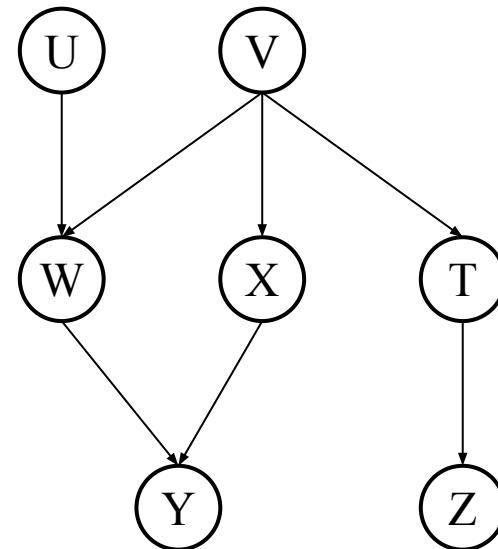
Quiz - 3

$U \perp V ?$



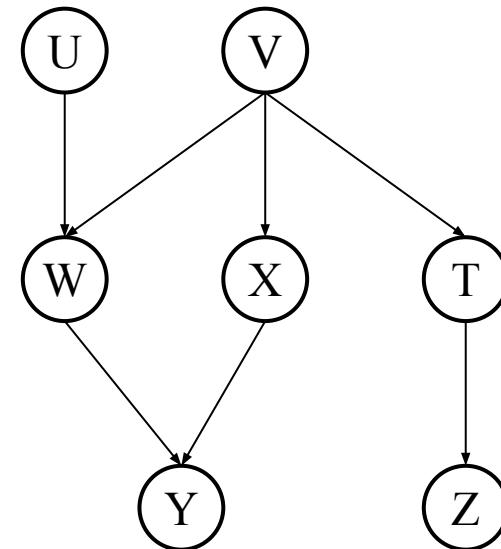
Quiz - 4

$U \perp V | W ?$



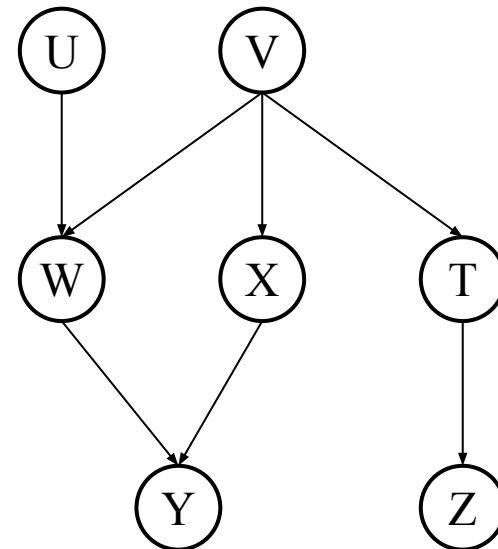
Quiz - 5

$U \perp V | X ?$



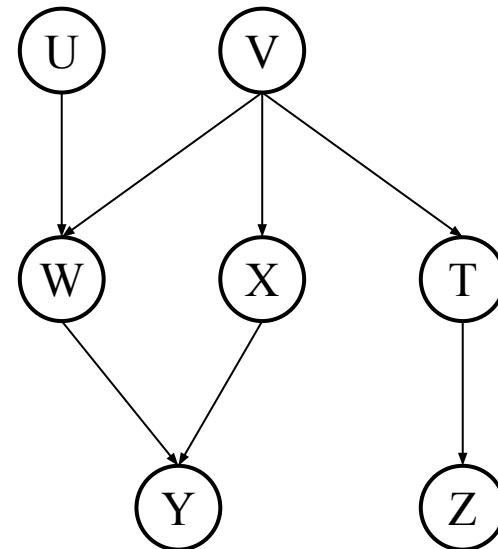
Quiz - 6

$U \perp V | Y ?$



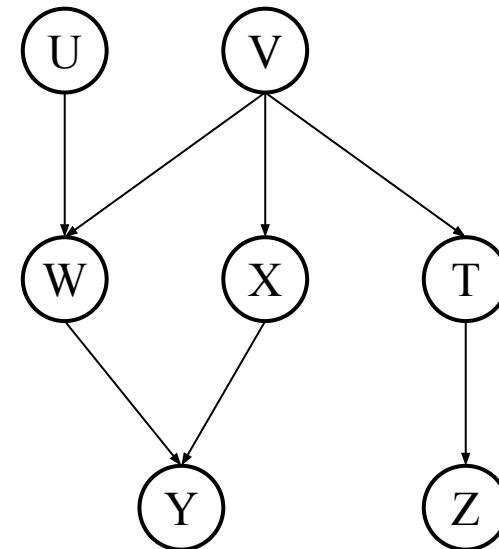
Quiz - 7

$U \perp V | Z ?$



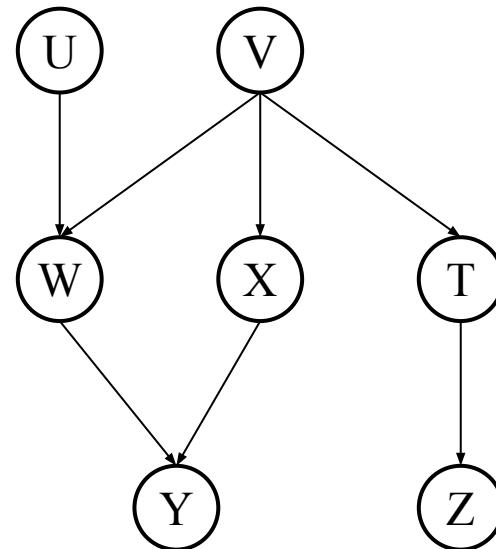
Quiz - 8

$W \perp X ?$



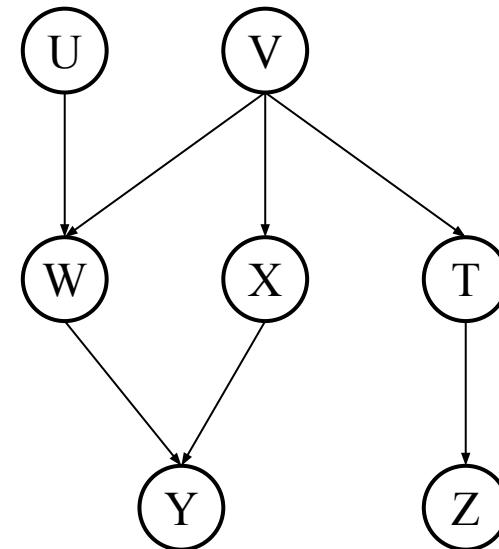
Quiz - 9

$X \perp\!\!\!\perp T \mid V ?$



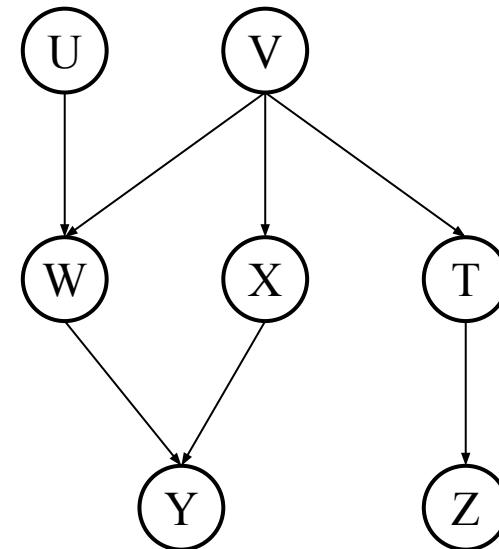
Quiz - 10

$X \perp W | U ?$



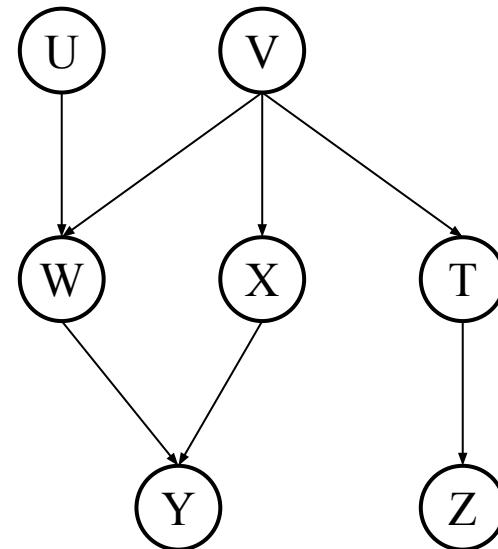
Quiz - 11

$Y \perp Z ?$



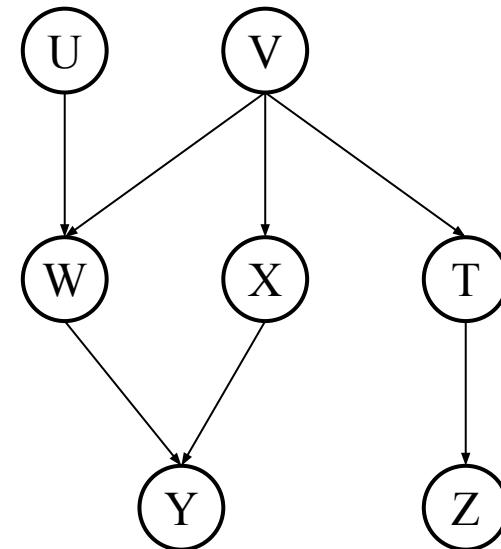
Quiz - 12

$Y \perp Z | T ?$



