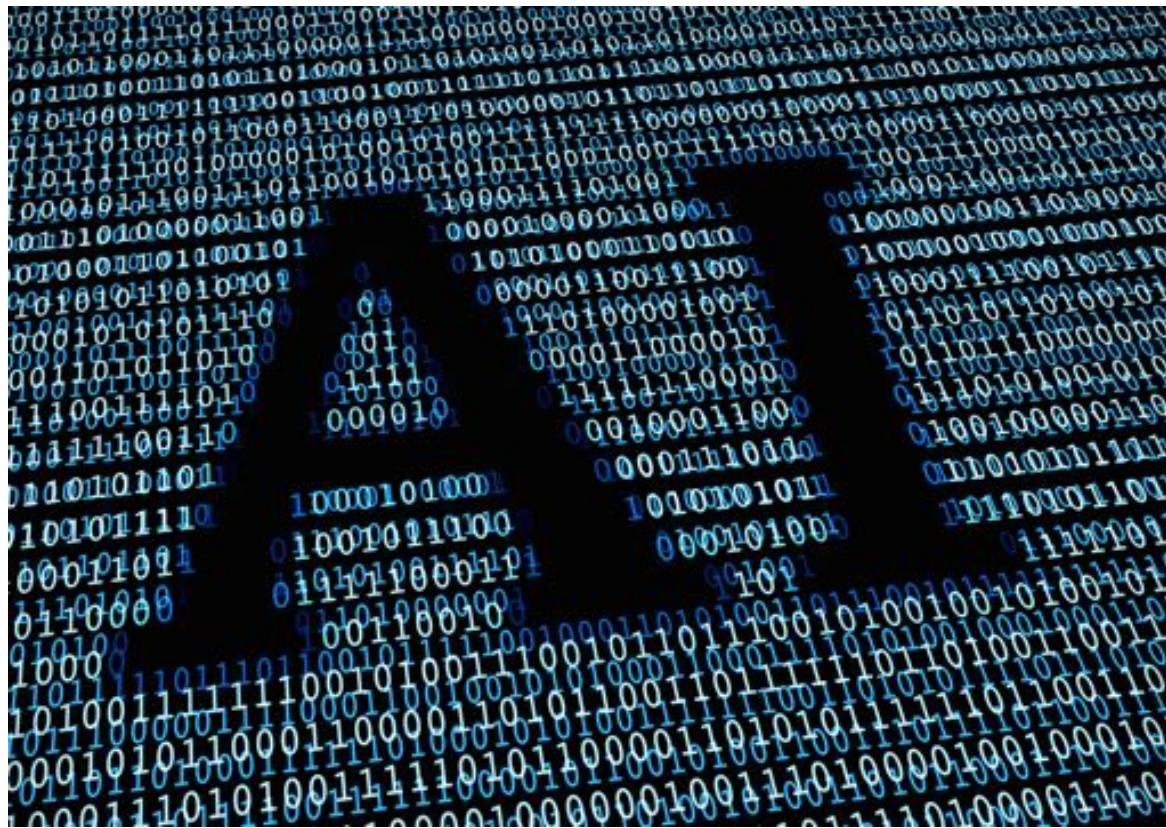


Artificial Intelligence

Introduction



AI in the movies



Definition of AI

“Intelligence: The ability to learn and solve problems”

Webster's Dictionary.

Definition of AI

“Intelligence: The ability to learn and solve problems”

Webster's Dictionary.

“Artificial intelligence (AI) is the intelligence exhibited by machines or software’

Wikipedia.

Definition of AI

“Intelligence: The ability to learn and solve problems”

Webster’s Dictionary.

“Artificial intelligence (AI) is the intelligence exhibited by machines or software”

Wikipedia.

“The science and engineering of making intelligent machines”

McCarthy.

Definition of AI

“Intelligence: The ability to learn and solve problems”

Webster’s Dictionary.

“Artificial intelligence (AI) is the intelligence exhibited by machines or software’

Wikipedia.

“The science and engineering of making intelligent machines”

McCarthy.

“The study and design of intelligent agents, where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success.”

Russel and Norvig AI book.

Why AI?

“Just as the Industrial Revolution freed up a lot of humanity from physical drudgery, I think AI has the potential to free up humanity from a lot of the mental drudgery.”

Andrew Ng.

What is AI?

Four schools of thoughts (Russel & Norvig)

Thinking humanly	Thinking rationally
"The exciting new effort to make computers think... <i>machines with minds</i> , in the full and literal sense." (Haugeland, 1985)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
Acting humanly	Acting rationally
"The study of how to make computers do things which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)

What is AI?

Thinking humanly: cognitive approach



Requires to determine how humans think!

1960's "cognitive revolution".

Requires scientific theories of internal activities of the brain

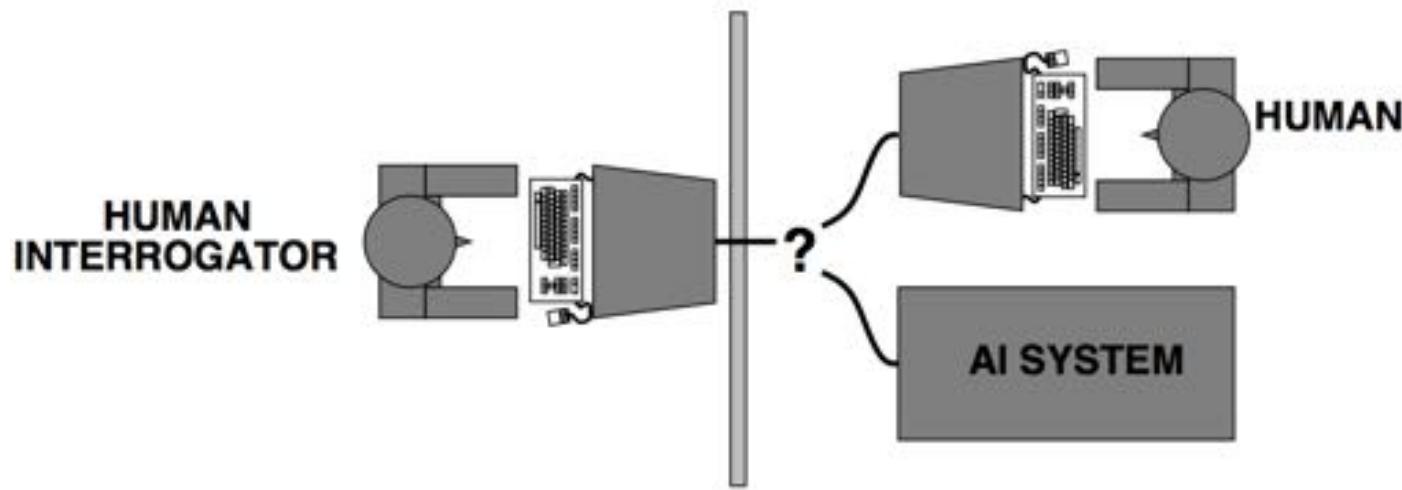
- What level of abstraction? "Knowledge" or "circuits" ?
- How to validate?

Today, Cognitive Science and Artificial Intelligence are distinct disciplines.

What is AI?

Acting humanly:

- **Turing test (Alan Turing 1950):** A computer passes the test of intelligence, if it can fool a human interrogator.



Credit: From Russel and Norvig slides.

- **Major components of AI:** knowledge, reasoning, language, understanding, learning.

What is AI?

Acting humanly:



What is AI?

Thinking rationally: Laws of thoughts.

- Codify “right thinking” with **logic**.
- Several Greek schools developed various forms of logic: *notation* and *rules of derivation* for thoughts.
- Problems:
 1. Not all knowledge can be expressed with logical notations.
 2. Computational blow up.

What is AI?

Acting rationally:

- The right thing: that which is expected to maximize goal achievement, given the available information.
- A **rational agent** is one that acts so as to achieve the best outcome, or when there is uncertainty, the best expected outcome.
- Aristotle (Nicomachean Ethics):
“Every art and every inquiry, and similarly every action and pursuit, is thought to aim at some good.”

What is AI?

Four schools of thoughts (Russel & Norvig)

Thinking humanly	Thinking rationally
"The exciting new effort to make computers think... <i>machines with minds</i> , in the full and literal sense." (Haugeland, 1985)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
Acting humanly	Acting rationally: Our approach
"The study of how to make computers do things which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)

Applications of AI



Applications of AI

Speech recognition

- Virtual assistants: Siri (Apple), Echo (Amazon), Google Now, Cortana (Microsoft).
- “They” helps get things done: send an email, make an appointment, find a restaurant, tell you the weather and more.
- Leverage deep neural networks to handle **speech recognition** and **natural language understanding**.



Applications of AI

Handwriting recognition (check, zipcode)



Applications of AI

Machine translation

- Historical motivation: translate Russian to English.
- First systems using **mechanical translation** (one-to-one correspondence) failed!
- “Out of sight, out of mind” ⇒ “Invisible, imbecile” .

Applications of AI

Machine translation

- Historical motivation: translate Russian to English.
- First systems using **mechanical translation** (one-to-one correspondence) failed!
- “Out of sight, out of mind” ⇒ “Invisible, imbecile” .

Oops!

Applications of AI

Machine translation

- MT has gone through ups and downs.
- Today, **Statistical Machine Translation** leverages the vast amounts of **available translated corpuses**.
- While there is room for improvement, machine translation has made significant progress.

Applications of AI

Machine translation

The screenshot shows the Google Translate homepage. At the top, there are language selection buttons for Arabic, English, French, Detect language, and a dropdown menu. Below these are two rows of language pairs. A dropdown menu is open over the 'Detect language' button, listing various languages. The language 'Azerbaijani' is highlighted in the dropdown. At the bottom left, there's a call-to-action button labeled 'JOIN THE TRANSLATE COMMUNITY'. At the bottom right, there are links for 'Google Translate for Business', 'Translator Toolkit', 'Website Translator', and 'Global Market Finder'.

	English	Arabic	French	detect language
Detect language	Corsican	Gujarati	Kazakh	Marathi
Afrikaans	Croatian	Haitian Creole	Khmer	Mongolian
Albanian	Czech	Hausa	Korean	Myanmar (Burmese)
Amharic	Danish	Hawaiian	Kurdish (Kurmanji)	Nepali
Arabic	Dutch	Hebrew	Kyrgyz	Norwegian
Armenian	English	Hindi	Lao	Pashto
Azerbaijani	Esperanto	Hmong	Latin	Persian
Basque	Estonian	Hungarian	Latvian	Polish
Belarusian	Filipino	Icelandic	Lithuanian	Portuguese
Bengali	Finnish	Igbo	Luxembourgish	Punjabi
Bosnian	French	Indonesian	Macedonian	Romanian
Bulgarian	Frisian	Irish	Malagasy	Russian
Catalan	Galician	Italian	Malay	Samoa
Cebuano	Georgian	Japanese	Malaysalam	Scots Gaelic
Chichewa	German	Javanese	Maltese	Serbian
Chinese	Greek	Kannada	Maori	Sesotho
				Shona
				Urdu
				Sindhi
				Uzbek
				Sinhala
				Vietnamese
				Slovak
				Welsh
				Slovenian
				Xhosa
				Somali
				Yiddish
				Spanish
				Youba
				Sundanese
				Zulu

100+ languages

Applications of AI

Machine translation

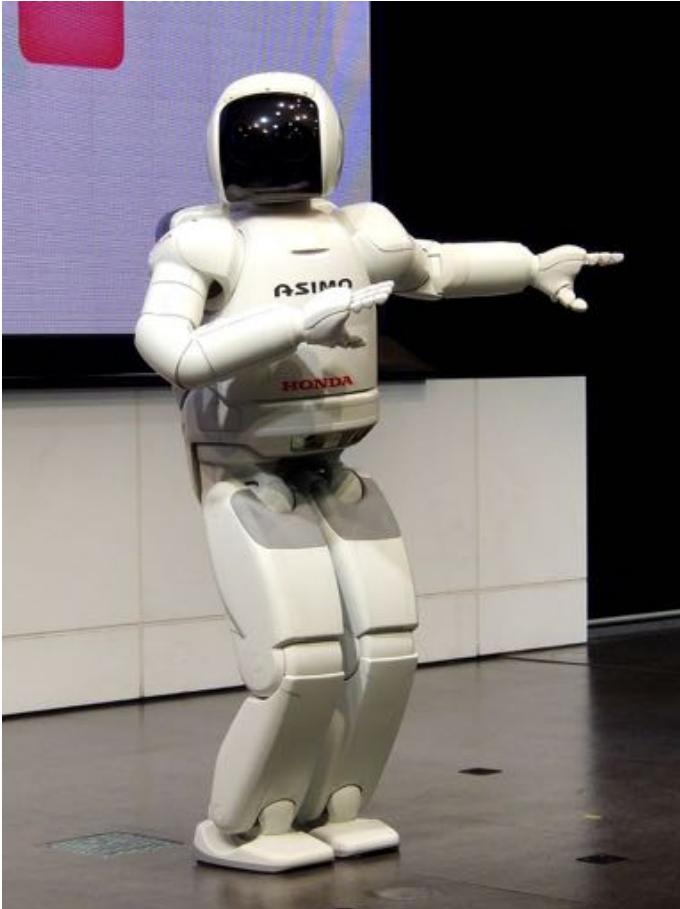


See also:

out of sight out of mind, out of mind, sight, out of, out of mind

Applications of AI

Robotics: Awesome robots today! NAO, ASIMO, and more!



Credit: By Momotarou2012, via Wikimedia Commons.

Applications of AI

Recommendation systems (collaborative filtering)

amazon

Your Amazon.com Today's Deals Gift Cards Sell Help

Shop by Department - Search All - kids dance wii u

Video Games Xbox One Xbox 360 PS4 PS3 Wii U 3DS PS Vita Digital Games Kindle Fire Games Deals Best Sellers Pre-orders Trade-in

Go

Just Dance Kids 2014 - Nintendo Wii U

By Ubisoft

Rated E10+ (6)

Read reviews (34 customer reviews)

Unit Price \$20.99

Price \$19.99 & FREE Shipping on orders over \$35. Details

You Save \$10.00 (30%)

In Stock.

Show from and sold by Amazon.com. Gift-wrap available.

Want it Thursday, Jan. 23? Order within 23 hrs 38 mins and choose One-Day Shipping at checkout. Details

Platforms: Nintendo Wii U

Nintendo Wii Xbox 360 Nintendo Wii U

- 30 Brand-New dances led by real kids
- Dance Director Mode: Use the Wii U gamepad to make your entire family dance to silly dance moves during any
- Kids can create custom playlists for endless fun
- Play with up to 5 Players

\$2.99 from \$15.99 Used from \$11.75

Customers Who Bought This Item Also Bought

SNG Party with Wii U Microphone

Nintendo

★★★★★ (25)

Nintendo Wii U \$15.99 ✓Prime

Wii U Microphone

Nintendo

★★★★★ (6)

Nintendo Wii U \$8.99 ✓Prime

Barbie Dreamhouse Party - Nintendo Wii U

Majesco Sales Inc.

★★★★★ (5)

Nintendo Wii U \$39.95 ✓Prime

Wii Party U

Nintendo

★★★★★ (40)

Nintendo Wii U \$39.99 ✓Prime

Just Dance 2014 - Nintendo Wii U

Ubisoft

★★★★★ (50)

Nintendo Wii U \$35.21 ✓Prime

Just Dance 4 - Nintendo Wii U

Ubisoft

★★★★★ (70)

Nintendo Wii U \$17.89 ✓Prime

ESPN Sports Connection - Nintendo Wii U

Ubisoft

★★★★★ (24)

Nintendo Wii U \$19.23 ✓Prime

Page 1 of 12

Applications of AI

Search engines

The screenshot shows a Google search results page for the query "Machine learning". The search bar at the top contains "Machine learning". Below the search bar, the "Web" tab is selected, followed by "Images", "Maps", "Shopping", "Patents", "More", and "Search tools". A search icon is located to the right of the search bar. The results section starts with a snippet about the number of results: "About 148,000,000 results (0.27 seconds)". The first result is a link to the Wikipedia page on Machine learning, with the title "Machine learning - Wikipedia, the free encyclopedia" and the URL "en.wikipedia.org/wiki/Machine_learning". The snippet below the link describes machine learning as a branch of artificial intelligence concerned with the construction and study of systems that can learn from data. The second result is a link to a Coursera course titled "Machine Learning", with the URL "https://www.coursera.org/course/ml". The snippet describes machine learning as the science of getting computers to act without being explicitly programmed. The third result is a link to the Springer journal "Machine Learning", with the URL "www.springer.com". The snippet describes it as an international forum for research on computational approaches to learning. The fourth result is a link to "Artificial Intelligence and Machine Learning - Research at Google", with the URL "research.google.com/pubs/ArtificialIntelligenceandMachineLearning.html". The snippet mentions Frame-Semantic Parsing, Dipanjan Das, Desai Chen, André F. T. Martins, and Training Highly Multi-class Linear Classifiers. The fifth result is a link to "ICML Beijing", with the URL "icml.cc".

Machine learning - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Machine_learning

Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. For example, a machine learning ...

List of machine learning ... - Supervised learning - Computational learning theory

Machine Learning | Coursera
https://www.coursera.org/course/ml

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self- driving ...

Machine Learning - Springer
www.springer.com > Home > Computer Science > Artificial Intelligence

Machine Learning is an international forum for research on computational approaches to learning. The journal publishes articles reporting substantive results on ...

Artificial Intelligence and Machine Learning - Research at Google
research.google.com/pubs/ArtificialIntelligenceandMachineLearning.html

by FS Parsing - 2013
340+ items - What We Do. Much of our work on language, speech, ...
Frame-Semantic Parsing, Dipanjan Das, Desai Chen, André F. T. Martins ...
Training Highly Multi-class Linear Classifiers, Maya R. Gupta, Samy Bengio ...

ICML Beijing
icml.cc

Applications of AI

Email

The screenshot shows a Gmail inbox with 1,886 messages. A red oval highlights the 'Spam (15)' link in the sidebar. Another red oval highlights the top message, which is an ad from Lumosity.com. A callout box provides information about the ad being based on emails from the user's mailbox and their Google account, with a link to 'Ads Settings'.

From	Subject	Time
Lumosity.com	Challenge Your Brain - Challenge your brain with Lumosity, the personal trainer designed by neuroscientists.	1:50 of 2,006
Groupon Getaways	NYC Dominican Republic Niagara Falls Turkey O	
WebMD	Goat Cheese Grits With Fresh Corn - Daily Bite Tip	
1-800-FLOWERS.COM	Free Shipping Today & Tomorrow! - Send a smile, e	
The Body Shop	Buy 3 Get 3 or Buy 2 Get 2 FREE All Bath & Body - Mega Moisture, Mini Poo... 9:35 am	
WebMD	Have you logged your food and fitness today? - Food & Fitness Planner Dear fi 9:26 am	
Century 21 Dept Store	Say Spaaaaal! 50% Off Setal Pampering Package + More V-Day Gifts - This Just 7:06 am	
Banana Republic	35% off starts right now! - 35% off ends 1/22. Online only. Can't see the images if 5:04 am	

Applications of AI

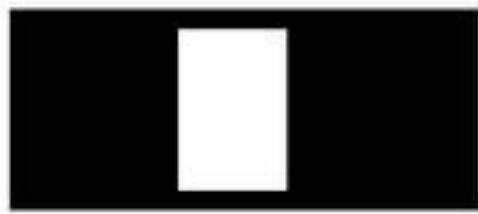
Face detection



Viola-Jones method.

Applications of AI

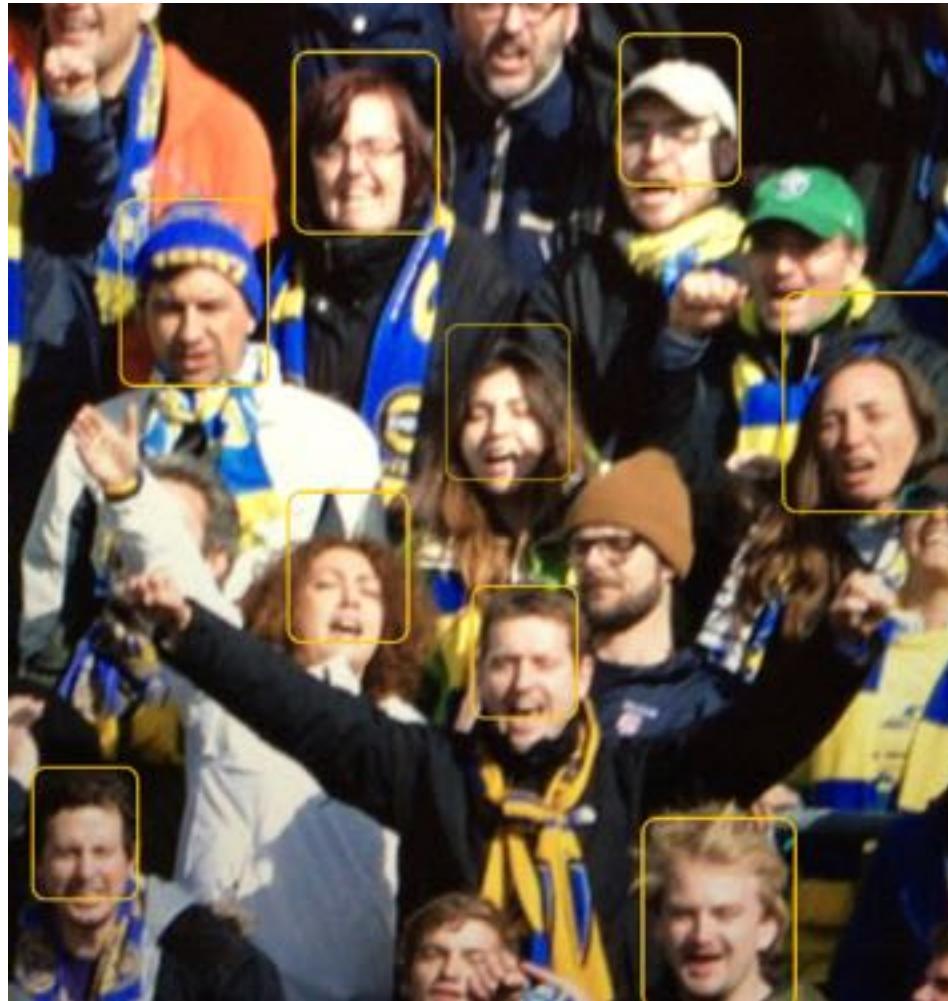
Face detection



Viola-Jones method.

Applications of AI

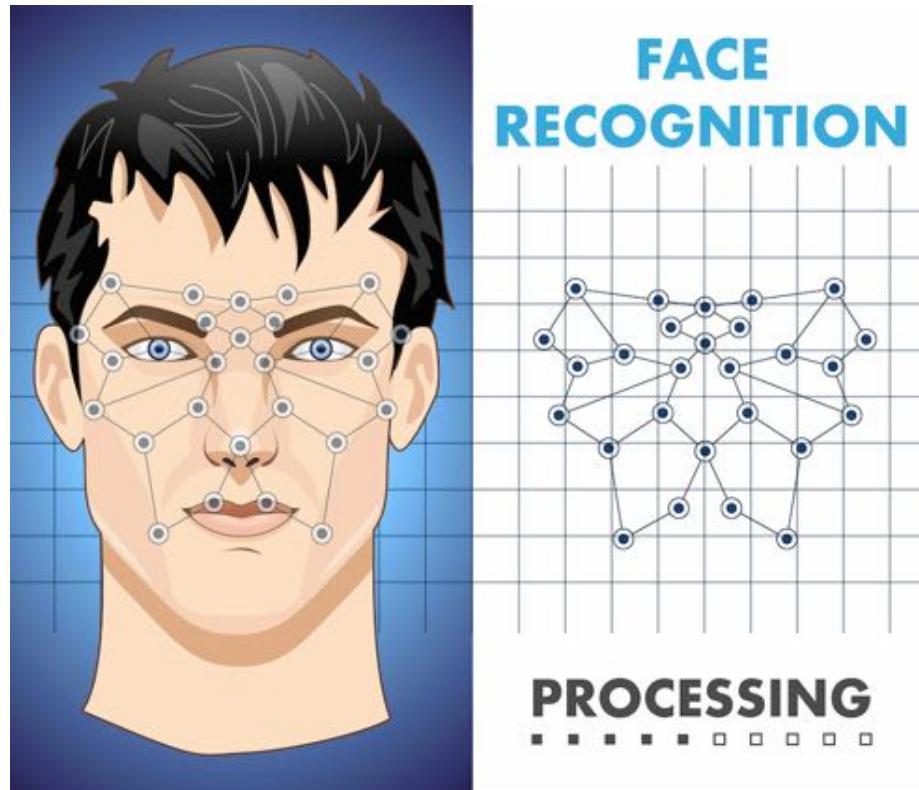
Face detection



Viola-Jones method.

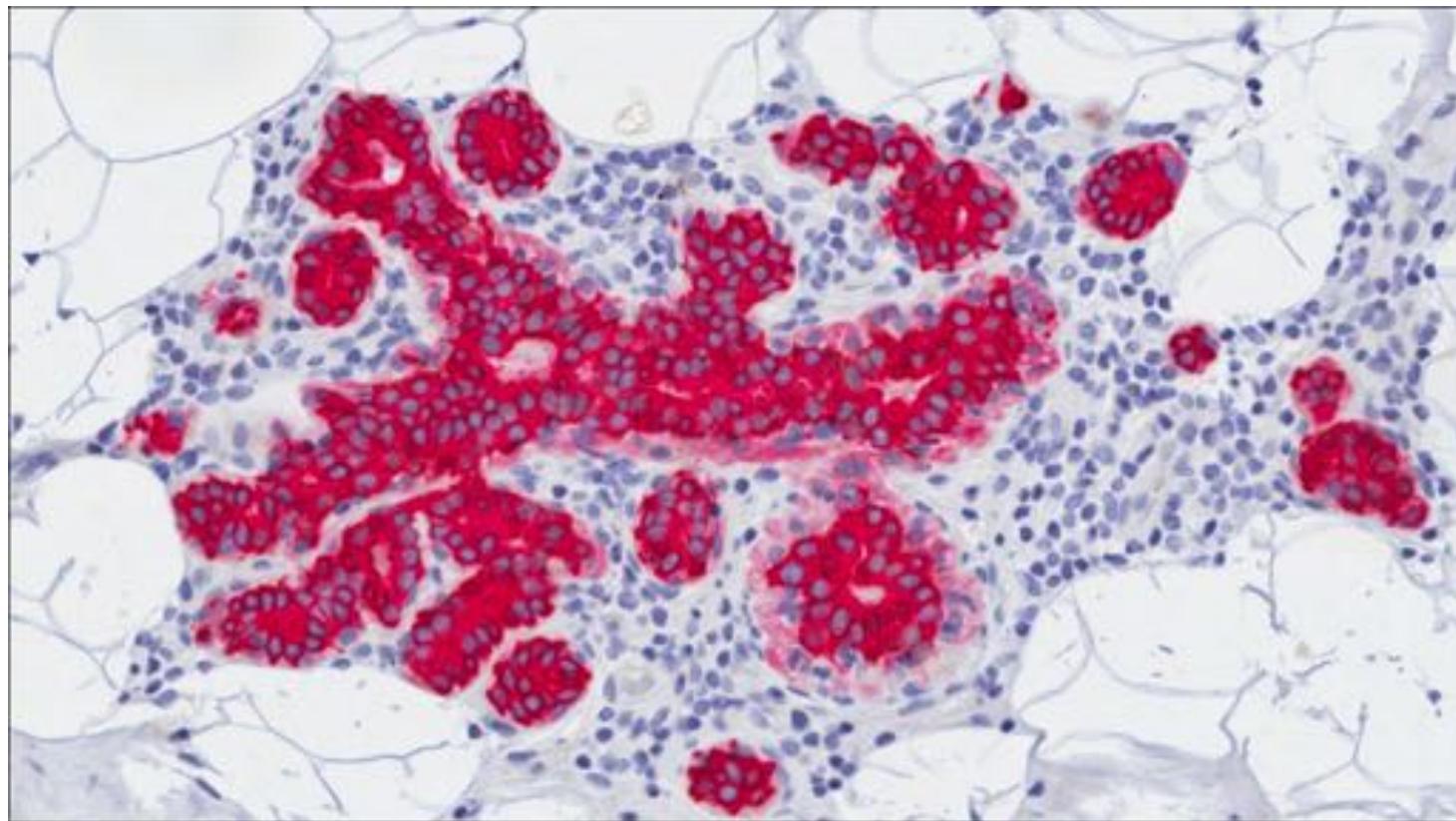
Applications of AI

Face recognition



Applications of AI

Detection of breast cancer in mammography images



Applications of AI

Chess (1997): Kasparov vs. IBM Deep Blue



(Left) Copyright 2007, S.M.S.I., Inc. - Owen Williams, The Kasparov Agency, via Wikimedia Commons (Right) By James the photographer, via Wikimedia Commons

Powerful search algorithms!

Applications of AI

Jeopardy! (2011): Humans vs. IBM Watson



By Rosemaryetoufee (Own work), via Wikimedia Commons

Natural Language Understanding and information extraction!

Applications of AI

Go (2016): Lee Sedol versus Google AlphaGo



(Left) By LG Electronics, via Wikimedia Commons (Right) By Google DeepMind, via
Wikimedia Commons

Deep Learning, reinforcement learning, and search algorithms!

Applications of AI

Autonomous driving



By User Spaceape on en.wikipedia, via Wikimedia Commons

- DARPA Grand Challenge
 - 2005: 132 miles
 - 2007: Urban challenge
 - 2009: Google self-driving car

State-of-the-art applications

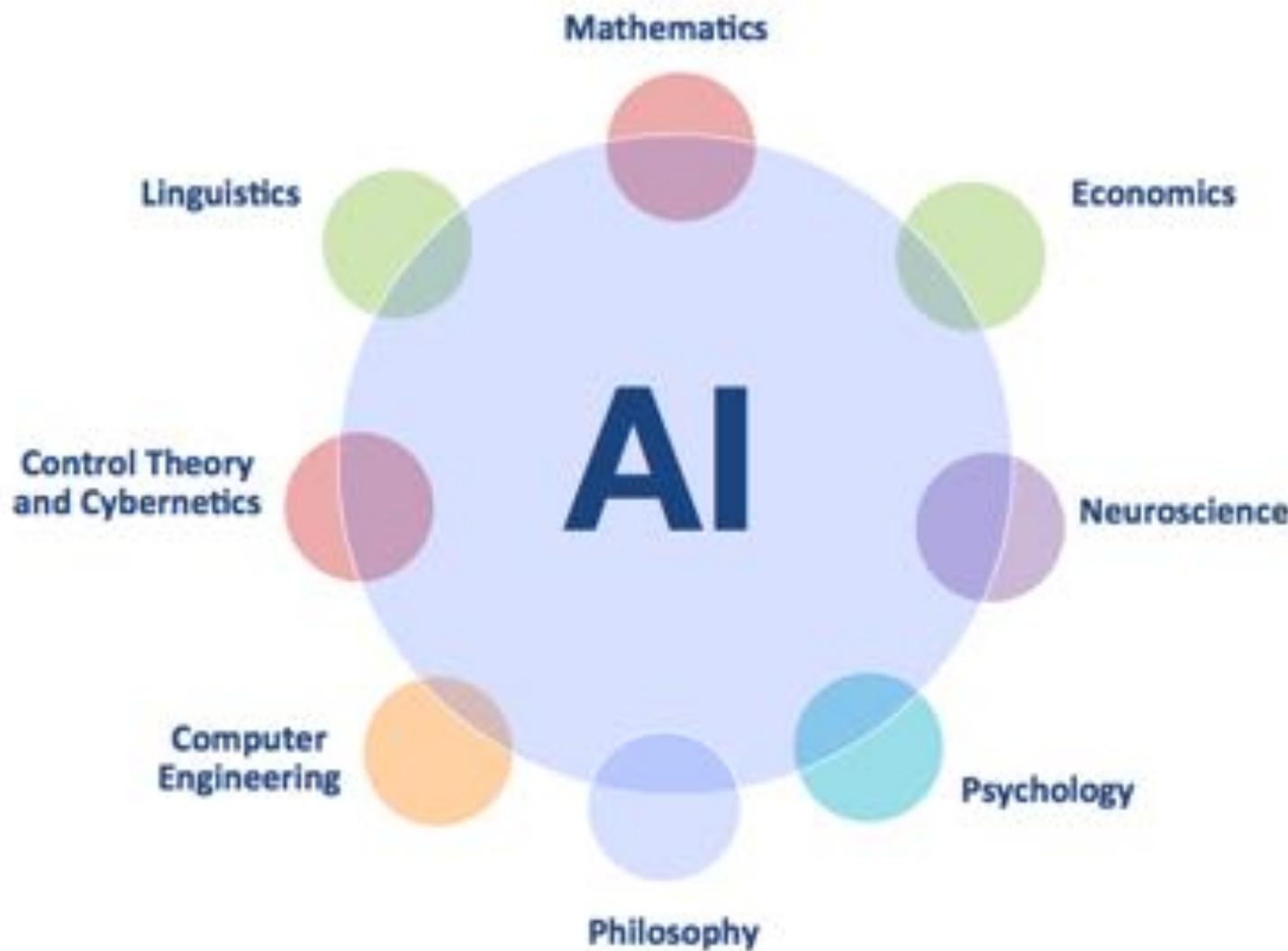
- Speech recognition
- Autonomous planning and scheduling
- Financial forecasting
- Game playing, video games
- Spam fighting
- Logistics planning
- Robotics (household, surgery, navigation)
- Machine translation
- Information extraction
- VLSI layout
- Automatic assembly
- Sentiment analysis
- Fraud detection
- Recommendation systems
- Web search engines
- Autonomous cars
- Energy optimization
- Question answering systems
- Social network analysis
- Medical diagnosis, imaging
- Route finding
- Traveling salesperson
- Protein design
- Document summarization
- Transportation/scheduling
- Computer animation

State-of-the-art applications

- Speech recognition
- Autonomous planning and scheduling
- Financial forecasting
- Game playing, video games
- Spam fighting
- Logistics planning
- Robotics (household, surgery, navigation)
- Machine translation
- Information extraction
- VLSI layout
- Automatic assembly
- Sentiment analysis
- Fraud detection
- Recommendation systems
- Web search engines
- Autonomous cars
- Energy optimization
- Question answering systems
- Social network analysis
- Medical diagnosis, imaging
- Route finding
- Traveling salesperson
- Protein design
- Document summarization
- Transportation/scheduling
- Computer animation

Many more!

Foundation of AI



Foundation of AI

- **Philosophy**

- Logic, methods of reasoning.
- Mind as physical system that operates as a set of rules.
- Foundations of learning, language, rationality.

- **Mathematics**

- Logic: Formal representation and proof.
- Computation, algorithms.
- Probability.

- **Economics**

- Formal theory of rational decisions.
- Combined decision theory and probability theory for decision making under uncertainty.
- Game theory.
- Markov decision processes.

Foundation of AI

- **Neuroscience**
 - Study of brain functioning.
 - How brains and computers are (dis)similar.
- **Psychology**
 - How do we think and act?
 - Cognitive psychology perceives the brain as an information processing machine.
 - Led to the development of the field *cognitive science*: how could computer models be used to study *language, memory, and thinking* from a psychological perspective.
- **Computer engineering**
 - Cares about how to build powerful machines to make AI possible.
 - E.g., Self-driving cars are possible today thanks to advances in computer engineering.

Foundation of AI

- **Control theory and cybernetics**
 - Design simple optimal agents receiving feedback from the environment.
 - Modern control theory design systems that maximize an objective function over time.
- **Linguistics**
 - How are language and thinking related.
 - Modern linguistics + AI = Computational linguistics (Natural language processing).

AI founders

- Aristotle
- Alan Turing
- John Mc Carthy
- Warren McCulloh
- Walter Pitts
- Claude Shannon
- Marvin Minsky
- Dean Edmonds
- Herbert Simon
- Allen Newell
- David Waltz
- Tom Mitchell
- Stuart J. Russell
- Peter Norvig
- etc.

AI Resources

- Major journals/conferences: JAIR, TPAMI, JMLR, IJCAI, AAAI, IAAI, CVPR, ECAI, ICML, NIPS, etc.
- Video lectures:

http://videolectures.net/Top/Computer_Science/Artificial_Intelligence/

History of AI

- **1940-1950:** Gestation of AI
 - McCulloch & Pitts: Boolean circuit to model of brain
 - Turing's Computing Machinery and Intelligence
<http://www.turingarchive.org/browse.php/B/9>
- **1950-1970:** Early enthusiasm, great expectations
 - Early AI programs, Samuel's checkers program
 - Birth of AI @ Dartmouth meeting 1956.
 - Check out the MIT video “The thinking Machine” on youtube
<https://www.youtube.com/watch?v=aygSMgK3BEM>
- **1970-1990:** Knowledge-based AI
 - Expert systems, AI becomes an industry
 - AI winter

History of AI

- **1990-present:** Scientific approaches
 - Neural Networks: le retour
 - The emergence of intelligent agents
 - AI becomes “scientific”, use of probability to model uncertainty
 - AI Spring!
 - The availability of very large datasets.
 - * Data will drive future discoveries and alleviate the complexity in AI.

Course logistics

- **Course level:** Master's – challenging!
- **Prerequisites:** You are required to have some knowledge of programming and an understanding of probability. Python is the programming language in this course.
- **Assignments:** There will be two kinds of assignments:
 - **Quizzes (conceptual):** Test your understanding of the lectures. **Please read the questions very carefully.**
 - **Projects (programming):** The course offers an excellent opportunity for students to dive into Python while solving AI problems and learning its applications.

Course logistics

- **Suggested readings:**
 - We recommend this book, which is the main reference in the field:
Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.
<http://aima.cs.berkeley.edu/>
 - Check out the list of readings, useful links we suggest for this course.

What you will learn

- **Introduction** to artificial intelligence, **history** of Artificial Intelligence.
- **Building intelligent agents** (search, games, logic, constraint satisfaction problems).
- **Machine Learning** algorithms.
- **Applications** of AI (Natural Language Processing, Robotics, and Vision).
- Solving real AI problems through **programming Python**.

Course roadmap

1. Rational intelligent agents
2. Search agents (uninformed search, informed search)
3. Adversarial search/games
4. Machine Learning (ML)
5. Constraint satisfaction problems (CSPs)
6. Logic (propositional logic, first order logic)
7. Markov Decision Processes (MDPs) and Reinforcement Learning (RL)
8. Application to Natural language Processing (NLP)
9. Application to vision and robotics

Rational intelligent agents

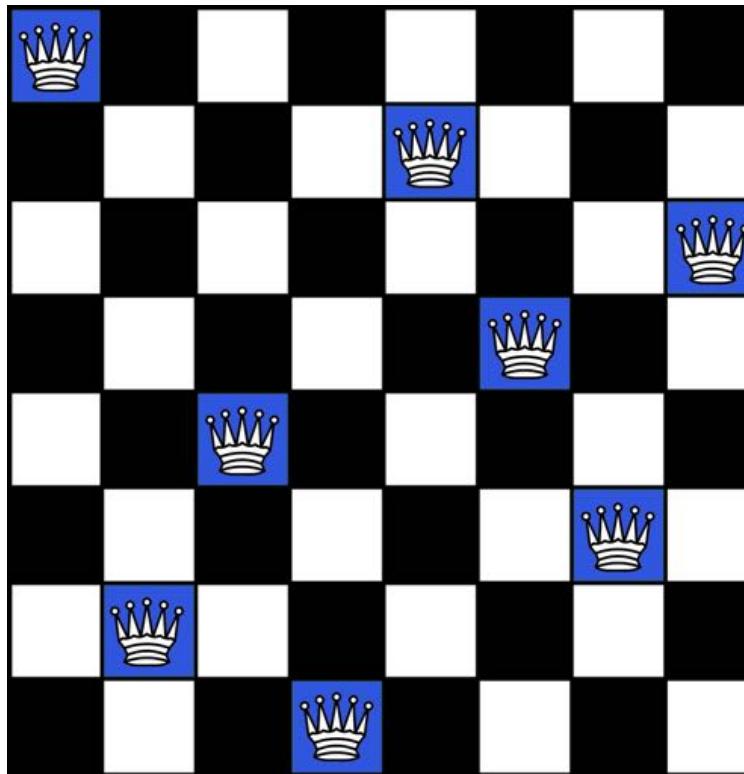
- This course is about designing **intelligent agents**.
- An agent perceives the environment and act upon that environment to achieve some task.
- An agent is function from percepts to actions.
- We care specifically about **rational agents**.
- Rationality is relative to how to act to maximize a **performance measure**.
- AI aims to design the best agents (programs) that achieve the best performance given the computational limitations.

Agent = Architecture + Program

Search agents

- Agents that work towards a **goal**.
- Agents consider the impact of **actions** on future **states**.
- Agent's job is to identify the action or series of actions that lead to the goal.
- Paths come with different costs and depths.
- Two kinds of search:
 - **Uninformed Search** (use no domain knowledge): BFS, DFS, UCS, etc.
 - **Informed Search** (use heuristic to reach the goal faster): Greedy search, A*, etc.

Search agents

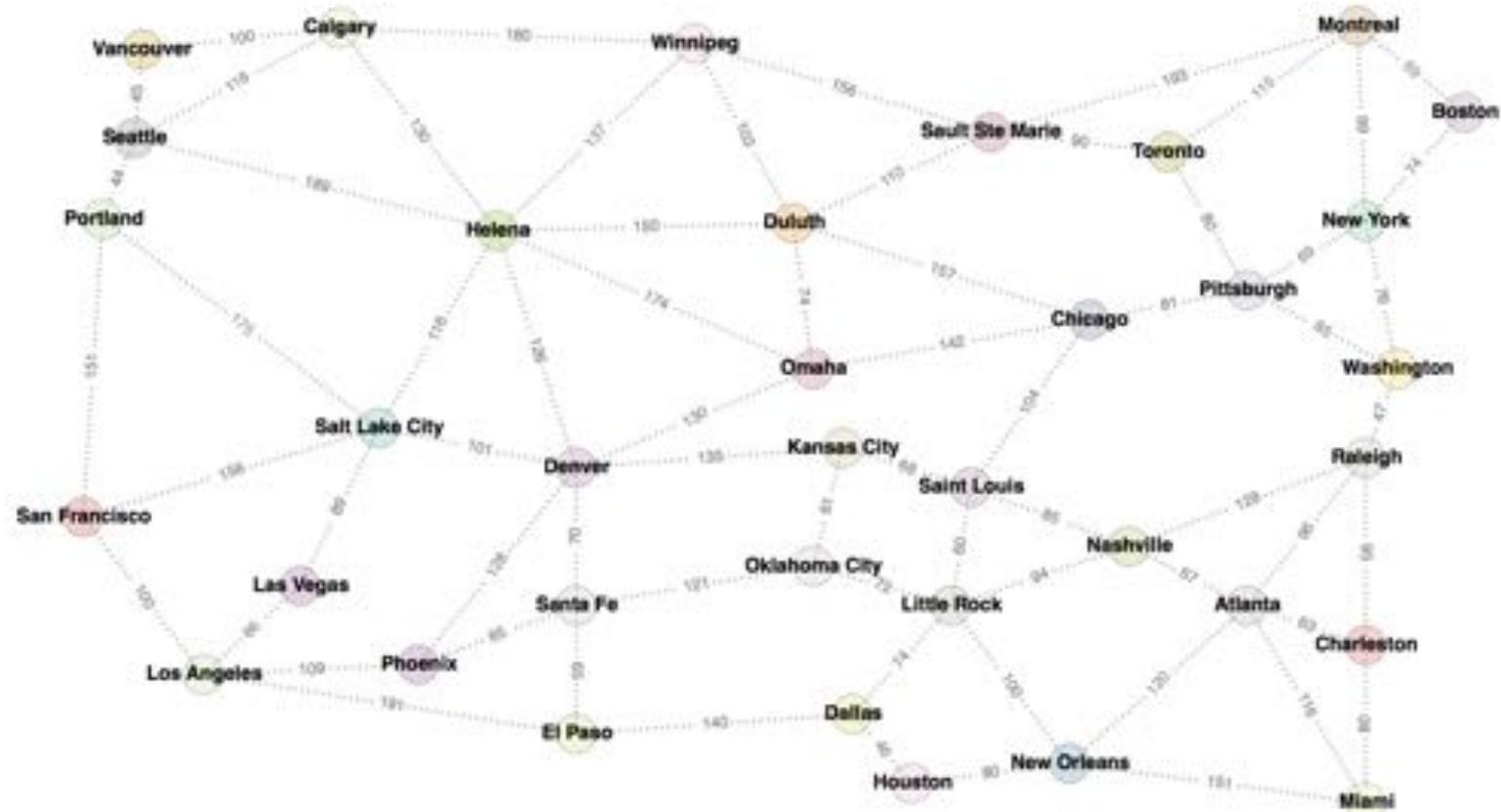


The 8-queen problem: on a chess board, place 8 queens so that no queen is attacking any other horizontally, vertically or diagonally.

Search agents

Start: Las Vegas

Goal: Calgary



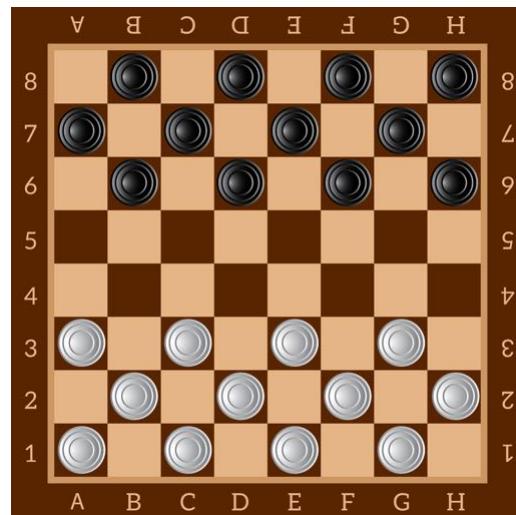
Explore + Execute

Adversarial Search: games

Solved games!

Checkers:

- Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.



Adversarial search: games

- Adversarial search problems \equiv game
- Adversarial \equiv There is an **opponent** we can't control!
- Game vs. search: optimal solution is not a sequence of actions but a **strategy** (policy). If opponent does a , agent does b , else if opponent does c , agent does d , etc.
- Tedious and fragile if hard-coded (i.e., implemented with rules).
- **Concepts/methods:** Minimax algorithm, $\alpha - \beta$ pruning, stochastic games.

Machine learning

“How do we create computer programs that improve with experience?”

Tom Mitchell

Machine learning

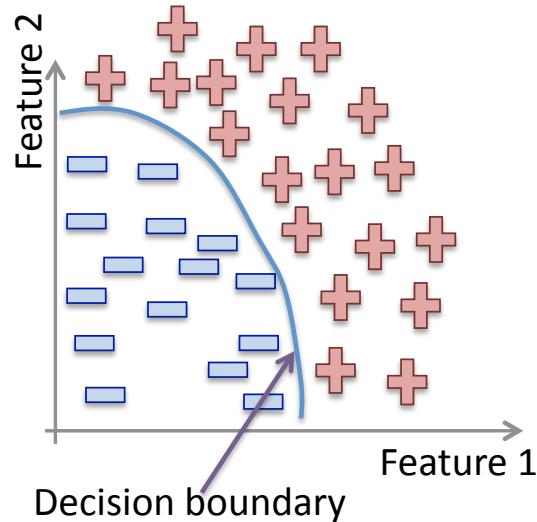
Binary classification (categorization)

Input: “Examples” with labels.

$$(x_1, y_1), \dots, (x_n, y_n) / x_i \in X \subset \mathbb{R}^n, y_i \in Y = \{-1, +1\}$$

Output: $h : X \rightarrow Y$

Example: Approve credit yes/no, spam/ham.



Concepts/methods: Supervised learning, classification, K nearest neighbors, perceptrons, neural networks, linear regression, etc.

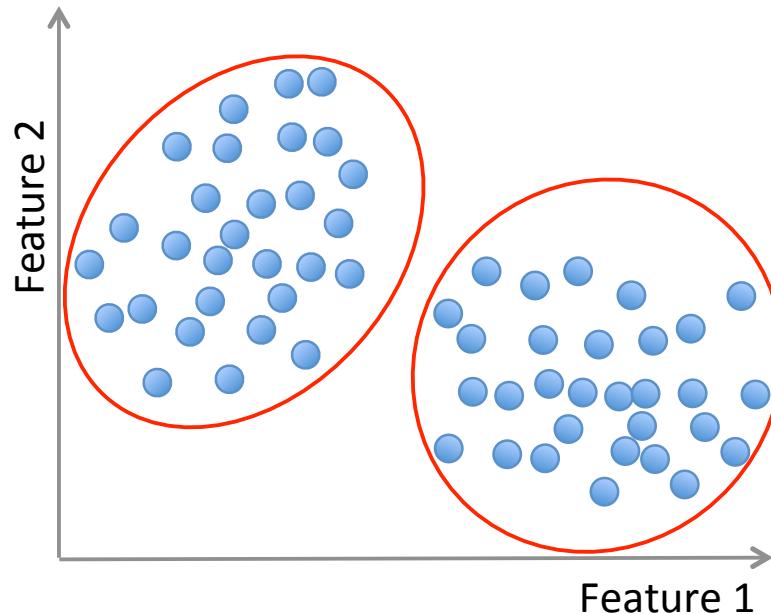
Machine learning

Data segmentation or Clustering

Input: “Examples” without labels.

$$x_1, \dots, x_n, x_i \in X \subset \mathbb{R}^n$$

Output: $f : X \rightarrow \{C_1, \dots, C_k\}$ (set of clusters).



Concepts/methods: Unsupervised learning, clustering, k-means, association rules, etc.

Constraint satisfaction

- A search problem too!
- We don't care about the path but about the **goal itself**.
- All paths are of same depth.
- Problem is formulated using variables, domains and constraints.
- Solving the CSP: **finding the assignment(s)** that **satisfy all constraints**.
- **Concepts/methods:** problem formalization, backtracking search, arc consistency, etc.

Constraint satisfaction

8		9	5		1	7	3	6
2		7		6	3			
1	6							
				9		4		7
	9		3		7		2	
7		6		8				
							6	3
			9	3		5		2
5	3	2	6		4	8		9

Variables: $X_{l,c}$ for $1 \leq l \leq 9$ and $1 \leq c \leq 9$.

Constraints: All 3x3 grid, row, column, must contain digits 1..9
and all of them!

Solution: Find the assignments to the variables that satisfy the constraints.

Constraint satisfaction

8	4	9	5	2	1	7	3	6
2	5	7	8	6	3	9	1	4
1	6	3	7	4	9	2	5	8
3	2	5	1	9	6	4	8	7
4	9	8	3	5	7	6	2	1
7	1	6	4	8	2	3	9	5
9	8	4	2	7	5	1	6	3
6	7	1	9	3	8	5	4	2
5	3	2	6	1	4	8	7	9

Variables: $X_{l,c}$ for $1 \leq l \leq 9$ and $1 \leq c \leq 9$.

Constraints: All 3x3 grid, row, column, must contain digits 1..9
and all of them!

Solution: Find the assignments to the variables that satisfy the constraints.

Logical Agents

- Logic can be used by an agent to model the world.
- Sentences in PL and FOL have a fixed **syntax**.
- With symbols and connectives we can form logical sentences:
Example: $hot \wedge sunny \Rightarrow beach \vee pool$
- Syntax and **Semantic** represent two important and distinct aspects in logic.
- **Inference:** Given a Knowledge Base (KB) (set of sentences in logic), given a query α , output whether KB entails α , noted:
 $KB \models \alpha$
- **Concepts/methods:** Modus Ponens, sound and complete inference, horn clauses, etc.

Reinforcement learning

- Agent evolves in a stochastic and uncertain environment.
- Agent learns from **reinforcement** or delayed **reward**.
- Learning approaches for **decision making** in situations where outcomes are stochastic.
- Agent continues to plan and learn to affect its environment.
- Reinforcement learning agents are driven by **maximizing their rewards on the long run**.

Applications

- **Natural Language Processing (NLP)**: concerned with the interactions between computers and human languages.
- **Vision/perception**: concerned with image processing and building computer vision agents. Goals: information extraction for tasks such as manipulation, navigation, and object recognition.
- **Robotics**: concerned with intelligent agents that manipulate the physical world. Different aspects: planning of robot motion, vision and object recognition.

Historical moment today



In memory of Alan Turing (1912-1954)

- Famous British mathematician.
- Code breaker during World War II.
- Proposed an operational test for intelligent behavior: The Imitation Game.
- In “Computing machinery and intelligence” (1950), he laid down AI major components:
(language, reasoning, knowledge, learning, understanding).

<http://www.turingarchive.org/browse.php/B/9>

Summary

- AI is a hard (computational complexity, language, vision, etc), and a broad field with high impact on humanity and society.
- What can AI do for us is already amazing!
- AI systems do not have to model human/nature but can act like or be inspired by human/nature.
- How human think is beyond the scope of this course.
- Rational (do the right thing) agents are central to our approach of AI.
- Note that rationality is not always possible in complicated environment but we will still aim to build rational agents.

Summary

- AI may be perceived as a scary area! Is AI a threat to our humankind?
- Professor Stephen Hawking, eminent scientist told BBC:

“The development of full artificial intelligence could spell the end of the human race.”

- AI is a flourishing and exciting field: everyone can contribute.
- Looking forward for an exciting journey together!

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

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

Artificial Intelligence

Intelligent Agents

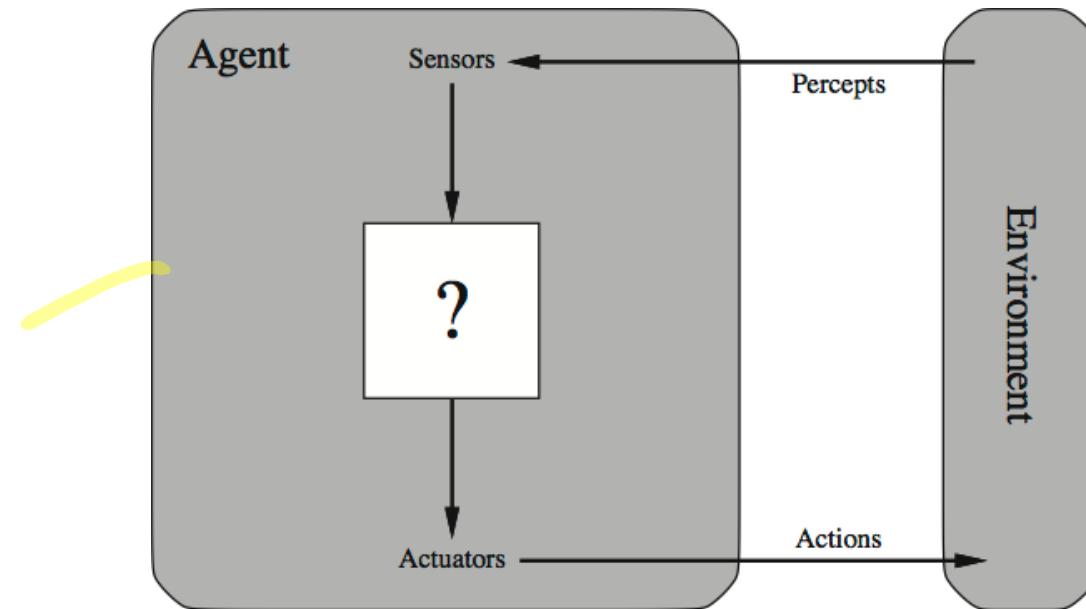


Agents and environments

- **Agent:** An **agent** is anything that can be viewed as:
 - perceiving its **environment** through **sensors** and
 - acting upon that **environment** through **actuators**.

Agents and environments

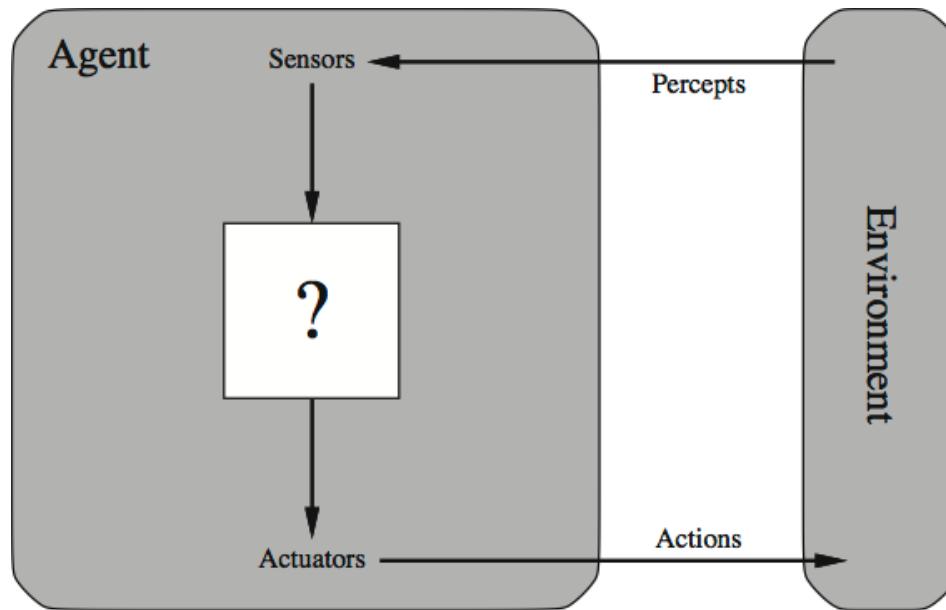
- **Agent:** An **agent** is anything that can be viewed as:
 - perceiving its **environment** through **sensors** and
 - acting upon that **environment** through **actuators**.



- An agent program runs in cycles of: (1)perceive, (2)think, and (3)act.

Agents and environments

- **Agent:** An **agent** is anything that can be viewed as:
 - perceiving its **environment** through **sensors** and
 - acting upon that **environment** through **actuators**.



- An agent program runs in cycles of: (1)perceive, (2)think, and (3)act.
Hardware
- **Agent = Architecture + Program**

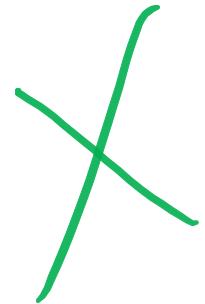
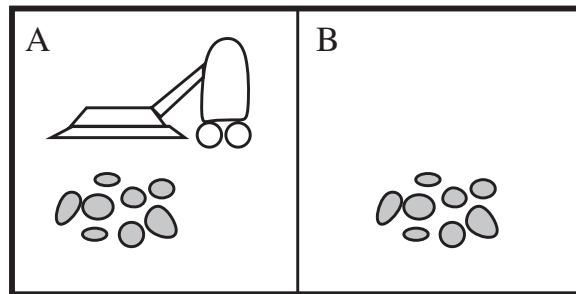
Agents and environments

- **Human agent:**
 - Sensors: eyes, ears, and other organs.
 - Actuators: hands, legs, mouth, and other body parts.
- **Robotic agent:**
 - Sensors: Cameras and infrared range finders.
 - Actuators: Various motors.

Agents and environments

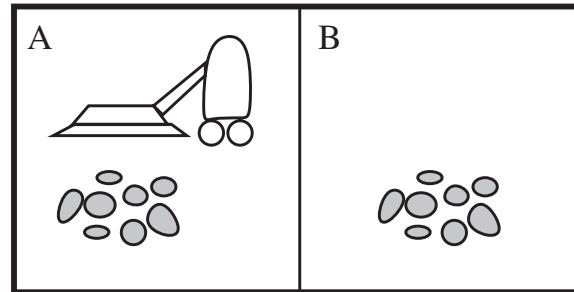
- **Human agent:**
 - Sensors: eyes, ears, and other organs.
 - Actuators: hands, legs, mouth, and other body parts.
- **Robotic agent:**
 - Sensors: Cameras and infrared range finders.
 - Actuators: Various motors.
- **Agents everywhere!**
 - Thermostat
 - Cell phone
 - Vacuum cleaner
 - Robot
 - Alexa Echo
 - Self-driving car
 - Human
 - etc.

Vacuum cleaner



- Percepts: location and contents e.g., [A, Dirty]
- Actions: Left, Right, Suck, NoOp
- Agent function: mapping from percepts to actions.

Vacuum cleaner



- Percepts: location and contents e.g., [A, Dirty]
- Actions: Left, Right, Suck, NoOp
- Agent function: mapping from percepts to actions.

Percept	Action
[A, clean]	Right
[A, dirty]	Suck
[B, clean]	Left
[B, dirty]	Suck

Action
done based
on percept

Well-behaved agents

Rational Agent:

*“For each possible percept sequence, a rational agent should select an action that is expected to maximize its **performance measure**, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.”*

Rationality

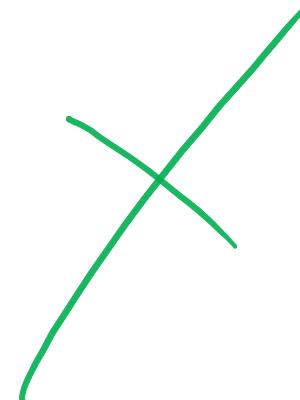
- Rationality is relative to a **performance measure**.
- Judge rationality based on:
 - The performance measure that defines the criterion of success.
 - The agent prior knowledge of the environment.
 - The possible actions that the agent can perform.
 - The agent's percept sequence to date.

PEAS

- When we define a rational agent, we group these properties under **PEAS**, the problem specification for the task environment.
- The rational agent we want to design for this task environment is the solution.
- PEAS stands for:
 - **P**erformance
 - **E**nvironment
 - **A**ctuators
 - **S**ensors

PEAS

What is PEAS for a self-driving car?



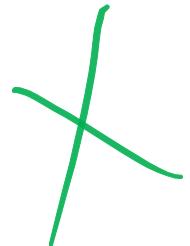
- **P**erformance:
- **E**nvironment:
- **A**ctuators:
- **S**ensors:

PEAS

What is PEAS for a self-driving car?

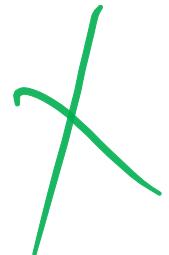


- **P**erformance: Safety, time, legal drive, comfort.
- **E**nvironment:
- **A**ctuators:
- **S**ensors:



PEAS

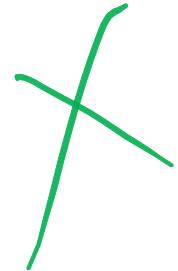
What is PEAS for a self-driving car?



- **P**erformance: Safety, time, legal drive, comfort.
- **E**nvironment: Roads, other cars, pedestrians, road signs.
- **A**ctuators:
- **S**ensors:

PEAS

What is PEAS for a self-driving car?



- **P**erformance: Safety, time, legal drive, comfort.
- **E**nvironment: Roads, other cars, pedestrians, road signs.
- **A**ctuators: Steering, accelerator, brake, signal, horn.
- **S**ensors:

PEAS

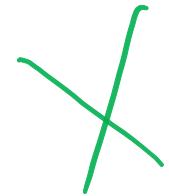
What is PEAS for a self-driving car?



- **P**erformance: Safety, time, legal drive, comfort.
- **E**nvironment: Roads, other cars, pedestrians, road signs.
- **A**ctuators: Steering, accelerator, brake, signal, horn.
- **S**ensors: Camera, sonar, GPS, Speedometer, odometer, accelerometer, engine sensors, keyboard.

PEAS

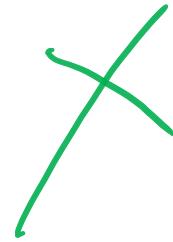
How about a vacuum cleaner?



iRobot Roomba series

PEAS

How about a vacuum cleaner?



iRobot Roomba series

- **P**erformance: cleanliness, efficiency: distance traveled to clean, battery life, security.
- **E**nvironment:
- **A**ctuators:
- **S**ensors:

PEAS

How about a vacuum cleaner?



iRobot Roomba series

- **P**erformance: cleanliness, efficiency: distance traveled to clean, battery life, security.
- **E**nvironment: room, table, wood floor, carpet, different obstacles.
- **A**ctuators:
- **S**ensors:

PEAS

How about a vacuum cleaner?



iRobot Roomba series

- **P**erformance: cleanliness, efficiency: distance traveled to clean, battery life, security.
- **E**nvironment: room, table, wood floor, carpet, different obstacles.
- **A**ctuators: wheels, different brushes, vacuum extractor.
- **S**ensors:

PEAS

How about a vacuum cleaner?



iRobot Roomba series

- **P**erformance: cleanliness, efficiency: distance traveled to clean, battery life, security.
- **E**nvironment: room, table, wood floor, carpet, different obstacles.
- **A**ctuators: wheels, different brushes, vacuum extractor.
- **S**ensors: camera, dirt detection sensor, cliff sensor, bump sensors, infrared wall sensors.

Environment types

- **Fully observable (vs. partially observable):** An agent's sensors give it access to the complete state of the environment at each point in time.
- **Deterministic (vs. stochastic):** The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic) *(an agent cannot know what happens in another street)*
- **Episodic (vs. sequential):** The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.

Environment types

- **Static (vs. dynamic):** The environment is unchanged while an agent is deliberating. (The environment is semi-dynamic if the environment itself does not change with the passage of time but the agent's performance score does.) *Eg: chess + (1/1)*
- **Discrete (vs. continuous):** A limited number of distinct, clearly defined percepts and actions. E.g., checkers is an example of a discrete environment, while self-driving car evolves in a continuous one.
- **Single agent (vs. multi-agent):** An agent operating by itself in an environment.
- **Known (vs. Unknown):** The designer of the agent may or may not have knowledge about the environment makeup. If the environment is unknown the agent will need to know how it works in order to decide. *Eg: TomTom*

Environment types

Environment	Observable	Agents	Deterministic	Static	Discrete
8-puzzle	Fully	Single	Deterministic	Static	Discrete
Chess	Fully	Multi	Deterministic	(Semi)Static	Discrete
Pocker	Partially	Multi	Stochastic	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Static	Discrete
Car	Partially	Multi	Stochastic	Dynamic	Continuous
Roomba	Partially	Single	Stochastic	Dynamic	Continuous

Handwritten annotations:

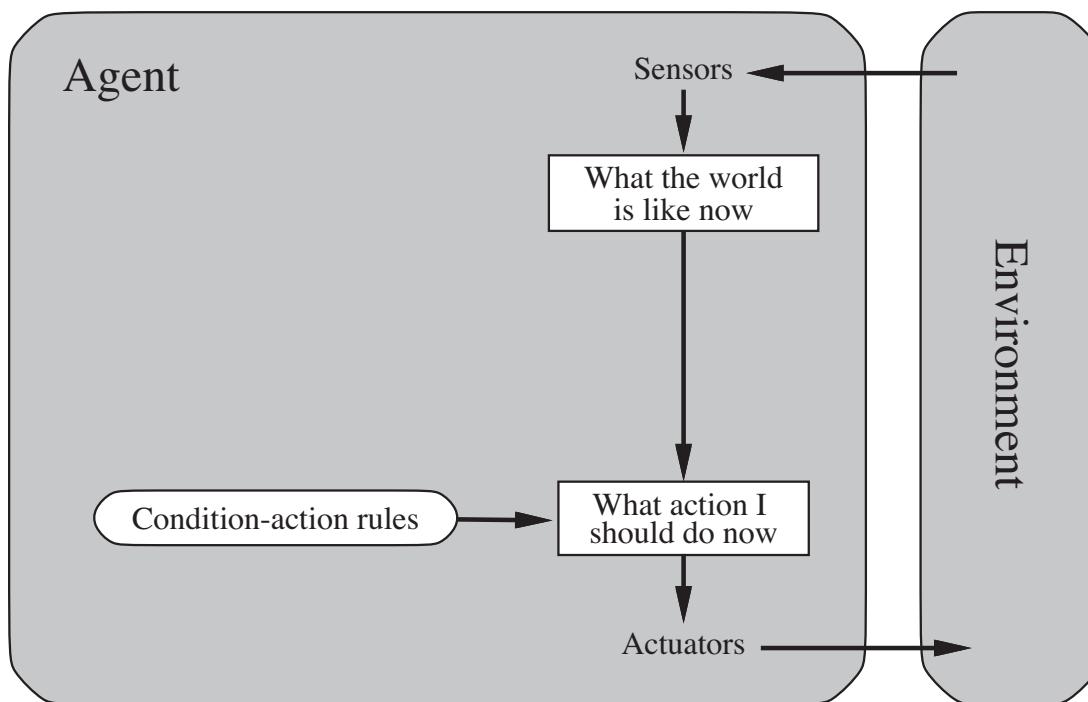
- A blue arrow points from "Agents" to "One player".
- A blue arrow points from "Deterministic" to "deck cards".
- A blue arrow points from "Discrete" to "finite things to do".
- A blue arrow points from "Dynamic" to "always changing".

Agent types

- Four basic types in order of increasing generality:
 - Simple reflex agents
 - Model-based reflex agents
 - Goal-based agents
 - Utility-based agents
- All of which can be generalized into learning agents that can improve their performance and generate better actions.

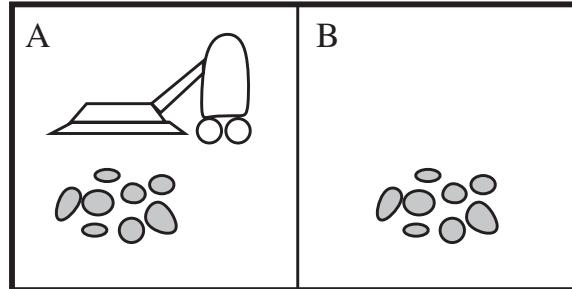
Simple reflex agents

- Simple reflex agents select an action based on the current state only ignoring the percept history.
- Simple but limited.
- Can only work if the environment is **fully observable**, that is the correct action is based on the **current** percept only.



Vacuum (reflex) agent

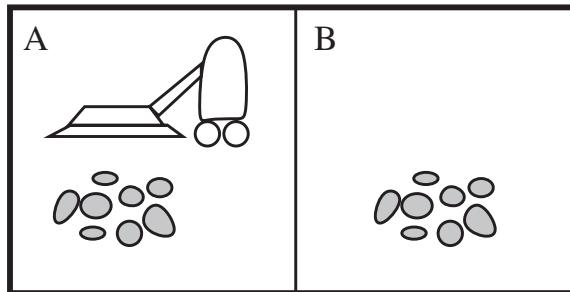
Simple reflex



- Let's write the algorithm for the Vacuum cleaner...
- Percepts: location and content (location sensor, dirt sensor).
- Actions: Left, Right, Suck, NoOp

Percept	Action
[A, clean]	Right
[A, dirty]	Suck
[B, clean]	Left
[B, dirty]	Suck

Vacuum (reflex) agent



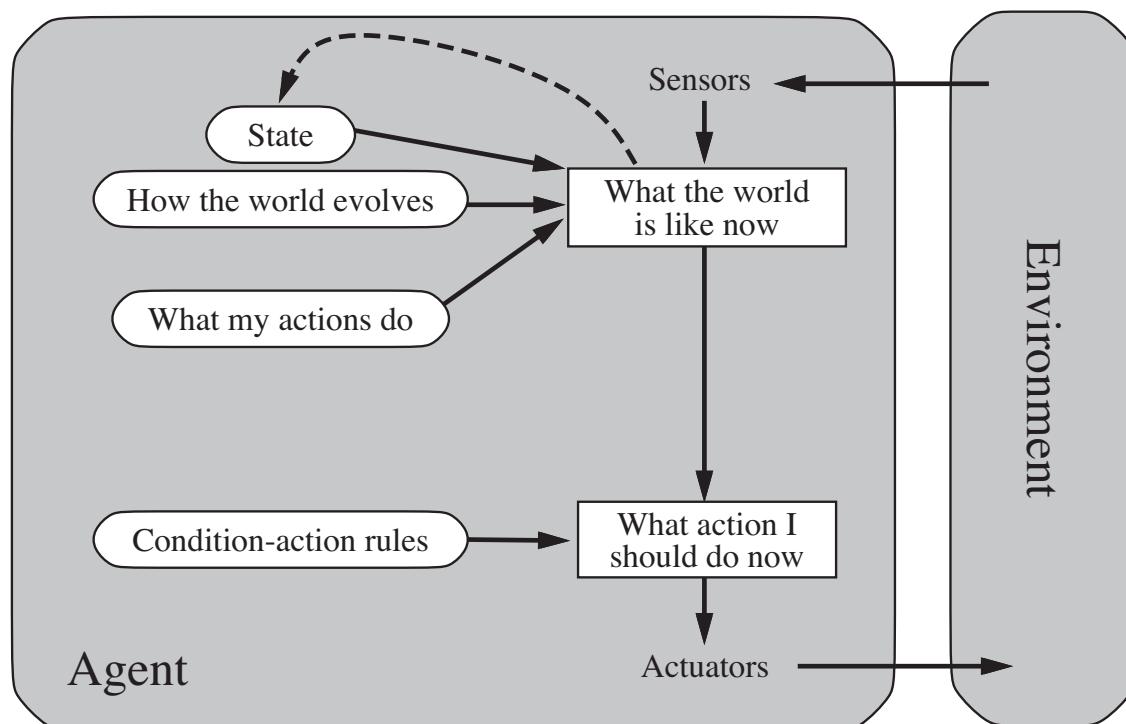
- Let's write the algorithm for the Vacuum cleaner...
- Percepts: location and content (location sensor, dirt sensor).
- Actions: Left, Right, Suck, NoOp

Percept	Action
[A, clean]	Right
[A, dirty]	Suck
[B, clean]	Left
[B, dirty]	Suck

What if the vacuum agent is deprived from its location sensor? Turn the agent to randomized for the location

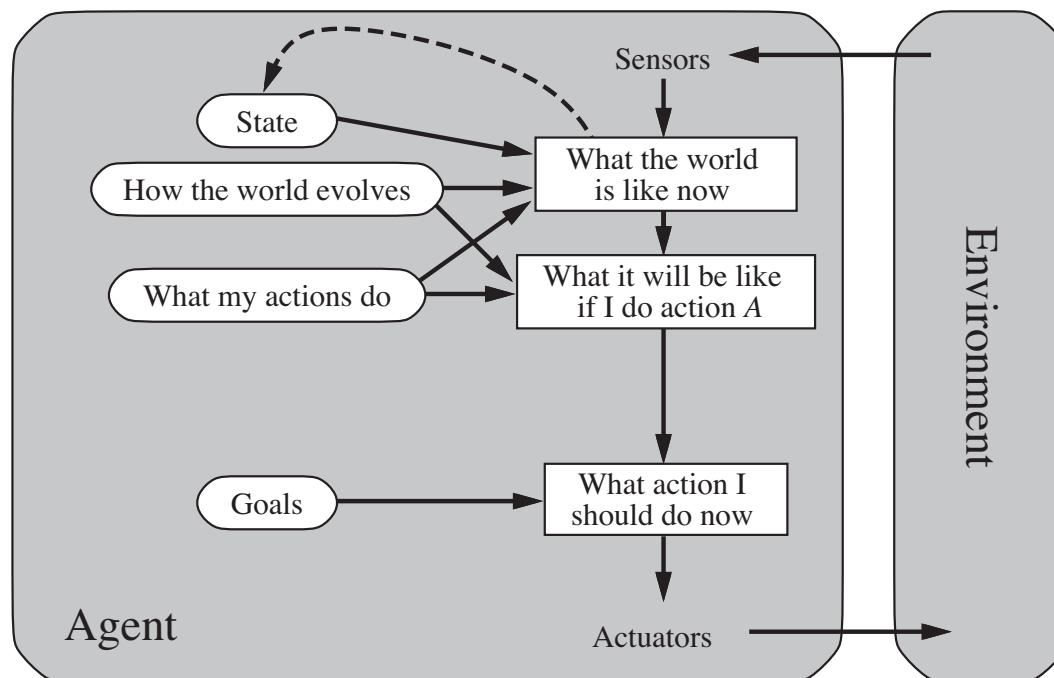
Model-based reflex agents

- Handle partial observability by keeping track of the part of the world it can't see now.
- Internal state depending on the percept history (best guess).
- Model of the world based on (1) how the world evolves independently from the agent, and (2) how the agent actions affects the world.



Goal-based agents

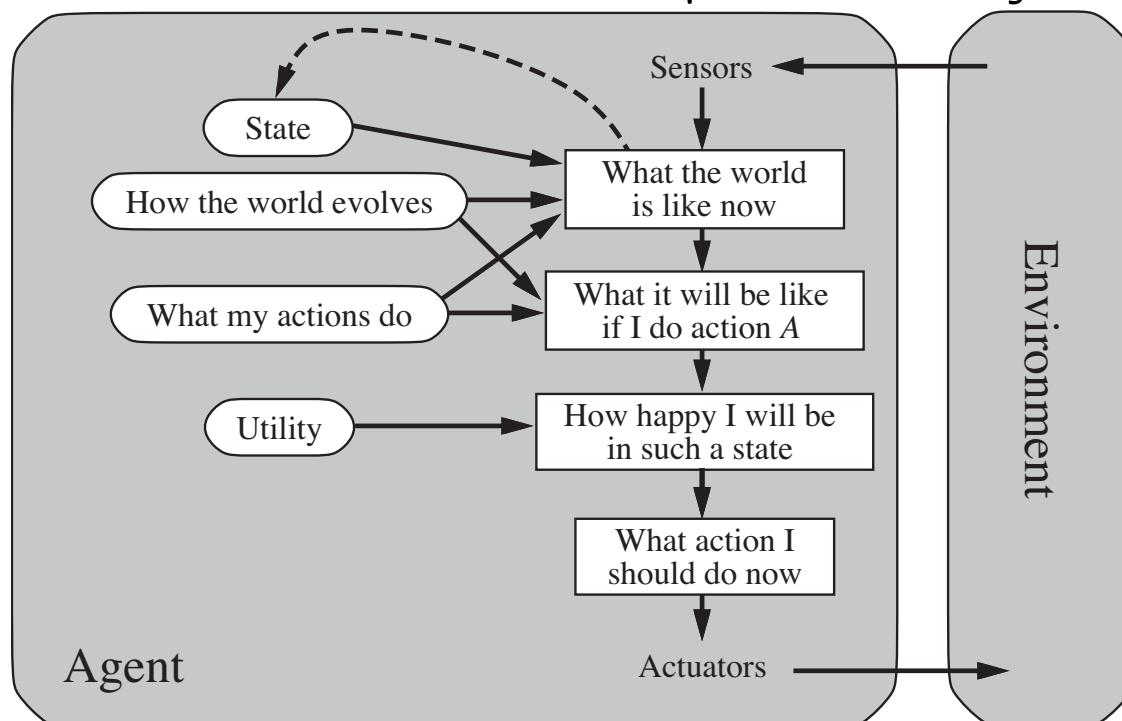
- Knowing the current state of the environment is not enough. The agent needs some **goal information**.
- Agent program combines the goal information with the environment model to choose the actions that achieve that goal.
- Consider the future with “What will happen if I do A?”
- Flexible as knowledge supporting the decisions is explicitly represented and can be modified.



having a table
for lookup will
be large

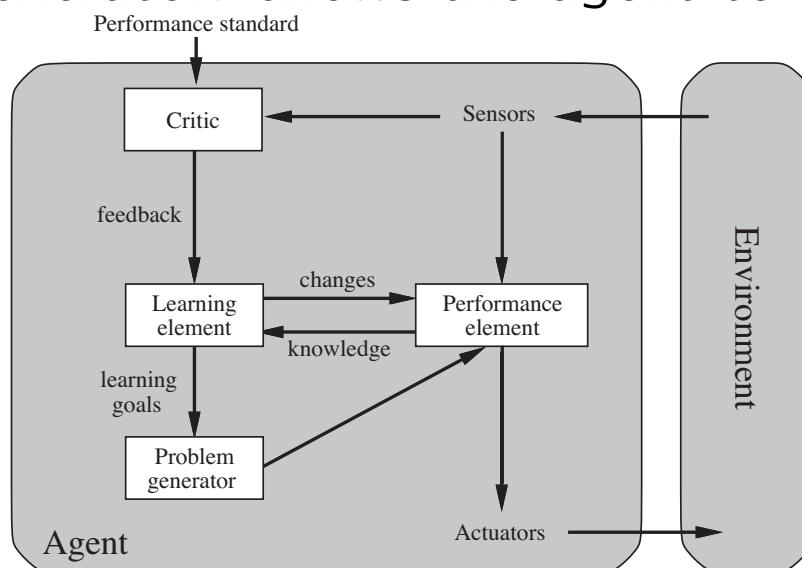
Utility-based agents

- Sometimes achieving the desired goal is not enough. We may look for quicker, safer, cheaper trip to reach a destination.
- Agent happiness should be taken into consideration. We call it **utility**.
- A utility function is the agent's performance measure
- Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility.



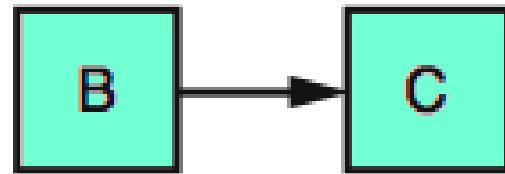
Learning agents

- Programming agents by hand can be very tedious. “Some more expeditious method seem desirable” Alan Turing, 1950.
- Four conceptual components:
 - Learning element: responsible for making improvements
 - Performance element: responsible for selecting external actions. It is what we considered as agent so far.
 - Critic: How well is the agent is doing w.r.t. a fixed performance standard.
 - Problem generator: allows the agent to explore.



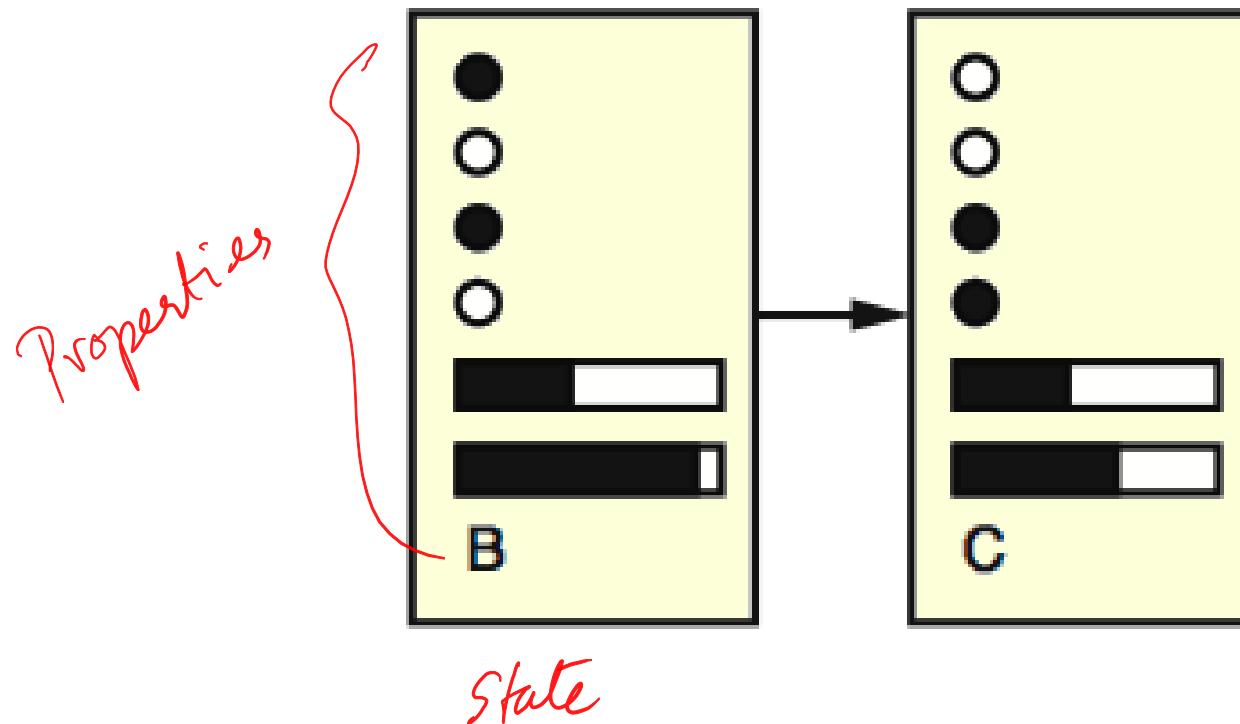
Agent's organization

a) **Atomic Representation:** Each state of the world is a **black-box** that has no internal structure. E.g., finding a driving route, each state is a city. AI algorithms: search, games, Markov decision processes, hidden Markov models, etc.



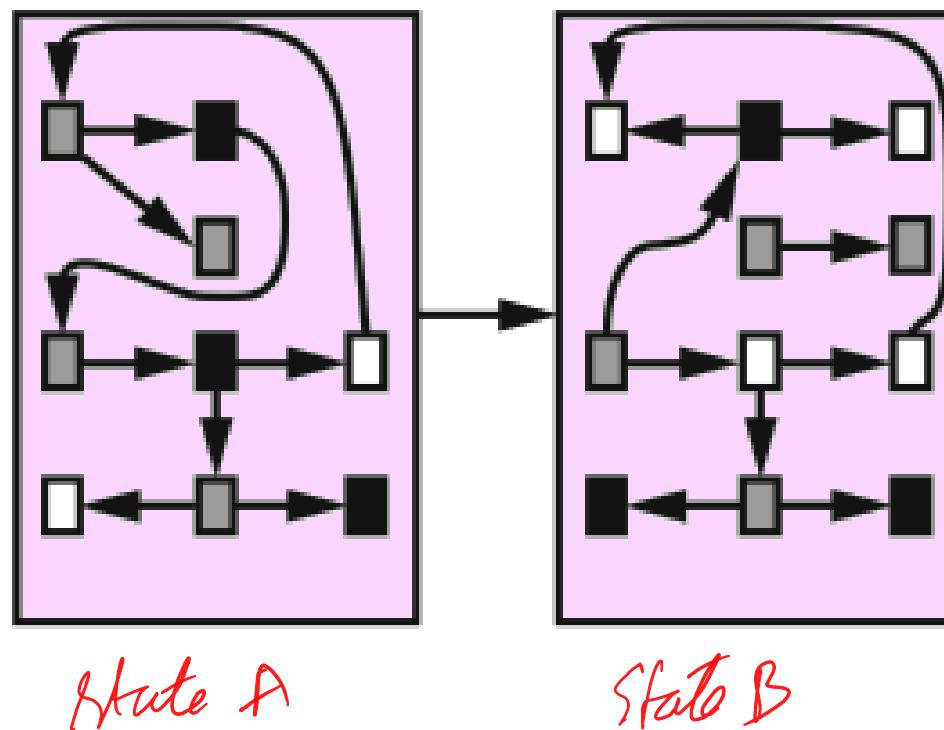
Agent's organization

b) **Factored Representation:** Each state has some **attribute-value properties**. E.g., GPS location, amount of gas in the tank. AI algorithms: constraint satisfaction, and Bayesian networks.



Agent's organization

c) **Structured Representation:** Relationships between the objects of a state can be explicitly expressed. AI algorithms: first order logic, knowledge-based learning, natural language understanding.



Intelligent agents

-Summary

- The concept of intelligent agent is central in AI.
- AI aims to design intelligent agents that are useful, reactive, autonomous and even social and pro-active.
- An agent perceives its environment through percept and acts through actuators.
- A performance measure evaluates the behavior of the agent.
- An agent that acts to maximize its expected performance measure is called a rational agent.
- PEAS: A task environment specification that includes Performance measure, Environment, Actuators and Sensors.

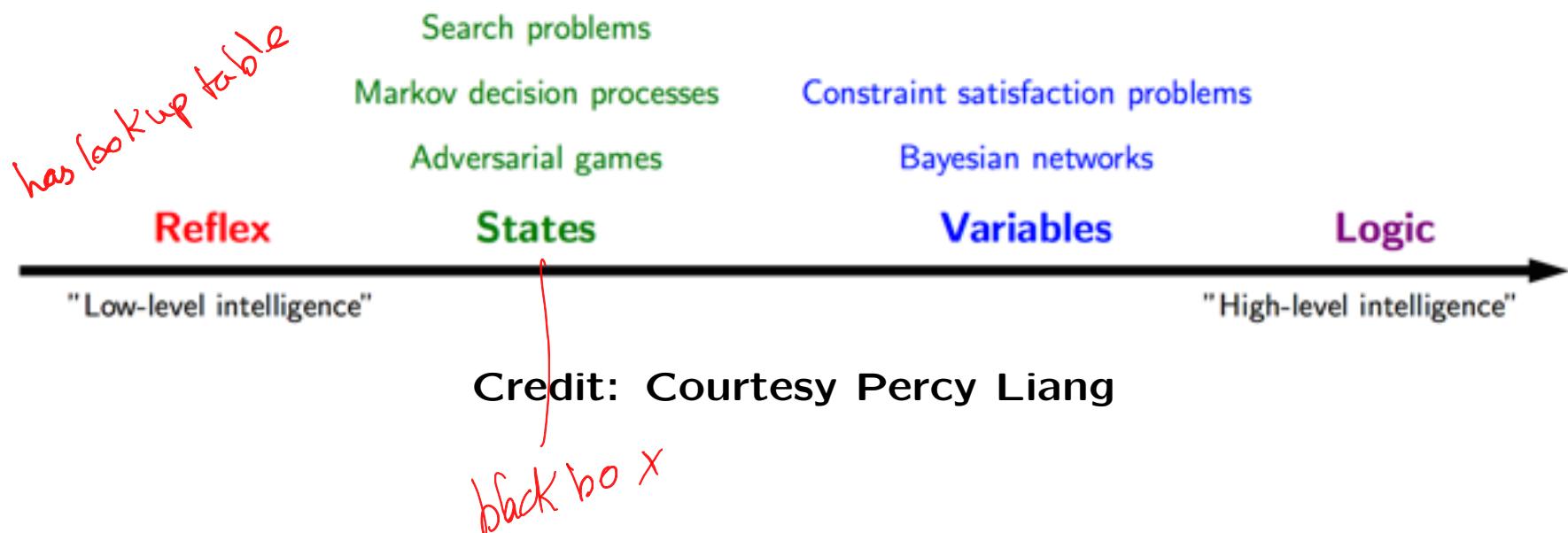
Agent = Architecture + Program

Intelligent agents

- Four types of agents: Reflex agents, model-based agents, goal-based agents, and utility-based agents. - *happiness ↑*
- Agents can improve their performance through **learning**.
- This is a high-level present of agent programs.
- States representations: atomic, factored, structured. Increasing expressiveness power.

Intelligent agents

- Four types of agents: Reflex agents, model-based agents, goal-based agents, and utility-based agents.
- Agents can improve their performance through **learning**.
- This is a high-level present of agent programs.
- States representations: atomic, factored, structured. Increasing expressiveness power.



Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

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

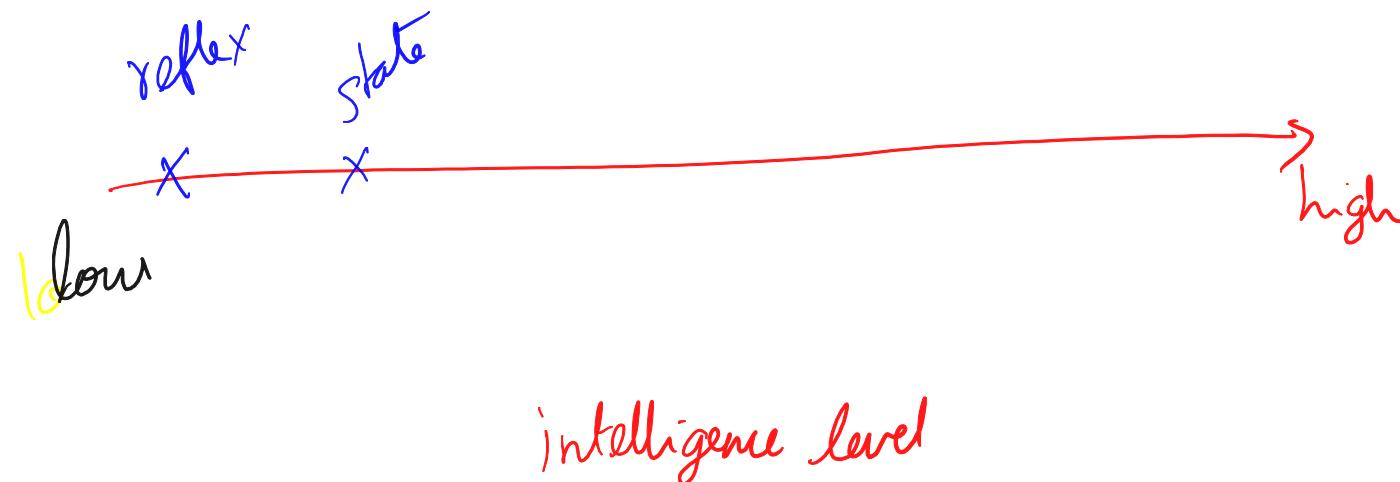
Artificial Intelligence

Search Agents



Goal-based agents

- no intelligence*
- **Reflex agents:** use a mapping from states to actions. (*table*)
 - **Goal-based agents:** problem solving agents or planning agents.



Goal-based agents

- Agents that work towards a **goal**.
- Agents consider the impact of **actions** on future **states**.
- Agent's job is to identify the action or series of actions that lead to the goal.

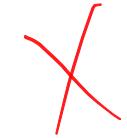


Goal-based agents

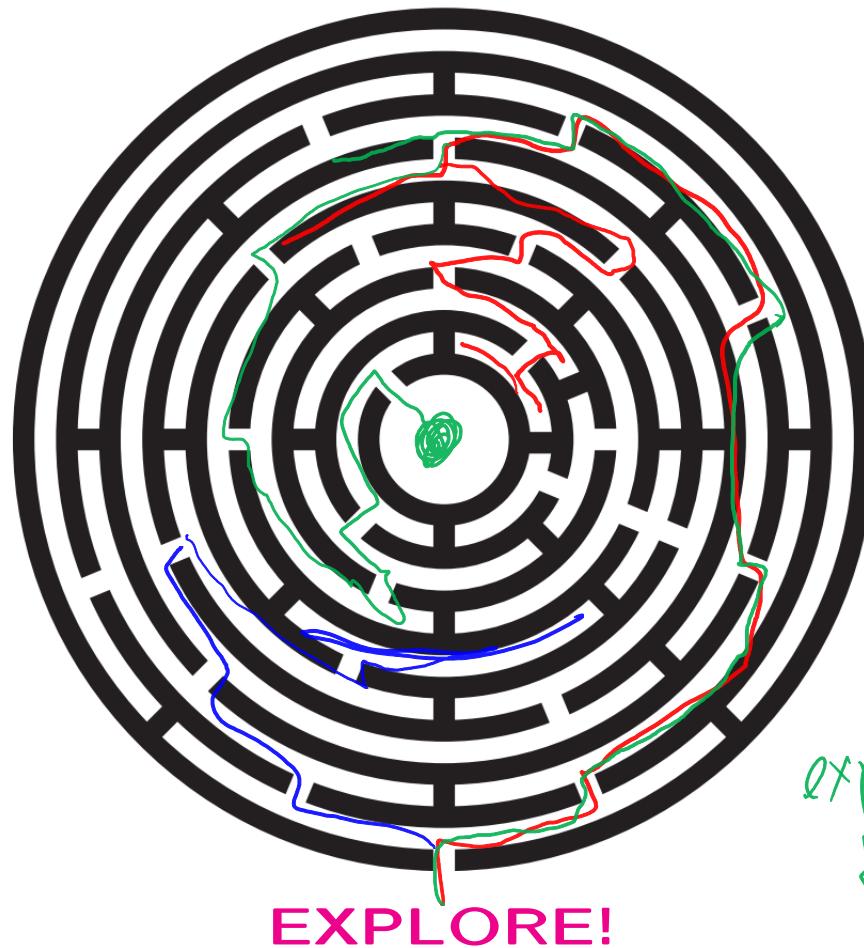
- Agents that work towards a **goal**.
- Agents consider the impact of **actions** on future **states**.
- Agent's job is to identify the action or series of actions that lead to the goal.
- Formalized as a **search** through possible **solutions**.



Examples



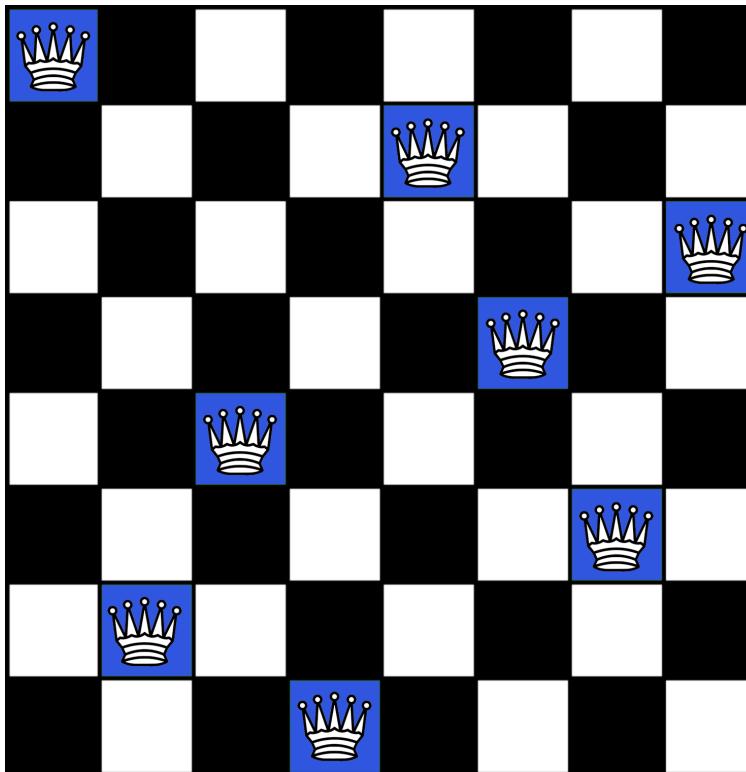
Examples



EXPLORE!

explored the search space
to reach a goal.

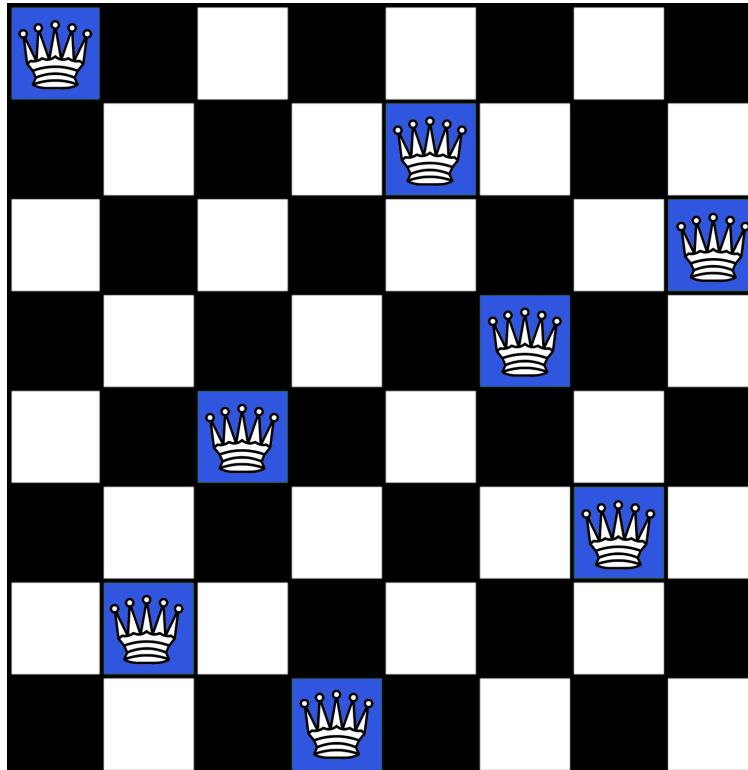
Examples



The 8-queen problem: on a chess board, place 8 queens so that no queen is attacking any other horizontally, vertically or diagonally.

Examples

good state
- put 8 Queens
without conflict



Number of possible sequences to investigate:

$$64 * 63 * 62 * \dots * 57 = 1.8 \times 10^{14}$$

search space

All possible sequences

Goal - find one (at least)

Problem solving as search

1. **Define the problem through:**

- (a) Goal formulation
- (b) Problem formulation

2. **Solving the problem as a 2-stage process:**

- (a) Search: “mental” or “offline” exploration of several possibilities
- (b) Execute the solution found

Problem formulation

- **Initial state:** the state in which the agent starts

✗

Problem formulation

- **Initial state:** the state in which the agent starts
- **States:** All states reachable from the initial state by any sequence of actions (**State space**)

X

Problem formulation

- **Initial state:** the state in which the agent starts
- **States:** All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions:** possible actions available to the agent. At a state s , $Actions(s)$ returns the set of actions that can be executed in state s . (**Action space**)



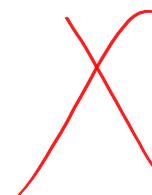
Problem formulation

- **Initial state:** the state in which the agent starts
- **States:** All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions:** possible actions available to the agent. At a state s , $Actions(s)$ returns the set of actions that can be executed in state s . (**Action space**)
- **Transition model:** A description of what each action does
 $Results(s, a)$



Problem formulation

- **Initial state:** the state in which the agent starts
- **States:** All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions:** possible actions available to the agent. At a state s , $Actions(s)$ returns the set of actions that can be executed in state s . (**Action space**)
- **Transition model:** A description of what each action does
 $Results(s, a)$
- **Goal test:** determines if a given state is a goal state

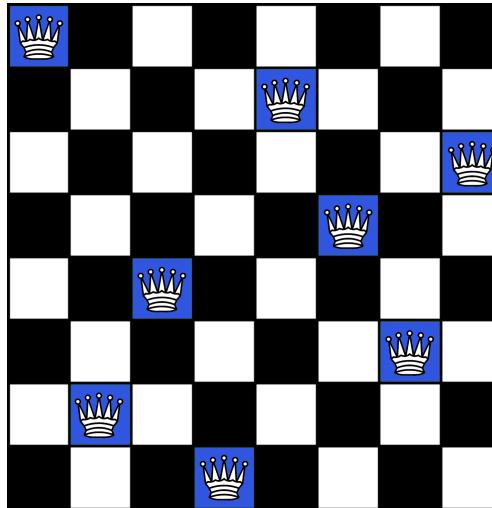


Problem formulation

- **Initial state:** the state in which the agent starts
- **States:** All states reachable from the initial state by any sequence of actions (State space)
- **Actions:** possible actions available to the agent. At a state s , $Actions(s)$ returns the set of actions that can be executed in state s . (Action space)
- **Transition model:** A description of what each action does
 $Results(s, a)$
- **Goal test:** determines if a given state is a goal state
- **Path cost:** function that assigns a numeric cost to a path w.r.t. performance measure

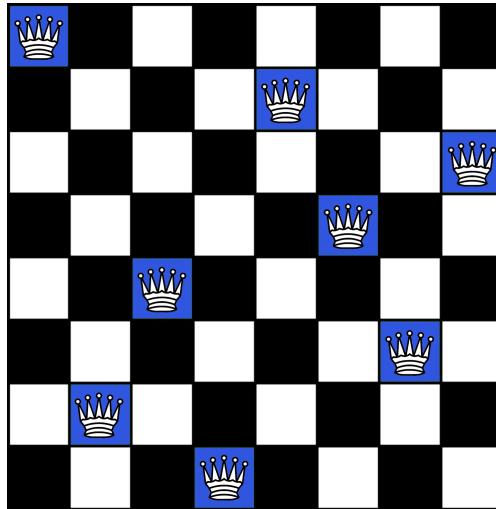
how much each solution
Costs

Examples



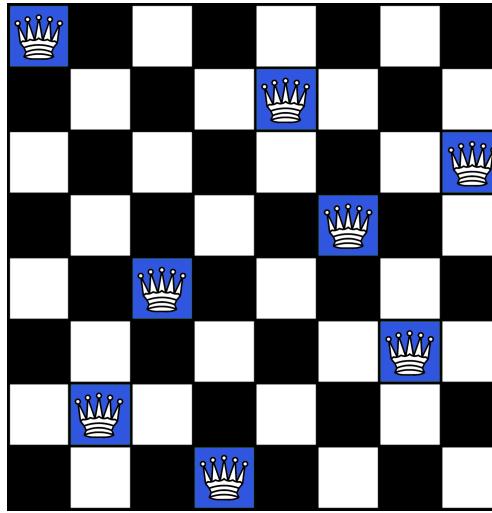
- **States:** all arrangements of 0 to 8 queens on the board.

Examples

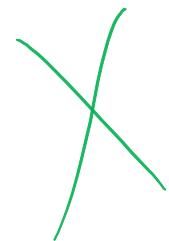


- **States:** all arrangements of 0 to 8 queens on the board.
- **Initial state:** No queen on the board

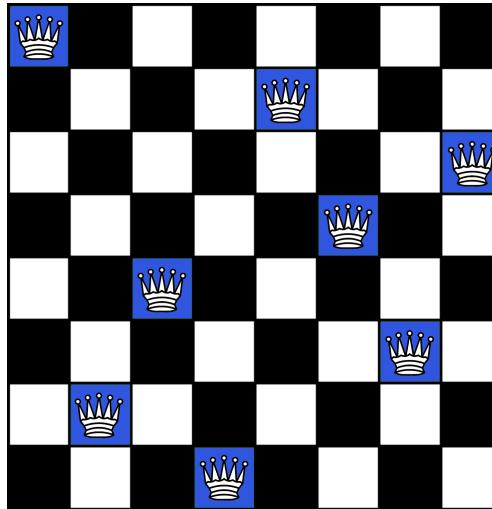
Examples



- **States:** all arrangements of 0 to 8 queens on the board.
- **Initial state:** No queen on the board
- **Actions:** Add a queen to any empty square



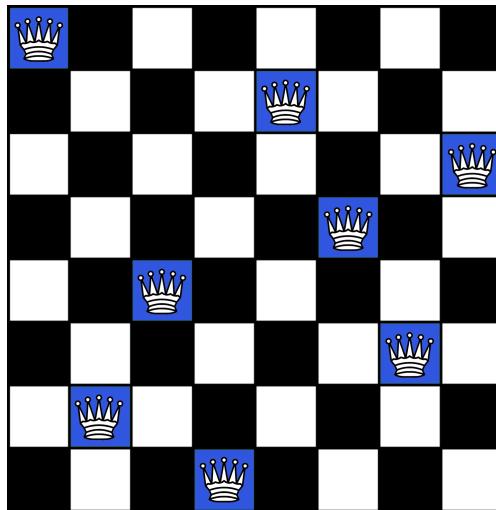
Examples



X

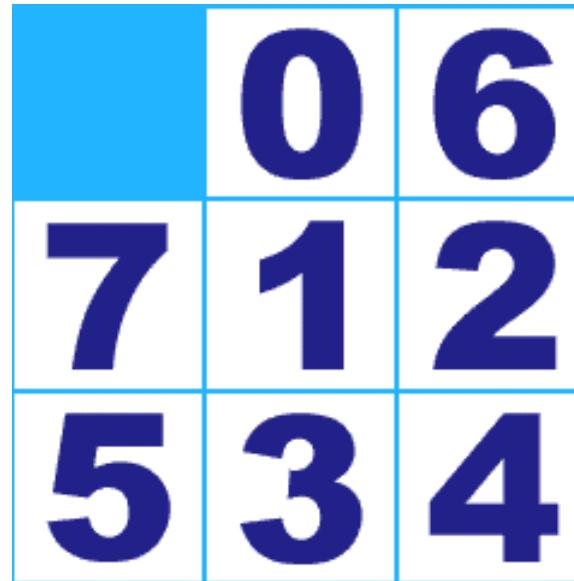
- **States:** all arrangements of 0 to 8 queens on the board.
- **Initial state:** No queen on the board
- **Actions:** Add a queen to any empty square
- **Transition model:** updated board

Examples



- **States:** all arrangements of 0 to 8 queens on the board.
- **Initial state:** No queen on the board
- **Actions:** Add a queen to any empty square
- **Transition model:** updated board
- **Goal test:** 8 queens on the board with none attacked

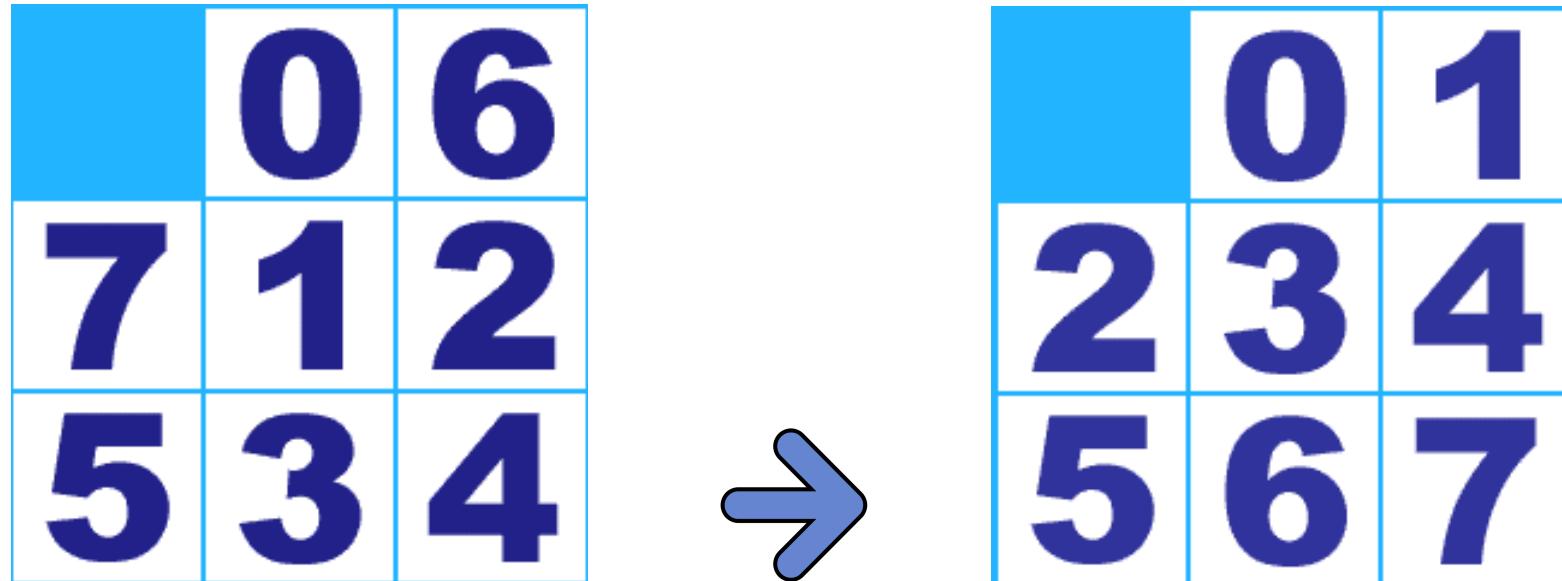
Examples



8 puzzles

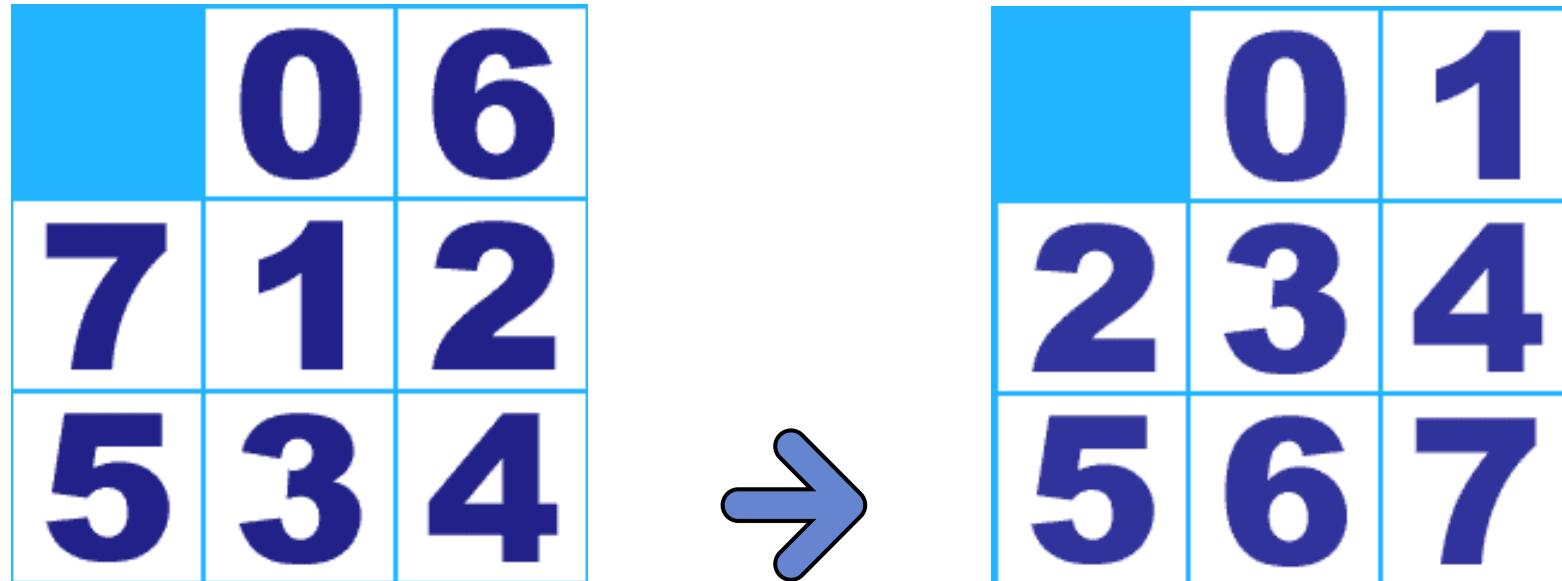


Examples



X

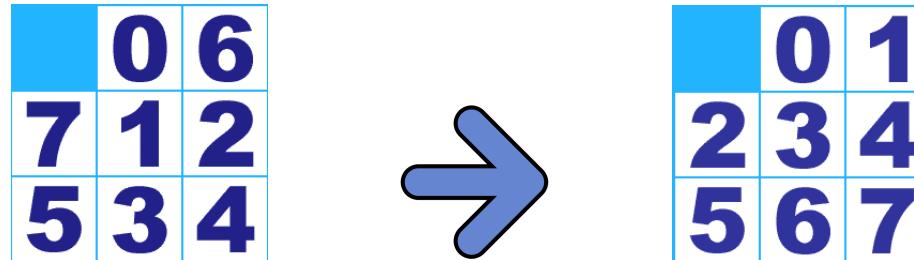
Examples



Start State

Goal State

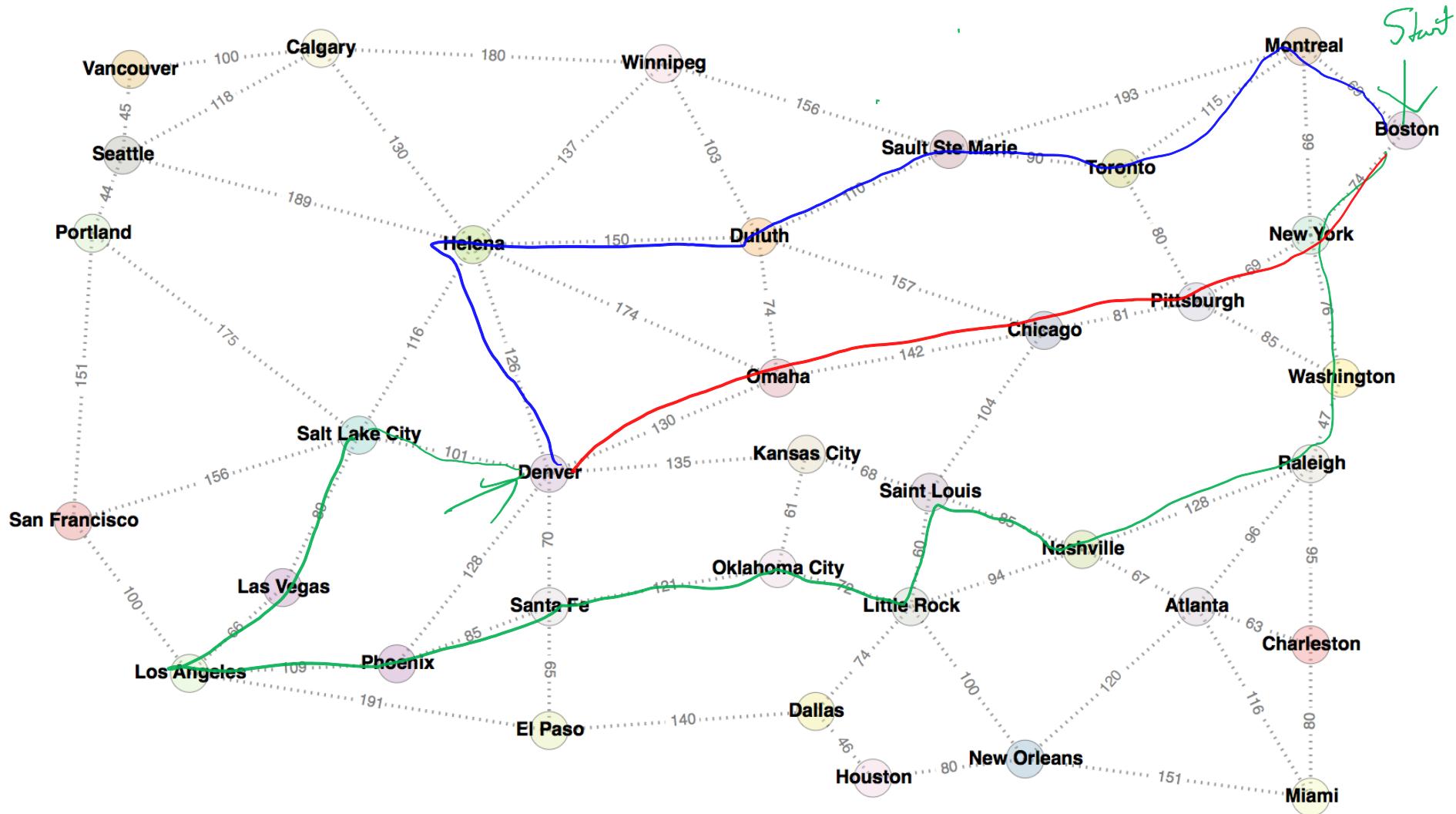
Examples



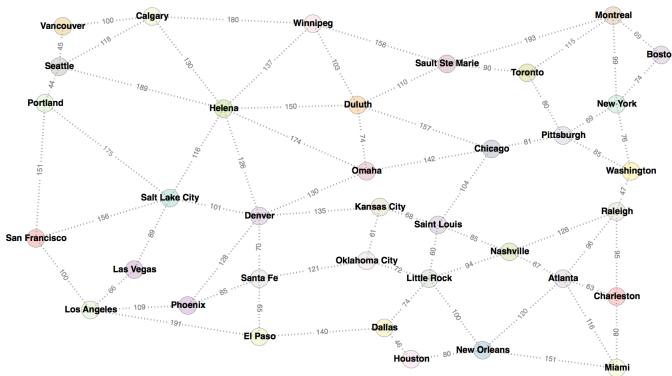
- **States:** Location of each of the 8 tiles in the 3x3 grid
- **Initial state:** Any state
- **Actions:** Move Left, Right, Up or Down
- **Transition model:** Given a state and an action, returns resulting state
- **Goal test:** state matches the goal state?
- **Path cost:** total moves, each move costs 1.

Examples of search agents

Routing problem



Examples



- **States:** In City where
City $\in \{\text{Los Angeles, San Francisco, Denver, ...}\}$
- **Initial state:** In Boston
- **Actions:** Go New York, etc.
- **Transition model:**
Results (In (Boston), Go (New York)) = In(New York)
- **Goal test:** In(Denver) ? No
- **Path cost:** path length in kilometers

Real-world examples

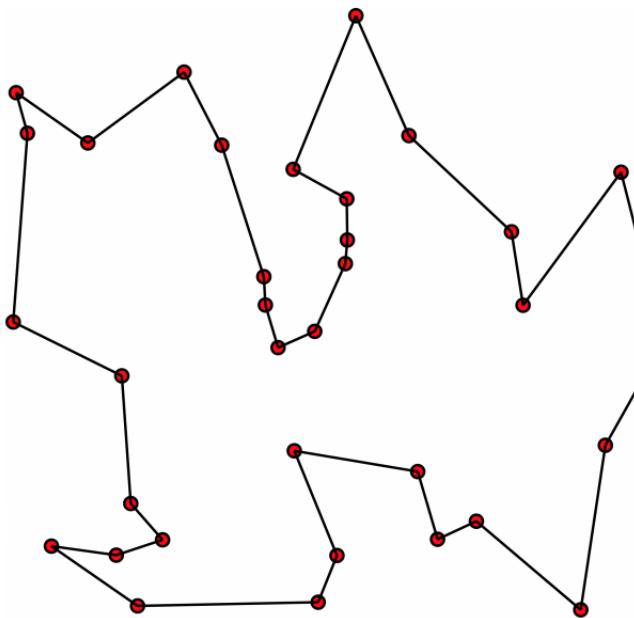
L.2.2.3

- **Route finding problem:** typically our example of map search, where we need to go from location to location using links or transitions. Example of applications include tools for driving directions in websites, in-car systems, etc.



Real-world examples

- **Traveling salesperson problem:** Find the shortest tour to visit each city exactly once.



Real-world examples

- **VLSI layout:** position million of components and connections on a chip to minimize area, shorten delays. Aim: put circuit components on a chip so as they don't overlap and leave space to wiring which is a complex problem.



Real-world examples

- **Robot navigation:** Special case of route finding for robots with no specific routes or connections. The robot navigates in 2D or 3D space or where the state space and action space are potentially infinite.



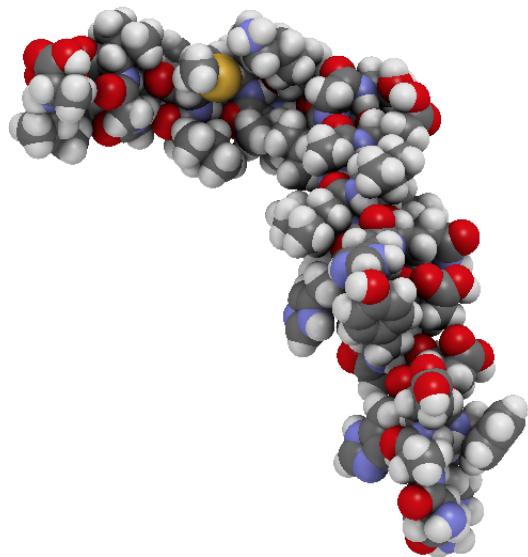
Real-world examples

- **Automatic assembly sequencing:** find an order in which to assemble parts of an object which is in general a difficult and expensive geometric search.



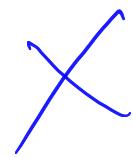
Real-world examples

- **Protein design:** find a sequence of amino acids that will fold into a 3D protein with the right properties to cure some disease.



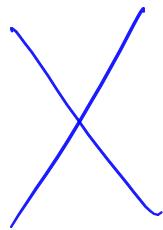
State space vs. search space

- **State space**: a *physical* configuration



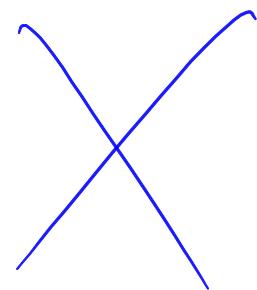
State space vs. search space

- **State space:** a *physical* configuration
- **Search space:** an *abstract* configuration represented by a search tree or graph of possible solutions.



State space vs. search space

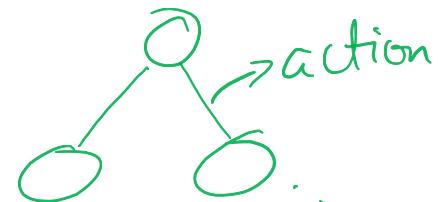
- **State space:** a *physical* configuration
- **Search space:** an *abstract* configuration represented by a search tree or graph of possible solutions.
- **Search tree: models the sequence of actions**
 - Root: initial state
 - Branches: actions
 - Nodes: results from actions. A node has: parent, children, depth, path cost, associated state in the state space.



State space vs. search space

for 8 puzzle { set of all possible configs }

- **State space:** a *physical* configuration
- **Search space:** an *abstract* configuration represented by a search tree or graph of possible solutions.
- **Search tree: models the sequence of actions**
 - Root: initial state
 - Branches: actions
 - Nodes: results from actions. A node has: parent, children, depth, path cost, associated state in the state space.
- **Expand:** A function that given a node, creates all children nodes



Search Space Regions

- The search space is divided into three regions:
 1. **Explored** (a.k.a. Closed List, Visited Set) *already visited in the search tree*
 2. **Frontier** (a.k.a. Open List, the Fringe) *→ next in line*
 3. **Unexplored**. —
- The essence of search is moving nodes from regions (3) to (2) to (1), and the essence of search strategy is deciding the order of such moves.
- In the following we adopt the following color coding: orange nodes are explored, grey nodes are the frontier, white nodes are unexplored, and black nodes are failures.

Tree search

L2.2.4

```
function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    initialize frontier with initialState

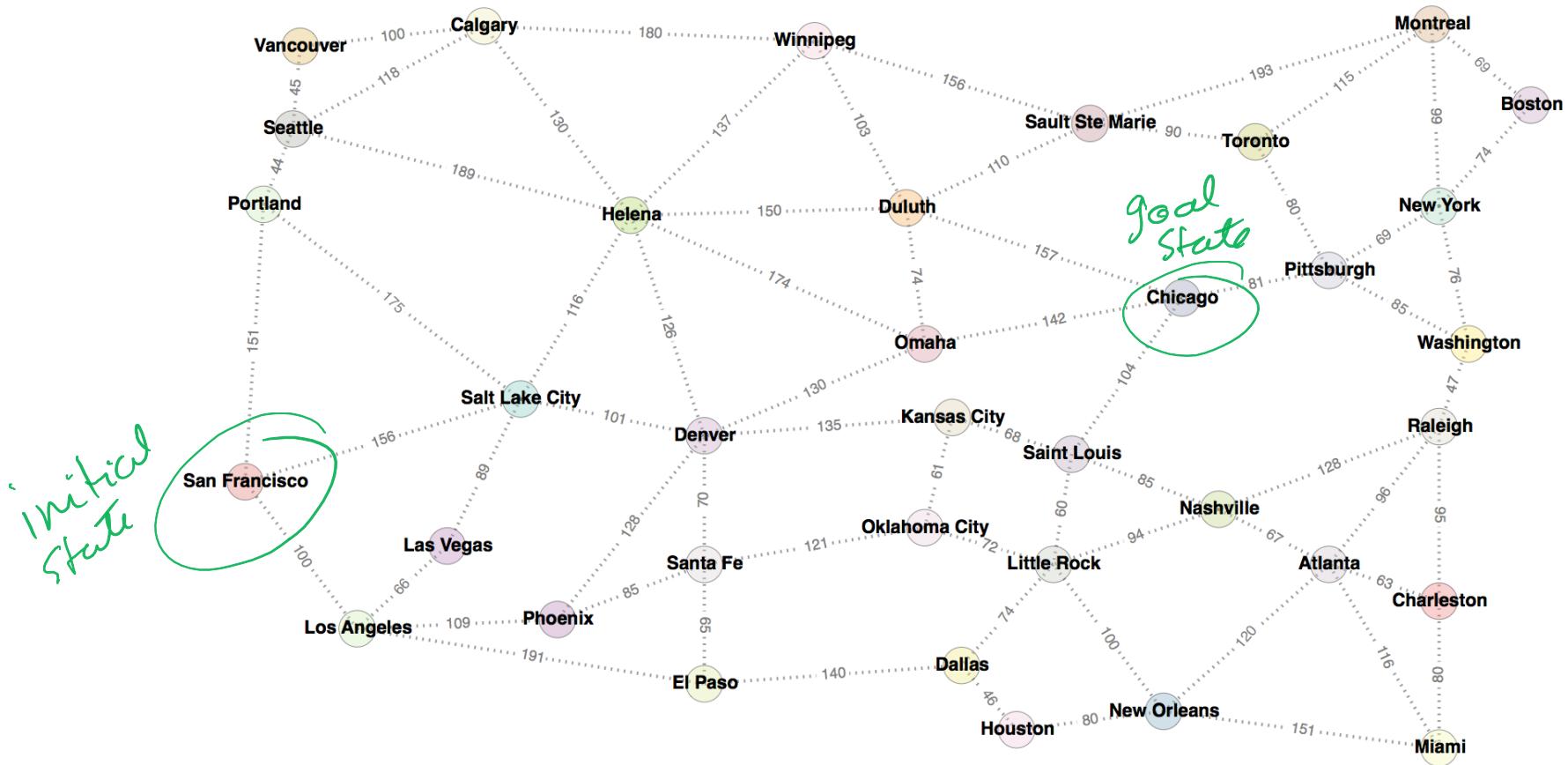
    while not frontier.isEmpty():
        state = frontier.remove()

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            frontier.add(neighbor)

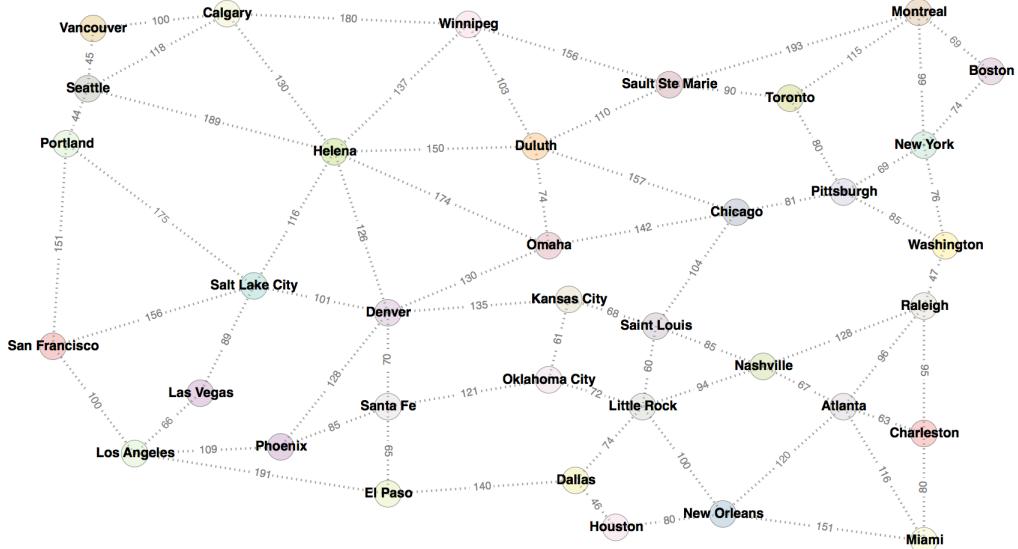
    return FAILURE
```

Examples of search agents



Let's show the first steps in growing the search tree to find a route from San Francisco to another city

Examples of search agents



```
function TREE-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE :

  initialize frontier with initialState

  while not frontier.isEmpty():
    state = frontier.remove()

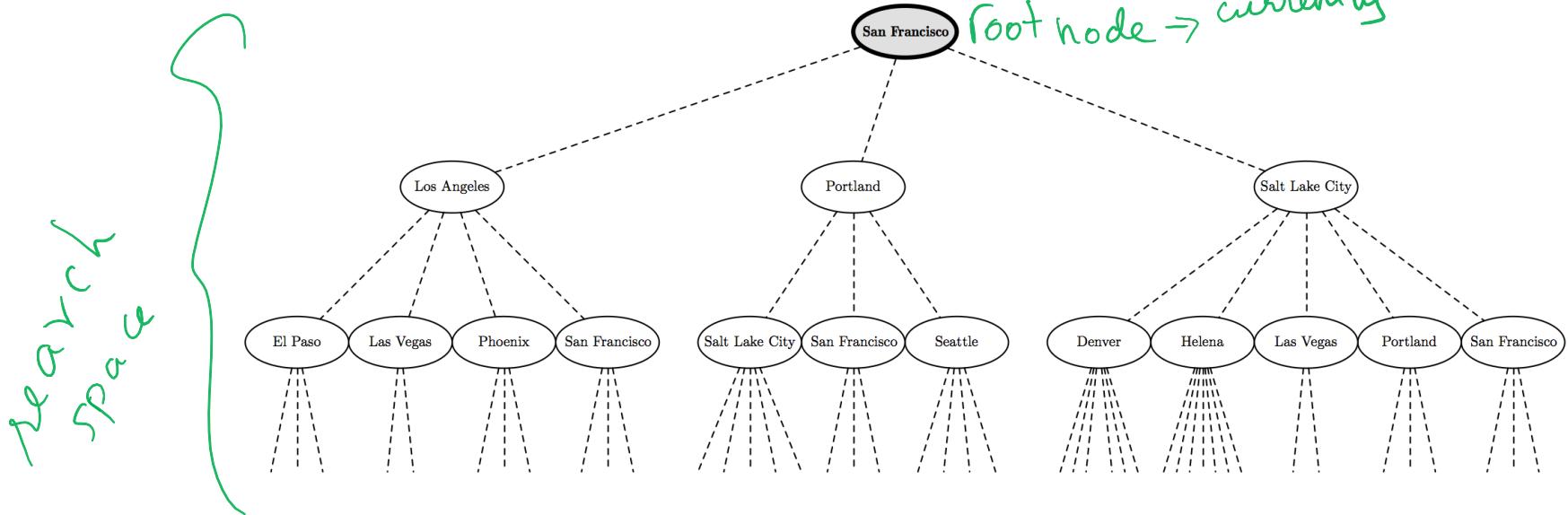
    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      frontier.add(neighbor)

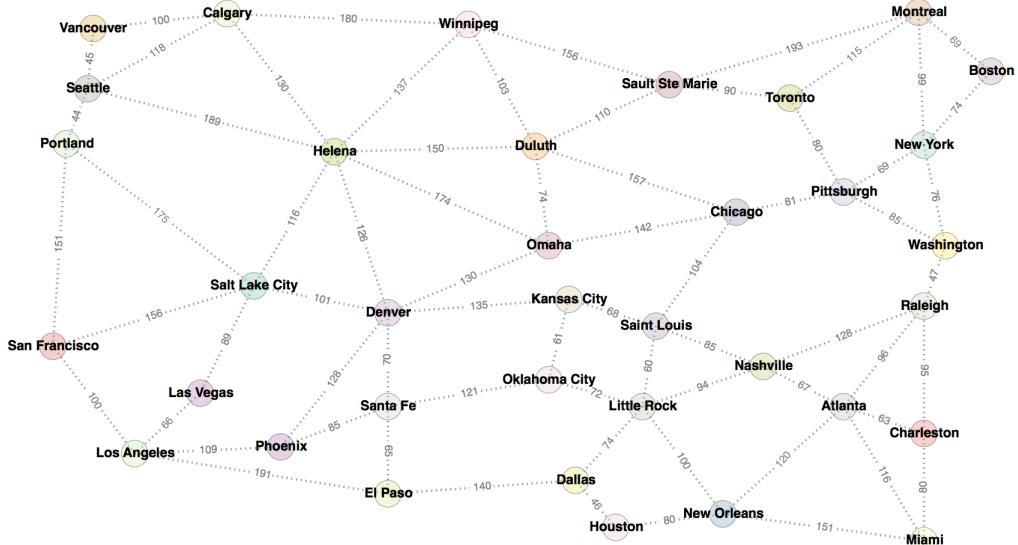
  return FAILURE
```

return FAILURE

return FAILURE
Root node \rightarrow currently in frontier



Examples of search agents



```
function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

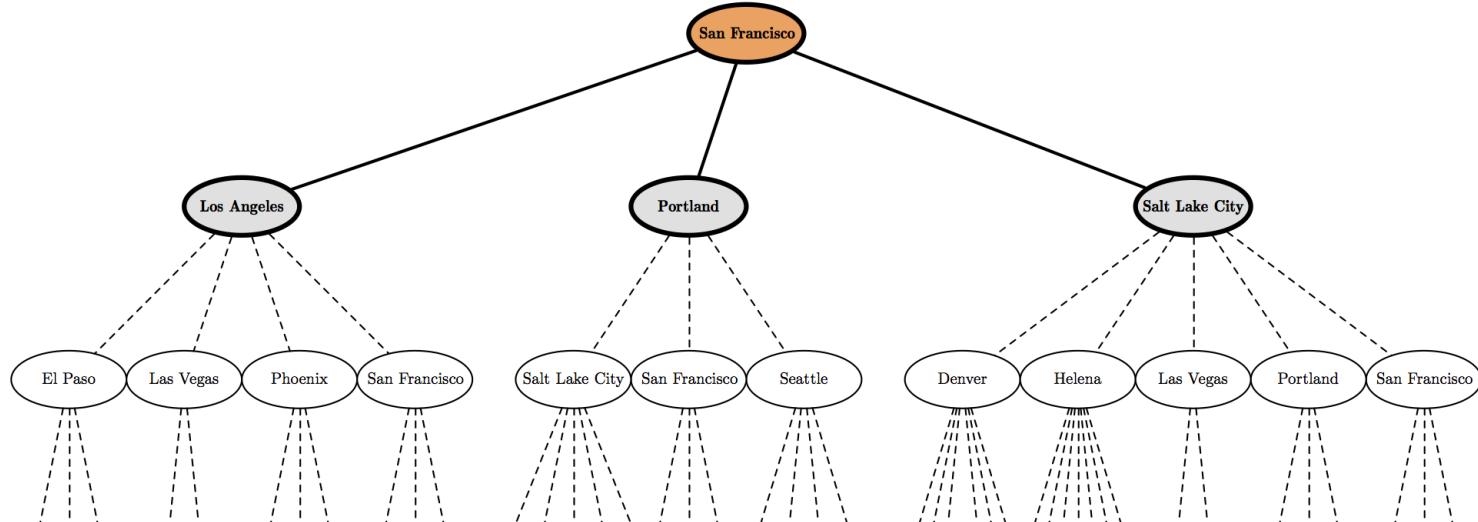
    initialize frontier with initialState

    while not frontier.isEmpty():
        state = frontier.remove()

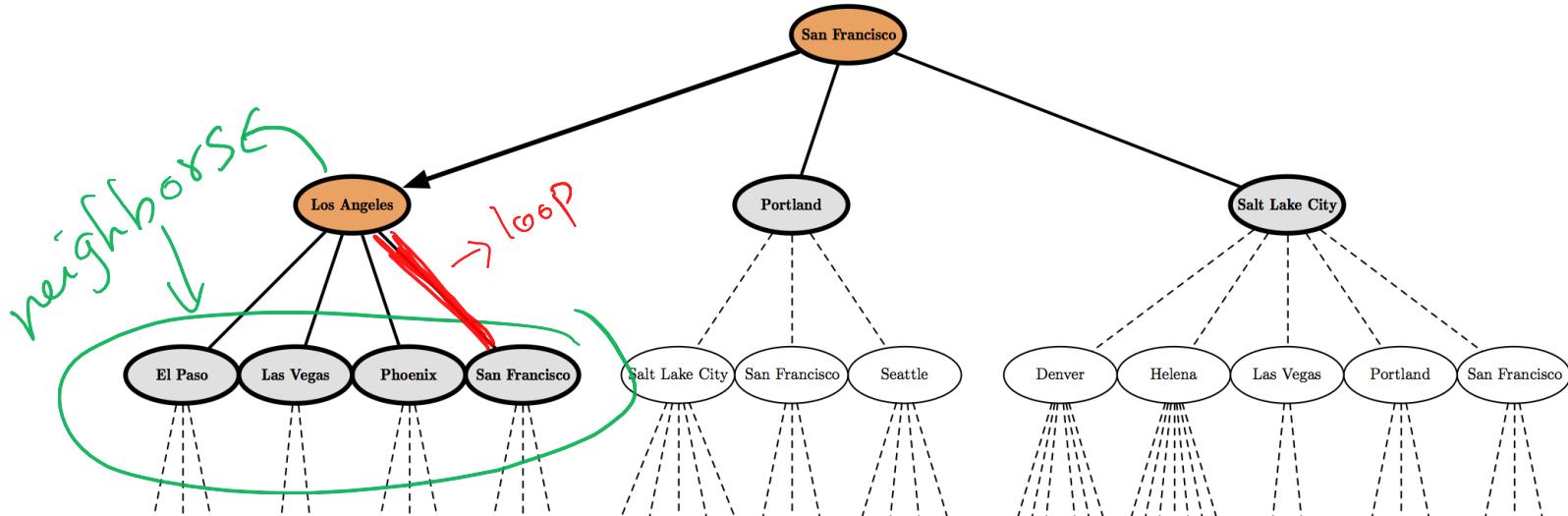
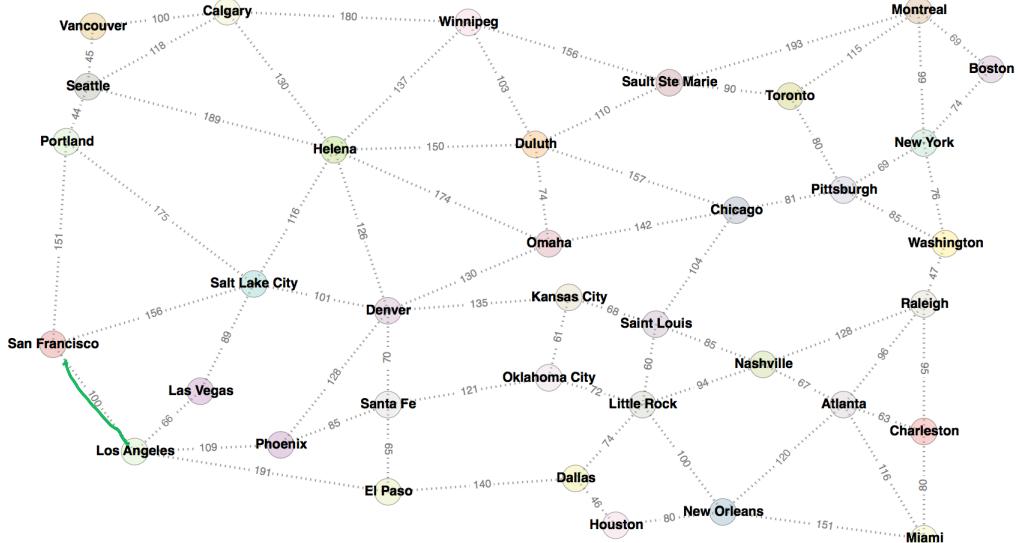
        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            frontier.add(neighbor)

    return FAILURE
```



Examples of search agents



```

function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    initialize frontier with initialState

    while not frontier.isEmpty():
        state = frontier.remove()

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            frontier.add(neighbor)

    return FAILURE
  
```

Graph search

How to handle repeated states?

Graph search

How to handle repeated states?

function GRAPH-SEARCH(*initialState*, *goalTest*)
returns **SUCCESS** or **FAILURE** :

initialize frontier **with** *initialState*
explored = Set.new()

while **not** frontier.isEmpty():
 state = frontier.remove()
 explored.add(state)

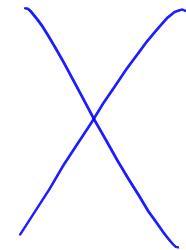
if *goalTest*(state):
 return **SUCCESS**(state)

for neighbor **in** state.neighbors():
 if neighbor **not** **in** frontier \cup explored:
 frontier.add(neighbor)

return **FAILURE**

Search strategies

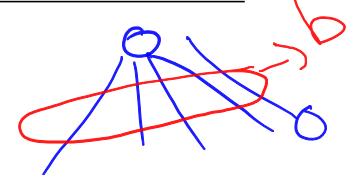
- A strategy is defined by picking the **order of node expansion**



Search strategies

- A strategy is defined by picking the **order of node expansion**
- Strategies are evaluated along the following dimensions:
 - **Completeness**
Does it always find a solution if one exists?
 - **Time complexity**
Number of nodes generated/expanded
 - **Space complexity**
Maximum number of nodes in memory
 - **Optimality**
Does it always find a least-cost solution?

Search strategies

- Time and space complexity are measured in terms of:
 - b : maximum branching factor of the search tree (actions per state).
 - d : depth of the solution
 - m : maximum depth of the state space (may be ∞) (also noted sometimes D).
-
- Two kinds of search: Uninformed and Informed.

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

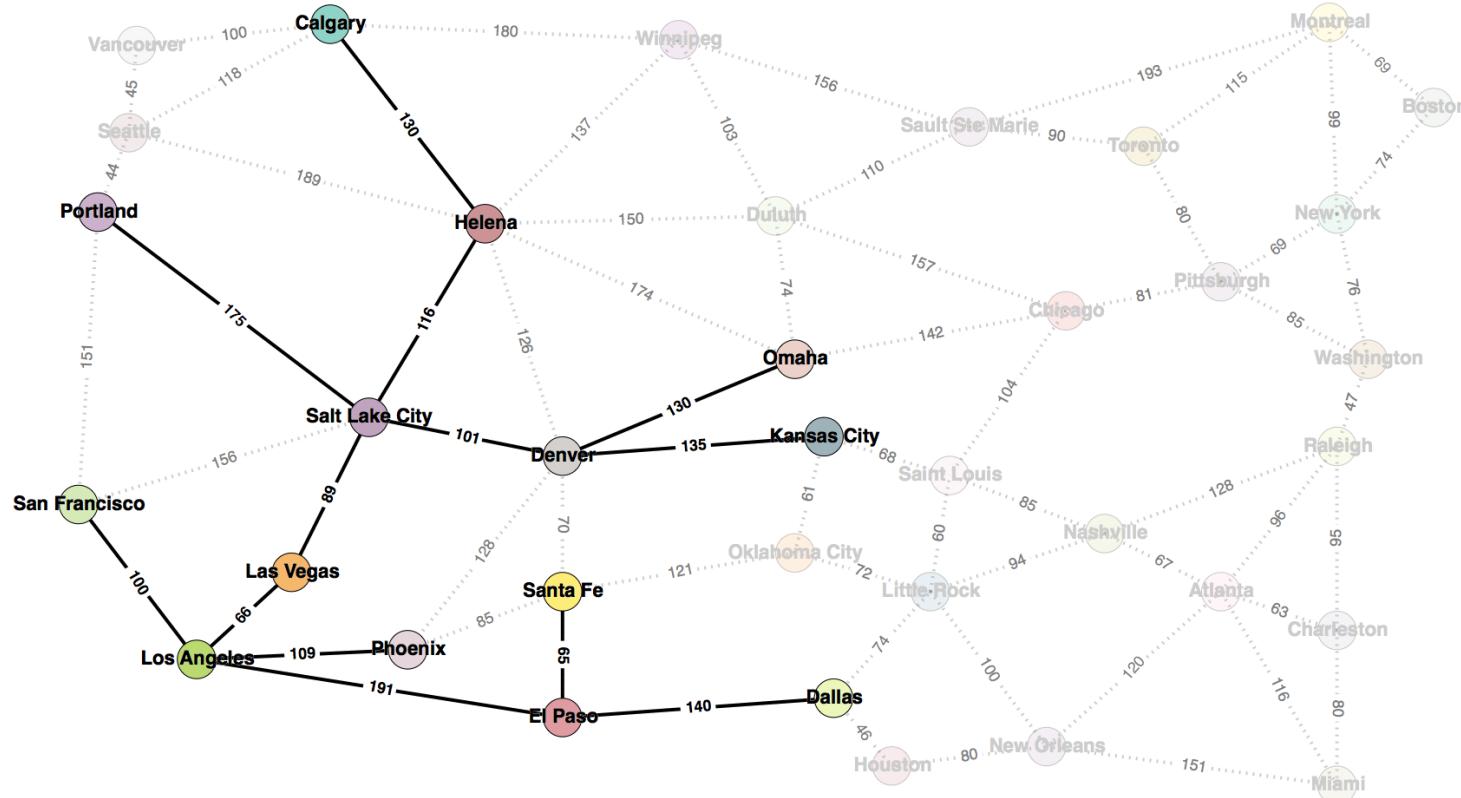
<http://aima.cs.berkeley.edu/>

Artificial Intelligence

L2.3.1

Search Agents

Uninformed search



Uninformed search

Use no domain knowledge!



Strategies:

1. Breadth-first search (BFS): Expand shallowest node

Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node



Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit

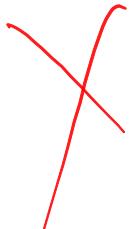


Uninformed search

Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit
4. Iterative-deepening search (IDS): DLS with increasing limit



Uninformed search

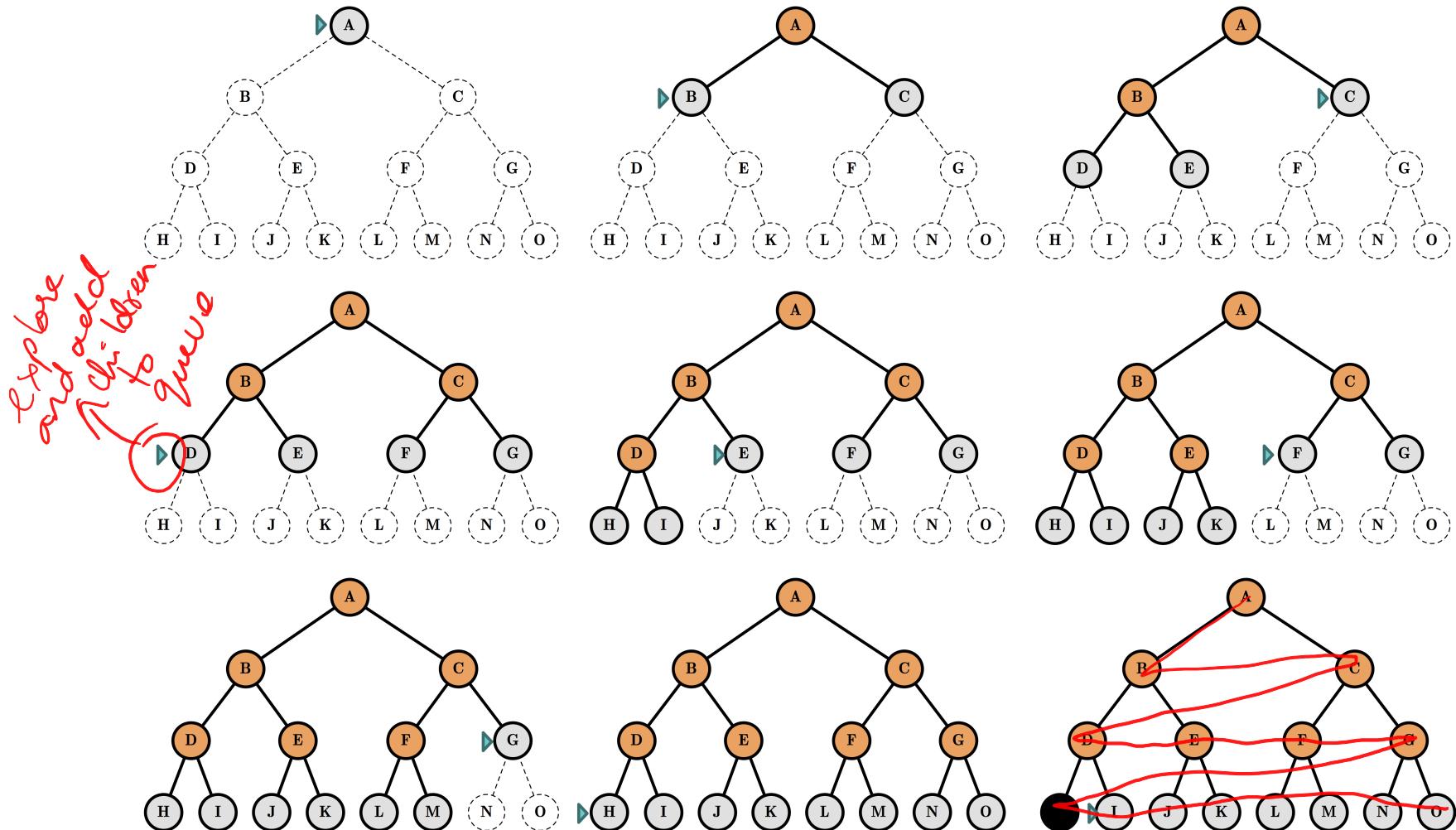
Use no domain knowledge!

Strategies:

1. Breadth-first search (BFS): Expand shallowest node
2. Depth-first search (DFS): Expand deepest node
3. Depth-limited search (DLS): Depth first with depth limit
4. Iterative-deepening search (IDS): DLS with increasing limit
5. Uniform-cost search (UCS): Expand least cost node

Breadth-first search (BFS)

BFS: Expand shallowest first.





BFS search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :
```

```
    frontier = Queue.new(initialState)
```

FIFO

```
    explored = Set.new()
```

```
    while not frontier.isEmpty():
```

```
        state = frontier.dequeue()
```

```
        explored.add(state)
```

```
        if goalTest(state):
```

```
            return SUCCESS(state)
```

```
        for neighbor in state.neighbors():
```

```
            if neighbor not in frontier ∪ explored:
```

```
                frontier.enqueue(neighbor)
```

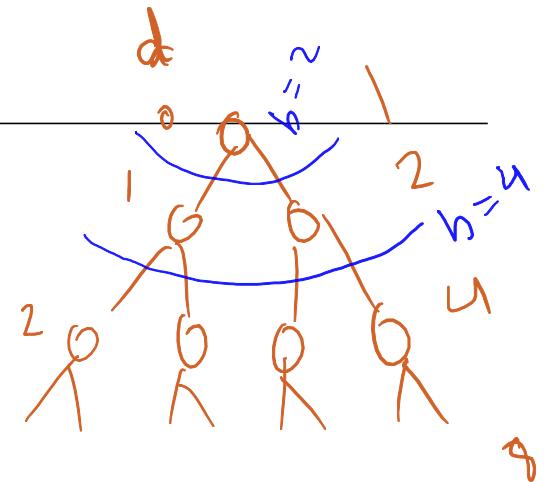
```
    return FAILURE
```

BFS Criteria

BFS criteria?

BFS

- **Complete** Yes (if b is finite)
- **Time** $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$
- **Space** $O(b^d)$
Note: If the *goal test* is applied at expansion rather than generation then $O(b^{d+1})$
- **Optimal** Yes (if cost = 1 per step).
- **implementation:** fringe: FIFO (Queue)

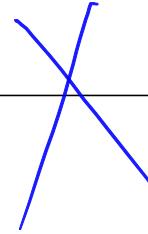


Question: If time and space complexities are exponential, why use BFS?

BFS

How bad is BFS?

BFS



How bad is BFS?

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

Time and Memory requirements for breadth-first search for a branching factor $b=10$; 1 million nodes per second; 1,000 bytes per node.

BFS

How bad is BFS?

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

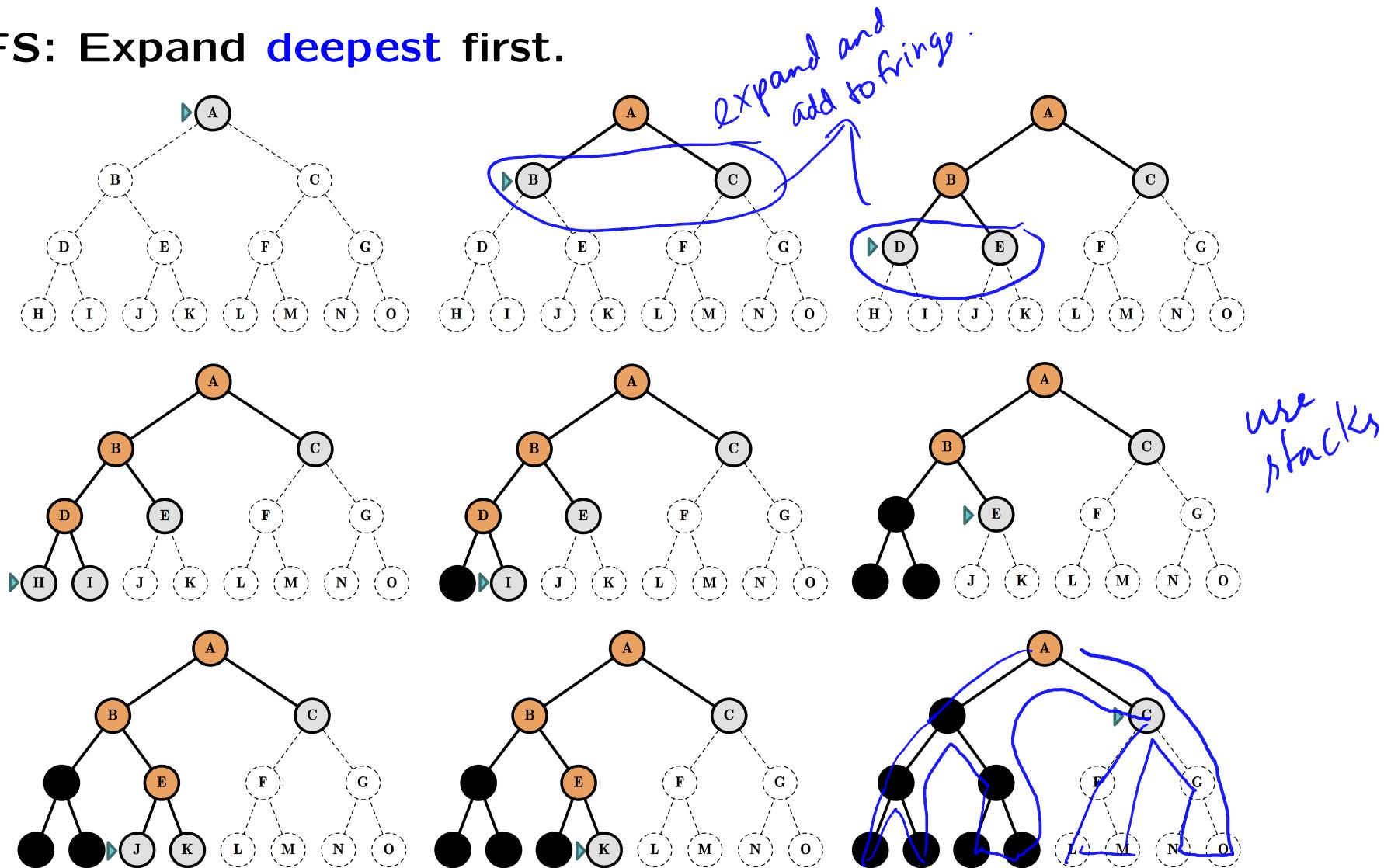
Time and Memory requirements for breadth-first search for a branching factor $b=10$; 1 million nodes per second; 1,000 bytes per node.

Memory requirement + exponential time complexity are the biggest handicaps of BFS!

DFS

L.2.3.2

DFS: Expand deepest first.





DFS search

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
```

returns SUCCESS or FAILURE :

Stack [LIFO]

```
frontier = Stack.new(initialState)  
explored = Set.new()
```

```
while not frontier.isEmpty():  
    state = frontier.pop()  
    explored.add(state)
```

```
    if goalTest(state):  
        return SUCCESS(state)
```

```
    for neighbor in state.neighbors():  
        if neighbor not in frontier ∪ explored:  
            frontier.push(neighbor)
```

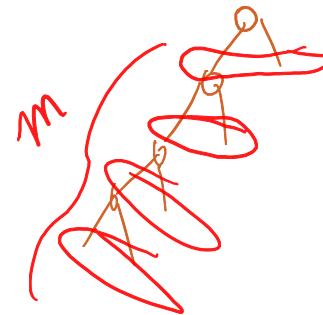
```
return FAILURE
```

DFS

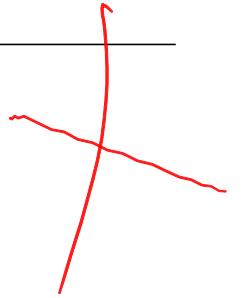
DFS criteria?

DFS

- **Complete** No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path.
⇒ complete in finite spaces
- **Time** $O(b^m)$: $1 + b + b^2 + b^3 + \dots + b^m = O(b^m)$
bad if m is much larger than d
but if solutions are dense, may be much faster than BFS.
- **Space** $O(bm)$ linear space complexity! (needs to store only a single path from the root to a leaf node, **along with the remaining unexpanded sibling nodes for each node on the path, hence the m factor.**)
- **Optimal** No
- **Implementation:** fringe: LIFO (Stack)



DFS



How bad is DFS?

Recall for BFS...

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

Depth = 16.

We go down from 10 exabytes in BFS to ... in DFS?

DFS

How bad is DFS?

Recall for BFS...

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

Depth = 16.

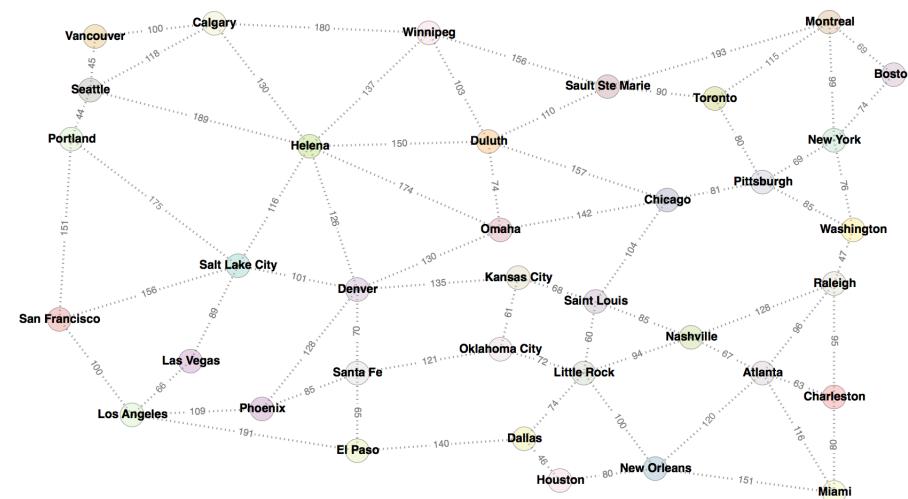
We go down from 10 exabytes in BFS to 156 kilobytes in DFS!

Depth-limited search

- DFS with depth limit l (nodes at level l has no successors).
→ avoid loops
- Select some limit L in depth to explore with DFS
- Iterative deepening: increasing the limit l

Depth-limited search

- If we know some knowledge about the problem, maybe we don't need to go to a full depth.



Idea: any city can be reached from another city in at most L steps with $L < 36$.

Iterative Deepening

- Combines the benefits of BFS and DFS.
- Idea: Iteratively increase the search limit until the depth of the shallowest solution d is reached.
- Applies DLS with increasing limits.
- The algorithm will stop if a solution is found or if DLS returns a failure (no solution).
- Because most of the nodes are on the bottom of the search tree, it not a big waste to iteratively re-generate the top
- Let's take an example with a depth limit between 0 and 3.



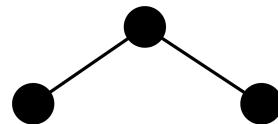
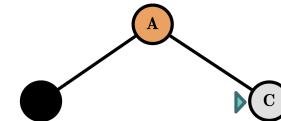
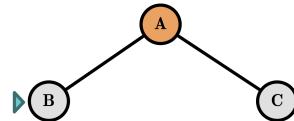
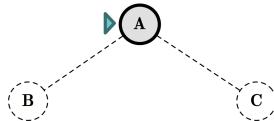
Iterative Deepening

Limit = 0



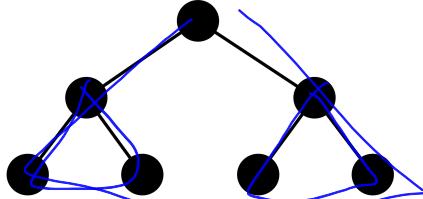
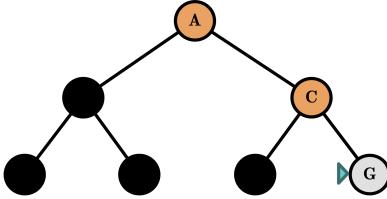
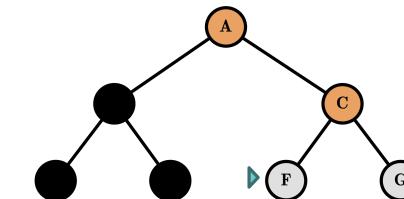
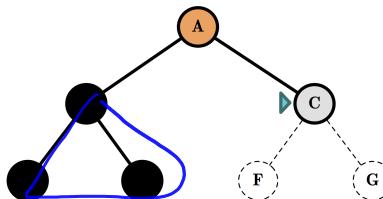
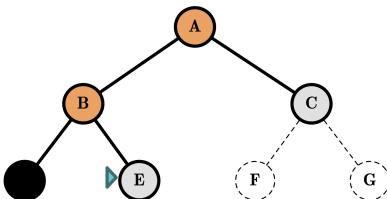
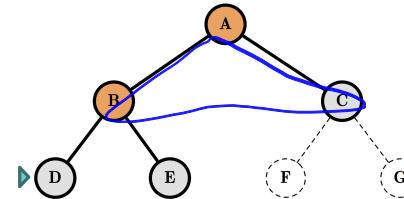
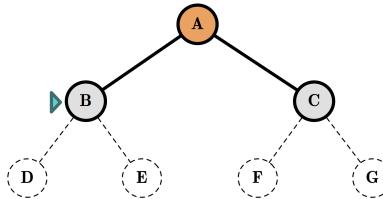
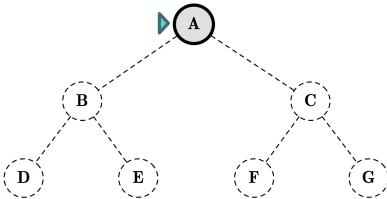
Iterative Deepening

Limit = 1



Iterative Deepening

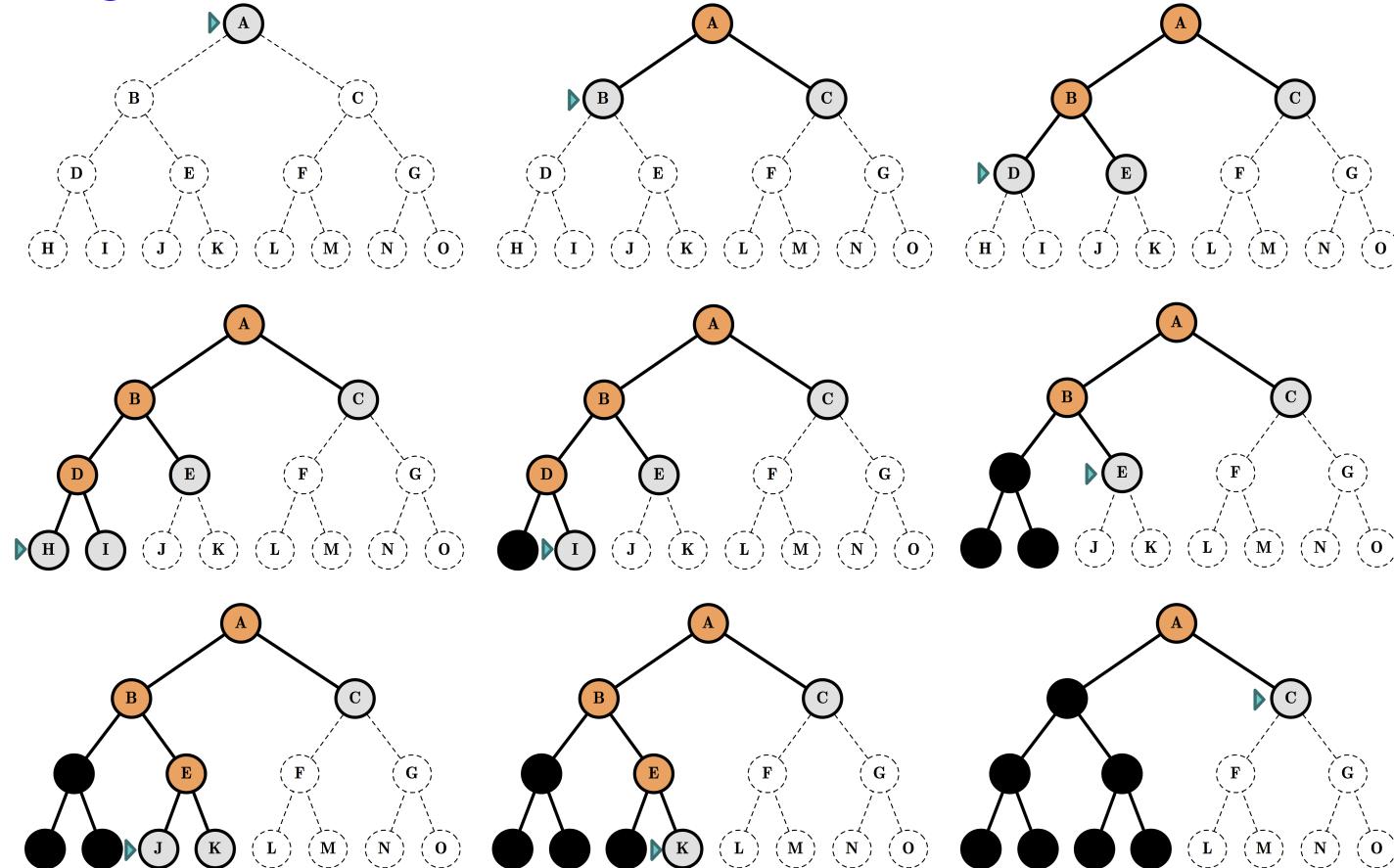
Limit = 2



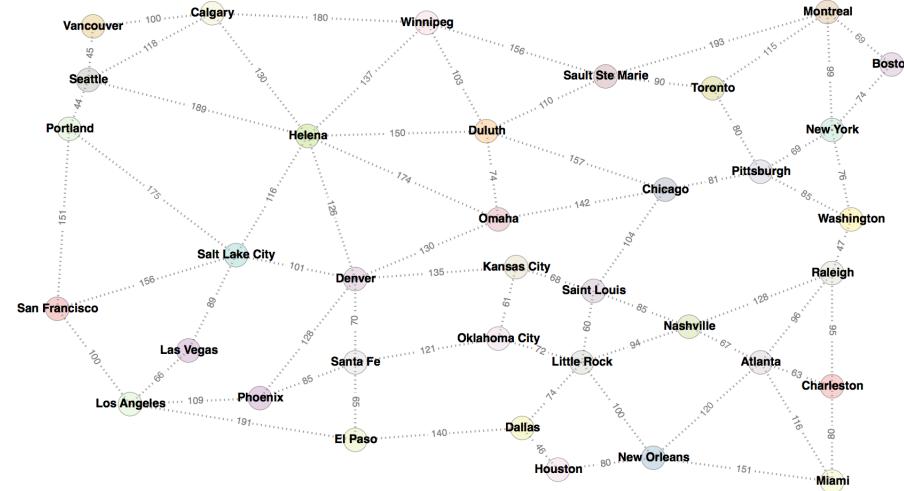
Iterative Deepening

(Project)

Limit = 3

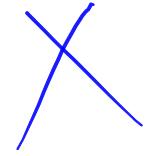
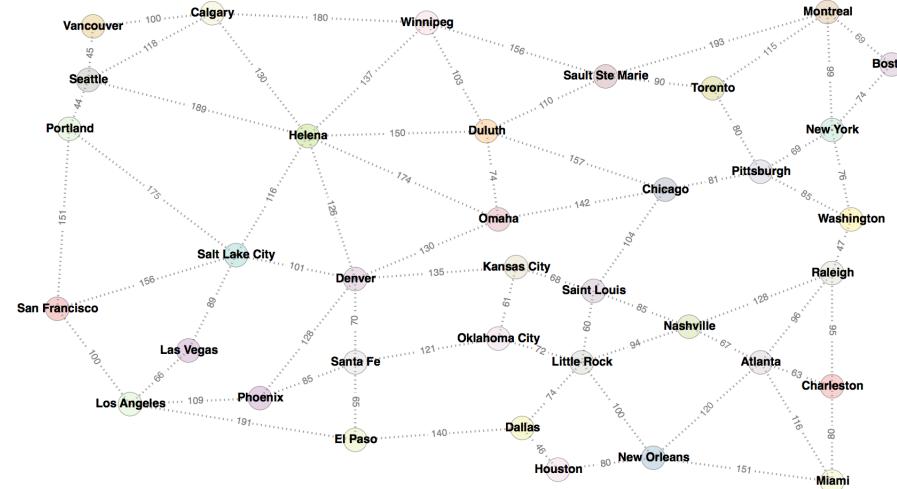


Uniform-cost search



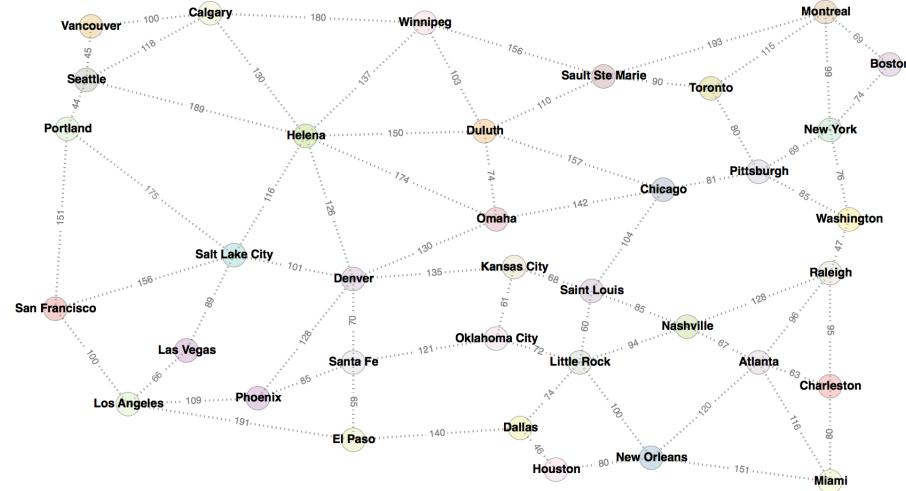
- The arcs in the search graph may have weights (different cost attached). How to leverage this information?

Uniform-cost search



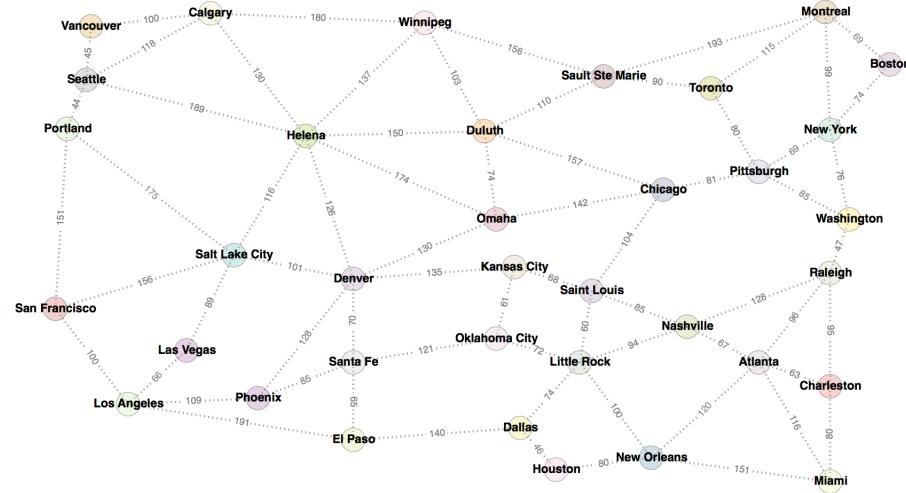
- The arcs in the search graph may have weights (different cost attached). How to leverage this information?
 - BFS will find the shortest path which may be costly.
 - We want the **cheapest** not shallowest solution.

Uniform-cost search



- The arcs in the search graph may have weights (different cost attached). How to leverage this information?
- BFS will find the shortest path which may be costly.
- We want the **cheapest** not shallowest solution.
- **Modify BFS:** Prioritize by cost not depth → **Expand node n with the lowest path cost $g(n)$**

Uniform-cost search



- The arcs in the search graph may have weights (different cost attached). How to leverage this information?
- BFS will find the shortest path which may be costly.
- We want the **cheapest** not shallowest solution.
- **Modify BFS:** Prioritize by cost not depth → **Expand node n with the lowest path cost $g(n)$**
- Explores increasing costs.



UCS algorithm

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n)$  */
```

```
frontier = Heap.new(initialState)
explored = Set.new()
```

```
while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)

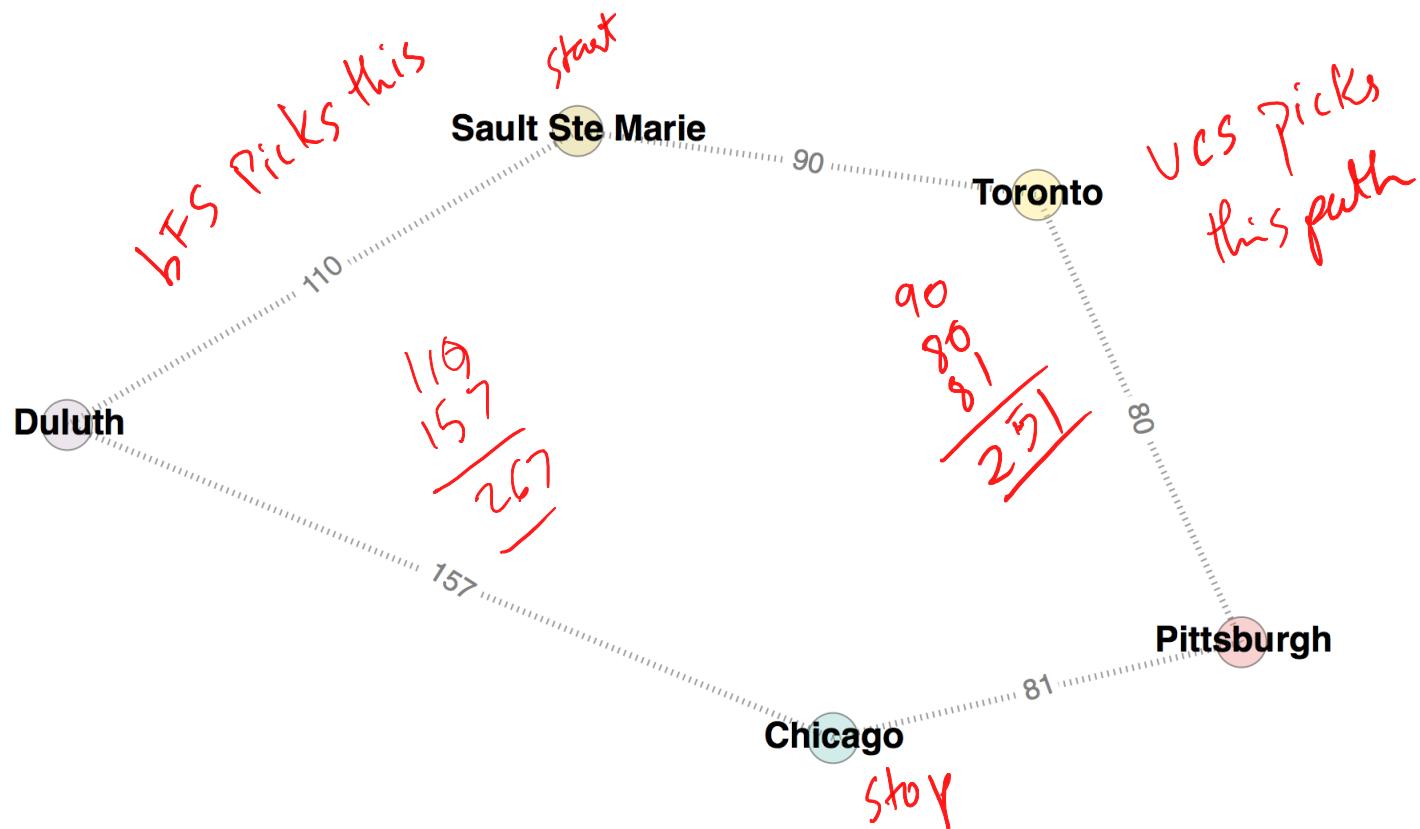
    if goalTest(state):
        return SUCCESS(state)
```

```
for neighbor in state.neighbors():
    if neighbor not in frontier ∪ explored:
        frontier.insert(neighbor)
    else if neighbor in frontier:
        frontier.decreaseKey(neighbor)