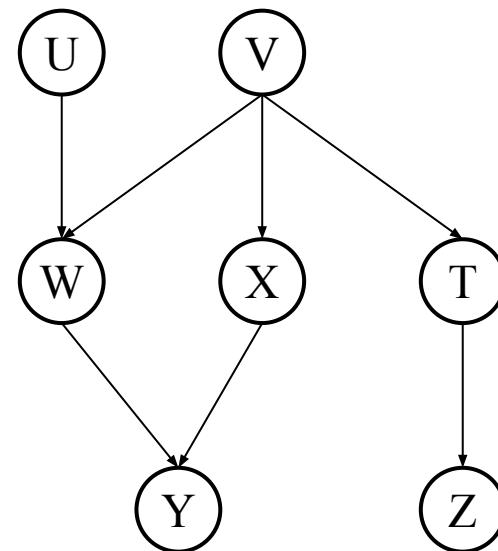
Quiz - 13

$Y \perp Z | X ?$



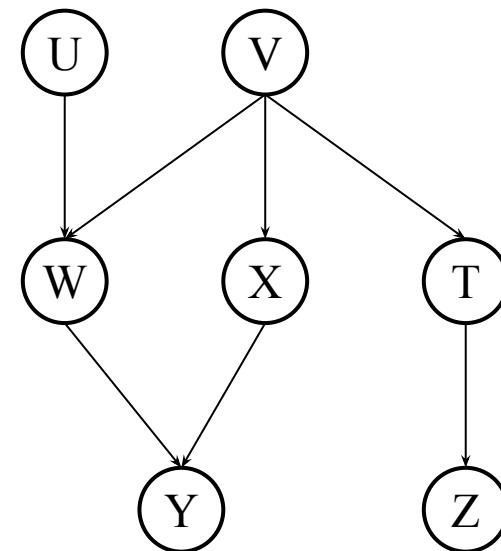
Quiz - 14

$Y \perp\!\!\!\perp Z | V ?$



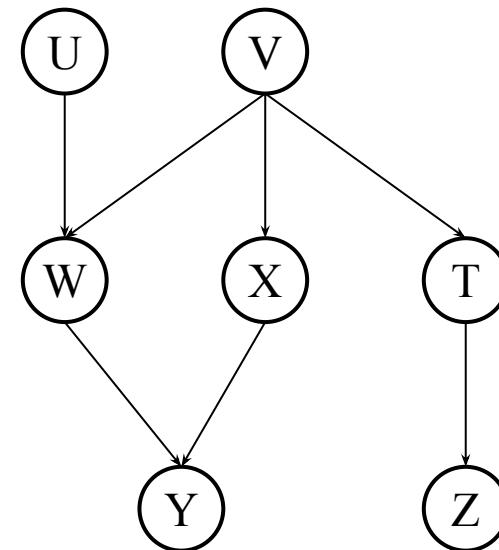
Quiz - 15

$W \perp Z | V ?$



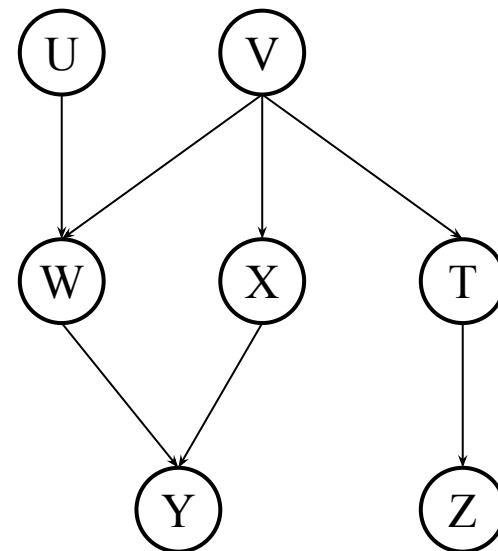
Quiz - 16

$U \perp Z ?$



Quiz - 17

$U \perp Z | Y ?$

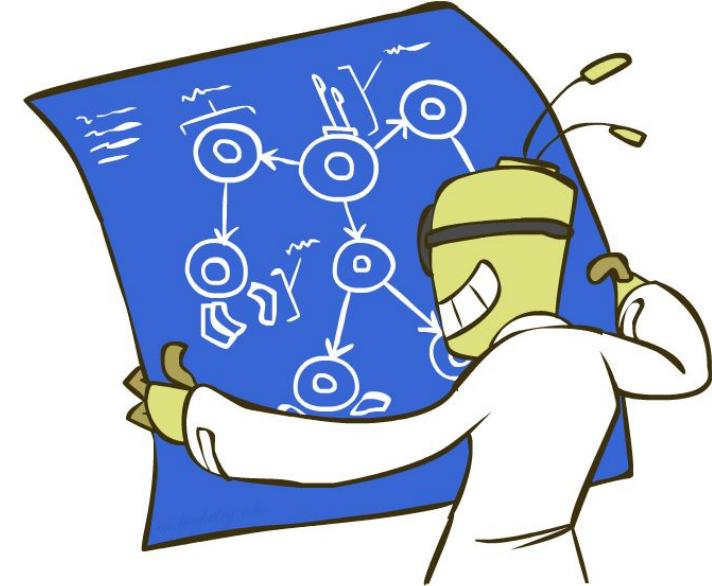


Structure Implications

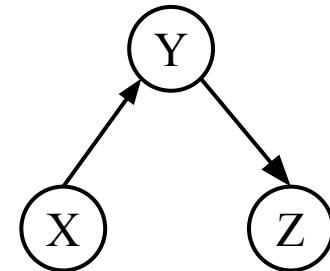
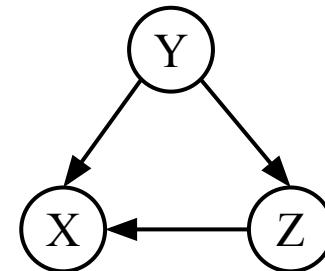
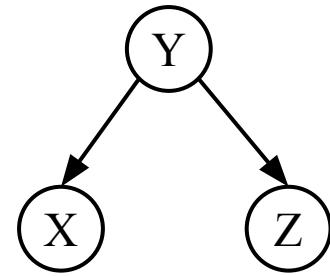
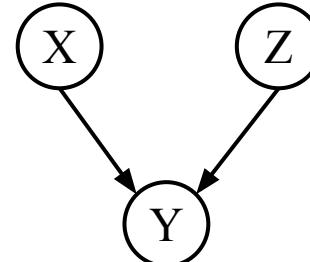
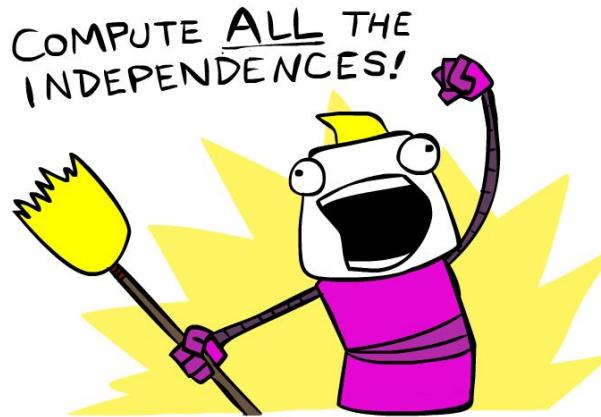
- Given a Bayes net structure, can run d-separation algorithm to build a complete list of conditional independencies that are necessarily true of the form

$$X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \dots, X_{k_n}\}$$

- This list determines the set of probability distributions that can be represented



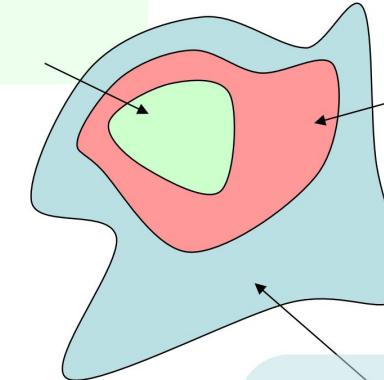
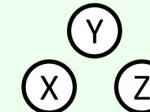
Quiz: Computing All Independences



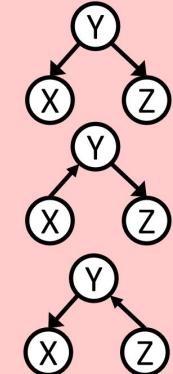
Topology Limits Distributions

- Given some graph topology G , only certain joint distributions can be encoded
- The graph structure guarantees certain (conditional) independencies
- (There might be more independence)
- Adding arcs increases the set of distributions, but has several costs
- Full conditioning can encode any distribution