return FAILURE
```

heap
↓
Priority Queue

Uniform-cost search



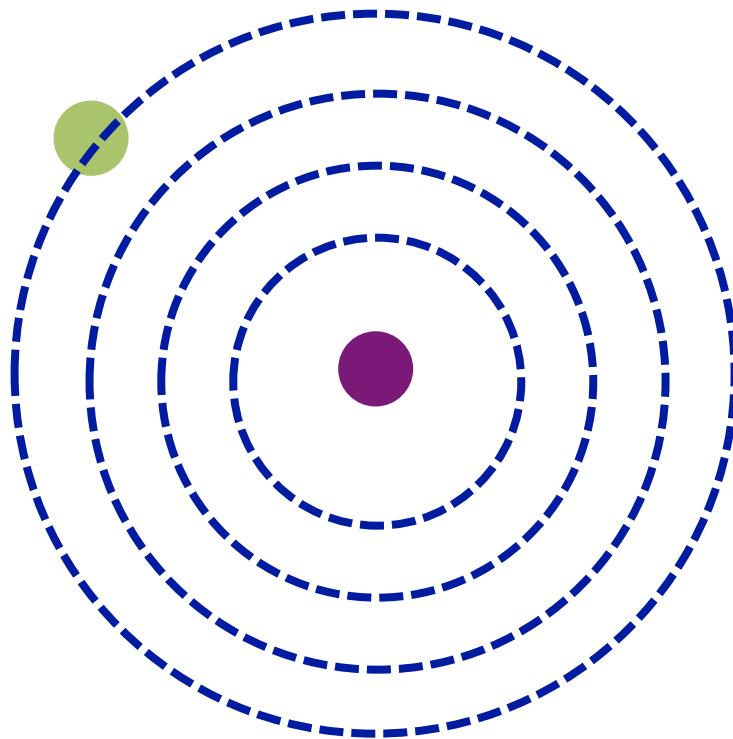
Go from Chicago to Sault Ste Marie. Using BFS, we would find Chicago-Duluth-Sault Ste Marie. However, using UCS, we would find Chicago-Pittsburgh-Toronto-Sault Ste Marie, which is actually the shortest path!

Uniform-cost search

- **Complete** Yes, if solution has a finite cost.
- **Time**
 - Suppose C^* : cost of the optimal solution
 - Every action costs at least ϵ (bound on the cost)
 - The effective depth is roughly C^*/ϵ (how deep the *cheapest* solution could be).
 - $O(b^{C^*/\epsilon})$
- **Space** # of nodes with $g \leq$ cost of optimal solution, $O(b^{C^*/\epsilon})$
- **Optimal** Yes
- **Implementation**: fringe = queue ordered by path cost $g(n)$, lowest first = Heap!

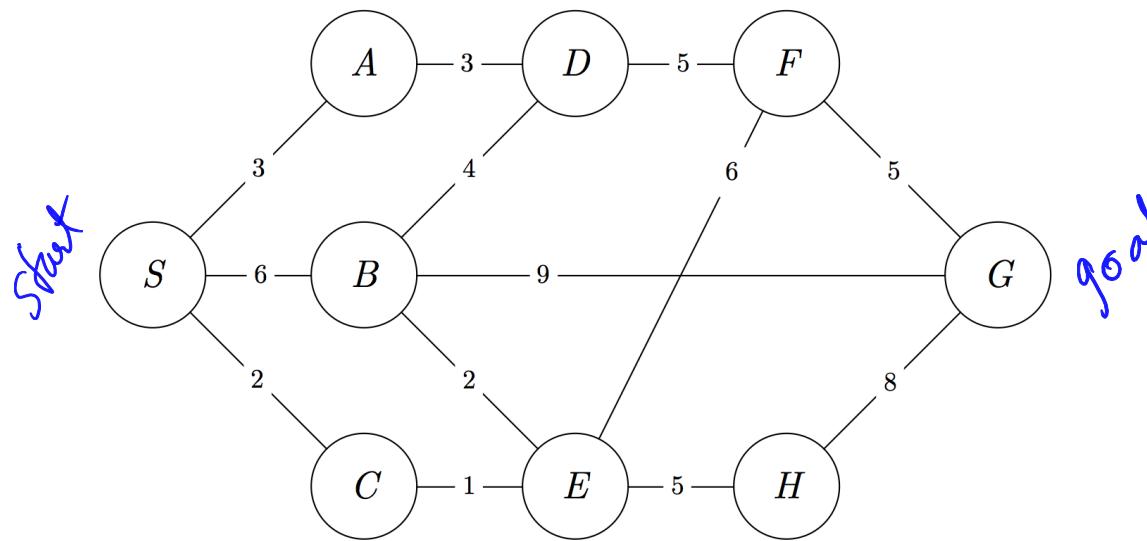
Uniform-cost search

While complete and optimal, UCS explores the space in every direction because no information is provided about the goal!



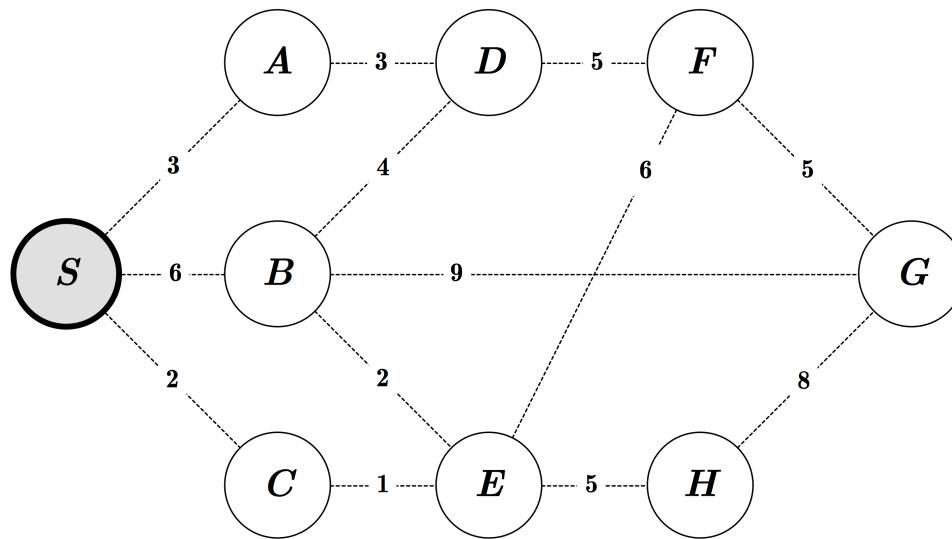
Exercise

L2.4

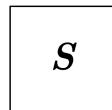


Question: What is the **order of visits of the nodes** and the **path** returned by BFS, DFS and UCS?

Exercise: BFS

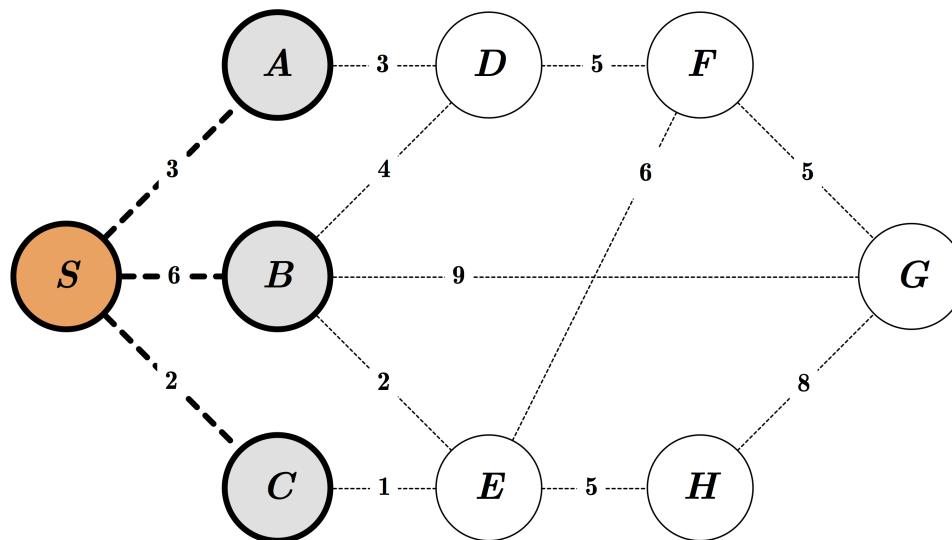


Queue:



Order of Visit:

Exercise: BFS



degree ↗

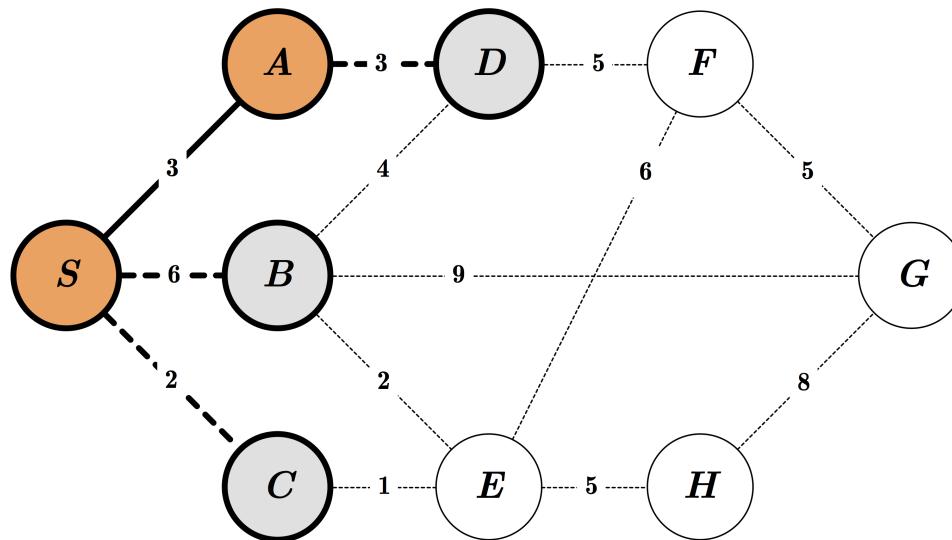
Queue: enque

S	A	B	C
---	---	---	---

Order of Visit:

S

Exercise: BFS



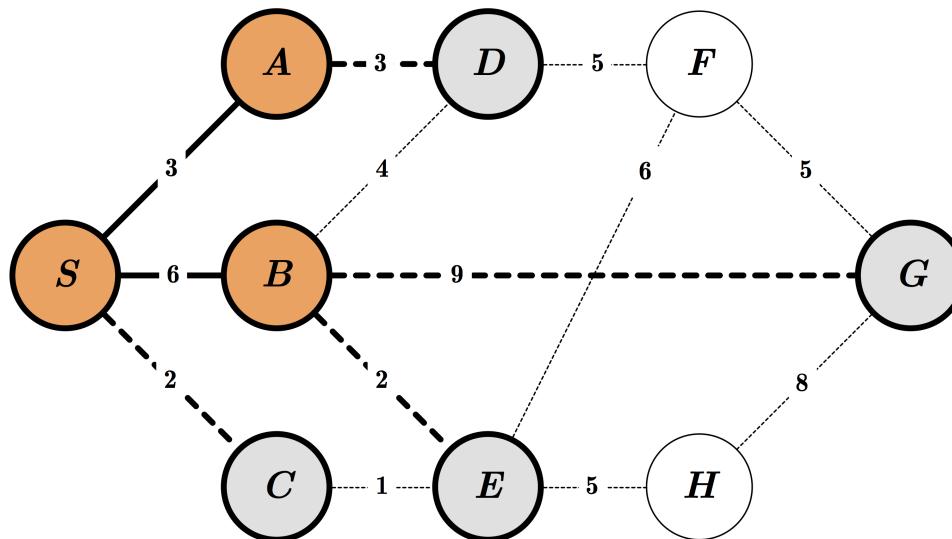
Queue:

S	A	B	C	D
---	---	---	---	---

Order of Visit:

S A

Exercise: BFS



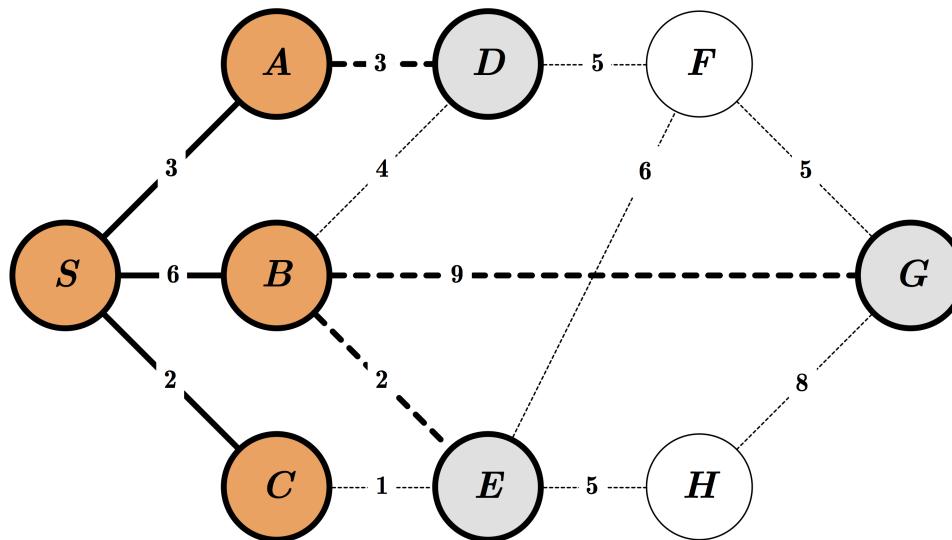
Queue:

<i>S</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>G</i>
----------	----------	----------	----------	----------	----------	----------

Order of Visit:

S A B

Exercise: BFS



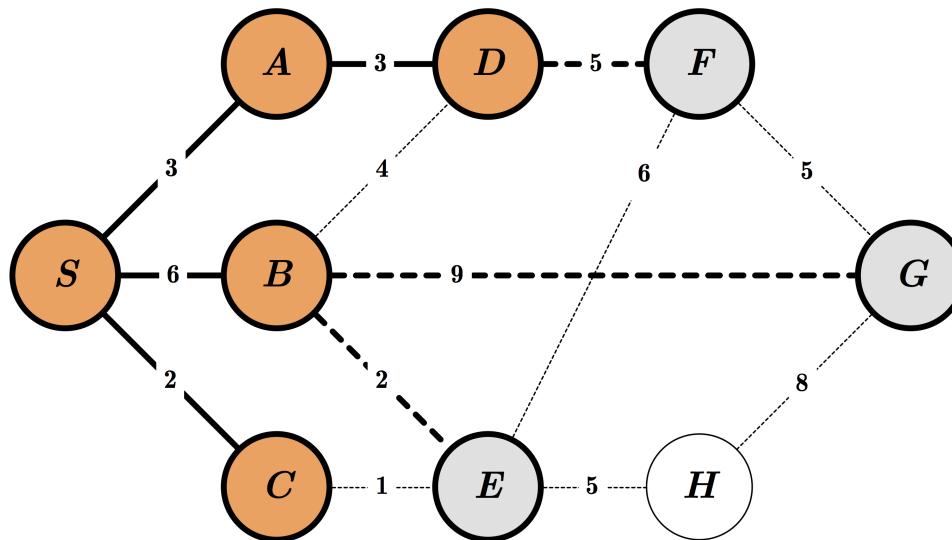
Queue:

S	A	B	C	D	E	G
---	---	---	---	---	---	---

Order of Visit:

S A B C

Exercise: BFS



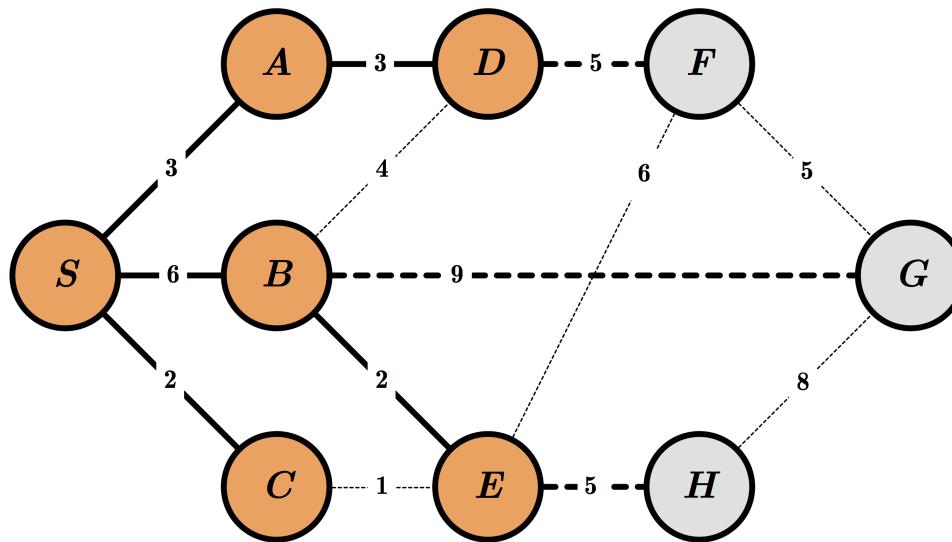
Queue:

S	A	B	C	D	E	G	F
---	---	---	---	---	---	---	---

Order of Visit:

S A B C D

Exercise: BFS



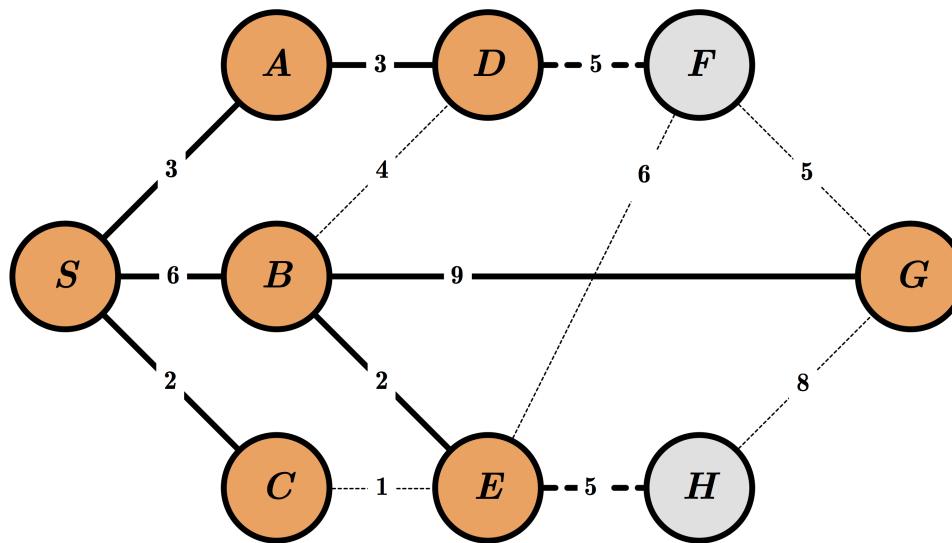
Queue:

S	A	B	C	D	E	G	F	H
---	---	---	---	---	---	---	---	---

Order of Visit:

S A B C D E

Exercise: BFS



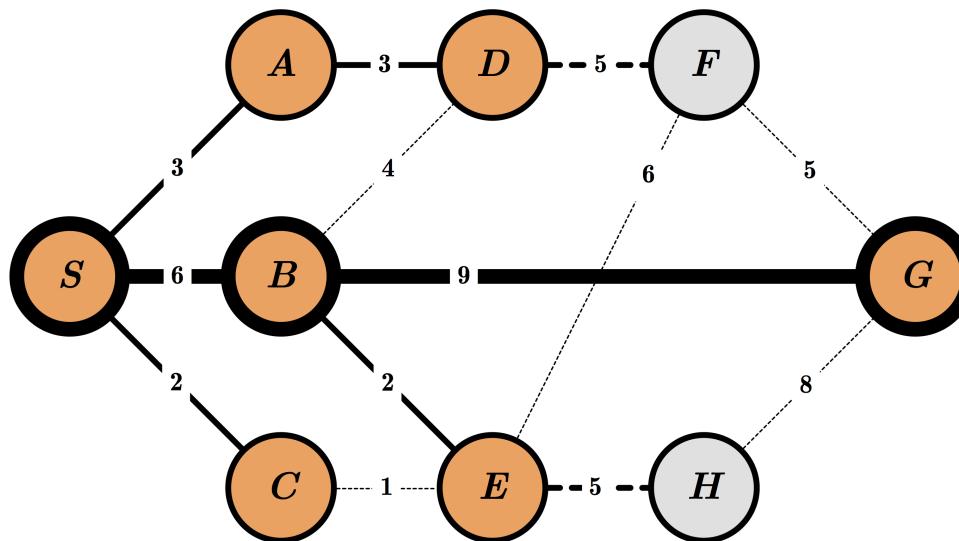
Queue:

S	A	B	C	D	E	G	F	H
---	---	---	---	---	---	----------	---	---

Order of Visit:

S A B C D E G

Exercise: BFS



Queue:

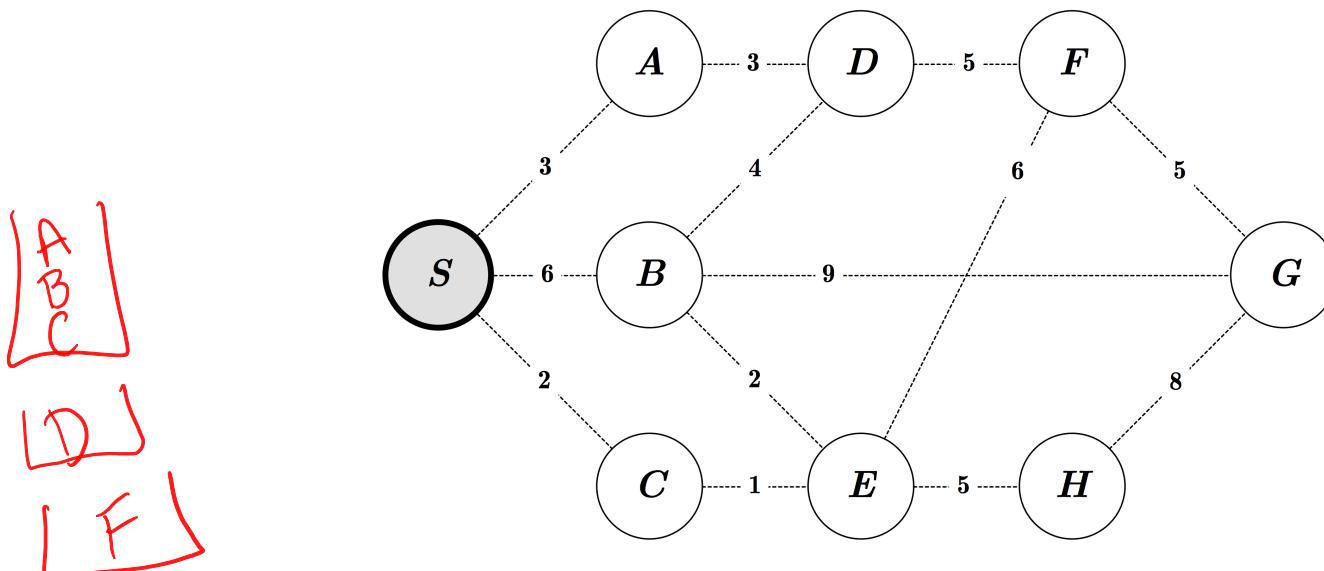
S	A	B	C	D	E	G	F	H
---	---	---	---	---	---	---	---	---

Order of Visit:

S A B C D E G



Exercise: DFS



[A
B
C]
[D]
[F]

[E
G]

[H]

Stack:

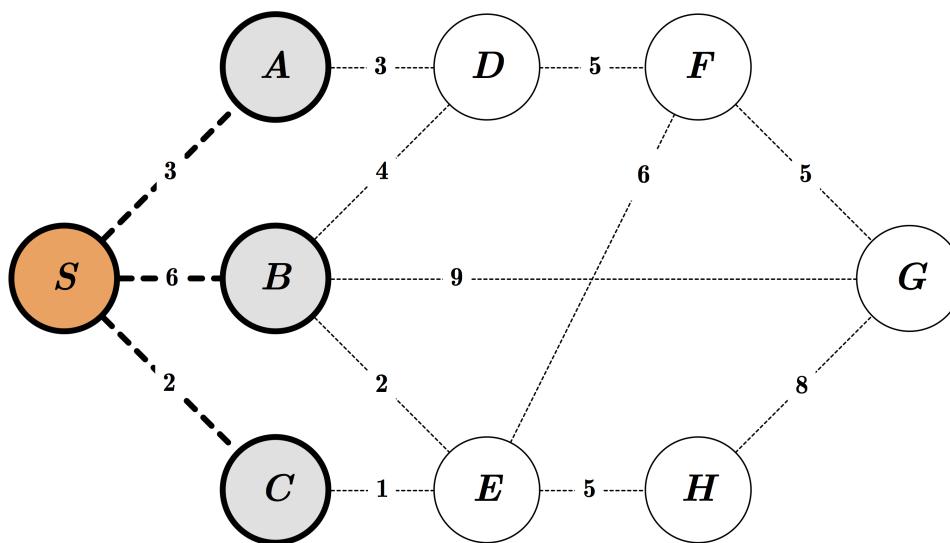
S

CBADFGEH

Order of Visit:

SADFEHGB

Exercise: DFS



Stack:

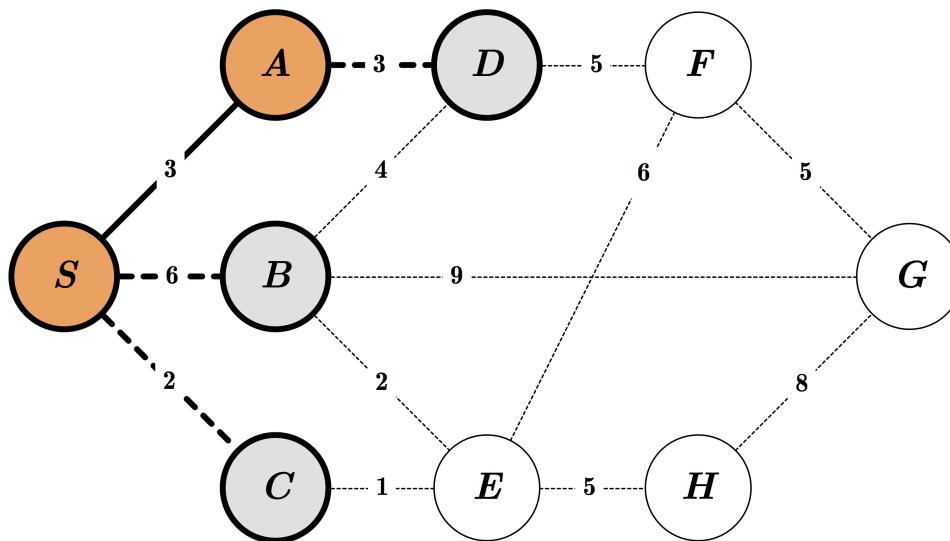
<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>
----------	----------	----------	----------

Order of Visit:

S

Push in
reverse lexicographic order
in order while popping
to get alphabetical order

Exercise: DFS



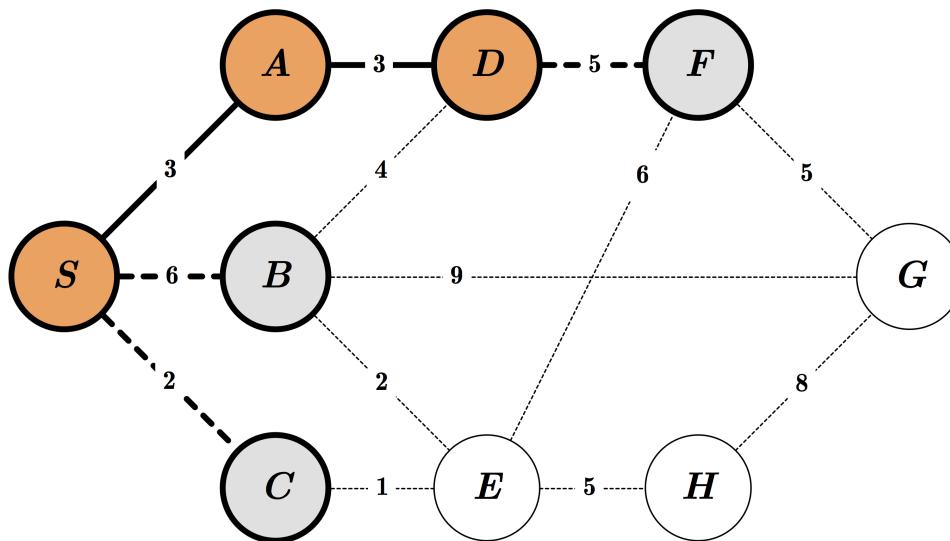
Stack:

<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>D</i>
----------	----------	----------	----------	----------

Order of Visit:

S A

Exercise: DFS



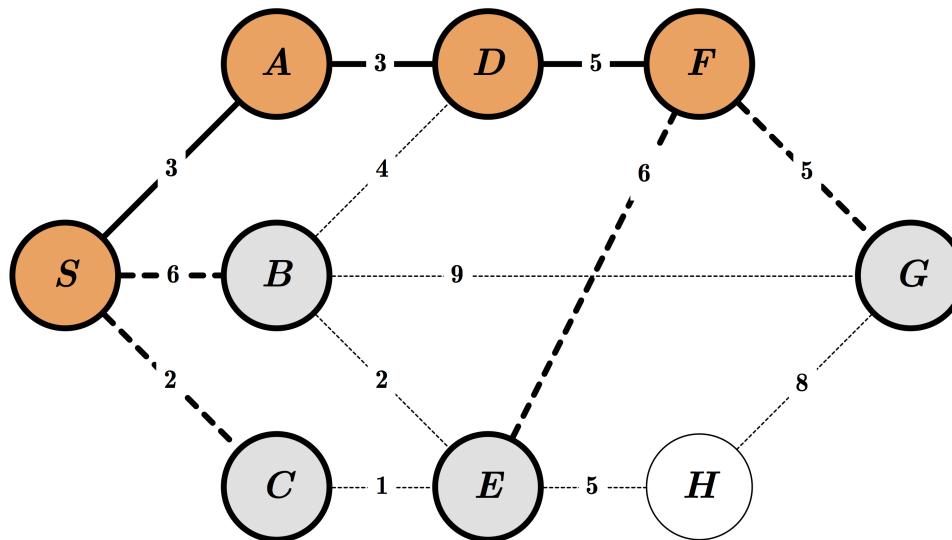
Stack:

S	C	B	A	D	F
---	---	---	---	---	---

Order of Visit:

S A D

Exercise: DFS



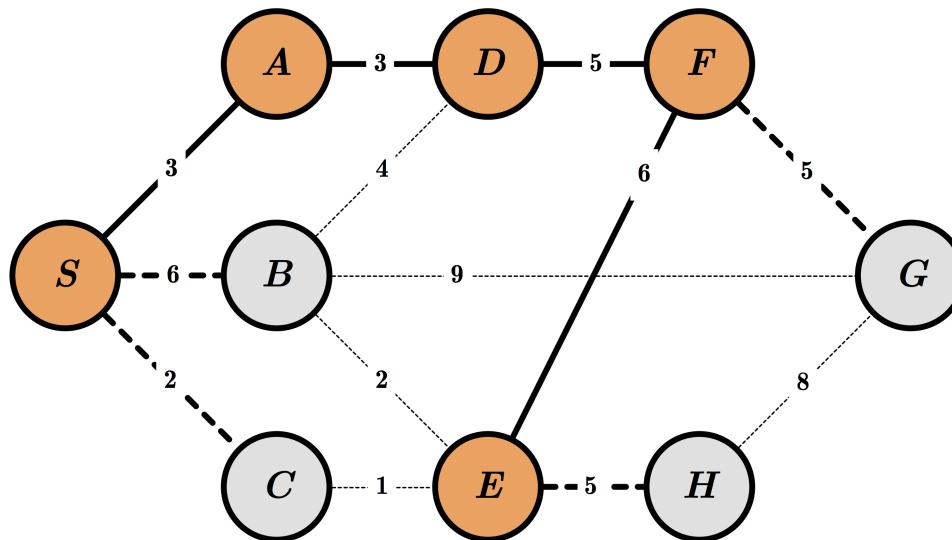
Stack:

S	C	B	A	D	F	G	E
---	---	---	---	---	---	---	---

Order of Visit:

S A D F

Exercise: DFS



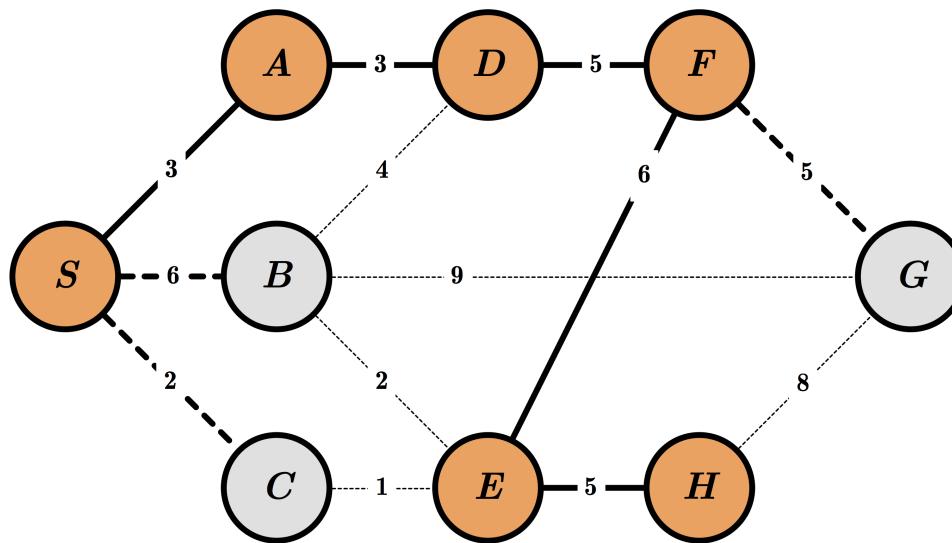
Stack:

<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>D</i>	<i>F</i>	<i>G</i>	<i>E</i>	<i>H</i>
----------	----------	----------	----------	----------	----------	----------	----------	----------

Order of Visit:

S A D F E

Exercise: DFS



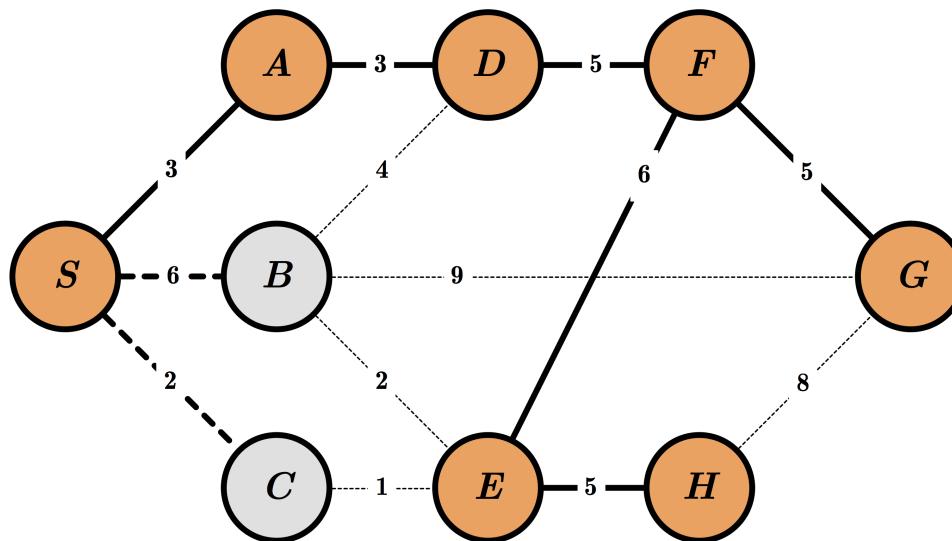
Stack:

<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>D</i>	<i>F</i>	<i>G</i>	<i>E</i>	<i>H</i>
----------	----------	----------	----------	----------	----------	----------	----------	----------

Order of Visit:

S A D F E H

Exercise: DFS



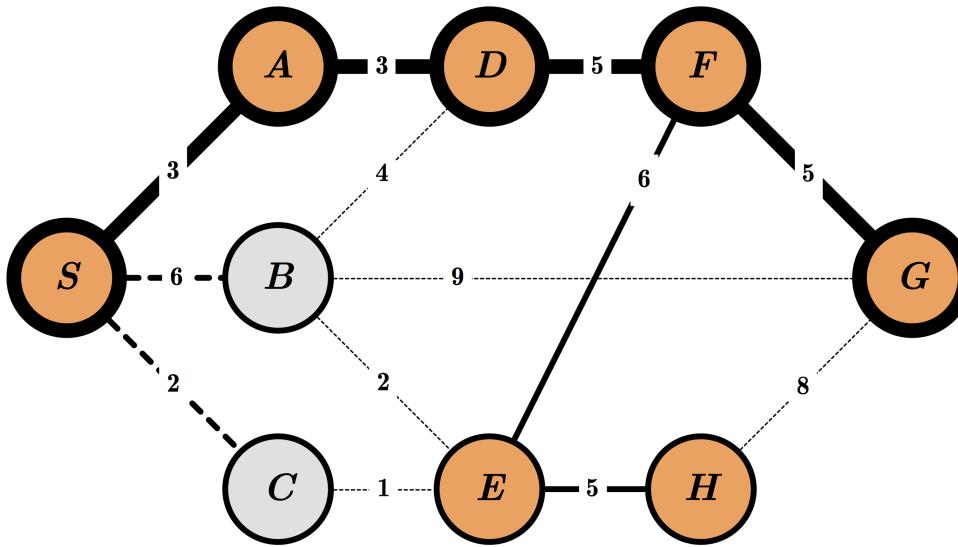
Stack:

<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>D</i>	<i>F</i>	<i>G</i>	<i>E</i>	<i>H</i>
----------	----------	----------	----------	----------	----------	----------	----------	----------

Order of Visit:

S A D F E H G

Exercise: DFS



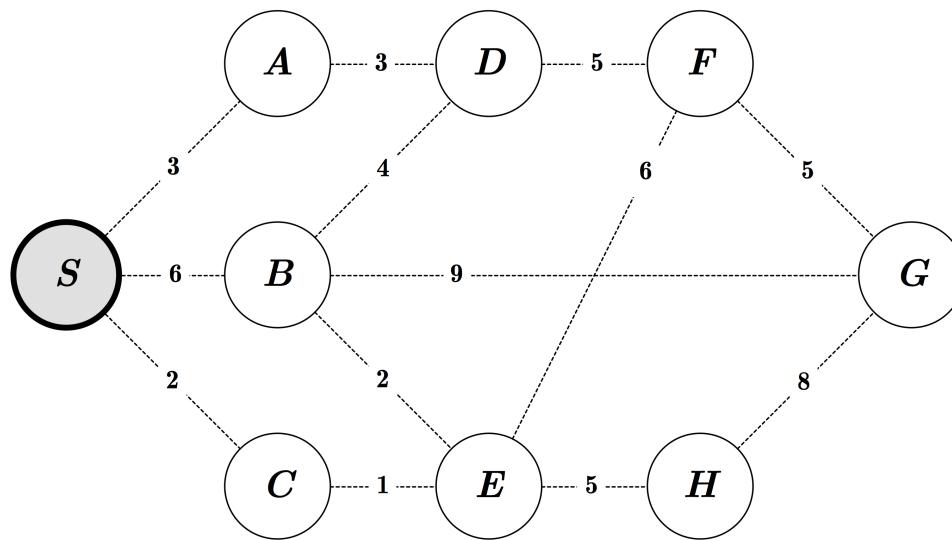
Stack:

<i>S</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>D</i>	<i>F</i>	<i>G</i>	<i>E</i>	<i>H</i>
----------	----------	----------	----------	----------	----------	----------	----------	----------

Order of Visit:

S A D F E H G

Exercise: UCS



Priority Queue:

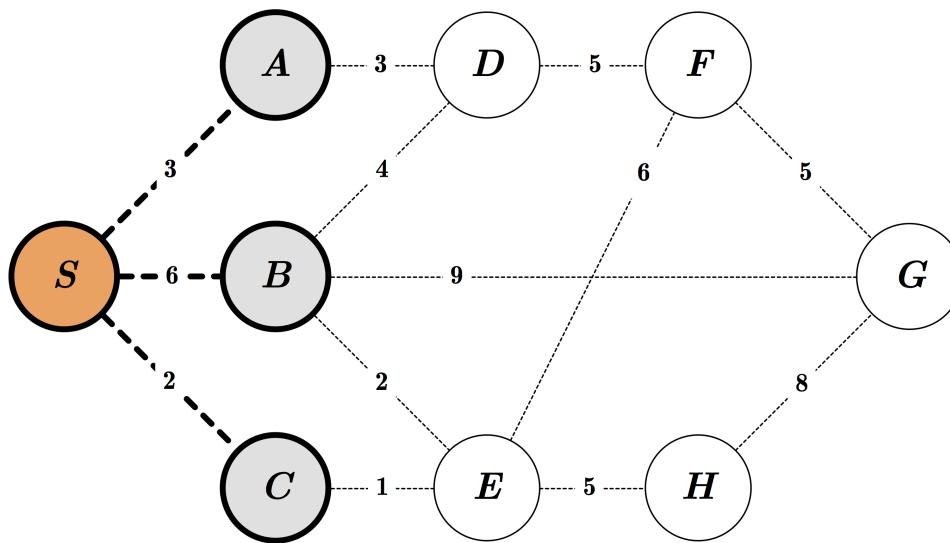
S_0

CAE β DHFG

Order of Visit:

SCAE β DHFG

Exercise: UCS



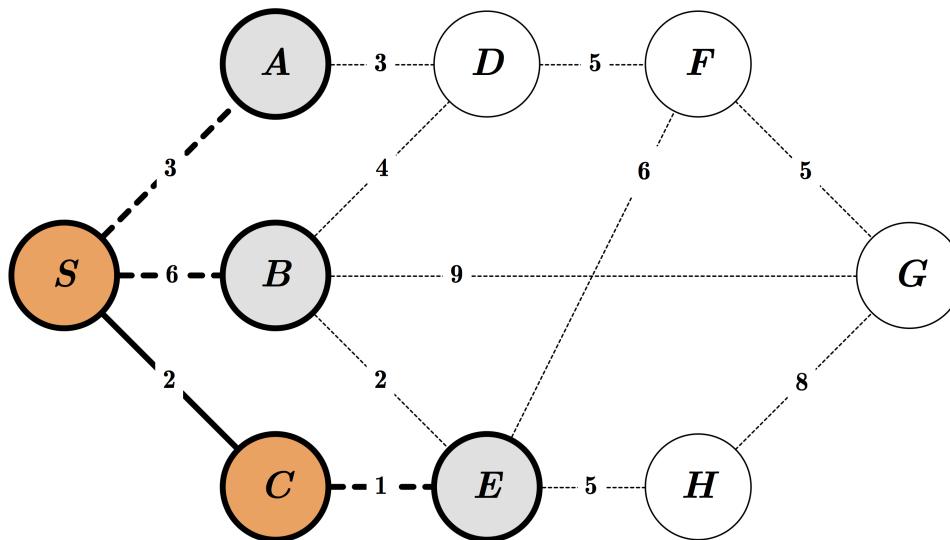
Priority Queue:

$S \ 0$	$C \ 2$	$A \ 3$	$B \ 6$
---------	---------	---------	---------

Order of Visit:

S

Exercise: UCS



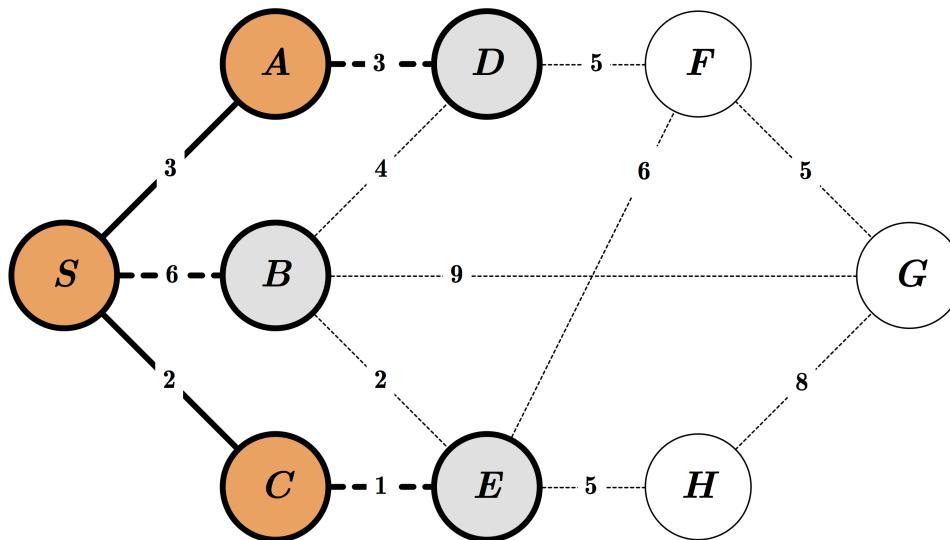
Priority Queue:

S_0	C_2	A_3	E_3	B_6
-------	-------	-------	-------	-------

Order of Visit:

$S \quad C$

Exercise: UCS



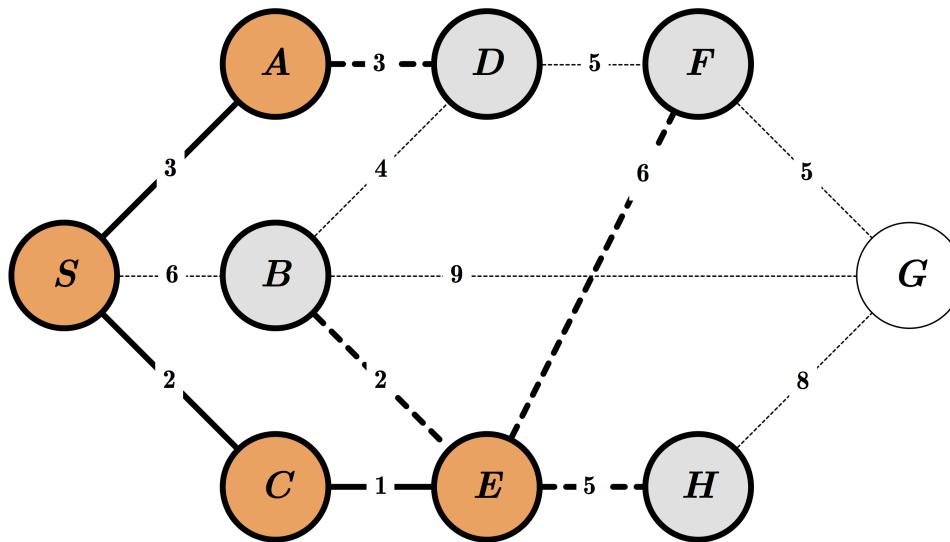
Priority Queue:

S_0	C_2	A_3	E_3	B_6	D_6
-------	-------	-------	-------	-------	-------

Order of Visit:

$S \quad C \quad A$

Exercise: UCS



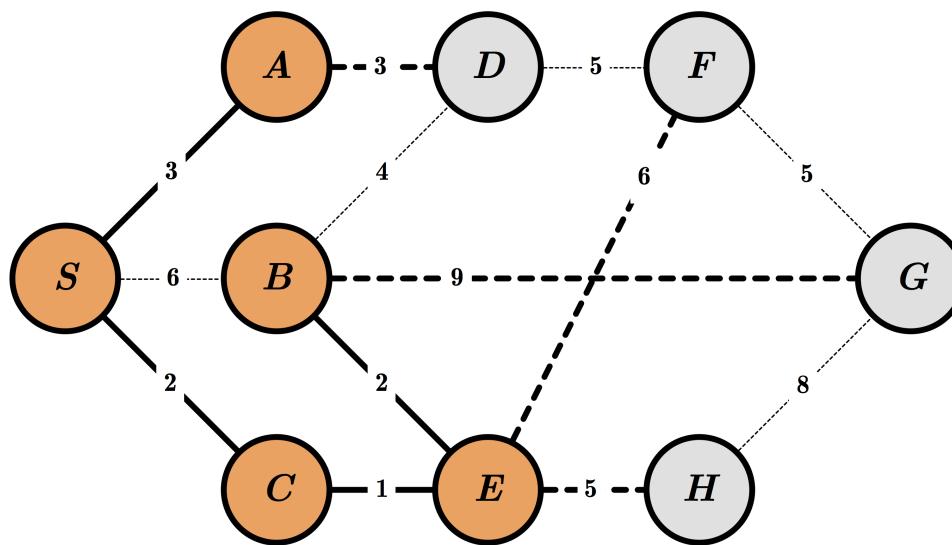
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9
-------	-------	-------	-------	-------	-------	-------	-------

Order of Visit:

$S \quad C \quad A \quad E$

Exercise: UCS



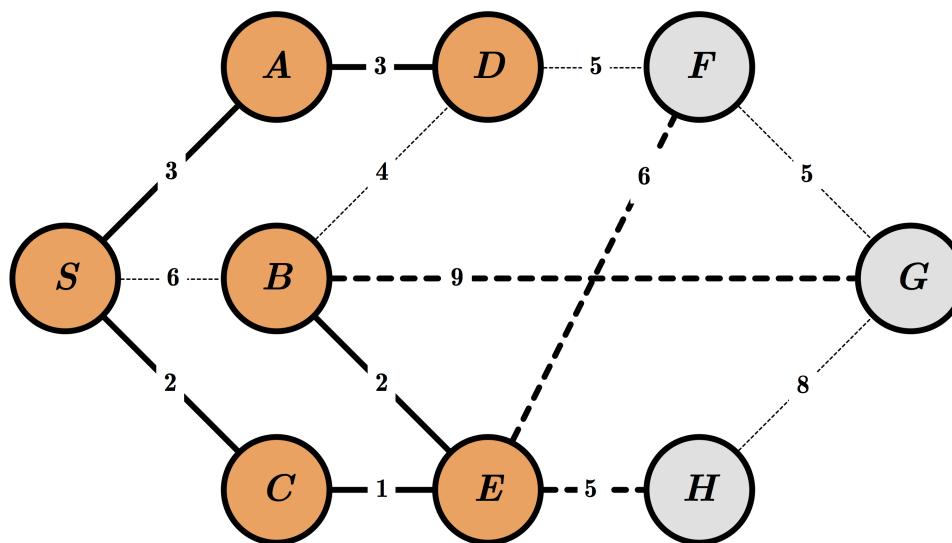
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

Order of Visit:

$S \quad C \quad A \quad E \quad B$

Exercise: UCS



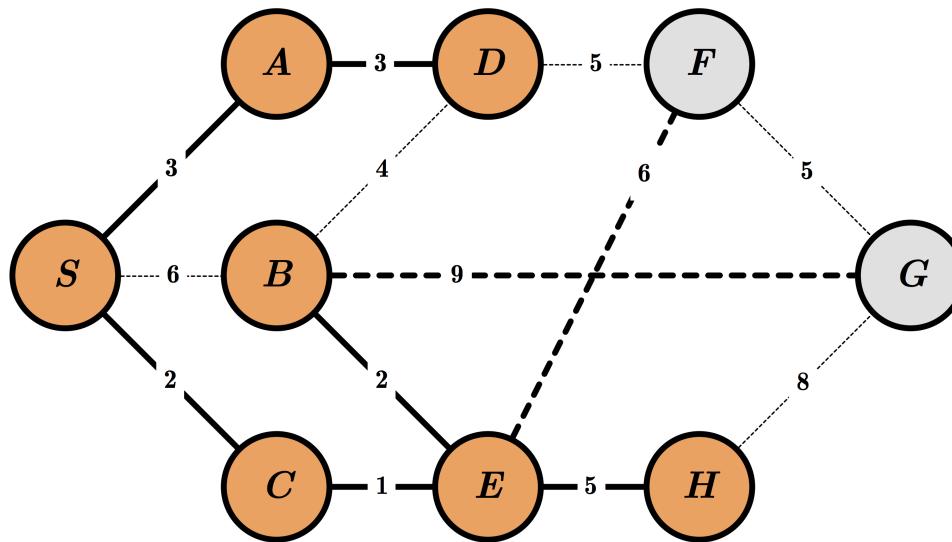
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

Order of Visit:

$S \quad C \quad A \quad E \quad B \quad D$

Exercise: UCS



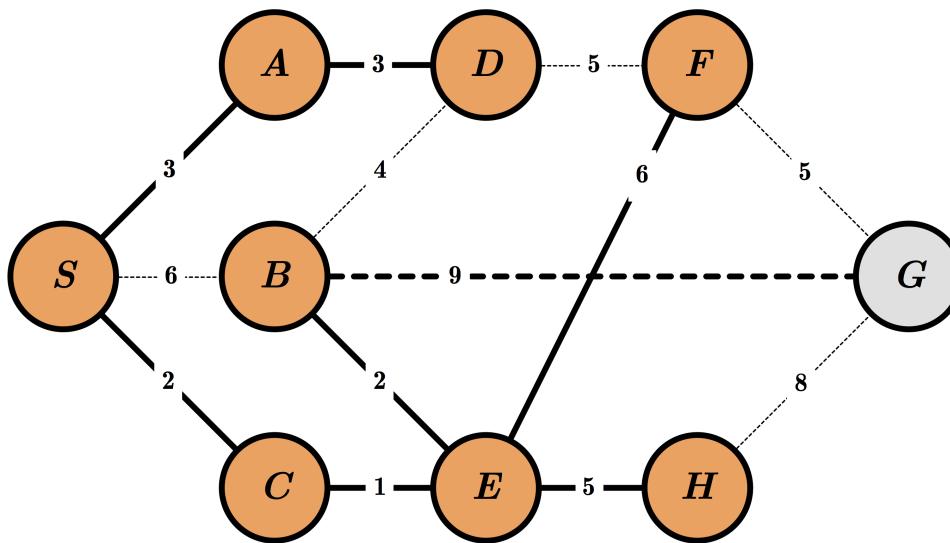
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

Order of Visit:

$S \quad C \quad A \quad E \quad B \quad D \quad H$

Exercise: UCS



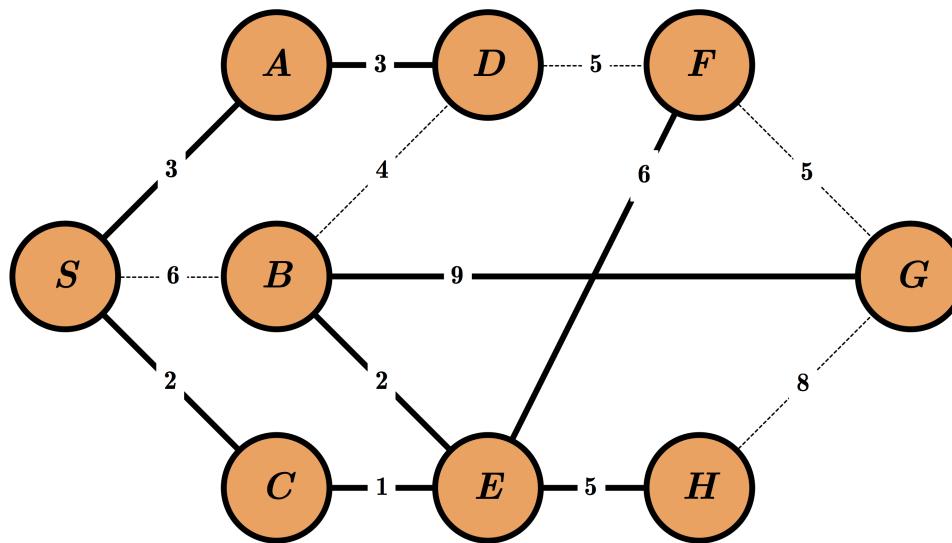
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

Order of Visit:

$S \quad C \quad A \quad E \quad B \quad D \quad H \quad F$

Exercise: UCS



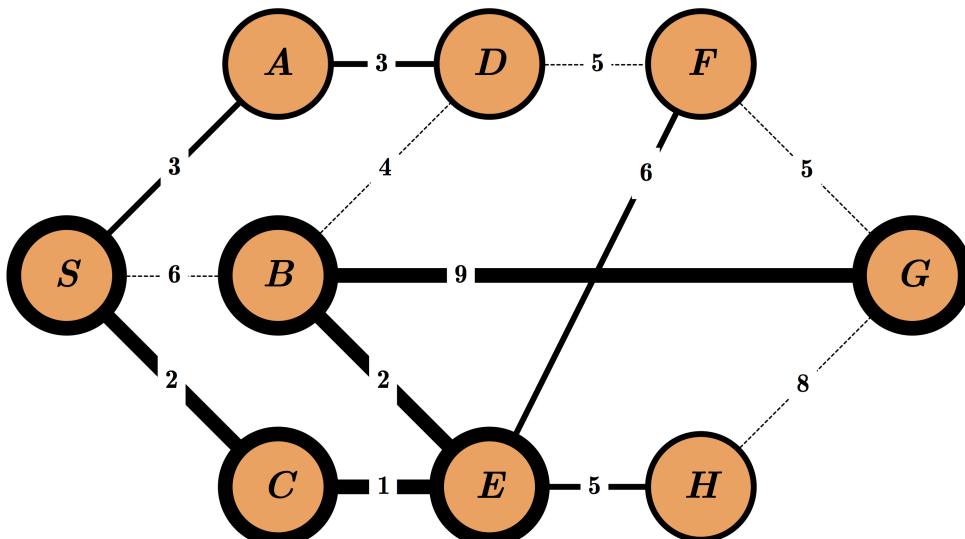
Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

Order of Visit:

$S \quad C \quad A \quad E \quad B \quad D \quad H \quad F \quad G$

Exercise: UCS



Priority Queue:

S_0	C_2	A_3	E_3	B_5	D_6	H_8	F_9	G_{14}
-------	-------	-------	-------	-------	-------	-------	-------	----------

lowest cost
path

14

Order of Visit:

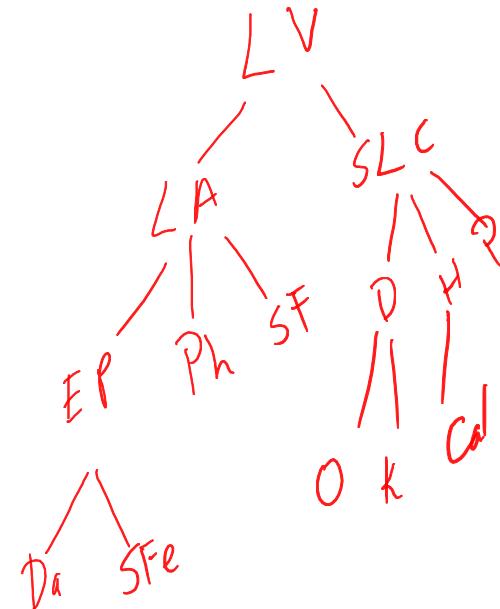
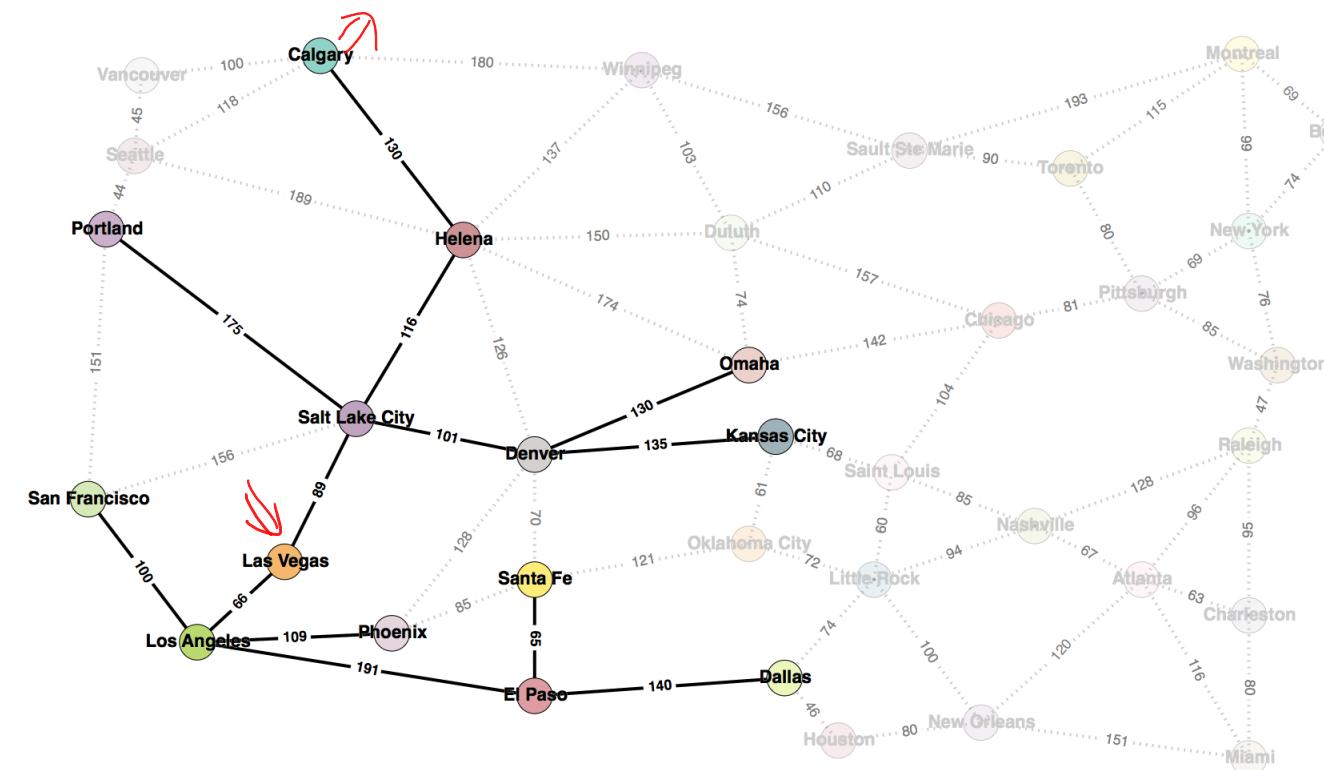
$S \quad C \quad A \quad E \quad B \quad D \quad H \quad F \quad G$

Examples using the map

L2H2

Start: Las Vegas

Goal: Calgary



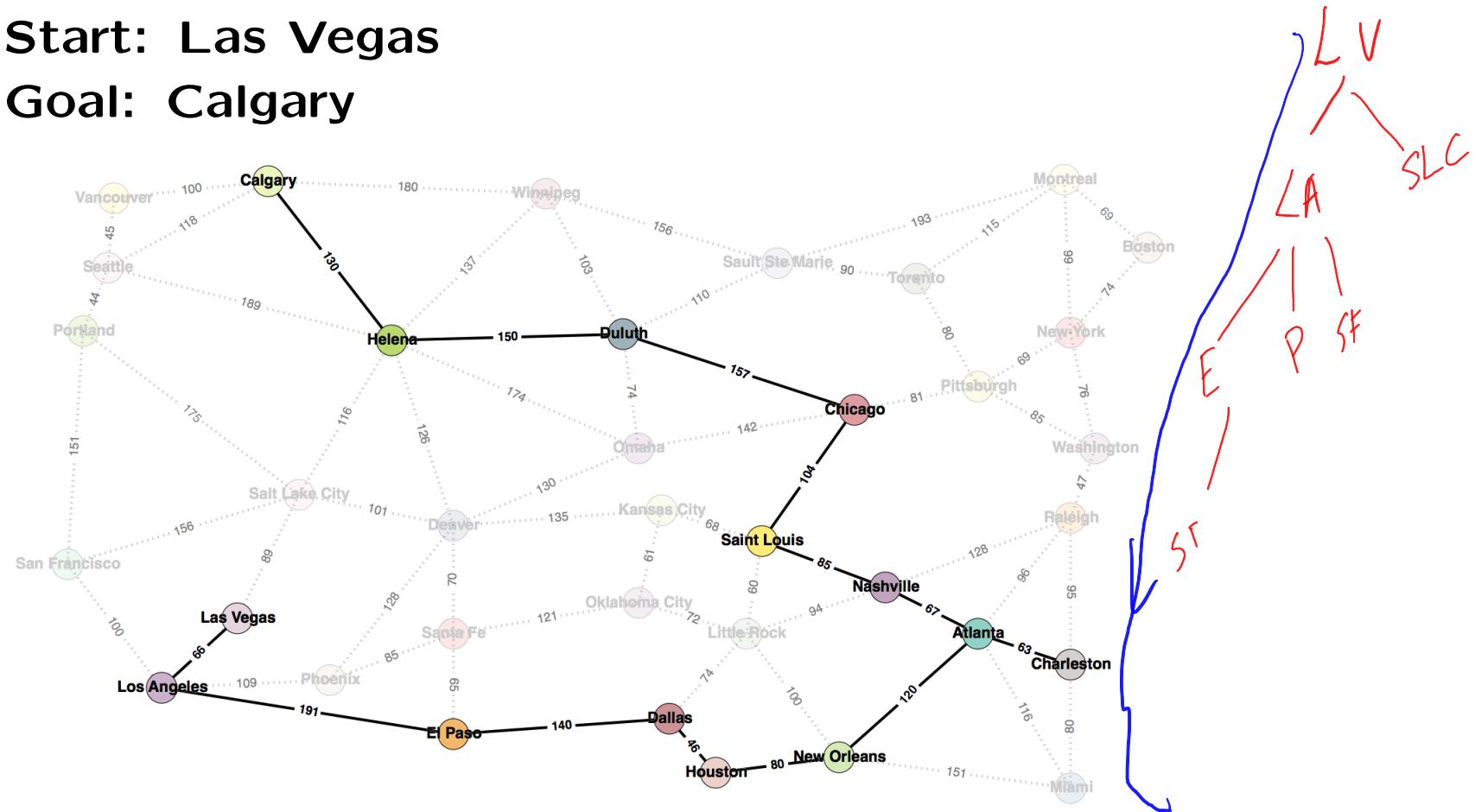
BFS

Order of Visit: Las Vegas, Los Angeles, Salt Lake City, El Paso, Phoenix, San Francisco, Helena, Portland, Dallas, Santa Fe, Kansas City, Omaha, Calgary.

Examples using the map

Start: Las Vegas

Goal: Calgary



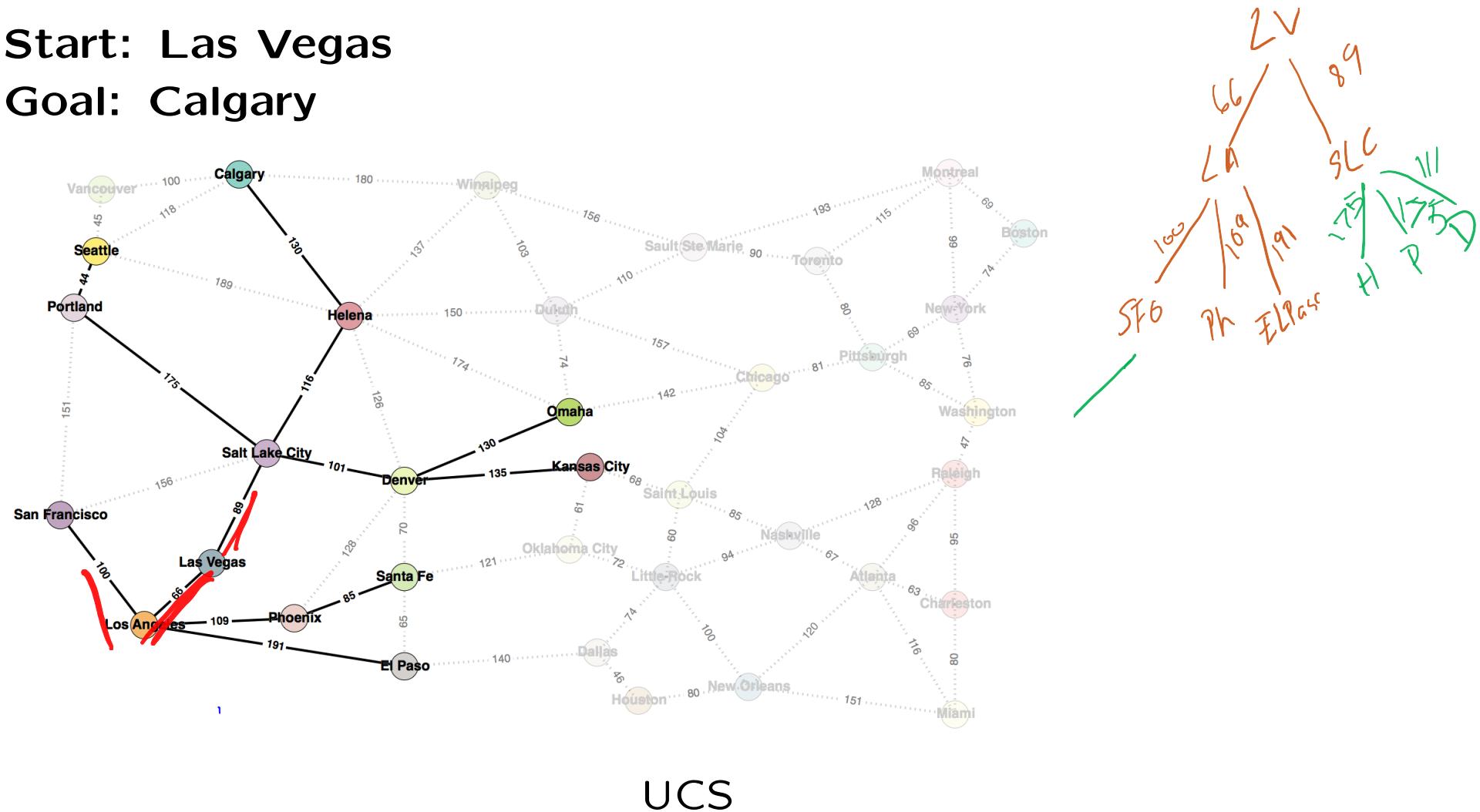
DFS

Order of Visit: Las Vegas, Los Angeles, El Paso, Dallas, Houston, New Orleans, Atlanta, Charleston, Nashville, Saint Louis, Chicago, Duluth, Helena, Calgary.

Examples using the map

Start: Las Vegas

Goal: Calgary



Order of Visit: Las Vegas, Los Angeles, Salt Lake City, San Francisco, Phoenix, Denver, Helena, El Paso, Santa Fe, Portland, Seattle, Omaha, Kansas City, Calgary.

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

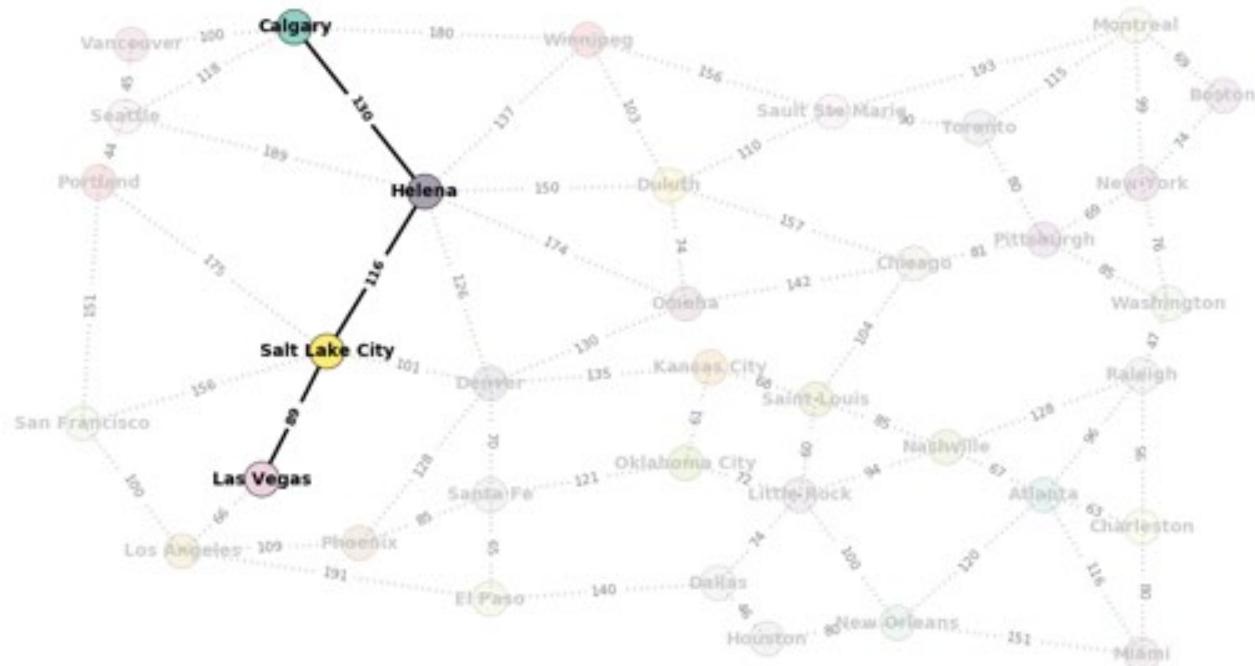
<http://aima.cs.berkeley.edu/>

L3.1

Artificial Intelligence

Search Agents

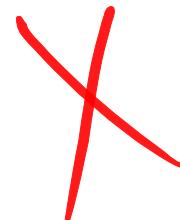
Informed search



Informed search

Use domain knowledge!

- Are we getting close to the goal?
- Use a heuristic function that estimates how close a state is to the goal
- A heuristic does NOT have to be perfect!

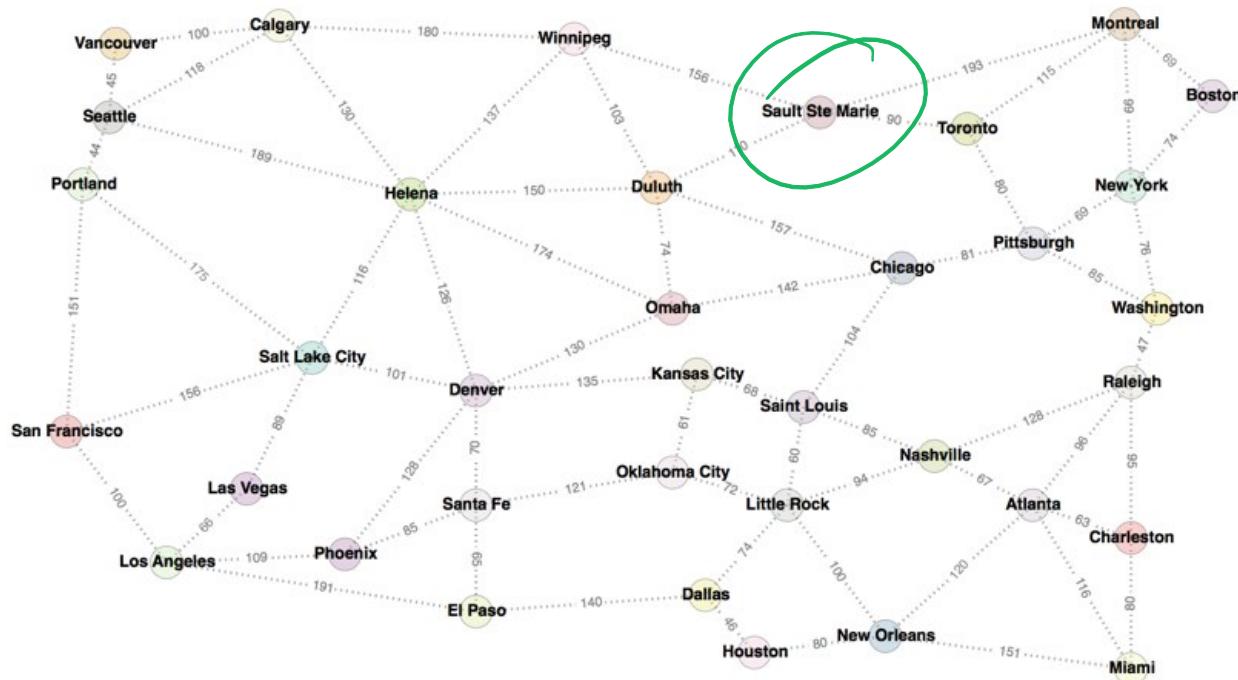


Informed search

Use domain knowledge!

- Are we getting close to the goal?
- Use a heuristic function that estimates how close a state is to the goal
- A heuristic does NOT have to be perfect!
- Example of strategies:
 1. Greedy best-first search
 2. A* search
 3. IDA*

Informed search

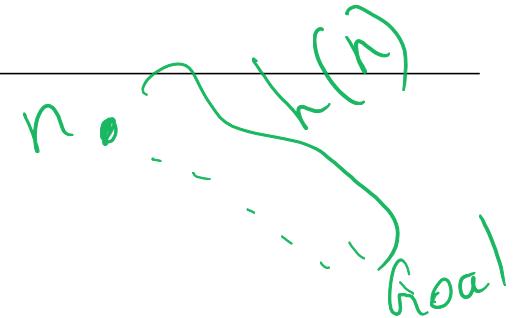


Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

Heuristic!

The distance is the straight line distance. The goal is to get to Sault Ste Marie, so all the distances are from each city to Sault Ste Marie.

Greedy search



- Evaluation function $h(n)$ (*heuristic*)
- $h(n)$ estimates the cost from n to the closest goal
- Example: $h_{\text{SLD}}(n) = \text{straight-line distance from } n \text{ to Sault Ste Marie}$
- Greedy search expands the node that **appears** to be closest to goal



Greedy search

- Picks the node that is closest to goal

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

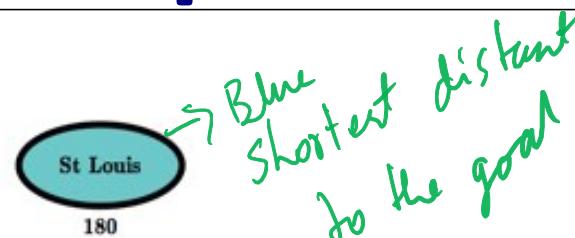
        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

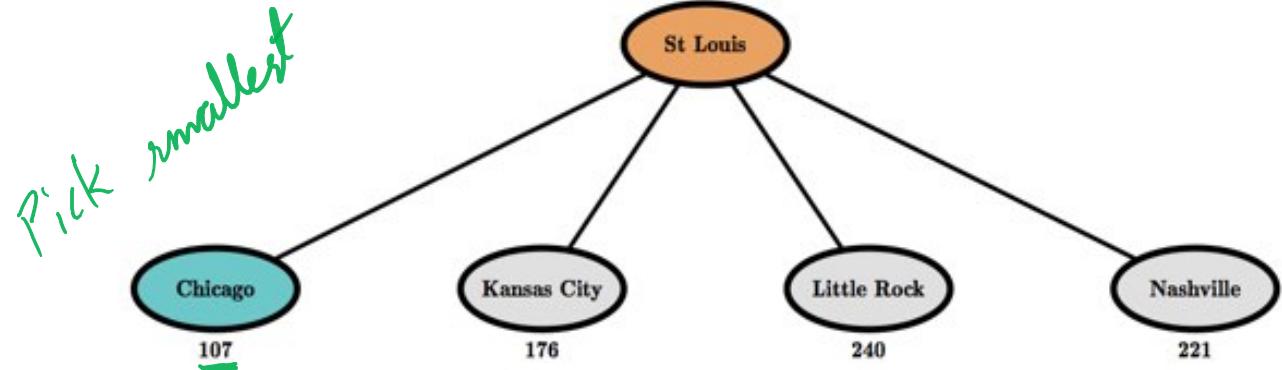
Greedy search example

The initial state:



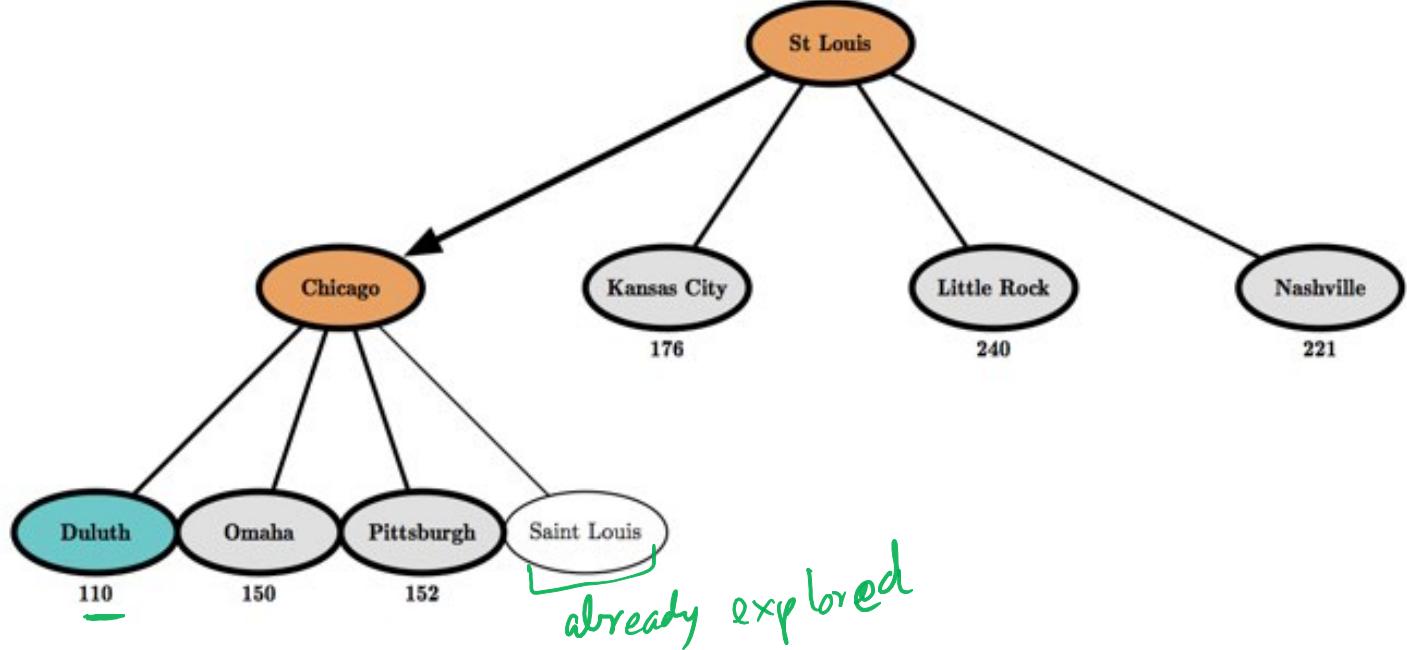
Greedy search example

After expanding St Louis:



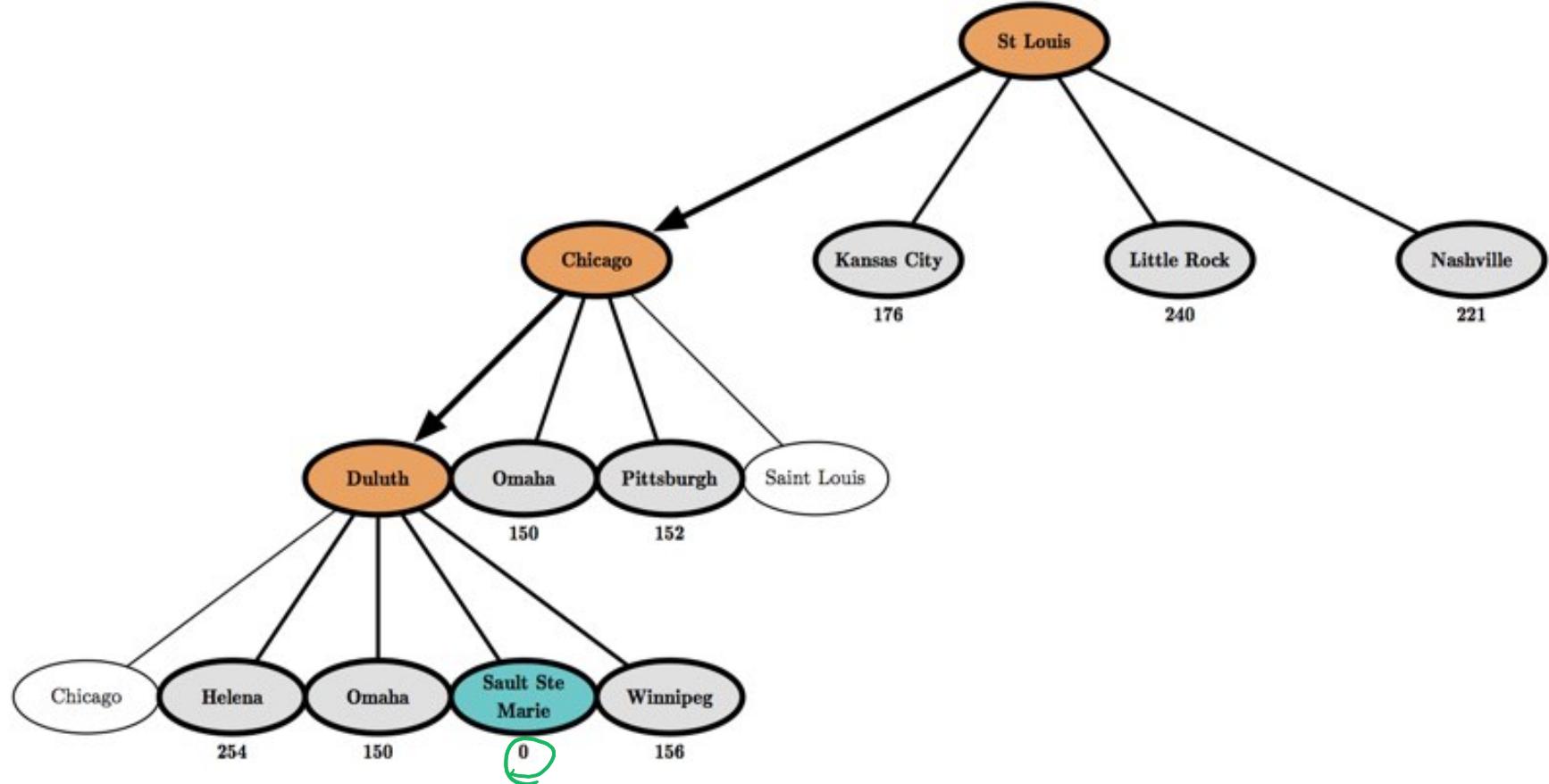
Greedy search example

After expanding Chicago:



Greedy search example

After expanding Duluth:

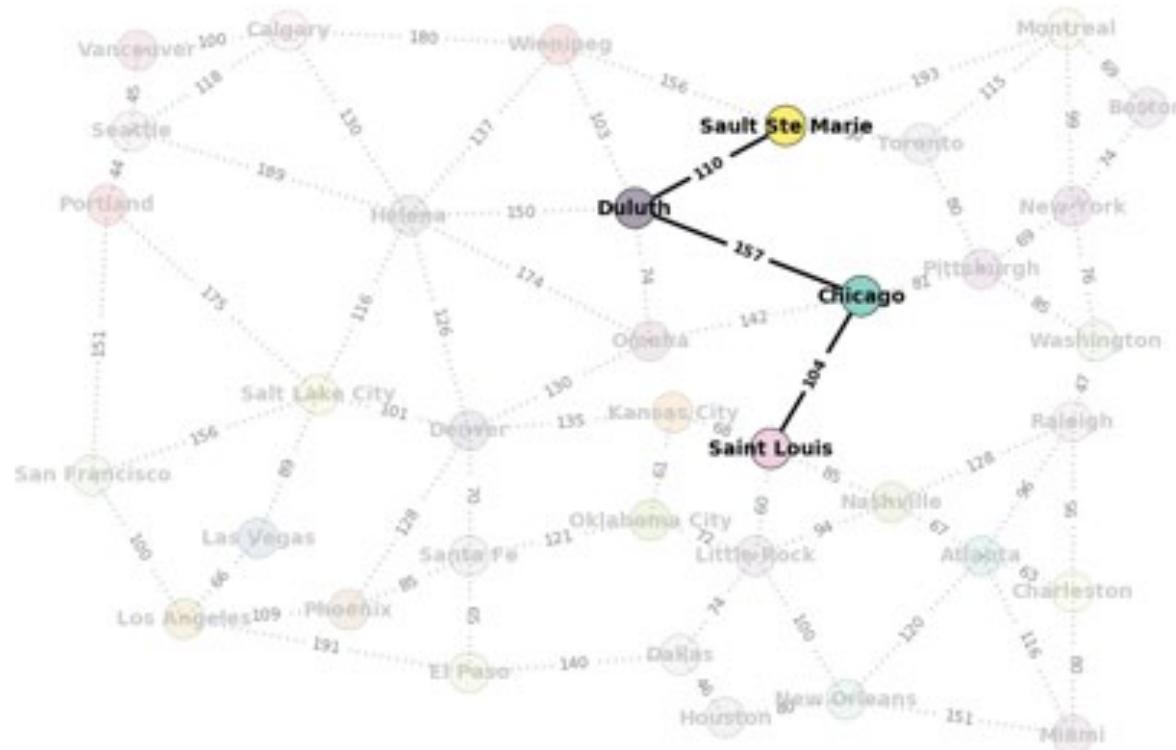


Examples using the map

Start: Saint Louis

Goal: Sault Ste Marie

Path : 371 km (not the shortest)



Greedy search

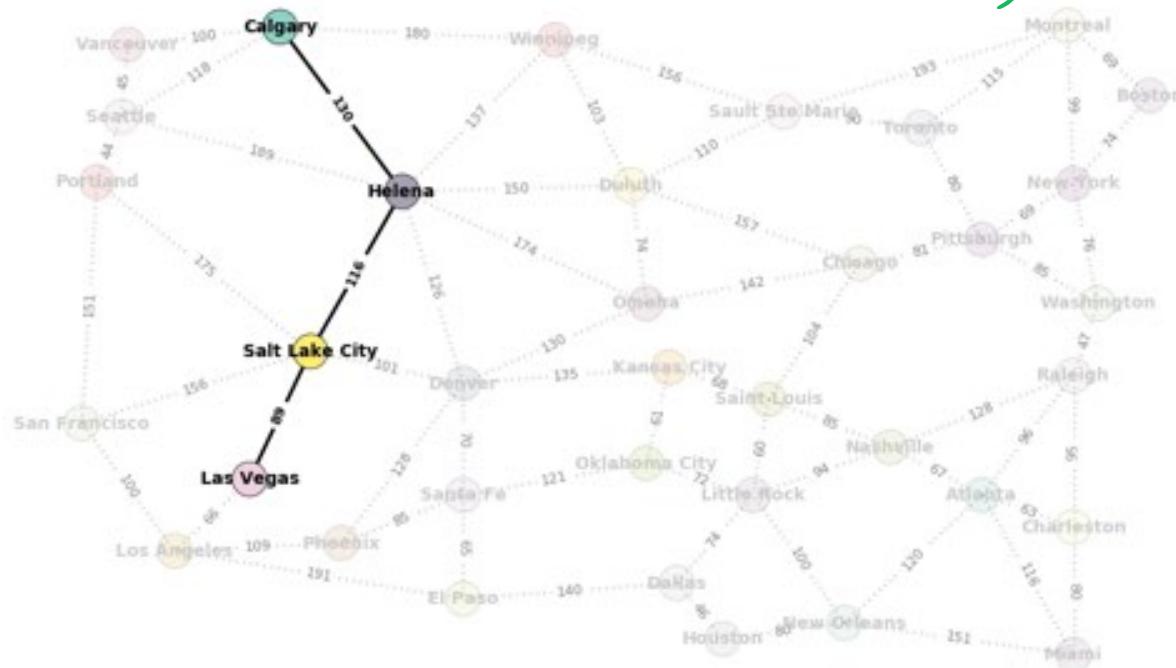
Examples using the map

Start: Las Vegas

Goal: Calgary

Path: 335

130
116
89
~~335~~



Greedy search

A* search

L3.2.1

greedy doesn't always give the best

- Minimize the total estimated solution cost
- Combines:
 - $g(n)$: cost to reach node n
 - $h(n)$: cost to get from n to the goal
 - $f(n) = g(n) + h(n)$



$f(n)$ is the estimated cost of the cheapest solution through n

A* search



```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

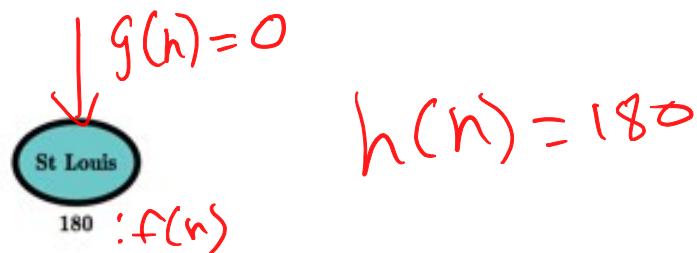
        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

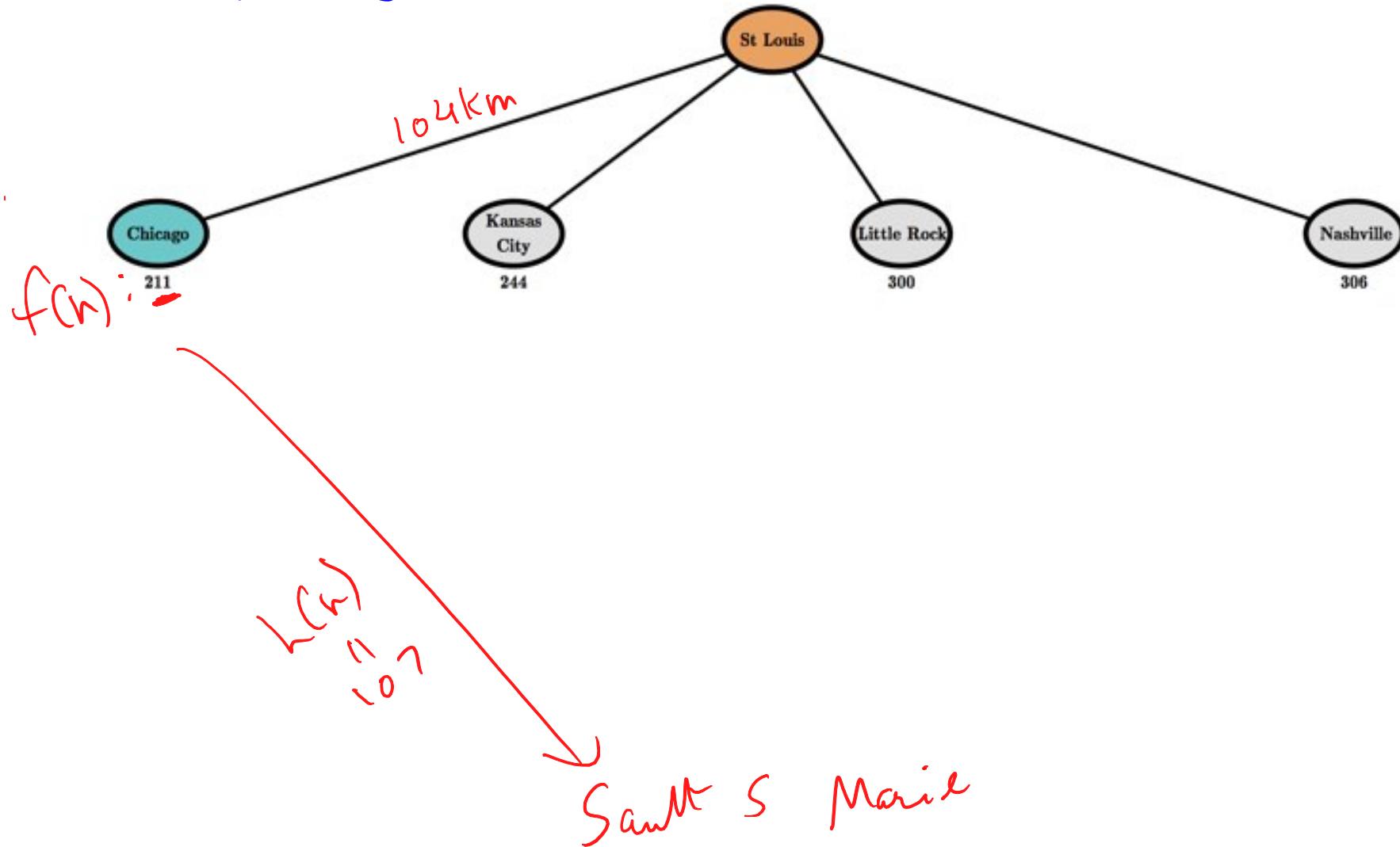
A* search example

The initial state:



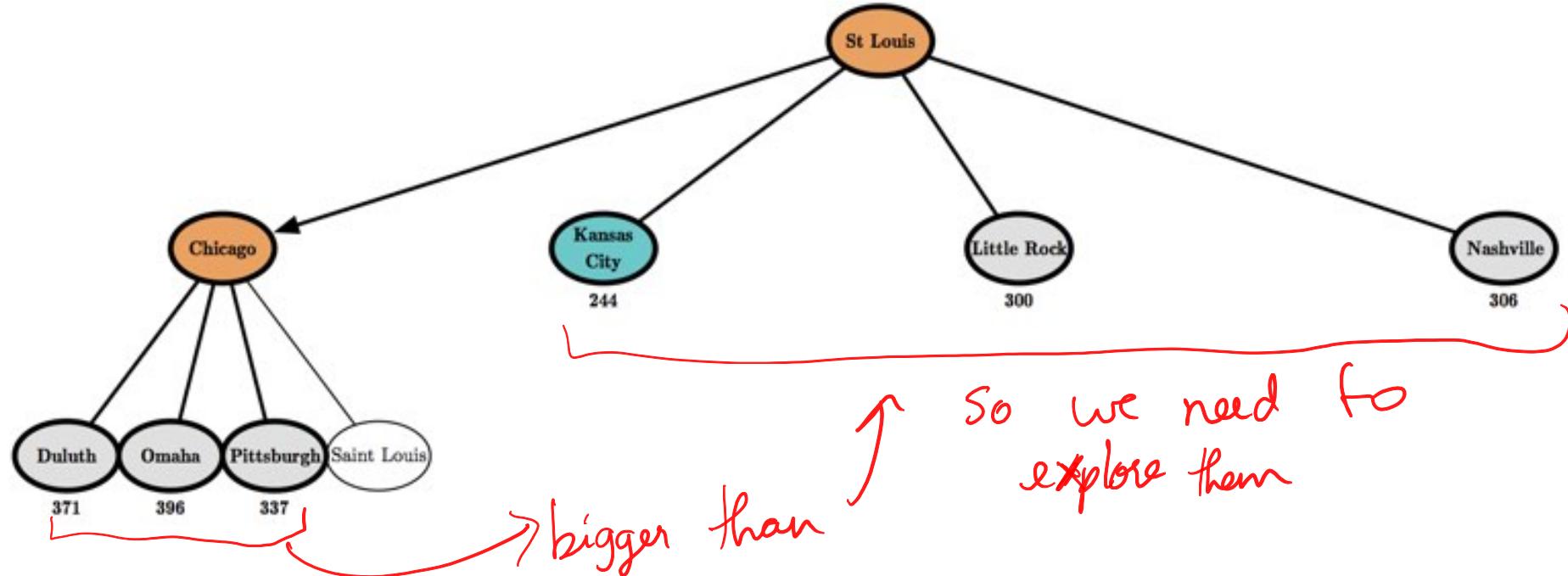
A* search example

After expanding St Louis:



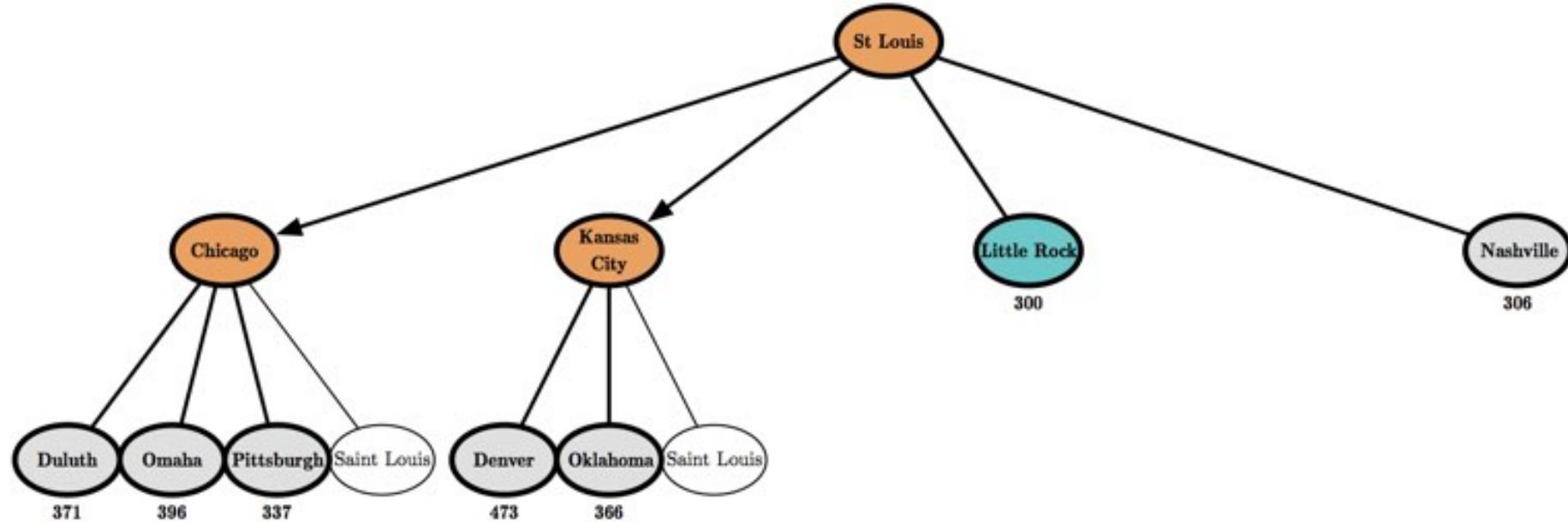
A* search example

After expanding Chicago:



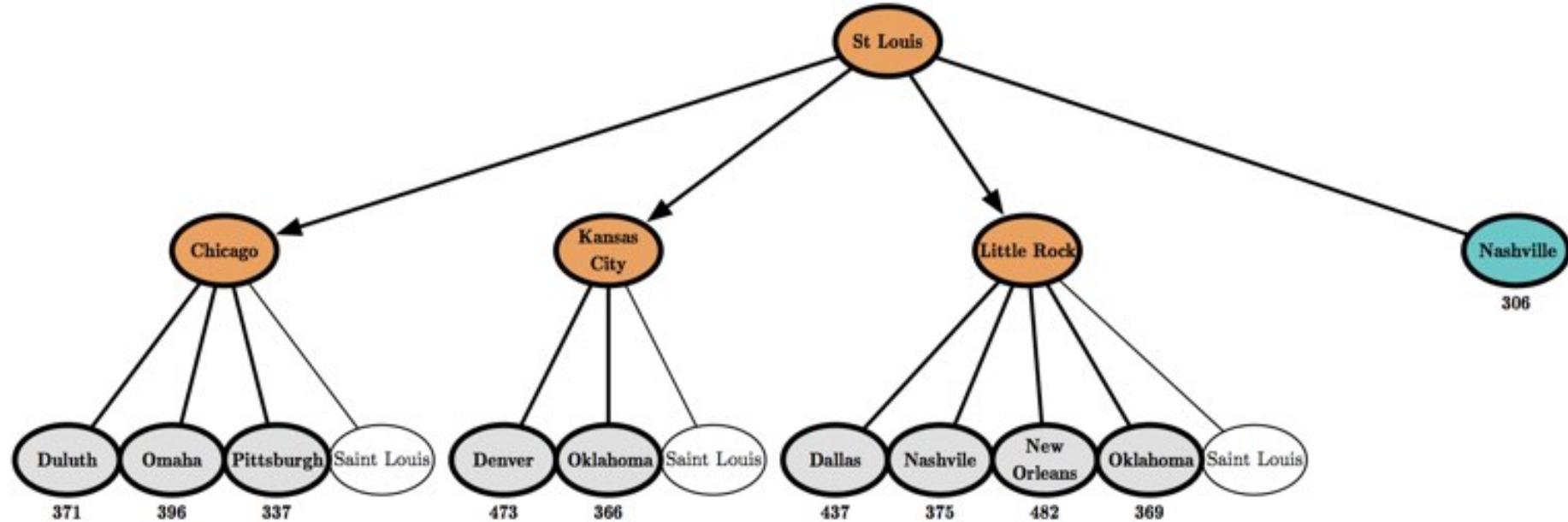
A* search example

After expanding Kansas City:



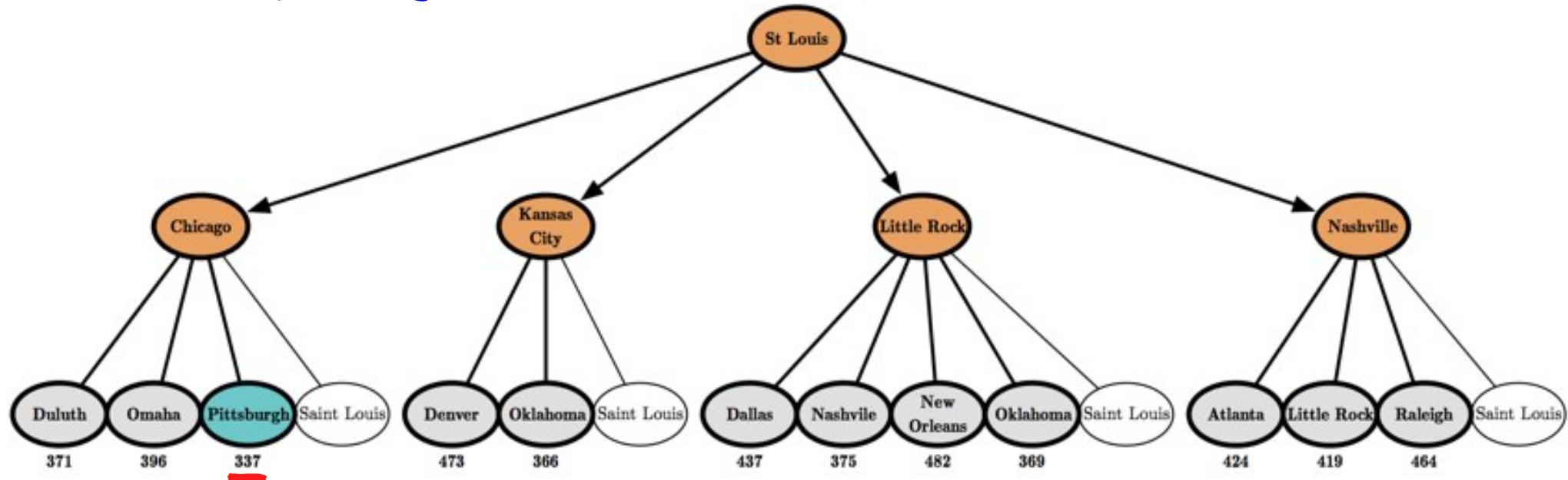
A* search example

After expanding Little Rock:



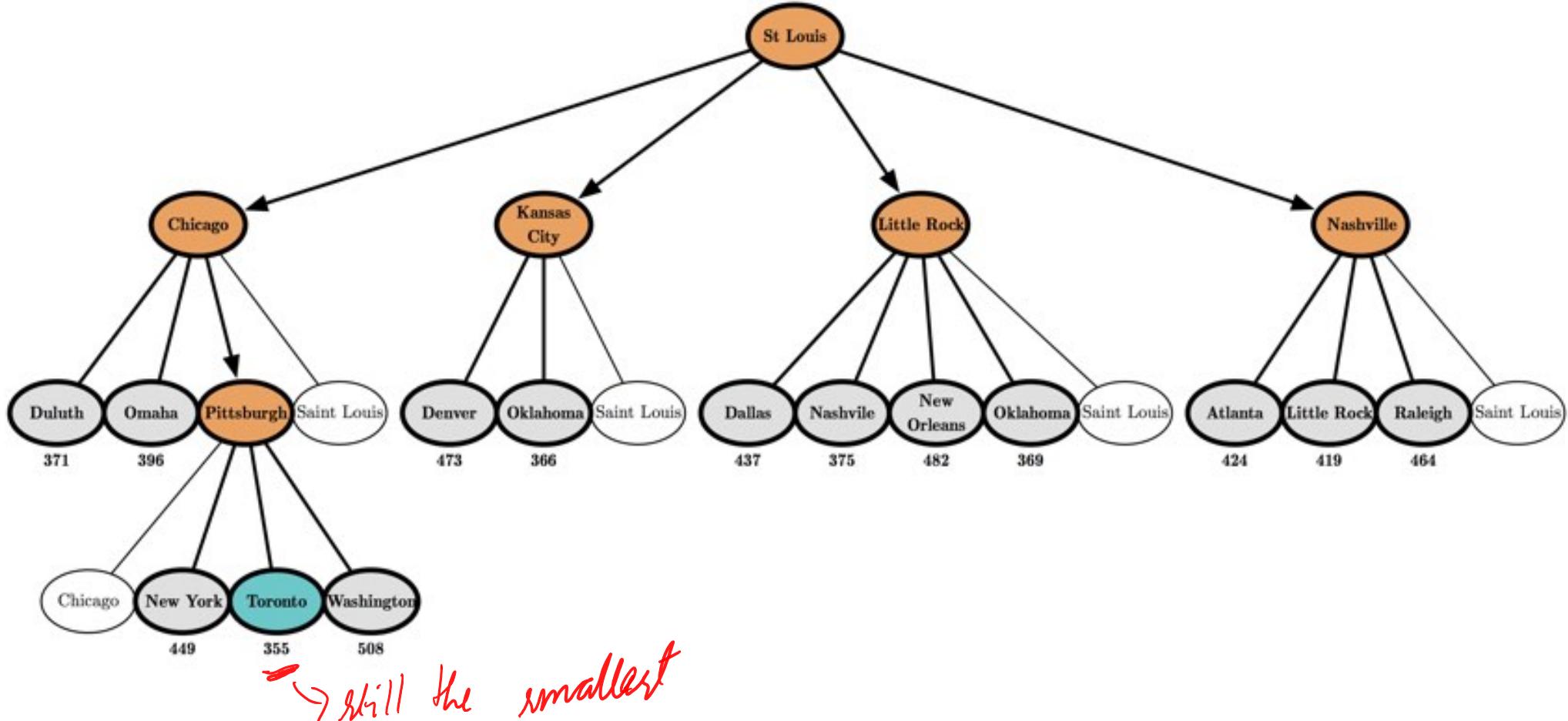
A* search example

After expanding Nashville:



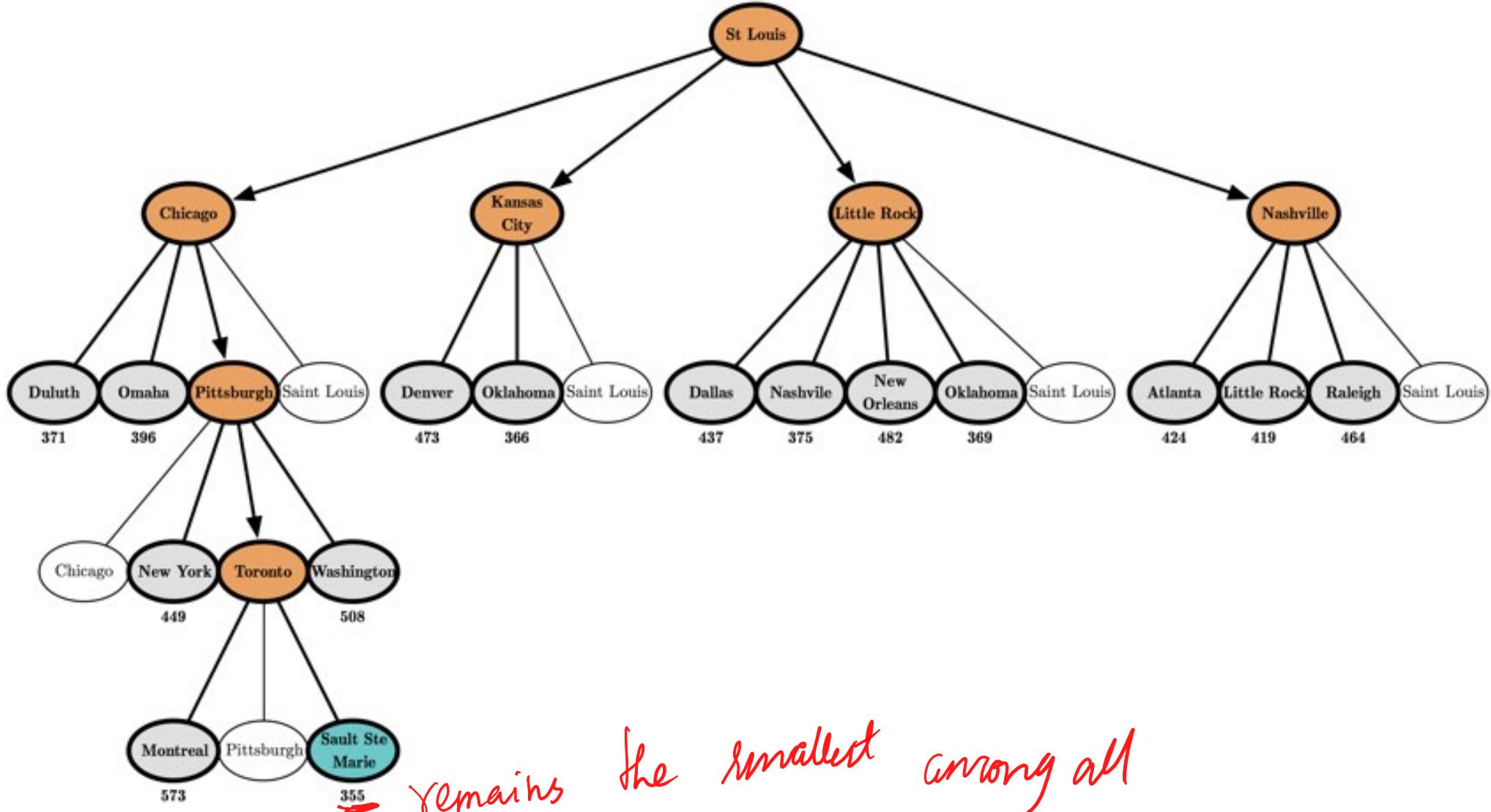
A* search example

After expanding Pittsburgh:



A* search example

After expanding Toronto:



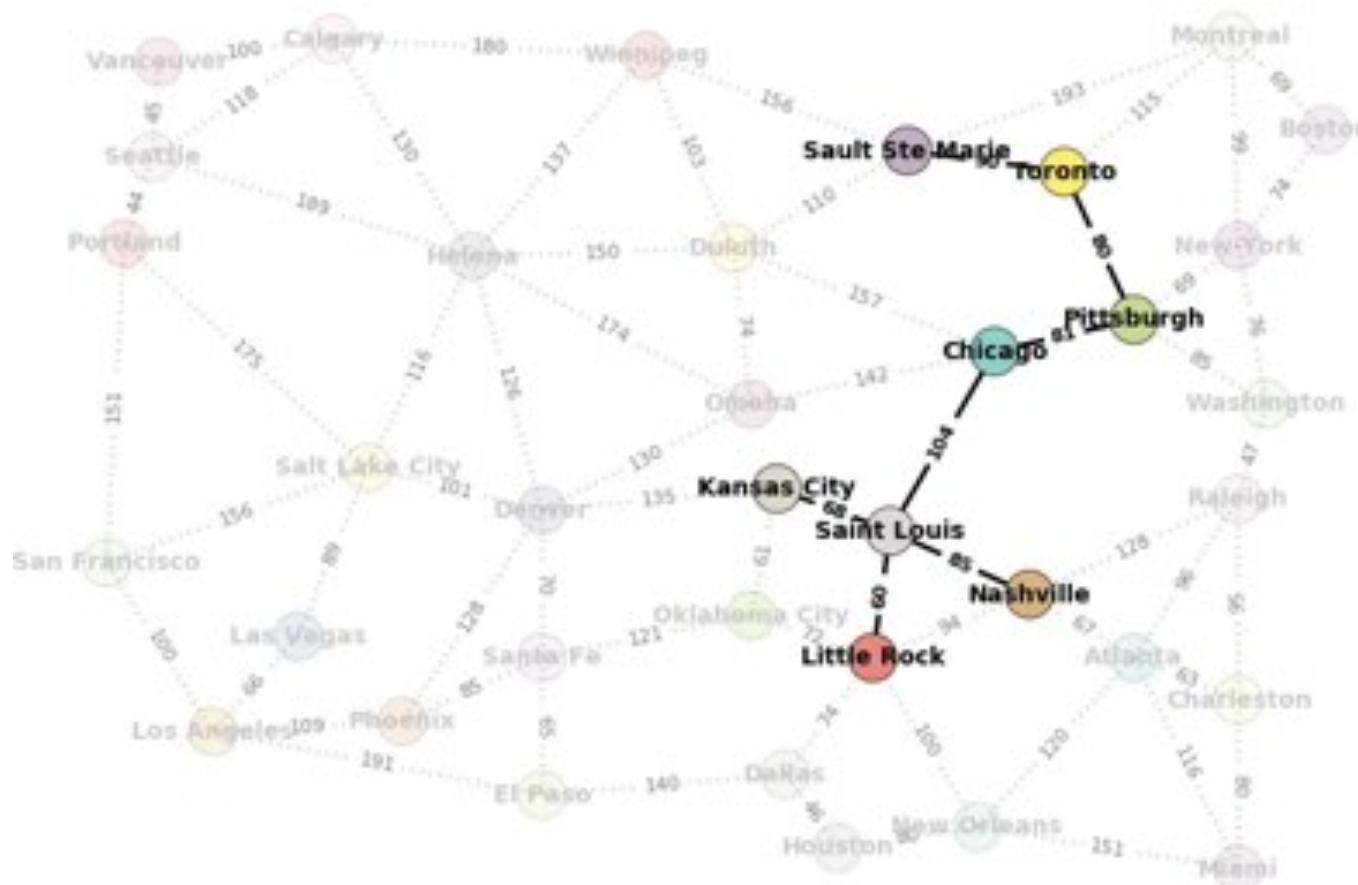
Examples using the map

Start: Saint Louis

Goal: Sault Ste Marie

Path: 255 < 371

104
91
90
90
255

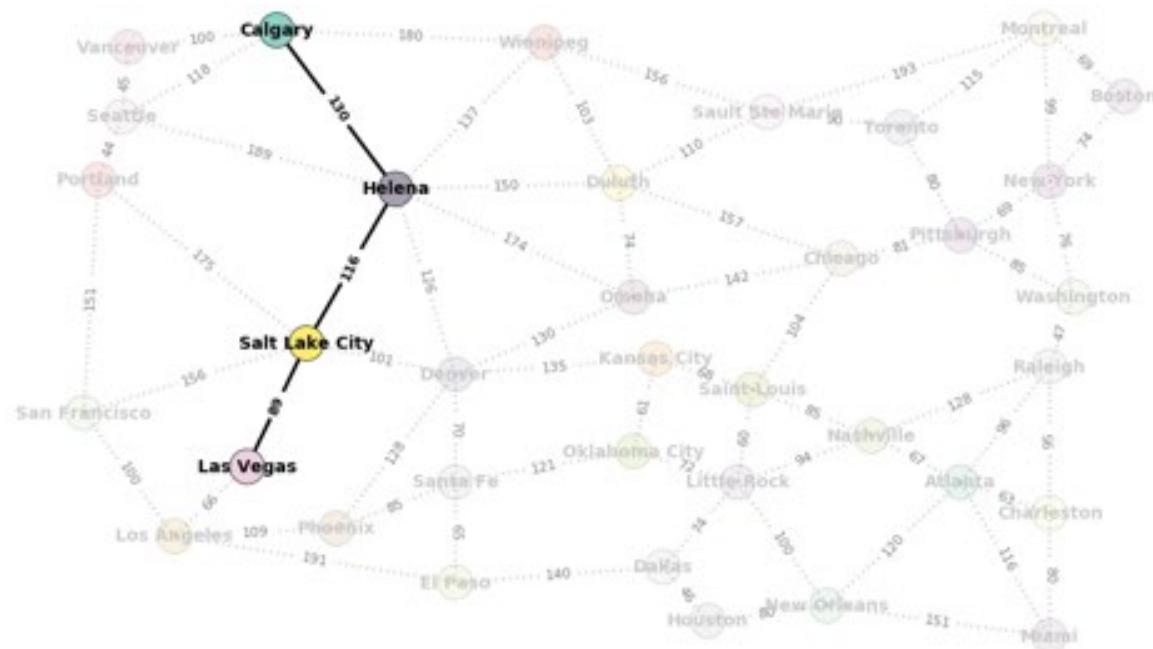


A*

Examples using the map

Start: Las Vegas

Goal: Calgary

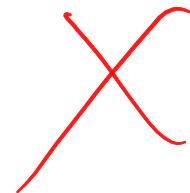


A*

Admissible heuristics

A good heuristic can be powerful.

Only if it is of a “good quality”



Admissible heuristics

*we need good heuristic
for A**

A good heuristic can be powerful.

Only if it is of a “good quality”

A good heuristic must be admissible.

Admissible heuristics

- An **admissible** heuristic never overestimates the cost to reach the goal, that is it is **optimistic**

- A heuristic h is admissible if

$$\underbrace{\forall \text{ node } n, h(n) \leq h^*(n)}$$

the heuristic estimate should always be less than the true cost

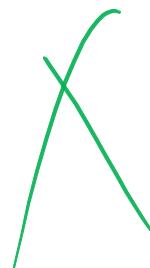
where h^* is true cost to reach the goal from n .

- h_{SLD} (used as a heuristic in the map example) is admissible because it is by definition the shortest distance between two points.



A* Optimality

If $h(n)$ is admissible, A* using tree search is optimal.



A* Optimality

L.3.2.2

If $h(n)$ is admissible, A* using tree search is optimal.

Rationale:

Proof

- Suppose G_o is the optimal goal.
- Suppose G_s is some suboptimal goal.
- Suppose n is an unexpanded node in the fringe such that n is on the shortest path to G_o .
- $f(G_s) = g(G_s)$ since $h(G_s) = 0$

$$f(G_o) = g(G_o) \text{ since } h(G_o) = 0$$

$$f(G_s) > g(G_o) \text{ since } G_s \text{ is suboptimal}$$

$$\text{Then } f(G_s) > f(G_o) \dots (1)$$

$h^*(n) = \text{true cost}$

- $h(n) \leq h^*(n)$ since h is admissible

$$g(n) + h(n) \leq g(n) + h^*(n) = g(G_o) = f(G_o)$$

$$\text{Then, } f(n) \leq f(G_o) \dots (2)$$

$$\text{From (1) and (2)} \quad f(G_s) > f(n)$$

so A* will never select G_s during the search and hence A* is optimal.

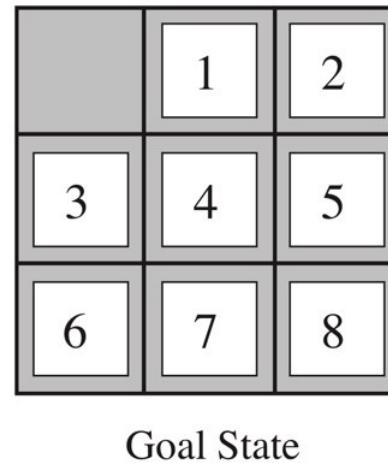
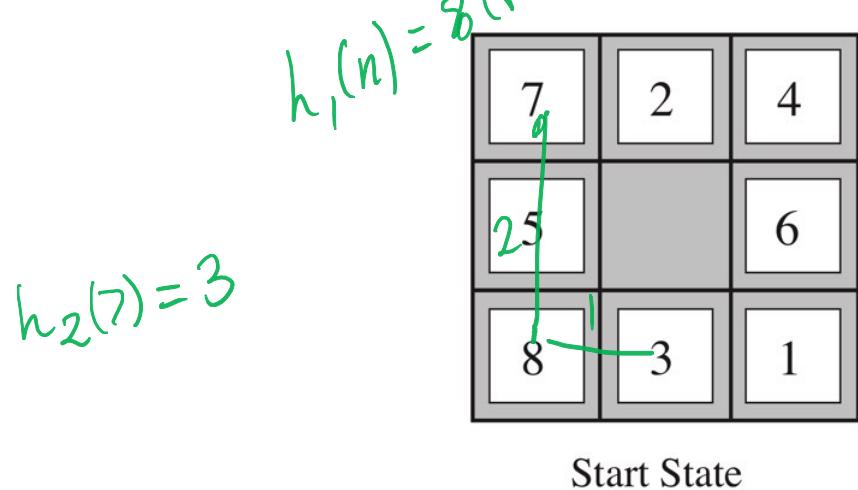
A* will always select the optimal Path

A* search criteria

- **Complete:** Yes
- **Time:** exponential
- **Space:** keeps every node in memory, the biggest problem
- **Optimal:** Yes!

↗ also exponential

Heuristics



- The solution is 26 steps long.
- $h_1(n) = \text{number of misplaced tiles}$
- $h_2(n) = \text{total Manhattan distance (sum of the horizontal and vertical distances).}$
- $h_1(n) = 8$
- Tiles 1 to 8 in the start state gives: $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$ which does not overestimate the true solution.

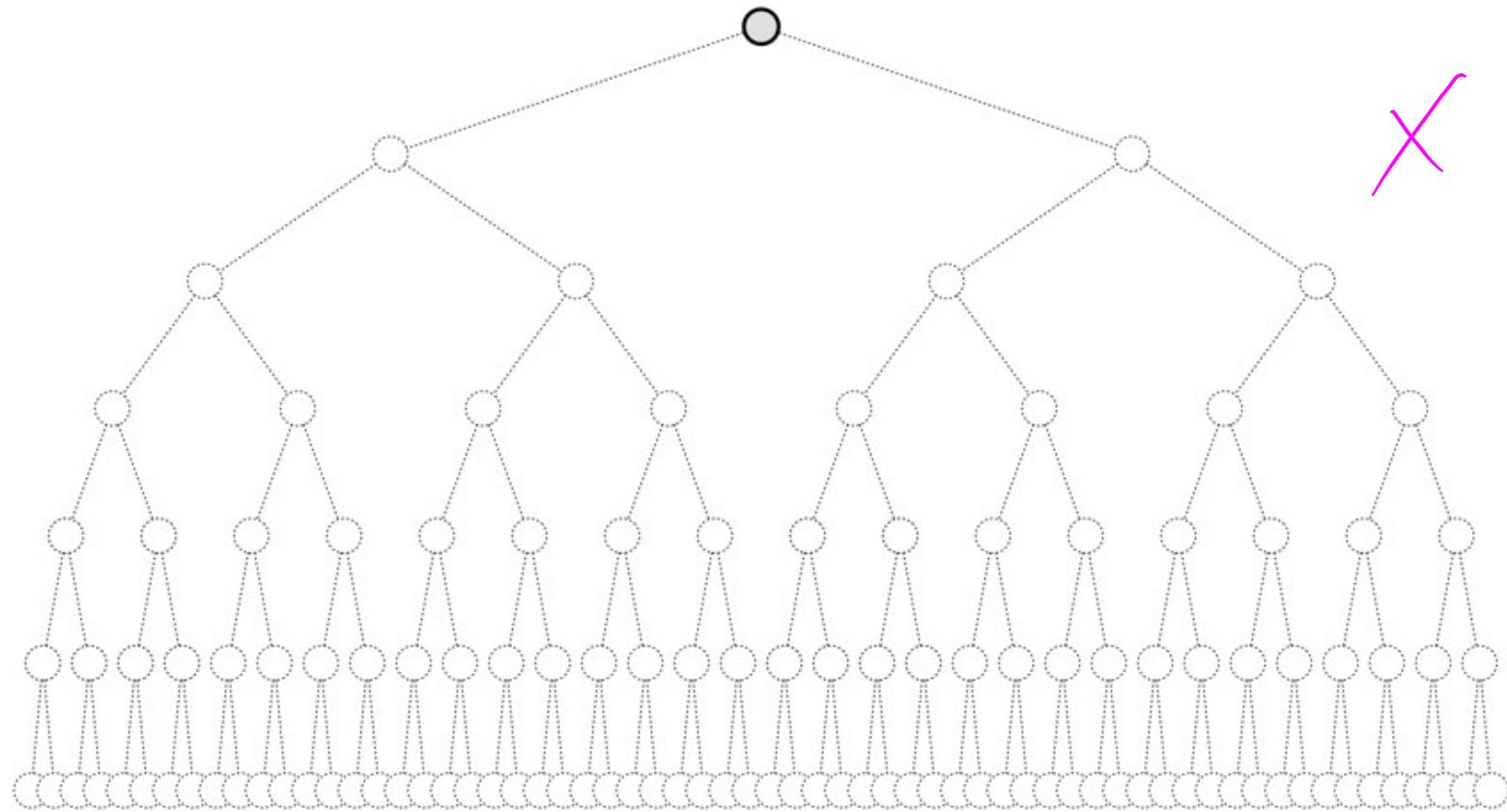
6 7 9

Both $h_1(n)$ and $h_2(n)$ are admissible heuristic

Search Algorithms: Recap

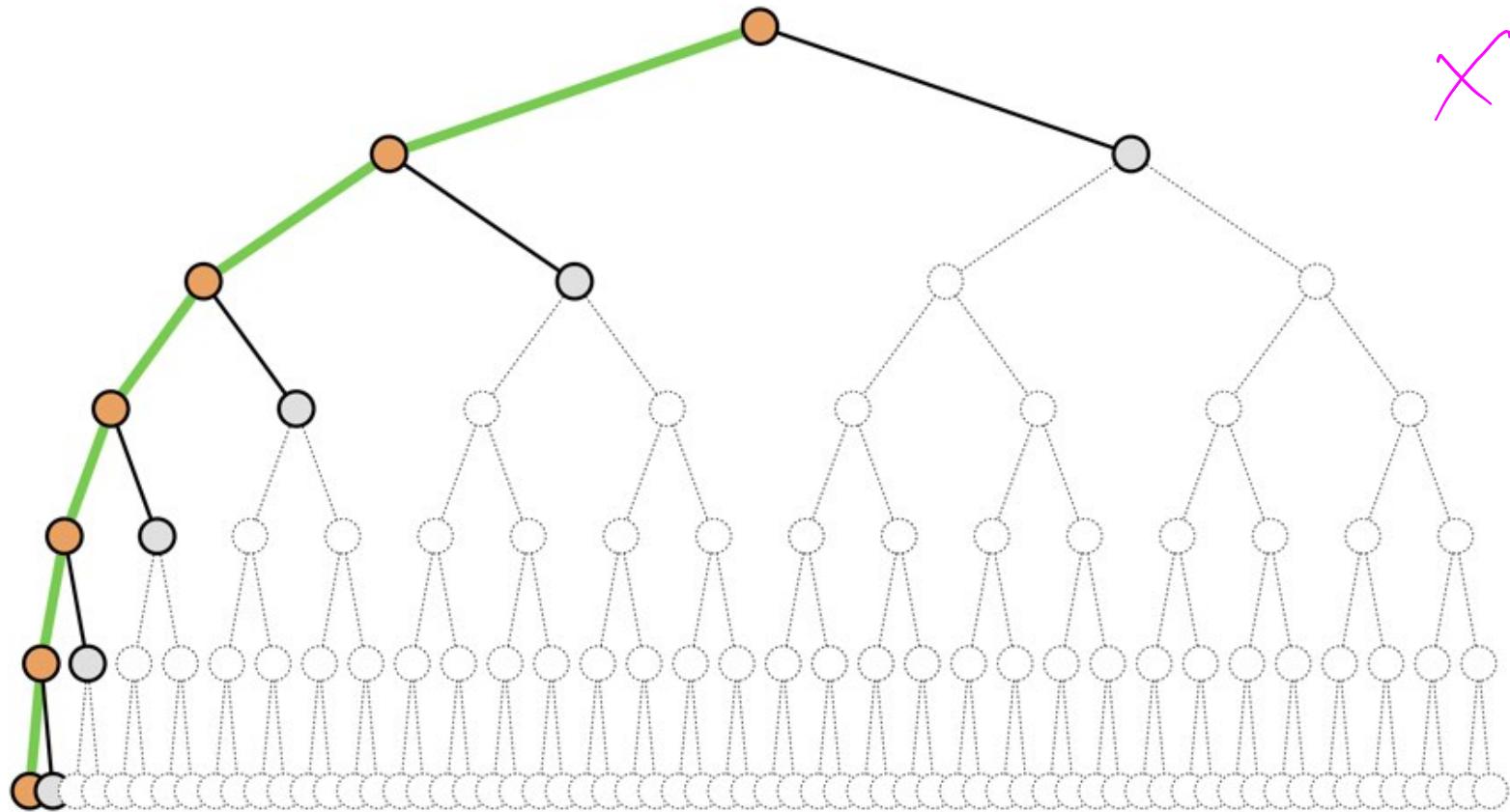
- **Uninformed Search:** Use no domain knowledge.
iterative Deepening search.
BFS, DFS, DLS, IDS, UCS.
- **Informed Search:** Use a heuristic function that estimates how close a state is to the goal.
Greedy search, A*, IDA*.