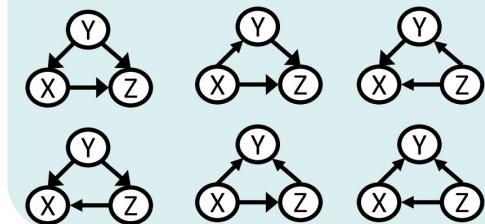
$$\{X \perp\!\!\!\perp Y, X \perp\!\!\!\perp Z, Y \perp\!\!\!\perp Z, \\ X \perp\!\!\!\perp Z | Y, X \perp\!\!\!\perp Y | Z, Y \perp\!\!\!\perp Z | X\}$$



$$\{X \perp\!\!\!\perp Z | Y\}$$



$$\{\}$$

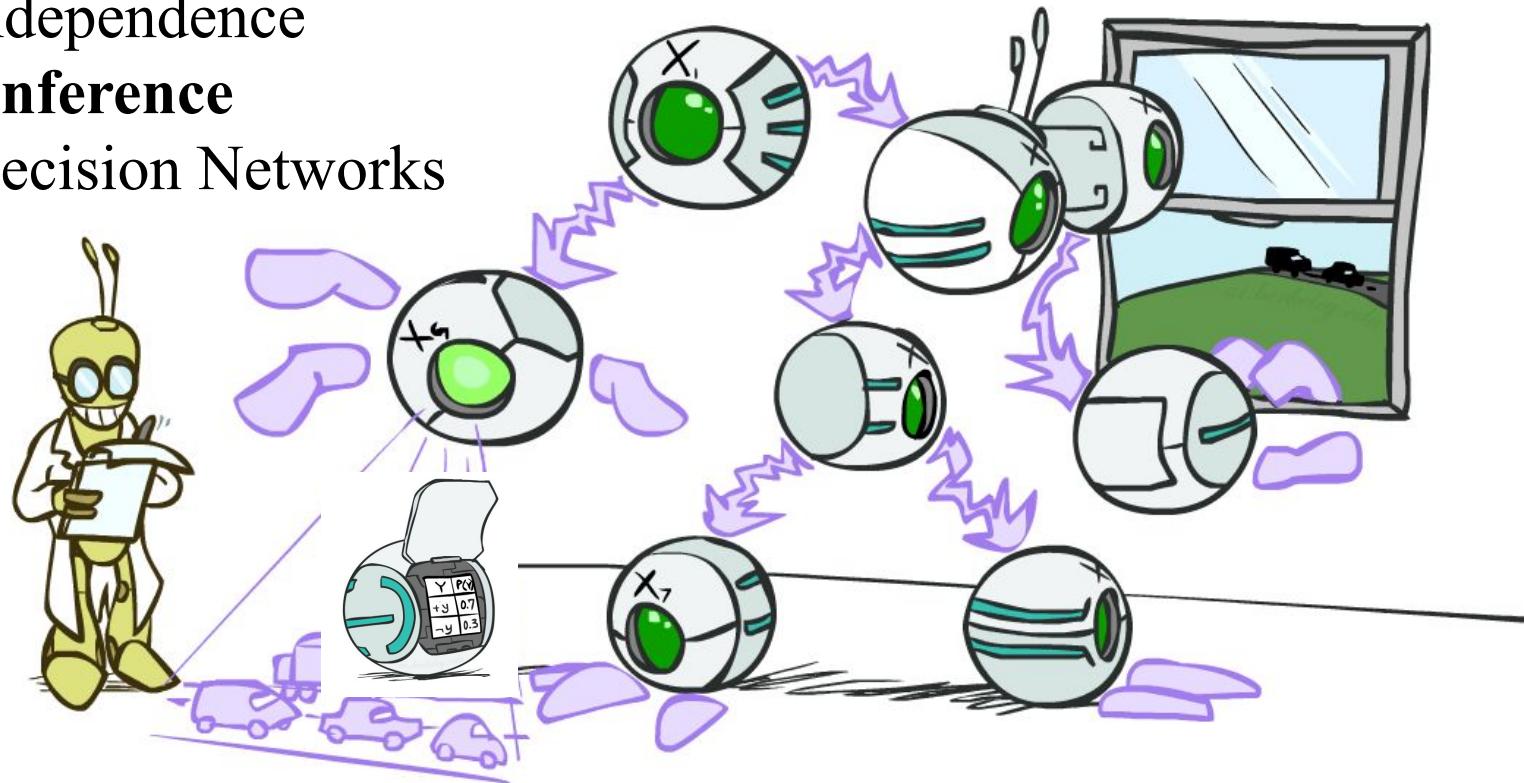


Bayes Nets Representation Summary

- Bayes nets compactly encode joint distributions
- Guaranteed independencies of distributions can be deduced from BN graph structure
- D-separation gives precise conditional independence guarantees from graph alone
- A Bayes' net's joint distribution may have further (conditional) independence that is not detectable until you inspect its specific distribution

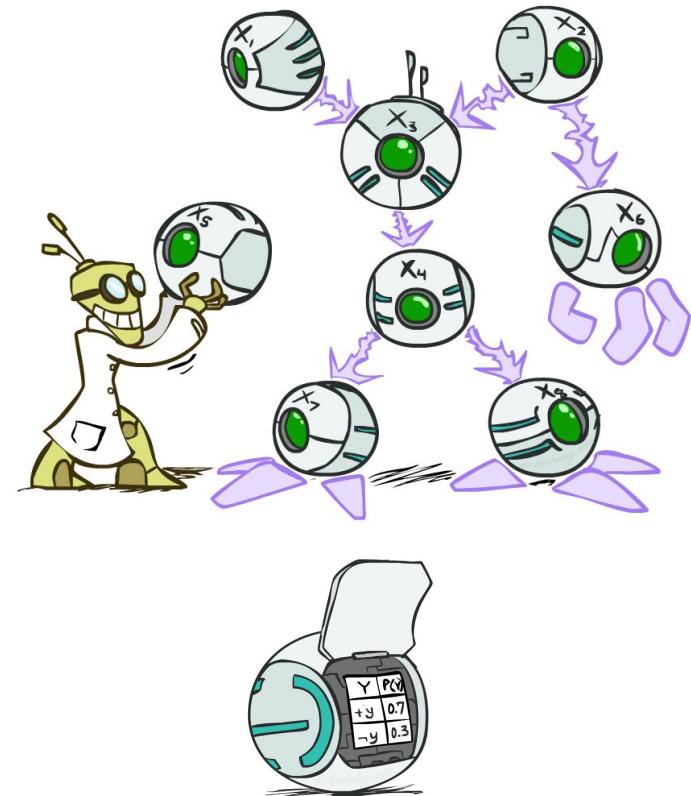
Outline

- Part A: Representation
- Part B: Independence
- **Part C: Inference**
- Part D: Decision Networks



Bayes' Net Representation

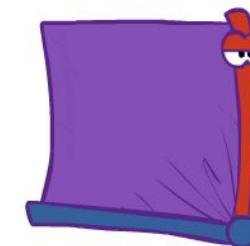
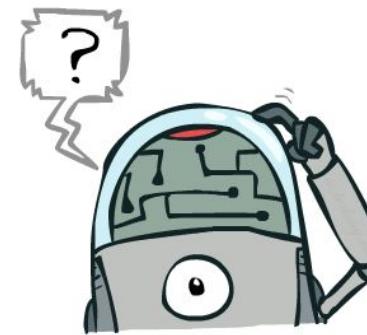
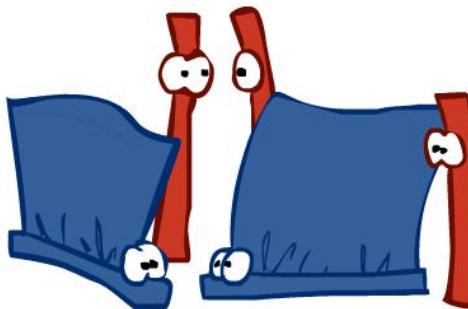
- A directed, acyclic graph, one node per random variable
- A conditional probability table (CPT) for each node
- Bayes' nets implicitly encode joint distributions



Inference

- Inference: calculating some useful quantity from a joint probability distribution
- Example: posterior probability

$$P(Q|E_1 = e_1, \dots, E_k = e_k)$$



Inference by Enumeration

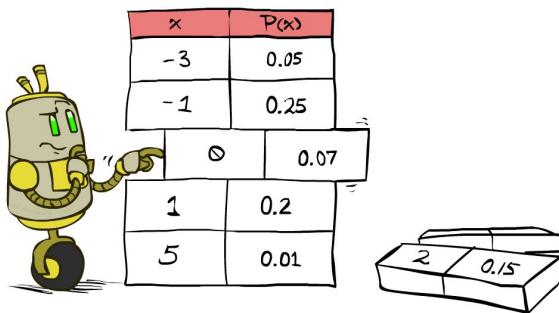
- General case

- Evidence variables: $E_1 \dots E_k = e_1 \dots e_k$
- Query variable: Q
- Hidden variables: $H_1 \dots H_r$

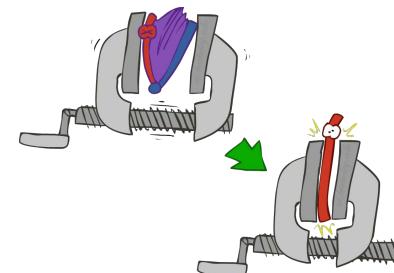
$X_1, X_2, \dots X_n$
All variables

We want: $P(Q|e_1 \dots e_k)$

Step 1: Select the entries
consistent with the evidence



Step 2: Sum out H to get joint of
Query and evidence



$$P(Q, e_1 \dots e_k) = \sum_{h_1 \dots h_r} P(Q, h_1 \dots h_r, e_1 \dots e_k)$$

$X_1, X_2, \dots X_n$

$$\times \frac{1}{Z}$$

$$Z = \sum_q P(Q, e_1 \dots e_k)$$

$$P(Q|e_1 \dots e_k) = \frac{1}{Z} P(Q, e_1 \dots e_k)$$

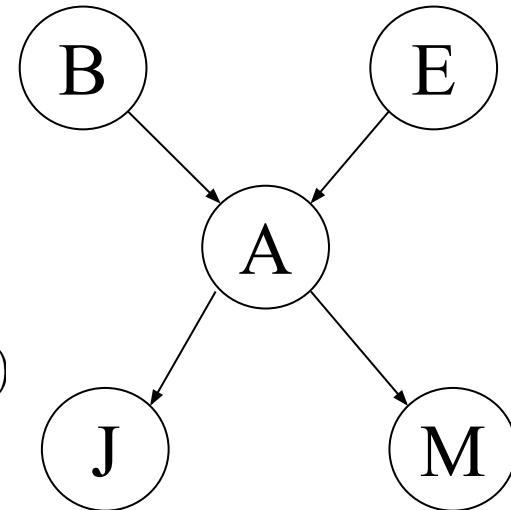
Example: Inference by Enumeration

$$P(B \mid +j, +m) \propto_B P(B, +j, +m)$$

$$= \sum_{e,a} P(B, e, a, +j, +m)$$

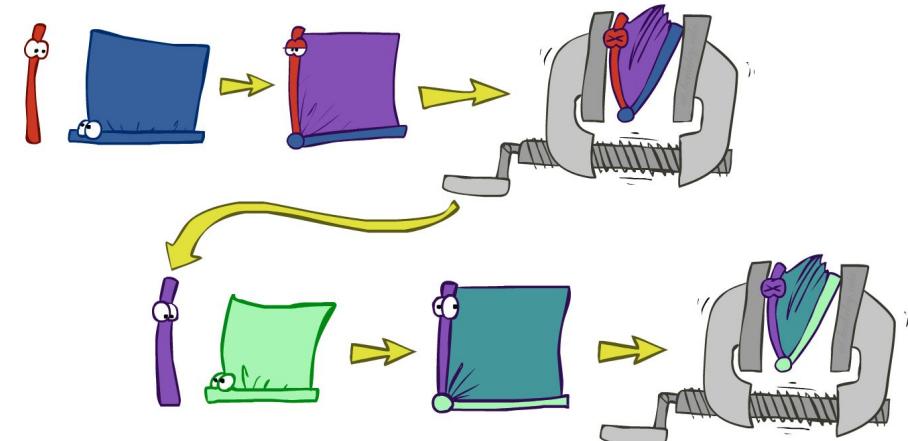
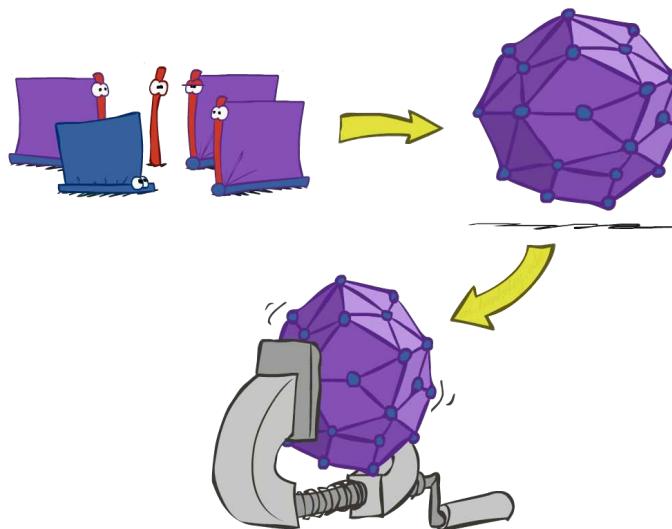
$$= \sum_{e,a} P(B)P(e)P(a|B, e)P(+j|a)P(+m|a)$$