DFS



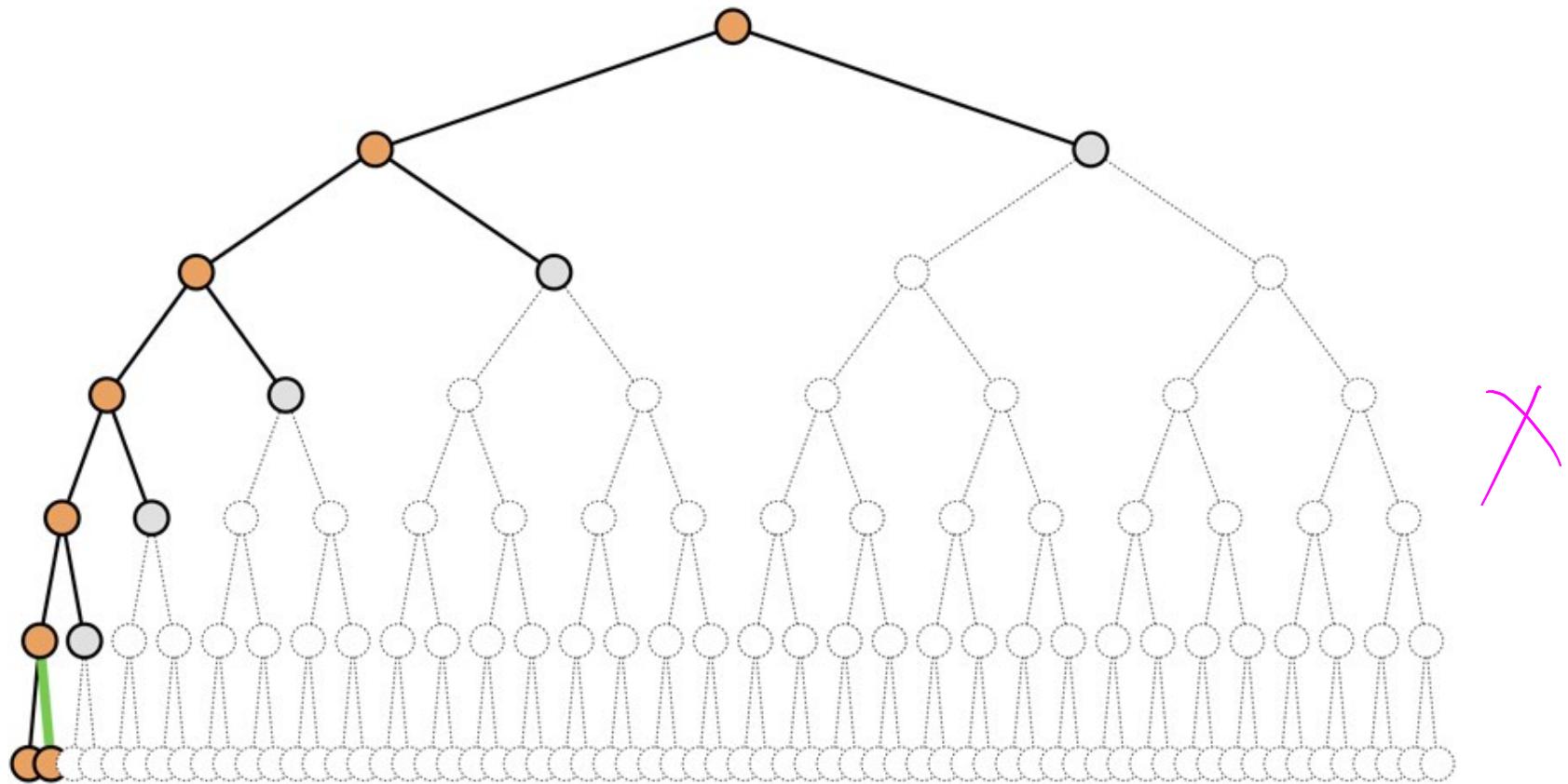
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



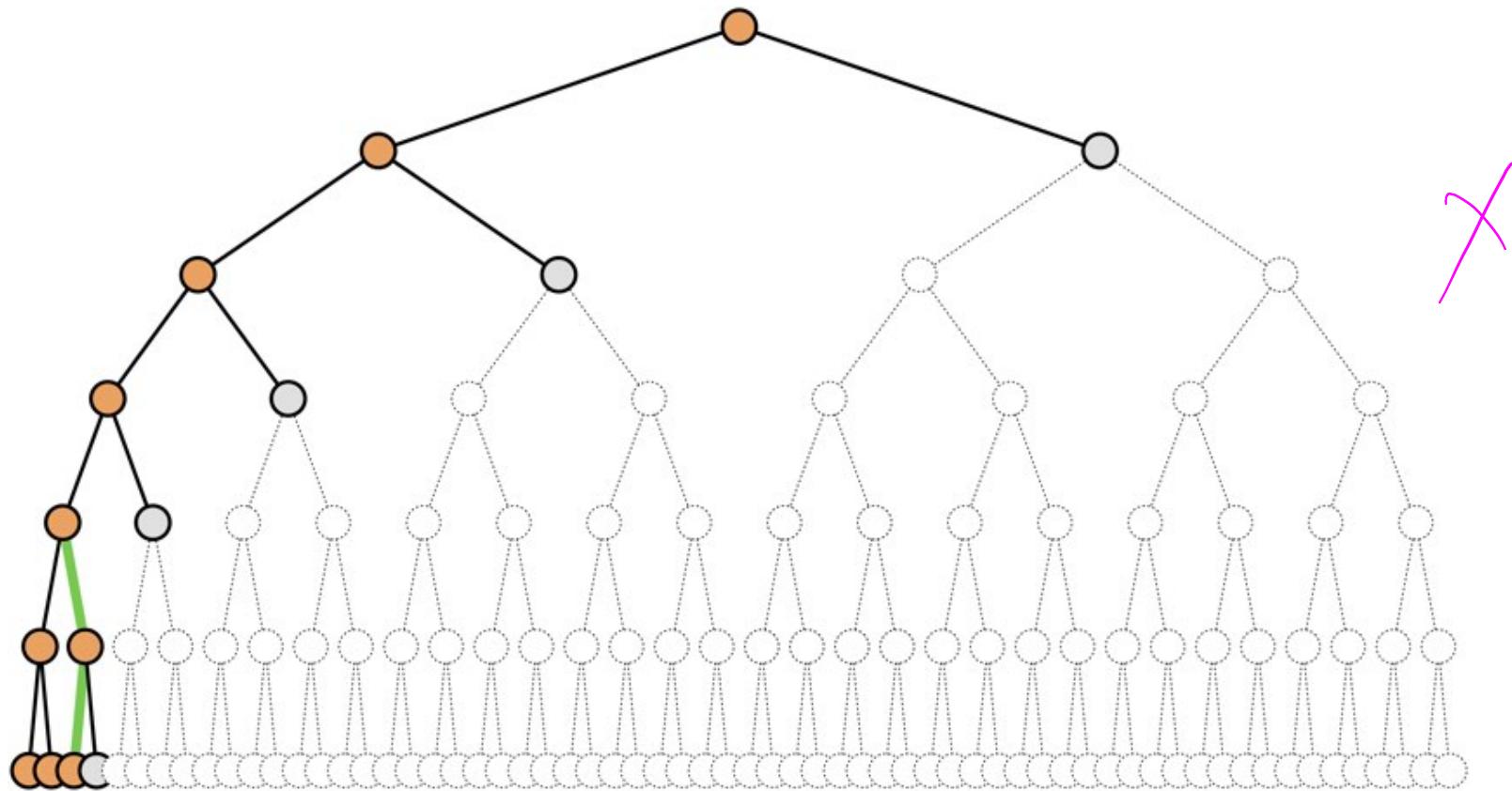
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



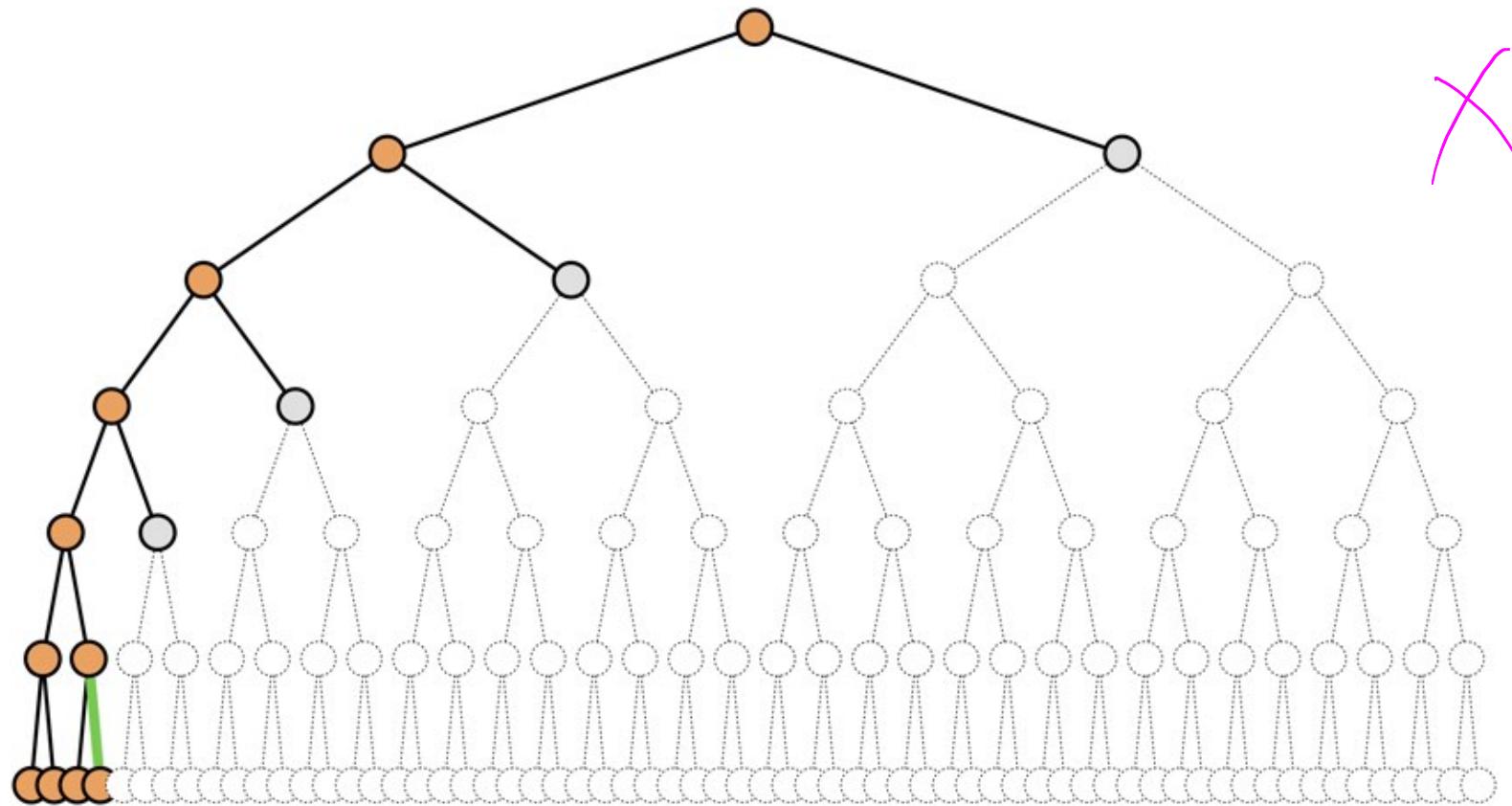
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



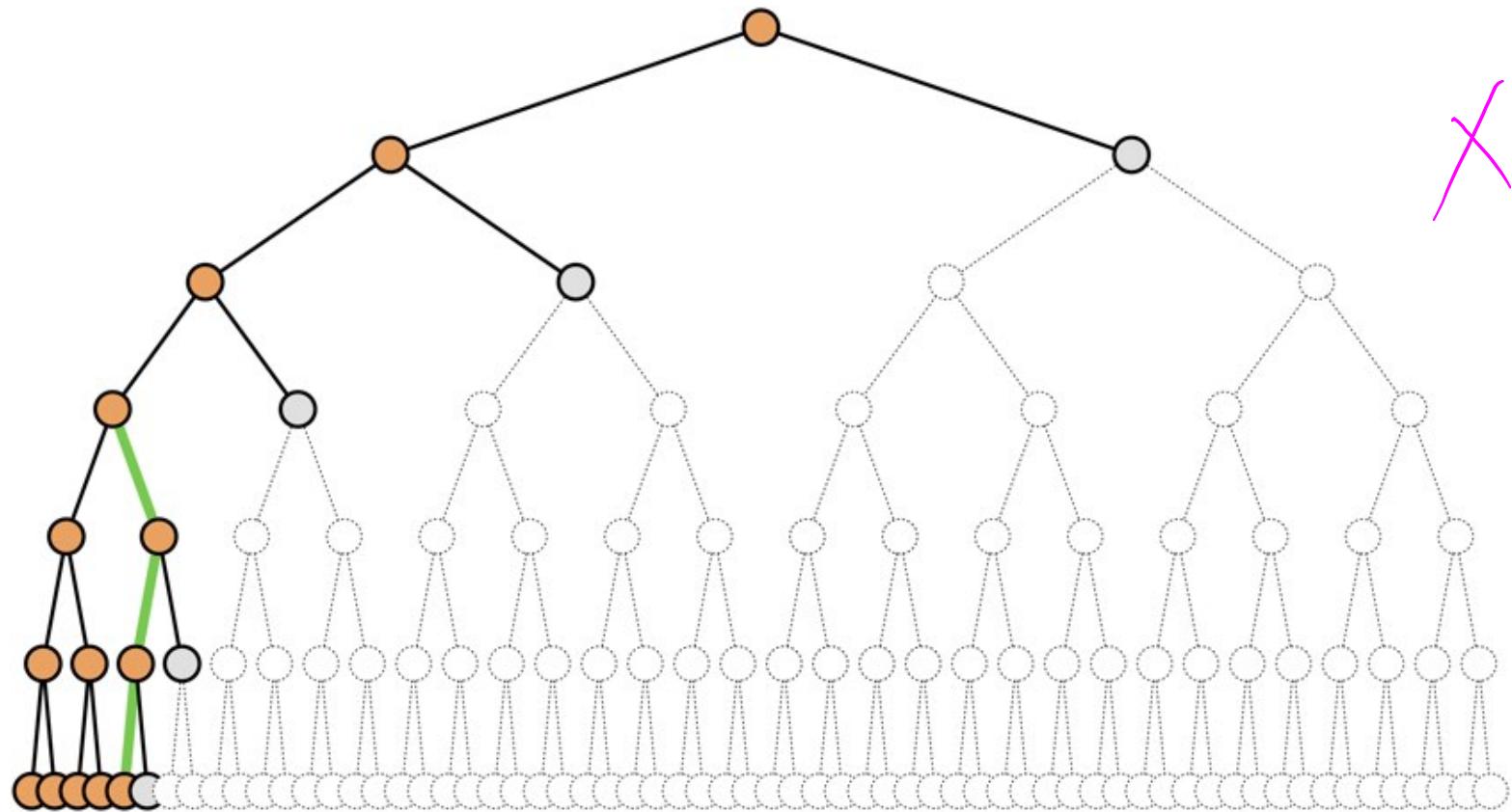
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



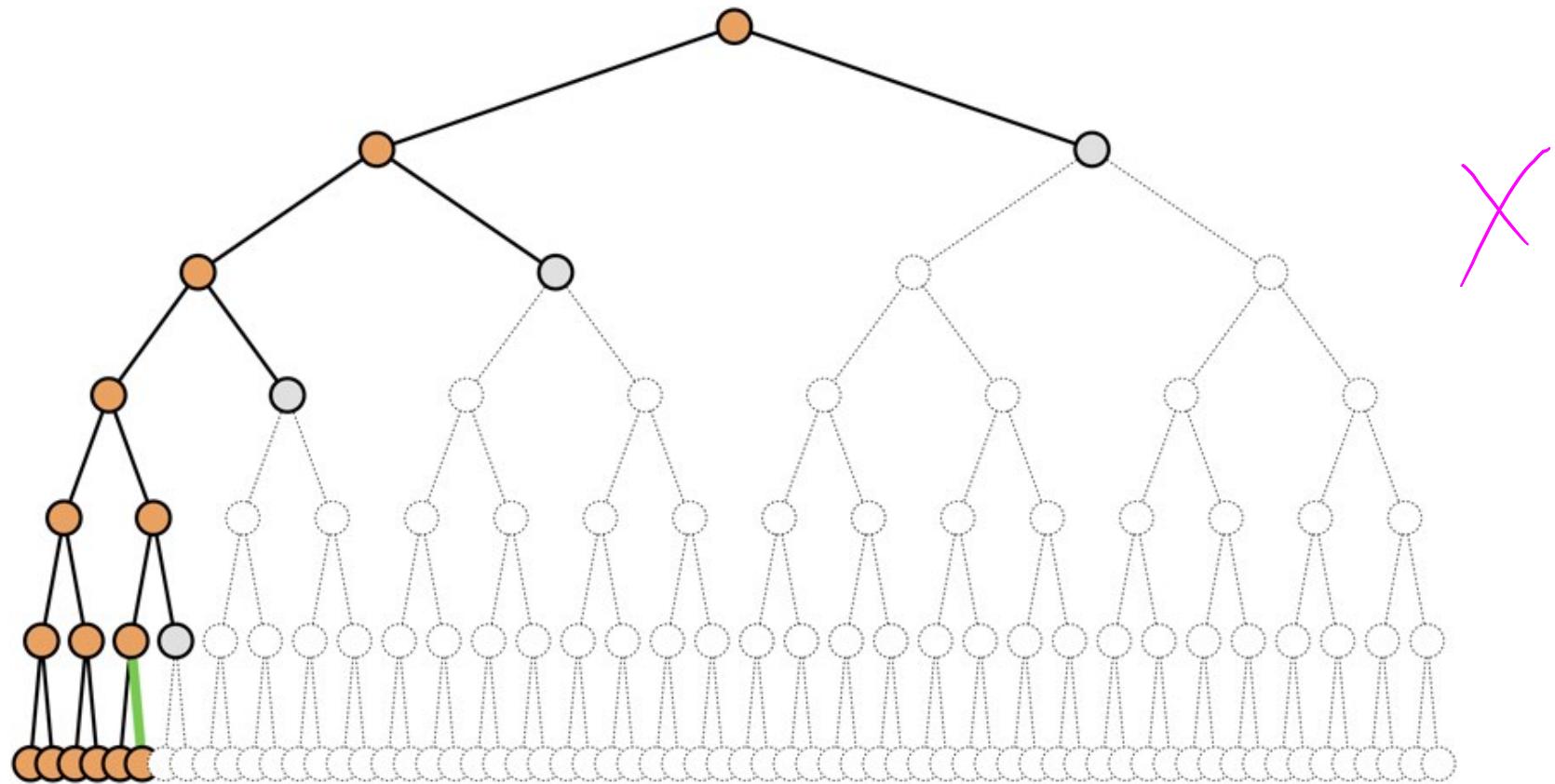
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



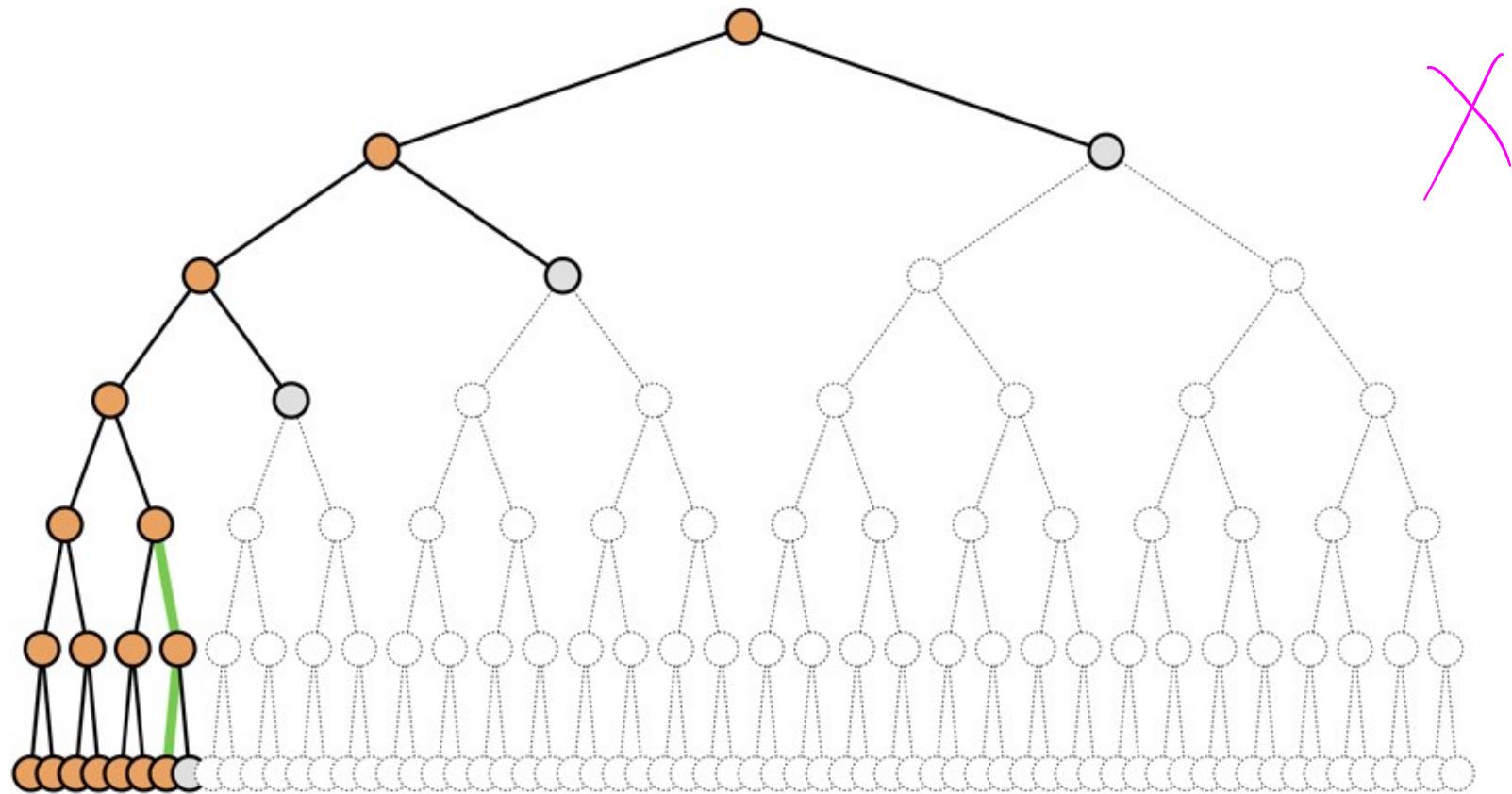
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



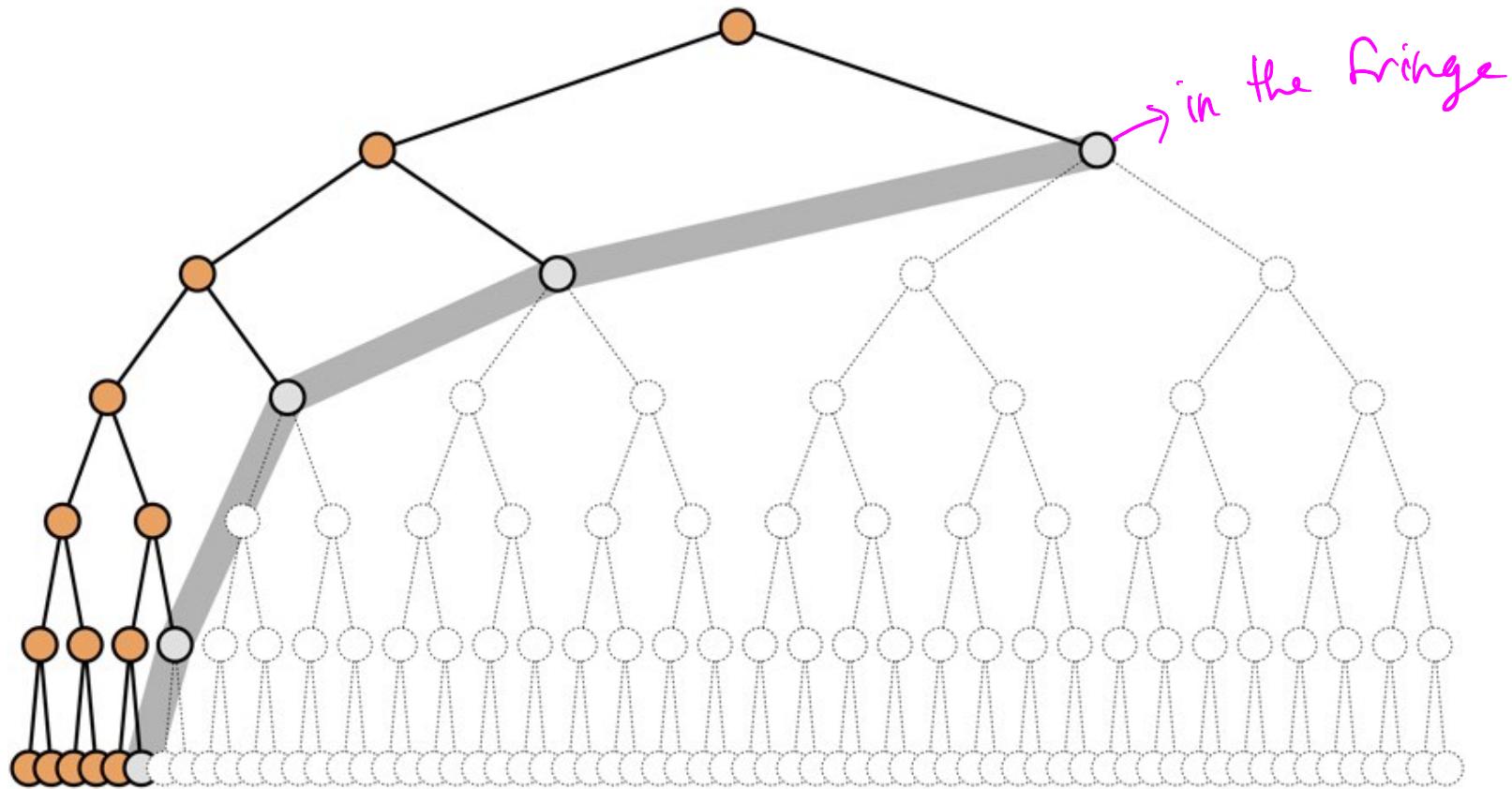
Searches branch by branch ...
... with each branch pursued to maximum depth.

DFS



Searches branch by branch ...
... with each branch pursued to maximum depth.

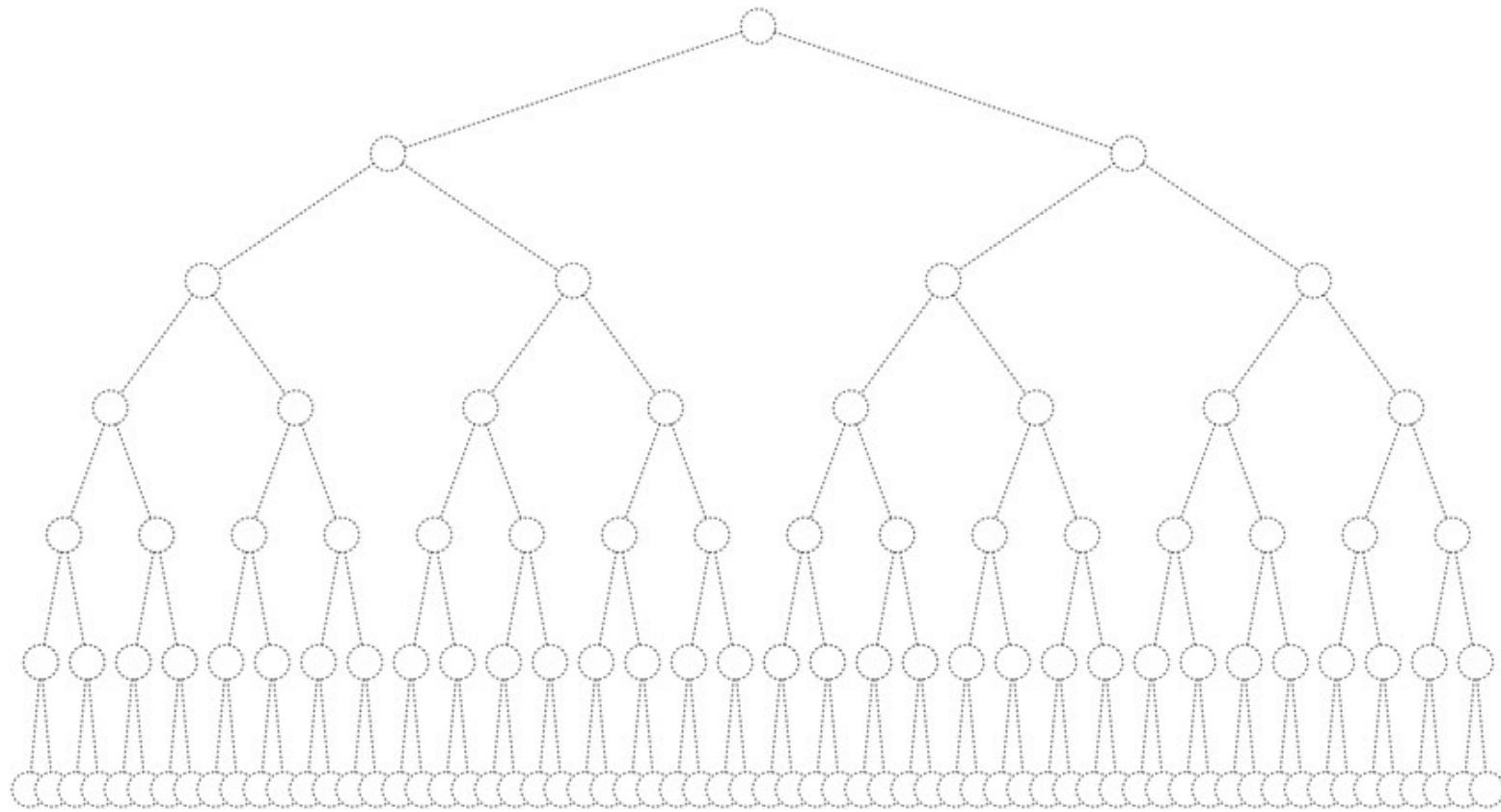
DFS



The frontier consists of unexplored siblings of all ancestors.
Search proceeds by exhausting one branch at a time.

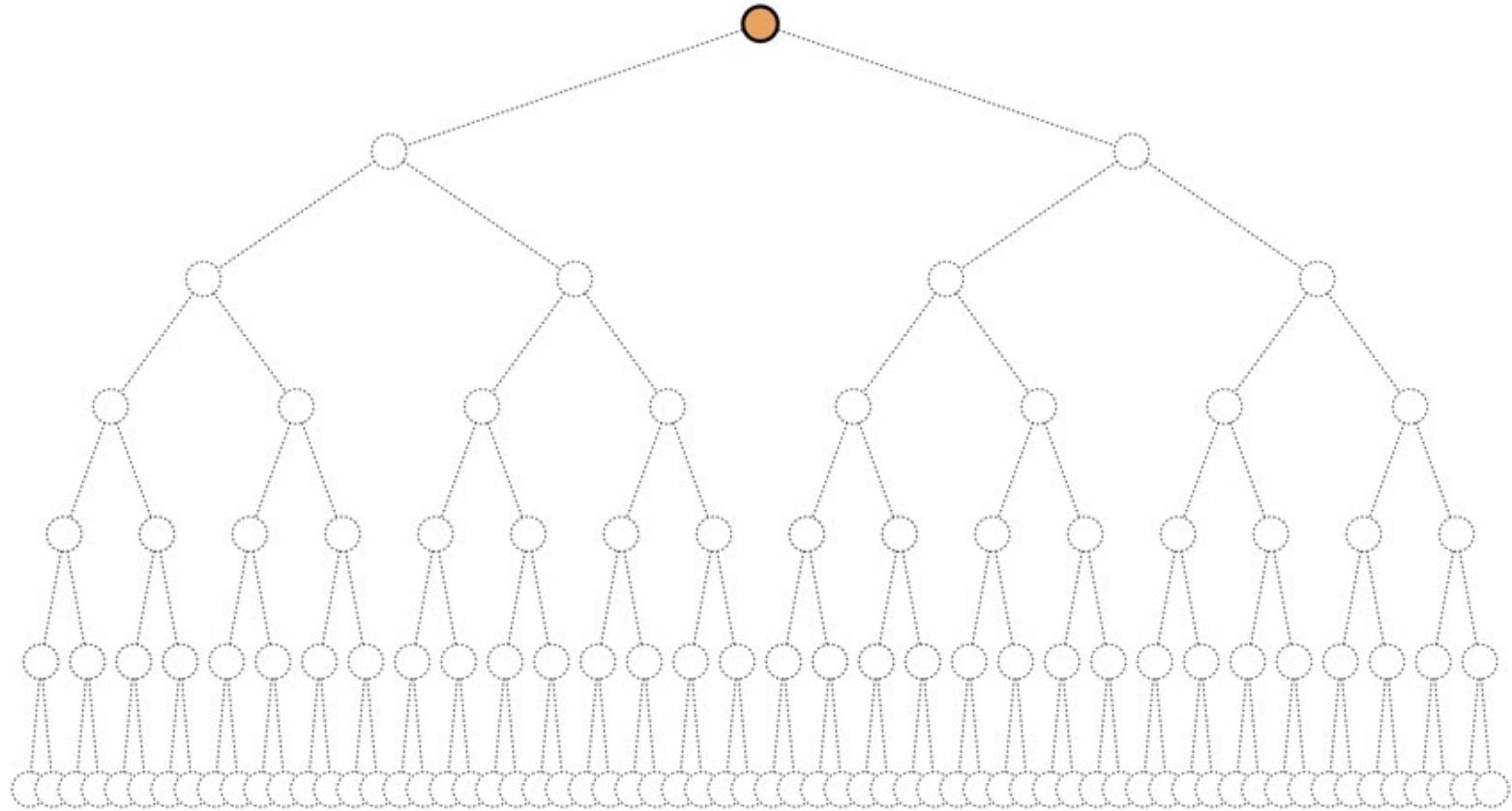
Search proceeds by exhausting one branch at a time.

IDS



Searches subtree by subtree ... instead of branch by branch
... with each subtree increasing by depth limit.

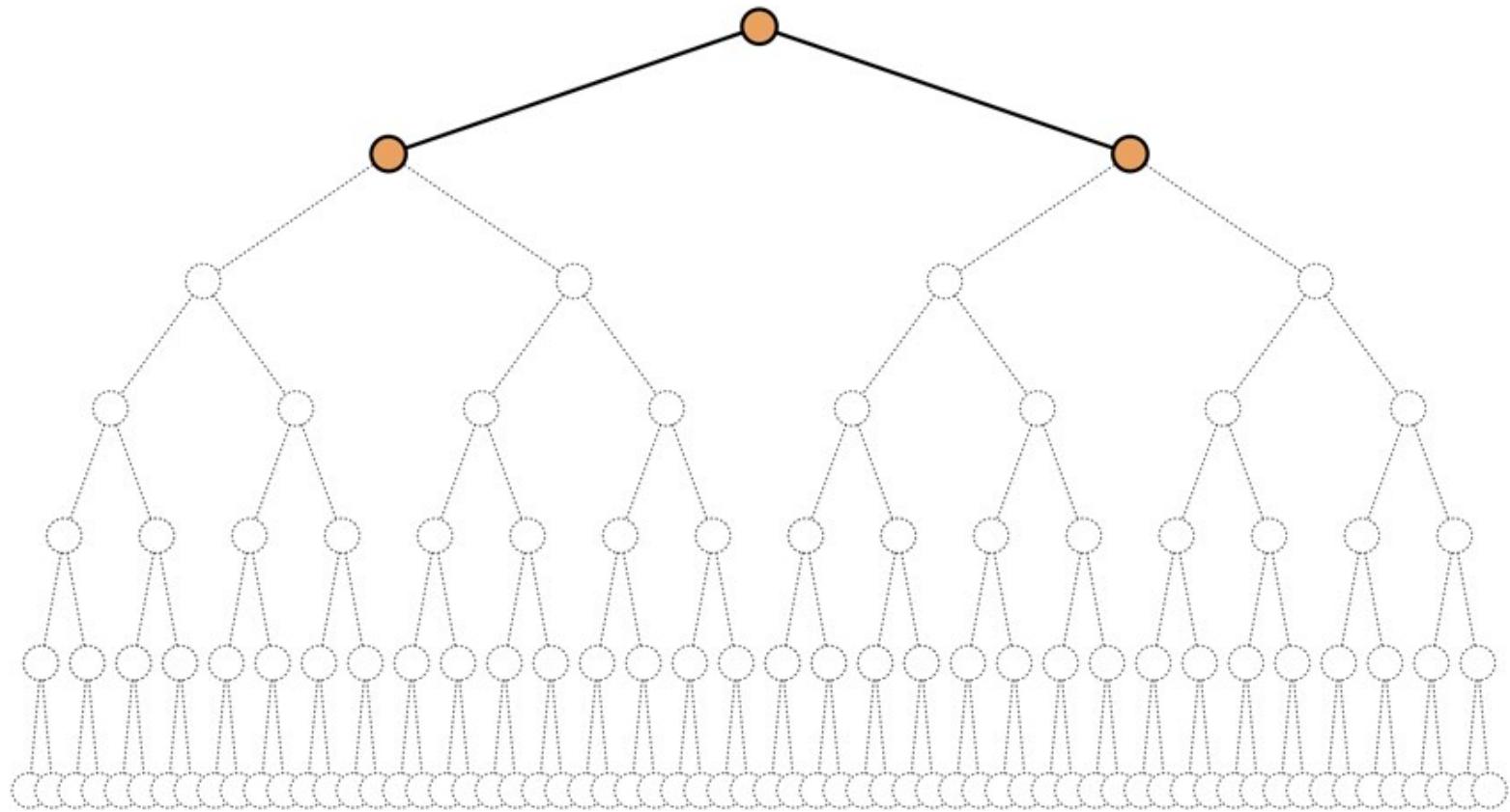
IDS



Searches subtree by subtree ...
... with each subtree increasing by depth limit.



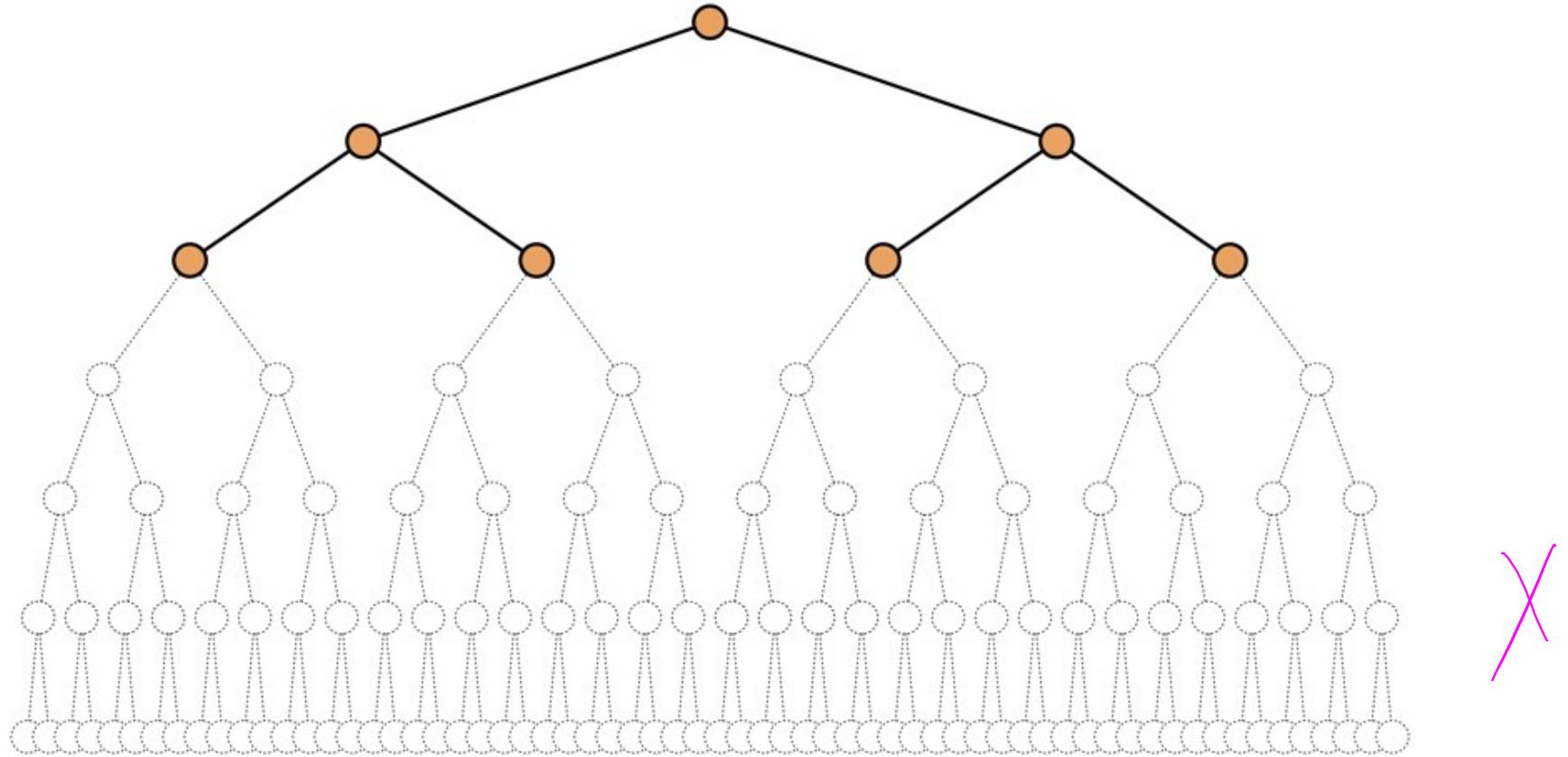
IDS



Searches subtree by subtree ...
... with each subtree increasing by depth limit.

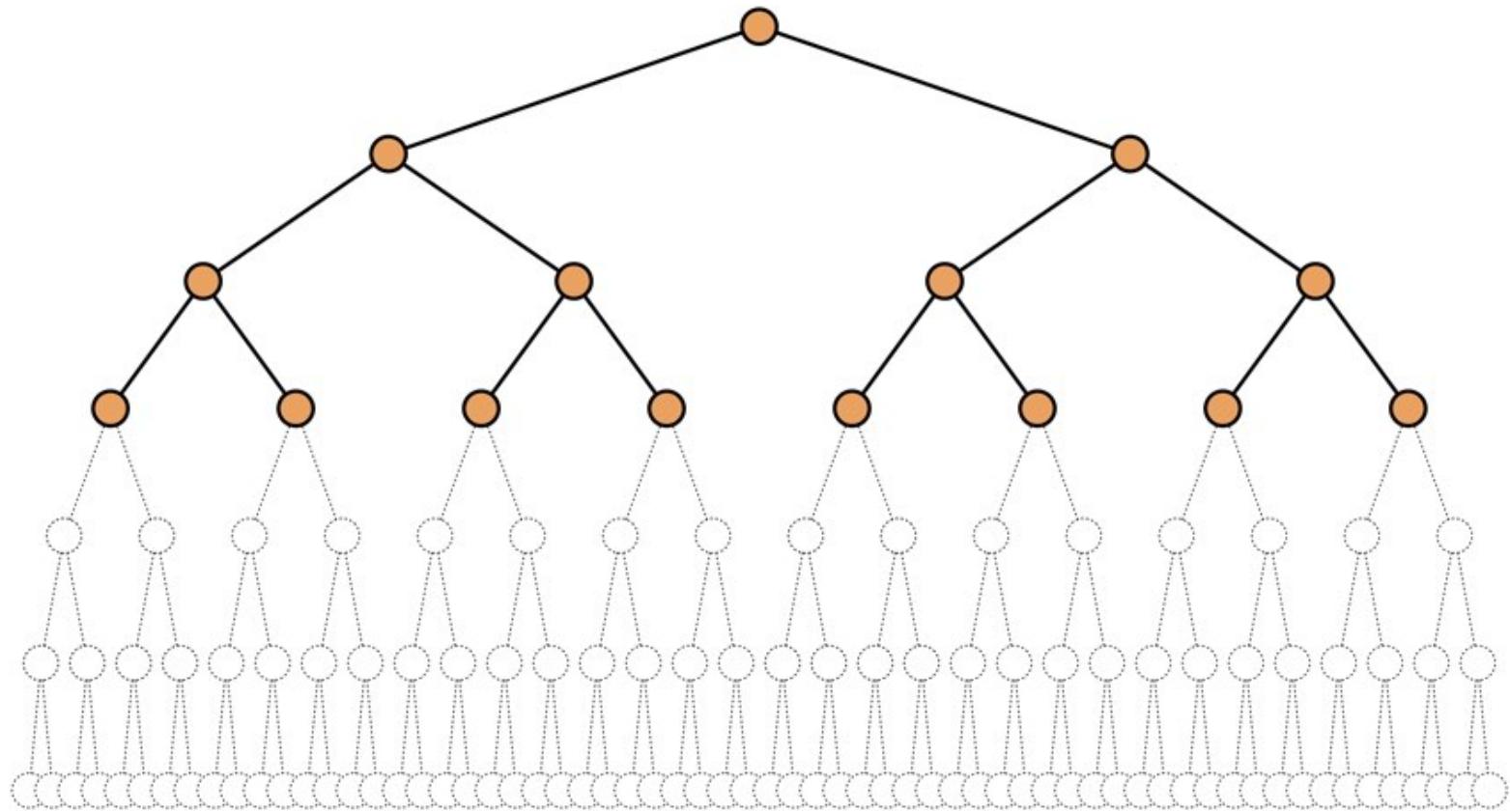


IDS



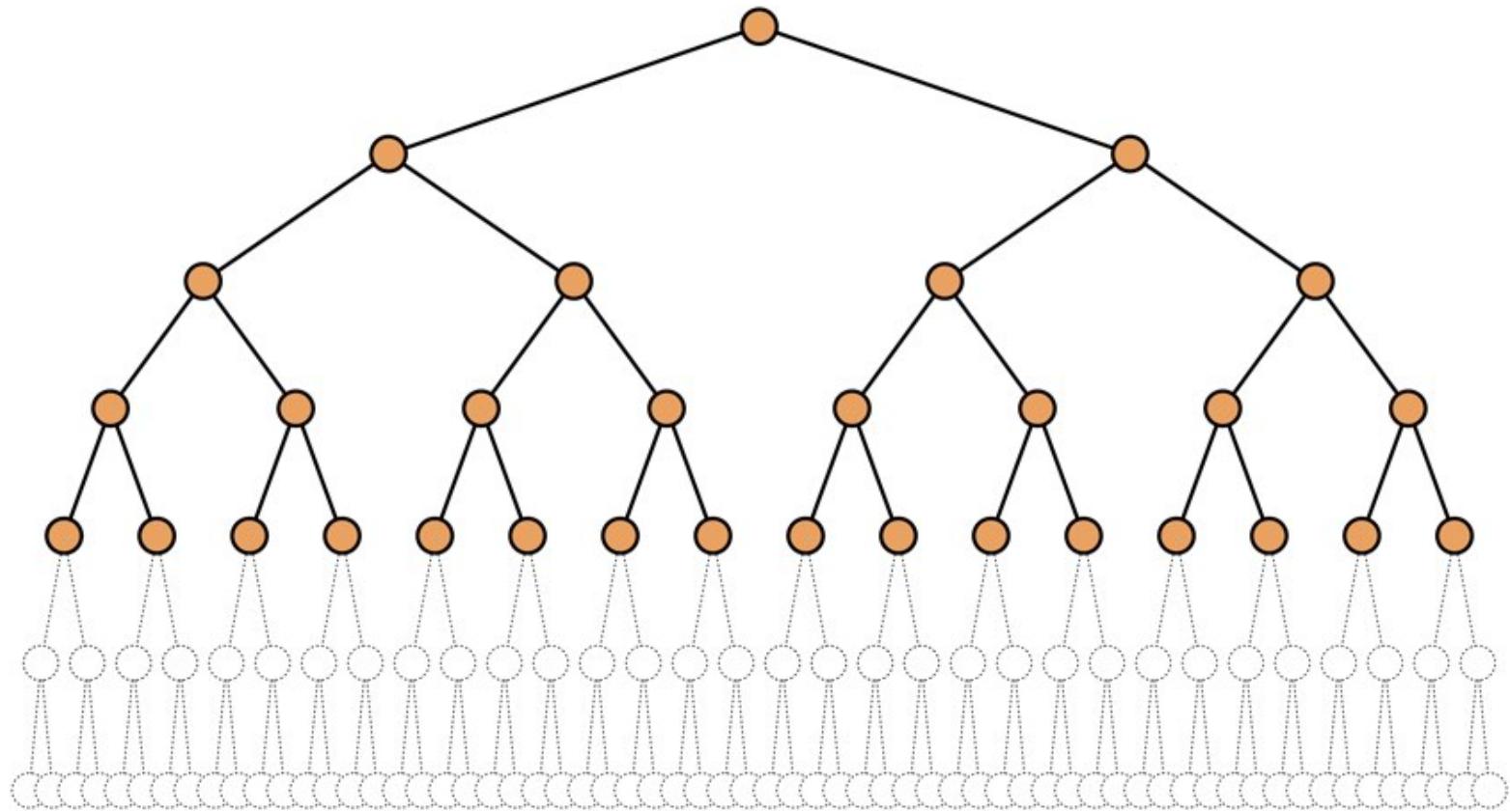
Searches subtree by subtree ...
... with each subtree increasing by depth limit.

IDS



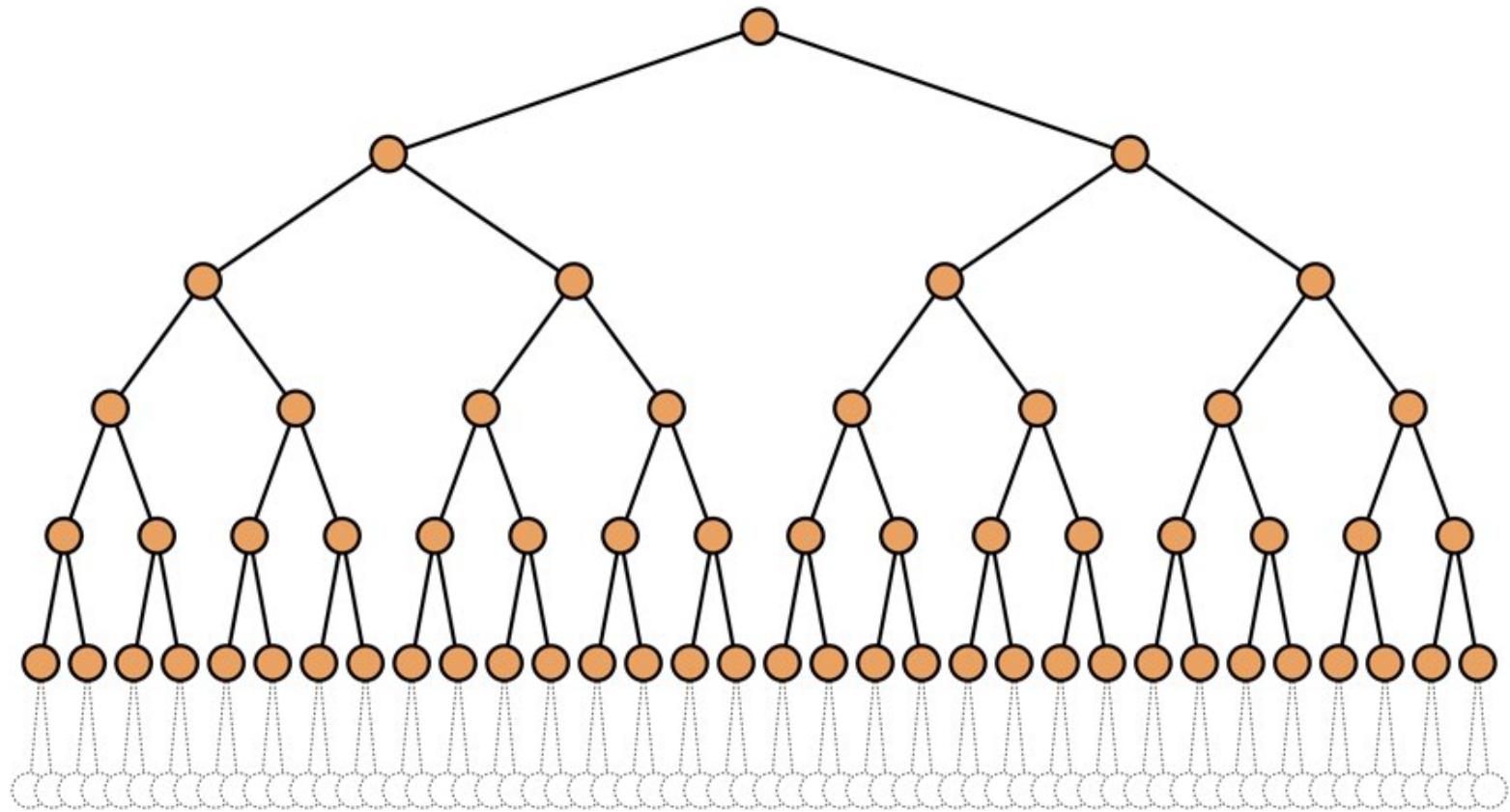
Searches subtree by subtree ...
... with each subtree increasing by depth limit.

IDS



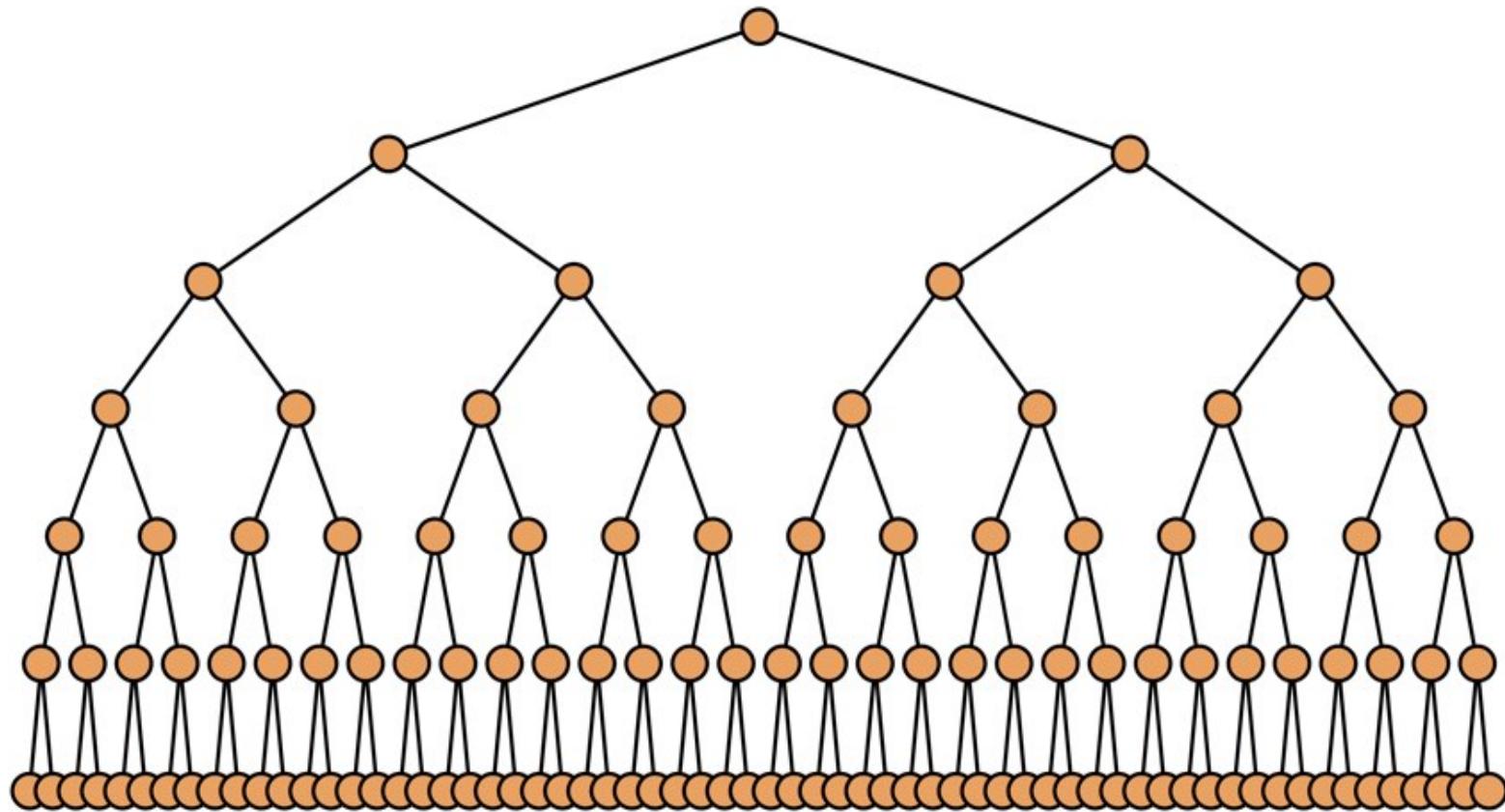
Searches subtree by subtree ...
... with each subtree increasing by depth limit.

IDS



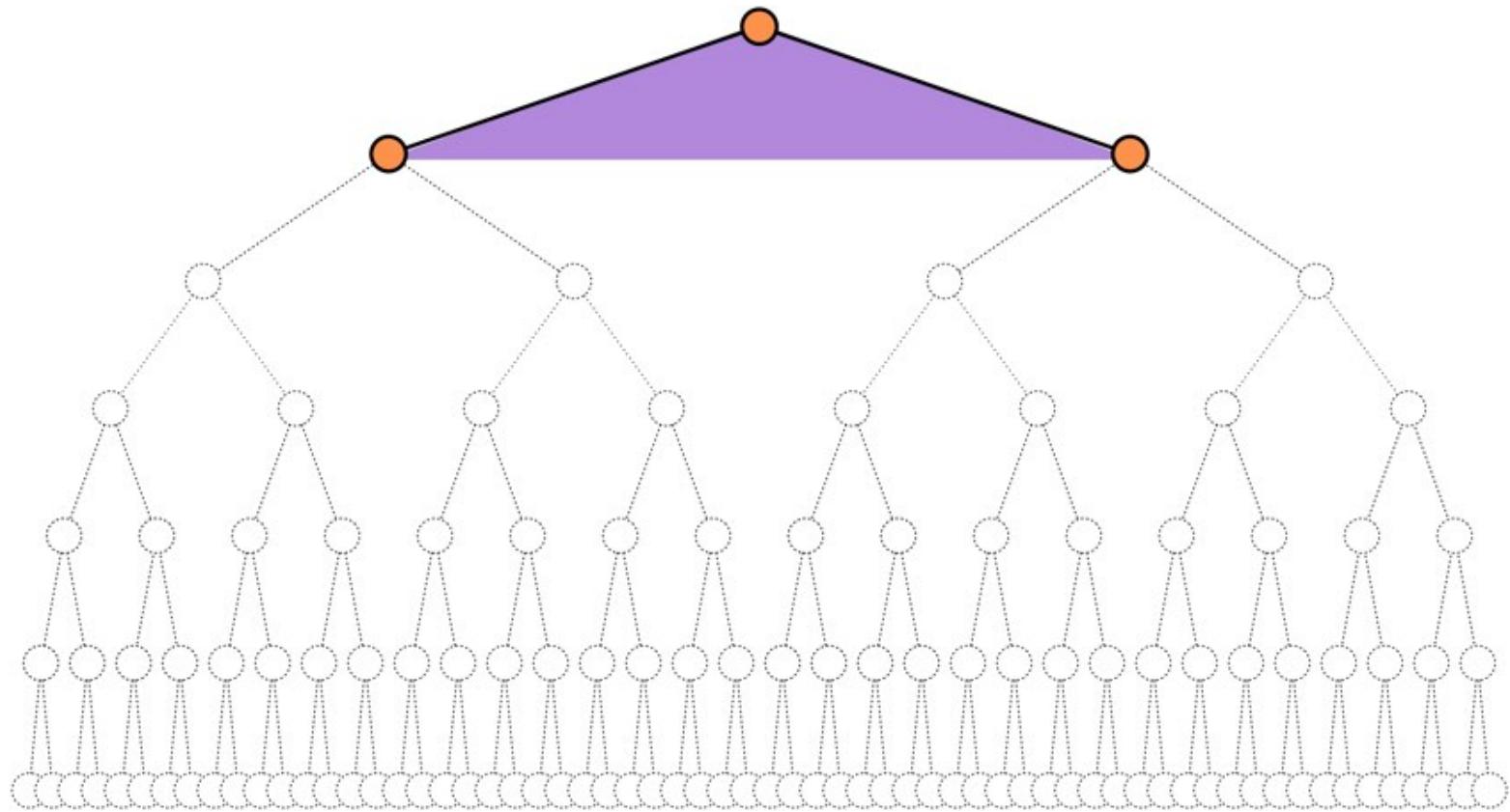
Searches subtree by subtree ...
... with each subtree increasing by depth limit.

IDS



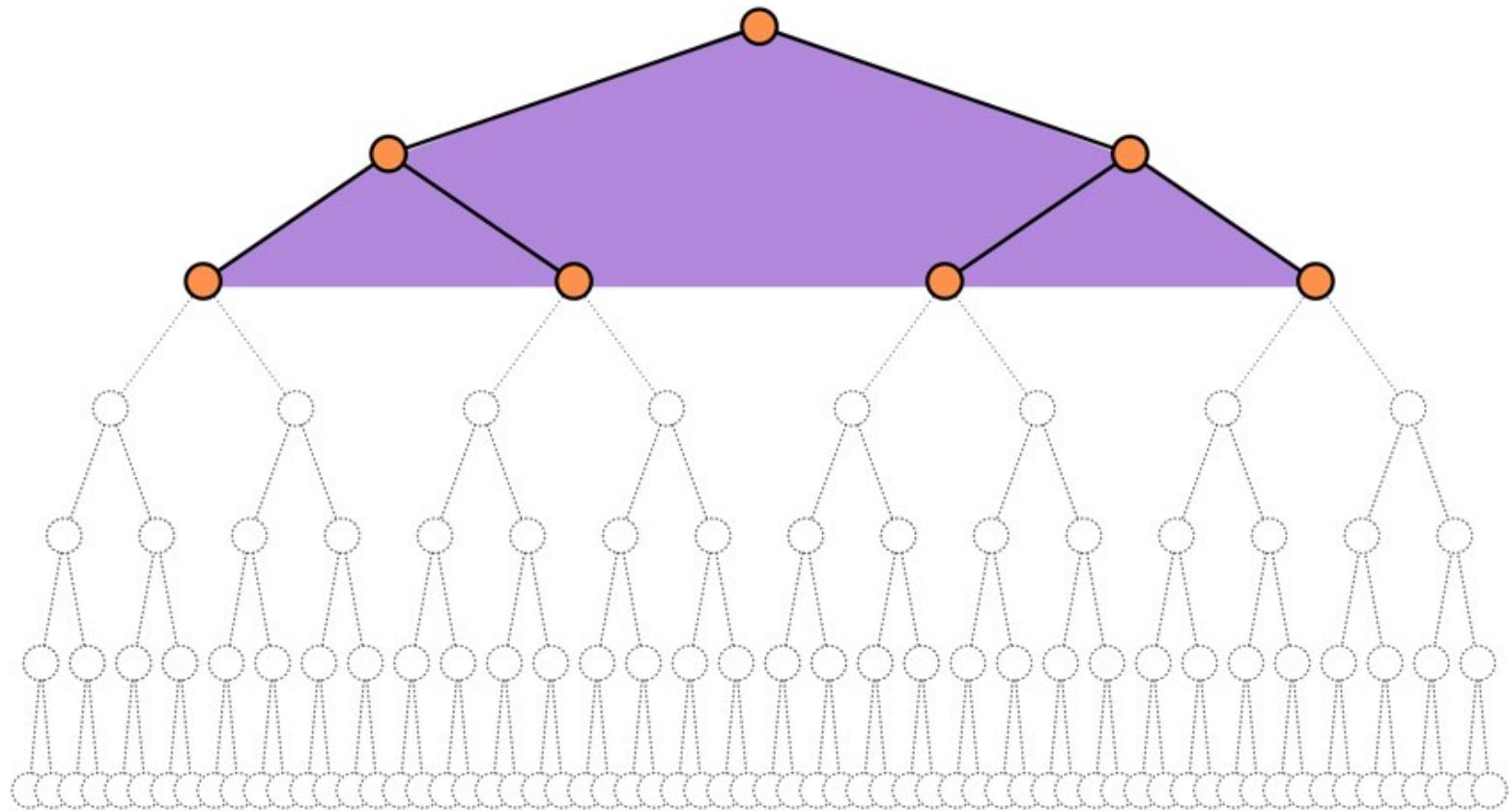
Searches subtree by subtree ...
... with each subtree increasing by depth limit.

IDS



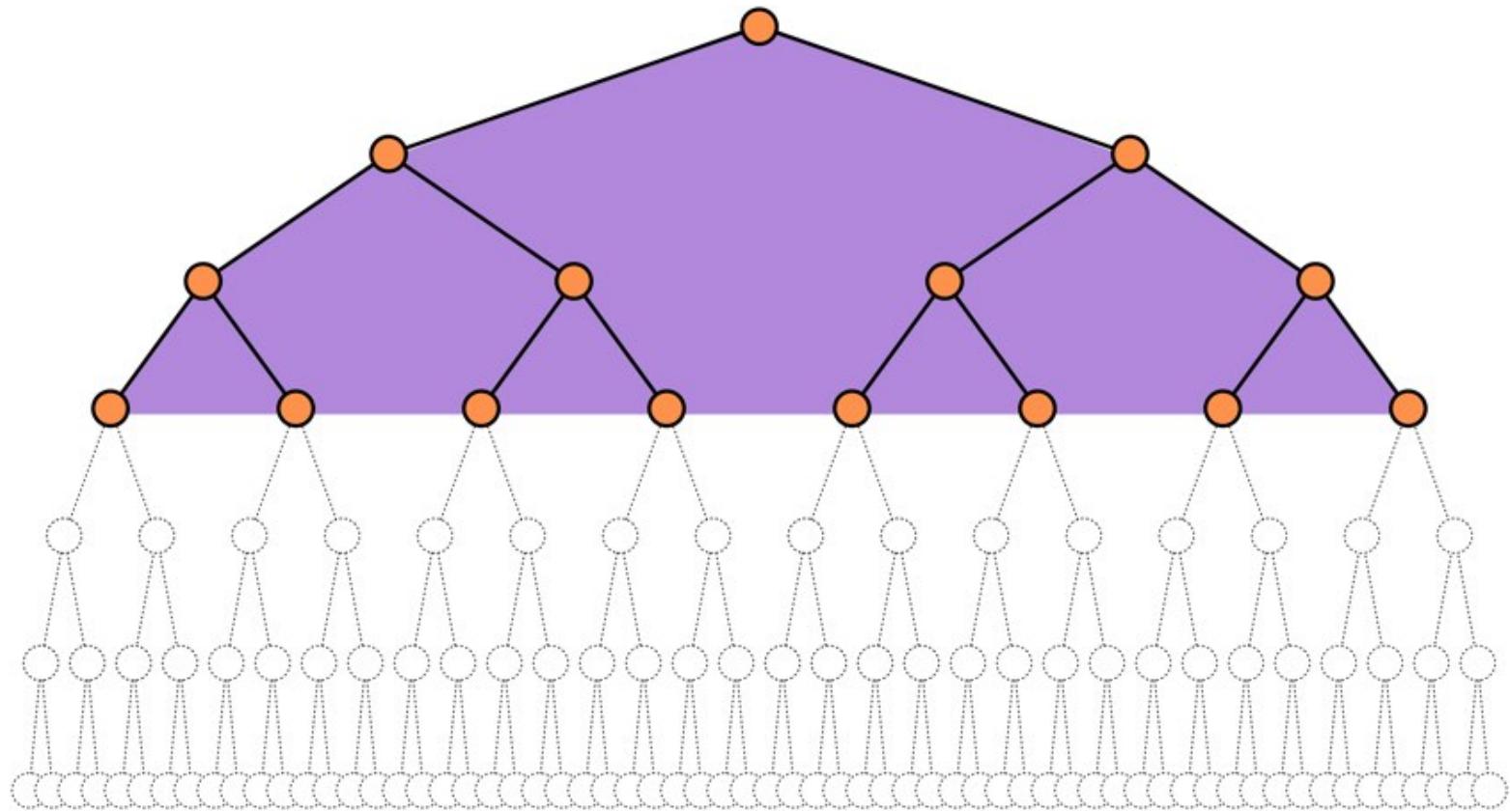
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

IDS



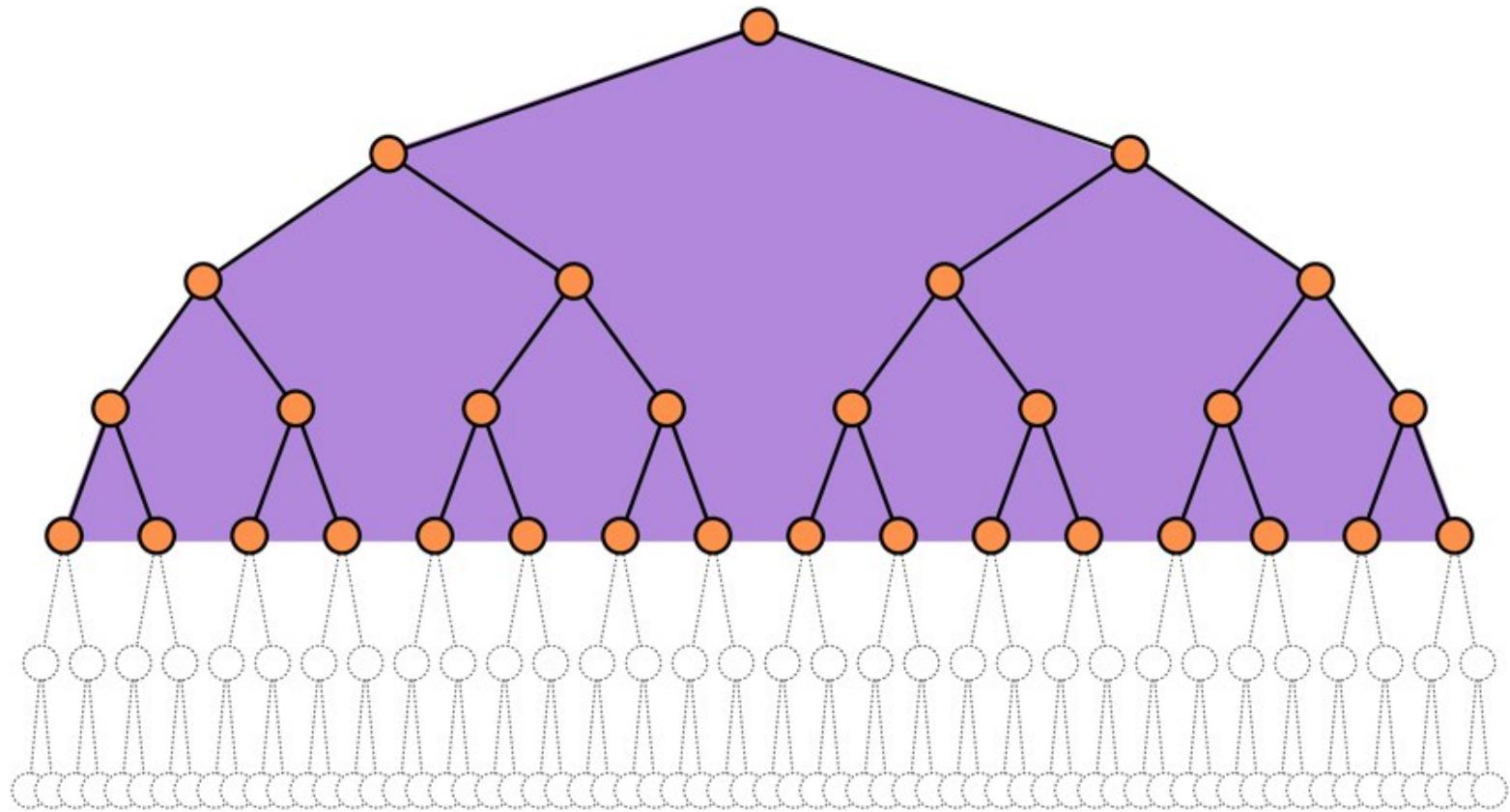
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

IDS

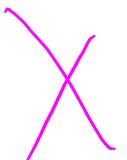


Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

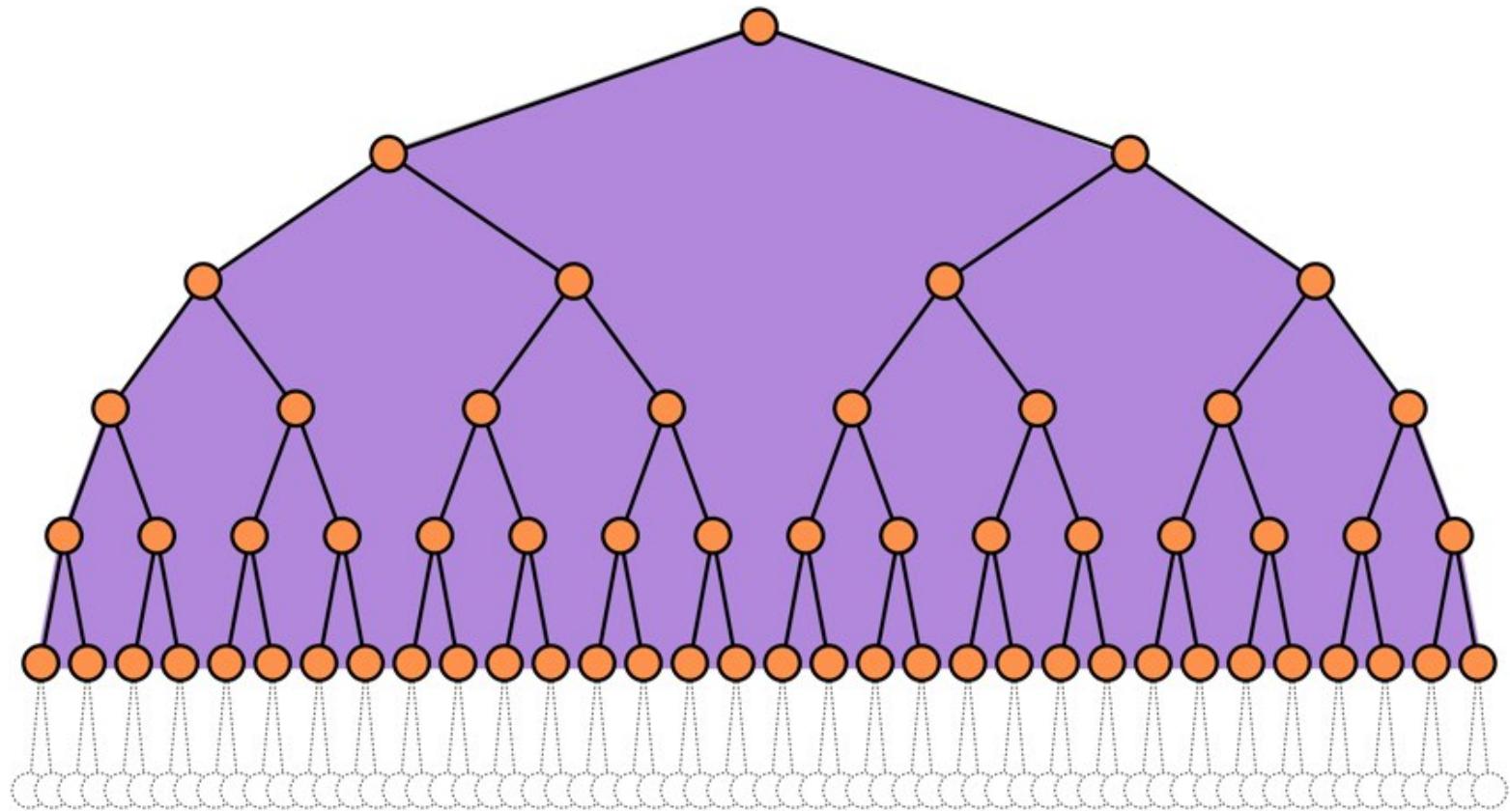
IDS



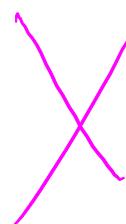
Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.



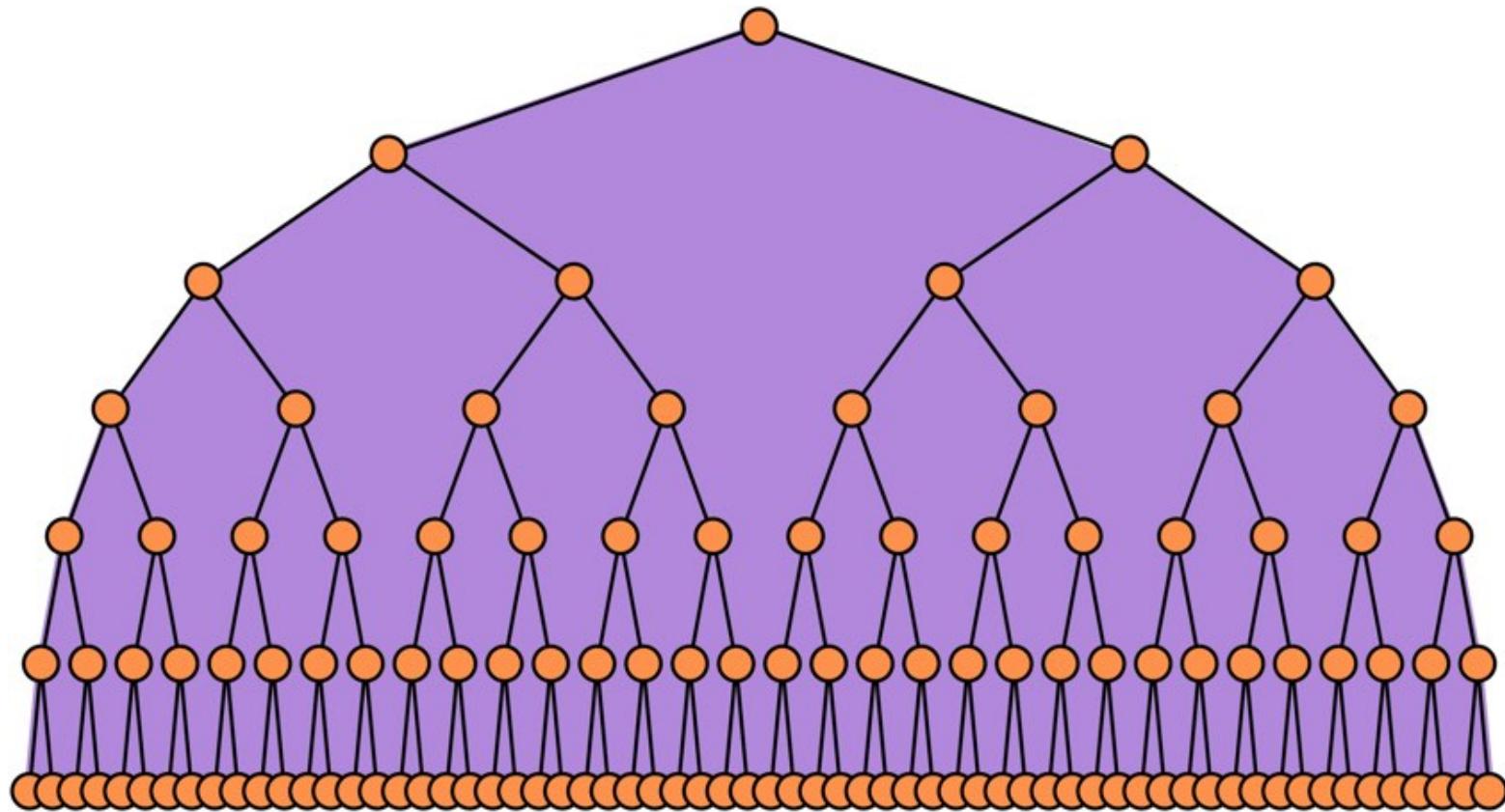
IDS



Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.



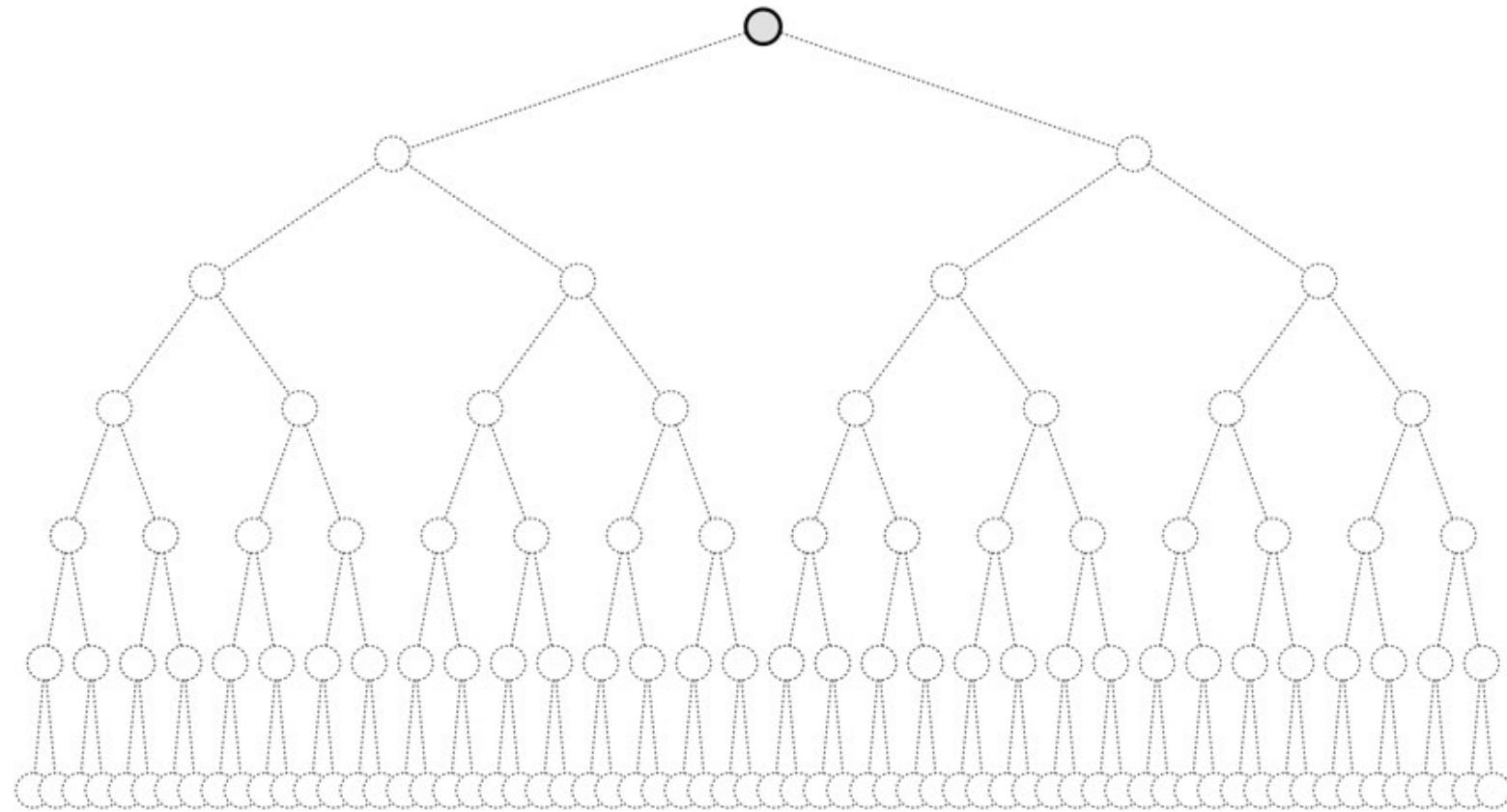
IDS



Each iteration is simply an instance of depth-limited search.
Search proceeds by exhausting larger and larger subtrees.

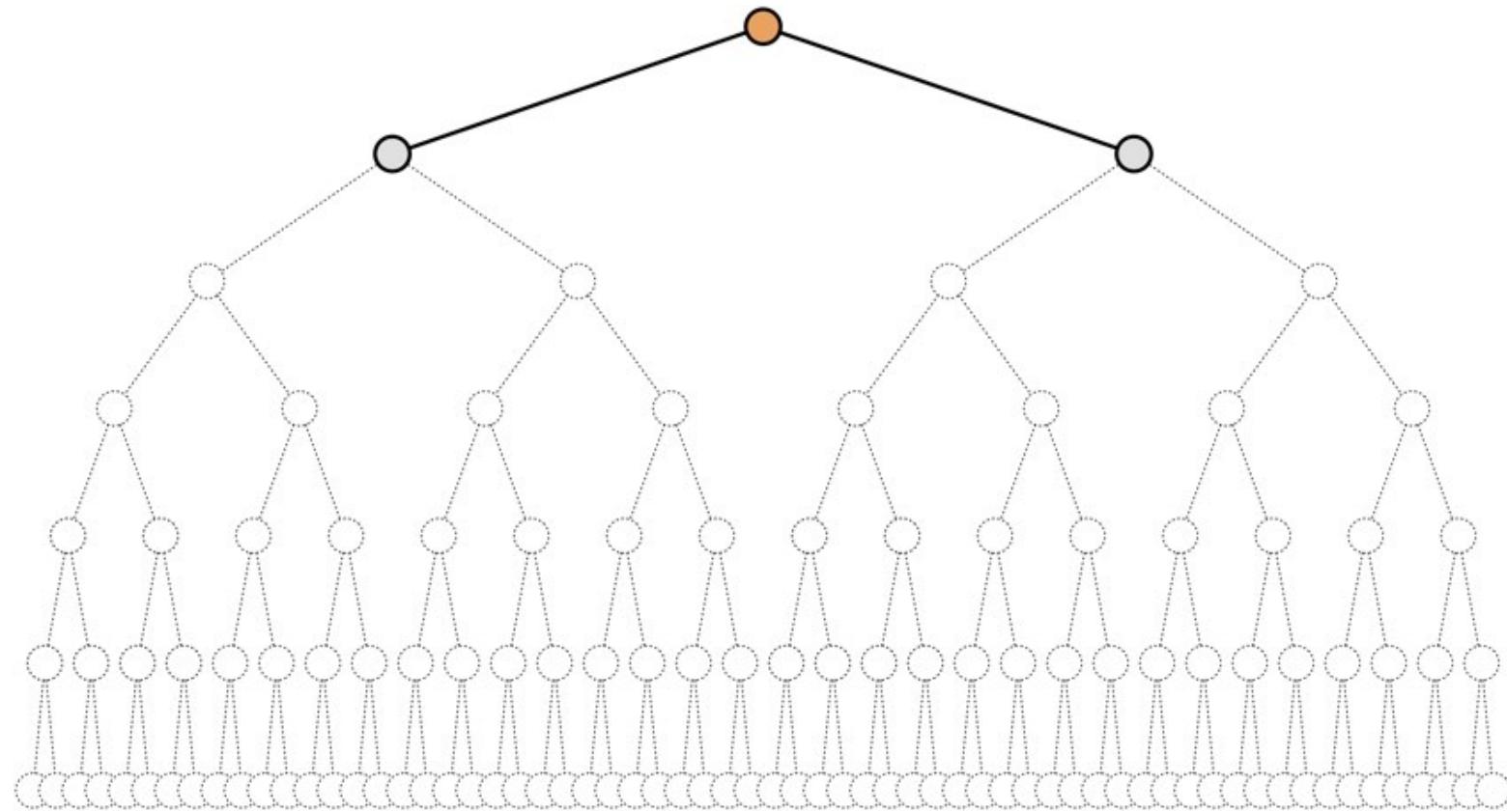
nothing in the fringe

BFS



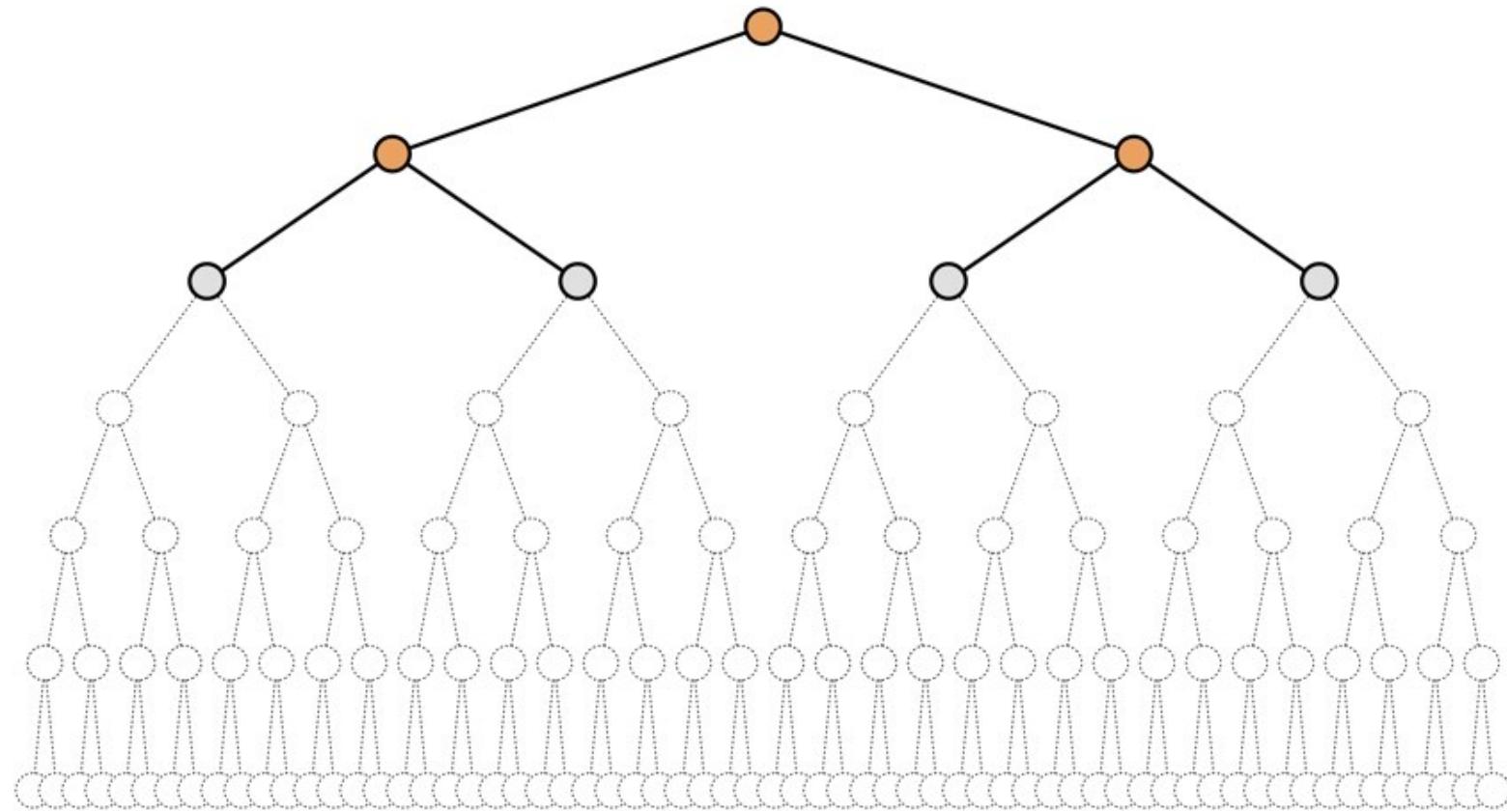
Searches layer by layer ...
... with each layer organized by node depth.

BFS



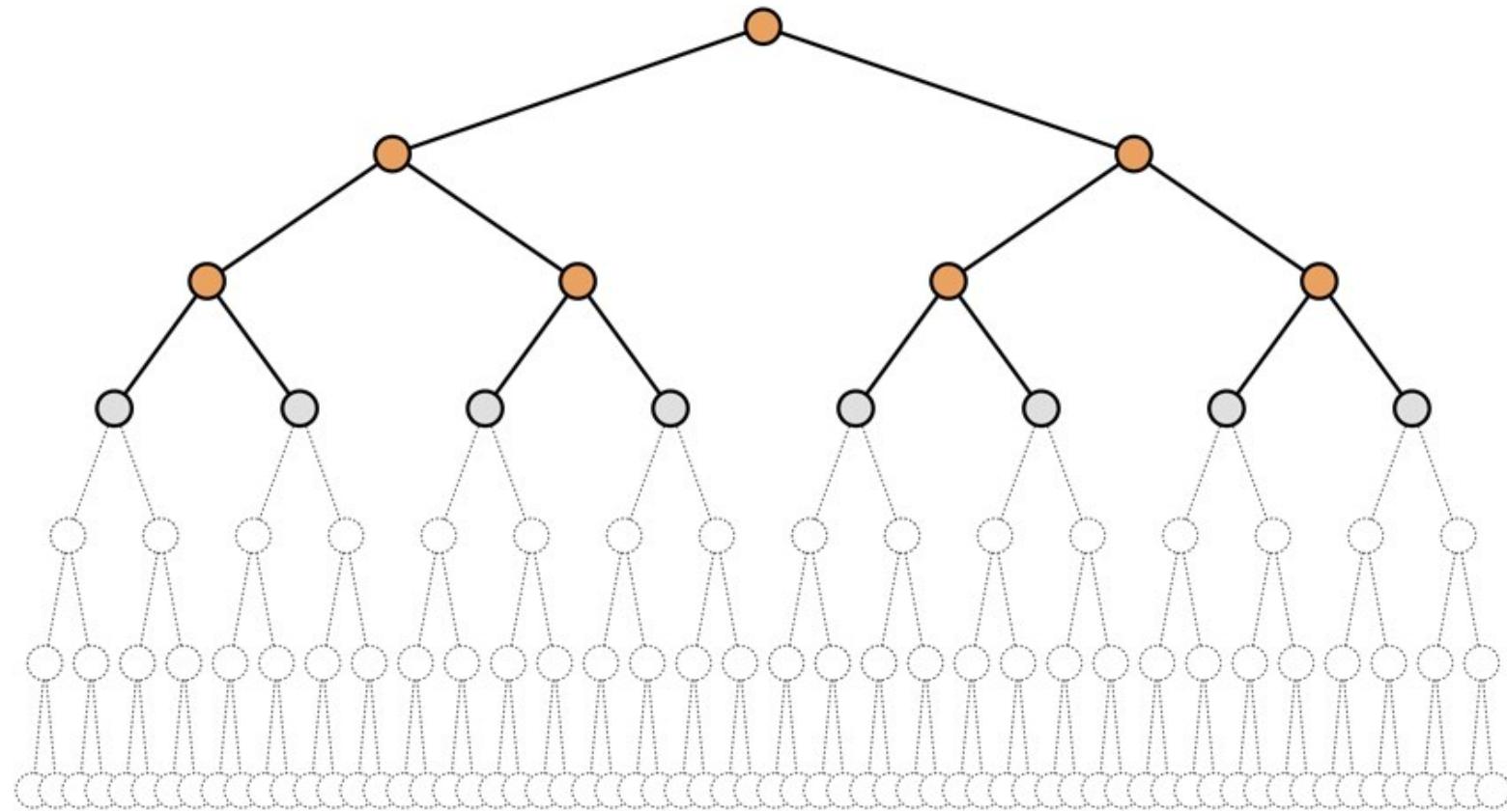
Searches layer by layer ...
... with each layer organized by node depth.

BFS



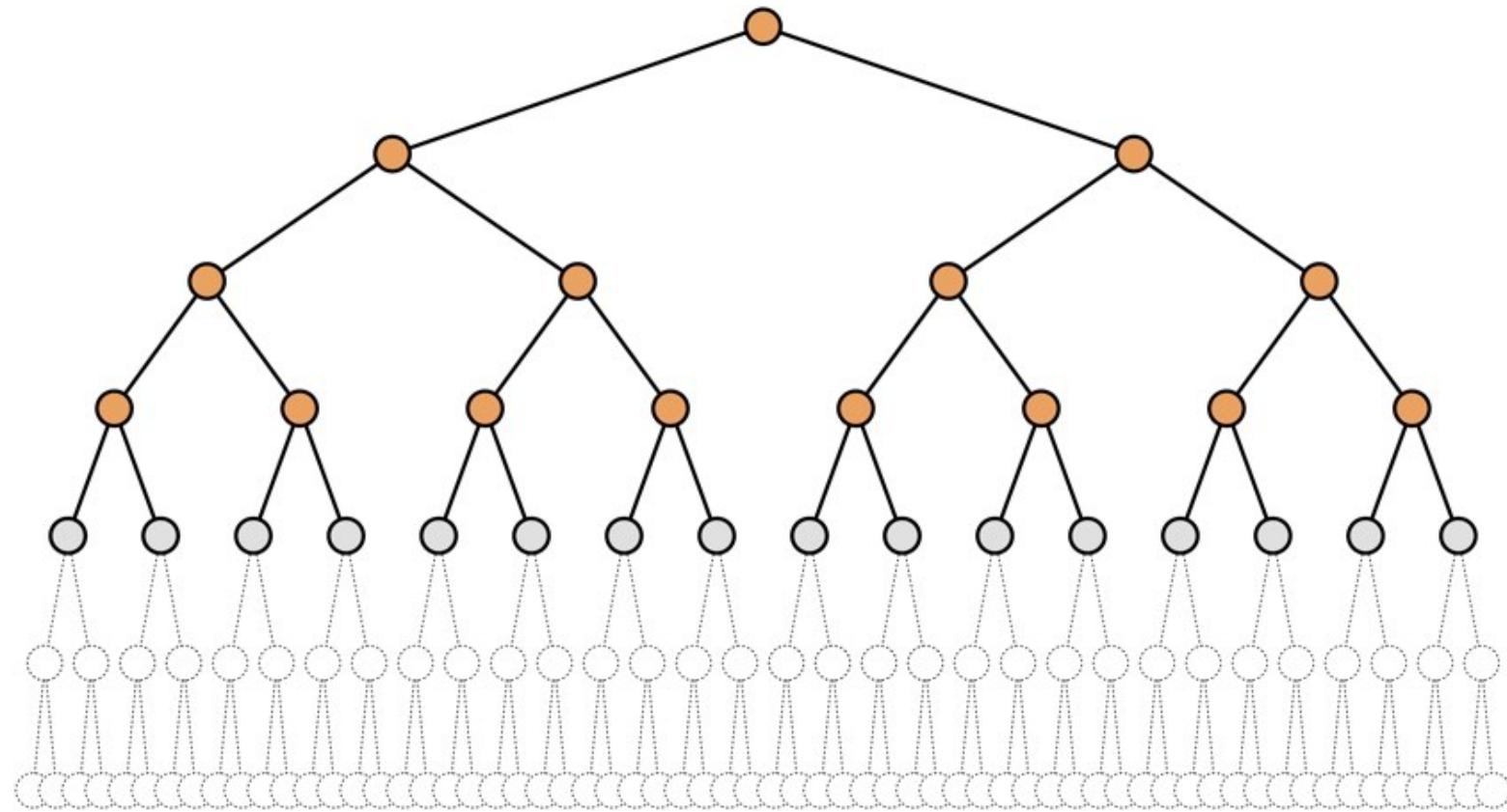
Searches layer by layer ...
... with each layer organized by node depth.

BFS



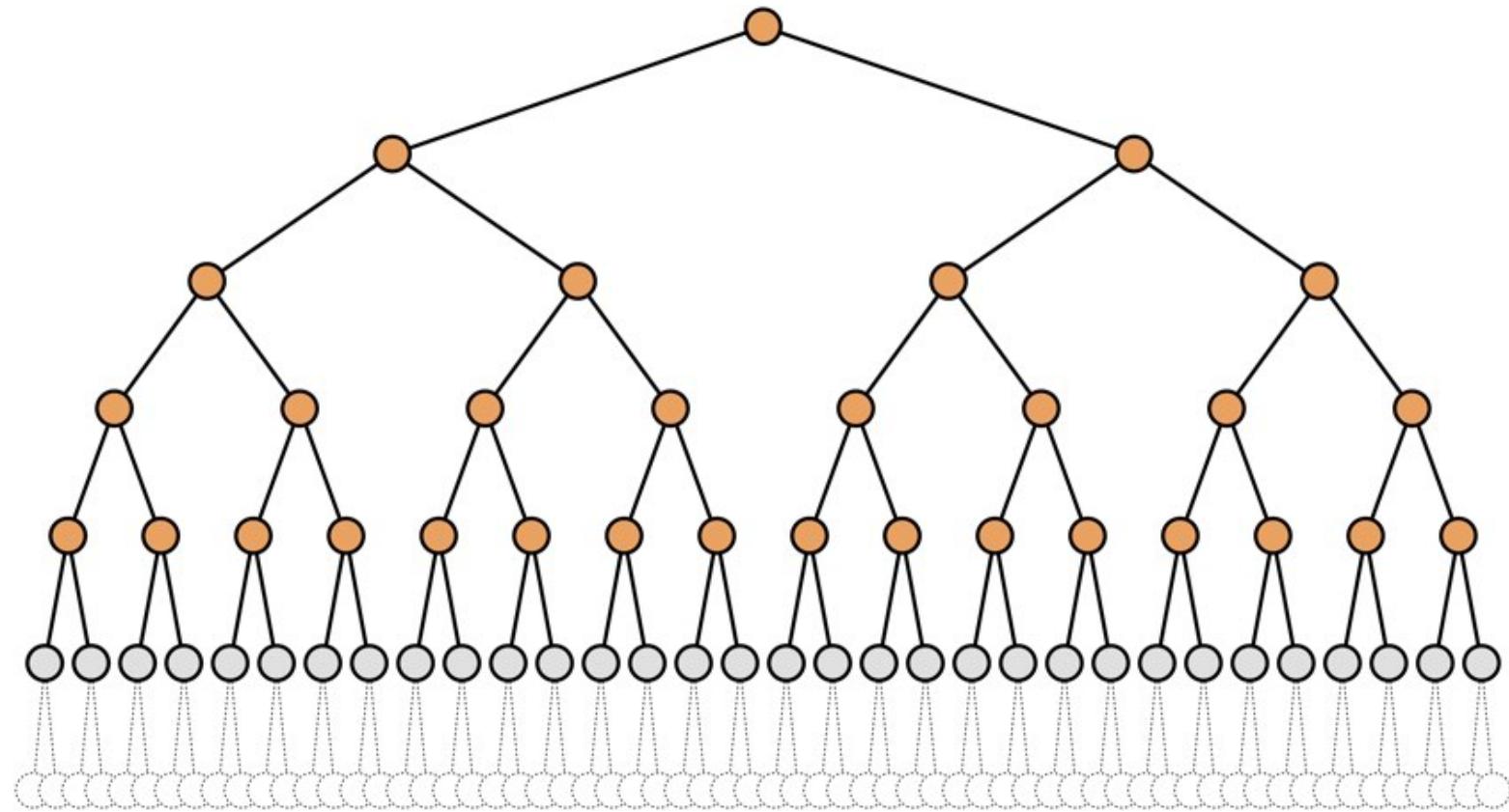
Searches layer by layer ...
... with each layer organized by node depth.

BFS



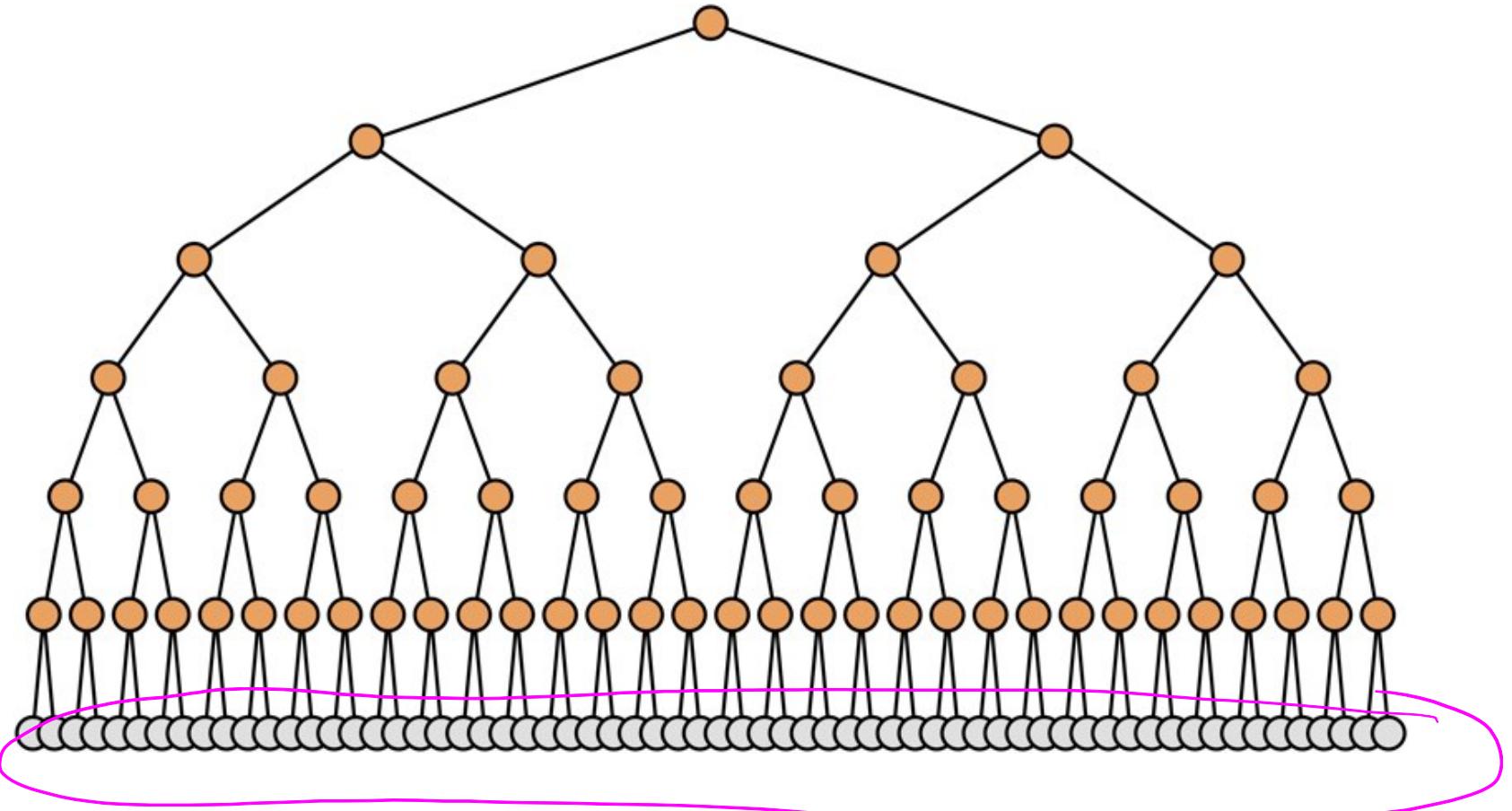
Searches layer by layer ...
... with each layer organized by node depth.

BFS



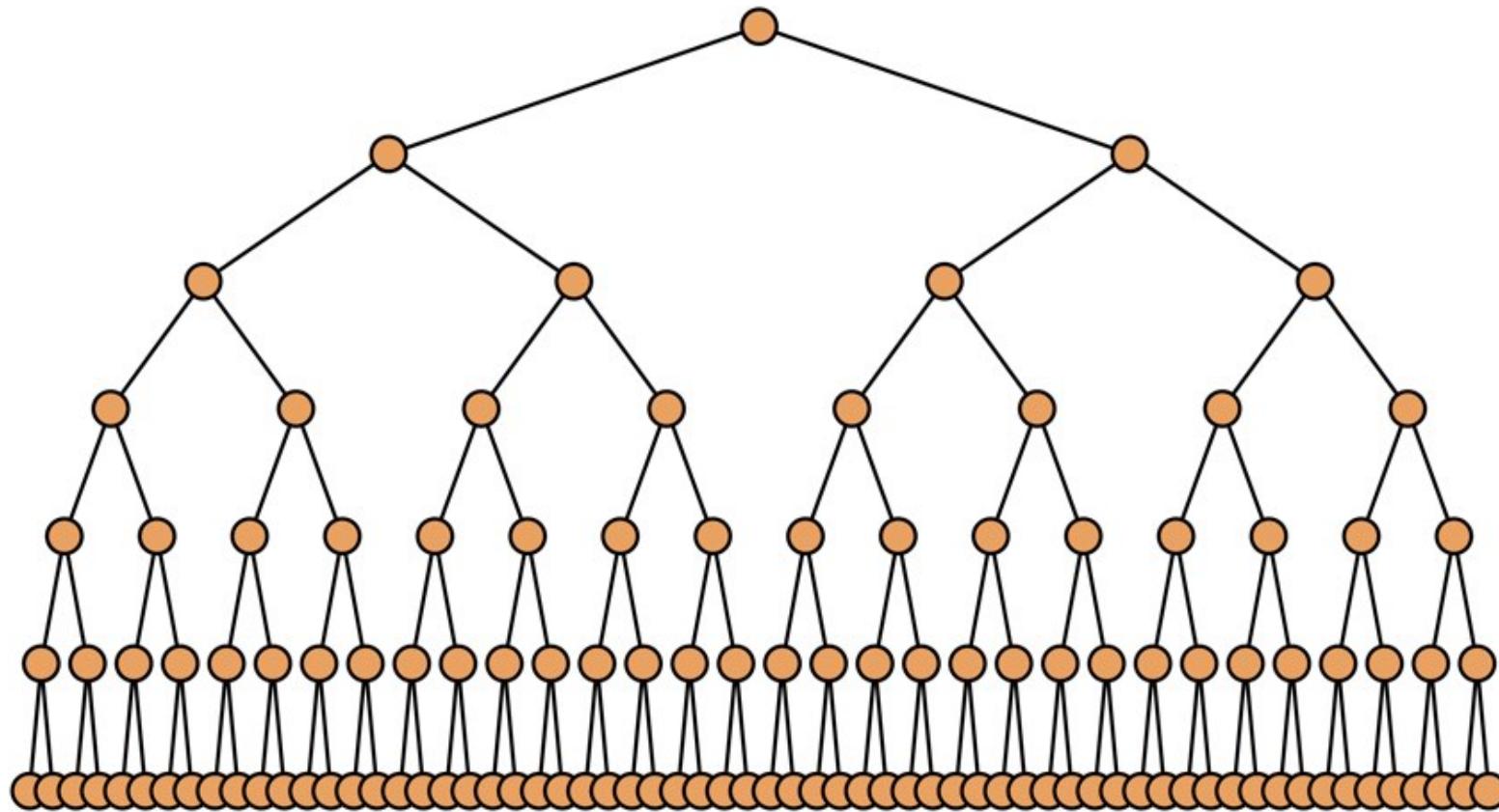
Searches layer by layer ...
... with each layer organized by node depth.

BFS



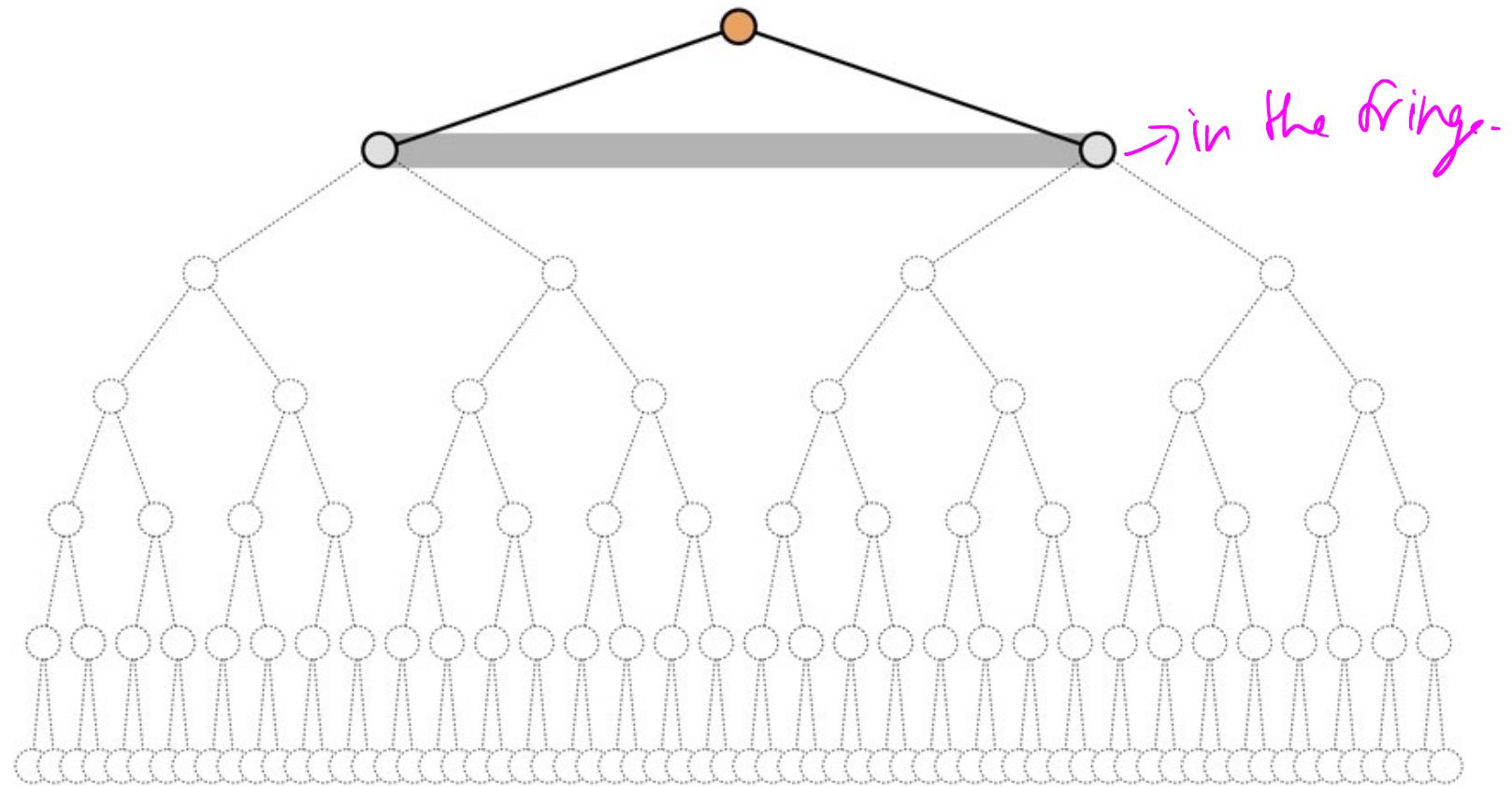
Searches layer by layer ...
... with each layer organized by node depth.

BFS



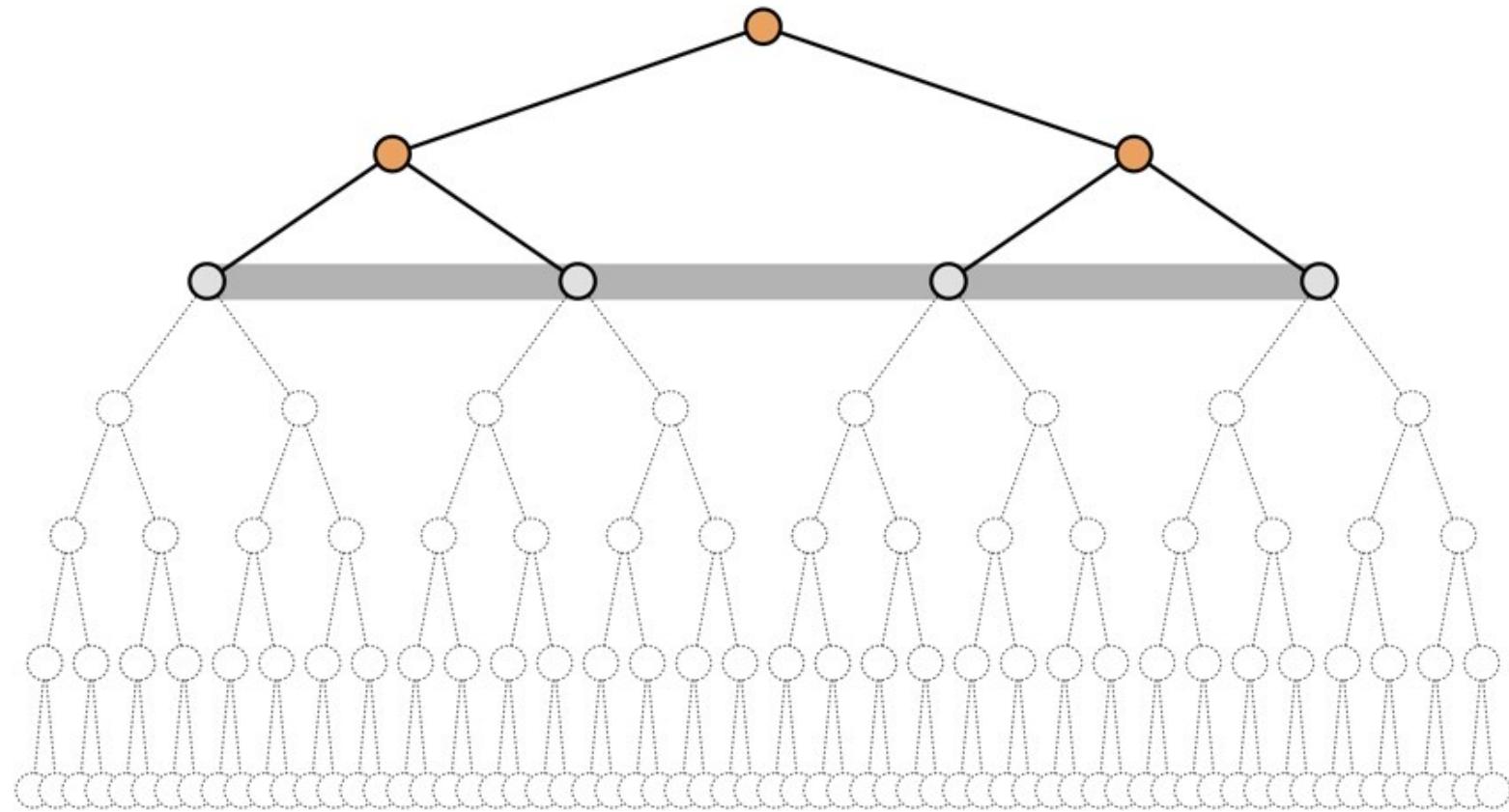
Searches layer by layer ...
... with each layer organized by node depth.

BFS



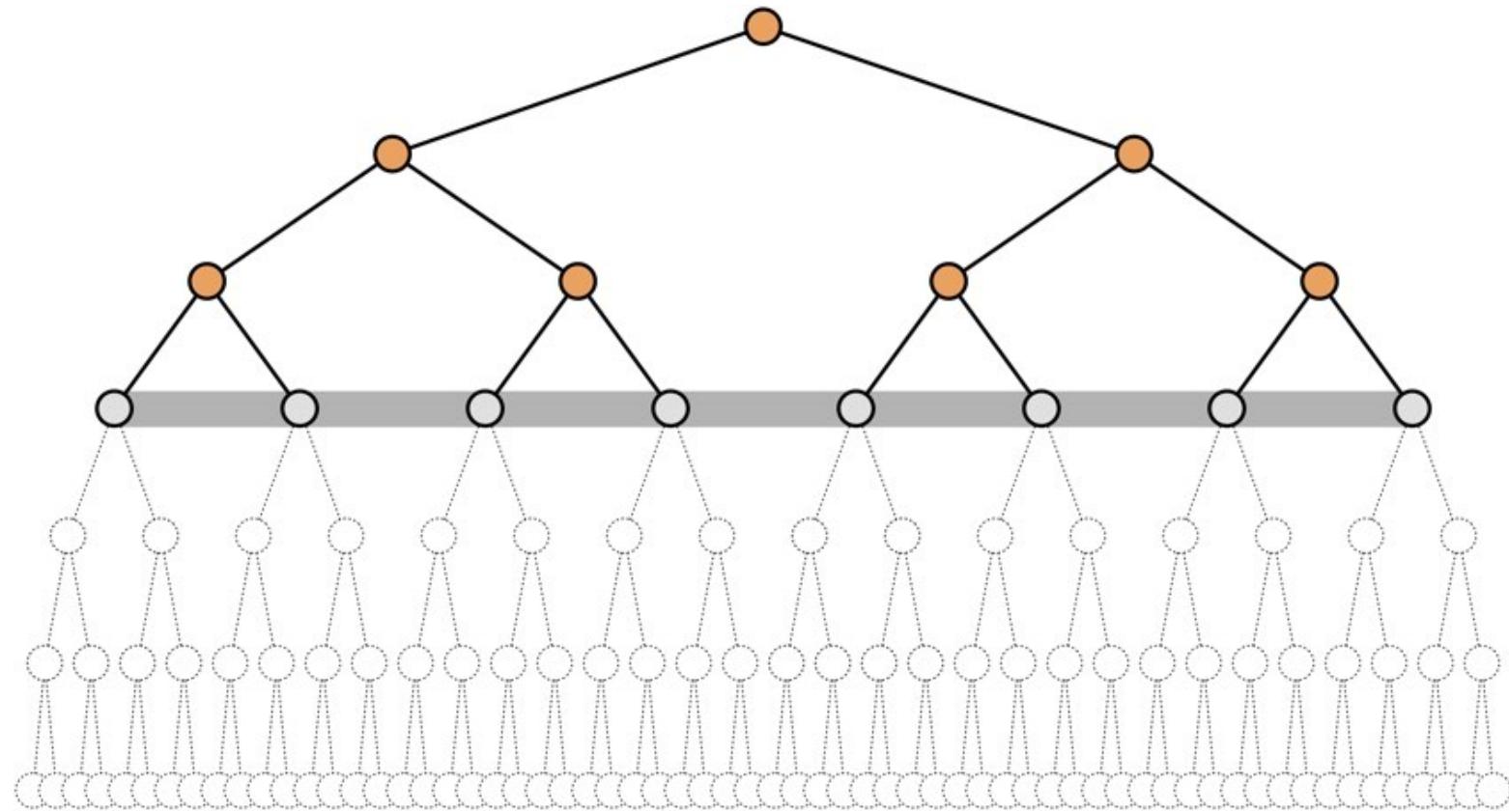
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



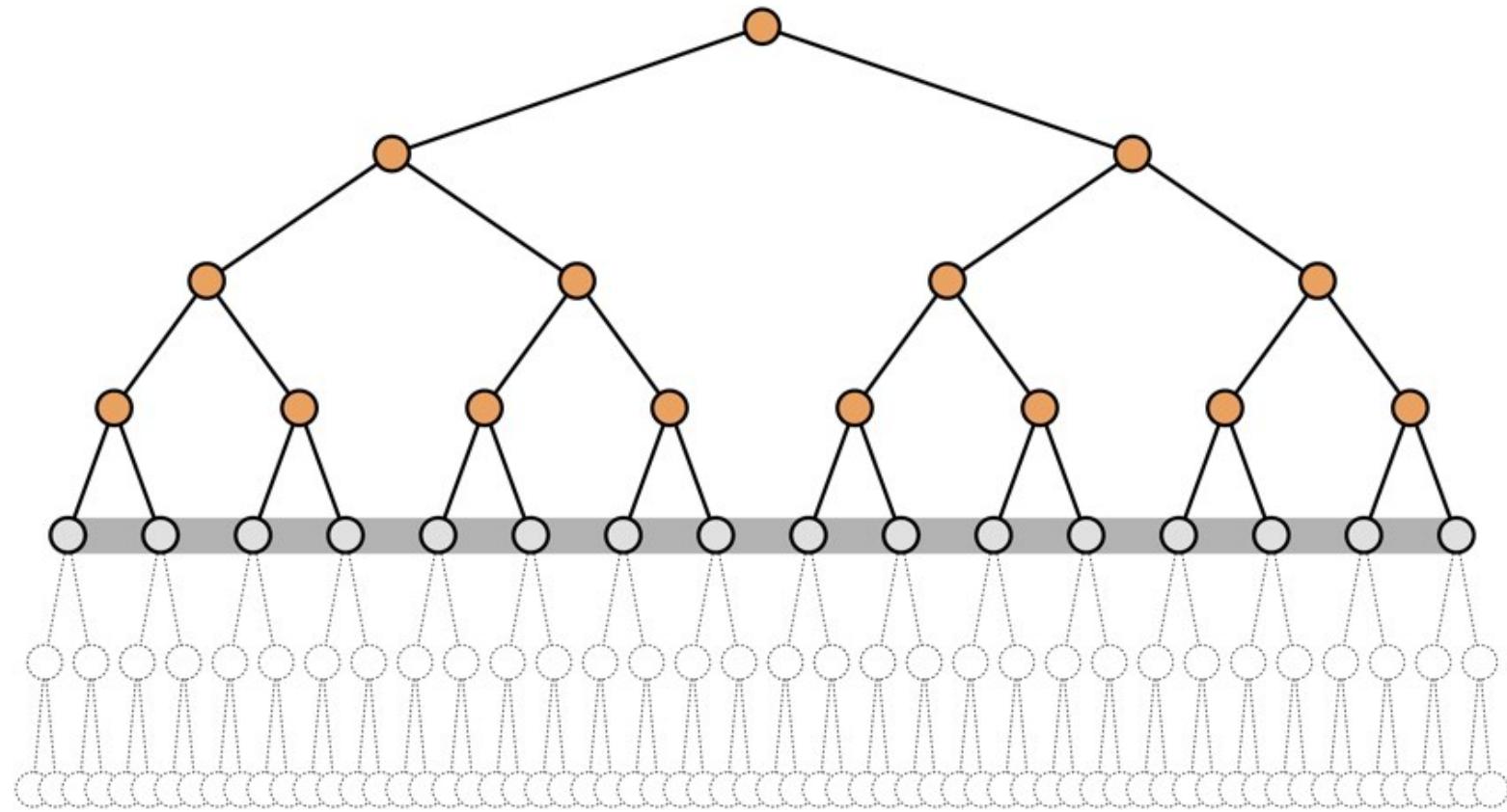
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



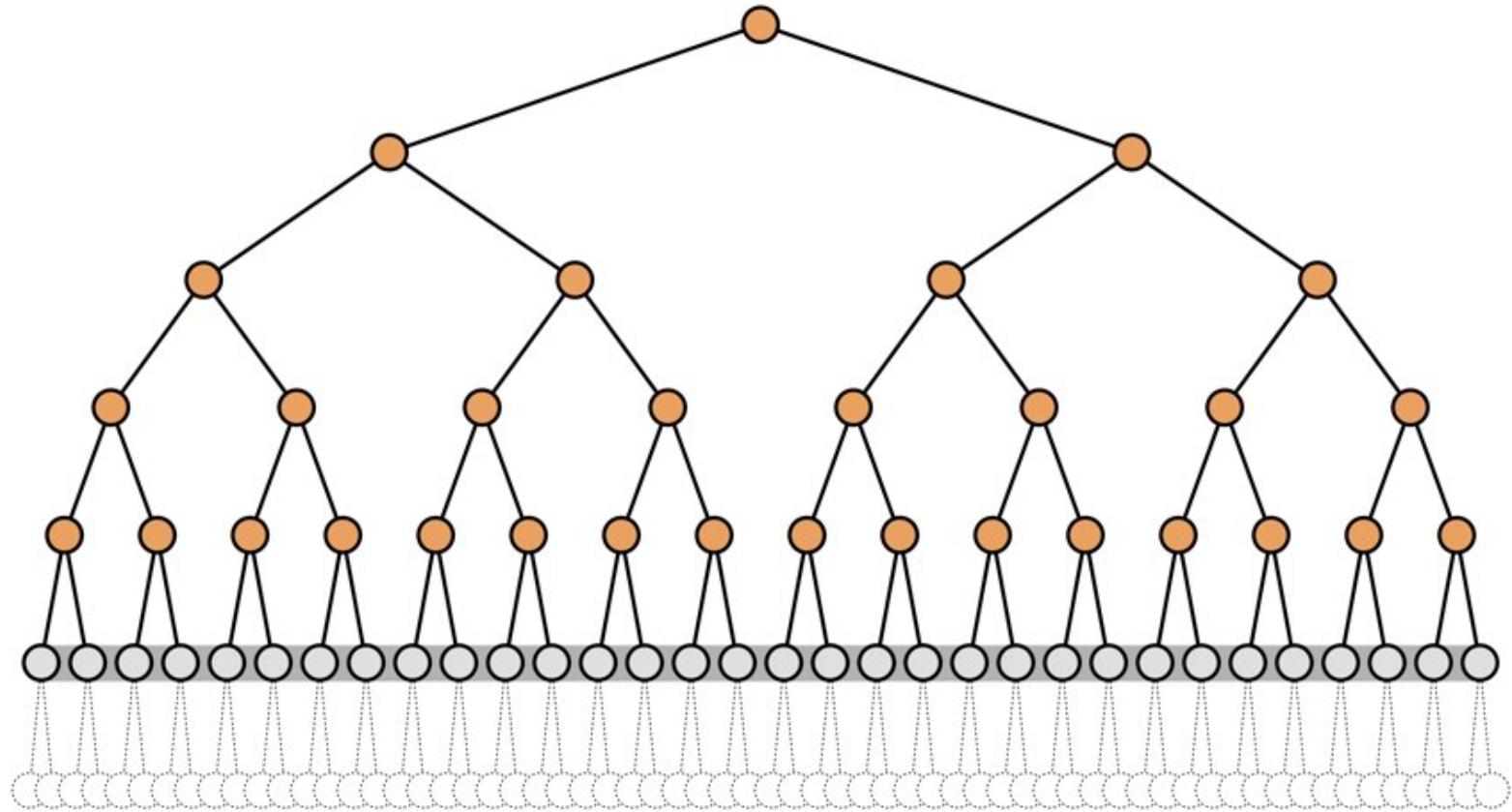
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



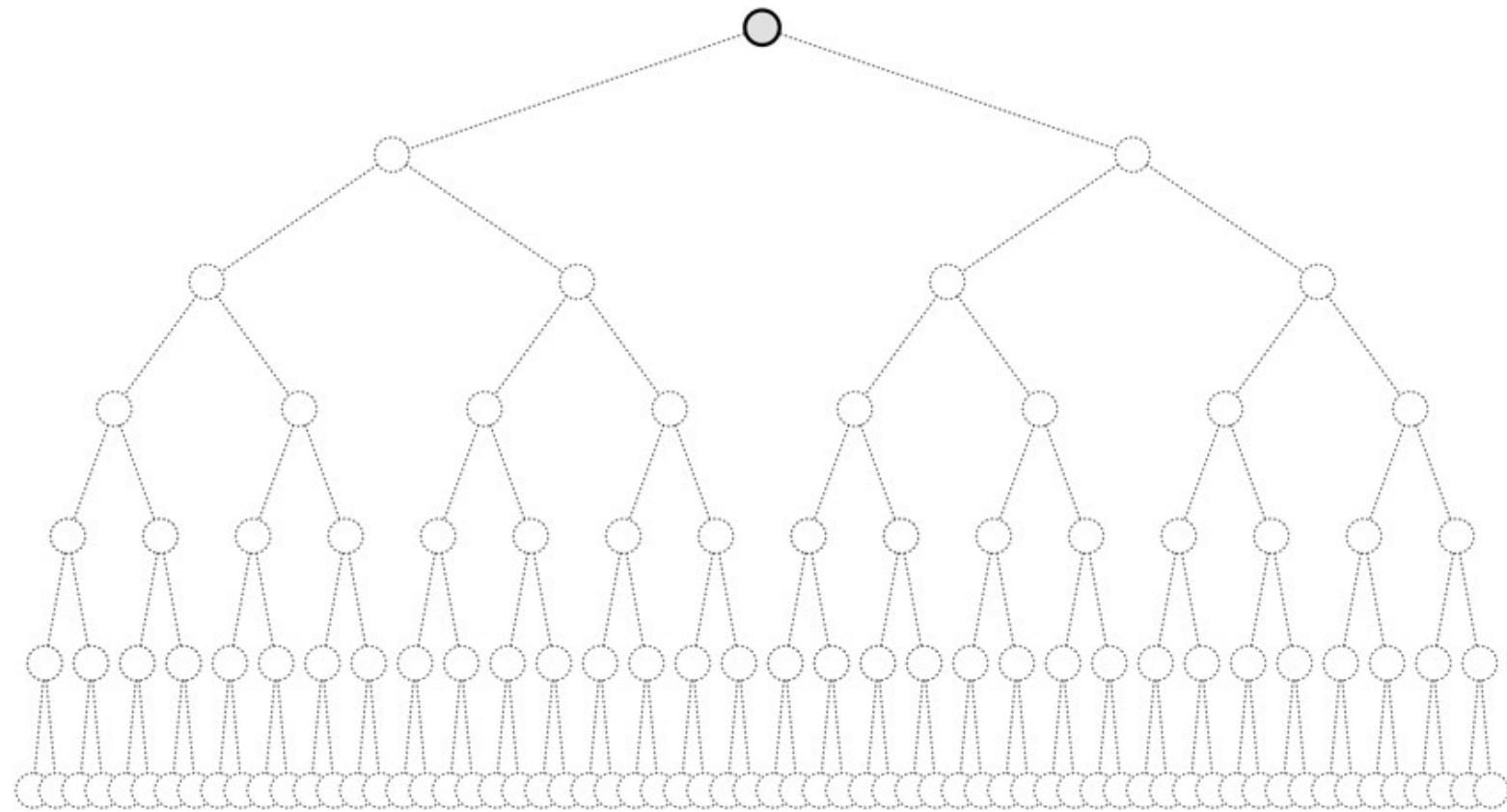
The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

BFS



The frontier consists of nodes of similar depth (horizontal).
Search proceeds by exhausting one layer at a time.

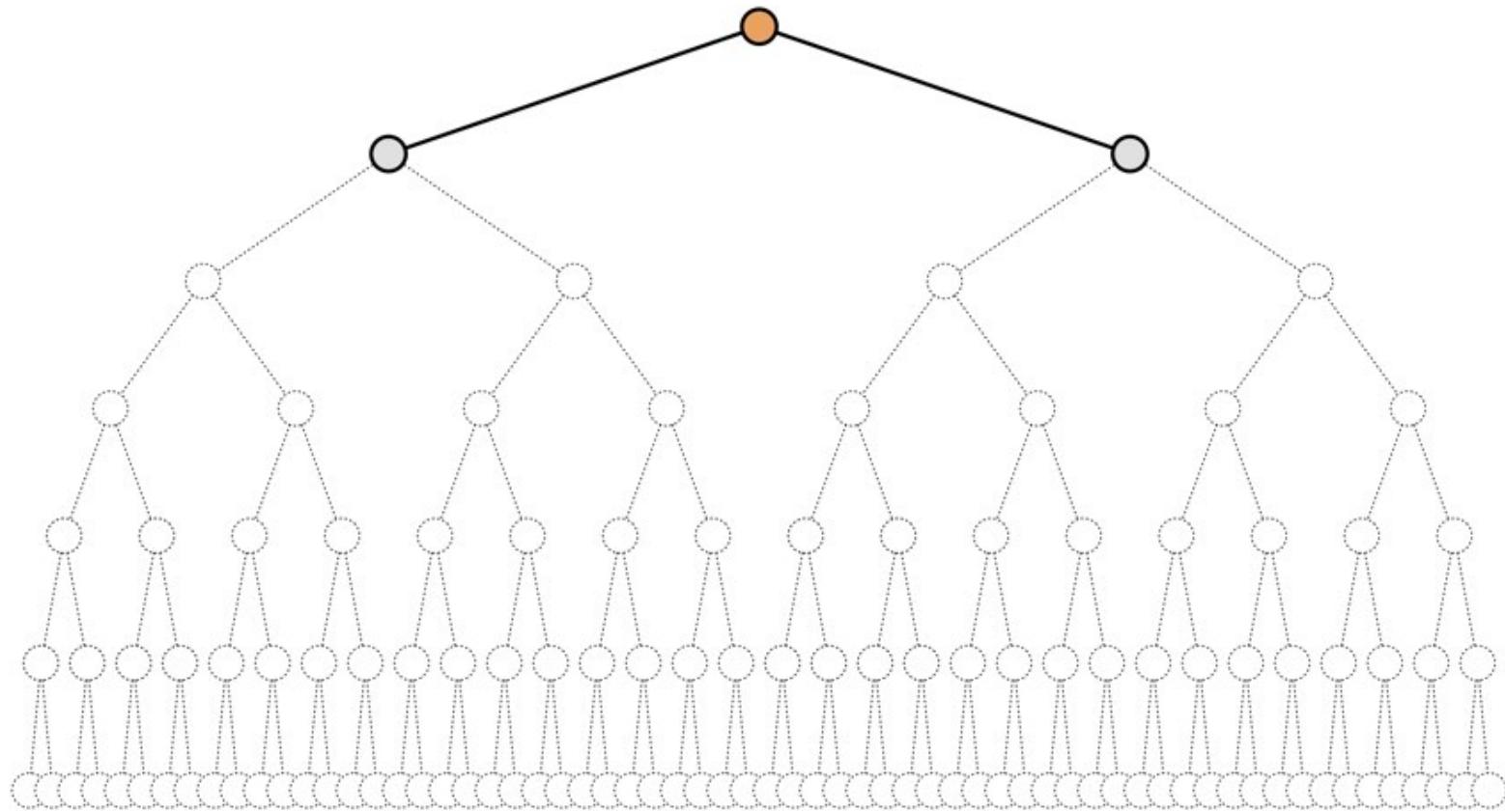
UCS



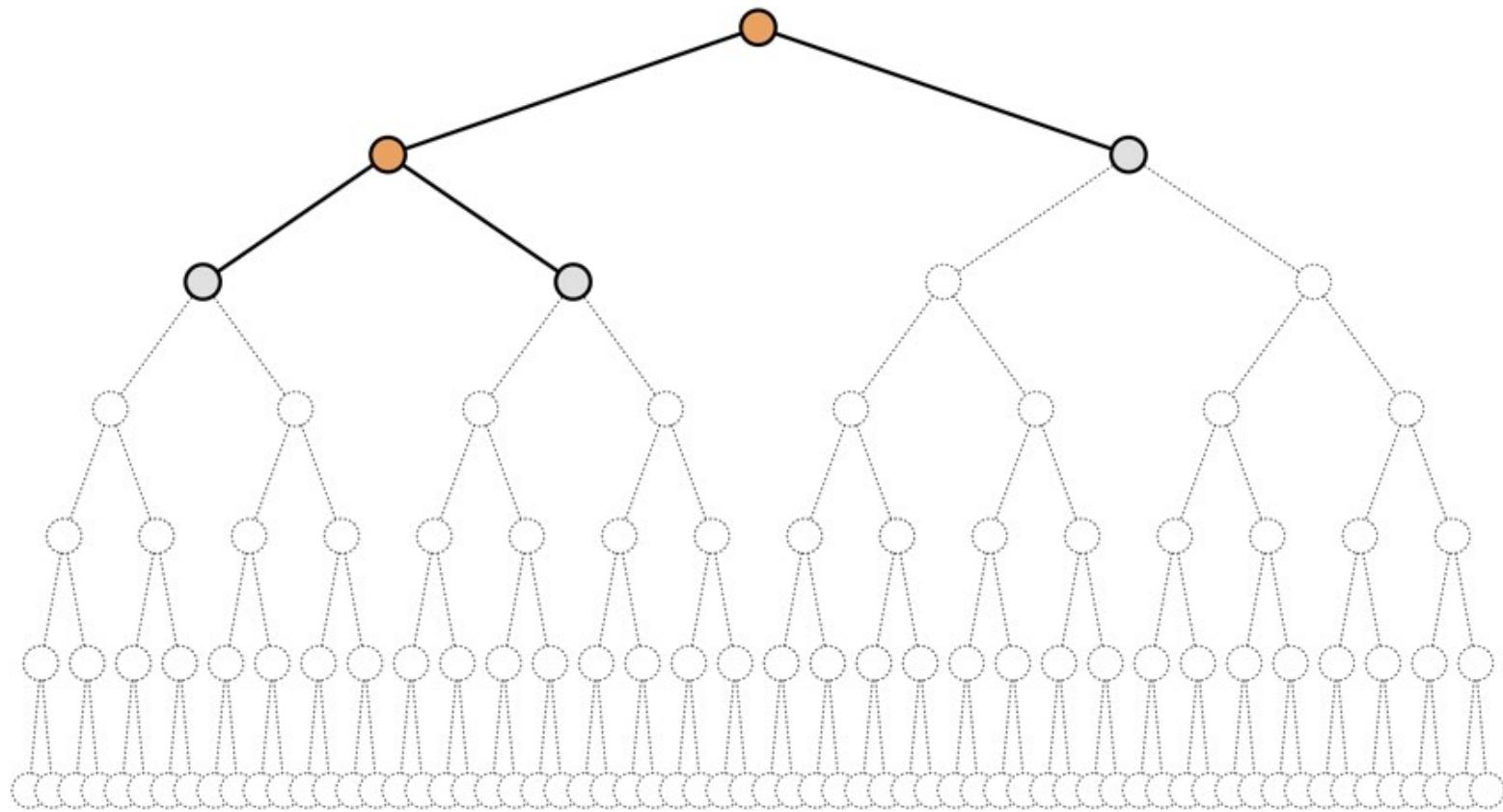
Searches layer by layer ...
... with each layer organized by path cost.

Prioritized by the lowest cost

UCS

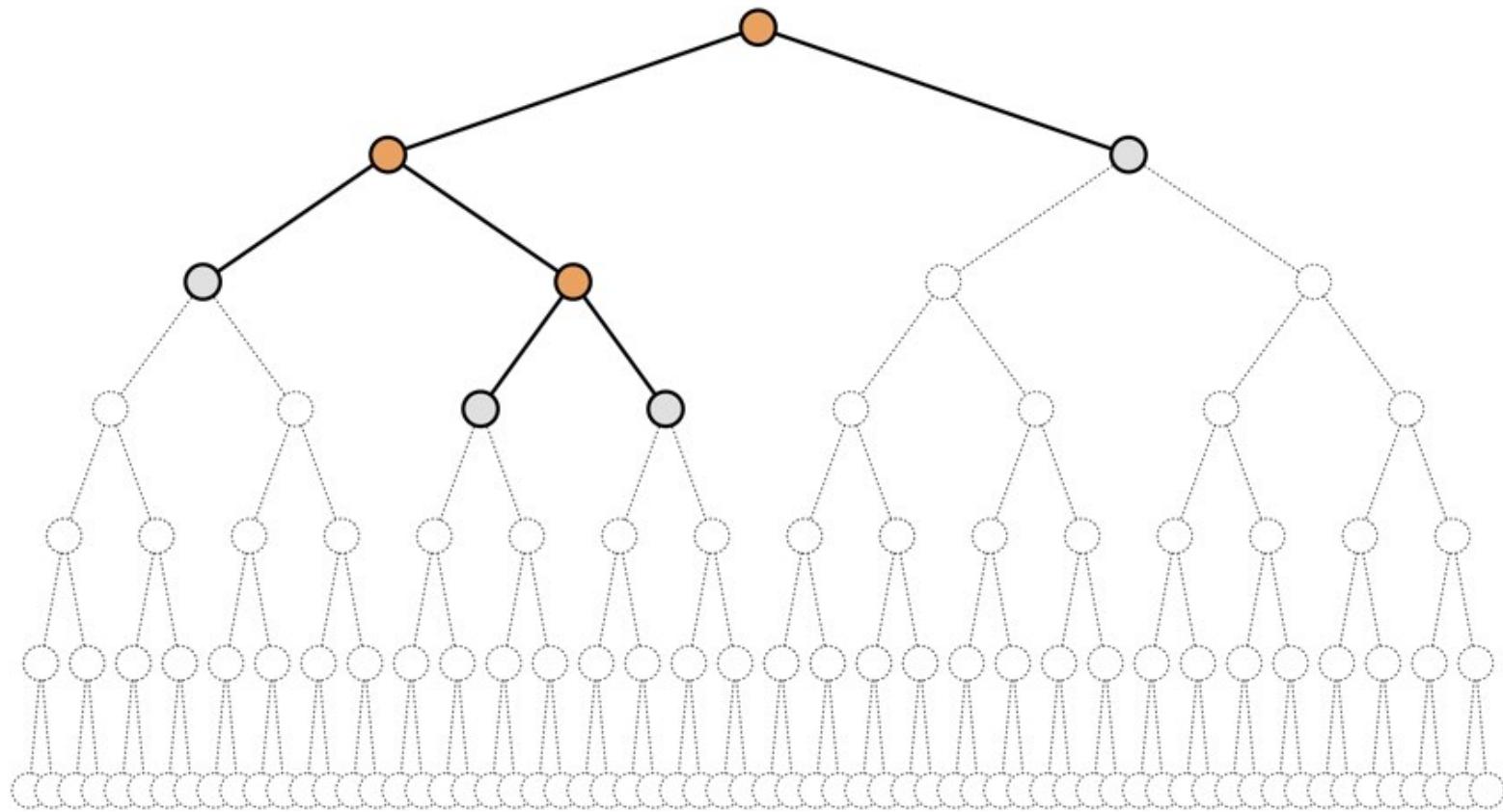


Searches layer by layer ...
... with each layer organized by path cost.

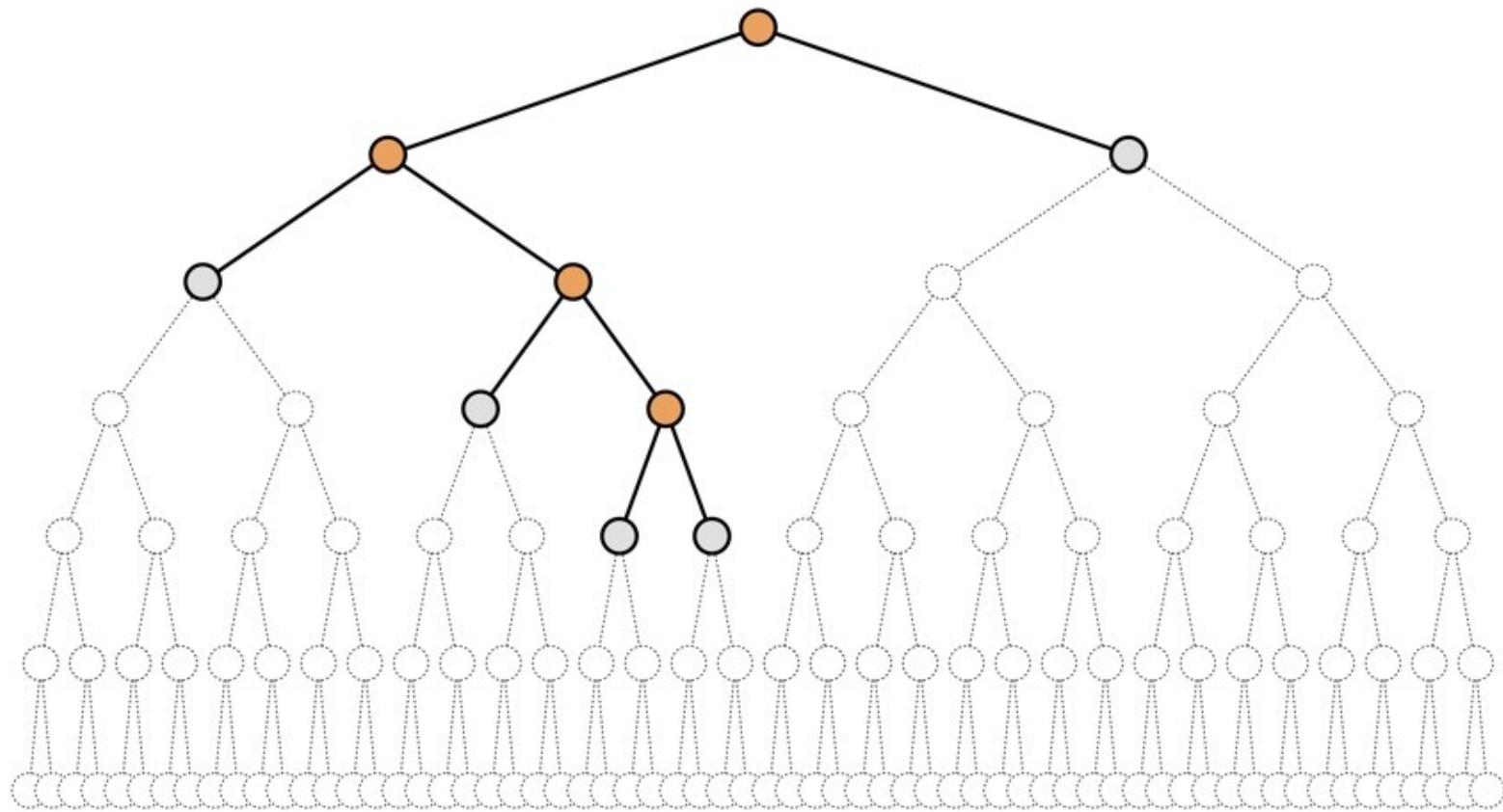


Searches layer by layer ...
... with each layer organized by path cost.

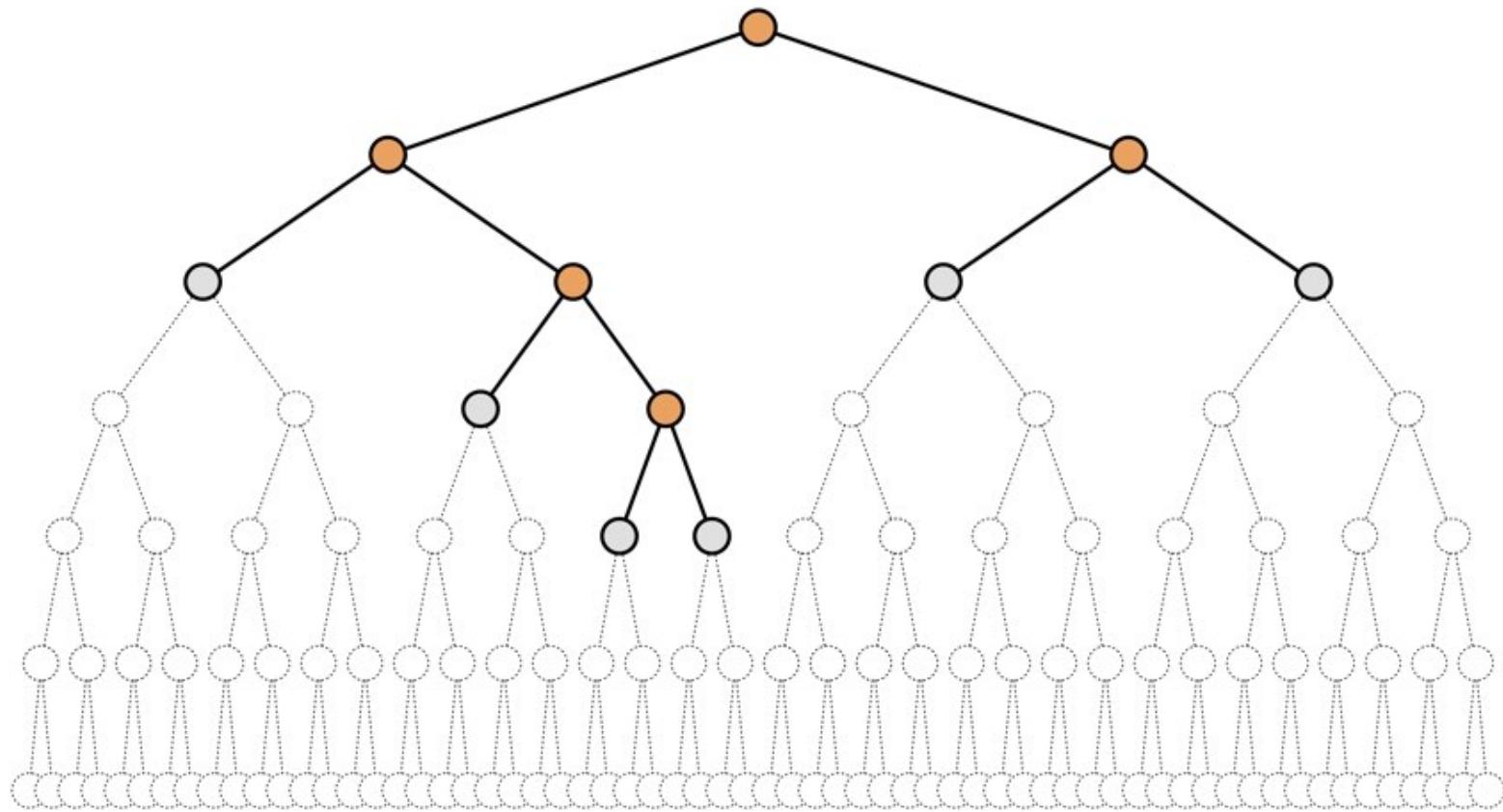
UCS



Searches layer by layer ...
... with each layer organized by path cost.

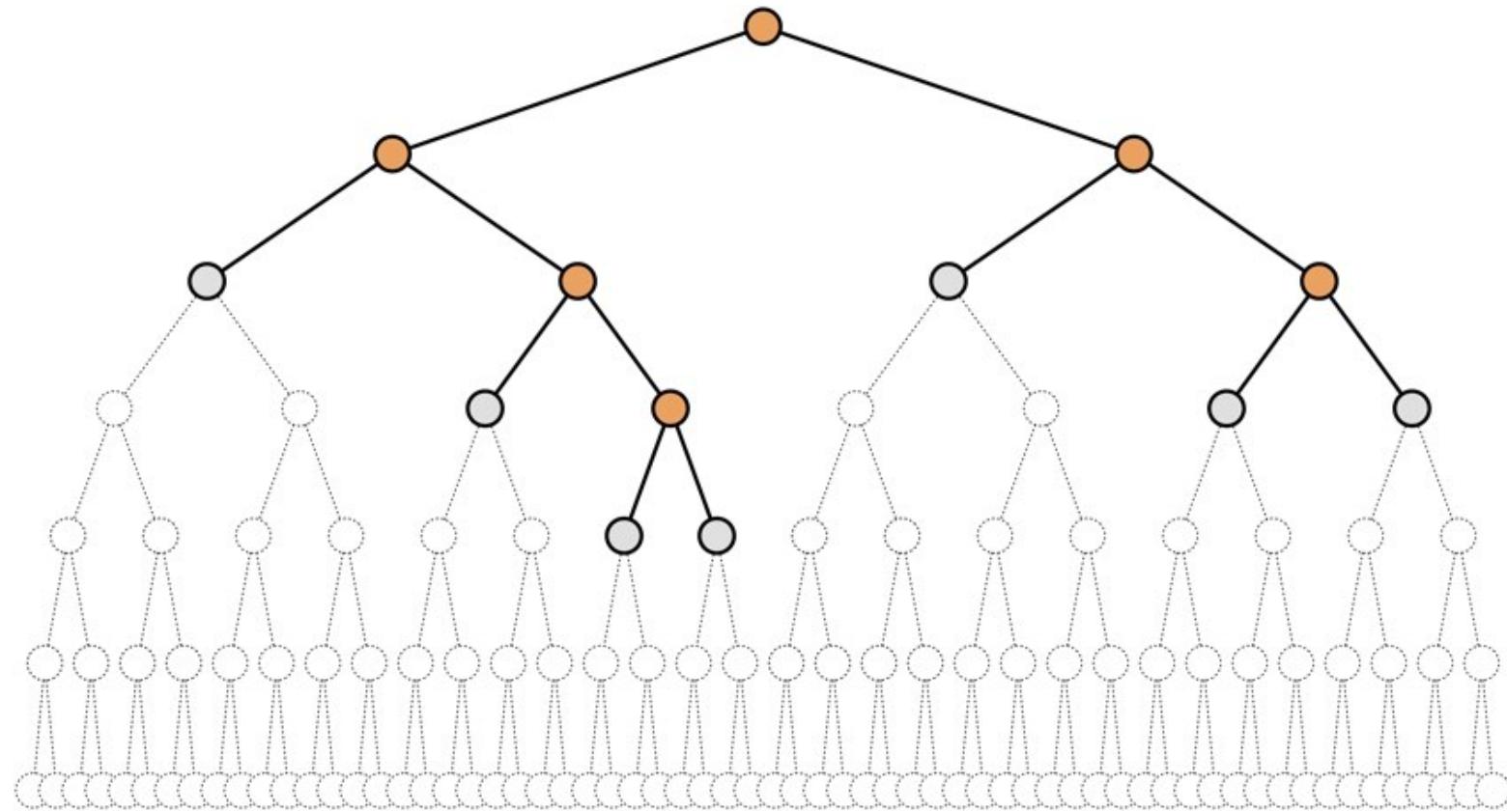


Searches layer by layer ...
... with each layer organized by path cost.



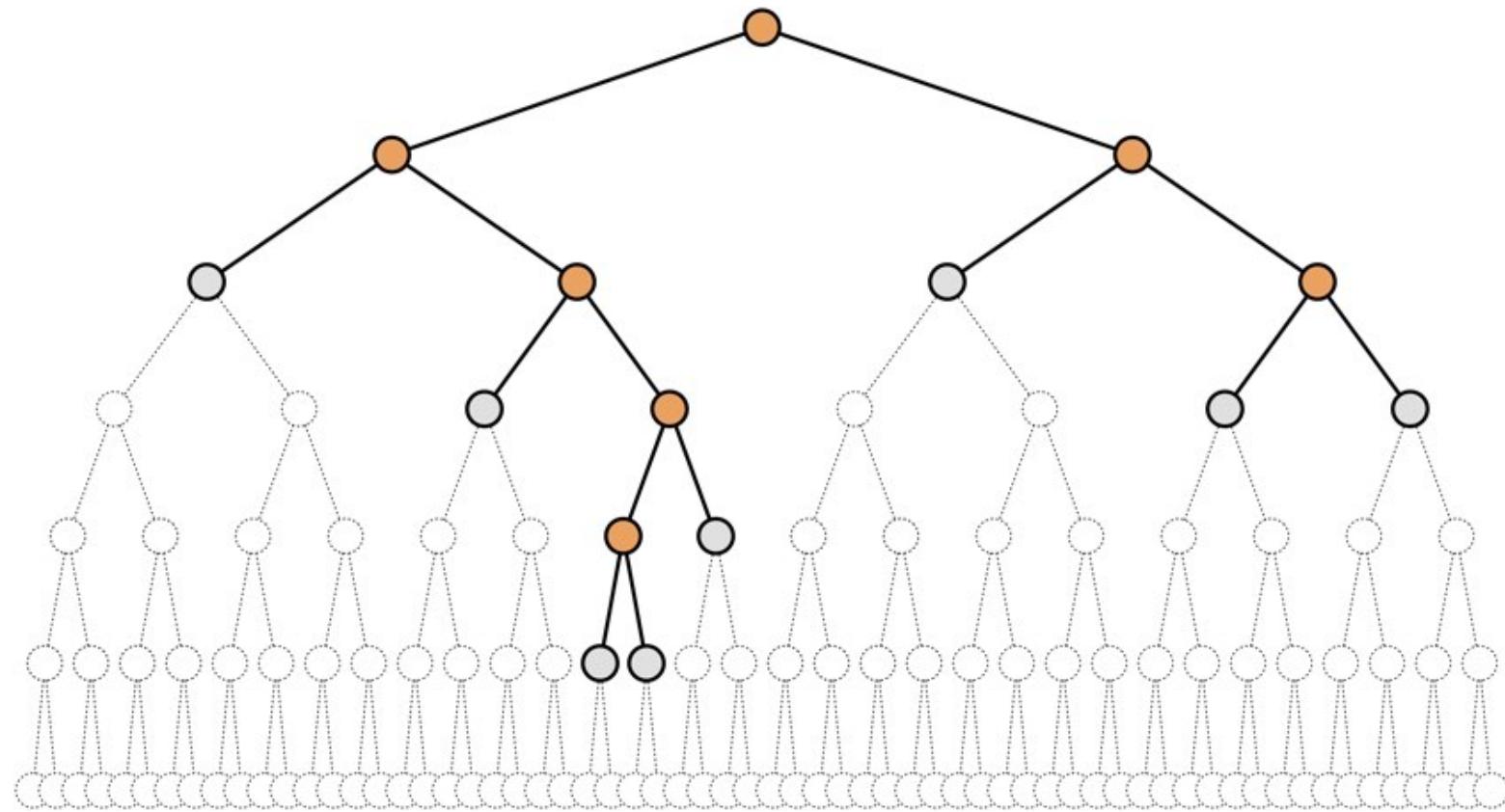
Searches layer by layer ...
... with each layer organized by path cost.

UCS

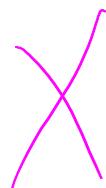


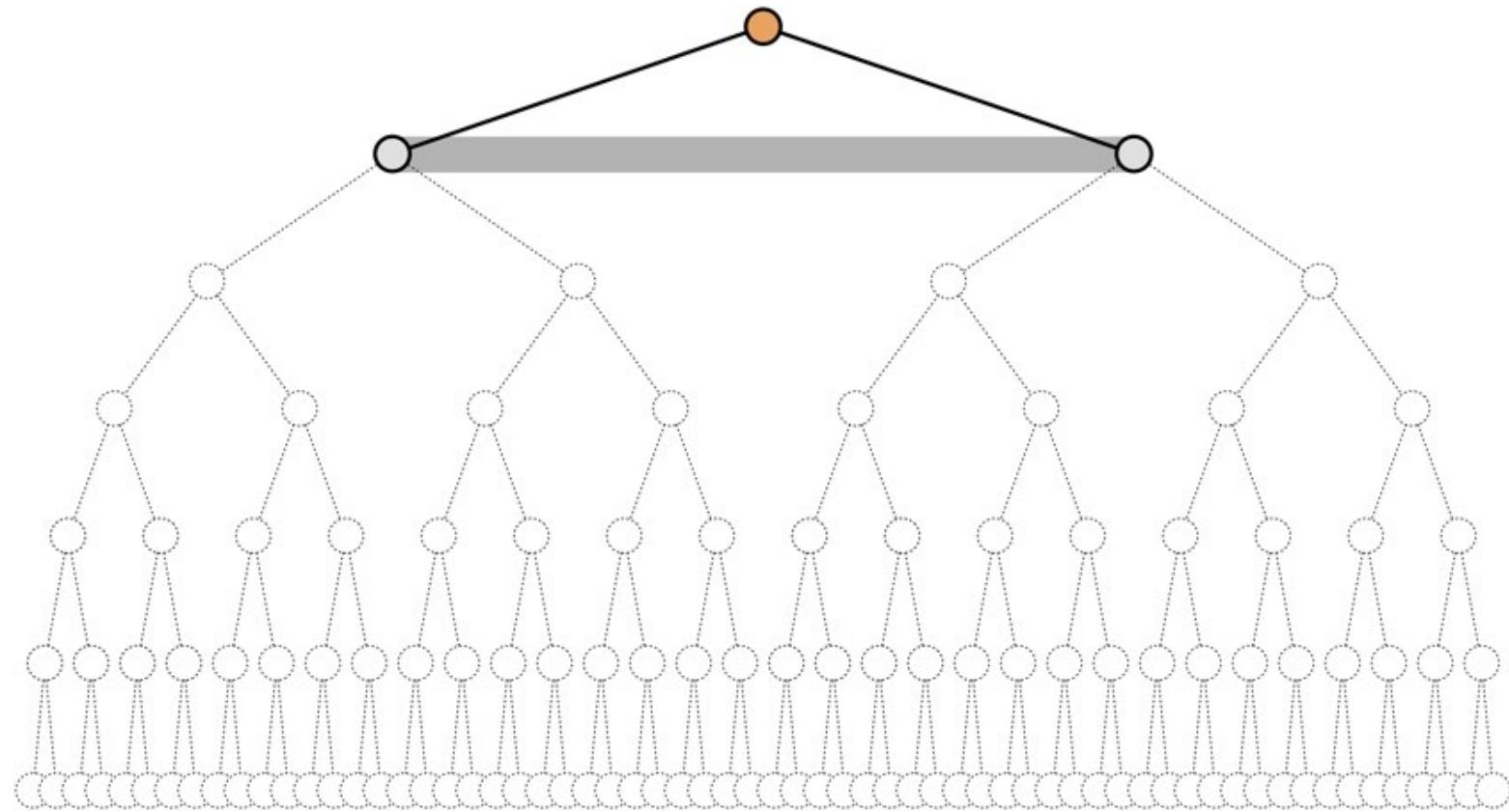
Searches layer by layer ...
... with each layer organized by path cost.



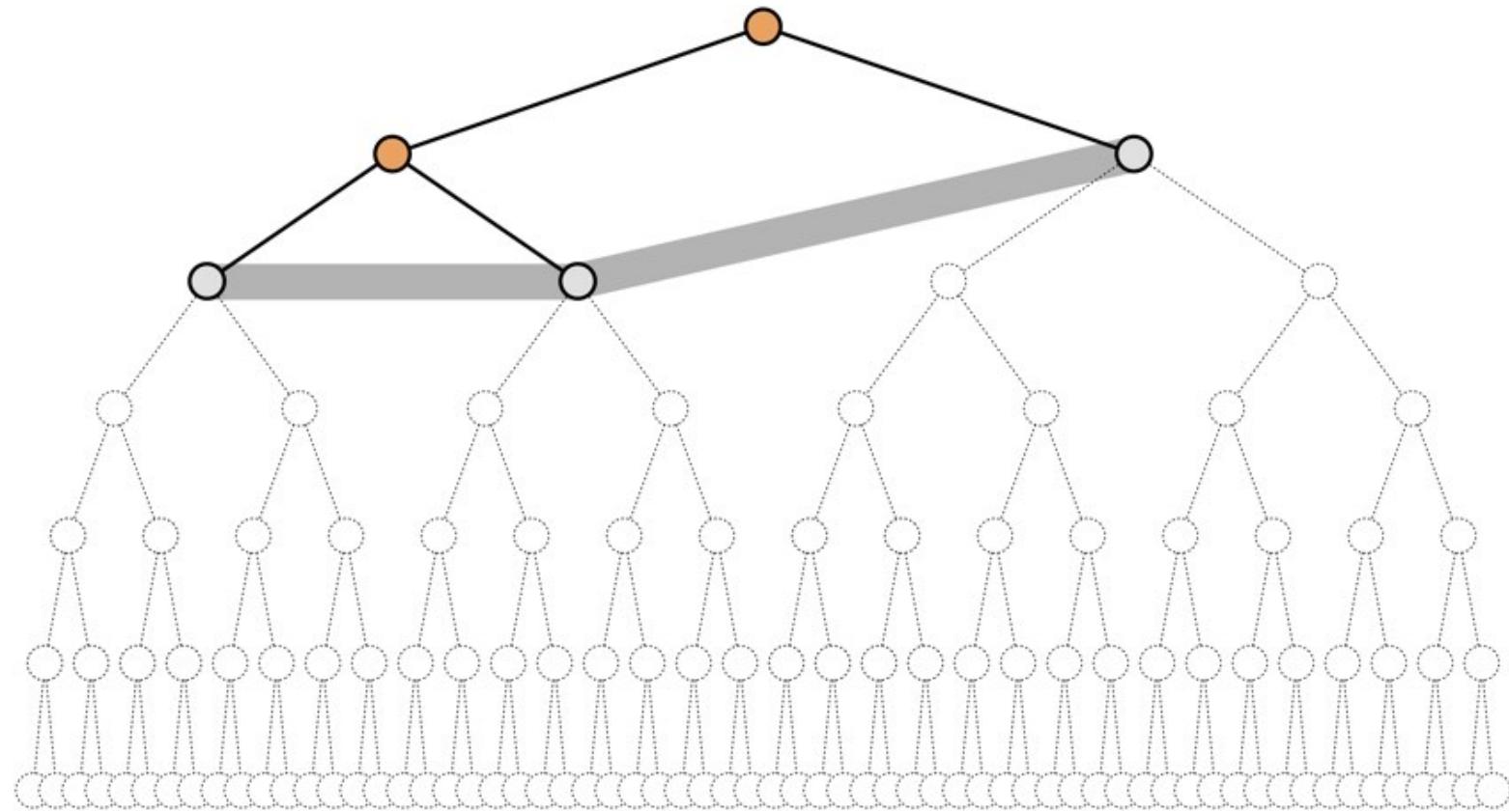


Searches layer by layer ...
... with each layer organized by path cost.

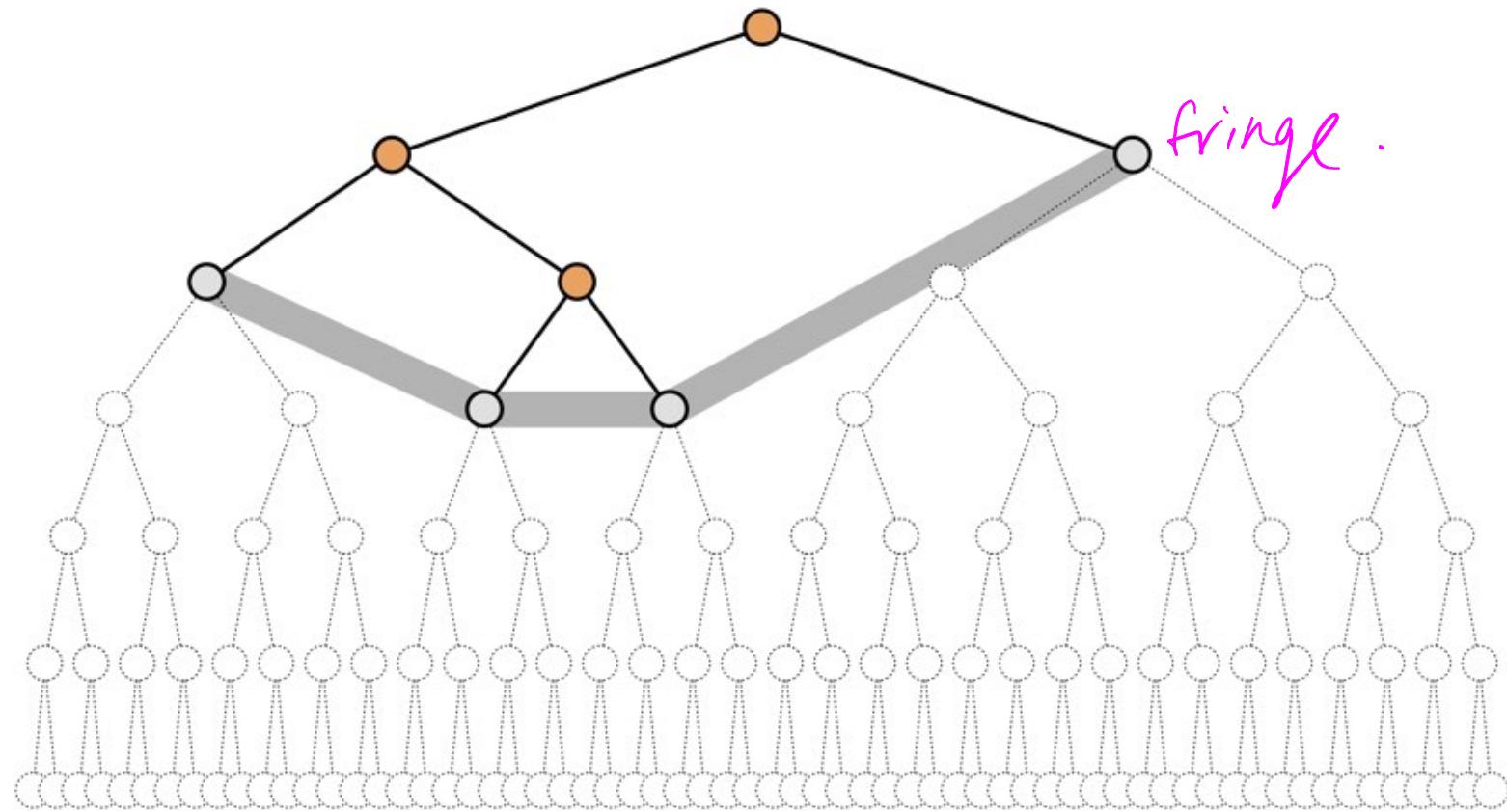




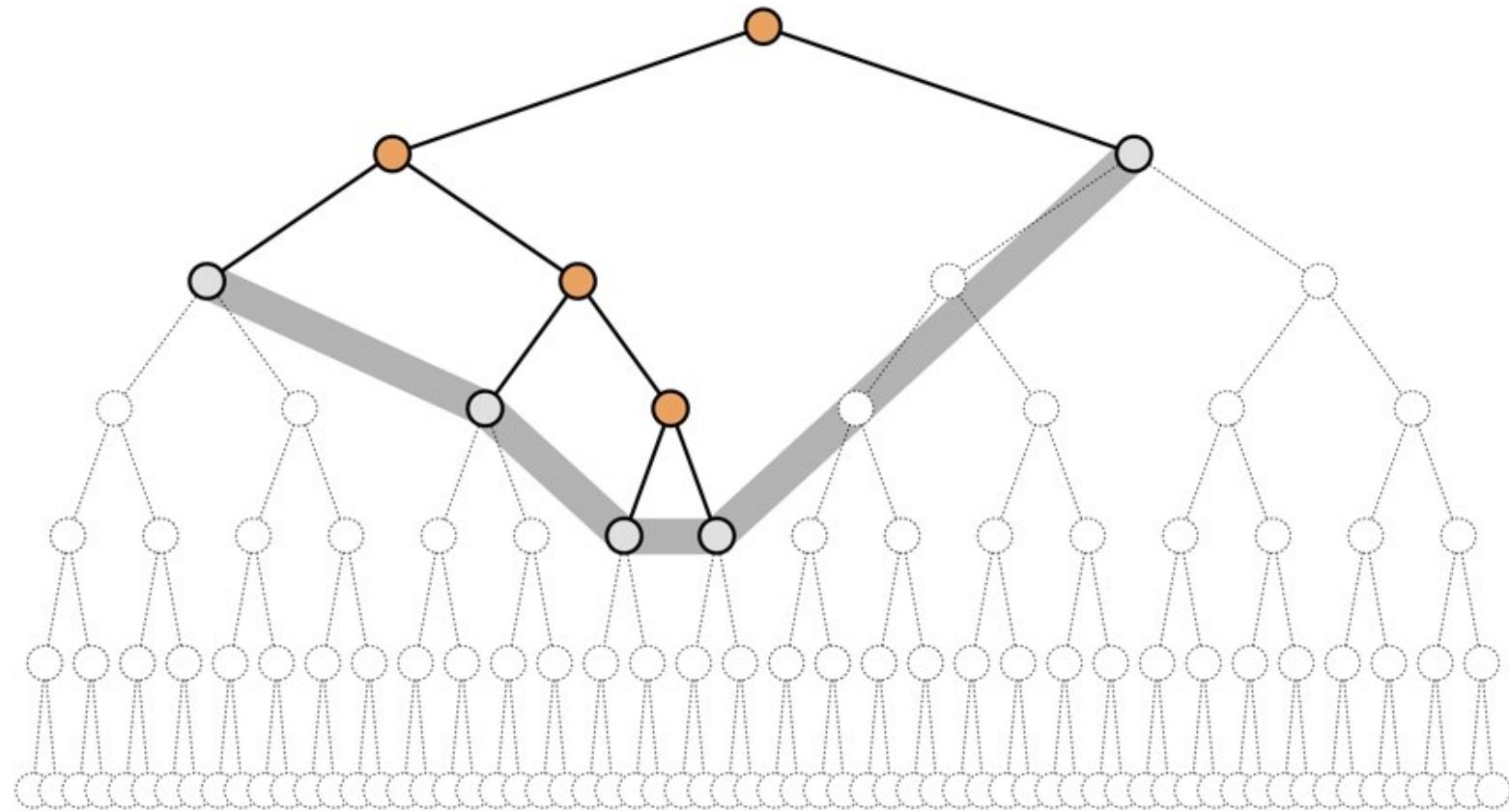
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



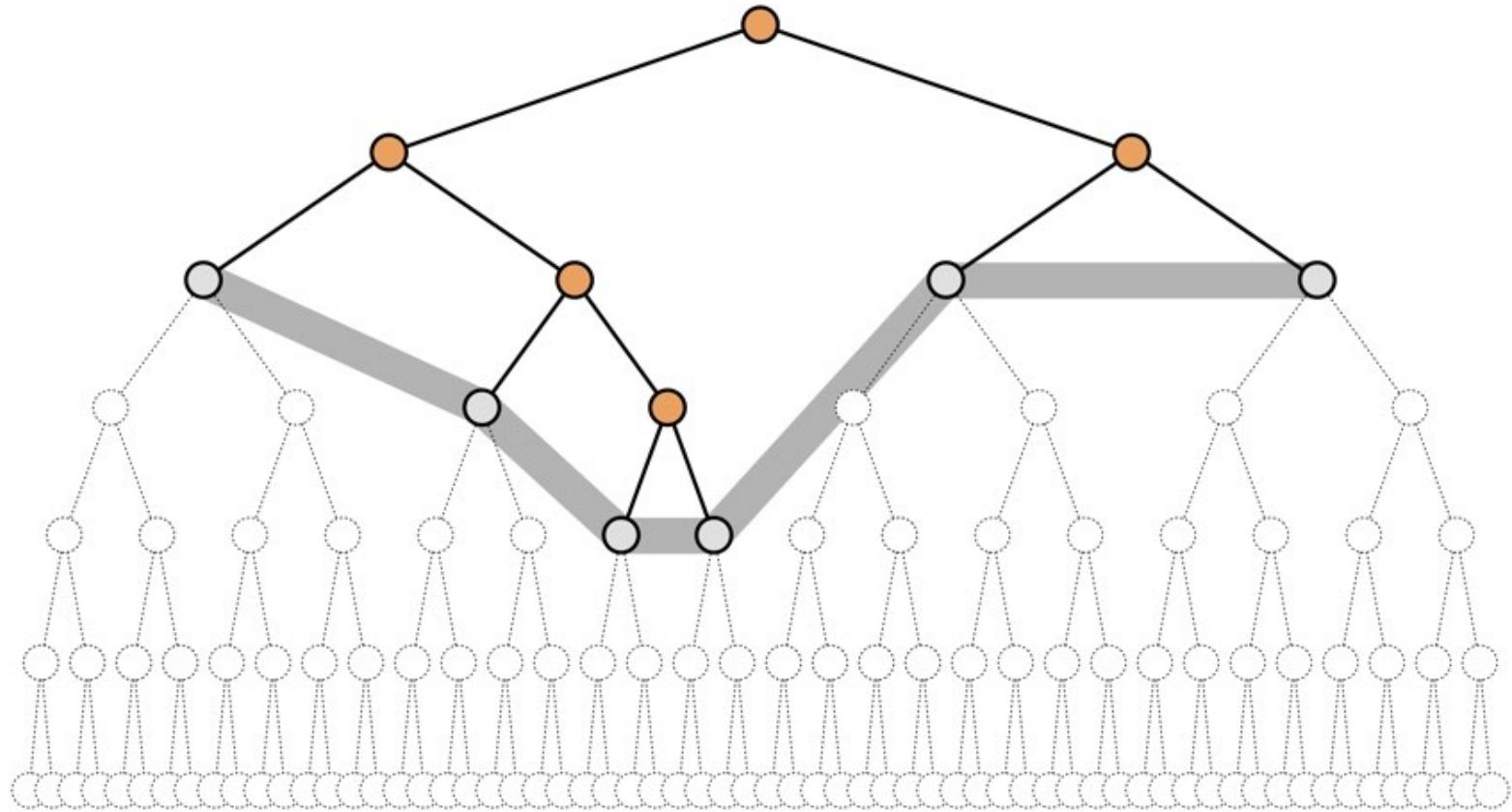
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



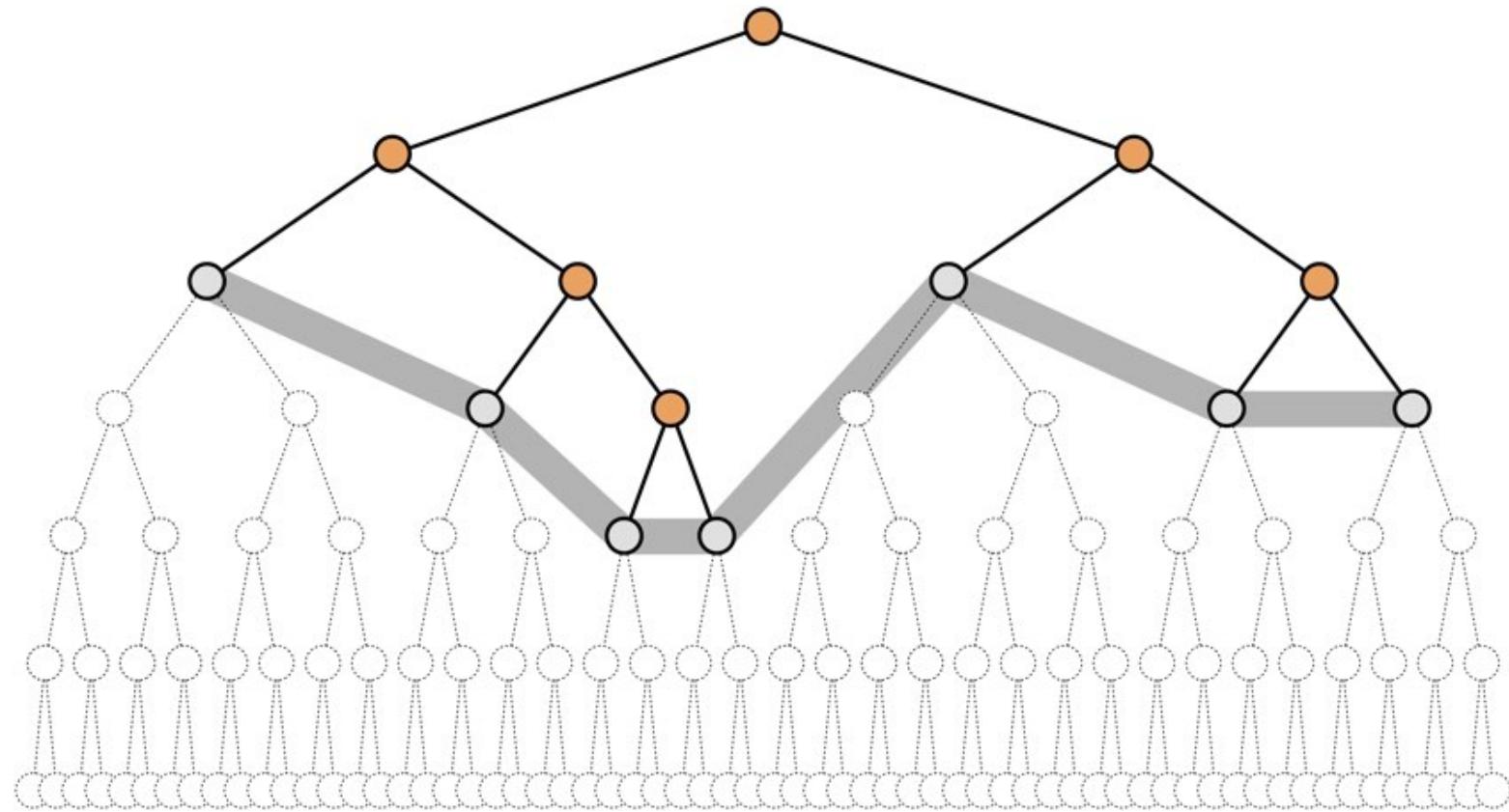
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



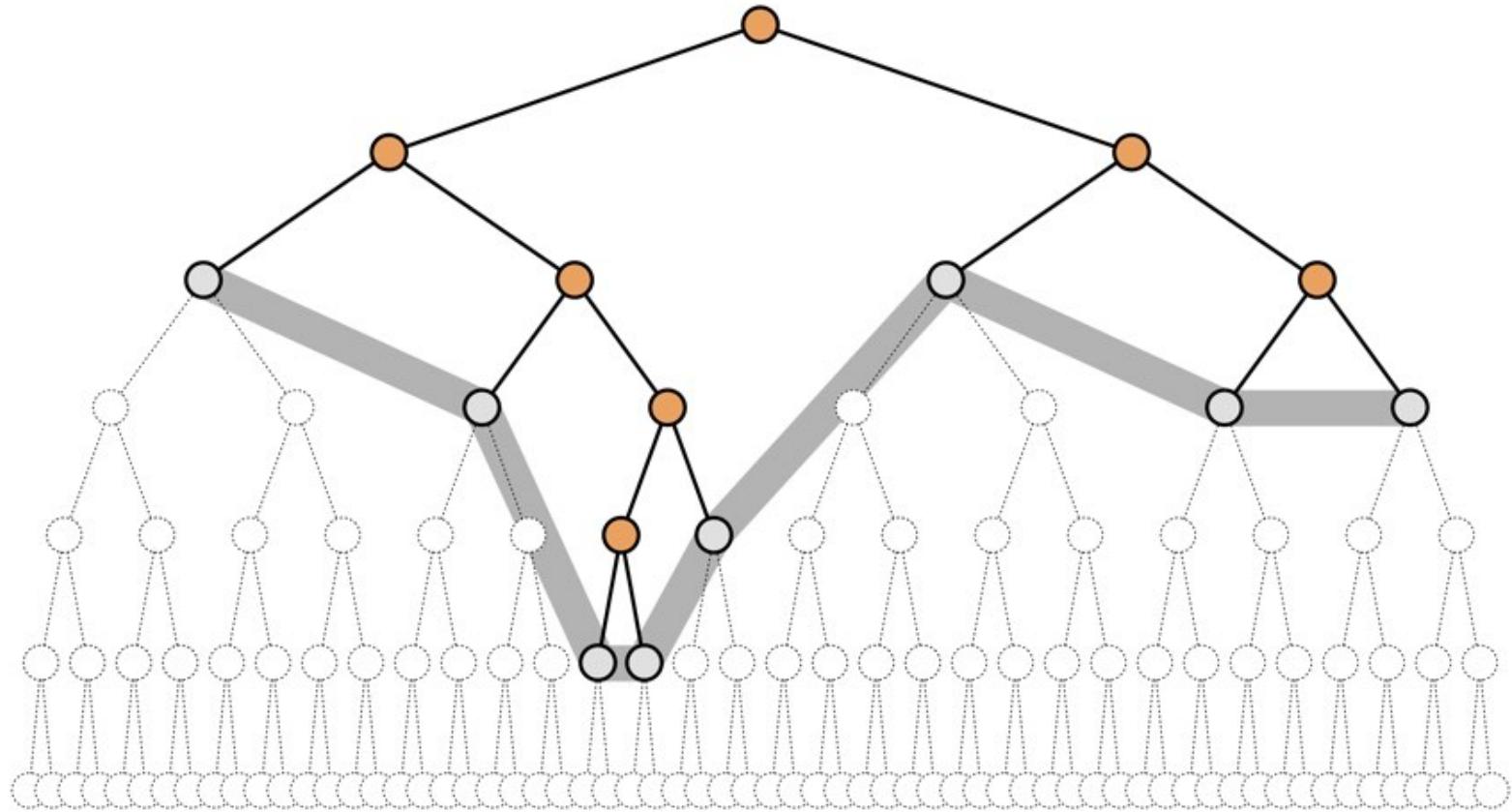
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.



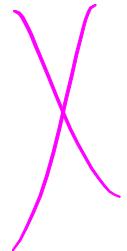
The frontier consists of nodes of various depths (jagged).
Search proceeds by expanding the lowest-cost nodes.

Recap

We can organize the algorithms into pairs where the first proceeds by layers, and the other proceeds by subtrees.

(1) Iterate on Node Depth:

- BFS searches layers of increasing node depth.
- IDS searches subtrees of increasing node depth.



Recap

We can organize the algorithms into pairs where the first proceeds by layers, and the other proceeds by subtrees.

(1) Iterate on Node Depth:

- BFS searches layers of increasing node depth.
- IDS searches subtrees of increasing node depth.

(2) Iterate on Path Cost + Heuristic Function:

- A* searches layers of increasing path cost + heuristic function.
- IDA* searches subtrees of increasing path cost + heuristic function.

→ Iterative Depth A* search

Recap

Which cost function?

- UCS searches layers of increasing path cost. g
- Greedy best first search searches layers of increasing heuristic function. h
- A* search searches layers of increasing path cost + heuristic function.

$$f = g + h$$

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

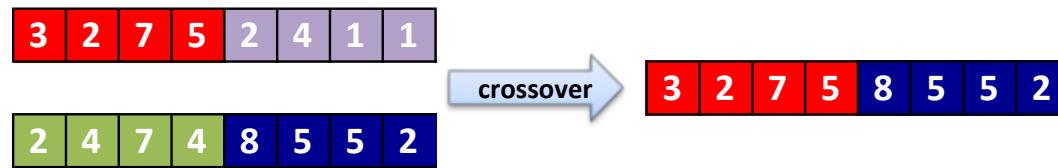
<http://aima.cs.berkeley.edu/>

L 3.4

Artificial Intelligence

Search Agents

Local search



Local search

- Search algorithms seen so far are designed to explore search spaces systematically.
- Problems: observable, deterministic, known environments where the solution is a sequence of actions.
- Real-World problems are more complex.
- When a goal is found, the path to that goal constitutes a solution to the problem. But, depending on the applications, the path may or may not matter.
- If the path does not matter/systematic search is not possible, then consider another class of algorithms.

Local search

- In such cases, we can use iterative improvement algorithms, **Local search**.
- Also useful in pure **optimization problems** where the goal is to find the best state according to an **optimization function**.
- **Examples:**
 - Integrated circuit design, telecommunications network optimization, etc.
 - N-puzzle or 8-queen: what matters is the final configuration of the puzzle, not the intermediary steps to reach it.

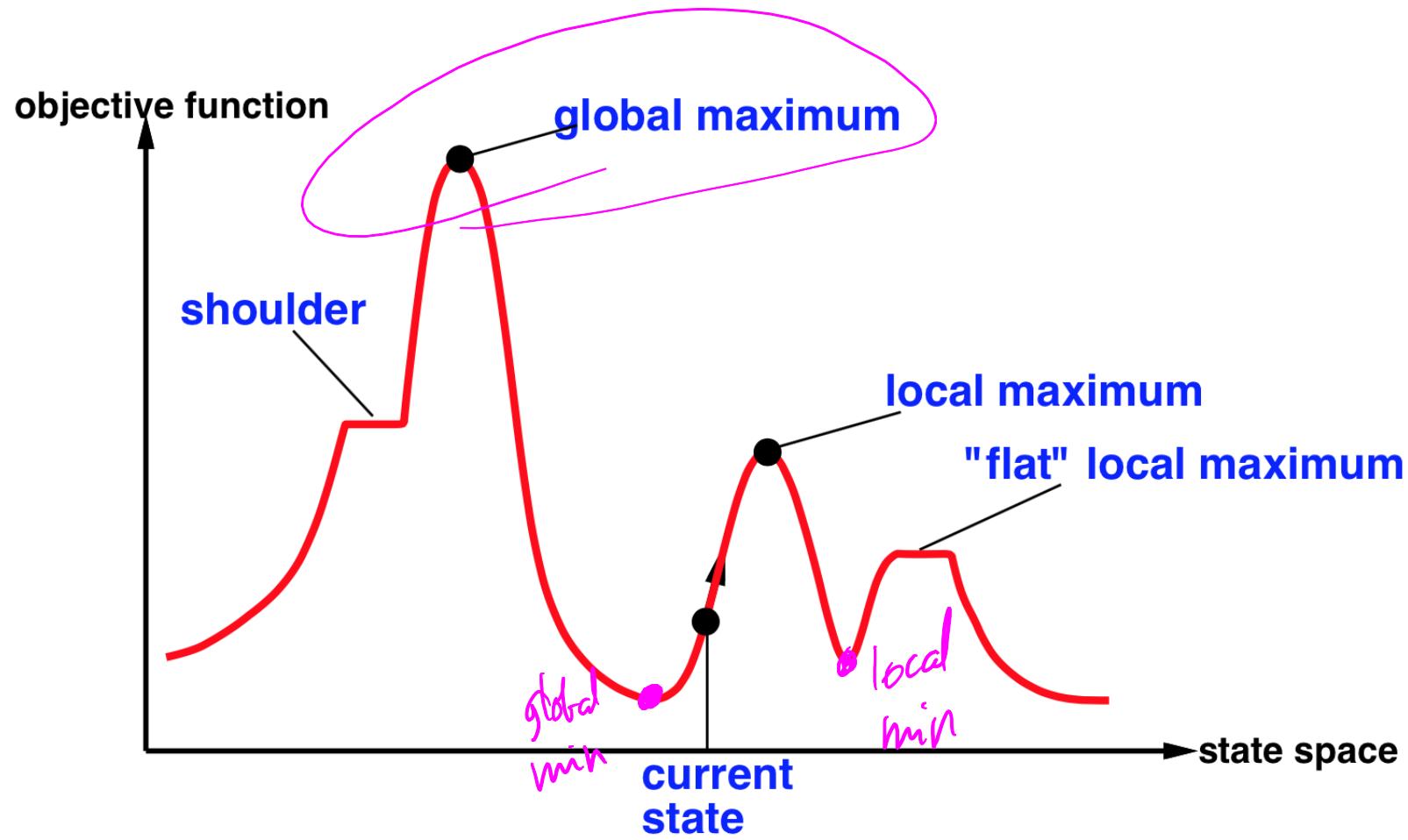
Local search

- **Idea:** keep a single “current” state, and try to improve it.
- Move only to neighbors of that node.
- **Advantages:**
 1. No need to maintain a search tree.
 2. Use very little memory.
 3. Can often find good enough solutions in continuous or large state spaces.

Local search

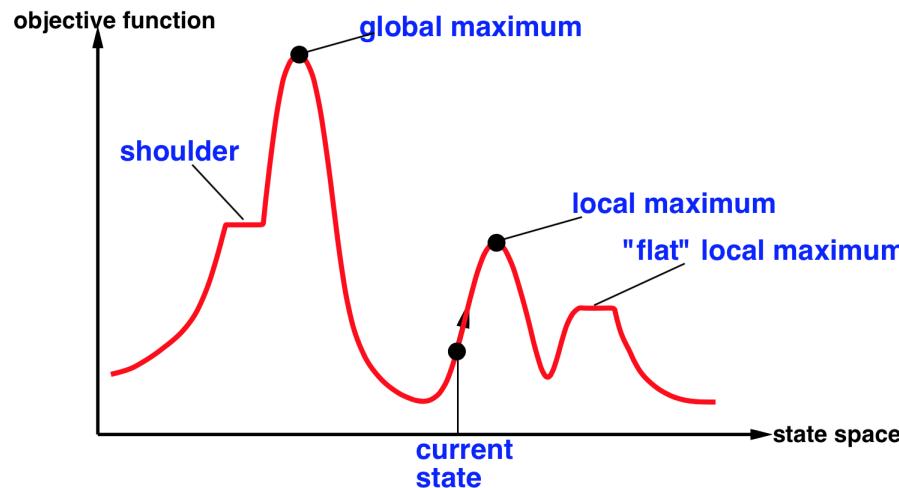
- **Local Search Algorithms:**
 - Hill climbing (steepest ascent/descent).
 - Simulated Annealing: inspired by statistical physics.
 - Local beam search.
 - Genetic algorithms: inspired by evolutionary biology.

Local search



State space landscape

Hill climbing



- Also called **greedy local search**.
- Looks only to immediate good neighbors and not beyond.
- Search moves uphill: moves in the direction of increasing elevation/value to find the top of the mountain.
- Terminates when it reaches a **pick**.
- Can terminate with a local maximum, global maximum or can get stuck and no progress is possible.
- A **node is a state and a value**.



Hill climbing

```
function HILL-CLIMBING(initialState)
    returns State that is a local maximum
```

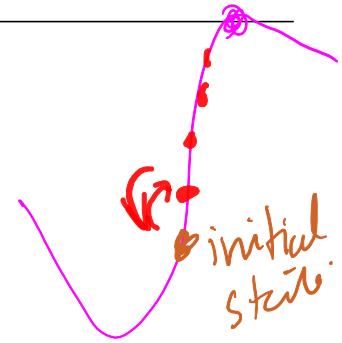
initialize current **with** initialState

loop do

 neighbor = a highest-valued successor of current

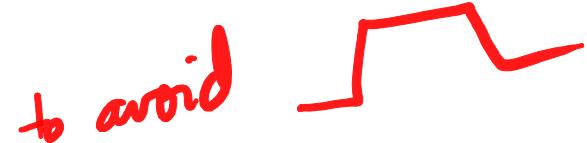
if neighbor.value \leq current.value:
 return **current.state**

 current = neighbor



Hill climbing

Other variants of hill climbing include



- **Sideways moves** escape from plateaux where best successor has same value as the current state.
- **Random-restart** hill climbing overcomes local maxima: keep trying! (either find a goal or get several possible solution and pick the max).
- **Stochastic** hill climbing chooses at random among the uphill moves.

Hill climbing

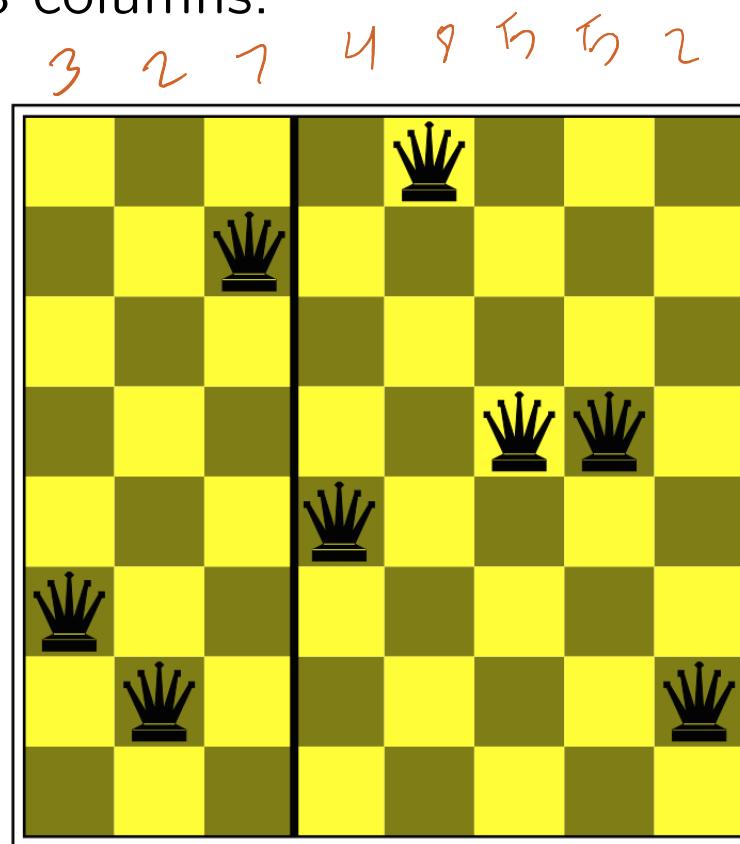
- **Hill climbing** effective in general but depends on shape of the landscape.
- Successful in many real-problems after a reasonable number of restarts.
- **Local beam search** maintains k states instead of one state.
- Select the k best successor, and useful information is passed among the states.
- **Stochastic beam search** choose k successors are random.
- Helps alleviate the problem of the the states agglomerating around the same part of the state space.

Genetic algorithms

- **Genetic algorithms (GA)** is a variant of stochastic beam search.
- Successor states are generated by combining two parents rather than modifying a single state.
- The process is inspired by **natural selection**.
- Starts with k **randomly generated states**, called **population**. Each state is an **individual**.
- An individual is usually represented by a **string** of 0's and 1's, or digits, a finite set.
- The objective function is called **fitness function**: better states have high values of fitness function.

Genetic algorithms

- In the 8-queen problem, an individual can be represented by a **string** digits 1 to 8, that represents the position of the 8 queens in the 8 columns.



Genetic algorithms

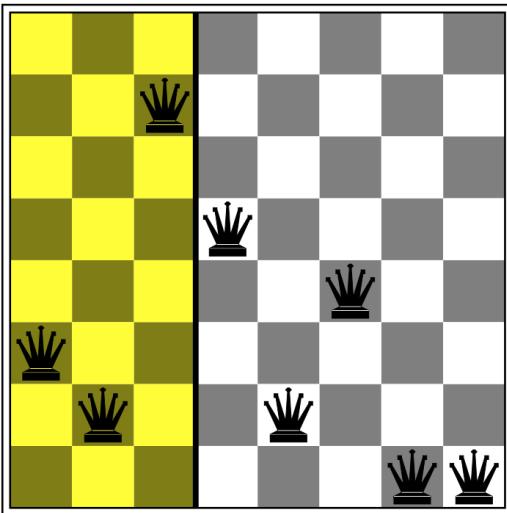
- The objective function is called **fitness function**: better states have high values of fitness function.
- Possible fitness function is the **number of non-attacking pairs of queens**.
- Fitness function of the solution: 28.

Genetic algorithms

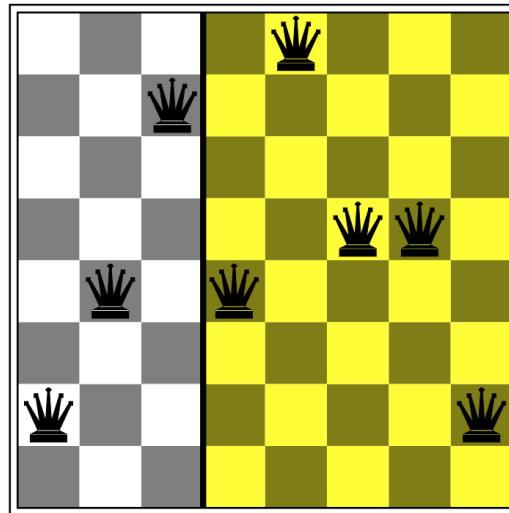
- Pairs of individuals are selected at random for **reproduction** w.r.t. some probabilities.
- A **crossover** point is chosen randomly in the string.
- **Offspring** are created by crossing the parents at the crossover point.
- Each element in the string is also subject to some **mutation** with a small probability.

Genetic algorithms

Parent 1

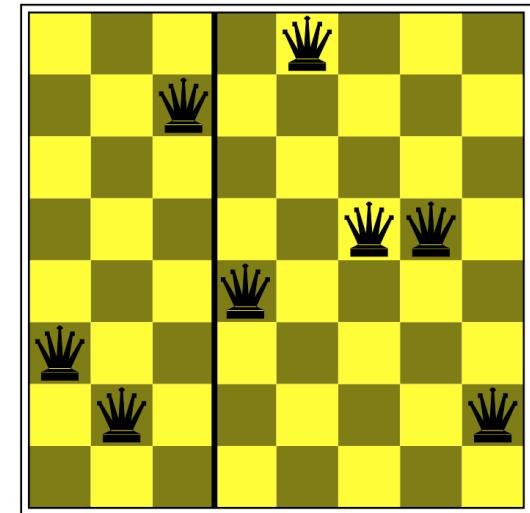


Parent 2



+

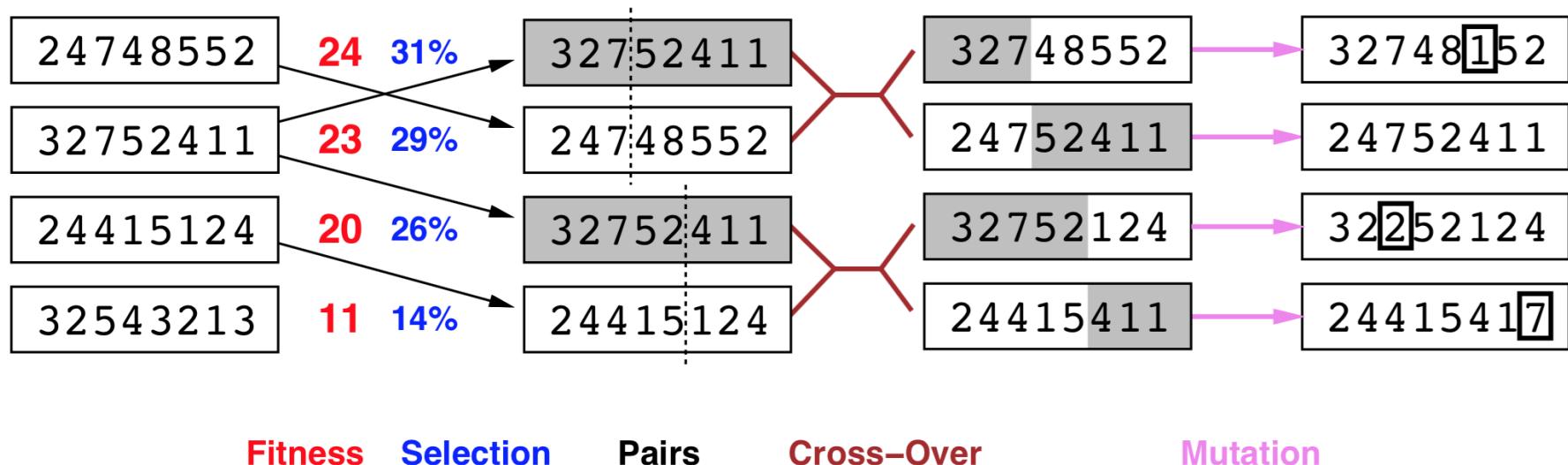
Child



=

Genetic algorithms

Generate successors from pairs of states.