$$= P(B)P(+e)P(+a|B, +e)P(+j|+a)P(+m|+a) + P(B)P(+e)P(-a|B, +e)P(+j|-a)P(+m|-a) \\ P(B)P(-e)P(+a|B, -e)P(+j|+a)P(+m|+a) + P(B)P(-e)P(-a|B, -e)P(+j|-a)P(+m|-a)$$



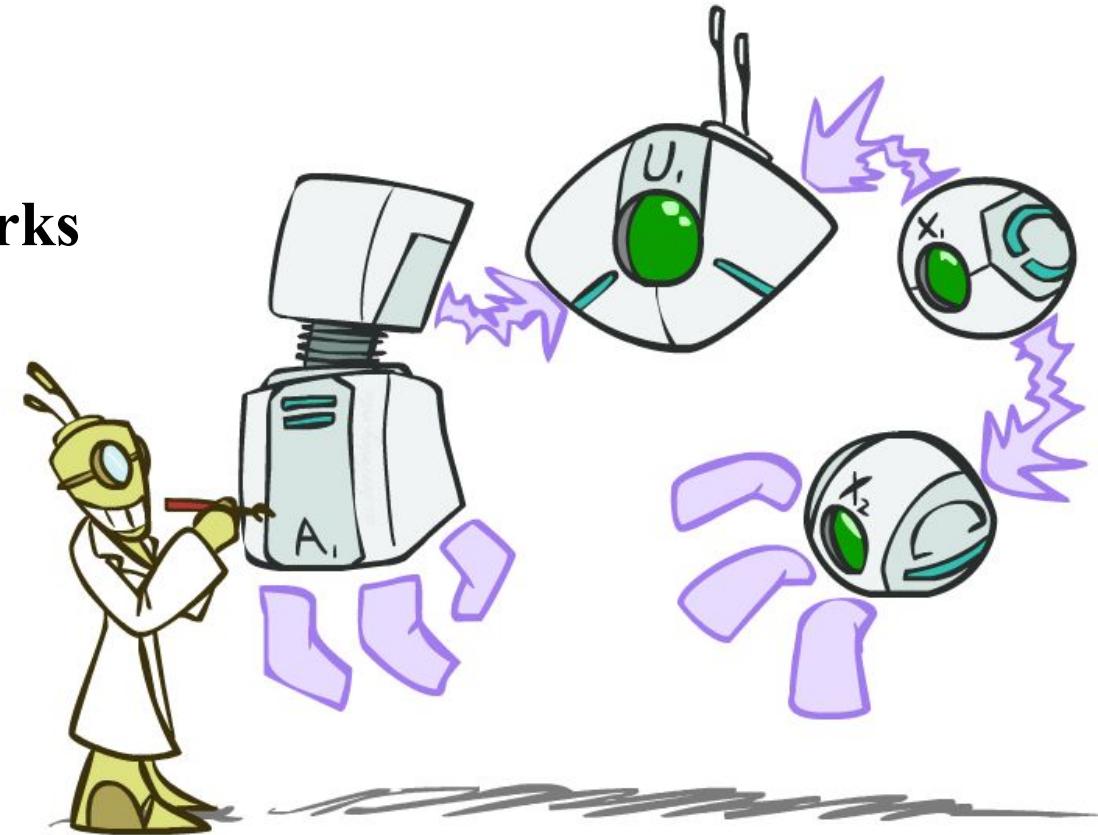
Inference by Enumeration vs. Variable Elimination

- Inference by enumeration is slow
 - You join up the whole joint distribution before you sum out the hidden variables
- Idea: interleave joining and marginalizing!
 - Called “Variable Elimination”
 - Usually much faster than inference by enumeration