Genetic algorithms

function GENETIC-ALGORITHM(*population*, *fitness-function*)
returns an individual

repeat

initialize new-population **with** \emptyset

for *i*=1 to size(*population*) **do**

x = random-select(*population,fitness-function*)

x = random-select(*population,fitness-function*)

 child = cross-over(*x,y*)

 mutate (*child*) with a small random probability

 add child to new-population

population = new-population

repeat until some individual is fit enough or enough time has elapsed

return the best individual in population w.r.t. fitness-function

Credit

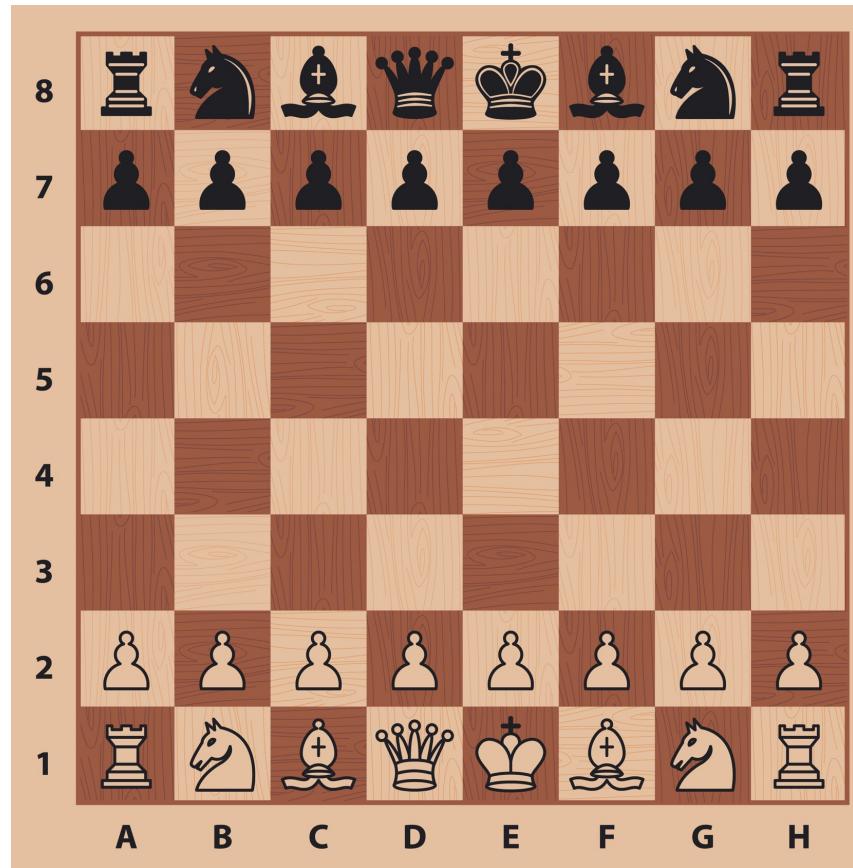
- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

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

L 4.1.1

Artificial Intelligence

Adversarial Search



Adversarial Search

- Adversarial search problems \equiv games
- They occur in multiagent competitive environments
- There is an **opponent** we can't control planning again us!
- Game vs. search: optimal solution is not a sequence of actions but a **strategy** (policy) If opponent does a , agent does b , else if opponent does c , agent does d , etc.
- Tedious and fragile if hard-coded (i.e., implemented with rules)
- Good news: Games are modeled as **search problems** and use **heuristic evaluation** functions.

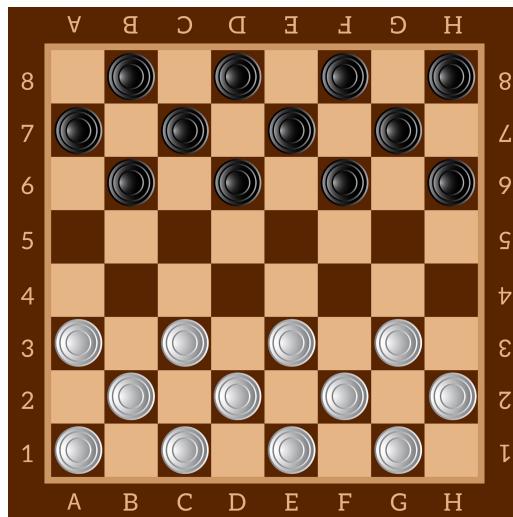
Games: hard topic

- Games are a big deal in AI
- Games are interesting to AI because they are too hard to solve
- Chess has a branching factor of 35, with 35^{100} nodes $\approx 10^{154}$
- Need to make some decision even when the optimal decision is infeasible

Adversarial Search

Checkers:

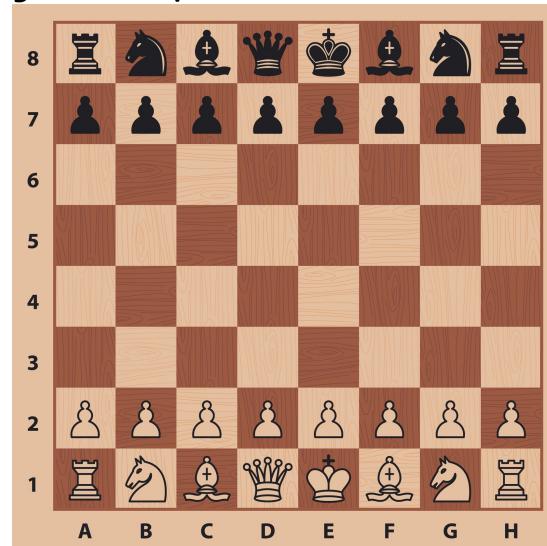
- Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.



Adversarial Search

Chess:

- In 1949, Claude E. Shannon in his paper “Programming a Computer for Playing Chess”, suggested *Chess* as an AI problem for the community.
- Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
- In 2006, Vladimir Kramnik, the undisputed world champion, was defeated 4-2 by Deep Fritz.



Adversarial Search

Go: $b > 300!$ Google Deep mind Project AlphaGo. In 2016, AlphaGo beat both Fan Hui, the European Go champion and Lee Sedol the worlds best player.

Othello: Several computer othello exists and human champions refuse to compete against computers, that are too good.



By Donarreiskoffer

via Wikimedia Commons



By Paul_012

Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

We are mostly interested in deterministic games, fully observable environments, zero-sum, where two agents act alternately.

Zero-sum Games

- Adversarial: Pure competition.
- Agents have different values on the outcomes.
- One agent maximizes one single value, while the other minimizes it.

Zero-sum Games

- Adversarial: Pure competition.
- Agents have different values on the outcomes.
- One agent maximizes one single value, while the other minimizes it.
- Each move by one of the players is called a “ply.”

One function: one agents maximizes it and one minimizes it!

Embedded thinking...

Embedded thinking or backward reasoning!



- One agent is trying to figure out what to do.
- How to decide? He thinks about the consequences of the possible actions.
- He needs to think about his opponent as well...
- The opponent is also thinking about what to do etc.
- Each will imagine what would be the response from the opponent to their actions.
- This entails an embedded thinking.

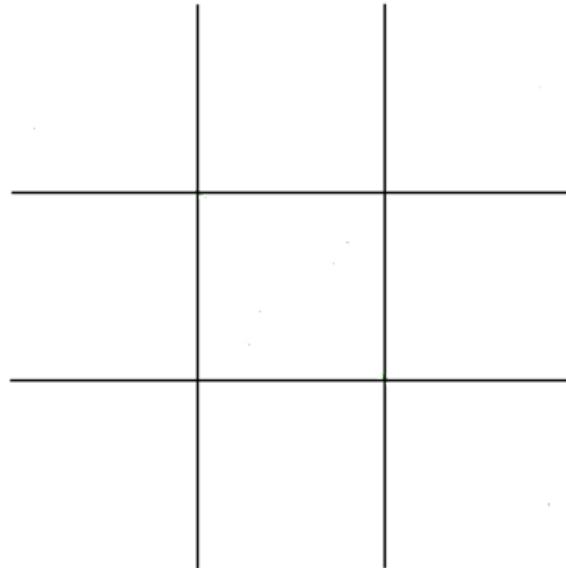
Formalization

- The **initial state**
- Player(s): defines which player has the move in state s . Usually taking turns.
- Actions(s): returns the set of legal moves in s
- **Transition** function: $S \times A \rightarrow S$ defines the result of a move
- Terminal_test: True when the game is over, False otherwise. States where game ends are called **terminal states**
- $Utility(s, p)$: **utility function** or objective function for a game that ends in terminal state s for player p . In Chess, the outcome is a win, loss, or draw with values +1, 0, 1/2. For tic-tac-toe we can use a utility of +1, -1, 0.

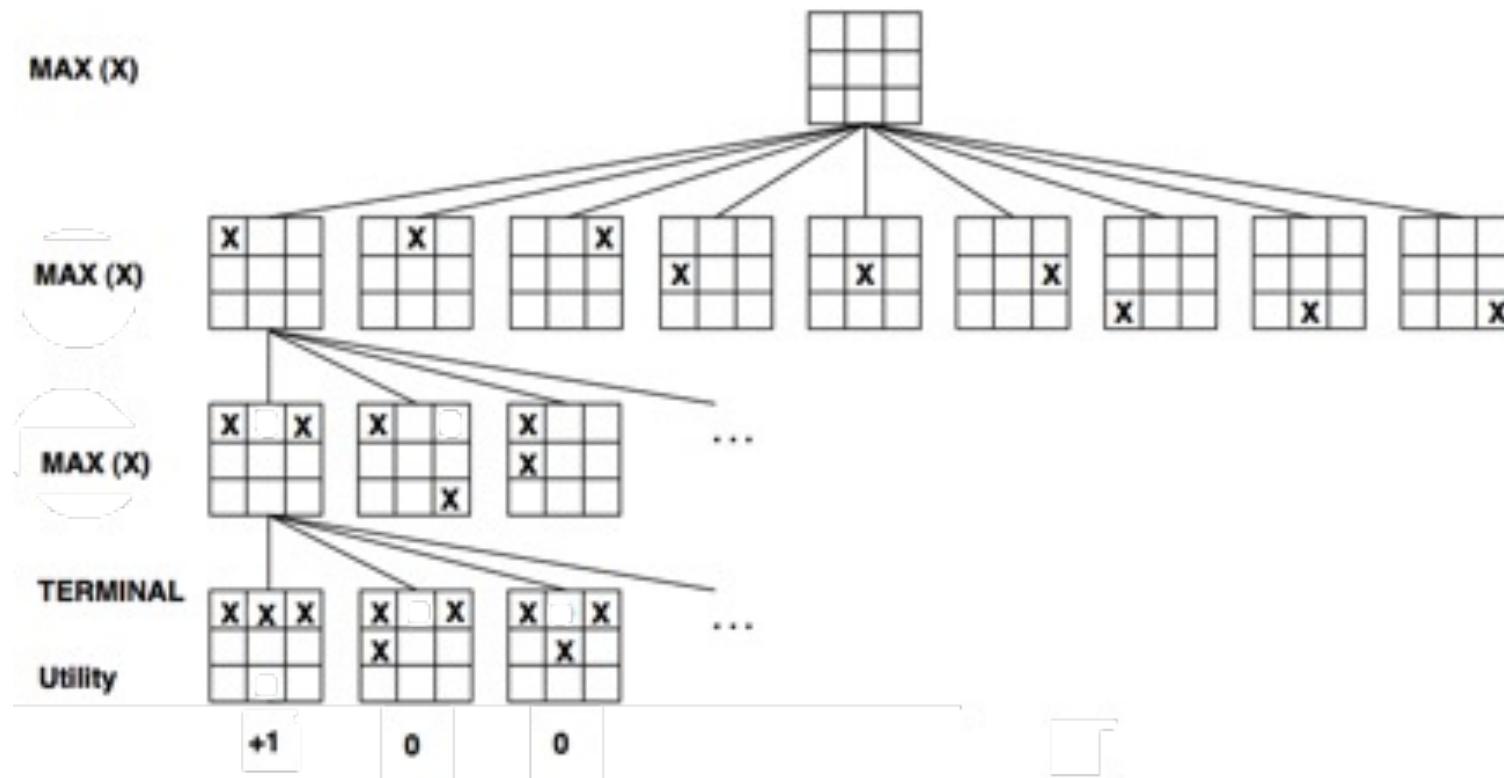
Single player...

Assume we have a tic-tac-toe with one player.

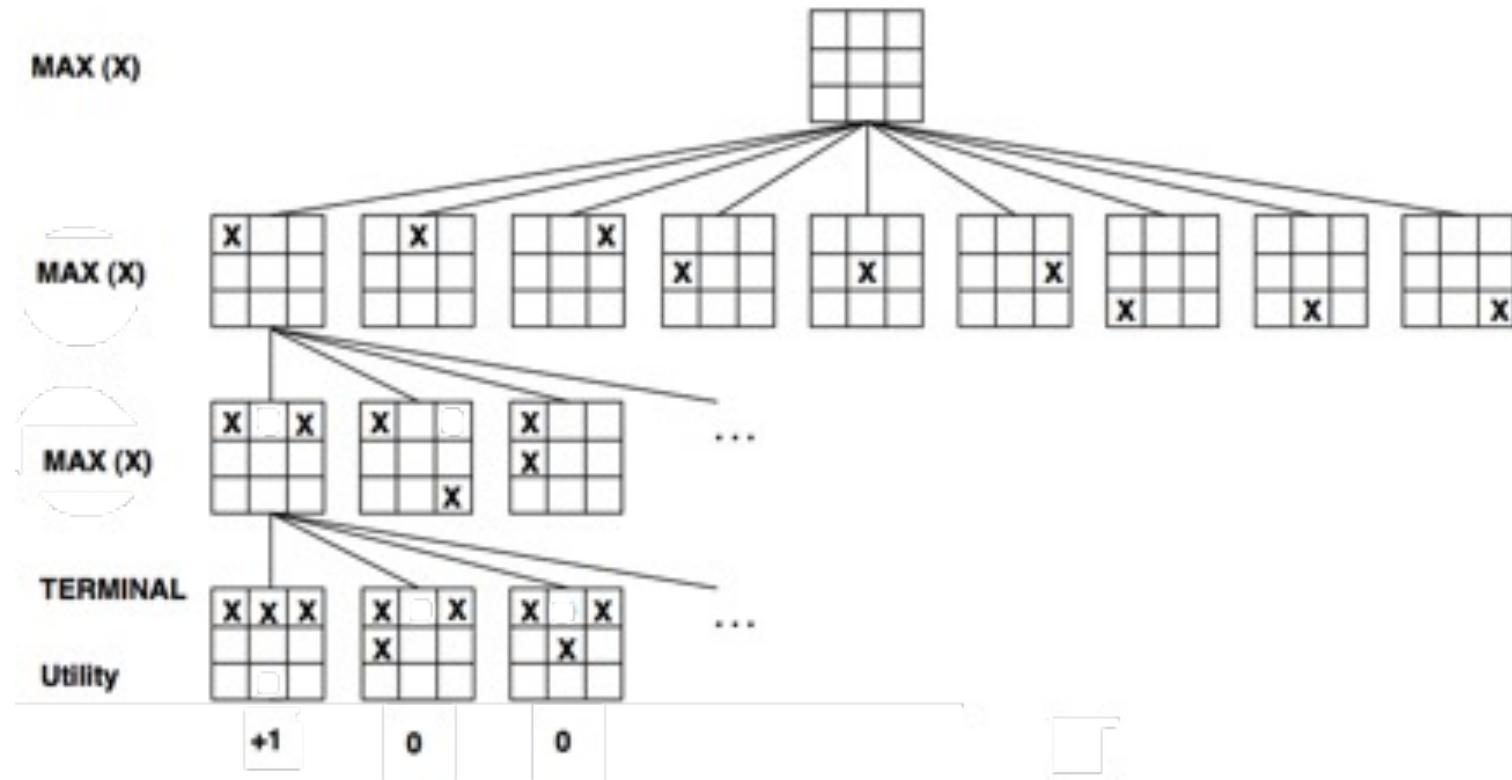
Let's call him Max and have him play three moves only for the sake of the example.



Single player...



Single player...

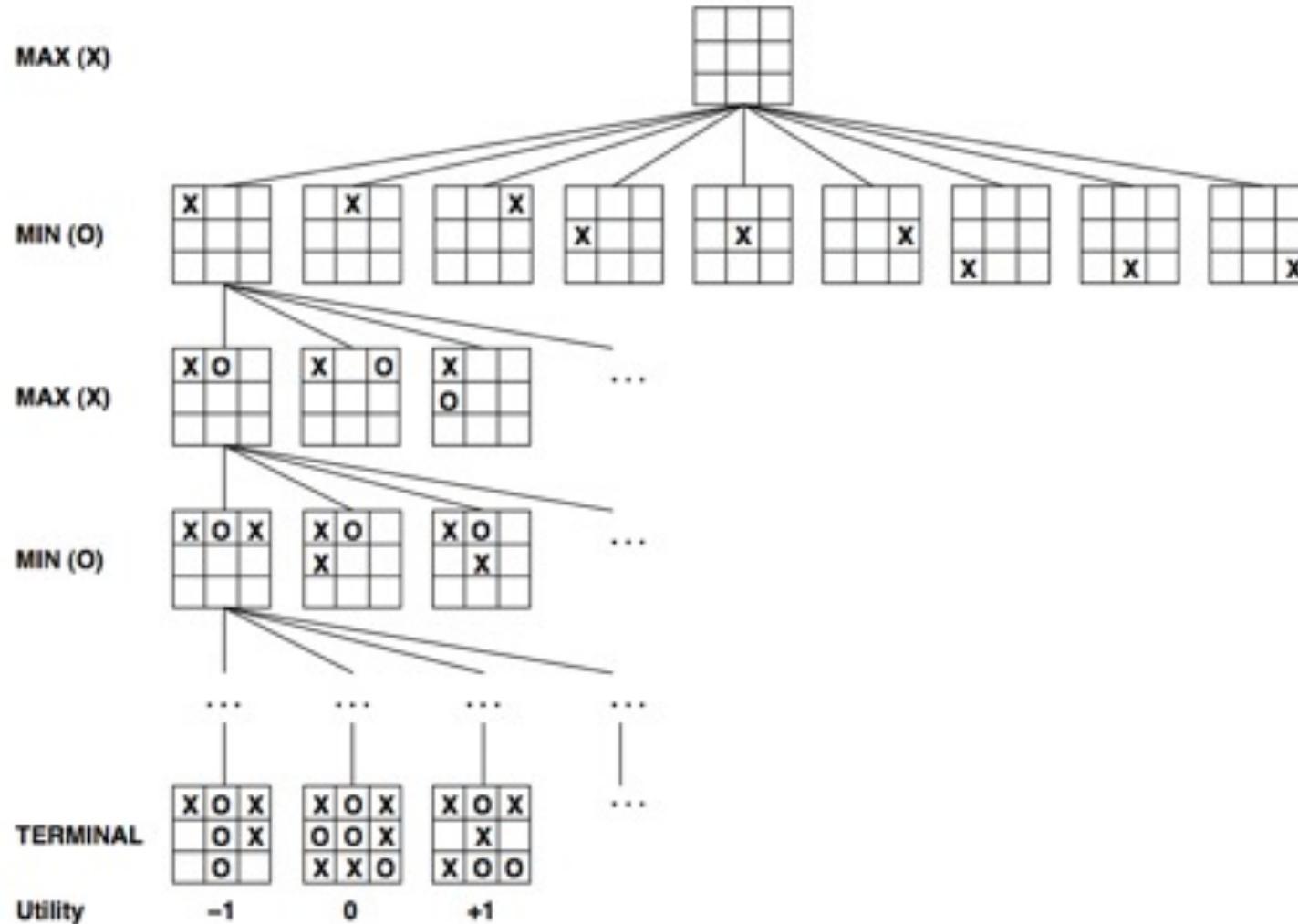


In the case of one player, nothing will prevent Max from winning (choose the path that leads to the desired utility here 1), unless there is another player who will do everything to make Max lose, let's call him Min (the Mean :))

Adversarial search: minimax

- Two players: Max and Min
- Players alternate turns
- Max moves first
- Max maximizes results
- Min minimizes the result
- Compute each node's minimax value's the best achievable utility against an optimal adversary
- Minimax value \equiv best achievable payoff against best play

Minimax example



Adversarial search: minimax

- Find the optimal strategy for Max:
 - Depth-first search of the game tree
 - An optimal leaf node could appear at any depth of the tree
 - Minimax principle: compute the utility of being in a state assuming both players play optimally from there until the end of the game
 - Propagate minimax values up the tree once terminal nodes are discovered

Adversarial search: minimax

- If state is terminal node: Value is utility(state)
- If state is MAX node: Value is highest value of all successor node values (children)
- If state is MIN node: Value is lowest value of all successor node values (children)

Adversarial search: minimax

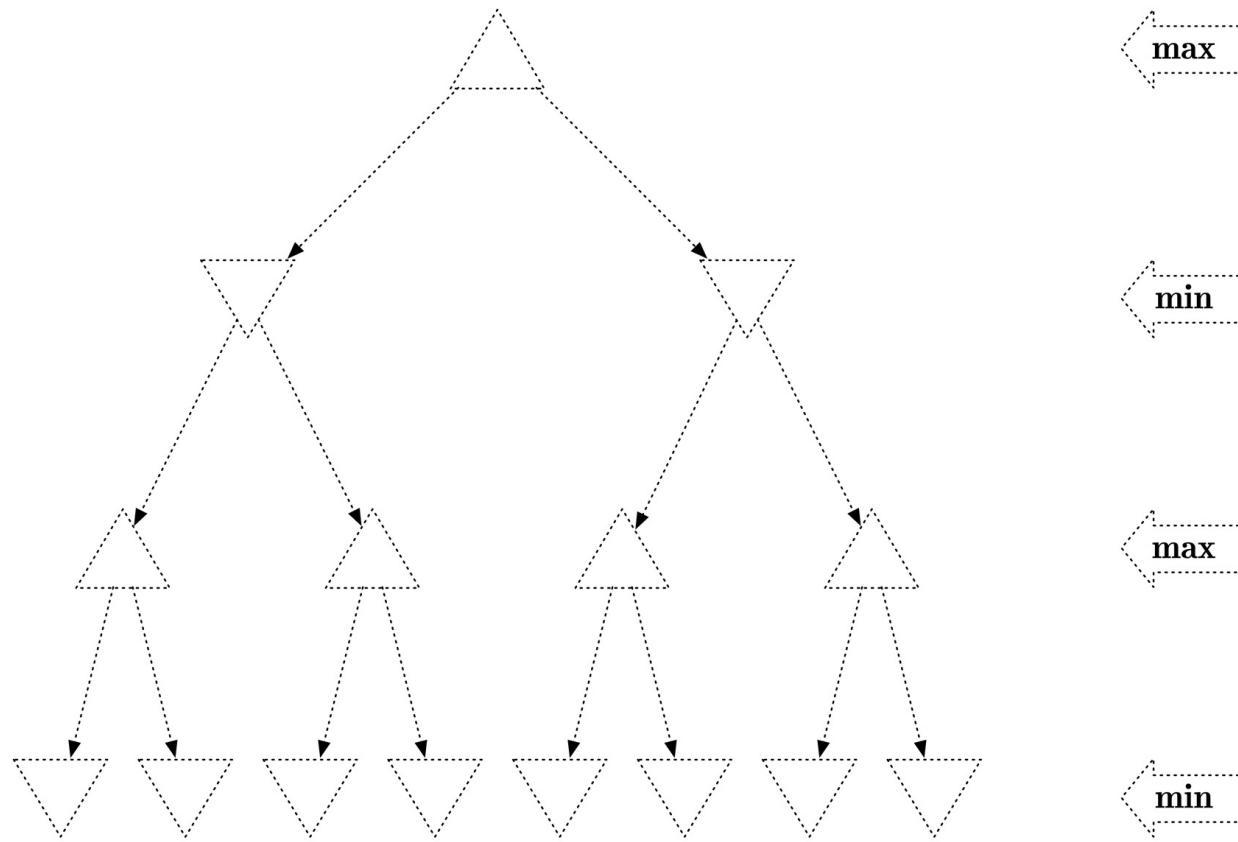
For a state s $\text{minimax}(s) =$

$$\begin{cases} \text{Utility}(s) & \text{if Terminal-test}(s) \\ \max_{a \in \text{Actions}(s)} \text{minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{Max} \\ \min_{a \in \text{Actions}(s)} \text{minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{Min} \end{cases}$$

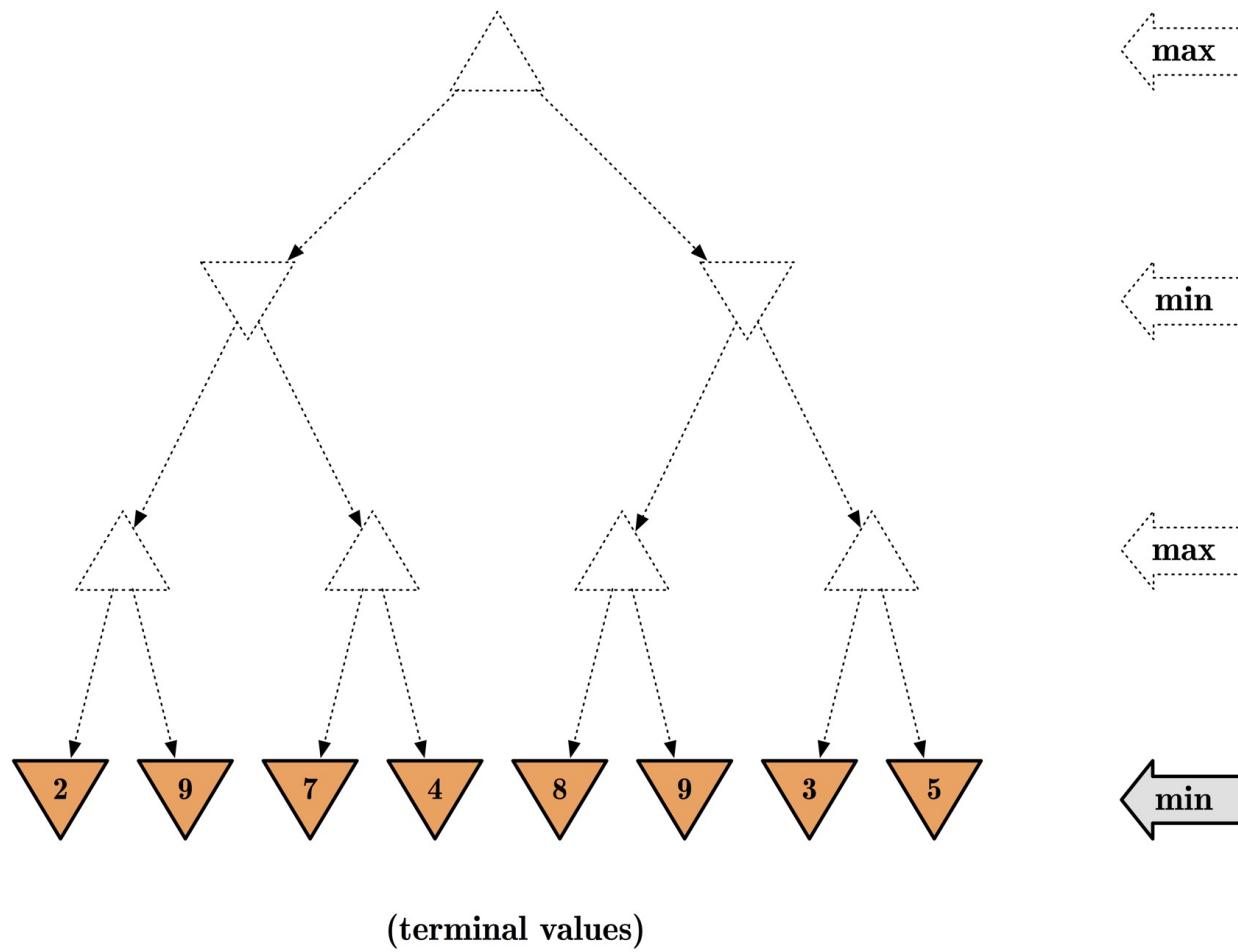
The minimax algorithm

```
/* Find the child state with the lowest utility value */  
  
function MINIMIZE(state)  
    returns TUPLE of <STATE, UTILITY> :  
  
    if TERMINAL-TEST(state):  
        return <NULL, EVAL(state)>  
  
    <minChild, minUtility> = <NULL, ∞>  
  
    for child in state.children():  
        <_, utility> = MAXIMIZE(child)  
  
        if utility < minUtility:  
            <minChild, minUtility> = <child, utility>  
  
return <minChild, minUtility>  
  
/* Find the child state with the highest utility value */  
  
function MAXIMIZE(state)  
    returns TUPLE of <STATE, UTILITY> :  
  
    if TERMINAL-TEST(state):  
        return <NULL, EVAL(state)>  
  
    <maxChild, maxUtility> = <NULL, -∞>  
  
    for child in state.children():  
        <_, utility> = MINIMIZE(child)  
  
        if utility > maxUtility:  
            <maxChild, maxUtility> = <child, utility>  
  
return <maxChild, maxUtility>  
  
/* Find the child state with the highest utility value */  
  
function DECISION(state)  
    returns STATE :  
  
    <child, _> = MAXIMIZE(state)  
  
return child
```

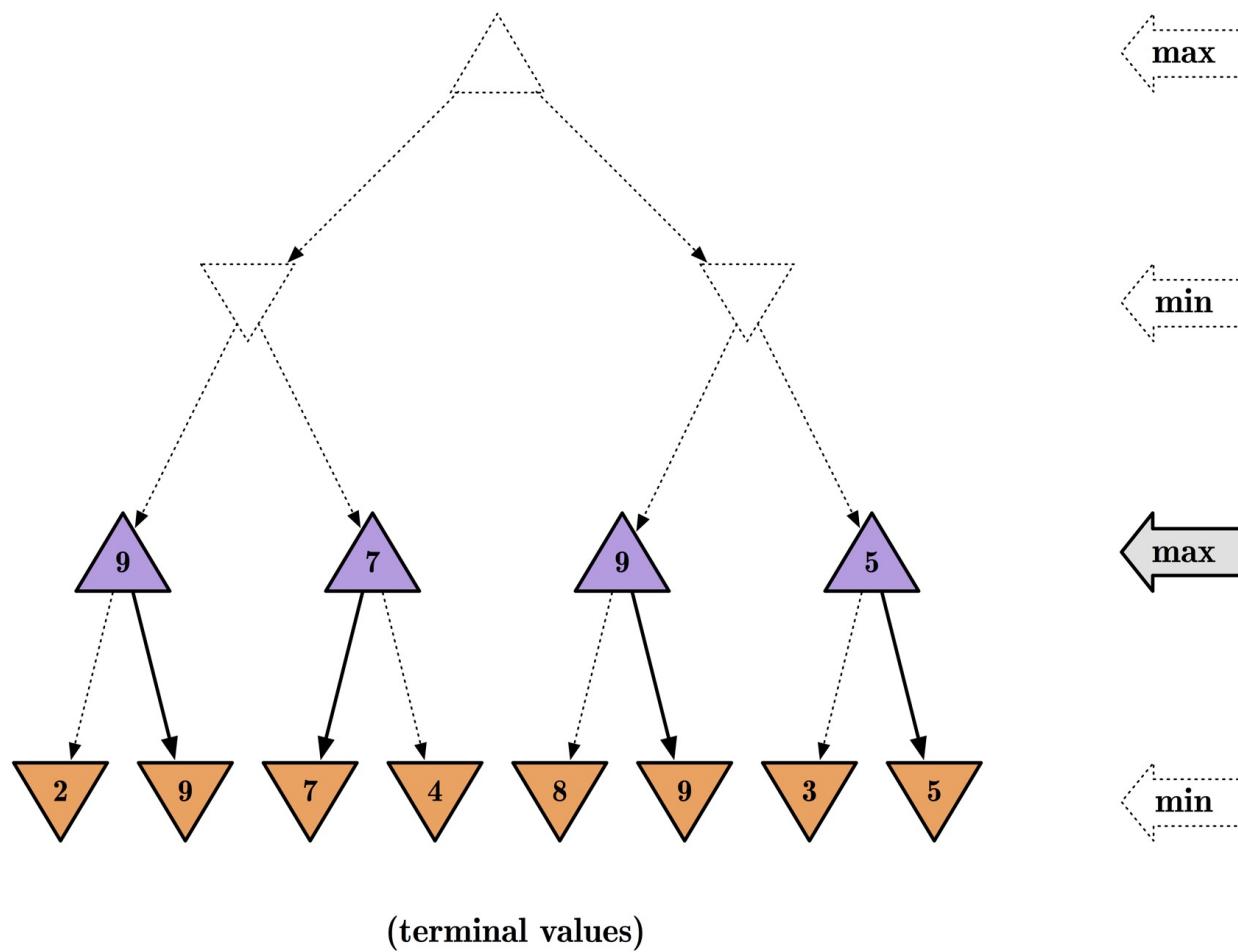
Minimax example



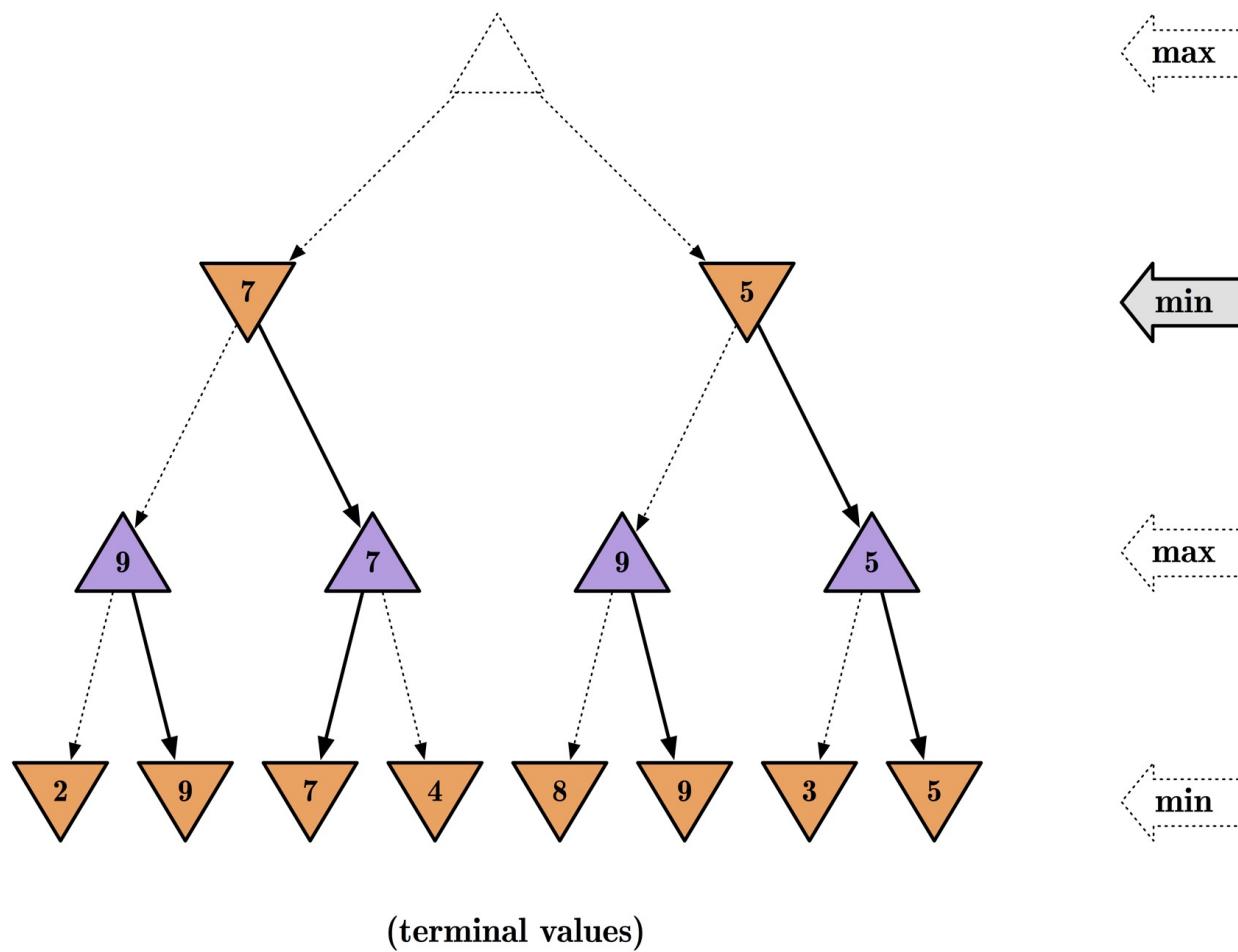
Minimax example



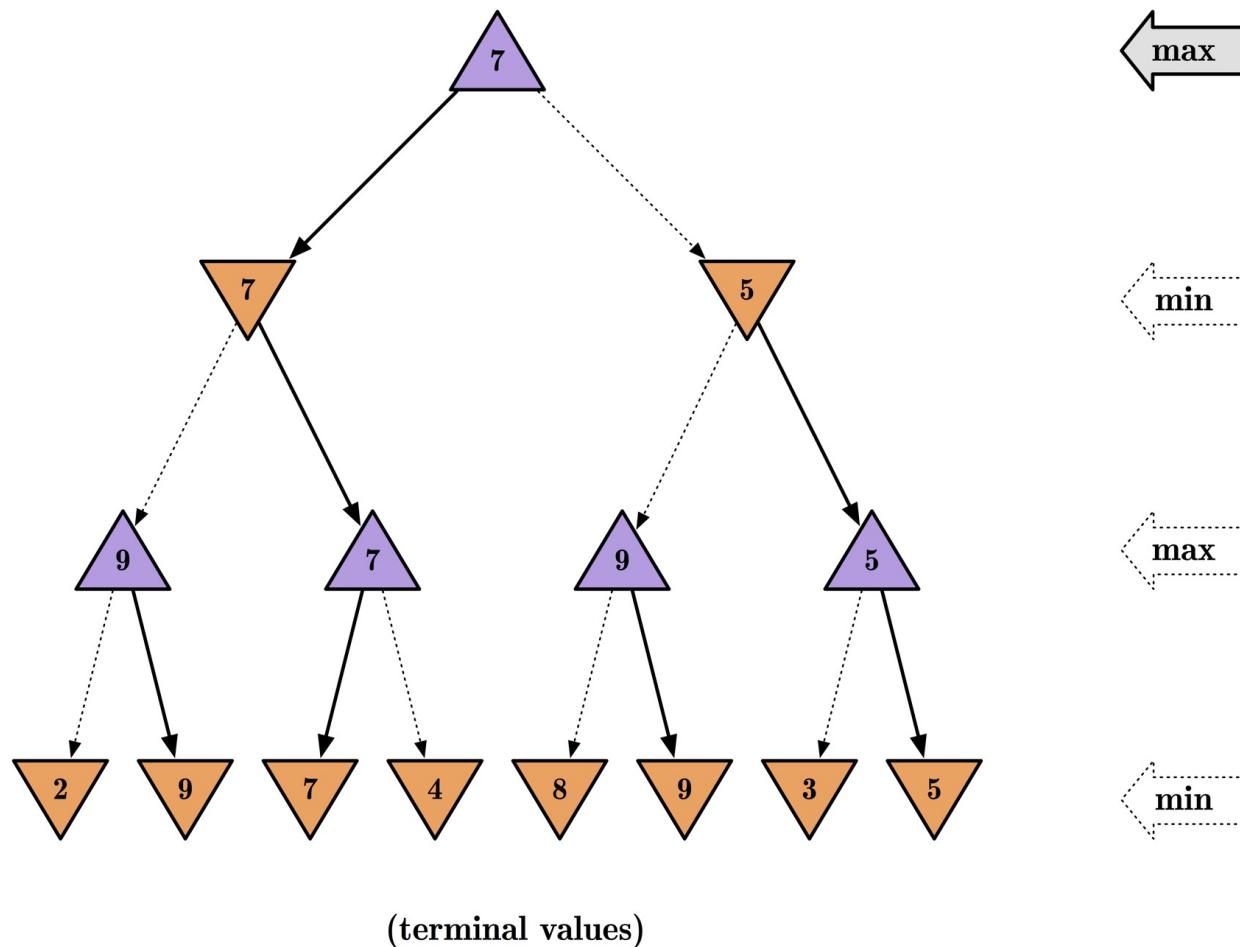
Minimax example



Minimax example



Minimax example



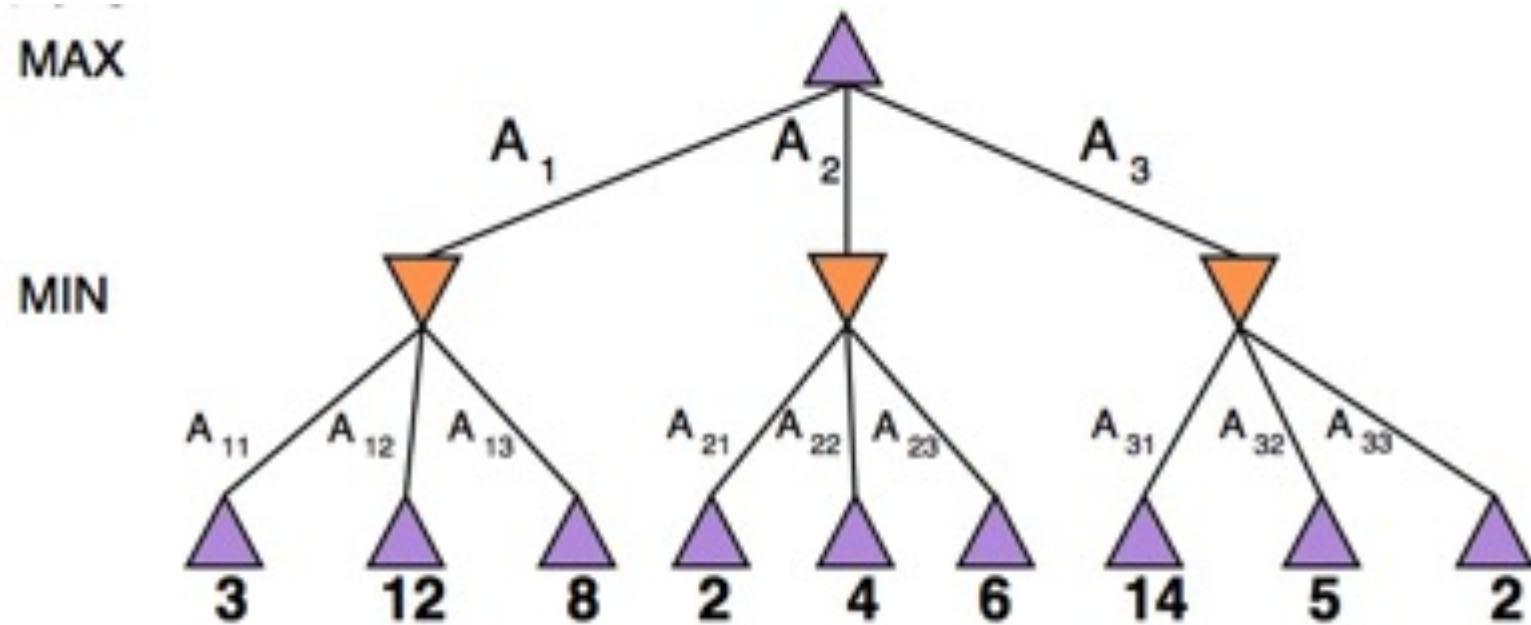
Properties of minimax

- Optimal (opponent plays optimally) and complete (finite tree)
- DFS time: $O(b^m)$
- DFS space: $O(bm)$
 - **Tic-Tac-Toe**
 - * ≈ 5 legal moves on average, total of 9 moves (9 plies).
 - * $5^9 = 1,953,125$
 - * $9! = 362,880$ terminal nodes
 - **Chess**
 - * $b \approx 35$ (average branching factor)
 - * $d \approx 100$ (depth of game tree for a typical game)
 - * $b^d \approx 35^{100} \approx 10^{154}$ nodes
 - **Go** branching factor starts at 361 (19×19 board)

Case of limited resources

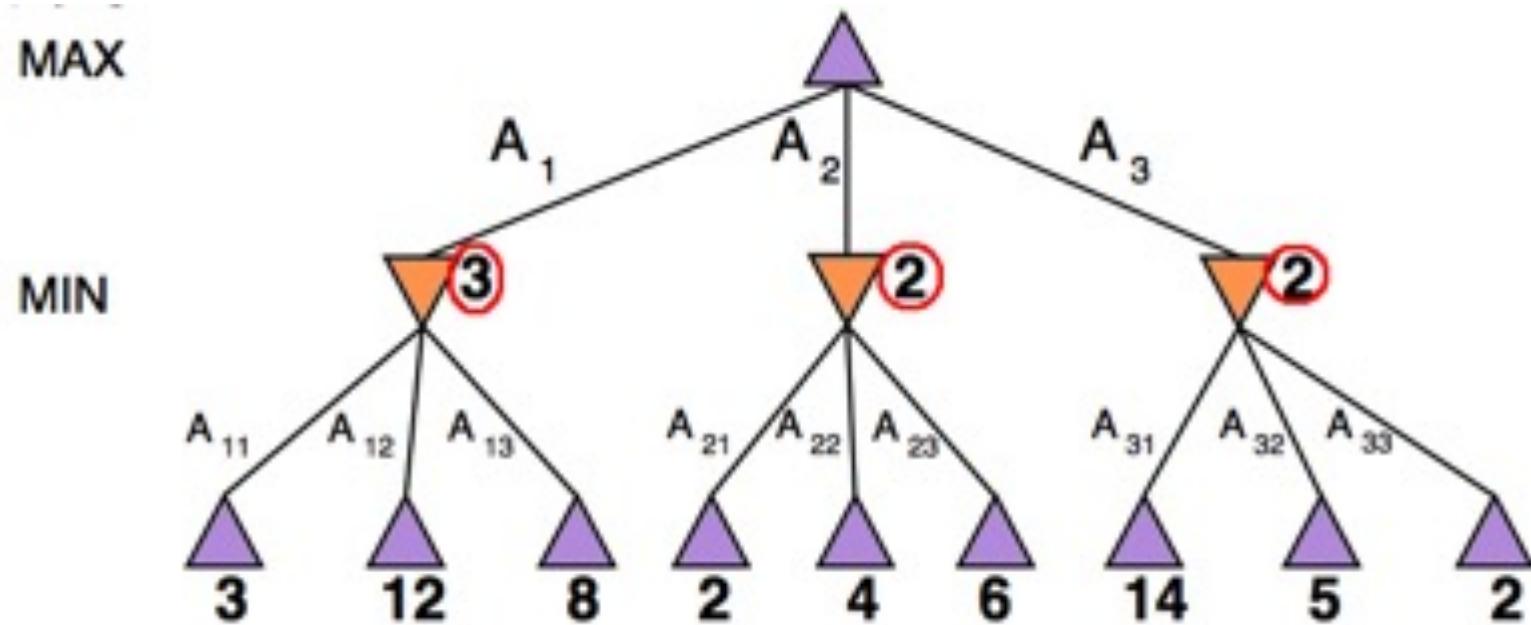
- **Problem:** In real games, we are limited in time, so we can't search the leaves.
 - To be practical and run in a reasonable amount of time, minimax can only search to some depth.
 - More plies make a big difference.
-
- **Solution:**
 1. Replace terminal utilities with an evaluation function for non-terminal positions.
 2. Use Iterative Deepening Search (IDS).
 3. Use pruning: eliminate large parts of the tree.

$\alpha - \beta$ pruning

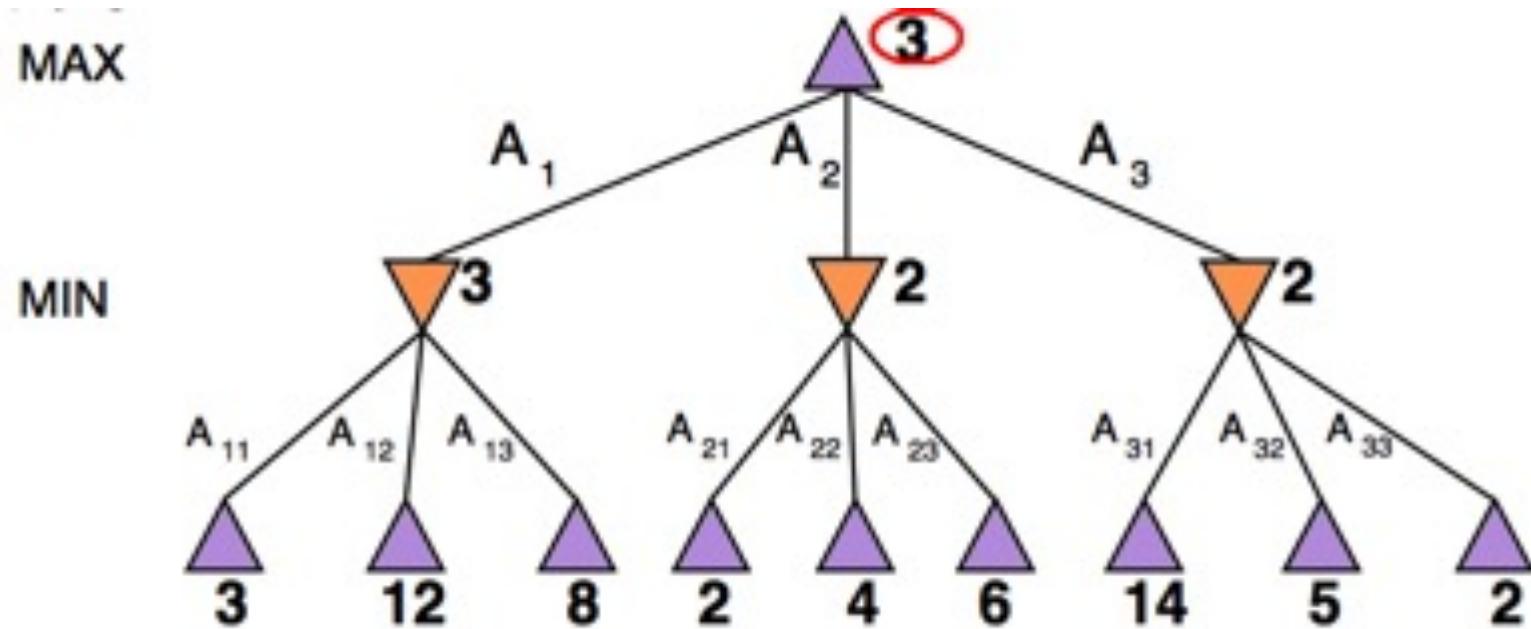


A two-ply game tree.

$\alpha - \beta$ pruning

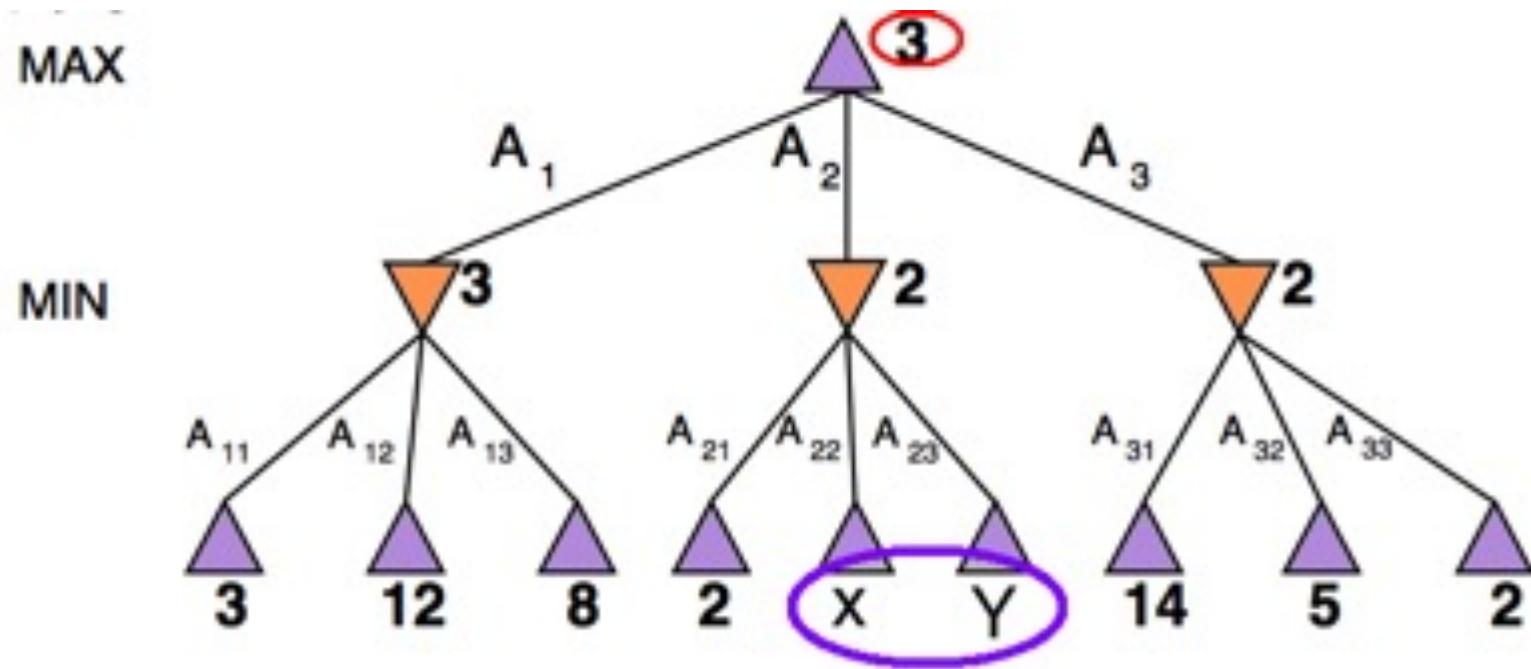


$\alpha - \beta$ pruning

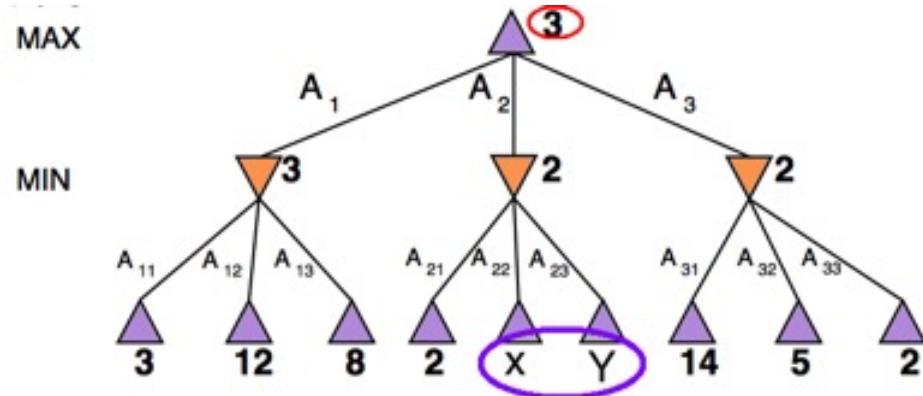


$\alpha - \beta$ pruning

Which values are necessary?

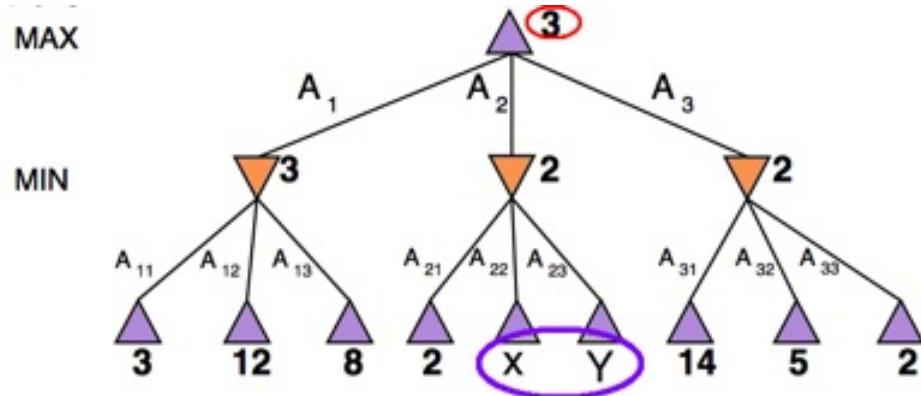


$\alpha - \beta$ pruning



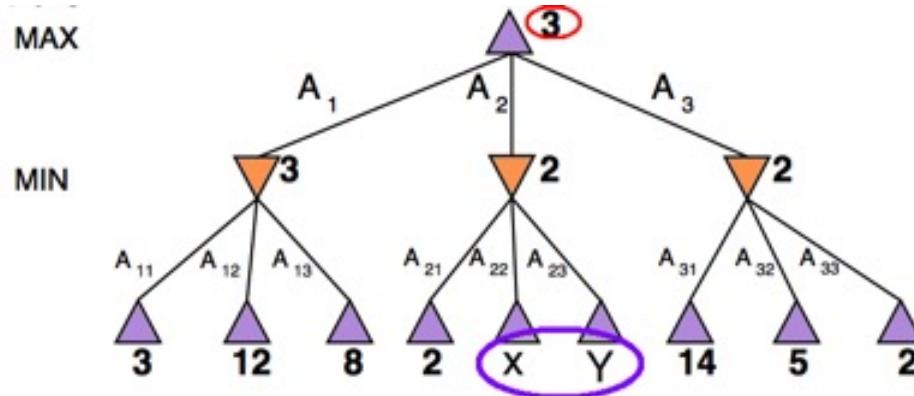
$$\text{Minimax}(\text{root}) = \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2))$$

$\alpha - \beta$ pruning



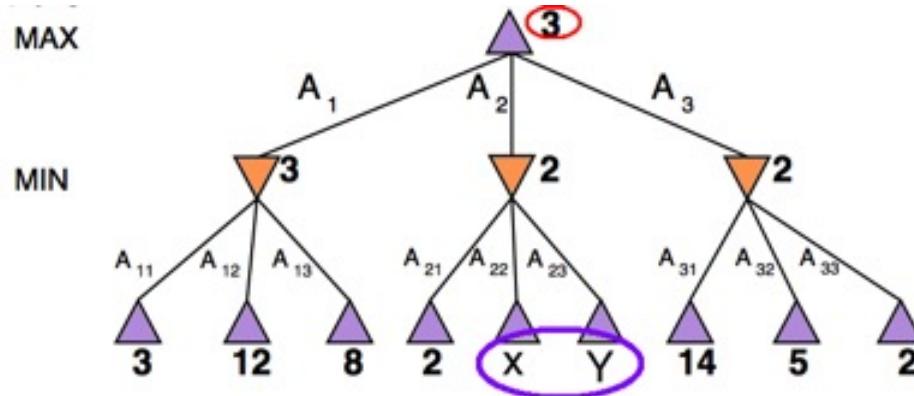
$$\begin{aligned} \text{Minimax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2)) \\ &= \max(3, \min(2, X, Y), 2) \end{aligned}$$

$\alpha - \beta$ pruning



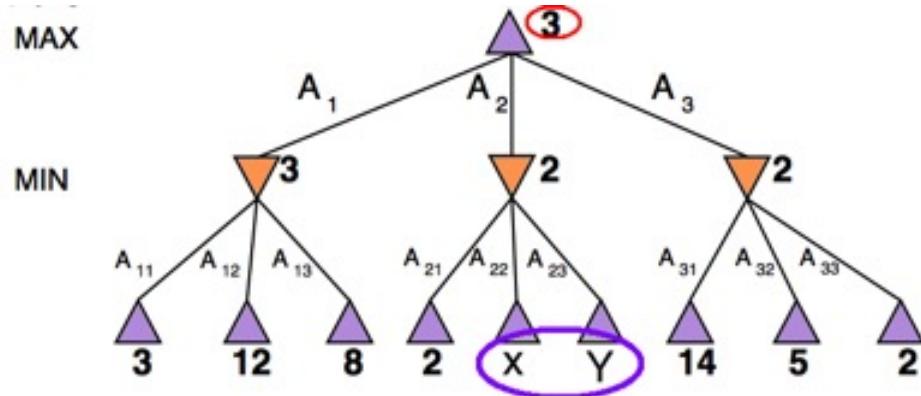
$$\begin{aligned} \text{Minimax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2)) \\ &= \max(3, \min(2, X, Y), 2) \\ &= \max(3, Z, 2) \quad \text{where } Z = \min(2, X, Y) \leq 2 \end{aligned}$$

$\alpha - \beta$ pruning



$$\begin{aligned} \text{Minimax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2)) \\ &= \max(3, \min(2, X, Y), 2) \\ &= \max(3, Z, 2) \quad \text{where } Z = \min(2, X, Y) \leq 2 \\ &= 3 \end{aligned}$$

$\alpha - \beta$ pruning



$$\begin{aligned} \text{Minimax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2)) \\ &= \max(3, \min(2, X, Y), 2) \\ &= \max(3, Z, 2) \quad \text{where } Z = \min(2, X, Y) \leq 2 \\ &= 3 \end{aligned}$$

Minimax decisions are independent of the values of X and Y .

$\alpha - \beta$ pruning

- **Strategy:** Just like minimax, it performs a DFS.

$\alpha - \beta$ pruning

- **Strategy:** Just like minimax, it performs a DFS.
- **Parameters:** Keep track of two bounds
 - α : largest value for Max across seen children (current lower bound on MAX's outcome).
 - β : lowest value for MIN across seen children (current upper bound on MIN's outcome).

$\alpha - \beta$ pruning

- **Strategy:** Just like minimax, it performs a DFS.
- **Parameters:** Keep track of two bounds
 - α : largest value for Max across seen children (current lower bound on MAX's outcome).
 - β : lowest value for MIN across seen children (current upper bound on MIN's outcome).
- **Initialization:** $\alpha = -\infty$, $\beta = \infty$

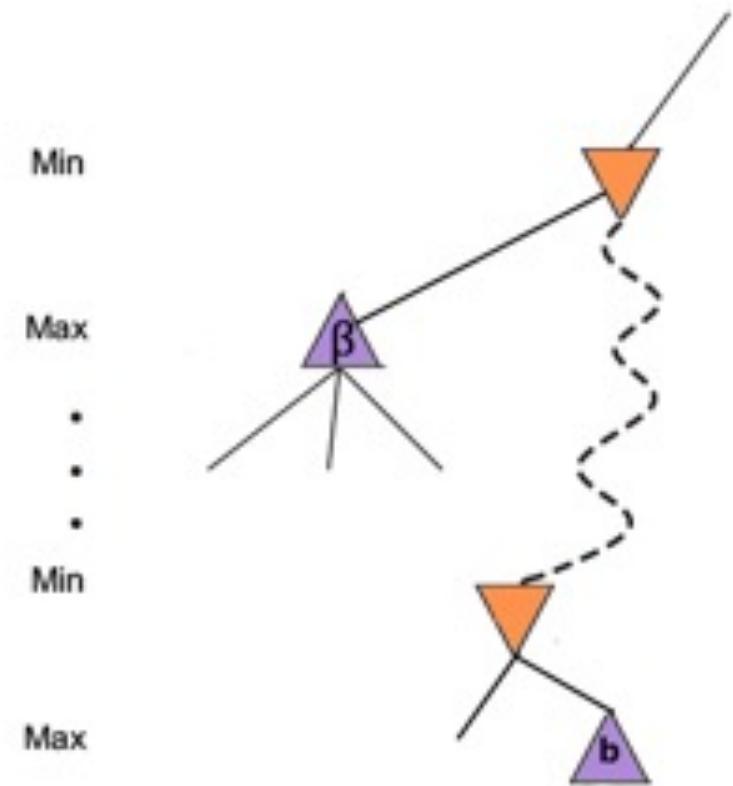
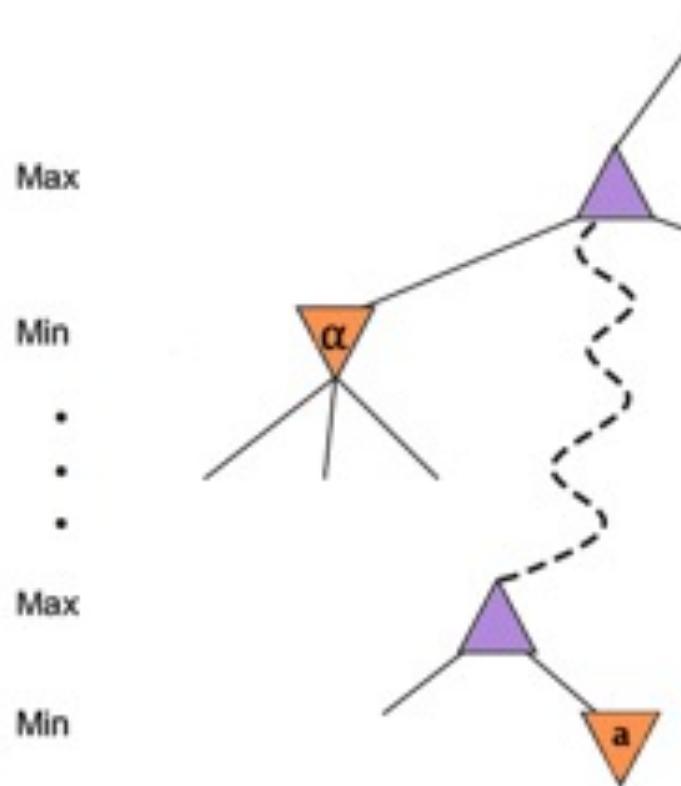
$\alpha - \beta$ pruning

- **Strategy:** Just like minimax, it performs a DFS.
- **Parameters:** Keep track of two bounds
 - α : largest value for Max across seen children (current lower bound on MAX's outcome).
 - β : lowest value for MIN across seen children (current upper bound on MIN's outcome).
- **Initialization:** $\alpha = -\infty$, $\beta = \infty$
- **Propagation:** Send α , β values *down* during the search to be used for pruning.
 - Update α , β values by *propagating upwards* values of terminal nodes.
 - Update α only at Max nodes and update β only at Min nodes.

$\alpha - \beta$ pruning

- **Strategy:** Just like minimax, it performs a DFS.
- **Parameters:** Keep track of two bounds
 - α : largest value for Max across seen children (current lower bound on MAX's outcome).
 - β : lowest value for MIN across seen children (current upper bound on MIN's outcome).
- **Initialization:** $\alpha = -\infty$, $\beta = \infty$
- **Propagation:** Send α , β values *down* during the search to be used for pruning.
 - Update α , β values by *propagating upwards* values of terminal nodes.
 - Update α only at Max nodes and update β only at Min nodes.
- **Pruning:** Prune any remaining branches whenever $\alpha \geq \beta$.

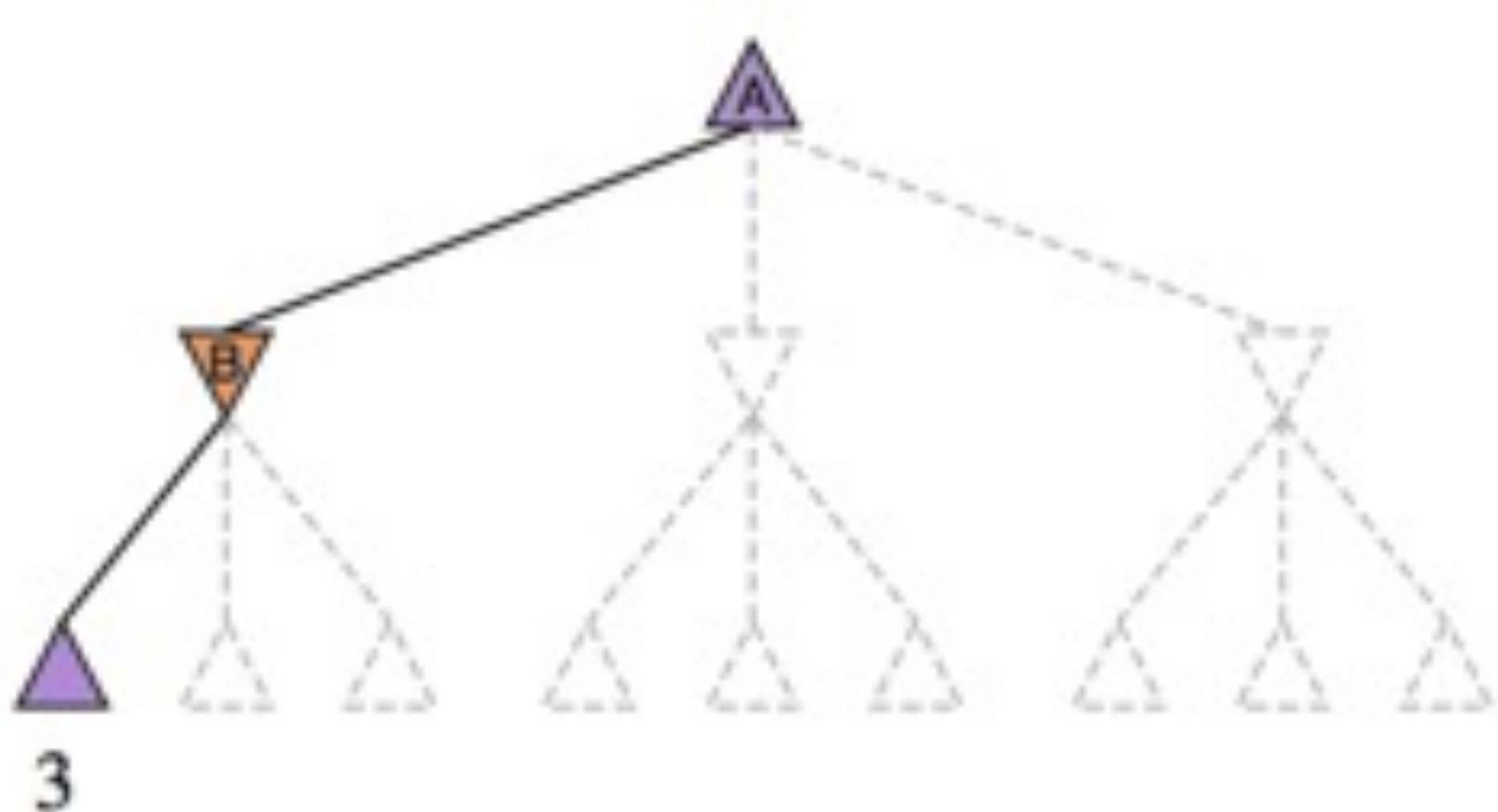
$\alpha - \beta$ pruning



- If α is better than a for Max, then Max will avoid it, that is prune that branch.
- If β is better than b for Min, then Min will avoid it, that is prune that branch.

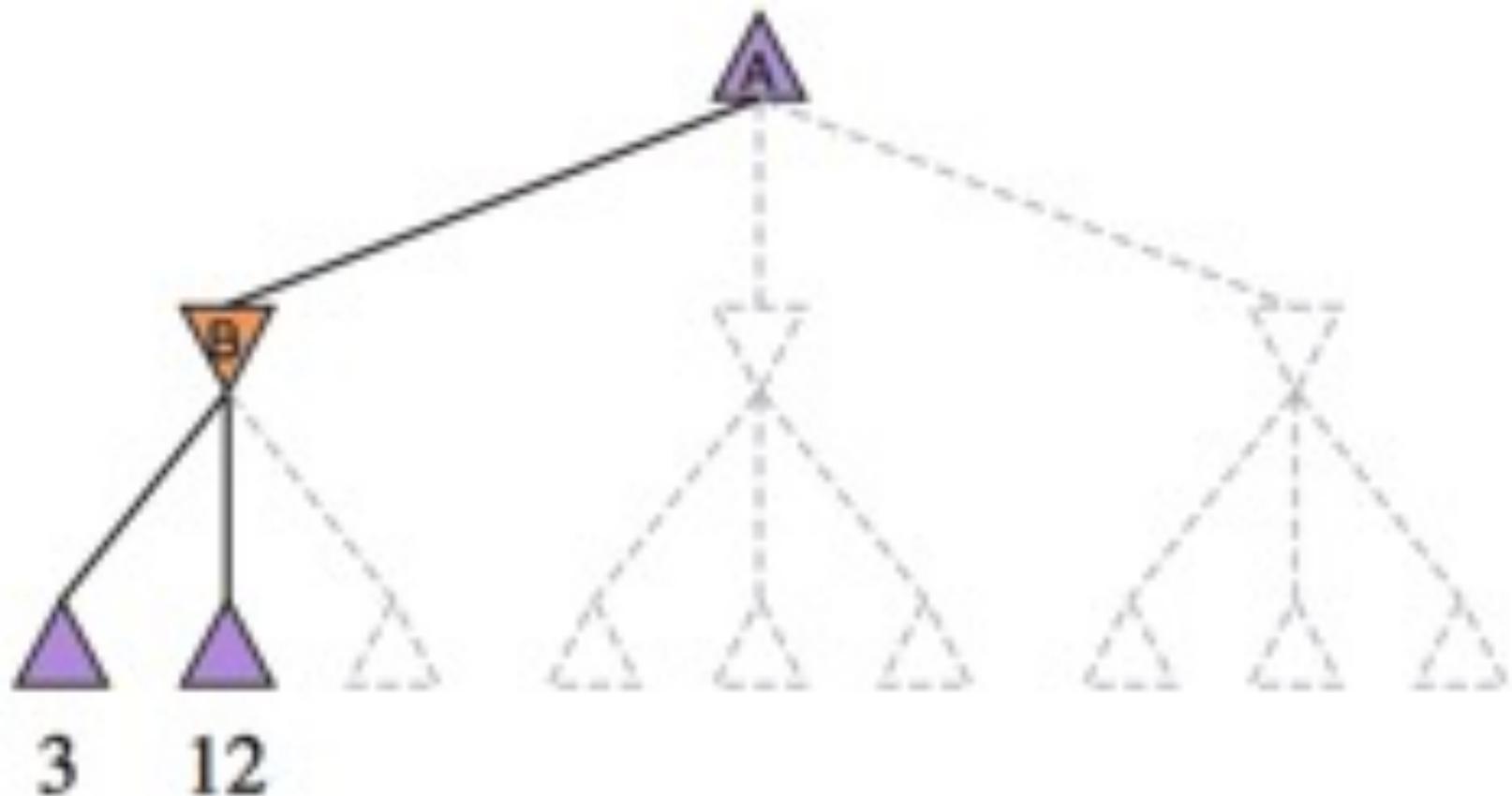
$\alpha - \beta$ pruning

(a)



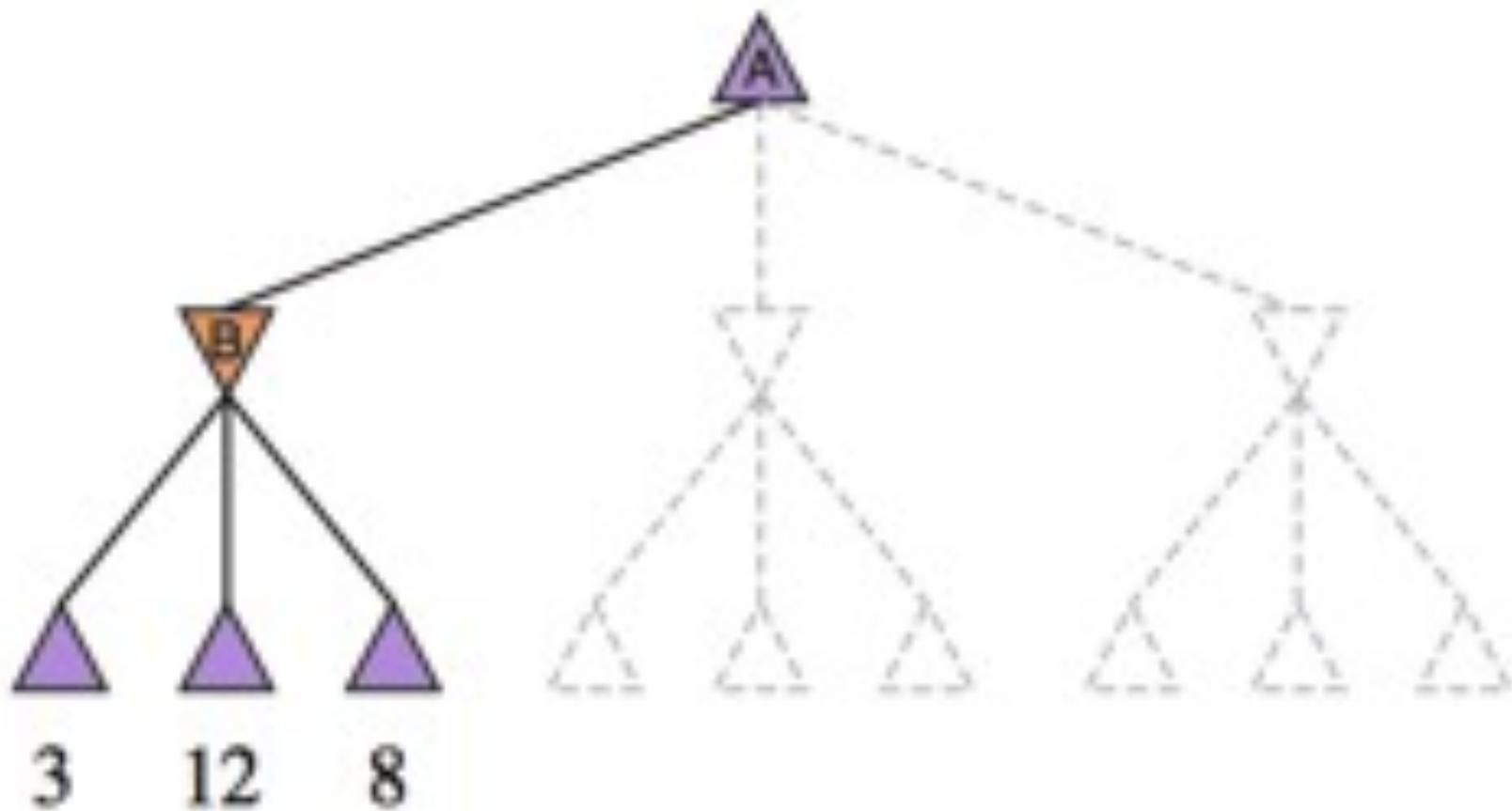
$\alpha - \beta$ pruning

(b)



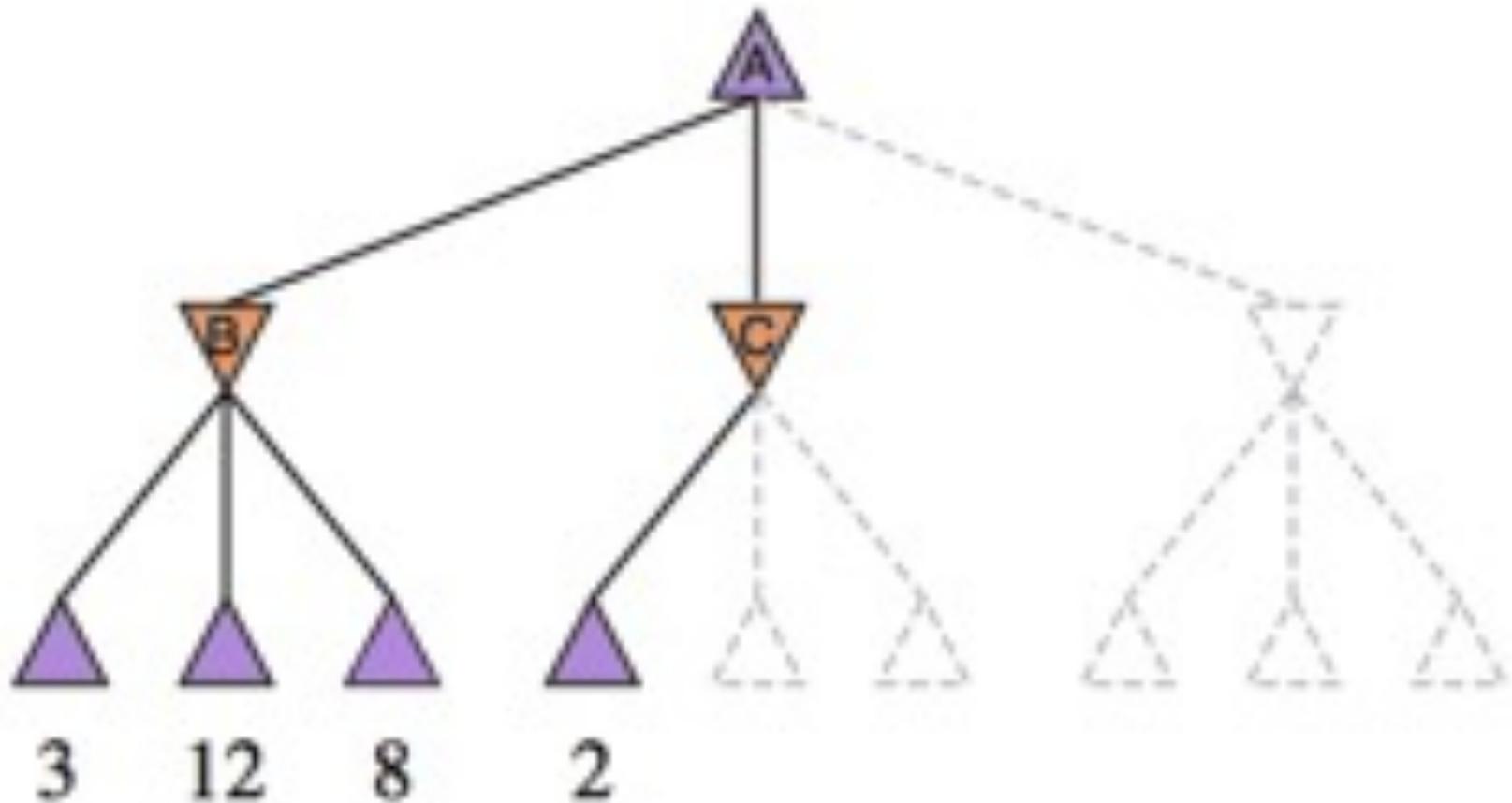
$\alpha - \beta$ pruning

(c)



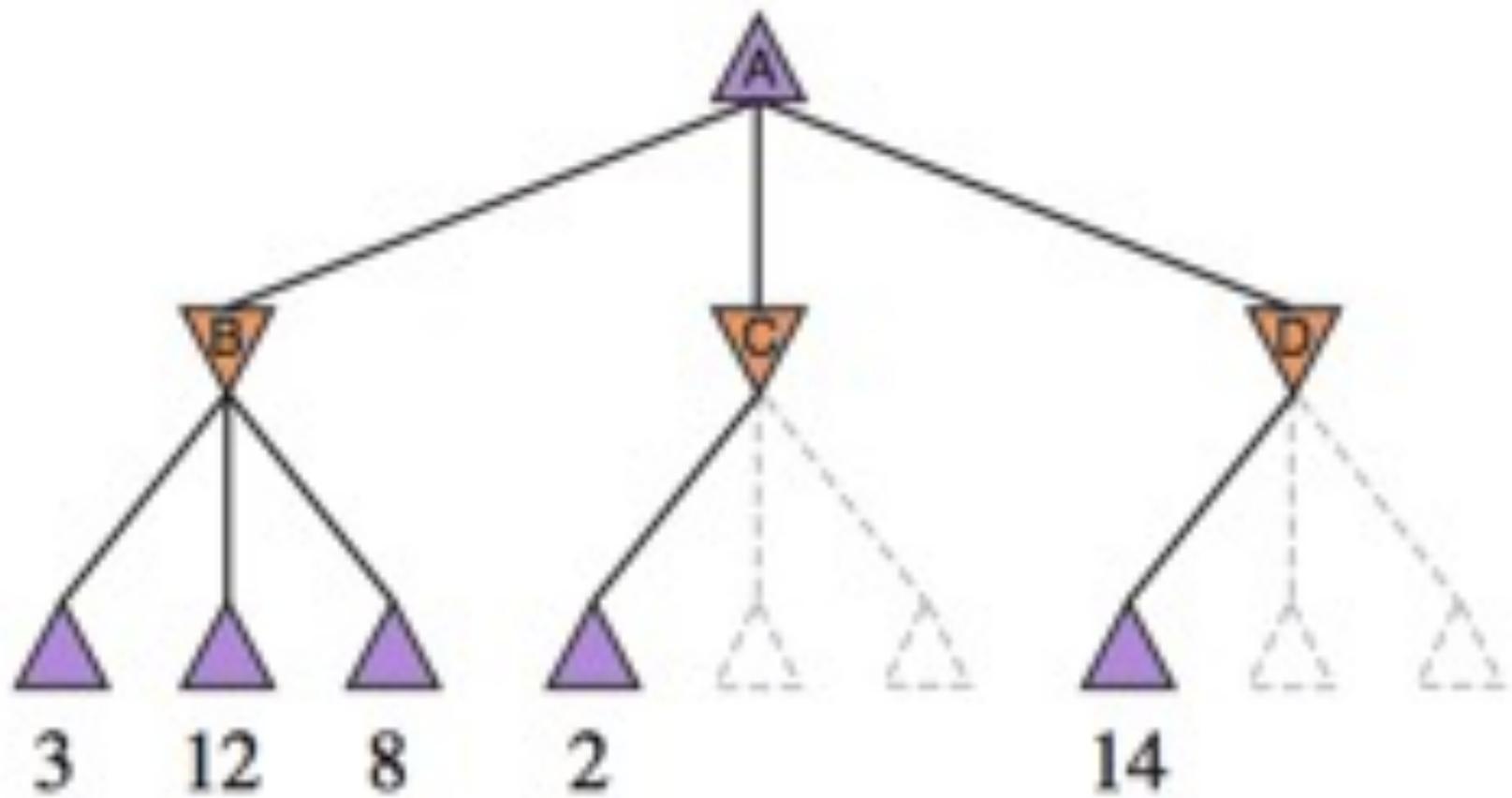
$\alpha - \beta$ pruning

(d)



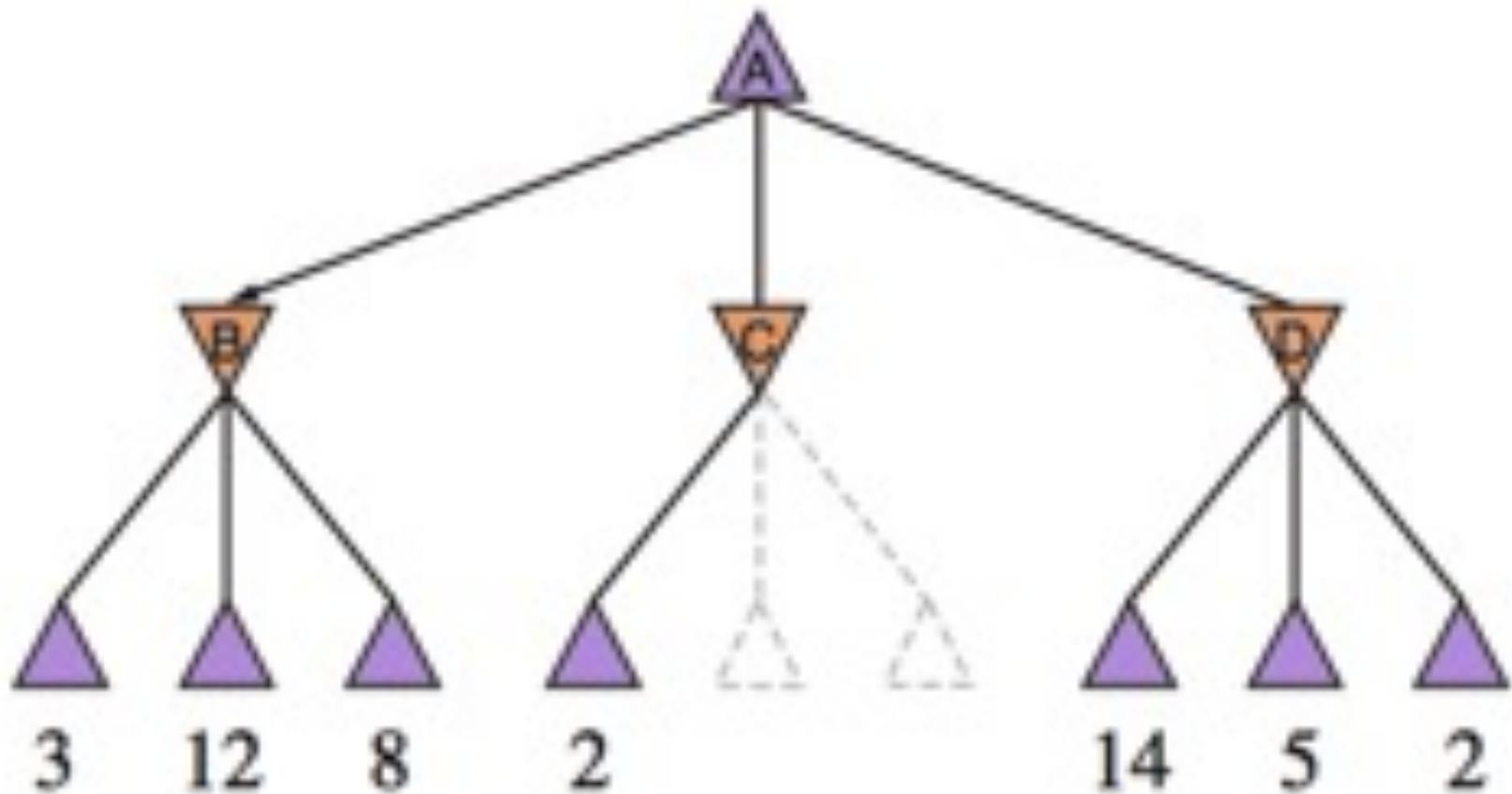
$\alpha - \beta$ pruning

(e)

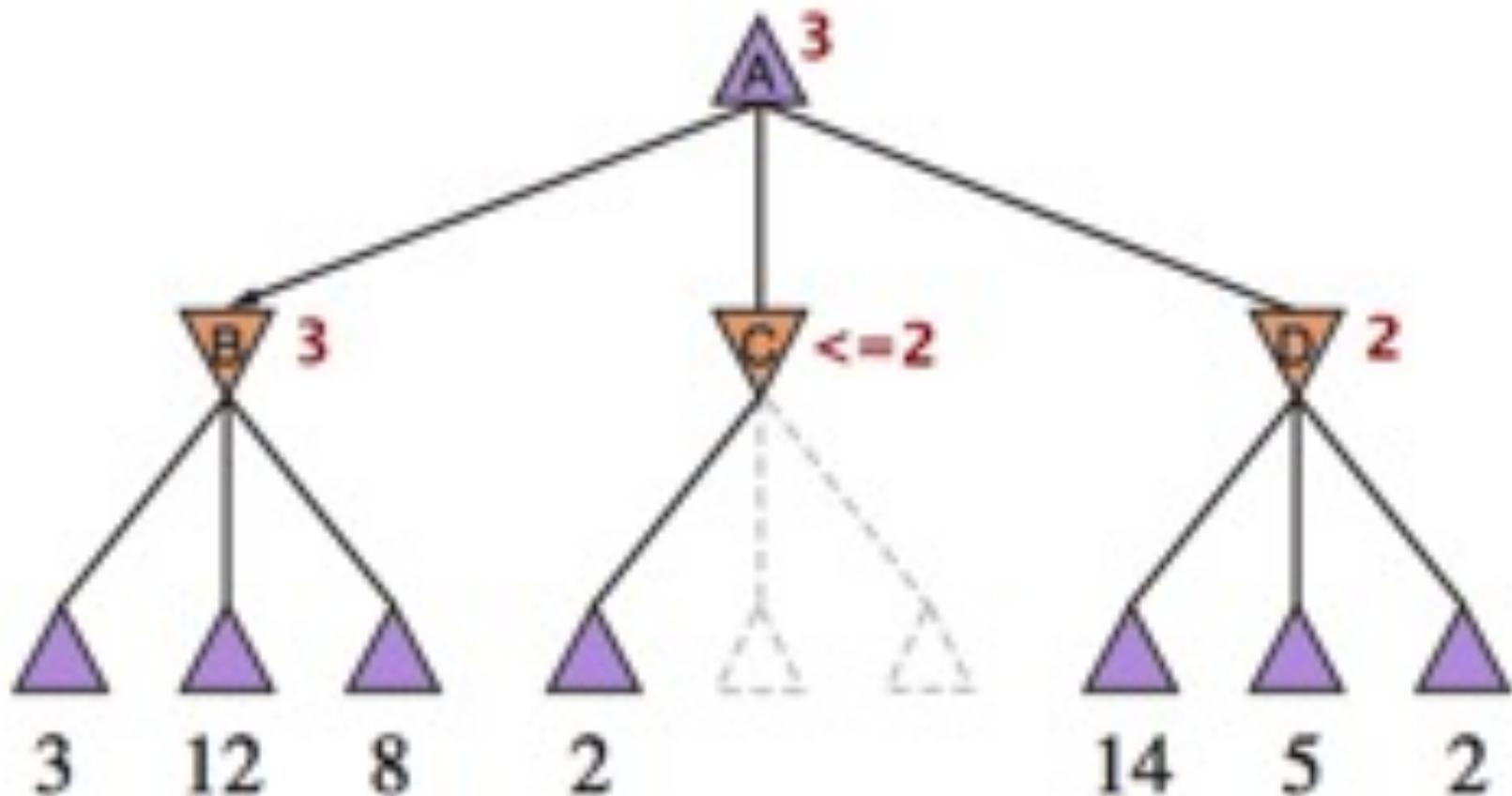


$\alpha - \beta$ pruning

(f)



$\alpha - \beta$ pruning



$\alpha - \beta$ pruning

```
/* Find the child state with the lowest utility value */

function MINIMIZE(state,  $\alpha$ ,  $\beta$ )
  returns TUPLE of <STATE, UTILITY> :

  if TERMINAL-TEST(state):
    return <NULL, EVAL(state)>

  <minChild, minUtility> = <NULL,  $\infty$ >

  for child in state.children():
    <_, utility> = MAXIMIZE(child,  $\alpha$ ,  $\beta$ )

    if utility < minUtility:
      <minChild, minUtility> = <child, utility>

    if minUtility  $\leq \alpha$ :
      break

    if minUtility <  $\beta$ :
       $\beta$  = minUtility

  return <minChild, minUtility>

  /* Find the child state with the highest utility value */

function MAXIMIZE(state,  $\alpha$ ,  $\beta$ )
  returns TUPLE of <STATE, UTILITY> :

  if TERMINAL-TEST(state):
    return <NULL, EVAL(state)>

  <maxChild, maxUtility> = <NULL,  $-\infty$ >

  for child in state.children():
    <_, utility> = MINIMIZE(child,  $\alpha$ ,  $\beta$ )

    if utility > maxUtility:
      <maxChild, maxUtility> = <child, utility>

    if maxUtility  $\geq \beta$ :
      break

    if maxUtility >  $\alpha$ :
       $\alpha$  = maxUtility

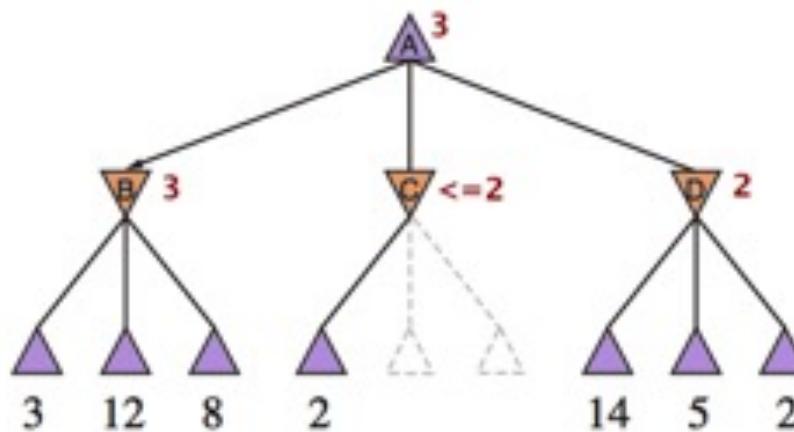
  return <maxChild, maxUtility>

function DECISION(state)
  returns STATE :

  <child, _> = MAXIMIZE(state,  $-\infty$ ,  $\infty$ )

  return child
```

Move ordering



- It does matter as it affects the effectiveness of $\alpha - \beta$ pruning.
- Example: We could not prune any successor of D because the worst successors for Min were generated first. If the third one (leaf 2) was generated first we would have pruned the two others (14 and 5).
- Idea of ordering: examine first successors that are likely best.

Move ordering

- **Worst ordering:** no pruning happens (best moves are on the right of the game tree). Complexity $O(b^m)$.
- **Ideal ordering:** lots of pruning happens (best moves are on the left of the game tree). This solves tree twice as deep as minimax in the same amount of time. Complexity $O(b^{m/2})$ (in practice). The search can go deeper in the game tree.
- **How to find a good ordering?**
 - Remember the best moves from shallowest nodes.
 - Order the nodes so as the best are checked first.
 - Use domain knowledge: e.g., for chess, try order: captures first, then threats, then forward moves, backward moves.
 - Bookkeep the states, they may repeat!

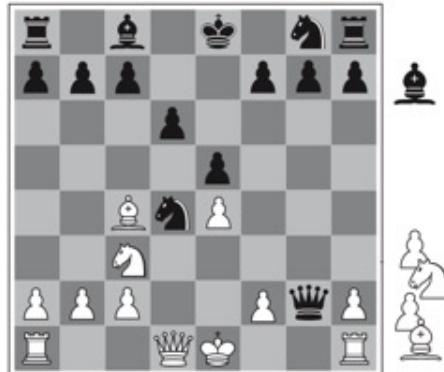
Real-time decisions

- Minimax: generates the entire game search space
- $\alpha - \beta$ algorithm: prune large chunks of the trees
- BUT $\alpha - \beta$ still has to go all the way to the leaves
- Impractical in real-time (moves has to be done in a reasonable amount of time)
- Solution: bound the depth of search (cut off search) and **replace** $utility(s)$ **with** $eval(s)$, an evaluation function to **estimate** value of current board configurations

Real-time decisions

- $\text{eval}(s)$ is a heuristic at state s
E.g., Othello: white pieces - black pieces
E.g., Chess: Value of all white pieces - Value of all black pieces
turn non-terminal nodes into terminal leaves!
- An ideal evaluation function would rank terminal states in the same way as the true utility function; but must be fast
- Typical to define features, make the function a linear weighted sum of the features
- Use domain knowledge to craft the best and useful features.

Real-time decisions



- How does it works?
 - Select useful features f_1, \dots, f_n e.g., Chess: # pieces on board, value of pieces (1 for pawn, 3 for bishop, etc.)
 - Weighted linear function:

$$eval(s) = \sum_{i=1}^n w_i f_i(s)$$

- Learn w_i from the examples
- Deep blue uses about 6,000 features!

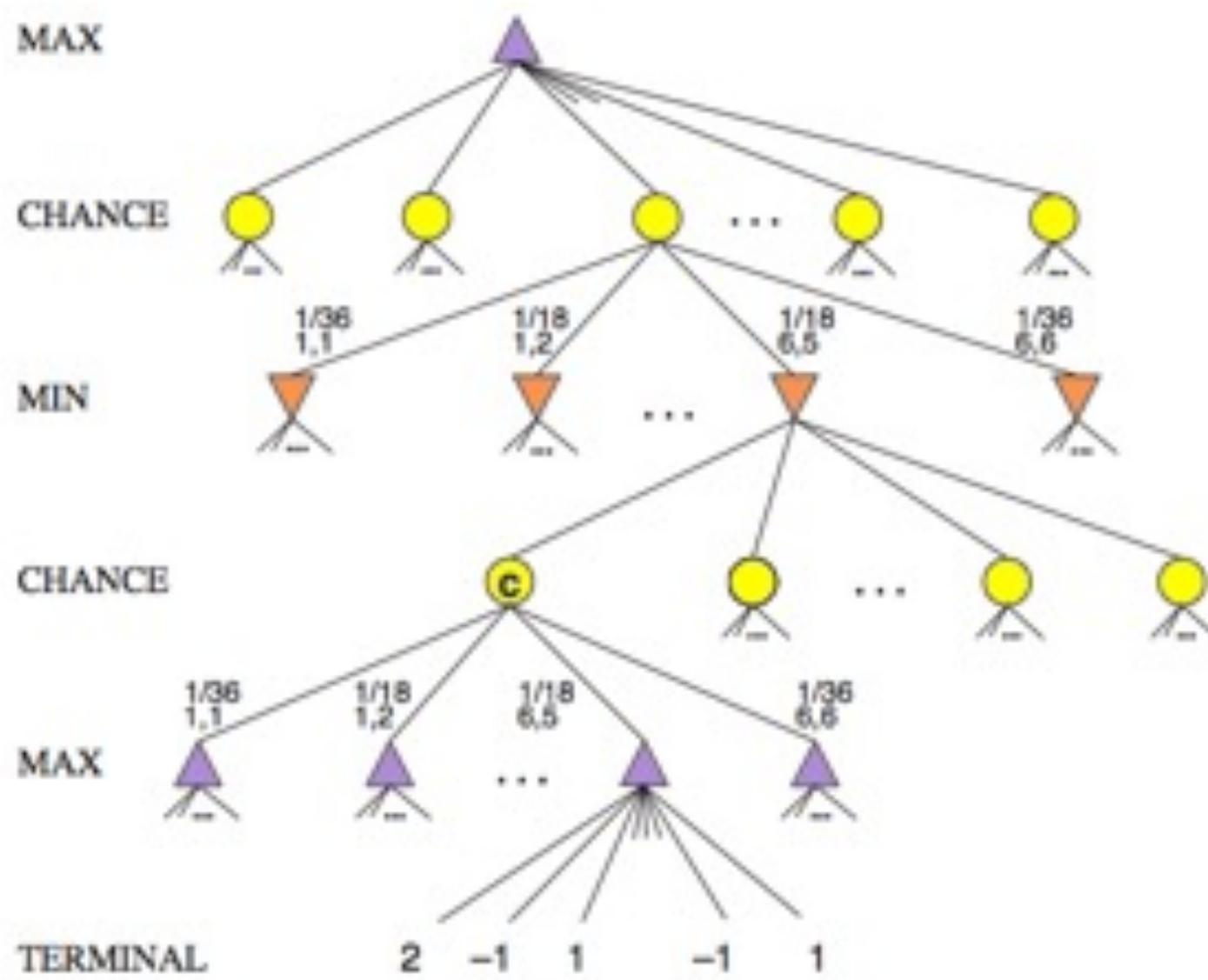
Stochastic games

- Include a random element (e.g., throwing a die).
- Include chance nodes.
- Backgammon: old board game combining skills and chance.
- The goal is that each player tries to move all of his pieces off the board before his opponent does.



Ptkfgs [Public domain], via Wikimedia Commons

Stochastic games



Partial game tree for Backgammon.

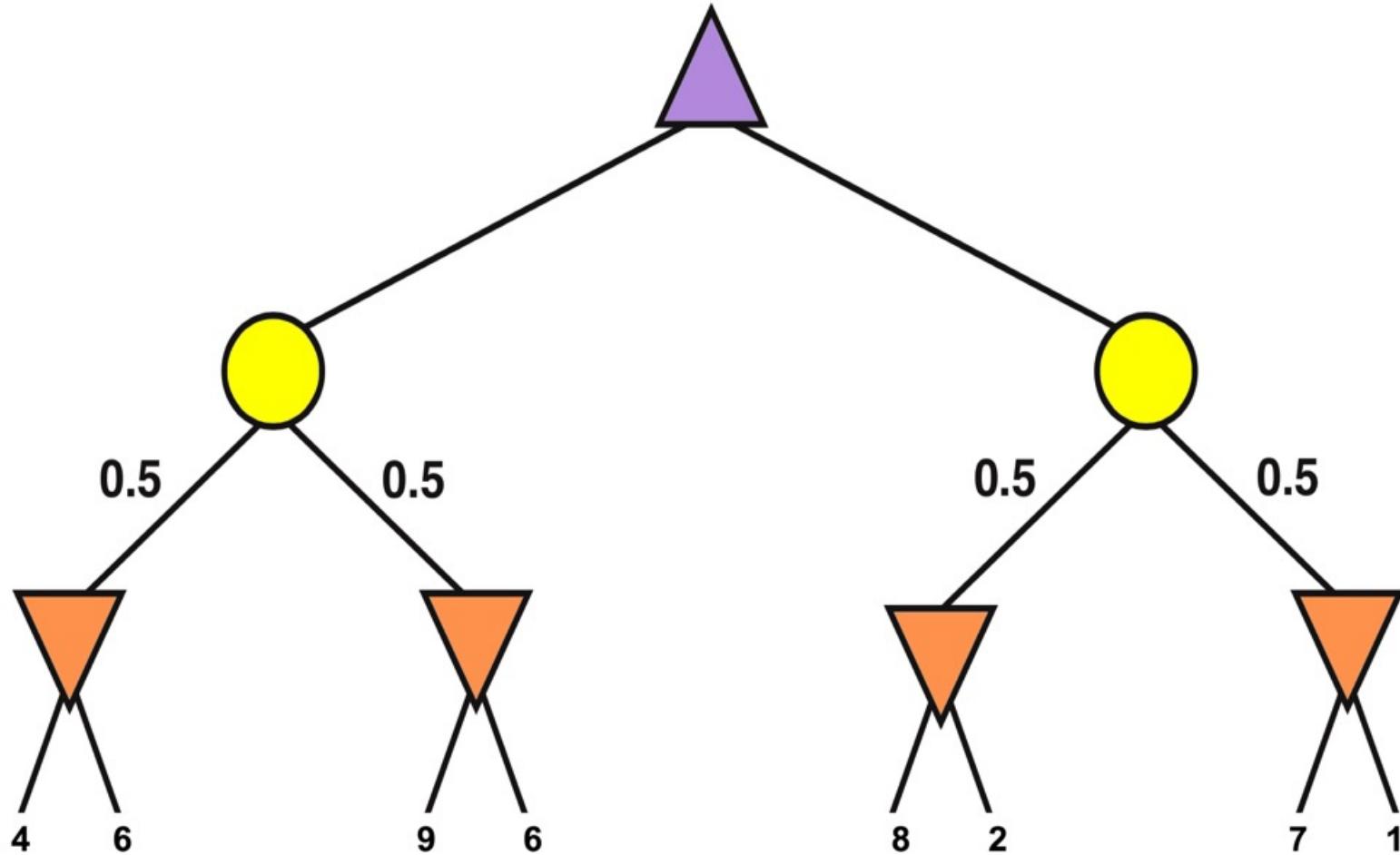
Stochastic games

Algorithm **Expectiminimax** generalized Minimax to handle chance nodes as follows:

- If state is a Max node then
return the highest Expectiminimax-Value of Successors(state)
- If state is a Min node then
return the lowest Expectiminimax-Value of Successors(state)
- If state is a chance node then
return average of Expectiminimax-Value of Successors(state)

Stochastic games

Example with coin-flipping:



Expectiminimax

For a state s :

$\text{Expectiminimax}(s) =$

$$\left\{ \begin{array}{ll} \text{Utility}(s) & \text{if Terminal-test}(s) \\ \max_{a \in \text{Actions}(s)} \text{Expectiminimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{Max} \\ \min_{a \in \text{Actions}(s)} \text{Expectiminimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{Min} \\ \sum_r P(r) \text{Expectiminimax}(\text{Result}(s,r)) & \text{if Player}(s) = \text{Chance} \end{array} \right.$$

Where r represents all chance events (e.g., dice roll), and $\text{Result}(s,r)$ is the same state as s with the result of the chance event is r .

Games: conclusion

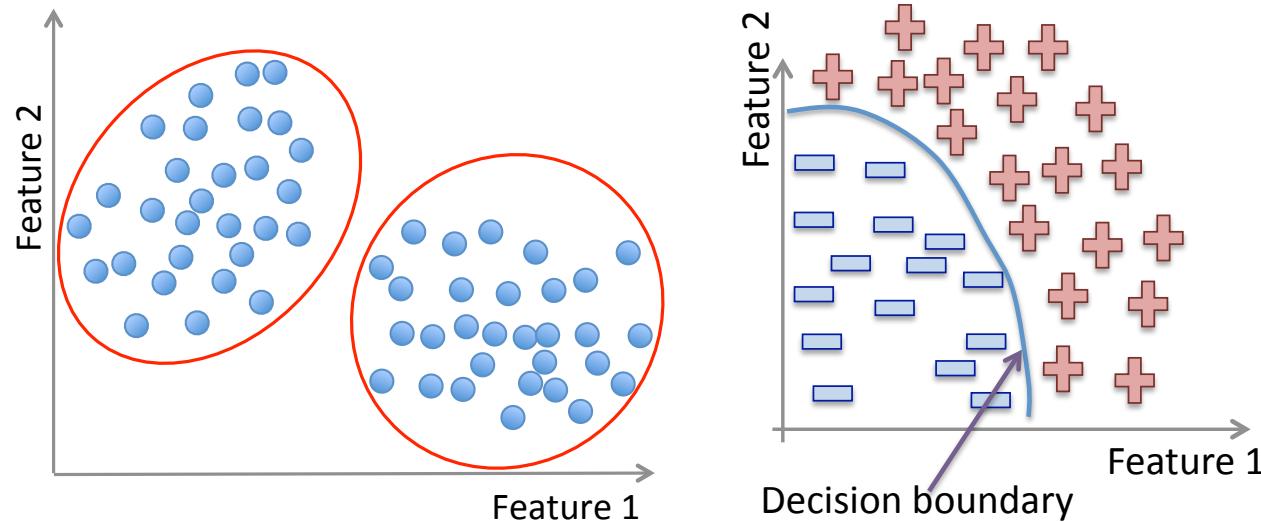
- Games are modeled in AI as a search problem and use heuristic to evaluate the game.
- Minimax algorithm chooses the best move given an optimal play from the opponent.
- Minimax goes all the way down the tree which is not practical given game time constraints.
- Alpha-Beta pruning can reduce the game tree search which allows to go deeper in the tree within the time constraints.
- Pruning, bookkeeping, evaluation heuristics, node re-ordering and IDS are effective in practice.

Games: conclusion

- Games is an exciting and fun topic for AI.
- Devising adversarial search agents is challenging because of the huge state space.
- We have just scratched the surface of this topic.
- Further topics to explore include partially observable games (card games such as bridge, poker, etc.).
- Except for robot football (a.k.a. soccer), there was no much interest from AI in physical games.
(see <http://www.robocup.org/>).
- Interested in chess? check out the evaluation functions in Claude Shannon's paper.
- You will implement a game in your homework assignment.

Machine Learning

Basic Concepts



Terminology

Machine Learning, Data Science, Data Mining, Data Analysis, Statistical Learning, Knowledge Discovery in Databases, Pattern Discovery.



Data everywhere!

1. **Google:** processes 24 peta bytes of data per day.
2. **Facebook:** 10 million photos uploaded every hour.
3. **Youtube:** 1 hour of video uploaded every second.
4. **Twitter:** 400 million tweets per day.
5. **Astronomy:** Satellite data is in hundreds of PB.
6. ...
7. **“By 2020 the digital universe will reach 44 zettabytes...”**

The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, April 2014.

That's 44 trillion gigabytes!

Data types

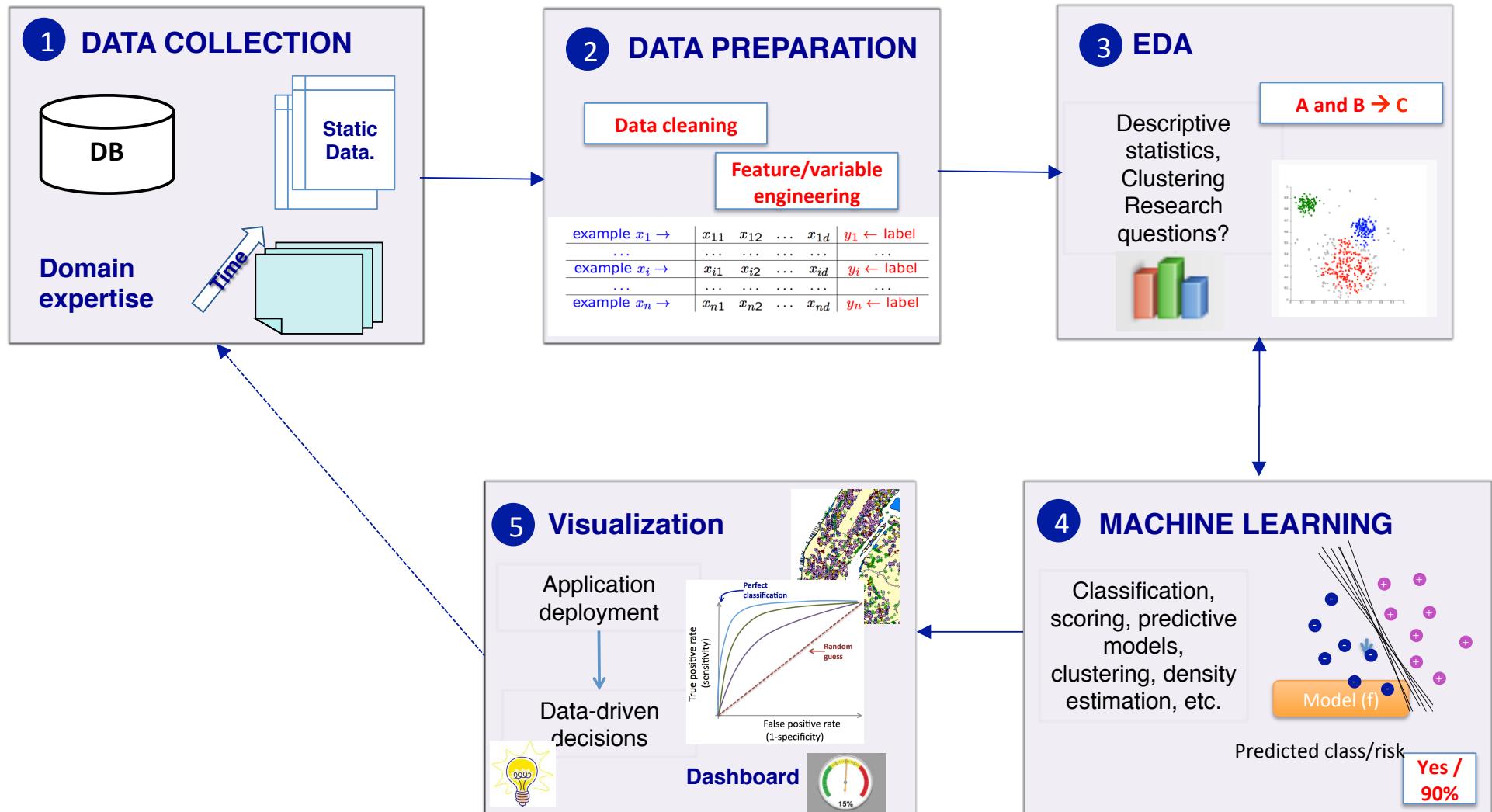
Data comes in different sizes and also flavors (types):

- ☒ **Texts**
- ☒ **Numbers**
- ☒ **Clickstreams**
- ☒ **Graphs**
- ☒ **Tables**
- ☒ **Images**
- ☒ **Transactions**
- ☒ **Videos**
- ☒ **Some or all of the above!**

Smile, we are 'DATAFIED' !

- Wherever we go, we are “datafied” .
- Smartphones are tracking our locations.
- We leave a data trail in our web browsing.
- Interaction in social networks.
- Privacy is an important issue in Data Science.

The Data Science process



Applications of ML

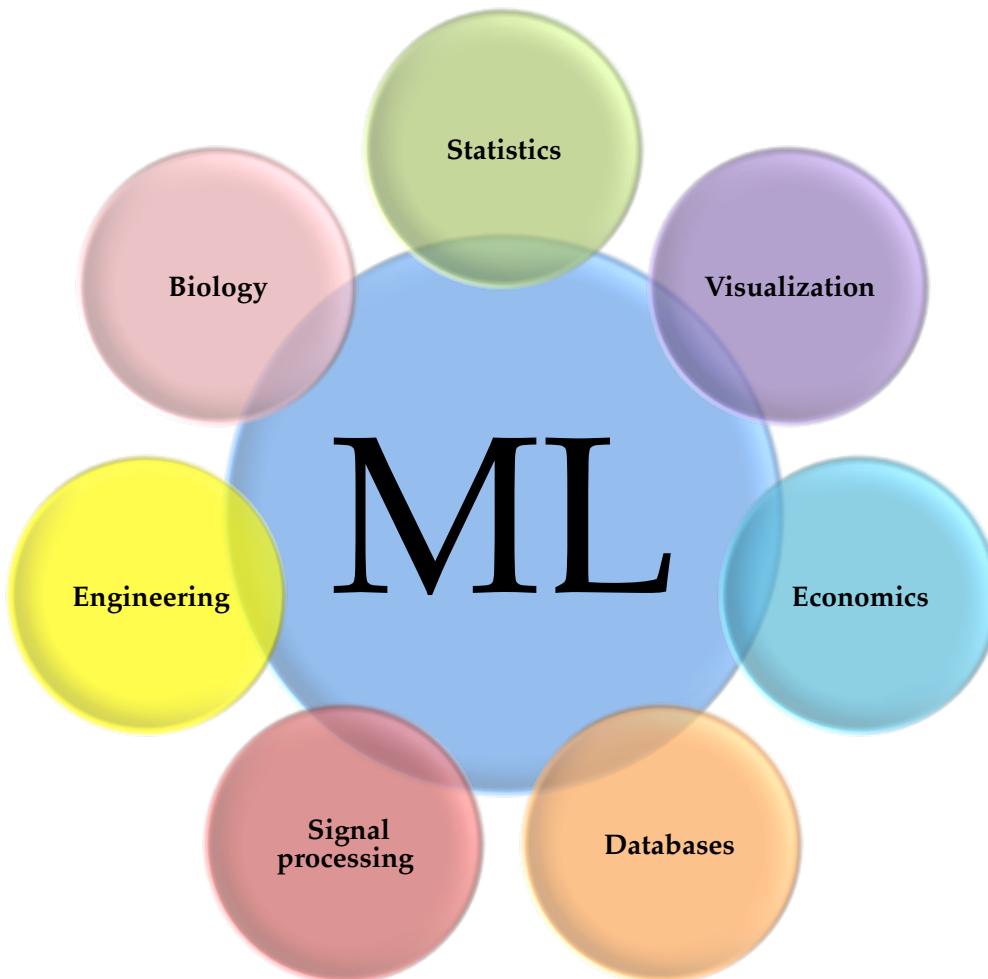
- We all use it on a daily basis. Examples:



Machine Learning

- Spam filtering
- Credit card fraud detection
- Digit recognition on checks, zip codes
- Detecting faces in images
- MRI image analysis
- Recommendation system
- Search engines
- Handwriting recognition
- Scene classification
- etc...

Interdisciplinary field



ML versus Statistics

Statistics:

- Hypothesis testing
- Experimental design
- Anova
- Linear regression
- Logistic regression
- GLM
- PCA

Machine Learning:

- Decision trees
- Rule induction
- Neural Networks
- SVMs
- Clustering method
- Association rules
- Feature selection
- Visualization
- Graphical models
- Genetic algorithm

<http://statweb.stanford.edu/~jhf/ftp/dm-stat.pdf>

Machine Learning definition

“How do we create computer programs that improve with experience?”

Tom Mitchell

http://videolectures.net/mlas06_mitchell_itm/

Machine Learning definition

“How do we create computer programs that improve with experience?”

Tom Mitchell

http://videolectures.net/mlas06_mitchell_itm/

“A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . ”

Tom Mitchell. Machine Learning 1997.

Supervised vs. Unsupervised

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and y_i is the label.

example $x_1 \rightarrow$	x_{11}	x_{12}	\dots	x_{1d}	$y_1 \leftarrow \text{label}$
\dots	\dots	\dots	\dots	\dots	\dots
example $x_i \rightarrow$	x_{i1}	x_{i2}	\dots	x_{id}	$y_i \leftarrow \text{label}$
\dots	\dots	\dots	\dots	\dots	\dots
example $x_n \rightarrow$	x_{n1}	x_{n2}	\dots	x_{nd}	$y_n \leftarrow \text{label}$

Supervised vs. Unsupervised

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and y_i is the label.

example $x_1 \rightarrow$	x_{11}	x_{12}	...	x_{1d}	$y_1 \leftarrow \text{label}$
...
example $x_i \rightarrow$	x_{i1}	x_{i2}	...	x_{id}	$y_i \leftarrow \text{label}$
...
example $x_n \rightarrow$	x_{n1}	x_{n2}	...	x_{nd}	$y_n \leftarrow \text{label}$

fruit	length	width	weight	label
fruit 1	165	38	172	Banana
fruit 2	218	39	230	Banana
fruit 3	76	80	145	Orange
fruit 4	145	35	150	Banana
fruit 5	90	88	160	Orange
...
fruit n

Supervised vs. Unsupervised

fruit	length	width	weight	label
fruit 1	165	38	172	Banana
fruit 2	218	39	230	Banana
fruit 3	76	80	145	Orange
fruit 4	145	35	150	Banana
fruit 5	90	88	160	Orange
...				
fruit n

Unsupervised learning:

Learning a model from **unlabeled** data.

Supervised learning:

Learning a model from **labeled** data.

Unsupervised Learning

Training data: “examples” x .

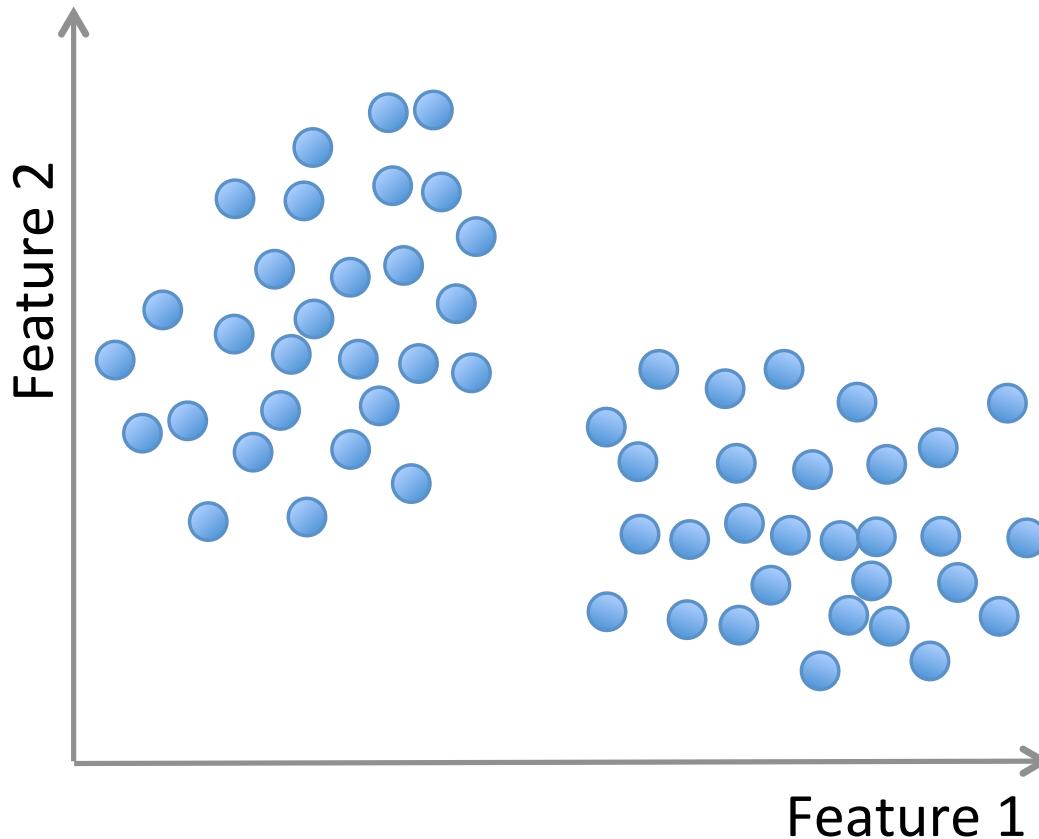
$$x_1, \dots, x_n, \quad x_i \in X \subset \mathbb{R}^n$$

- **Clustering/segmentation:**

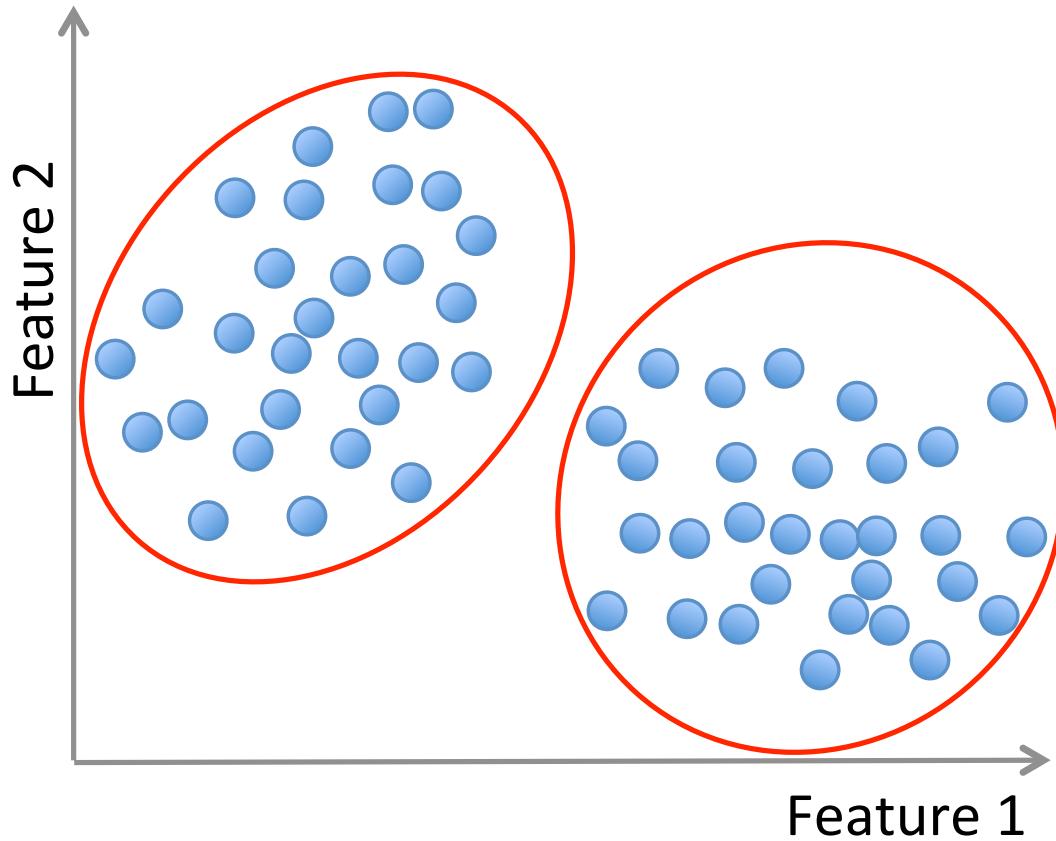
$$f : \mathbb{R}^d \longrightarrow \{C_1, \dots C_k\} \text{ (set of clusters).}$$

Example: Find clusters in the population, fruits, species.

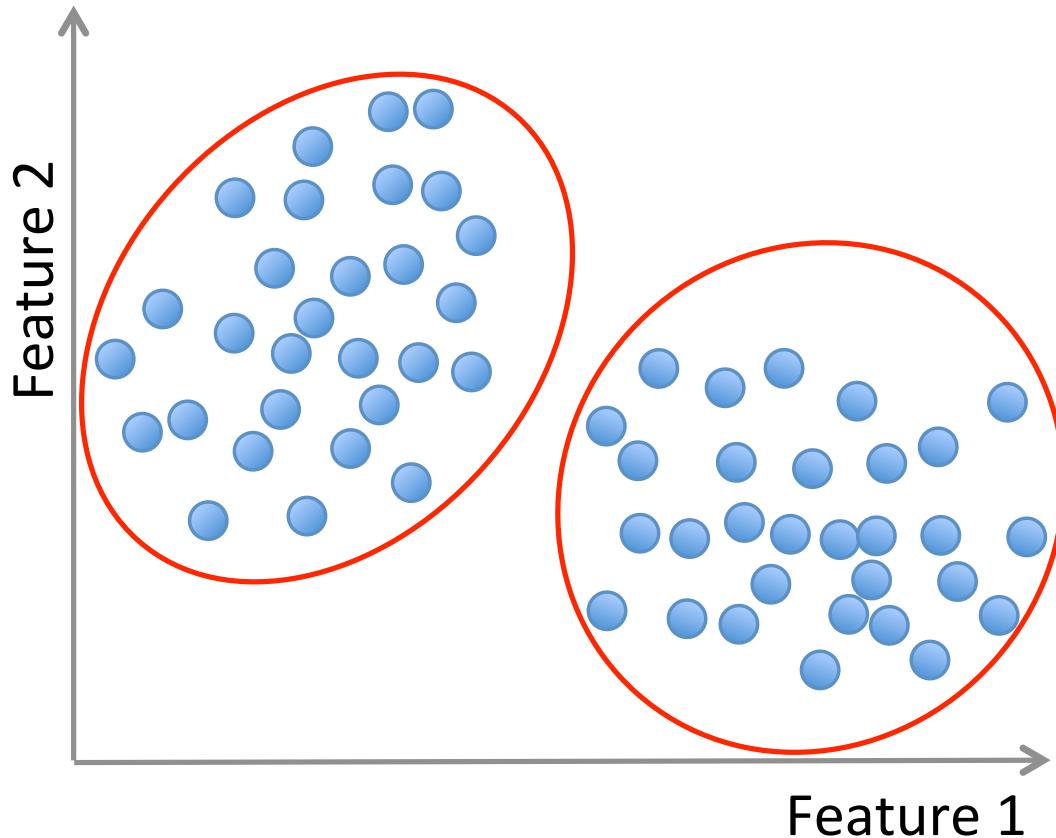
Unsupervised learning



Unsupervised learning



Unsupervised learning



Methods: K-means, gaussian mixtures, hierarchical clustering, spectral clustering, etc.

Supervised learning

Training data: “examples” x with “labels” y .

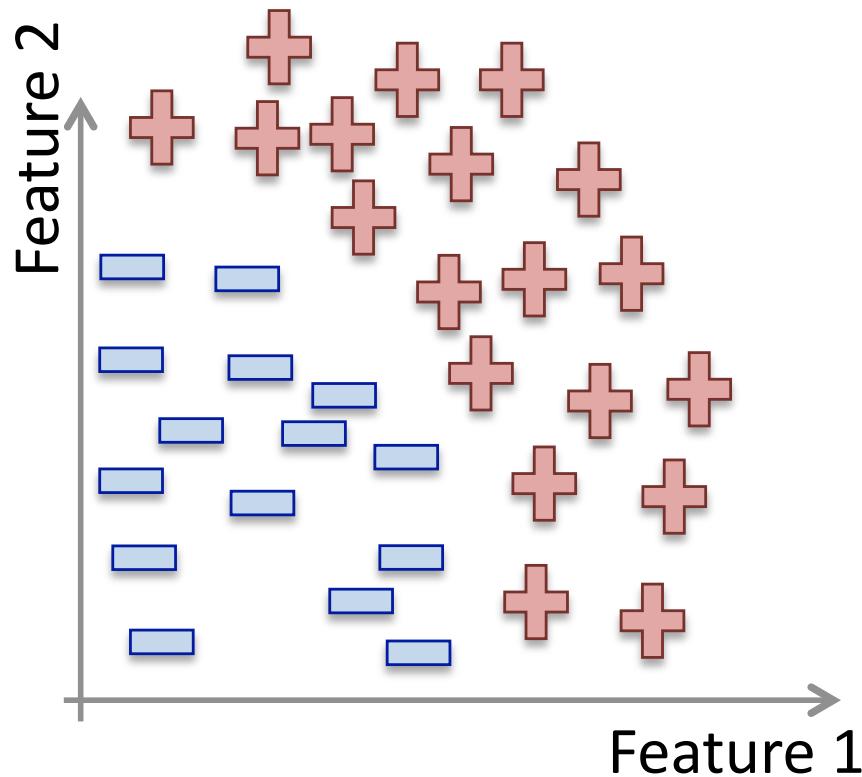
$$(x_1, y_1), \dots, (x_n, y_n) / x_i \in \mathbb{R}^d$$

- **Classification:** y is discrete. To simplify, $y \in \{-1, +1\}$

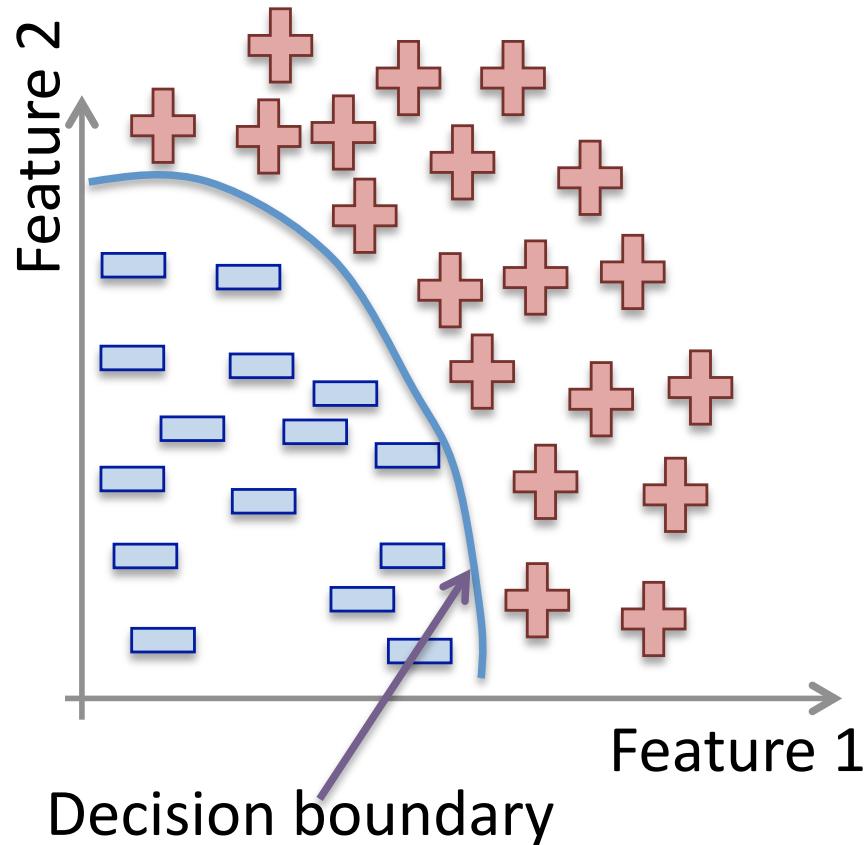
$$f : \mathbb{R}^d \longrightarrow \{-1, +1\} \quad f \text{ is called a } \mathbf{binary \ classifier}.$$

Example: Approve credit yes/no, spam/ham, banana/orange.

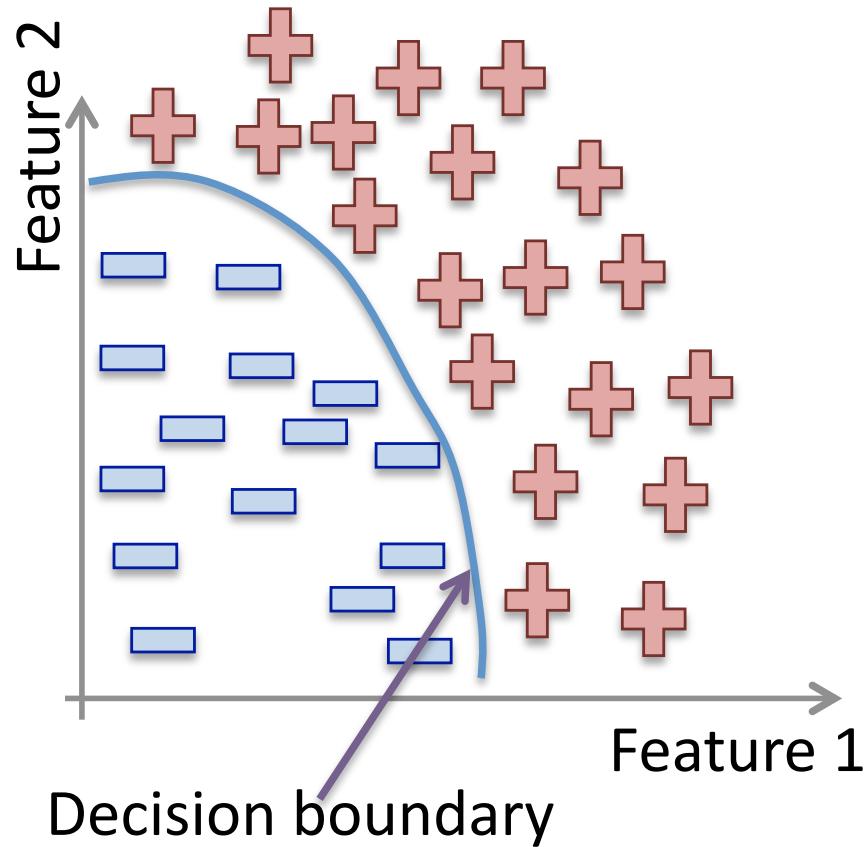
Supervised learning



Supervised learning



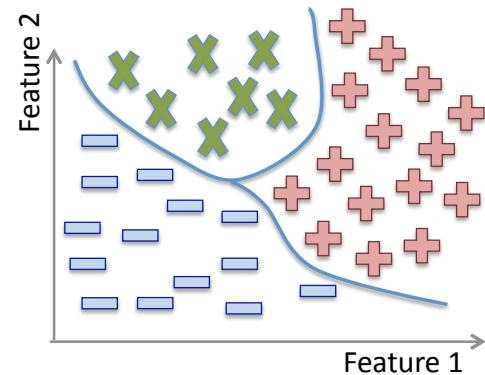
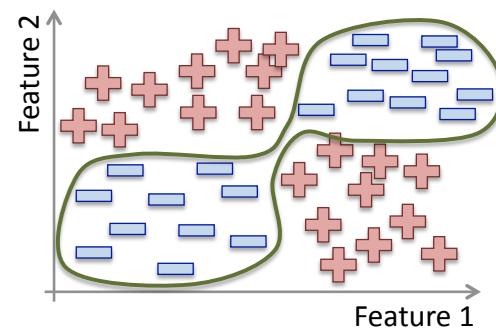
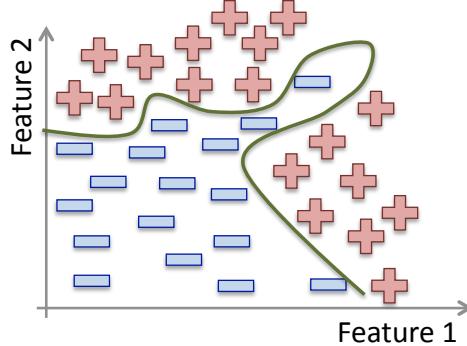
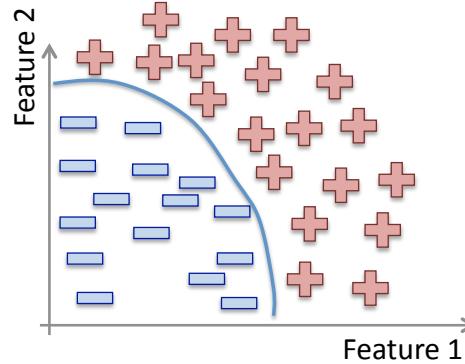
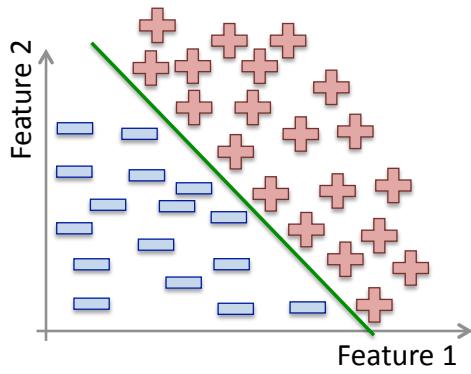
Supervised learning



Methods: Support Vector Machines, neural networks, decision trees, K-nearest neighbors, naive Bayes, etc.

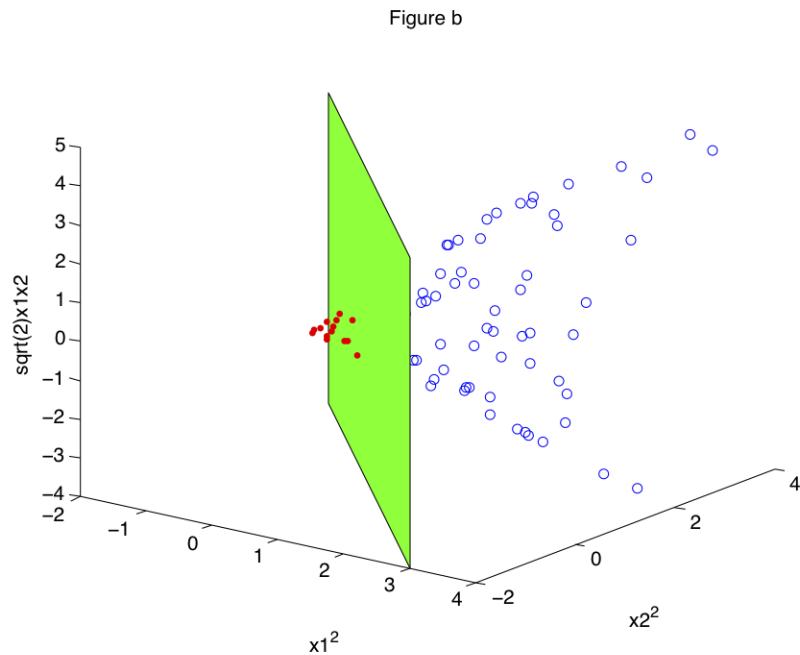
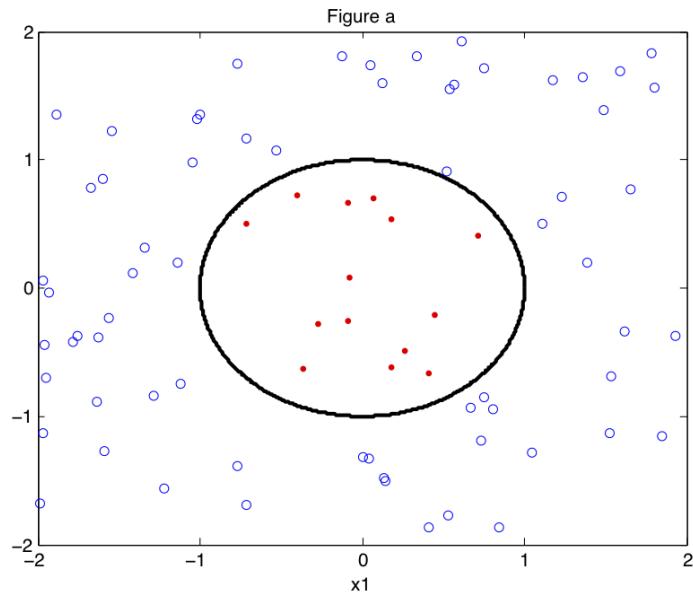
Supervised learning

Classification:



Supervised learning

Non linear classification



Supervised learning

Training data: “examples” x with “labels” y .

$$(x_1, y_1), \dots, (x_n, y_n) / x_i \in \mathbb{R}^d$$

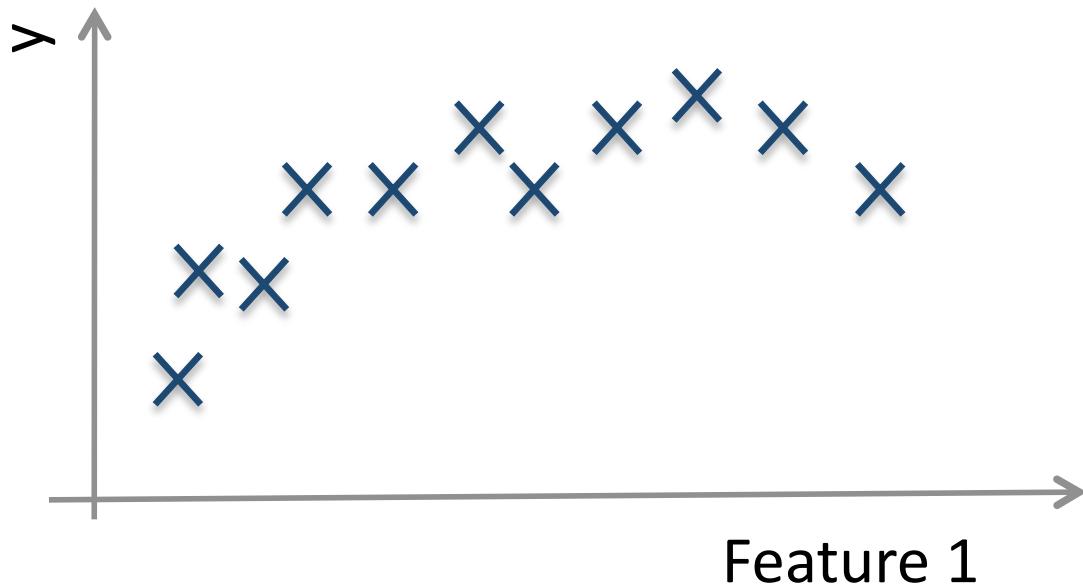
- **Regression:** y is a real value, $y \in \mathbb{R}$

$$f : \mathbb{R}^d \longrightarrow \mathbb{R} \quad f \text{ is called a } \textbf{regressor}.$$

Example: amount of credit, weight of fruit.

Supervised learning

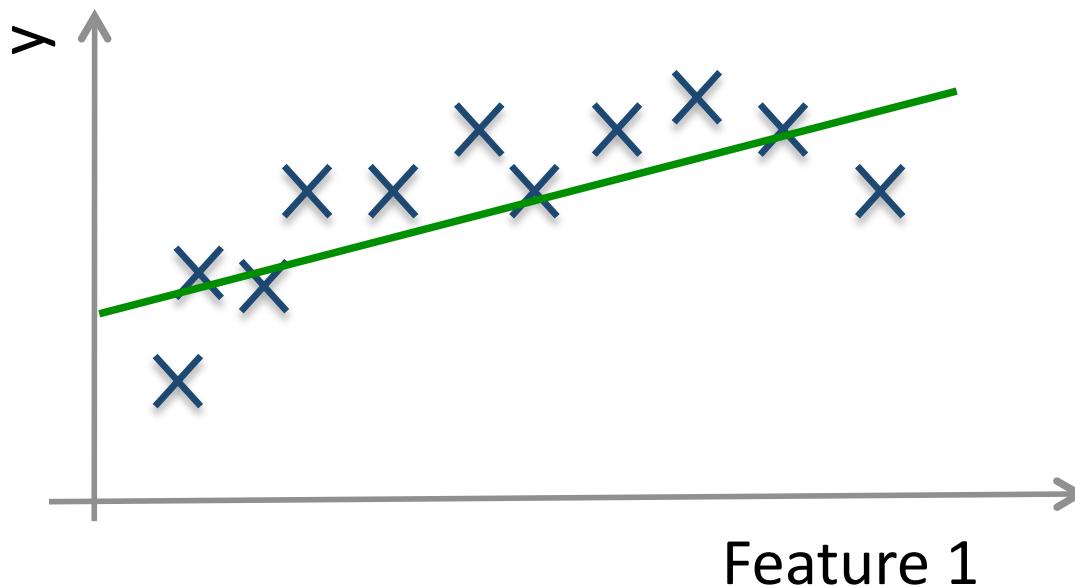
Regression:



Example: Income in function of age, weight of the fruit in function of its length.

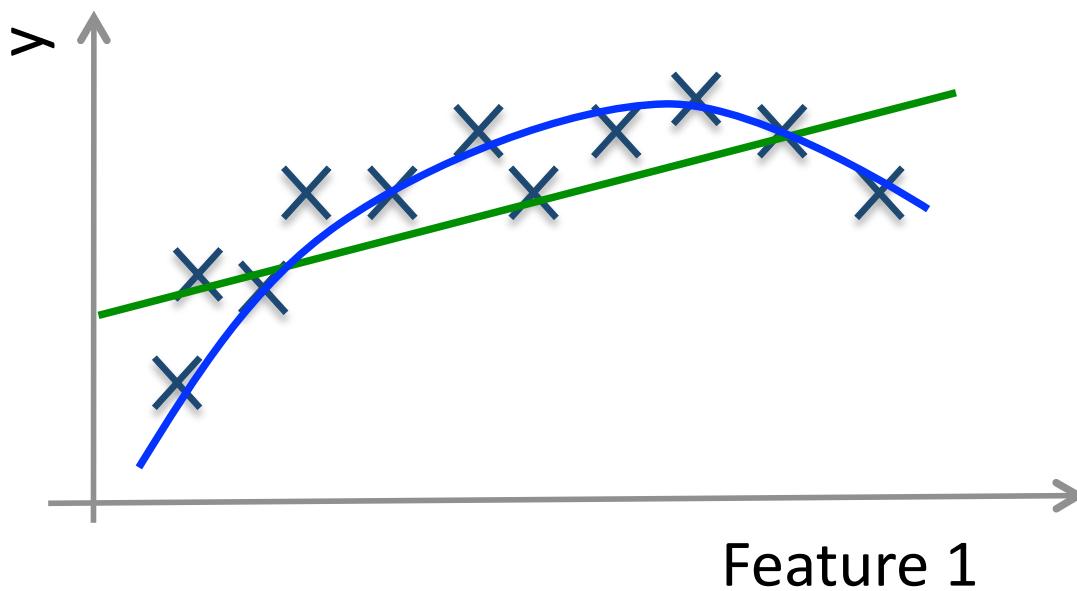
Supervised learning

Regression:



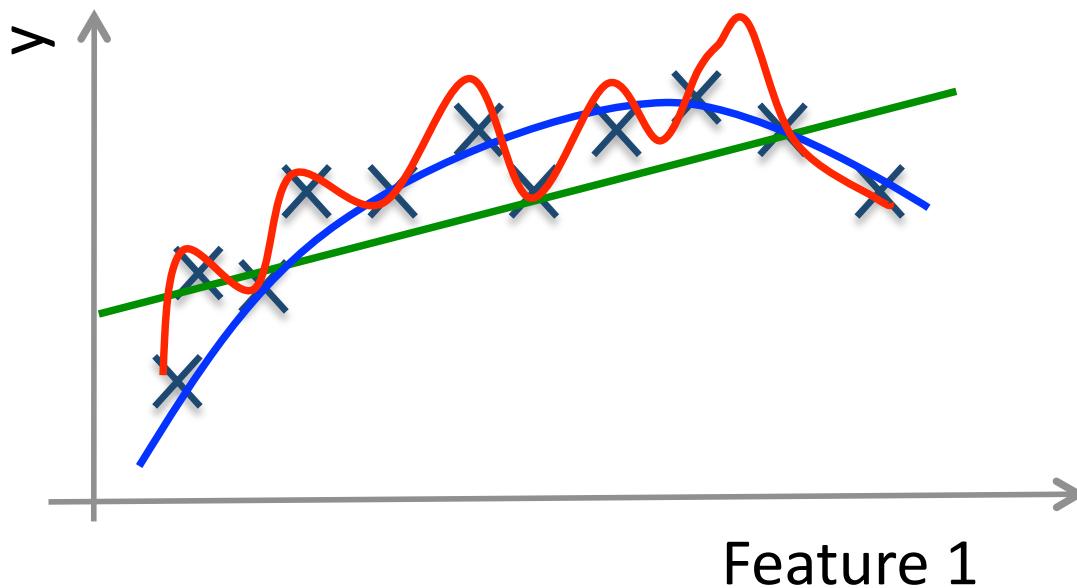
Supervised learning

Regression:

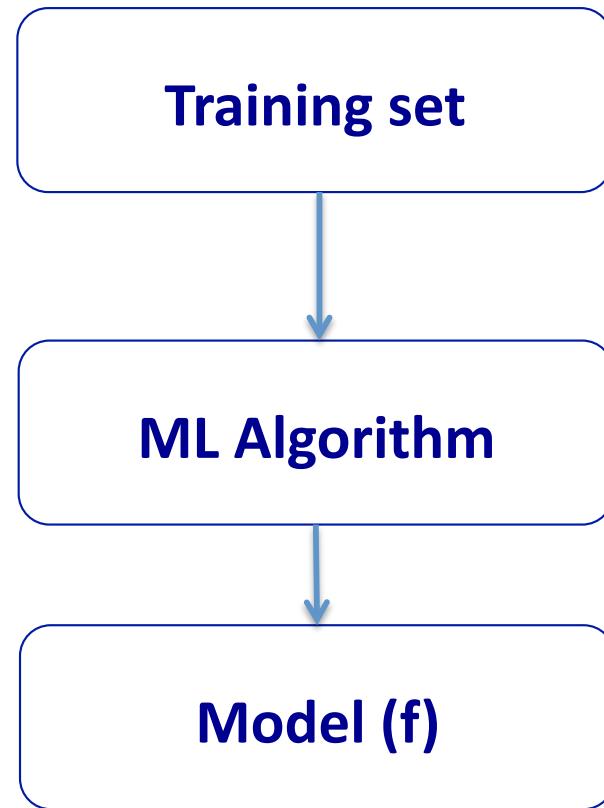


Supervised learning

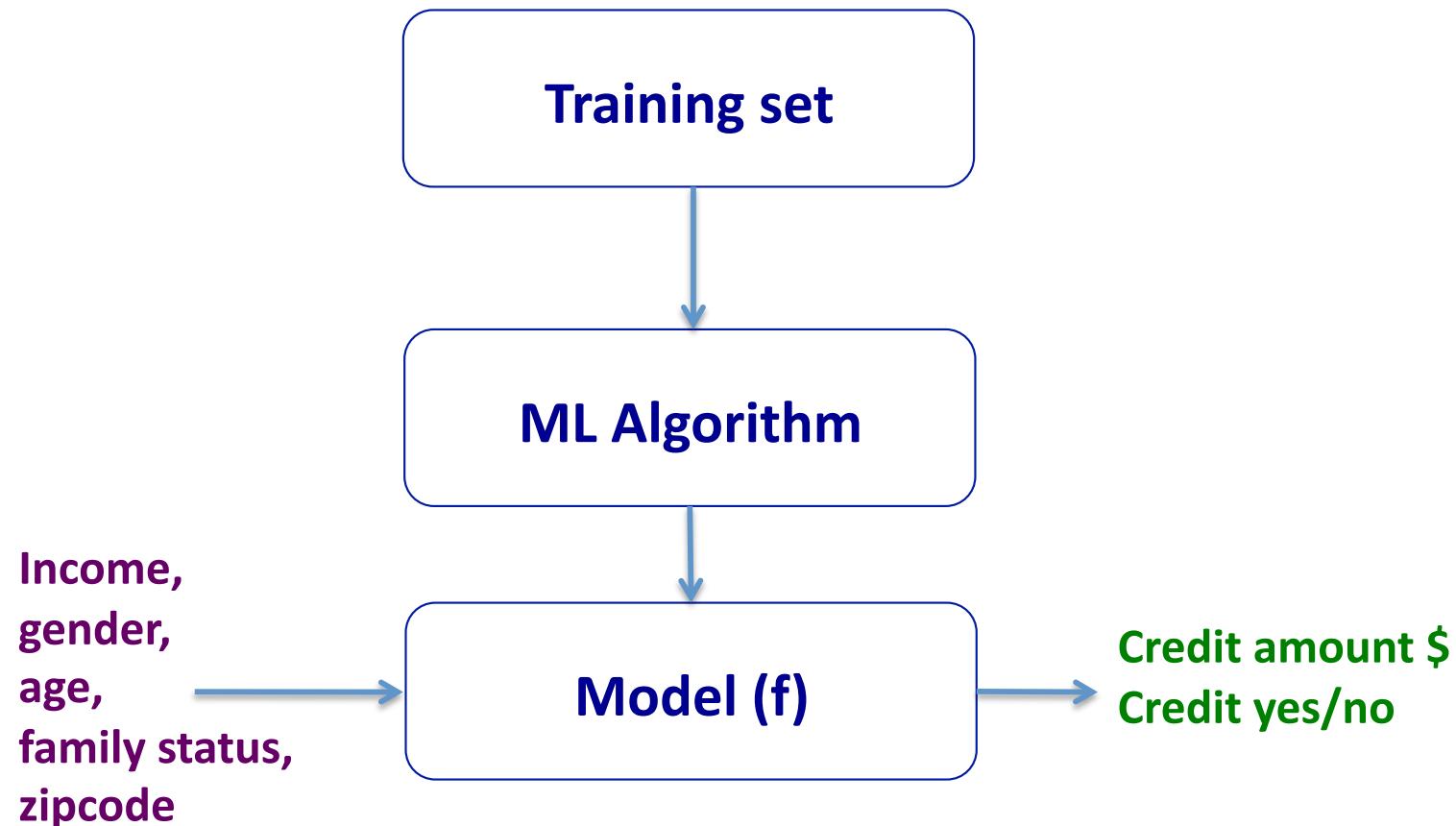
Regression:



Training and Testing



Training and Testing



K-nearest neighbors

- Not every ML method builds a model!
- Our first ML method: KNN.
- Main idea: Uses the **similarity** between examples.
- Assumption: Two similar examples should have same labels.
- Assumes all examples (instances) are points in the d dimensional space \mathbb{R}^d .

K-nearest neighbors

- KNN uses the standard **Euclidian distance** to define nearest neighbors.

Given two examples x_i and x_j :

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^d (x_{ik} - x_{jk})^2}$$

K-nearest neighbors

Training algorithm:

Add each training example (x, y) to the dataset \mathcal{D} .
 $x \in \mathbb{R}^d$, $y \in \{+1, -1\}$.

K-nearest neighbors

Training algorithm:

Add each training example (x, y) to the dataset \mathcal{D} .

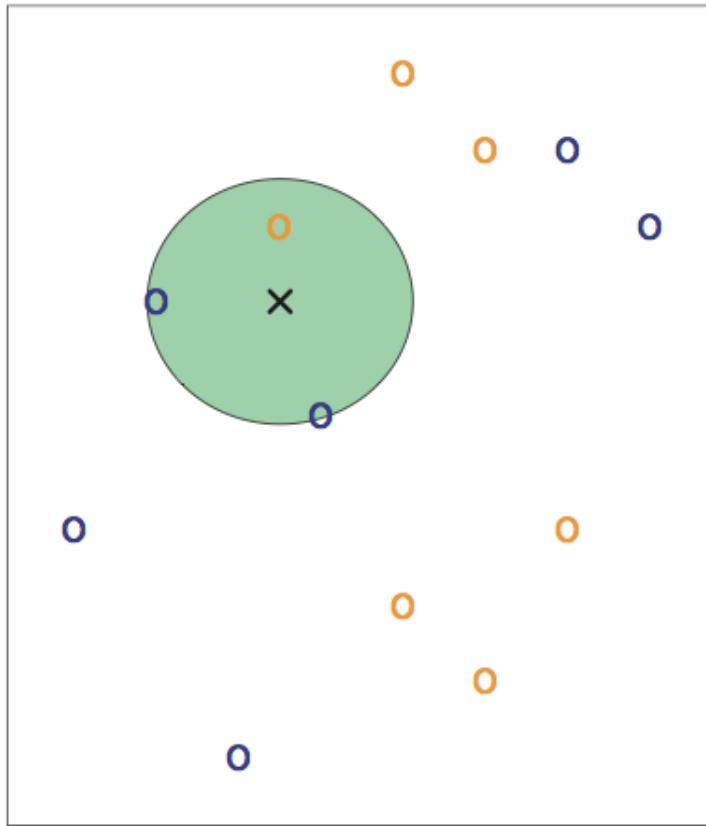
$x \in \mathbb{R}^d$, $y \in \{+1, -1\}$.

Classification algorithm:

Given an example x_q to be classified. Suppose $N_k(x_q)$ is the set of the K-nearest neighbors of x_q .

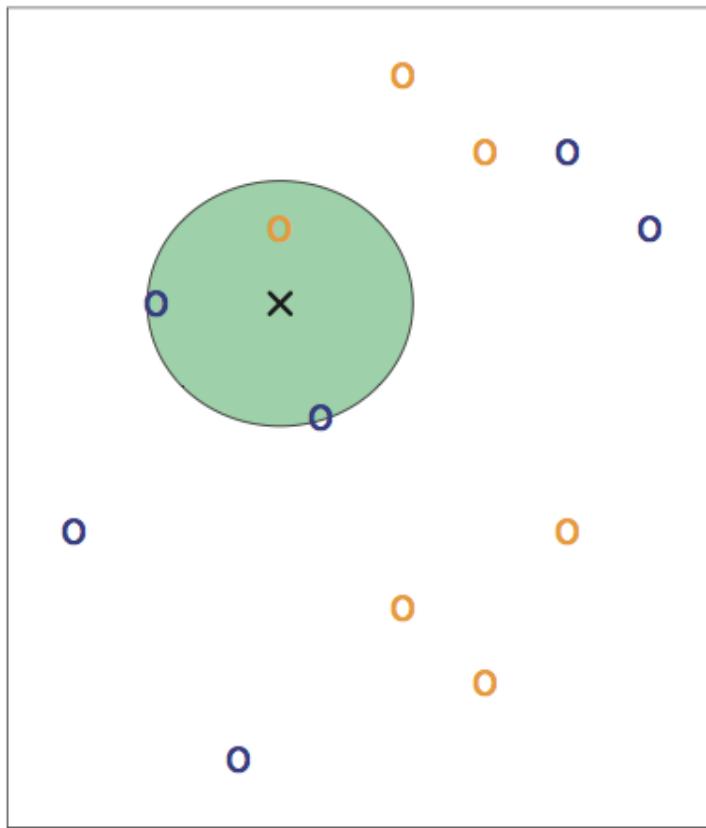
$$\hat{y}_q = \text{sign}\left(\sum_{x_i \in N_k(x_q)} y_i\right)$$

K-nearest neighbors



3-NN. Credit: [Introduction to Statistical Learning](#).

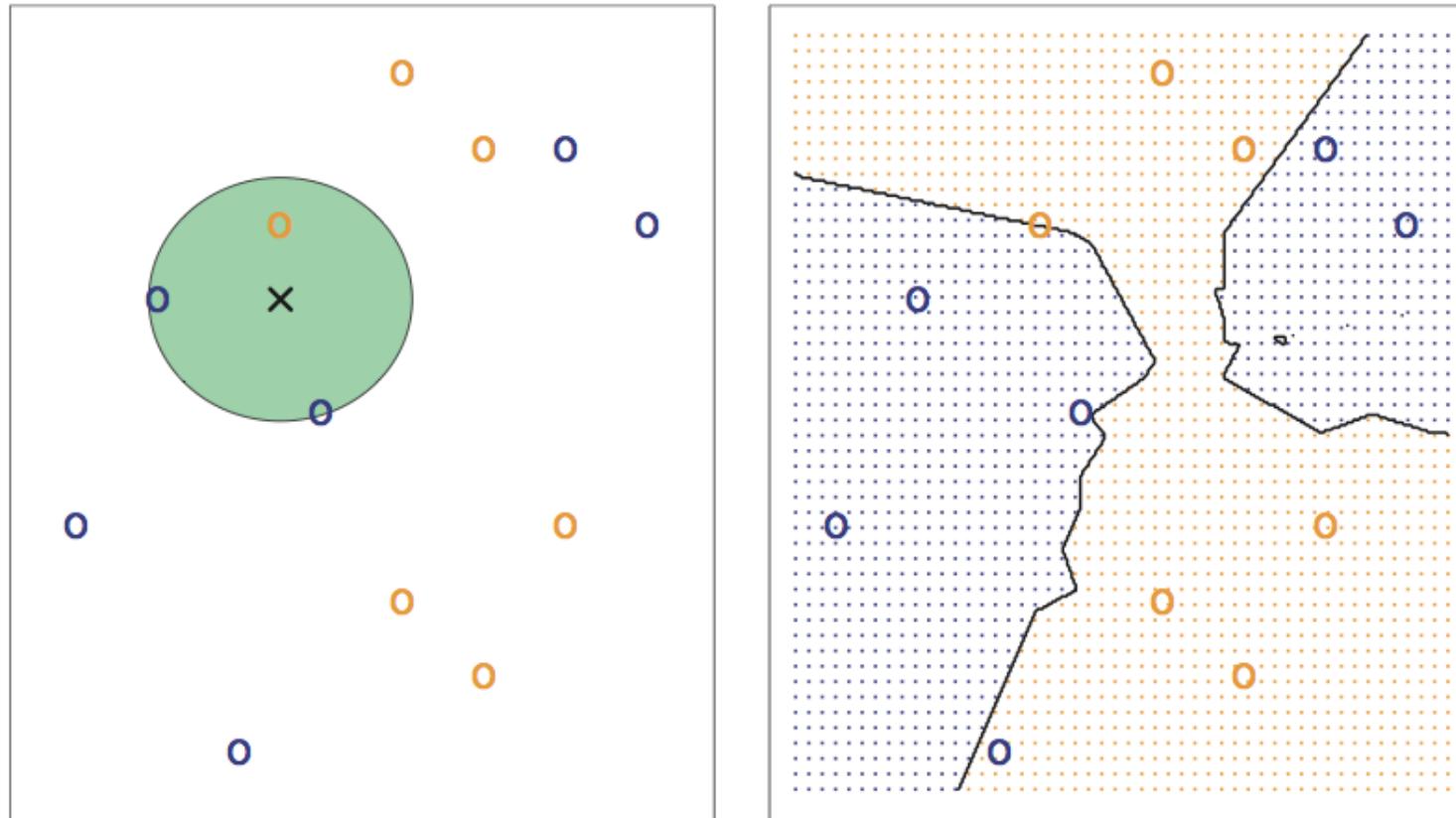
K-nearest neighbors



3-NN. Credit: [Introduction to Statistical Learning](#).

Question: Draw an approximate decision boundary for $K = 3$?

K-nearest neighbors



Credit: Introduction to Statistical Learning.

K-nearest neighbors

Question: What are the pros and cons of K-NN?

K-nearest neighbors

Question: What are the pros and cons of K-NN?

Pros:

- + Simple to implement.
- + Works well in practice.
- + Does not require to build a model, make assumptions, tune parameters.
- + Can be extended easily with new examples.

K-nearest neighbors

Question: What are the pros and cons of K-NN?

Pros:

- + Simple to implement.
- + Works well in practice.
- + Does not require to build a model, make assumptions, tune parameters.
- + Can be extended easily with new examples.

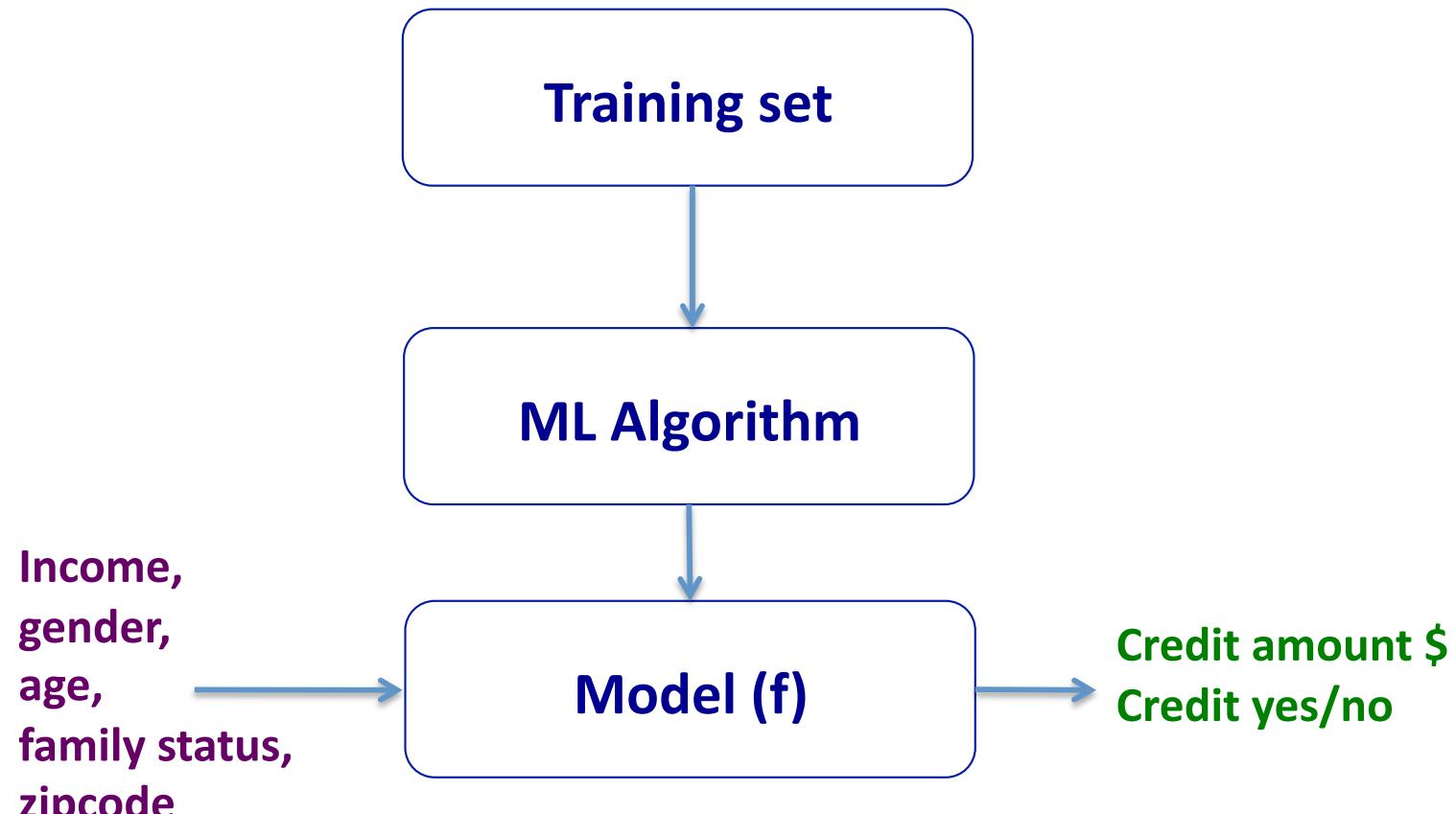
Cons:

- Requires large space to store the entire training dataset.
- Slow! Given n examples and d features. The method takes $O(n \times d)$ to run.
- Suffers from the *curse of dimensionality*.

Applications of K-NN

1. Information retrieval.
2. Handwritten character classification using nearest neighbor in large databases.
3. Recommender systems (user like you may like similar movies).
4. Breast cancer diagnosis.
5. Medical data mining (similar patient symptoms).
6. Pattern recognition in general.

Training and Testing



Question: How can we be confident about f ?

Training and Testing

- We calculate E^{train} the in-sample error (training error or empirical error/risk).

$$E^{train}(f) = \sum_{i=1}^n loss(y_i, f(x_i))$$

Training and Testing

- We calculate E^{train} the in-sample error (training error or empirical error/risk).

$$E^{train}(f) = \sum_{i=1}^n \text{loss}(y_i, f(x_i))$$

- Examples of loss functions:

- **Classification error:**

$$\text{loss}(y_i, f(x_i)) = \begin{cases} 1 & \text{if } \text{sign}(y_i) \neq \text{sign}(f(x_i)) \\ 0 & \text{otherwise} \end{cases}$$

Training and Testing

- We calculate E^{train} the in-sample error (training error or empirical error/risk).

$$E^{train}(f) = \sum_{i=1}^n \text{loss}(y_i, f(x_i))$$

- Examples of loss functions:

- **Classification error:**

$$\text{loss}(y_i, f(x_i)) = \begin{cases} 1 & \text{if } \text{sign}(y_i) \neq \text{sign}(f(x_i)) \\ 0 & \text{otherwise} \end{cases}$$

- **Least square loss:**

$$\text{loss}(y_i, f(x_i)) = (y_i - f(x_i))^2$$

Training and Testing

- We calculate E^{train} the in-sample error (training error or empirical error/risk).

$$E^{train}(f) = \sum_{i=1}^n \text{loss}(y_i, f(x_i))$$

- We aim to have $E^{train}(f)$ small, i.e., minimize $E^{train}(f)$

Training and Testing

- We calculate E^{train} the in-sample error (training error or empirical error/risk).

$$E^{train}(f) = \sum_{i=1}^n \text{loss}(y_i, f(x_i))$$

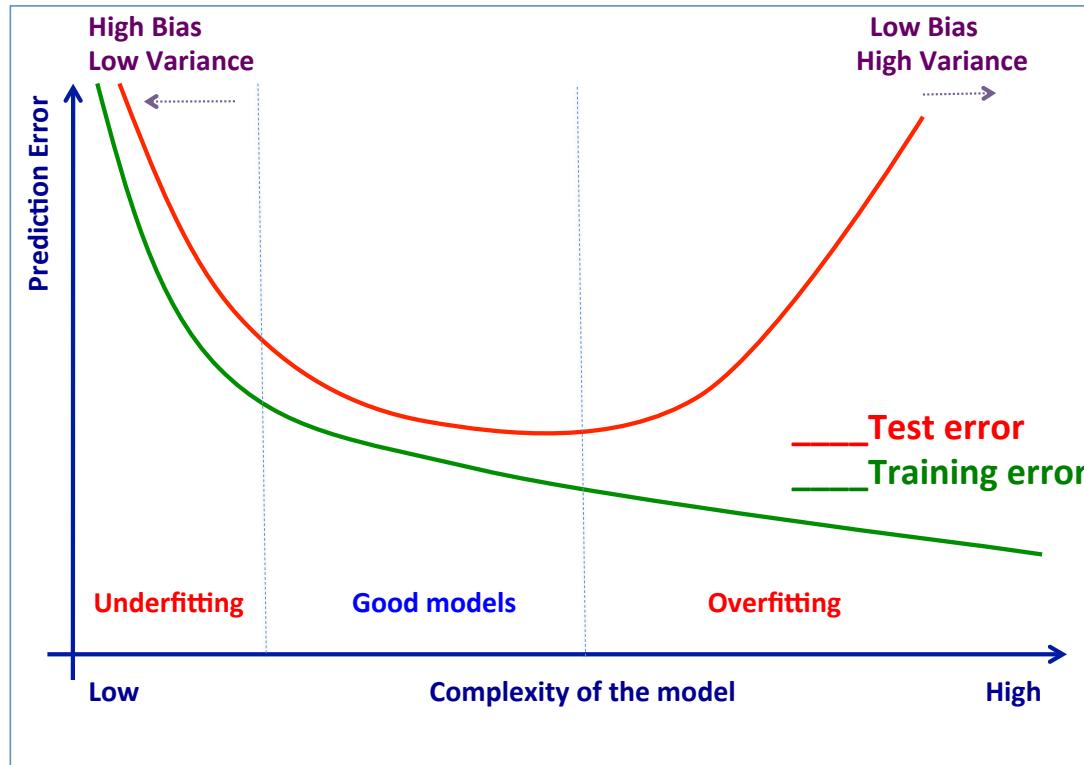
- We aim to have $E^{train}(f)$ small, i.e., minimize $E^{train}(f)$
- We hope that $E^{test}(f)$, the out-sample error (test/true error), will be small too.

Overfitting/underfitting



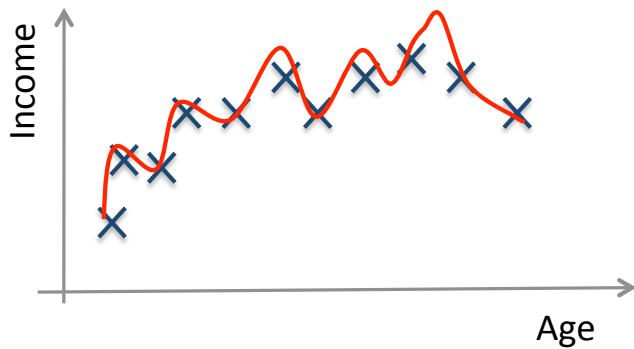
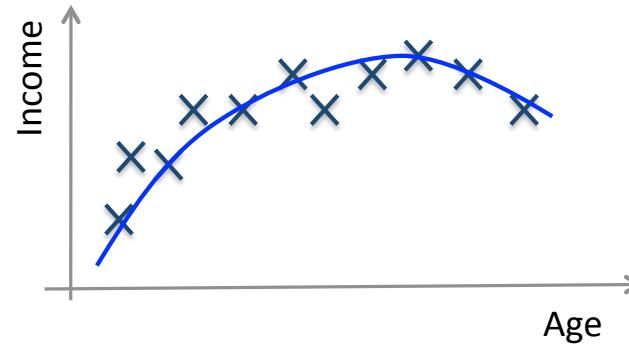
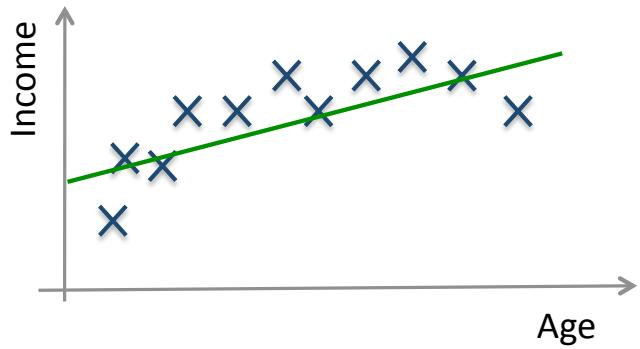
An intuitive example

Structural Risk Minimization

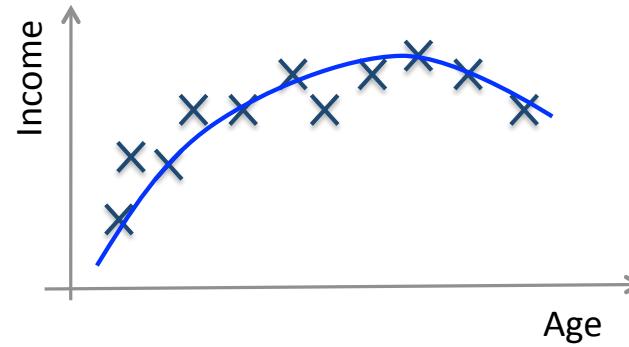
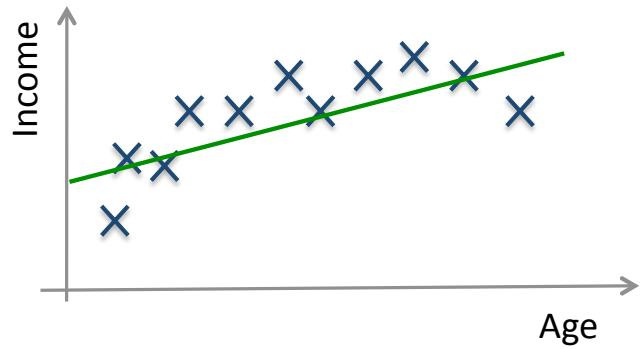


IMPORTANT

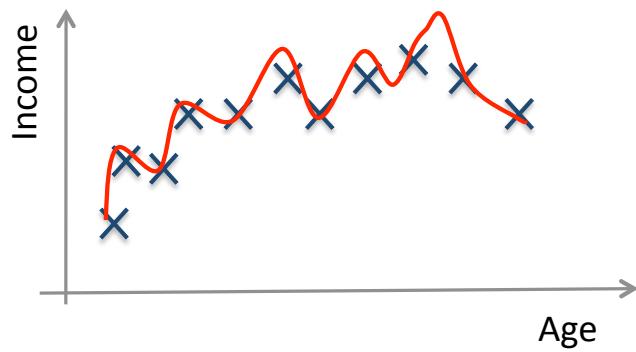
Training and Testing



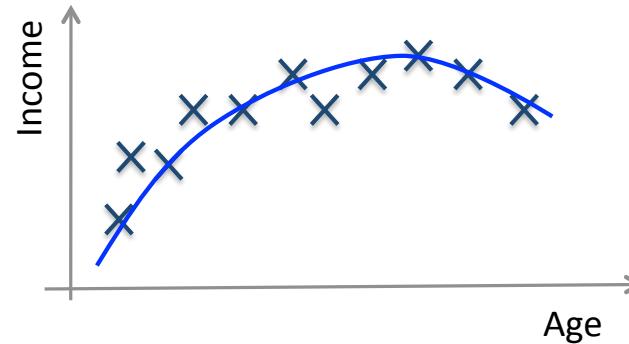
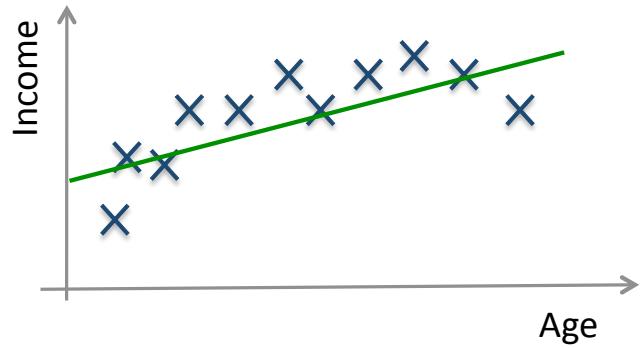
Training and Testing



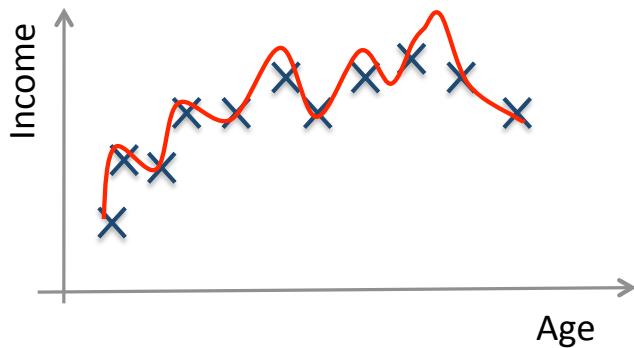
High bias (underfitting)



Training and Testing

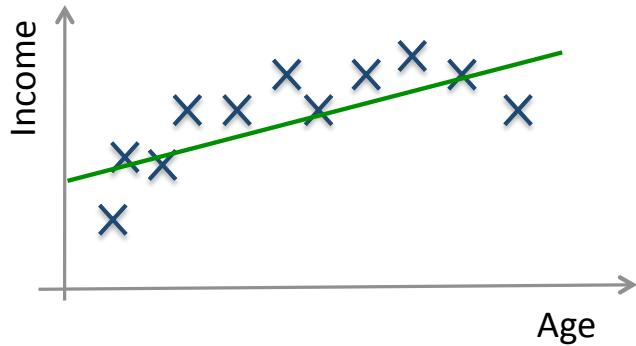


High bias (underfitting)

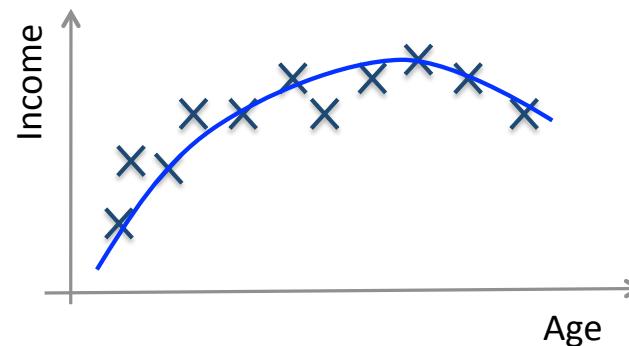


High variance (overfitting)

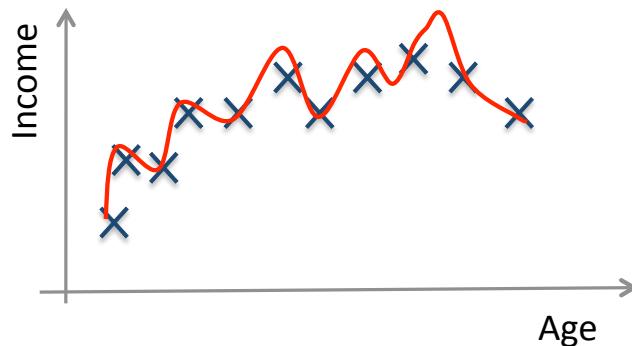
Training and Testing



High bias (underfitting)



Just right!



High variance (overfitting)

Avoid overfitting

In general, use simple models!

- **Reduce the number** of features manually or do feature selection.
- Do a **model selection** (ML course).
- Use **regularization** (keep the features but reduce their importance by setting small parameter values) (ML course).
- Do a **cross-validation** to estimate the test error.

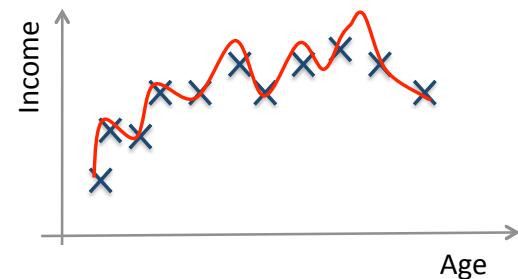
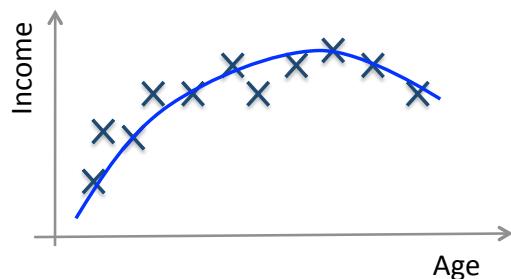
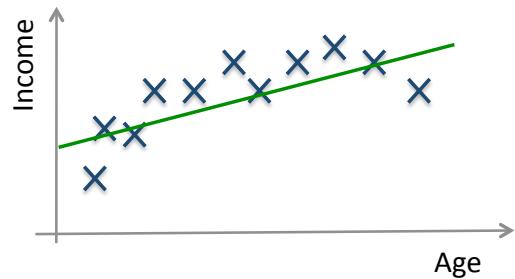
Regularization: Intuition

We want to minimize:

Classification term + $C \times$ Regularization term

$$\sum_{i=1}^n \text{loss}(y_i, f(x_i)) + C \times R(f)$$

Regularization: Intuition



$$f(x) = \lambda_0 + \lambda_1 x \dots \quad (1)$$

$$f(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 \dots \quad (2)$$

$$f(x) = \lambda_0 + \lambda_1 x + \lambda_2 x^2 + \lambda_3 x^3 + \lambda_4 x^4 \dots \quad (3)$$

Hint: Avoid high-degree polynomials.

Train, Validation and Test



Example: Split the data randomly into 60% for training, 20% for validation and 20% for testing.

Train, Validation and Test



1. Training set is a set of examples used for learning a model (e.g., a classification model).

Train, Validation and Test



1. Training set is a set of examples used for learning a model (e.g., a classification model).
2. Validation set is a set of examples that cannot be used for learning the model but can help tune model parameters (e.g., selecting K in K-NN). Validation helps control overfitting.

Train, Validation and Test



1. Training set is a set of examples used for learning a model (e.g., a classification model).
2. Validation set is a set of examples that cannot be used for learning the model but can help tune model parameters (e.g., selecting K in K-NN). Validation helps control overfitting.
3. Test set is used to assess the performance of the final model and provide an estimation of the test error.

Train, Validation and Test



1. Training set is a set of examples used for learning a model (e.g., a classification model).
2. Validation set is a set of examples that cannot be used for learning the model but can help tune model parameters (e.g., selecting K in K-NN). Validation helps control overfitting.
3. Test set is used to assess the performance of the final model and provide an estimation of the test error.

Note: Never use the test set in any way to further tune the parameters or revise the model.

K-fold Cross Validation

A method for estimating test error using training data.

Algorithm:

Given a learning algorithm \mathcal{A} and a dataset \mathcal{D}

Step 1: Randomly partition \mathcal{D} into k equal-size subsets $\mathcal{D}_1, \dots, \mathcal{D}_k$

Step 2:

For $j = 1$ to k

 Train \mathcal{A} on all \mathcal{D}_i , $i \in 1, \dots, k$ and $i \neq j$, and get f_j .

 Apply f_j to \mathcal{D}_j and compute $E^{\mathcal{D}_j}$

Step 3: Average error over all folds.

$$\sum_{j=1}^k (E^{\mathcal{D}_j})$$

Confusion matrix

		Actual Label	
		Positive	Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Evaluation metrics

		Actual Label	
		Positive	Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	The percentage of predictions that are correct
Precision	$TP / (TP + FP)$	The percentage of positive predictions that are correct
Sensitivity (Recall)	$TP / (TP + FN)$	The percentage of positive cases that were predicted as positive
Specificity	$TN / (TN + FP)$	The percentage of negative cases that were predicted as negative

Terminology review

Review the concepts and terminology:

Instance, example, feature, label, supervised learning, unsupervised learning, classification, regression, clustering, prediction, training set, validation set, test set, K-fold cross validation, classification error, loss function, overfitting, underfitting, regularization.

Machine Learning Books

1. Tom Mitchell, Machine Learning.
2. Abu-Mostafa, Yaser S. and Magdon-Ismail, Malik and Lin, Hsuan-Tien, Learning From Data, AMLBook.
3. The elements of statistical learning. Data mining, inference, and prediction T. Hastie, R. Tibshirani, J. Friedman.
4. Christopher Bishop. Pattern Recognition and Machine Learning.
5. Richard O. Duda, Peter E. Hart, David G. Stork. Pattern Classification. Wiley.

Machine Learning Resources

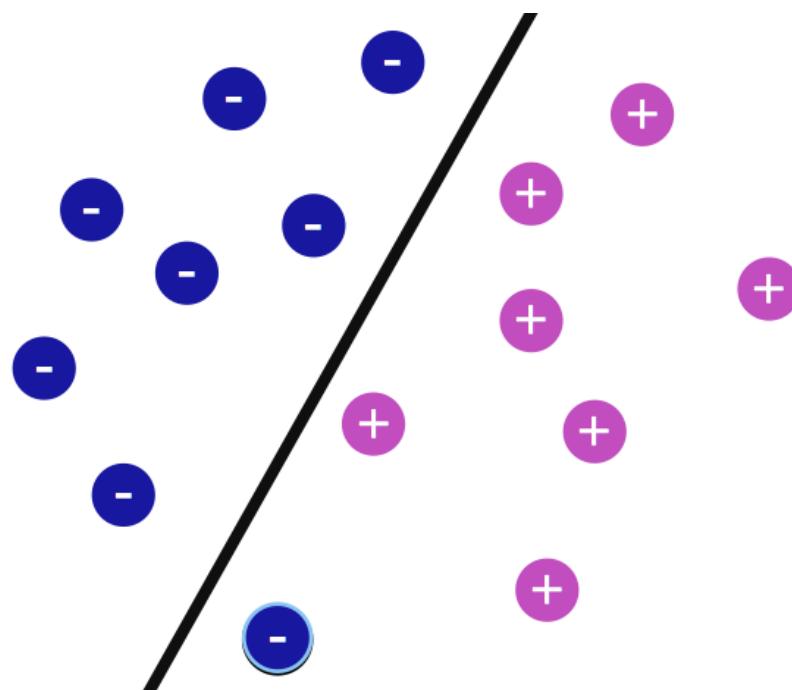
- Major journals/conferences: ICML, NIPS, UAI, ECML/PKDD, JMLR, MLJ, etc.
- Machine learning video lectures:
http://videolectures.net/Top/Computer_Science/Machine_Learning/
- Machine Learning (Theory):
<http://hunch.net/>
- LinkedIn ML groups: “Big Data” Scientist, etc.
- Women in Machine Learning:
<https://groups.google.com/forum/#!forum/women-in-machine-learning>
- KDD nuggets <http://www.kdnuggets.com/>

Credit

- The elements of statistical learning. Data mining, inference, and prediction. 10th Edition 2009. T. Hastie, R. Tibshirani, J. Friedman.
- Machine Learning 1997. Tom Mitchell.

Machine Learning

Linear Models



Outline

II - Linear Models

1. Linear Regression

- (a) Linear regression: History
- (b) Linear regression with Least Squares
- (c) Matrix representation and Normal Equation Method
- (d) Iterative Method: Gradient descent
- (e) Pros and Cons of both methods

2. Linear Classification: Perceptron

- (a) Definition and history
- (b) Example
- (c) Algorithm

Supervised Learning

Training data: “examples” x with “labels” y .

$$(x_1, y_1), \dots, (x_n, y_n) / x_i \in \mathbb{R}^d$$

- **Regression:** y is a real value, $y \in \mathbb{R}$

$$f : \mathbb{R}^d \longrightarrow \mathbb{R} \quad f \text{ is called a } \textcolor{red}{\text{regressor}}.$$

- **Classification:** y is discrete. To simplify, $y \in \{-1, +1\}$

$$f : \mathbb{R}^d \longrightarrow \{-1, +1\} \quad f \text{ is called a } \textcolor{blue}{\text{binary classifier}}.$$

Linear Regression: History

- A very popular technique.
- Rooted in Statistics.
- Method of Least Squares used as early as 1795 by Gauss.
- Re-invented in 1805 by Legendre.
- Frequently applied in **astronomy** to study the large scale of the universe.
- Still a very useful tool today.



Carl Friedrich Gauss

Linear Regression

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$

Linear Regression

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$

example $x_1 \rightarrow$	x_{11}	x_{12}	...	x_{1d}	$y_1 \leftarrow \text{label}$
...
example $x_i \rightarrow$	x_{i1}	x_{i2}	...	x_{id}	$y_i \leftarrow \text{label}$
...
example $x_n \rightarrow$	x_{n1}	x_{n2}	...	x_{nd}	$y_n \leftarrow \text{label}$

Linear Regression

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$

example $x_1 \rightarrow$	x_{11}	x_{12}	...	x_{1d}	$y_1 \leftarrow \text{label}$
...
example $x_i \rightarrow$	x_{i1}	x_{i2}	...	x_{id}	$y_i \leftarrow \text{label}$
...
example $x_n \rightarrow$	x_{n1}	x_{n2}	...	x_{nd}	$y_n \leftarrow \text{label}$

Task: Learn a regression function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{R}$$

$$f(x) = y$$

Linear Regression

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$

example $x_1 \rightarrow$	x_{11}	x_{12}	...	x_{1d}	$y_1 \leftarrow \text{label}$
...
example $x_i \rightarrow$	x_{i1}	x_{i2}	...	x_{id}	$y_i \leftarrow \text{label}$
...
example $x_n \rightarrow$	x_{n1}	x_{n2}	...	x_{nd}	$y_n \leftarrow \text{label}$

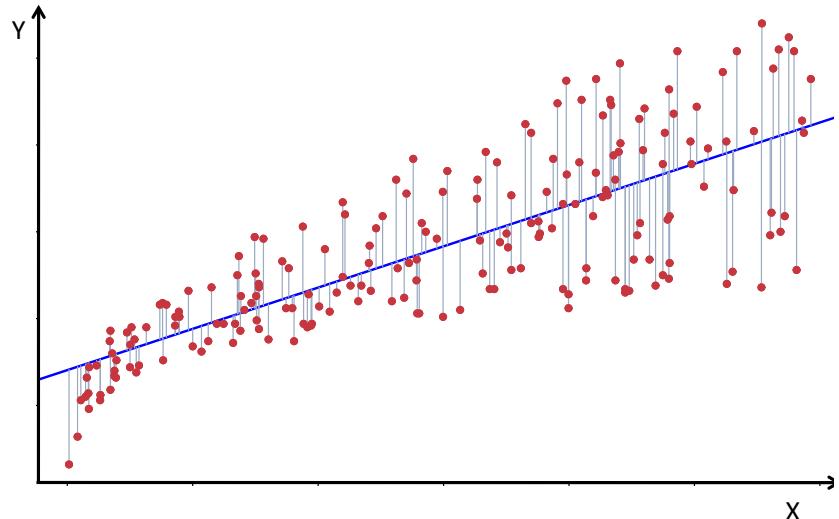
Task: Learn a regression function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{R}$$

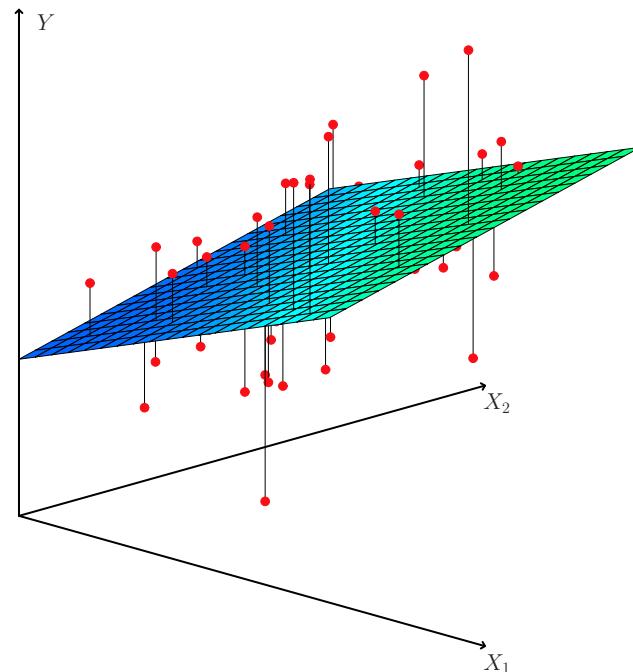
$$f(x) = y$$

Linear Regression: A regression model is said to be linear if it is represented by a linear function.

Linear Regression



$d = 1$, line in \mathbb{R}^2



$d = 2$, hyperplane is \mathbb{R}^3

Credit: Introduction to Statistical Learning.

Linear Regression

Linear Regression Model:

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j \quad \text{with } \beta_j \in \mathbb{R}, \quad j \in \{1, \dots, d\}$$

β 's are called parameters or coefficients or weights.

Linear Regression

Linear Regression Model:

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j \quad \text{with } \beta_j \in \mathbb{R}, \quad j \in \{1, \dots, d\}$$

β 's are called parameters or coefficients or weights.

Learning the linear model \rightarrow learning the β 's

Linear Regression

Linear Regression Model:

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j \quad \text{with } \beta_j \in \mathbb{R}, \quad j \in \{1, \dots, d\}$$

β 's are called parameters or coefficients or weights.

Learning the linear model \rightarrow learning the β 's

Estimation with Least squares:

Use least square loss: $\text{loss}(y_i, f(x_i)) = (y_i - f(x_i))^2$

Linear Regression

Linear Regression Model:

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j \quad \text{with } \beta_j \in \mathbb{R}, \quad j \in \{1, \dots, d\}$$

β 's are called parameters or coefficients or weights.

Learning the linear model \rightarrow learning the β 's

Estimation with Least squares:

Use least square loss: $\text{loss}(y_i, f(x_i)) = (y_i - f(x_i))^2$

We want to minimize the loss over all examples, that is minimize the *risk or cost function* R :

$$R = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2$$

Linear Regression

A simple case with one feature ($d = 1$):

$$f(x) = \beta_0 + \beta_1 x$$

Linear Regression

A simple case with one feature ($d = 1$):

$$f(x) = \beta_0 + \beta_1 x$$

We want to minimize:

$$R = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2$$

Linear Regression

A simple case with one feature ($d = 1$):

$$f(x) = \beta_0 + \beta_1 x$$

We want to minimize:

$$R = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2$$

$$R(\beta) = \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Linear Regression

A simple case with one feature ($d = 1$):

$$f(x) = \beta_0 + \beta_1 x$$

We want to minimize:

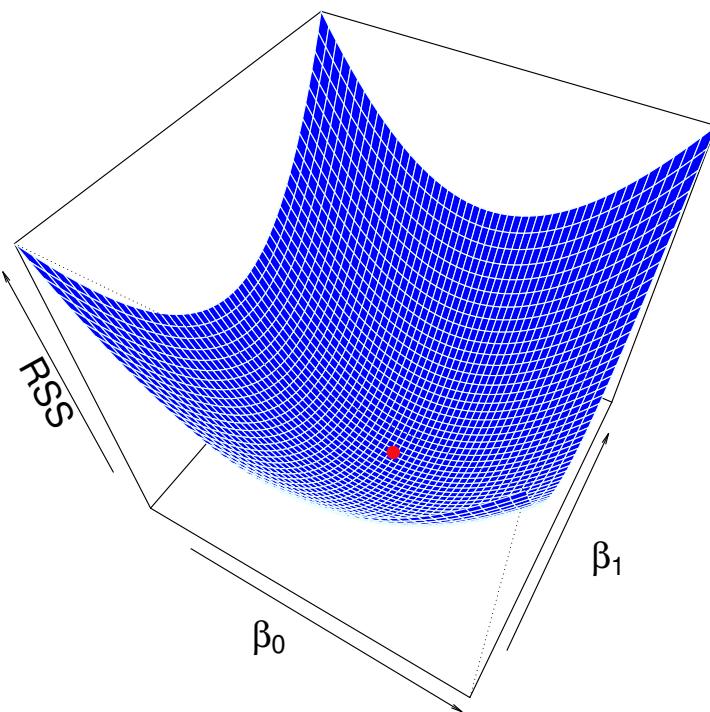
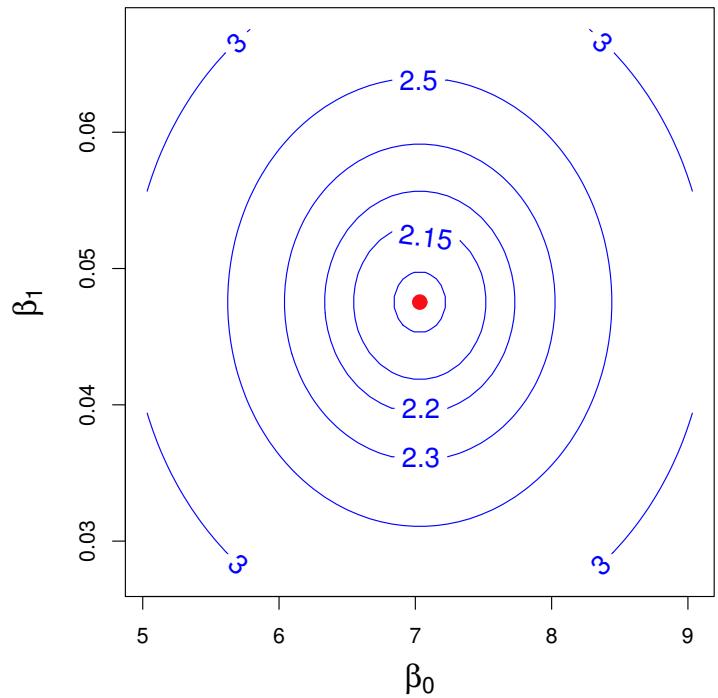
$$R = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2$$

$$R(\beta) = \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Find β_0 and β_1 that minimize:

$$R(\beta) = \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$$

Linear Regression



Credit: Introduction to Statistical Learning.

Linear Regression

Find β_0 and β_1 so that:

$$\operatorname{argmin}_{\beta} \left(\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Linear Regression

Find β_0 and β_1 so that:

$$\operatorname{argmin}_{\beta} \left(\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Minimize: $R(\beta_0, \beta_1)$, that is: $\frac{\partial R}{\partial \beta_0} = 0$ $\frac{\partial R}{\partial \beta_1} = 0$

Linear Regression

Find β_0 and β_1 so that:

$$\operatorname{argmin}_{\beta} \left(\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Minimize: $R(\beta_0, \beta_1)$, that is: $\frac{\partial R}{\partial \beta_0} = 0$ $\frac{\partial R}{\partial \beta_1} = 0$

$$\frac{\partial R}{\partial \beta_0} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_0} (y_i - \beta_0 - \beta_1 x_i)$$

Linear Regression

Find β_0 and β_1 so that:

$$\operatorname{argmin}_{\beta} \left(\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Minimize: $R(\beta_0, \beta_1)$, that is: $\frac{\partial R}{\partial \beta_0} = 0$ $\frac{\partial R}{\partial \beta_1} = 0$

$$\frac{\partial R}{\partial \beta_0} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_0} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-1) = 0$$

Linear Regression

Find β_0 and β_1 so that:

$$\operatorname{argmin}_{\beta} \left(\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \right)$$

Minimize: $R(\beta_0, \beta_1)$, that is: $\frac{\partial R}{\partial \beta_0} = 0$ $\frac{\partial R}{\partial \beta_1} = 0$

$$\frac{\partial R}{\partial \beta_0} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_0} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-1) = 0$$

$$\beta_0 = \frac{1}{n} \sum_{i=1}^n y_i - \beta_1 \frac{1}{n} \sum_{i=1}^n x_i$$

Linear Regression

$$\frac{\partial R}{\partial \beta_1} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_1} (y_i - \beta_0 - \beta_1 x_i)$$

Linear Regression

$$\frac{\partial R}{\partial \beta_1} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_1} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-x_i) = 0$$

Linear Regression

$$\frac{\partial R}{\partial \beta_1} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_1} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-x_i) = 0$$

$$\beta_1 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i - \sum_{i=1}^n \beta_0 x_i$$

Linear Regression

$$\frac{\partial R}{\partial \beta_1} = 2 \times \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times \frac{\partial}{\partial \beta_1} (y_i - \beta_0 - \beta_1 x_i)$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-x_i) = 0$$

$$\beta_1 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i - \sum_{i=1}^n \beta_0 x_i$$

Plugging β_0 in β_1 :

$$\beta_1 = \frac{\sum_{i=1}^n y_i x_i - \frac{1}{n} \sum_{i=1}^n y_i \sum_{i=1}^n x_i}{\sum_{i=1}^n x_i^2 - \frac{1}{n} \sum_{i=1}^n x_i \sum x_i}$$

Linear Regression

With more than one feature:

$$f(x) = \beta_0 + \sum_{j=1}^d \beta_j x_j$$

Find the β_j that minimize:

$$R = \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^d \beta_j x_{ij})^2$$

Let's write it more elegantly with matrices!

Matrix representation

Let X be an $n \times (d + 1)$ matrix where each row starts with a 1 followed by a feature vector.

Let y be the label vector of the training set.