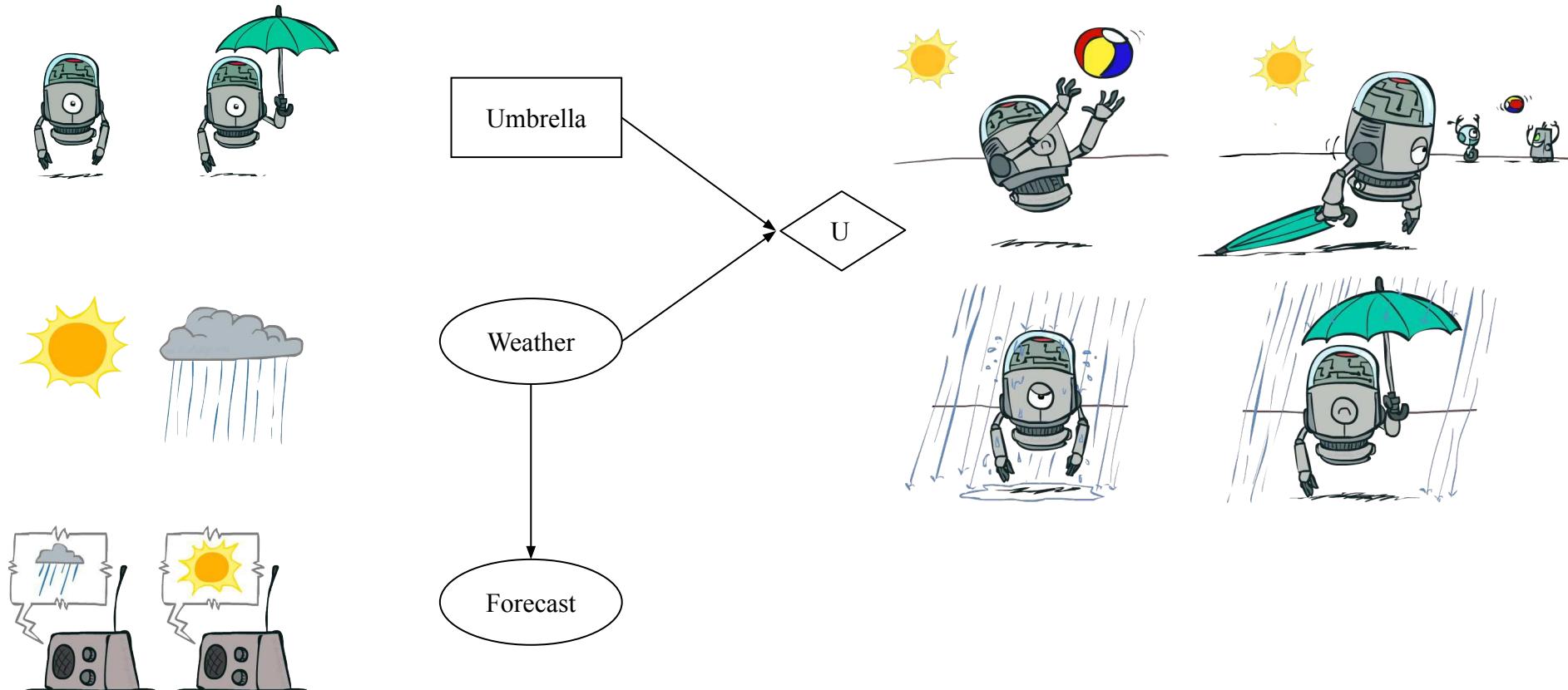


Outline

- Part A: Representation
- Part B: Independence
- Part C: Inference
- **Part D: Decision Networks**

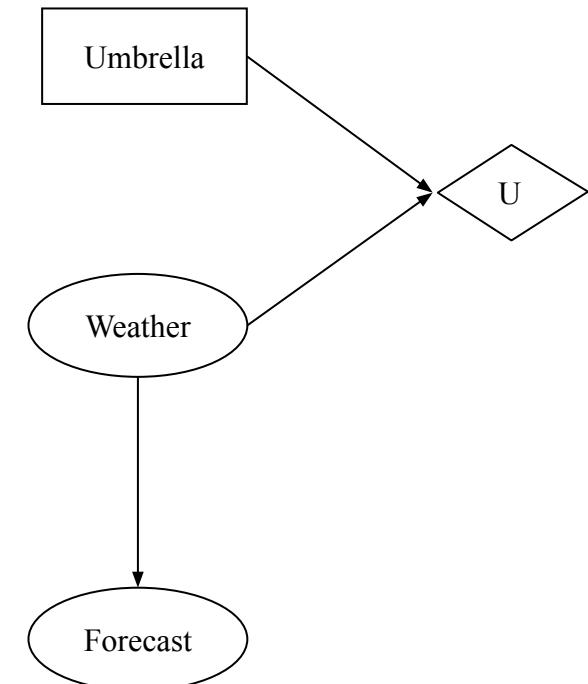


Decision Networks



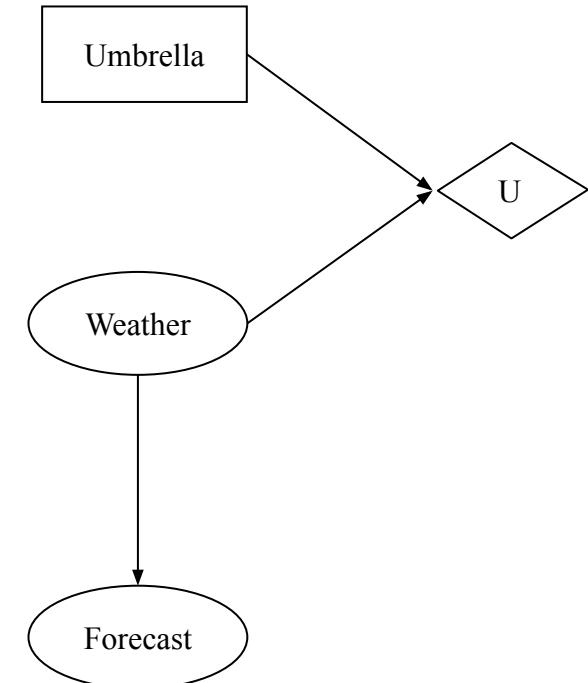
Decision Networks

- MEU: choose the action which maximizes the expected utility given the evidence
- Can directly operationalize this with decision networks
 - Bayes nets with nodes for utility and actions
 - Lets us calculate the expected utility for each action
- New node types
 - Chance nodes (just like BNs) 
 - Actions (rectangles, cannot have parents, act as observed evidence) 
 - Utility node (diamond, depends on action and chance nodes) 



Decision Networks

- Action selection
 - Instantiate all evidence
 - Set action node(s) each possible way
 - Calculate posterior for all parents of utility node, given the evidence
 - Calculate expected utility for each action
 - Choose maximizing action



Decision Networks

Umbrella = leave

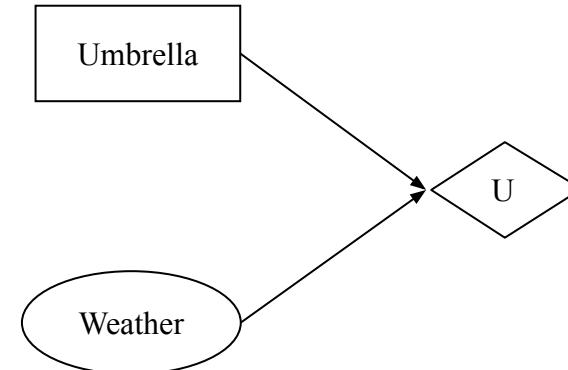
$$\begin{aligned} \text{EU}(\text{leave}) &= \sum_w P(w)U(\text{leave}, w) \\ &= 0.7 \cdot 100 + 0.3 \cdot 0 = 70 \end{aligned}$$

Umbrella = take

$$\begin{aligned} \text{EU}(\text{take}) &= \sum_w P(w)U(\text{take}, w) \\ &= 0.7 \cdot 20 + 0.3 \cdot 70 = 35 \end{aligned}$$

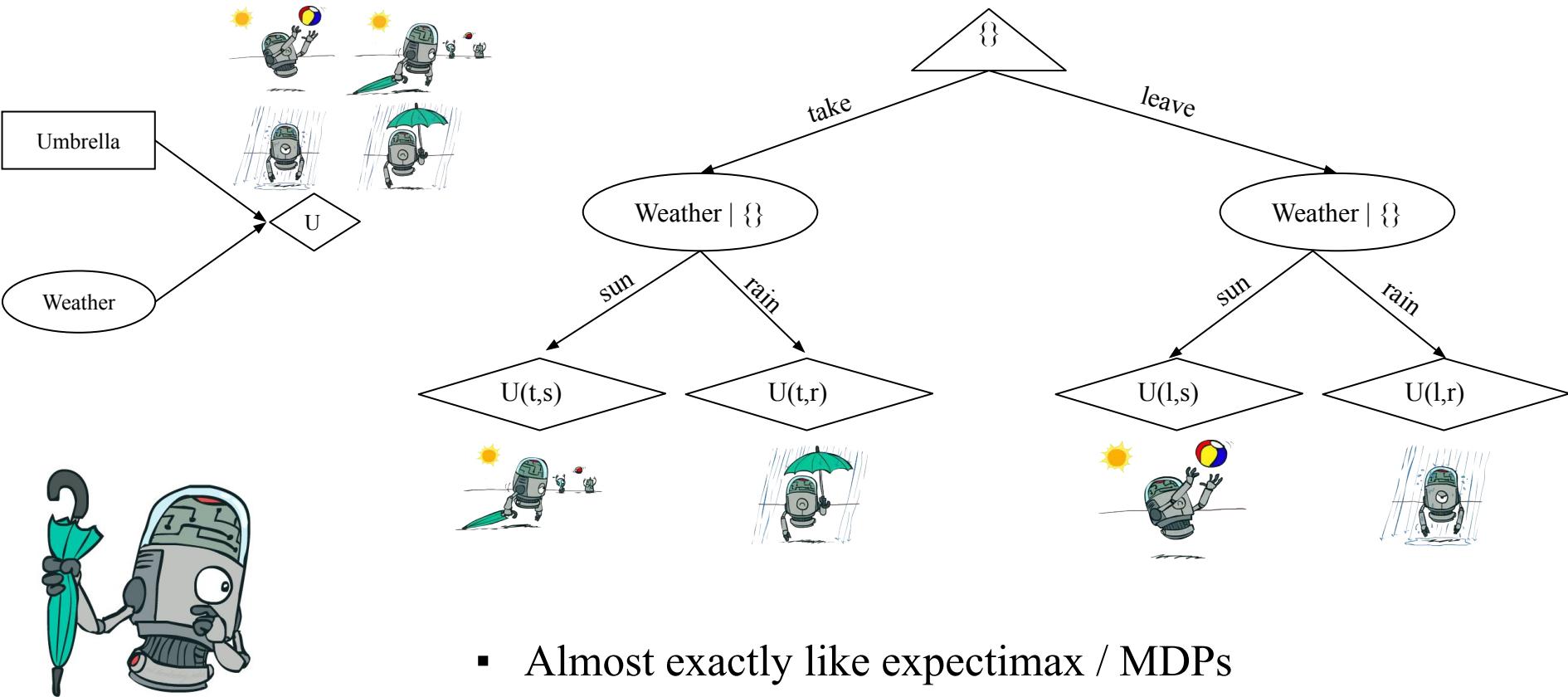
Optimal decision = leave

$$\text{MEU}(\emptyset) = \max_a \text{EU}(a) = 70$$



W	P(W)	A	W	U(A,W)
		leave	sun	100
sun	0.7	leave	rain	0
	0.3	take	sun	20
rain	0.3	take	rain	70

Decisions as Outcome Trees



Example: Decision Networks

Umbrella = leave

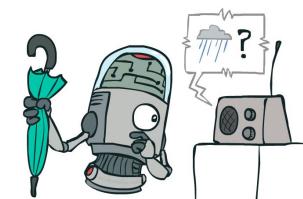
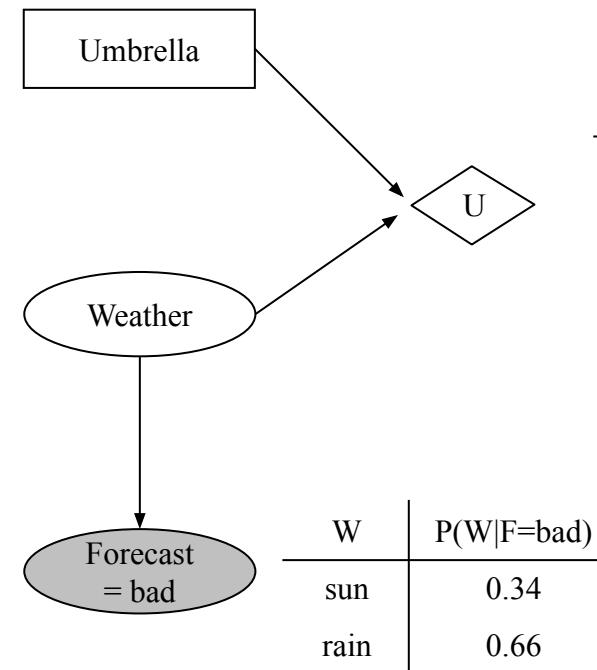
$$\begin{aligned} \text{EU}(\text{leave}|\text{bad}) &= \sum_w P(w|\text{bad})U(\text{leave}, w) \\ &= 0.34 \cdot 100 + 0.66 \cdot 0 = 34 \end{aligned}$$

Umbrella = take

$$\begin{aligned} \text{EU}(\text{take}|\text{bad}) &= \sum_w P(w|\text{bad})U(\text{take}, w) \\ &= 0.34 \cdot 20 + 0.66 \cdot 70 = 53 \end{aligned}$$

Optimal decision = take

$$\text{MEU}(F = \text{bad}) = \max_a \text{EU}(a|\text{bad}) = 53$$



Decisions as Outcome Trees