Let β be the vector of weights (that we want to estimate!).

$$X := \begin{pmatrix} \textcolor{blue}{1} & x_{11} & \cdots & x_{1j} & \cdots & x_{1d} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \textcolor{blue}{1} & x_{i1} & \cdots & x_{ij} & \cdots & x_{id} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \textcolor{blue}{1} & x_{n1} & \cdots & x_{nj} & \cdots & x_{nd} \end{pmatrix}$$

$$y := \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix} \qquad \qquad \beta := \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_j \\ \vdots \\ \beta_d \end{pmatrix}$$

Normal Equation

We want to find $(d + 1)$ β 's that minimize R . We write R :

$$R(\beta) = \frac{1}{2n} \|(y - X\beta)\|^2$$

$$R(\beta) = \frac{1}{2n} (y - X\beta)^T (y - X\beta)$$

$$\frac{\partial R}{\partial \beta} = -\frac{1}{n} X^T (y - X\beta)$$

We have that:

$$\frac{\partial^2 R}{\partial \beta} = -\frac{1}{n} X^T X$$

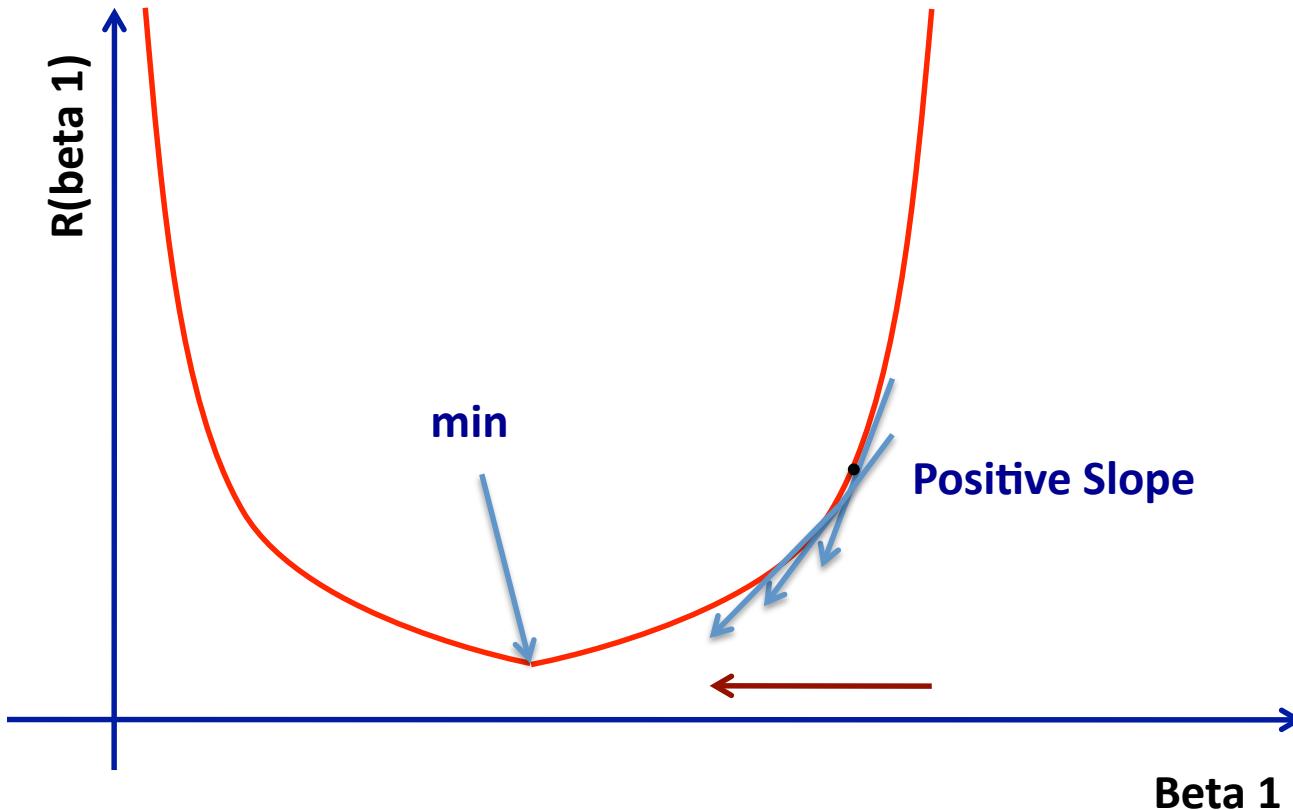
is positive definite which ensures that β is a minimum. We solve:

$$X^T (y - X\beta) = 0$$

The unique solution is:

$$\beta = (X^T X)^{-1} X^T y$$

Gradient descent



Gradient descent

Gradient Descent is an optimization method.

Repeat until convergence:

Update **simultaneously** all β_j for ($j = 0$ and $j = 1$)

$$\beta_0 := \beta_0 - \alpha \frac{\partial}{\partial \beta_0} R(\beta_0, \beta_1)$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial}{\partial \beta_1} R(\beta_0, \beta_1)$$

Gradient descent

Gradient Descent is an optimization method.

Repeat until convergence:

Update **simultaneously** all β_j for ($j = 0$ and $j = 1$)

$$\beta_0 := \beta_0 - \alpha \frac{\partial}{\partial \beta_0} R(\beta_0, \beta_1)$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial}{\partial \beta_1} R(\beta_0, \beta_1)$$

α is a learning rate.

Gradient descent

In the linear case:

$$\frac{\partial R}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-1) = 0$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-x_i)$$

Let's generalize it!

Gradient descent

In the linear case:

$$\frac{\partial R}{\partial \beta_0} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-1) = 0$$

$$\frac{\partial R}{\partial \beta_1} = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) \times (-x_i)$$

Repeat until convergence:

Update **simultaneously** all β_j for ($j = 0$ and $j = 1$)

$$\beta_0 := \beta_0 - \alpha \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)$$

$$\beta_1 := \beta_1 - \alpha \frac{1}{n} \sum_{i=1}^n (\beta_0 + \beta_1 x_i - y_i)(x_i)$$

Pros and Cons

Analytical approach: Normal Equation

- + No need to specify a convergence rate or iterate.
- Works only if $X^T X$ is invertible
- Very slow if d is large $O(d^3)$ to compute $(X^T X)^{-1}$

Iterative approach: Gradient Descent

- + Effective and efficient even in high dimensions.
- Iterative (sometimes need many iterations to converge).
- Needs to choose the rate α .

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{stdev(x_i)}$$

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{stdev(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{stdev(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.
3. **R should decrease** after each iteration.

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{stdev(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.
3. **R should decrease** after each iteration.
4. **Declare convergence** if it start decreasing by less ϵ

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{stdev(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.
3. **R should decrease** after each iteration.
4. **Declare convergence** if it start decreasing by less ϵ
5. If $X^T X$ is not **invertible**?

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

$$x_i := \frac{x_i - \mu_i}{\text{stdev}(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.
3. **R should decrease** after each iteration.
4. **Declare convergence** if it start decreasing by less ϵ
5. If $X^T X$ is not **invertible**?
 - (a) Too many features as compared to the number of examples (e.g., 50 examples and 500 features)
 - (b) Features linearly dependent: e.g., weight in pounds and in kilo.

Credit

- The elements of statistical learning. Data mining, inference, and prediction. 10th Edition 2009. T. Hastie, R. Tibshirani, J. Friedman.
- Machine Learning 1997. Tom Mitchell.

Classification

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n) / x_i \in \mathbb{R}^d$ and y_i is discrete (categorical/qualitative), $y_i \in \mathbb{Y}$.

Example $\mathbb{Y} = \{-1, +1\}, \mathbb{Y} = \{0, 1\}$.

Task: Learn a classification function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{Y}$$

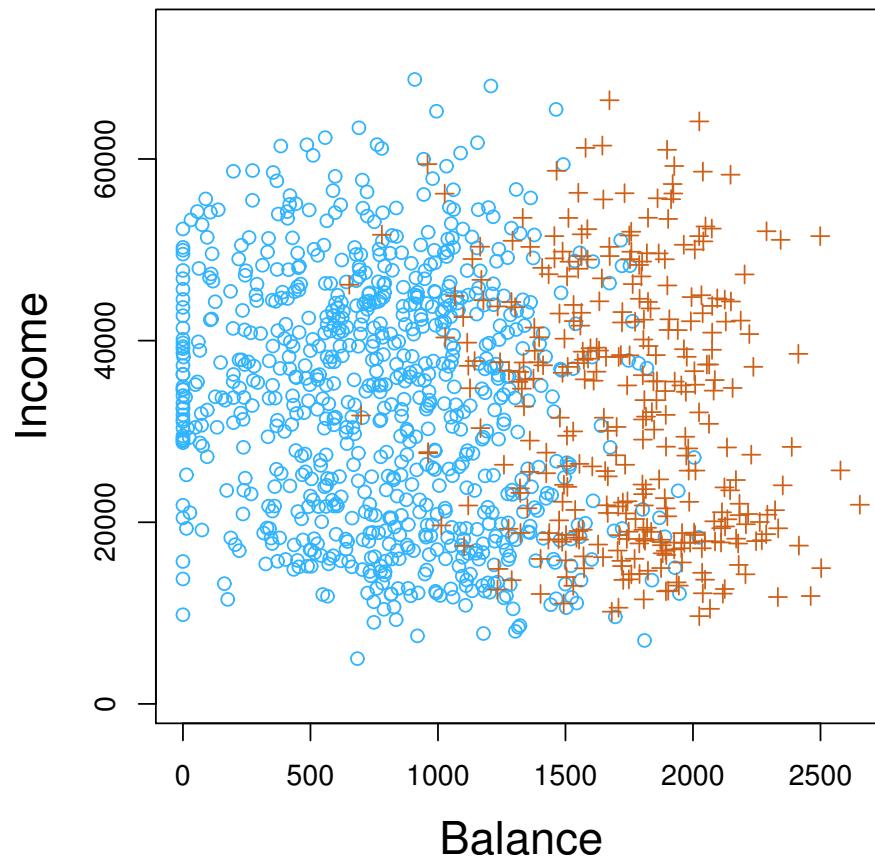
Linear Classification: A classification model is said to be linear if it is represented by a linear function f (linear hyperplane)

Classification: examples

1. Fruit classification → Banana/Orange?
2. Email Spam/Ham → Which email is junk?
3. Tumor benign/malignant → Which patient has cancer?
4. Credit default/not default → Which customers will default on their credit card debt?

Balance	Income	Default
300	\$20,000.00	no
2000	\$60,000.00	no
5000	\$45,000.00	yes
.	.	.
.	.	.
.	.	.

Classification: example

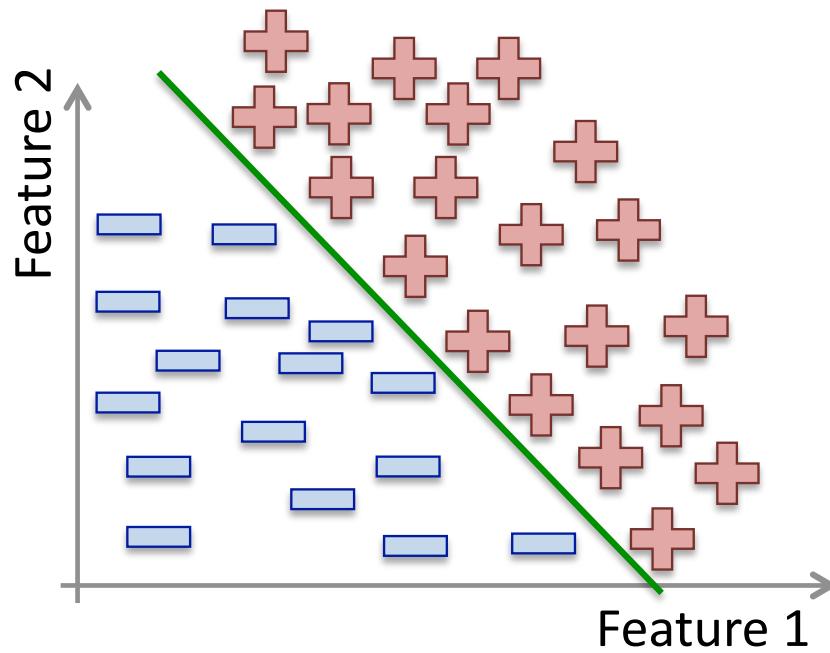


Credit: Introduction to Statistical Learning.

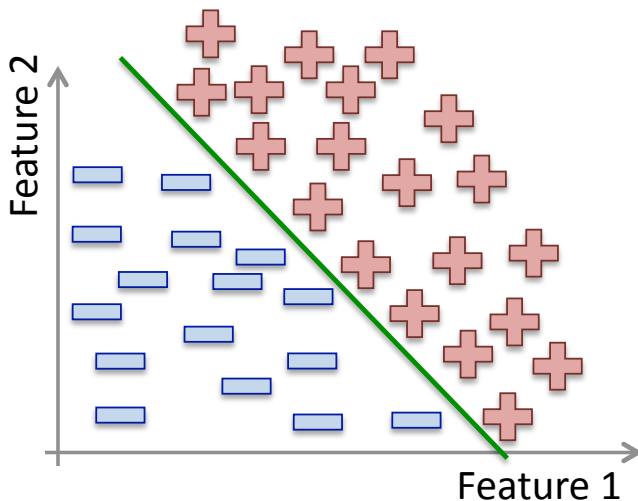
Perceptron

- Belongs to Neural Networks class of algorithms (algorithms that try to mimic how the brain functions).
- The first algorithm used was the Perceptron (Resenblatt 1959).
- Worked extremely well to recognize:
 1. handwritten characters (LeCun et al. 1989),
 2. spoken words (Lang et al. 1990),
 3. faces (Cottrel 1990)
- NN were popular in the 90's but then lost some of its popularity.
- Now NN back with deep learning.

Perfectly separable data

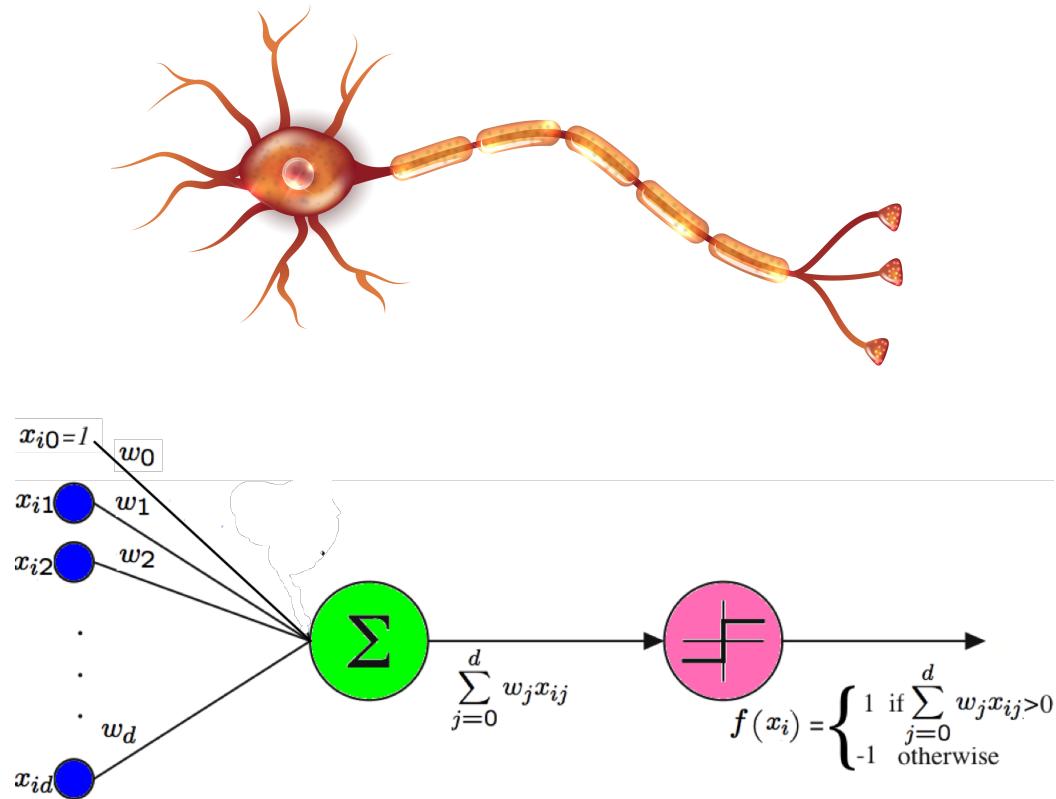


Perceptron



- Linear classification method.
- Simplest classification method.
- Simplest neural network.
- For perfectly separated data.

Perceptron



Given n examples and d features.

$$f(x_i) = \text{sign}\left(\sum_{j=0}^d w_j x_{ij}\right)$$

Perceptron

- Works perfectly if data is linearly separable. If not, it will not converge.
- Idea: Start with a random hyperplane and adjust it using your training data.
- Iterative method.

Perceptron

Perceptron Algorithm

Input: A set of examples, $(x_1, y_1), \dots, (x_n, y_n)$

Output: A perceptron defined by (w_0, w_1, \dots, w_d)

Begin

2. Initialize the weights w_j to 0 $\forall j \in \{0, \dots, d\}$
3. Repeat until convergence
 4. For each example $x_i \ \forall i \in \{1, \dots, n\}$
 5. if $y_i f(x_i) \leq 0$ #an error?
 6. update all w_j with $w_j := w_j + y_i x_i$ #adjust the weights

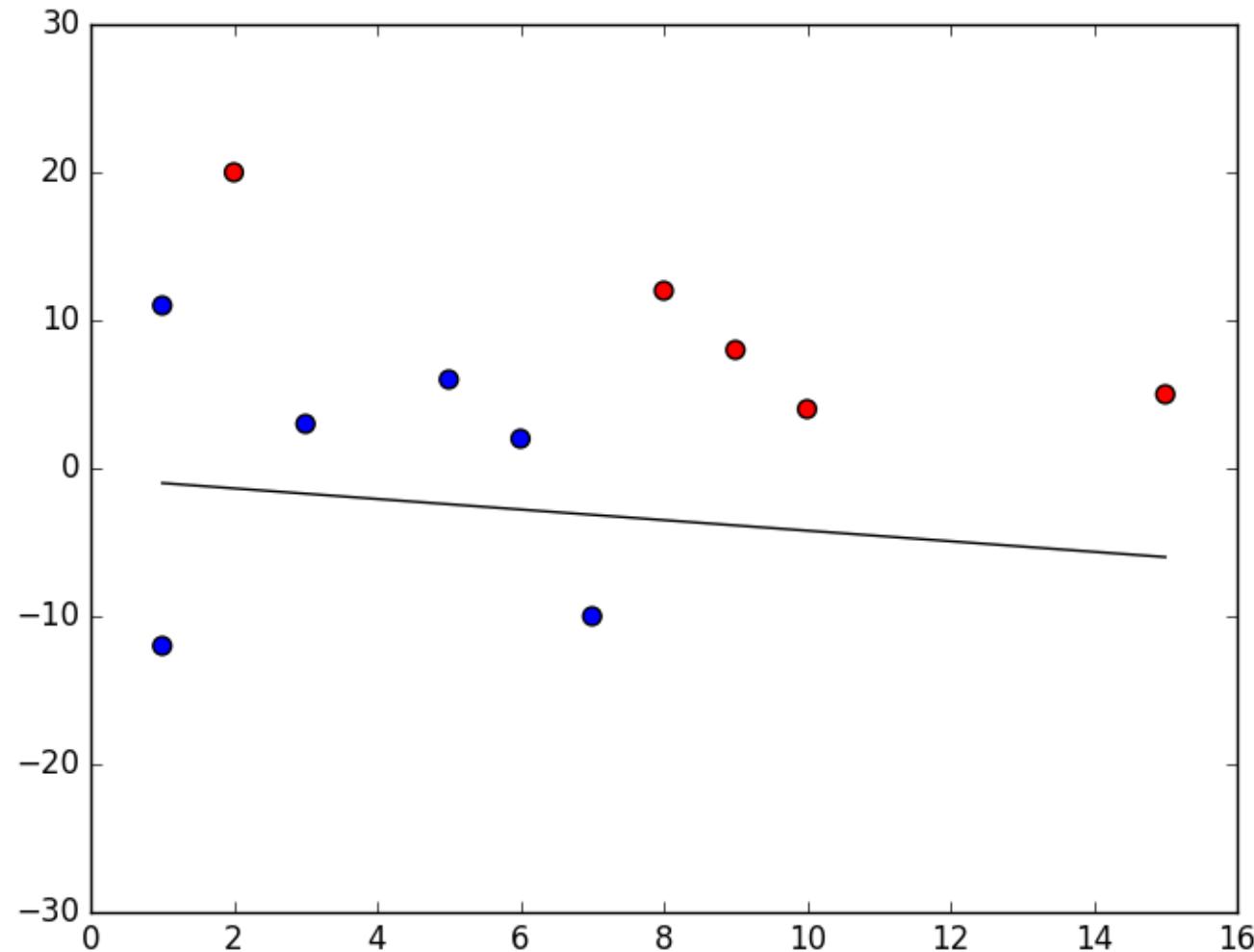
End

Perceptron

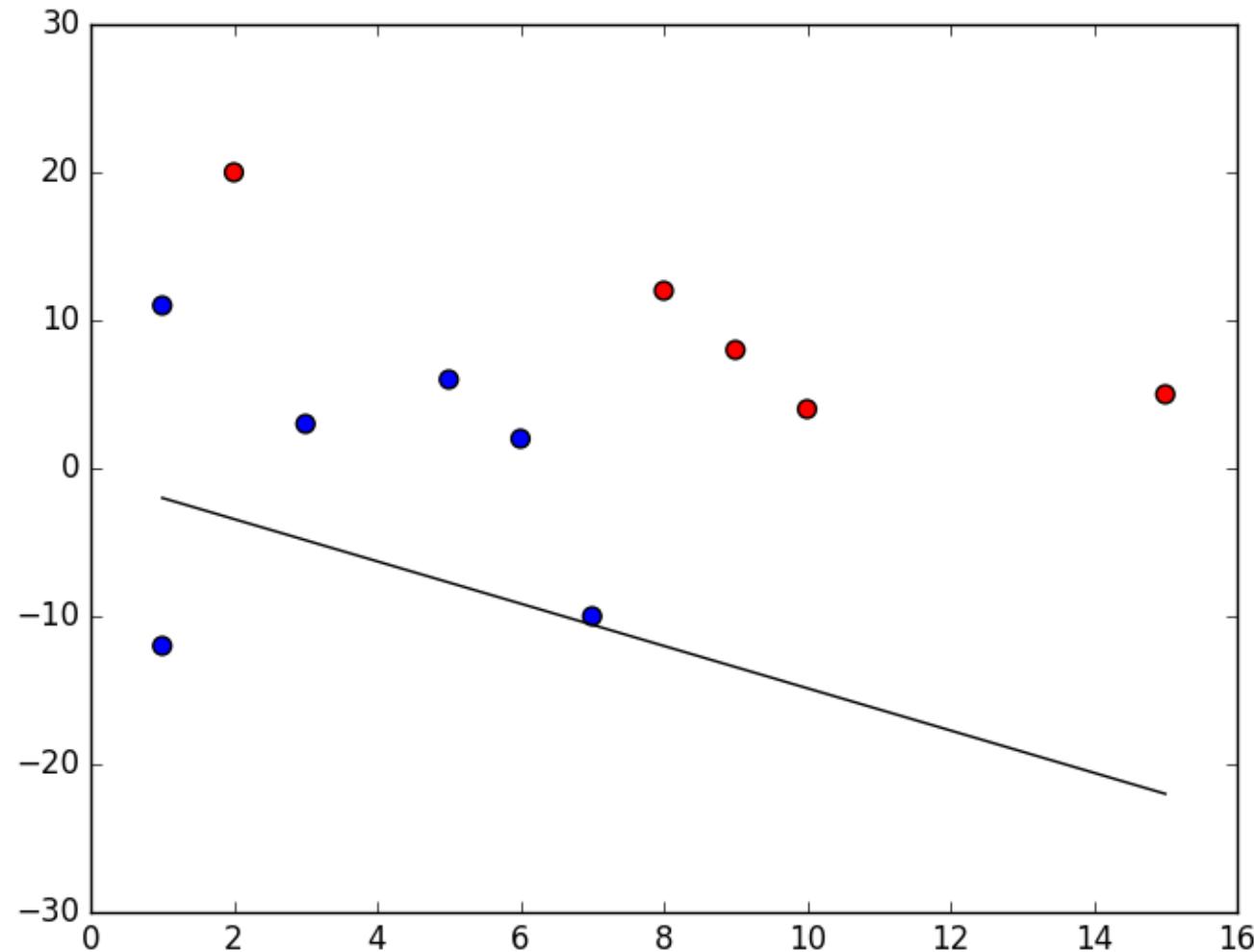
Some observations:

- The weights w_1, \dots, w_d determine the slope of the decision boundary.
- w_0 determines the offset of the decision boundary (sometimes noted b).
- Line 6 corresponds to:
 - Mistake on positive: add x to weight vector.
 - Mistake on negative: subtract x from weight vector.
 - Some other variants of the algorithm add or subtract 1.
- Convergence happens when the weights do not change anymore (difference between the last two weight vectors is 0).

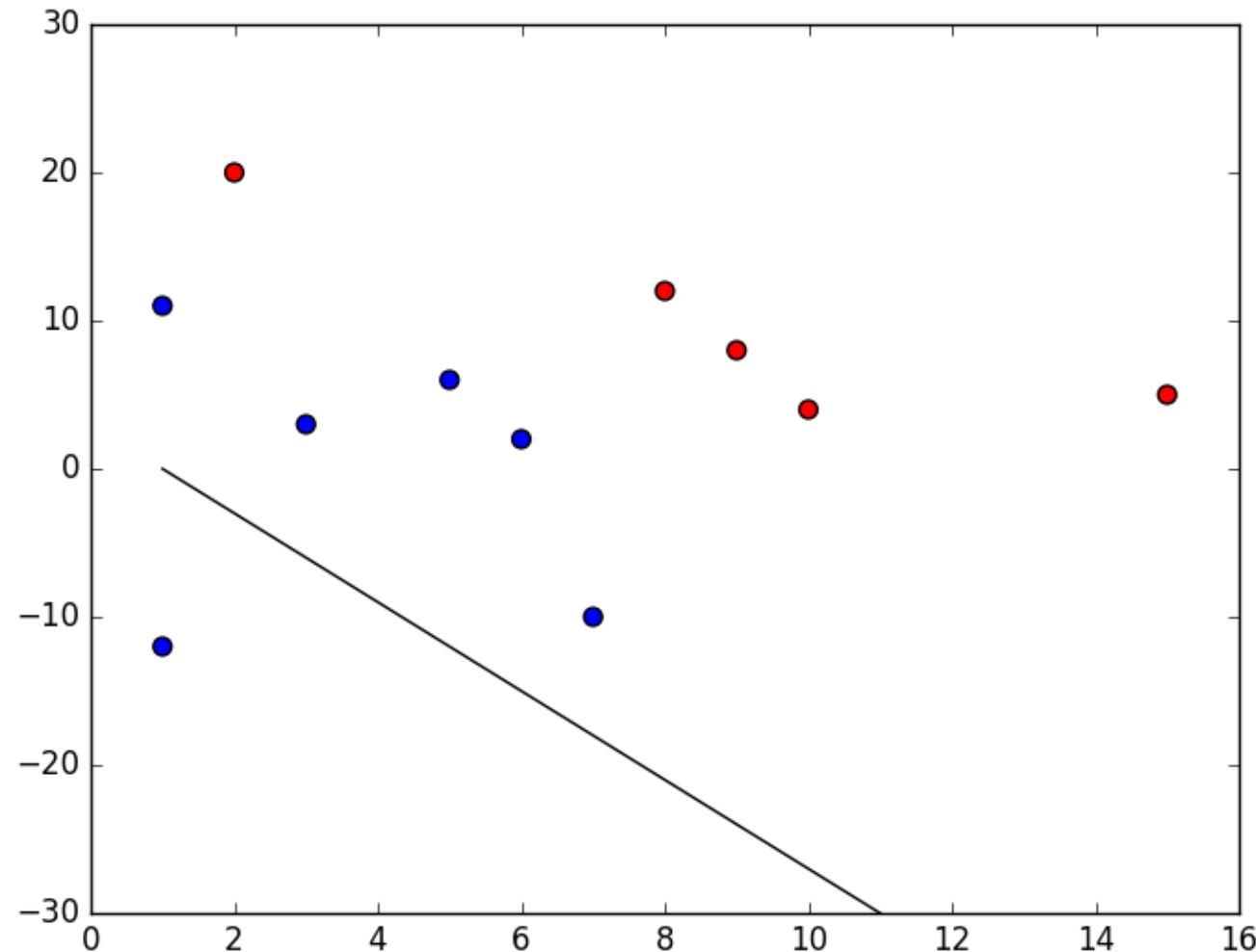
Perceptron: Example



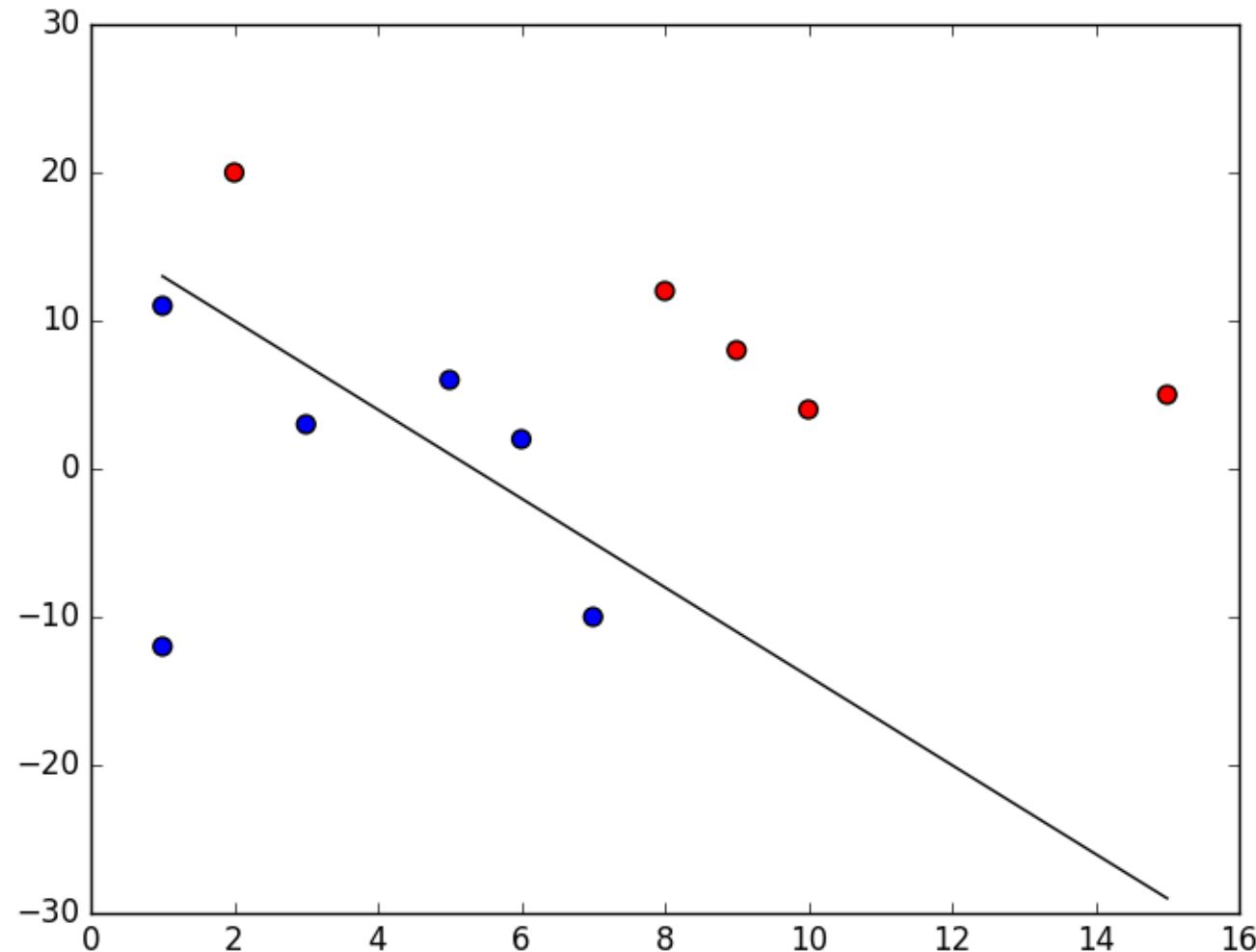
Perceptron: Example



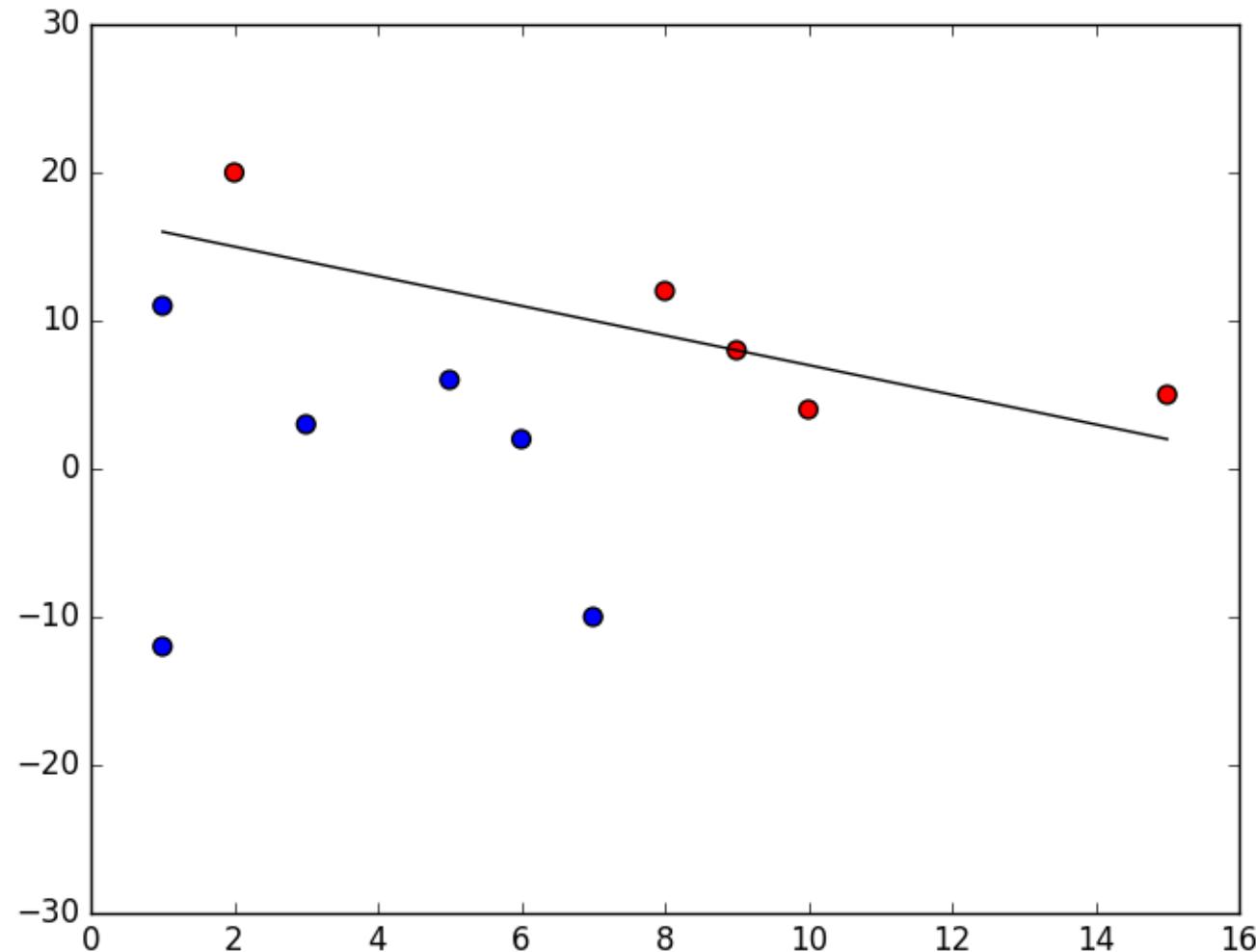
Perceptron: Example



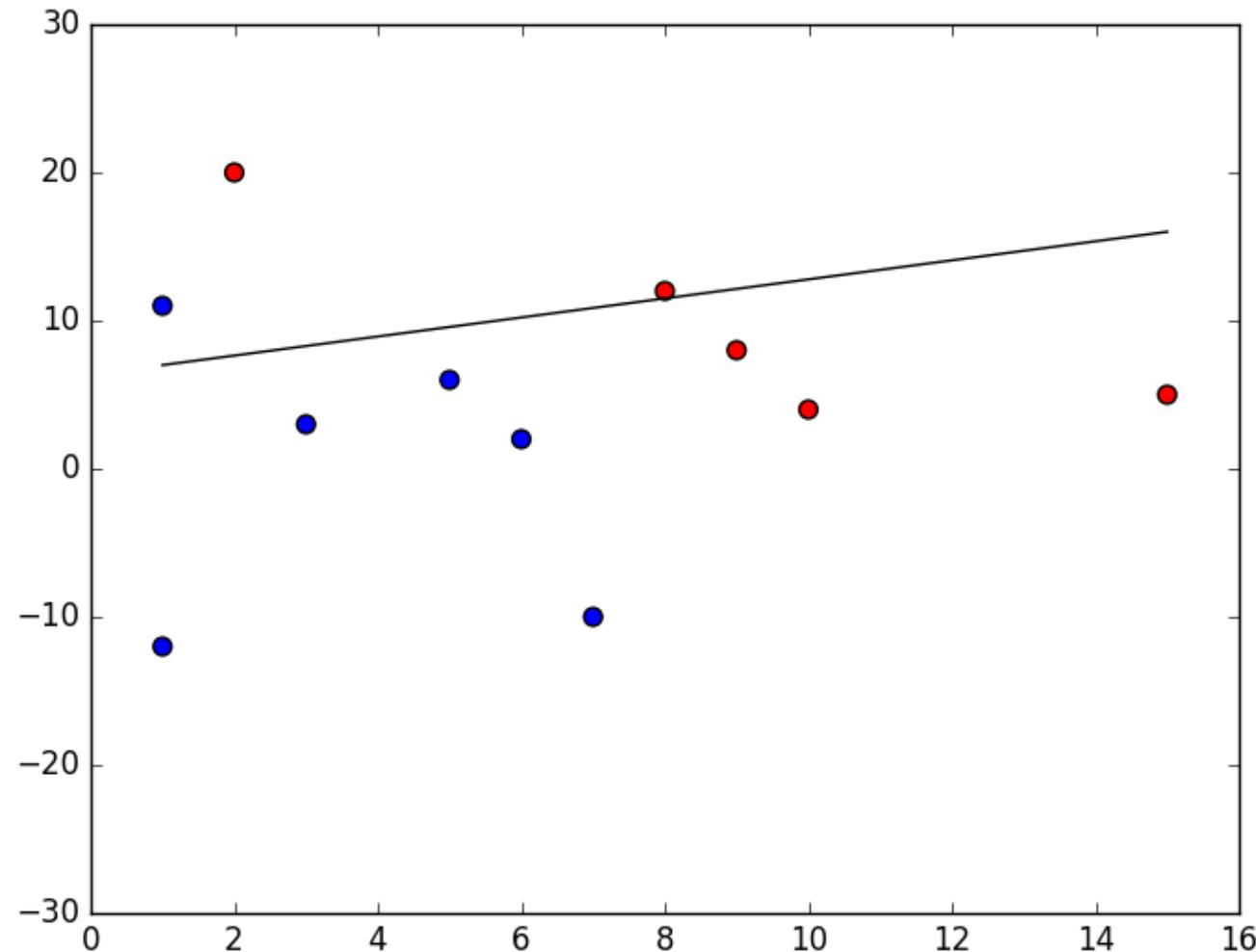
Perceptron: Example



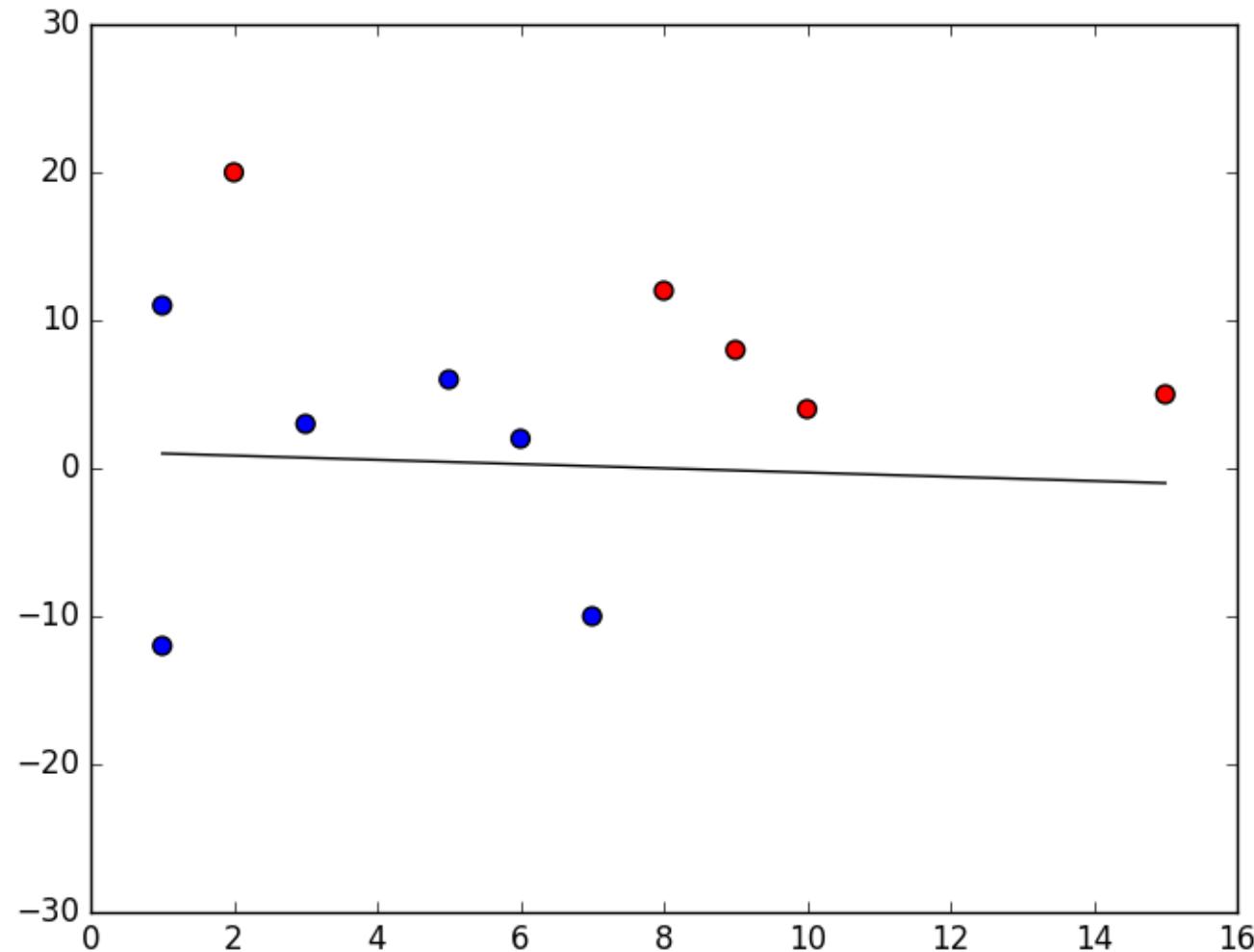
Perceptron: Example



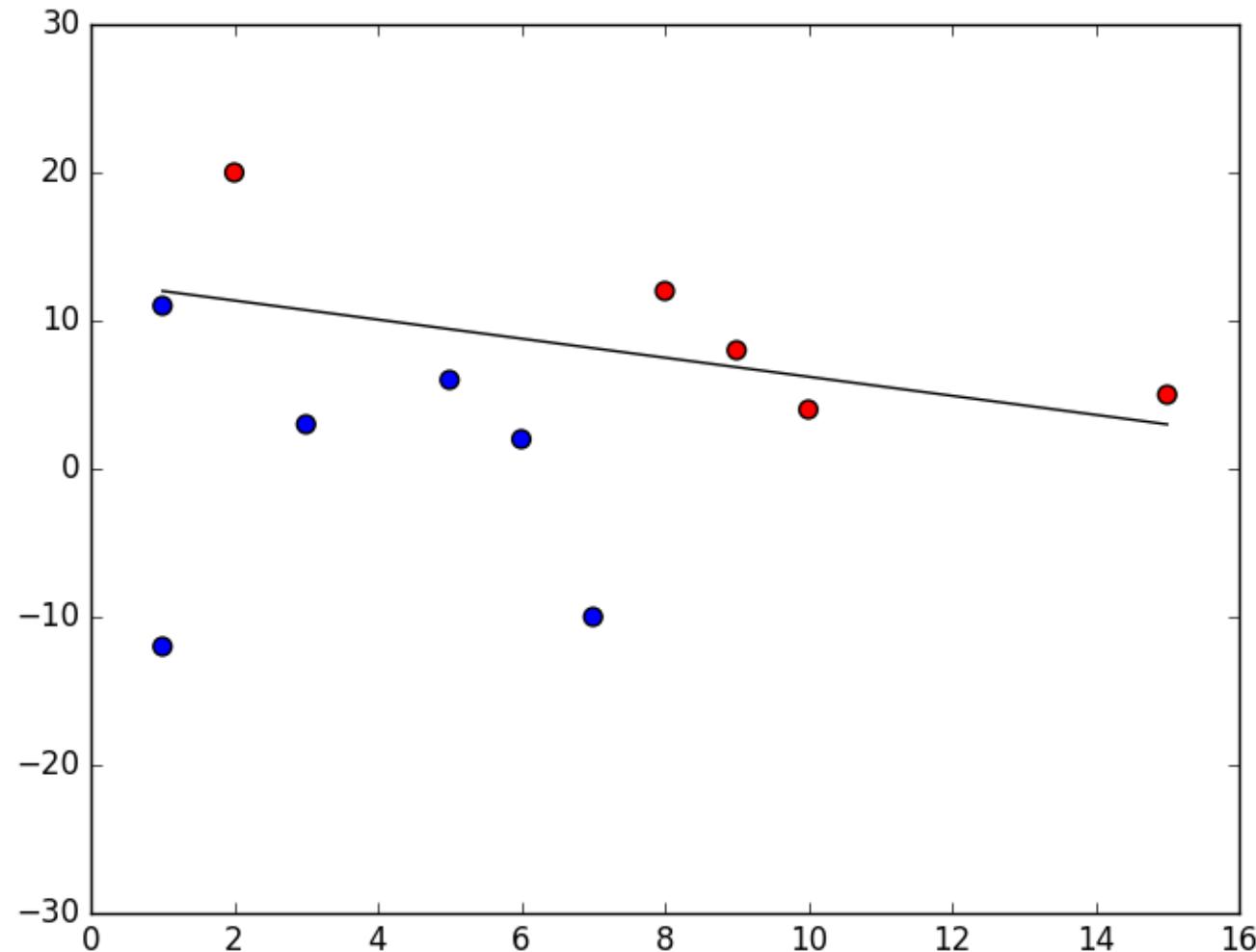
Perceptron: Example



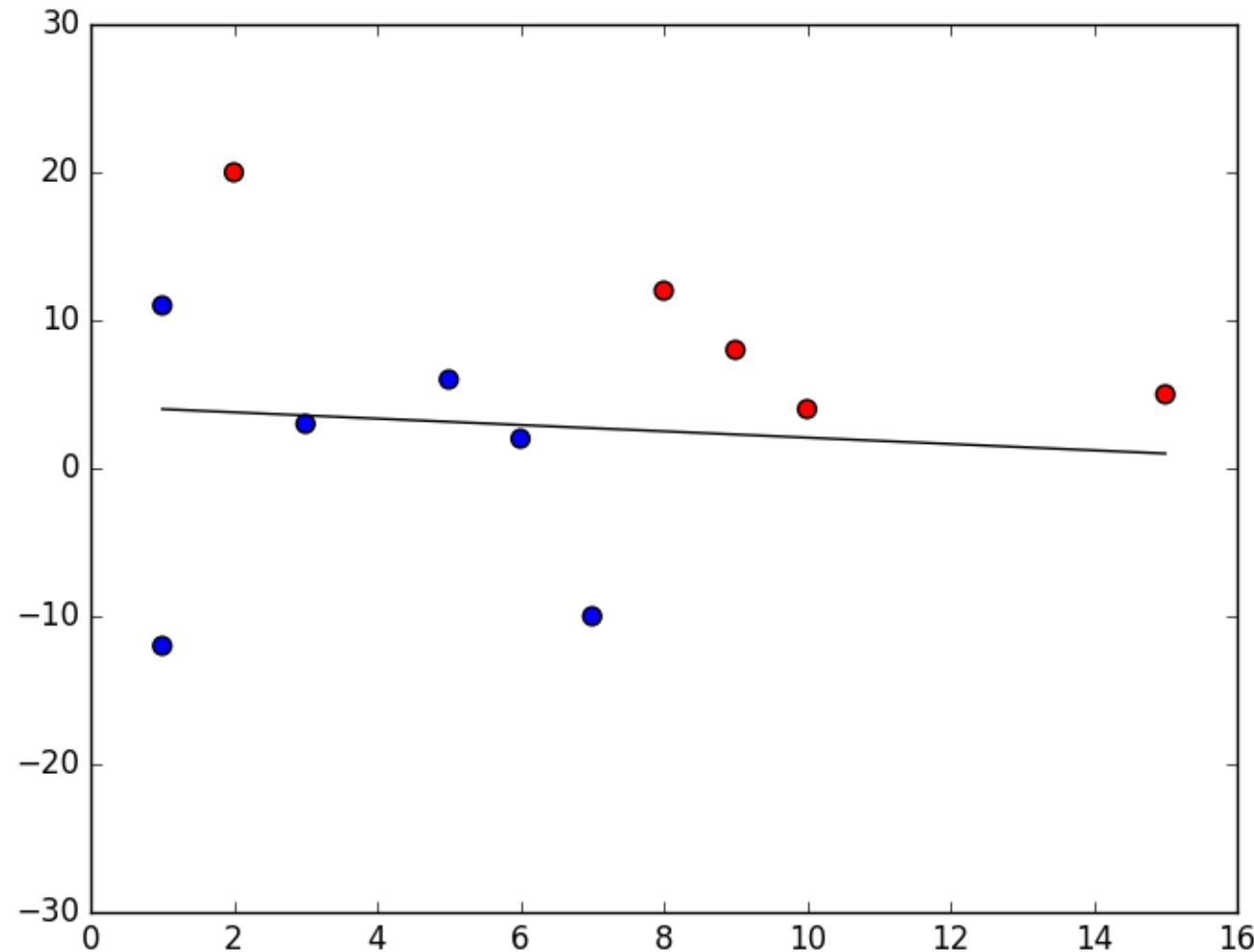
Perceptron: Example



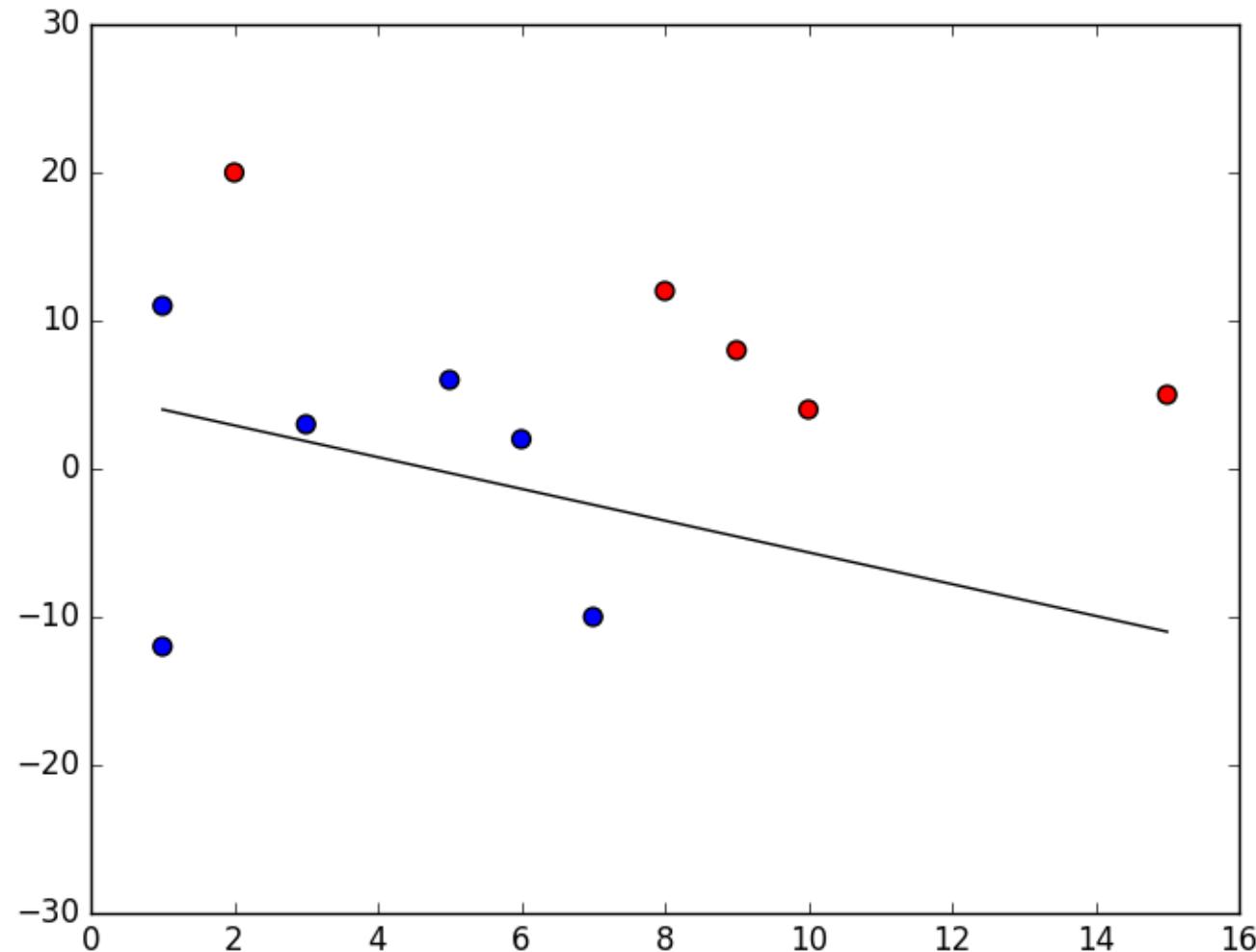
Perceptron: Example



Perceptron: Example

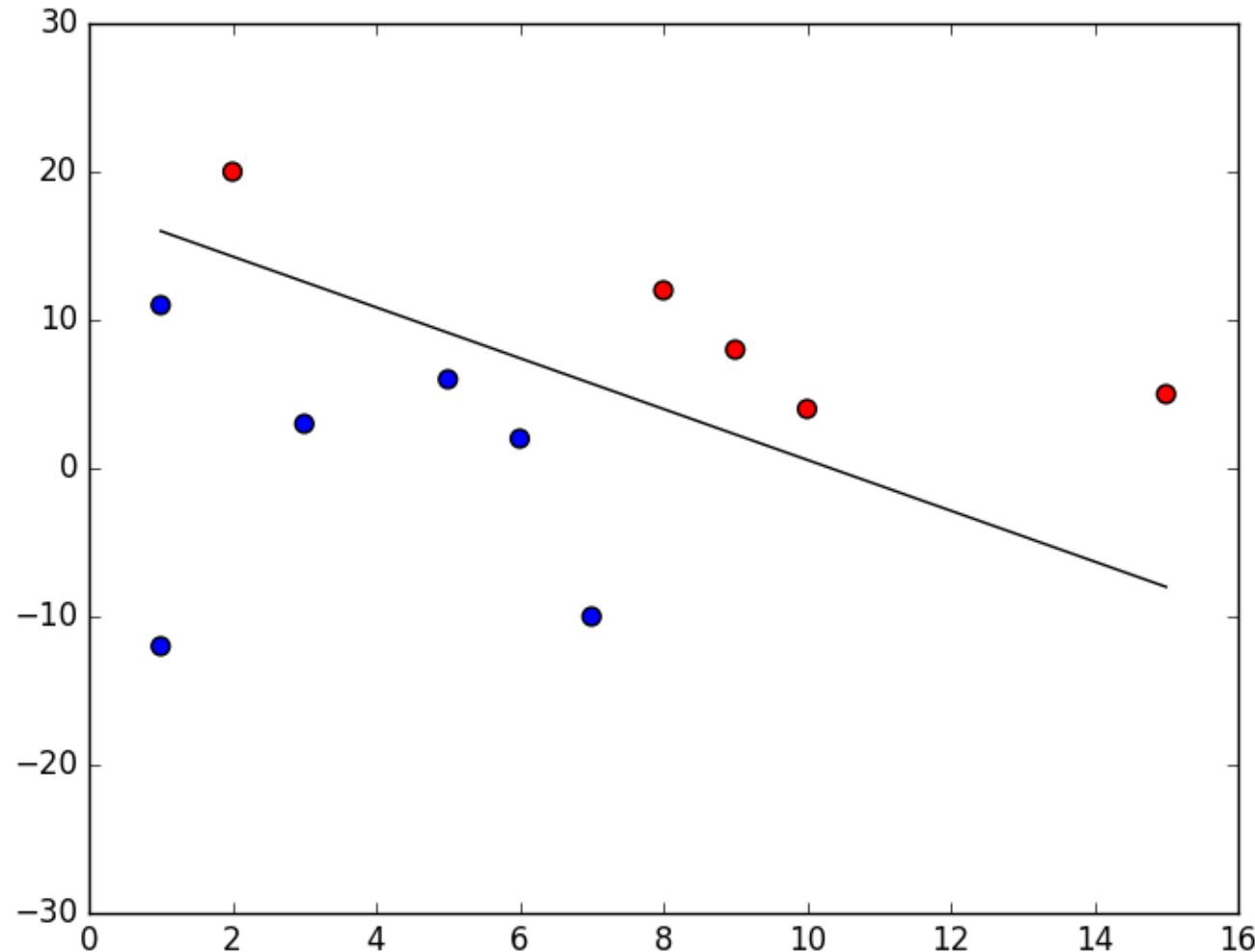


Perceptron: Example



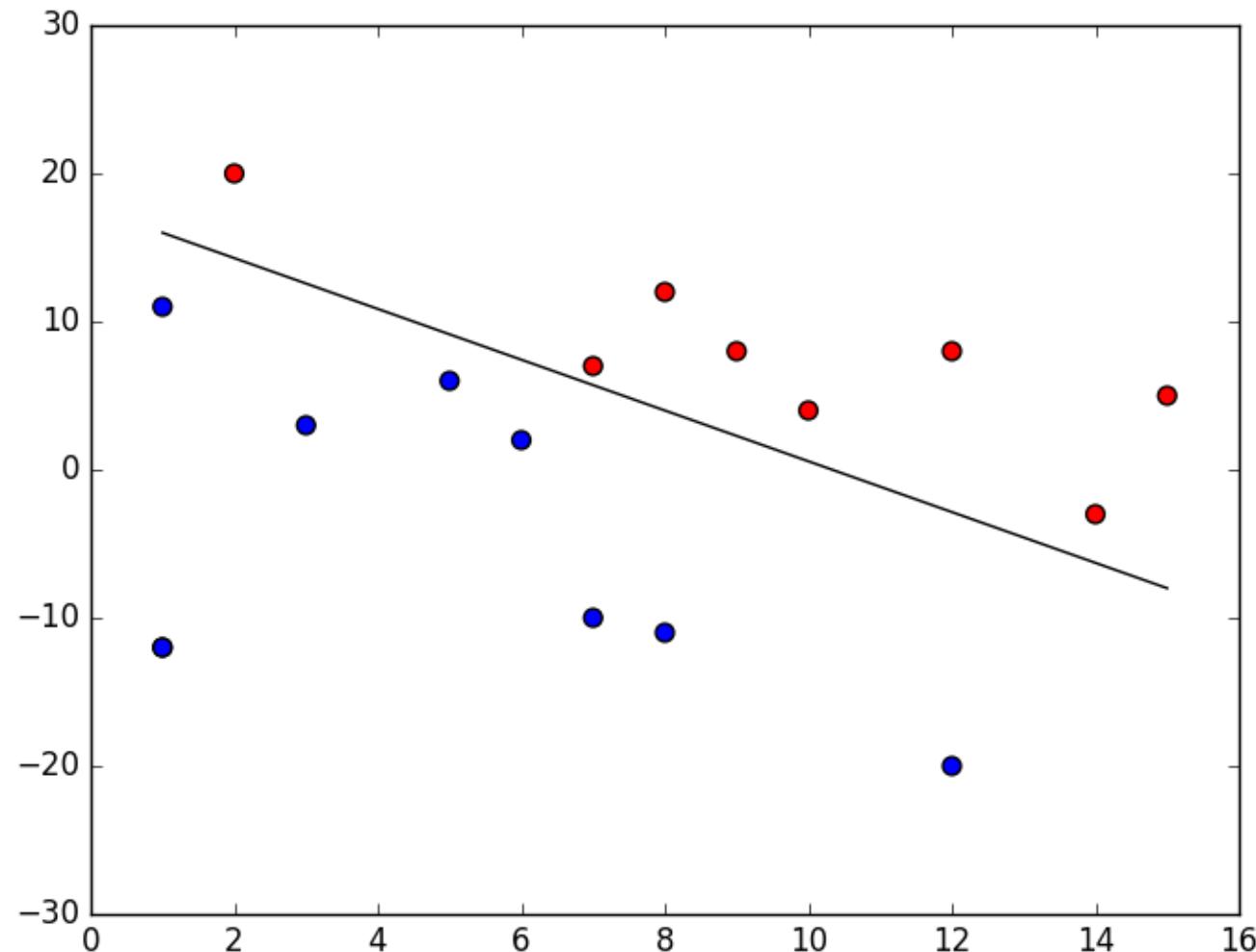
Perceptron: Example

Finally converged!



Perceptron: Example

With some test data:



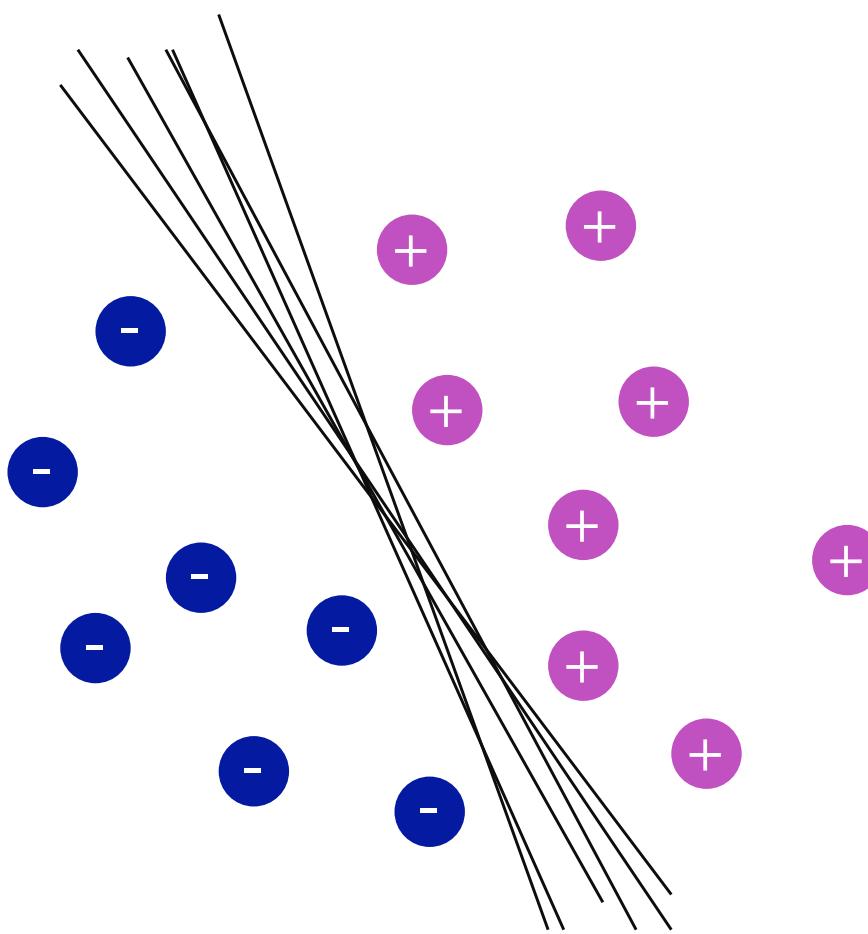
Perceptron

- The w_i determine the contribution of x_i to the label.
- $-w_0$ is a quantity that $\sum_{i=1}^n w_i x_i$ needs to exceed for the perceptron to output 1.
- Can be used to represent many Boolean functions: AND, OR, NAND, NOR, NOT but not all of them (e.g., XOR).

From perceptron to NN

- Neural networks use the ability of the perceptrons to represent elementary functions and combine them in a network of layers of elementary questions.
- However, a cascade of linear functions is still linear!
- And we want networks that represent highly non-linear functions.

Choice of the hyperplane



Lots of possible solutions!

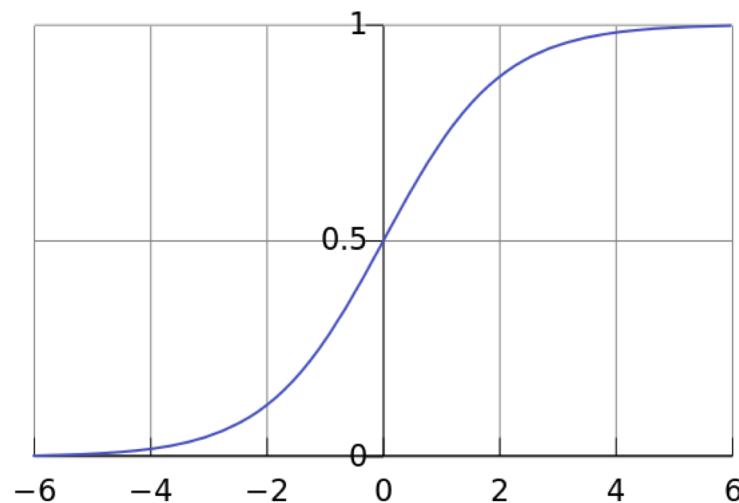
Digression: Idea of SVM is to find the optimal solution.

Credit

- The elements of statistical learning. Data mining, inference, and prediction. 10th Edition 2009. T. Hastie, R. Tibshirani, J. Friedman.
- Machine Learning 1997. Tom Mitchell.

Artificial Intelligence

Machine Learning Logistic Regression



Classification

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n) / x_i \in \mathbb{R}^d$ and y_i is discrete (categorical/qualitative), $y_i \in \mathbb{Y}$.

Example $\mathbb{Y} = \{-1, +1\}, \mathbb{Y} = \{0, 1\}$.

Task: Learn a classification function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{Y}$$

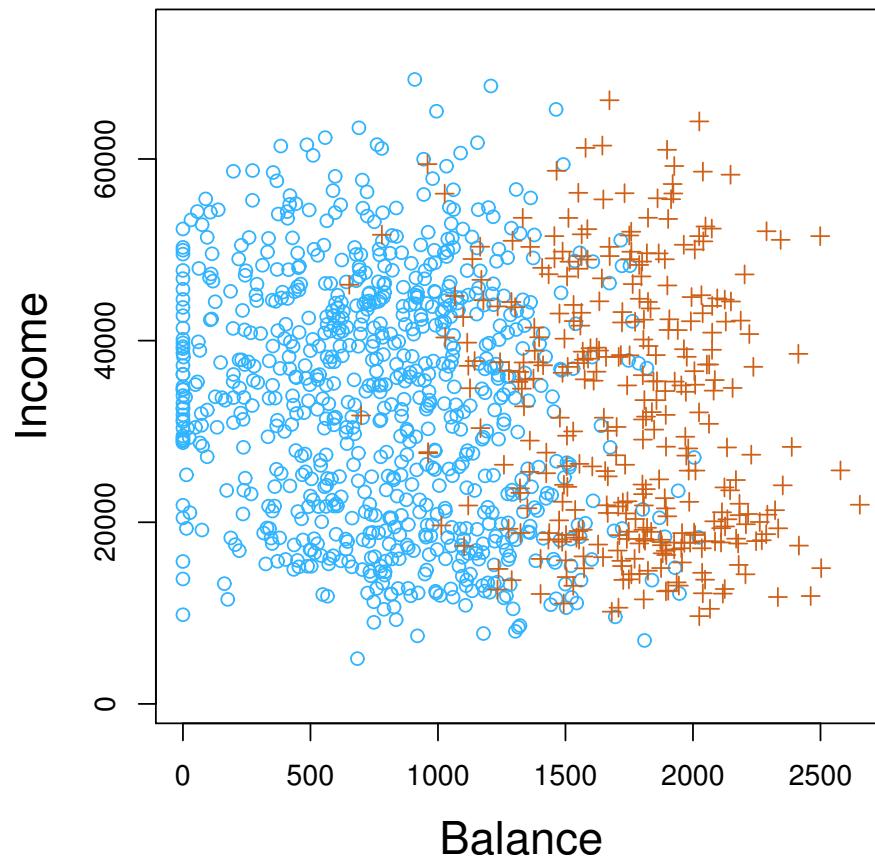
Linear Classification: A classification model is said to be linear if it is represented by a linear function f (linear hyperplane)

Classification: examples

1. Email Spam/Ham → Which email is junk?
2. Tumor benign/malignant → Which patient has cancer?
3. Credit default/not default → Which customers will default on their credit card debt?

Balance	Income	Default
300	\$20,000.00	no
2000	\$60,000.00	no
5000	\$45,000.00	yes
.	.	.
.	.	.
.	.	.

Classification: example



Credit: Introduction to Statistical Learning.

Classification

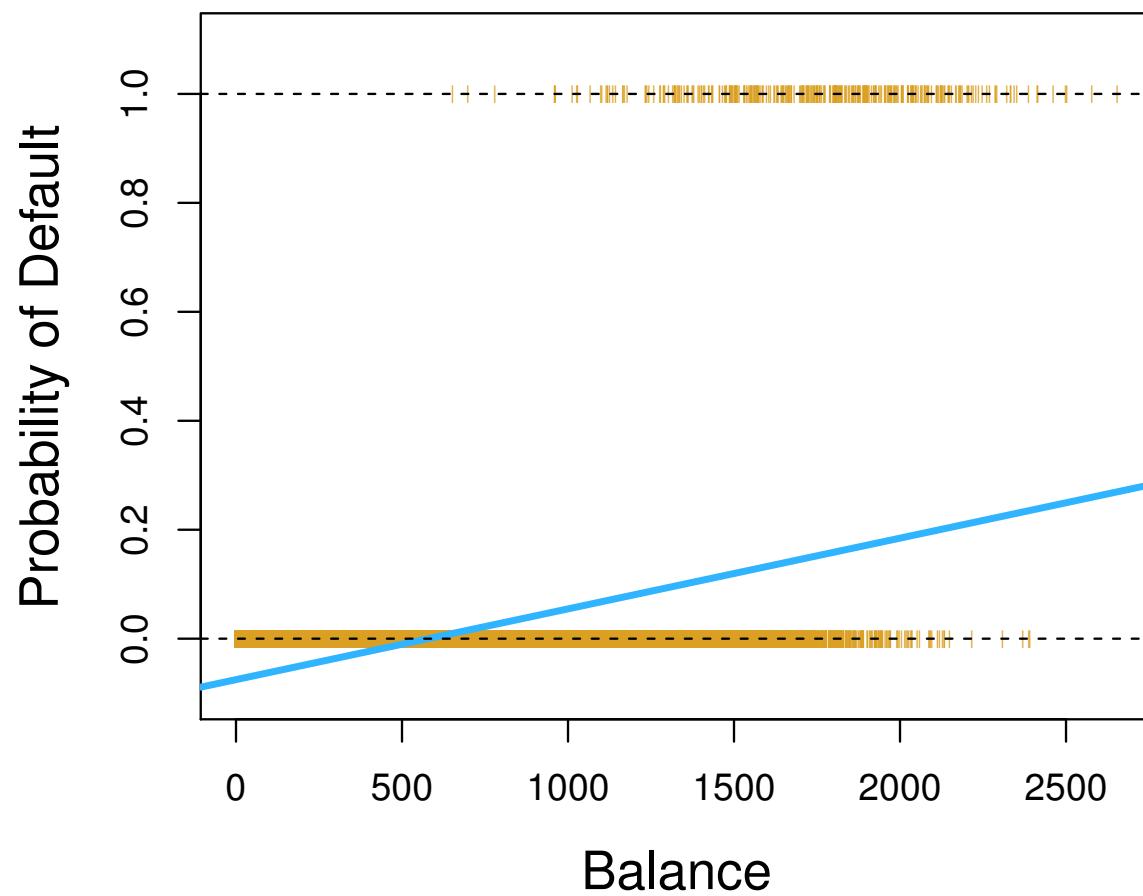
- We can't predict Credit Card Default with any certainty. Suppose we want to predict how likely is a customer to default. That is output a probability between 0 and 1 that a customer will default.
- It makes sense and would be suitable and practical.
- In this case, the output is real (regression) but is bounded (classification).

$$P(y|x) = P(\text{default} = \text{yes} \mid \text{balance})$$

Classification

- Can we use linear regression?
- Yes. However...
 - Works only for *Binary* classification (2 classes). Won't work for *Multiclass* classification e.g.,
 $\mathbb{Y} = \{ \text{green, blue, brown} \}$
 $\mathbb{Y} = \{ \text{stroke, heart attack, drug overdose} \}$
 - If we use linear regression, some of the predictions will be outside of [0,1].
 - Model can be poor. Example.

Classification: example



Credit: Introduction to Statistical Learning.

Classification

$$y = f(x) = \beta_0 + \beta_1 x$$

Default = $\beta_0 + \beta_1 \times \text{Balance}$

Classification

$$y = f(x) = \beta_0 + \beta_1 x$$

$$\text{Default} = \beta_0 + \beta_1 \times \text{Balance}$$

We want $0 \leq f(x) \leq 1$; $f(x) = P(y = 1|x)$

Classification

$$y = f(x) = \beta_0 + \beta_1 x$$

$$\text{Default} = \beta_0 + \beta_1 \times \text{Balance}$$

We want $0 \leq f(x) \leq 1$; $f(x) = P(y = 1|x)$

We use the sigmoid function:

$$g(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$

Classification

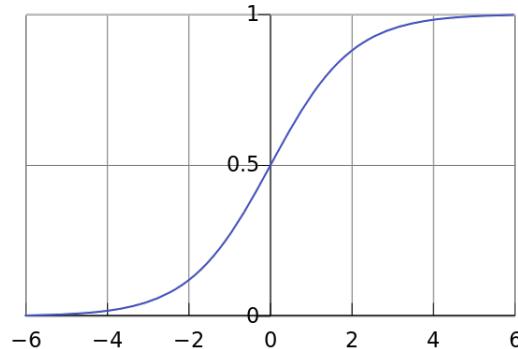
$$y = f(x) = \beta_0 + \beta_1 x$$

Default = $\beta_0 + \beta_1 \times \text{Balance}$

We want $0 \leq f(x) \leq 1$; $f(x) = P(y = 1|x)$

We use the sigmoid function:

$$g(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$



$g(z) \rightarrow 1$ when $z \rightarrow +\infty$ $g(z) \rightarrow 0$ when $z \rightarrow -\infty$

Logistic Regression

$$g(\beta_0 + \beta_1 x) = \frac{e^{(\beta_0 + \beta_1 x)}}{1 + e^{(\beta_0 + \beta_1 x)}}$$

$f(x) = g(\beta_0 + \beta_1 x)$

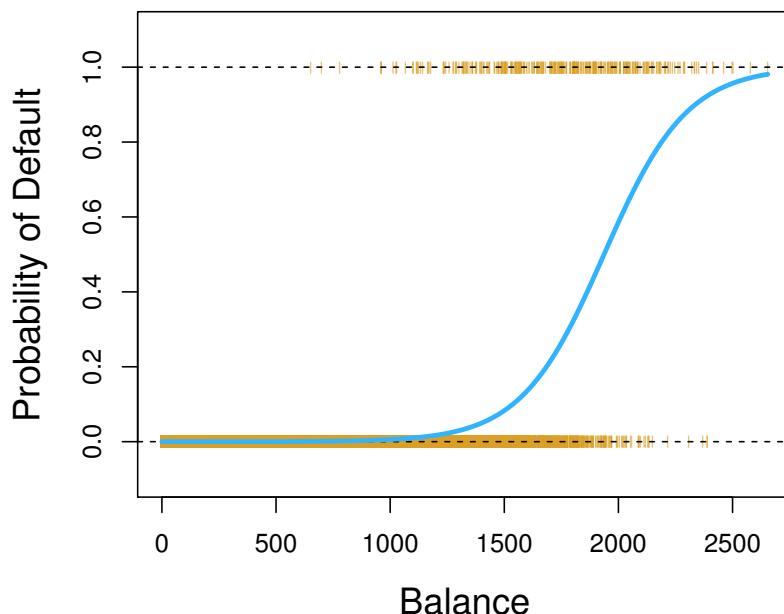
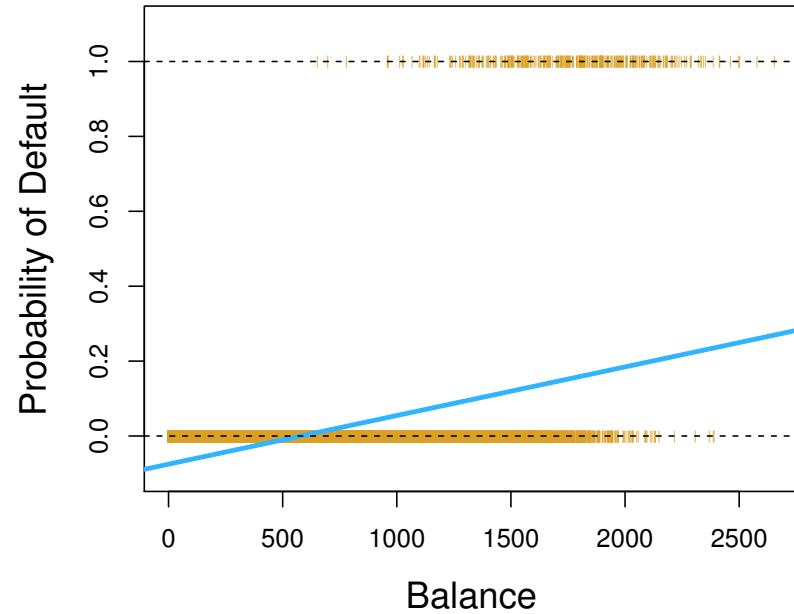
In general:

$$f(x) = g\left(\sum_{j=1}^d \beta_j x_j\right)$$

In other words, cast the output to bring the linear function quantity between 0 and 1.

Note: One can use other S-shaped functions.

Logistic Regression



Credit: [Introduction to Statistical Learning](#).

Logistic regression is not a regression method but a classification method!

Logistic Regression

How to make a prediction?

- Suppose $\beta_0 = -10.65$ and $\beta_1 = 0.0055$. What is the probability of default for a customer with \$1,000 balance?

Logistic Regression

How to make a prediction?

- Suppose $\beta_0 = -10.65$ and $\beta_1 = 0.0055$. What is the probability of default for a customer with \$1,000 balance?

$$P(\text{default} = \text{yes} | \text{balance} = 1000) = \frac{1}{1 + e^{10.65 - 0.0055 * 1000}}$$

$$P(\text{default} = \text{yes} | \text{balance} = 1000) = 0.00576$$

Logistic Regression

How to make a prediction?

- Suppose $\beta_0 = -10.65$ and $\beta_1 = 0.0055$. What is the probability of default for a customer with \$1,000 balance?

$$P(\text{default} = \text{yes} | \text{balance} = 1000) = \frac{1}{1 + e^{10.65 - 0.0055 * 1000}}$$

$$P(\text{default} = \text{yes} | \text{balance} = 1000) = 0.00576$$

- To predict the class:

If $g(z) \geq 0.5$ predict $y = 1$ ($z \geq 0$)

If $g(z) < 0.5$ predict $y = 0$ ($z < 0$)

Logistic Regression

How to find the β 's?

$$R(\beta) = \frac{1}{n} \sum_{i=1}^m \frac{1}{2} (f(x) - y)^2$$

$$Loss = \frac{1}{2} (f(x) - y)^2$$

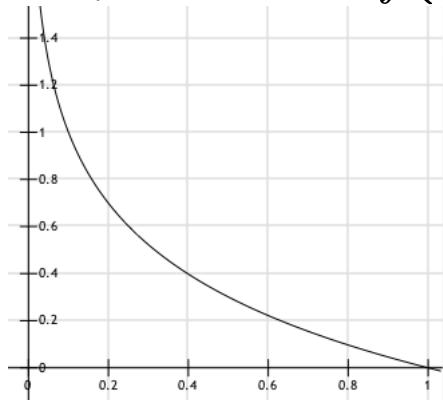
- Remember, $f(x)$ is now the logistic function so the $(f(x) - y)^2$ is not the quadratic function we had when f was linear.
- Cost is a complicated non-linear function!
- Many local optima, hence Gradient Descent will not find the global optimum!
- We need a different function that is convex.

Logistic Regression

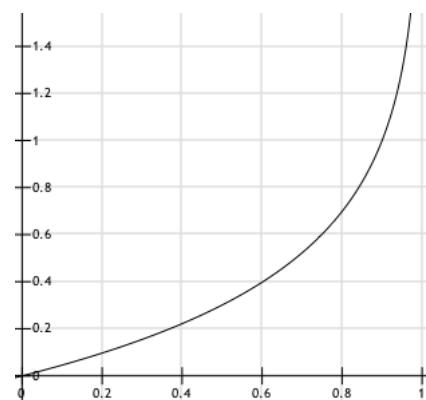
New Convex function:

$$Cost(f(x), y) = \begin{cases} -\log(f(x)) & \text{if } y = 1 \\ -\log(1 - f(x)) & \text{if } y = 0 \end{cases}$$

1. If $y = 1$ if the prediction $f(x) = 1$ then cost = 0
If $y = 1$ if the prediction $f(x) = 0$ then cost $\rightarrow \infty$
2. If $y = 0$ if the prediction $f(x) = 0$ then cost $\rightarrow 0$
If $y = 0$ if the prediction $f(x) = 1$ then cost = ∞



Case 1



Case 2

Logistic Regression

Nice convex functions!

Let's combine them in a compact function (because $y = 0$ or $y = 1$):

$$Loss(f(x), y) = -y \log f(x) - (1 - y) \log(1 - f(x))$$

$$R(\beta) = -\frac{1}{m} \left[\sum_{i=1}^m y \log f(x) + (1 - y) \log(1 - f(x)) \right]$$

Gradient Descent

Repeat {

 Simultaneously update for all β 's

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} R(\beta)$$

}

After some calculus:

Repeat {

 Simultaneously update for all β 's

$$\beta_j := \beta_j - \alpha \sum_{i=1}^m (f(x) - y)x_j$$

}

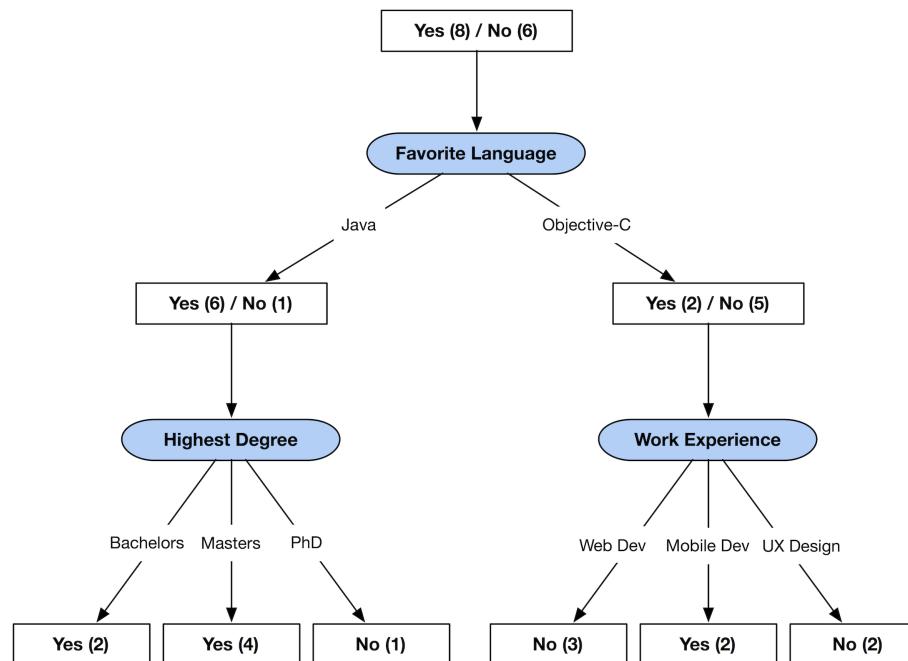
Note: Same as linear regression BUT with the new function f .

Credit

- When mentioned, some of the figures in this presentation are taken from “An Introduction to Statistical Learning, with applications in R” (Springer, 2013)” with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Artificial Intelligence

Machine Learning Tree classifiers



Outline

1. Tree classifiers: Definition & History
2. A toy example
3. Splitting criteria in C4.5
4. Numerical features
5. Pruning strategies
6. Alternative method: CART
7. Practical considerations
8. Tree classifiers: Pros & cons

Tree Classifiers

- Popular classification methods.
- Easy to understand, simple algorithmic approach.
- No assumption about linearity.
- **History:**
 - CART (Classification And Regression Trees): Friedman 1977.
 - ID3 and C4.5 family: Quilan 1979-1983.
 - Refinements in mid 1990's (e.g., pruning, numerical features etc.).
- **Applications:**
 - Botany (e.g., New Flora of the British Isles Stace 1991).
 - Medical research (e.g., Pima Indian diabetes diagnosis, early diagnosis of acute myocardial infarction).
 - Computational biology (e.g., interaction between genes)

Tree Classifiers

- The terminology **Tree** is graphic.
- However, a decision tree is grown from the root downward. The idea is to send the examples down the tree, using the concept of information entropy.

Tree Classifiers

- The terminology **Tree** is graphic.
- However, a decision tree is grown from the root downward. The idea is to send the examples down the tree, using the concept of information entropy.
- **General Steps to build a tree:**
 1. Start with the root node that has all the examples.
 2. Greedy selection of the next best feature to build the branches. The splitting criteria is *node purity*.
 3. Class majority will be assigned to the leaves.

Classification

Given: Training data:

$$(x_1, y_1), \dots, (x_n, y_n)$$

Where $x_i \in \mathbb{R}^d$ and y_i is discrete (categorical/qualitative), $y_i \in \mathbb{Y}$.

Example $\mathbb{Y} = \{-1, +1\}$, $\mathbb{Y} = \{0, 1\}$.

Task: Learn a classification function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{Y}$$

Classification

Given: Training data:

$$(x_1, y_1), \dots, (x_n, y_n)$$

Where $x_i \in \mathbb{R}^d$ and y_i is discrete (categorical/qualitative), $y_i \in \mathbb{Y}$.

Example $\mathbb{Y} = \{-1, +1\}$, $\mathbb{Y} = \{0, 1\}$.

Task: Learn a classification function:

$$f : \mathbb{R}^d \longrightarrow \mathbb{Y}$$

In the case of Tree Classifiers:

1. No need for $x_i \in \mathbb{R}^d$, so no need to turn categorical features into numerical features.
2. The model is a tree.

Toy example

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Bachelors	Mobile Dev	Objective-C	TRUE	yes
Masters	Web Dev	Java	FALSE	yes
Masters	Mobile Dev	Java	TRUE	yes
PhD	Mobile Dev	Objective-C	TRUE	yes
PhD	Web Dev	Objective-C	TRUE	no
Bachelors	UX Design	Objective-C	TRUE	no
Bachelors	Mobile Dev	Java	FALSE	yes
PhD	Web Dev	Objective-C	FALSE	no
Bachelors	UX Design	Java	FALSE	yes
Masters	UX Design	Objective-C	TRUE	no
Masters	UX Design	Java	FALSE	yes
PhD	Mobile Dev	Java	FALSE	no
Masters	Mobile Dev	Java	TRUE	yes
Bachelors	Web Dev	Objective-C	FALSE	no

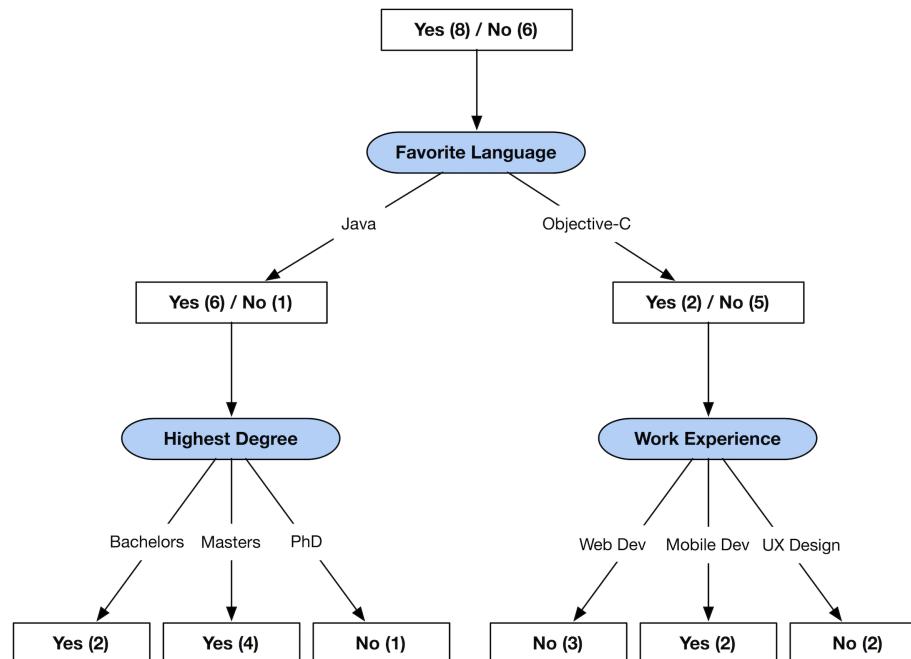
Toy example

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Bachelors	Mobile Dev	Objective-C	TRUE	yes
Masters	Web Dev	Java	FALSE	yes
Masters	Mobile Dev	Java	TRUE	yes
PhD	Mobile Dev	Objective-C	TRUE	yes
PhD	Web Dev	Objective-C	TRUE	no
Bachelors	UX Design	Objective-C	TRUE	no
Bachelors	Mobile Dev	Java	FALSE	yes
PhD	Web Dev	Objective-C	FALSE	no
Bachelors	UX Design	Java	FALSE	yes
Masters	UX Design	Objective-C	TRUE	no
Masters	UX Design	Java	FALSE	yes
PhD	Mobile Dev	Java	FALSE	no
Masters	Mobile Dev	Java	TRUE	yes
Bachelors	Web Dev	Objective-C	FALSE	no

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Masters	UX Design	Java	TRUE	?

Toy example

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Bachelors	Mobile Dev	Objective-C	TRUE	yes
Masters	Web Dev	Java	FALSE	yes
Masters	Mobile Dev	Java	TRUE	yes
PhD	Mobile Dev	Objective-C	TRUE	yes
PhD	Web Dev	Objective-C	TRUE	no
Bachelors	UX Design	Objective-C	TRUE	no
Bachelors	Mobile Dev	Java	FALSE	yes
PhD	Web Dev	Objective-C	FALSE	no
Bachelors	UX Design	Java	FALSE	yes
Masters	UX Design	Objective-C	TRUE	no
Masters	UX Design	Java	FALSE	yes
PhD	Mobile Dev	Java	FALSE	no
Masters	Mobile Dev	Java	TRUE	yes
Bachelors	Web Dev	Objective-C	FALSE	no



Splitting criteria in C4.5

1. The central choice is selecting the next attribute to split on.
2. We want some criteria that measures the **homogeneity** or impurity of examples in the nodes:

Splitting criteria in C4.5

1. The central choice is selecting the next attribute to split on.
2. We want some criteria that measures the **homogeneity or impurity of examples in the nodes**:
 - (a) Quantify the mix of classes at each node.
 - (b) Maximum if equal number of examples from each class.
 - (c) Minimum if the node is pure.

Splitting criteria in C4.5

1. The central choice is selecting the next attribute to split on.
2. We want some criteria that measures the **homogeneity or impurity of examples in the nodes**:
 - (a) Quantify the mix of classes at each node.
 - (b) Maximum if equal number of examples from each class.
 - (c) Minimum if the node is pure.
3. A perfect measure commonly used in *Information Theory*:

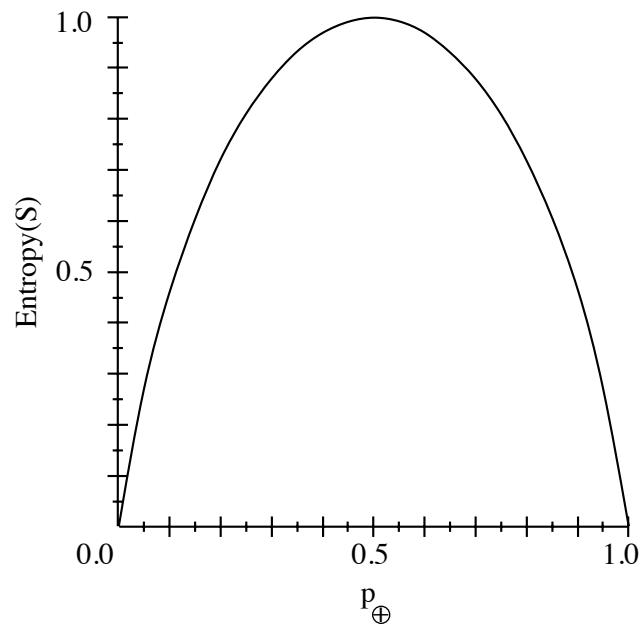
$$\text{Entropy}(S) = - p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

p_{\oplus} is the proportion of positive examples.

p_{\ominus} is the proportion of negative examples.

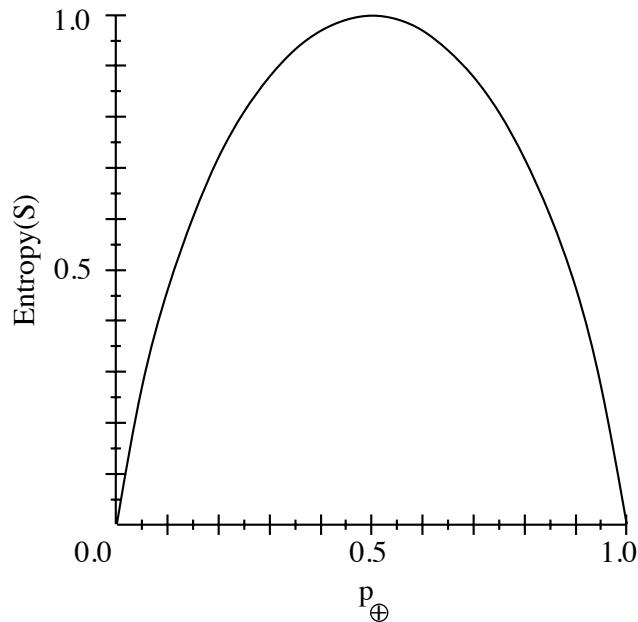
Splitting criteria in C4.5

$$\text{Entropy}(S) = - p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$



Splitting criteria in C4.5

$$\text{Entropy}(S) = - p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$



In general, for c classes:

$$\text{Entropy}(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Splitting Criteria in C4.5

- Now each node has some entropy that measures the homogeneity in the node.
- How to decide which attribute is best to split on based on entropy?

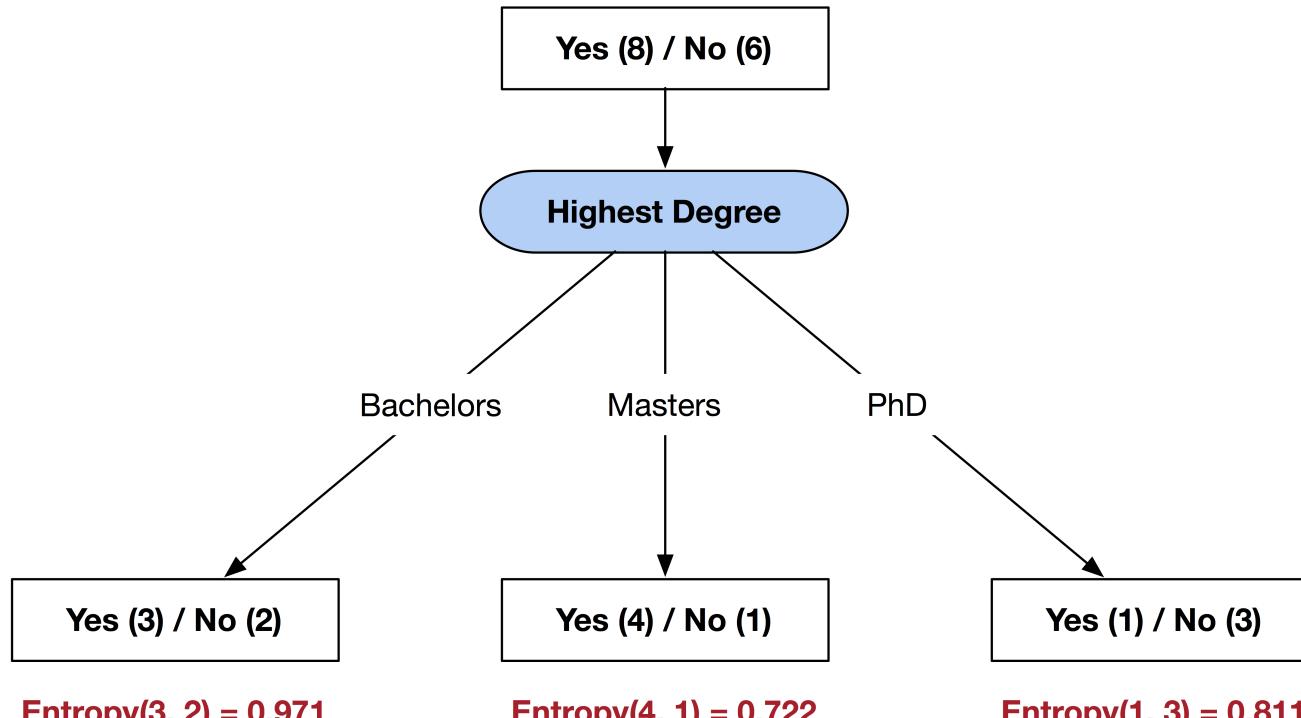
Splitting Criteria in C4.5

- Now each node has some entropy that measures the homogeneity in the node.
- How to decide which attribute is best to split on based on entropy?
- We use **Information Gain** that measures the expected reduction in entropy caused by partitioning the examples according to the attributes:

$$Gain(S, A) = Entropy(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Back to the example

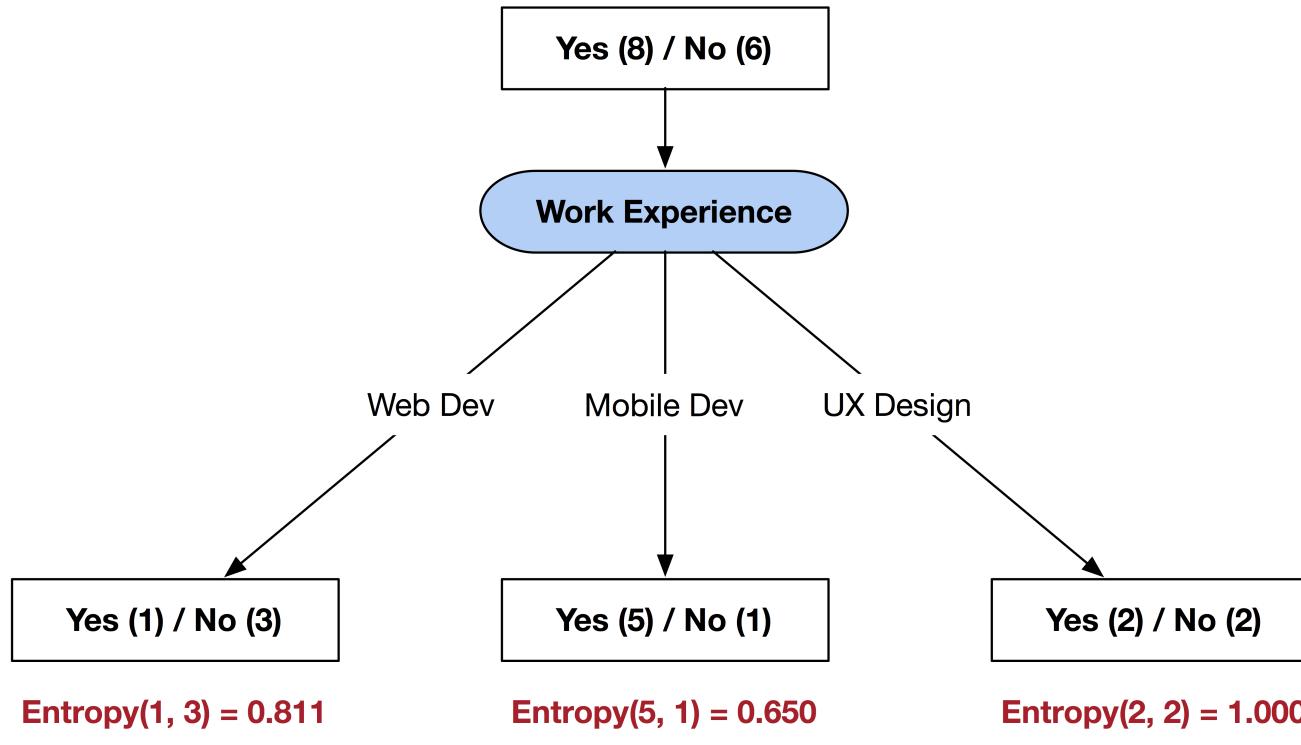
$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



$$\text{Gain}(S, \text{Highest Degree}) = 0.985 - (5/14) \times 0.971 - (5/14) \times 0.722 - (4/14) \times 0.811 = 0.149$$

Back to the example

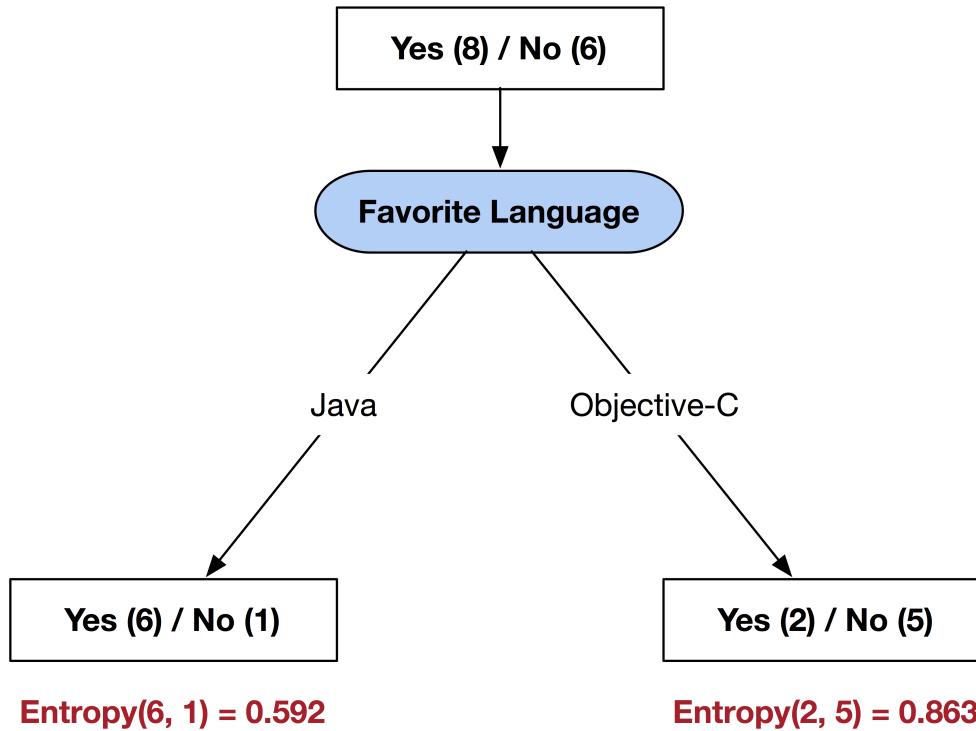
$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



$$\text{Gain}(S, \text{Work Experience}) = 0.985 - (4/14) \times 0.811 - (6/14) \times 0.650 - (4/14) \times 1.000 = 0.189$$

Back to the example

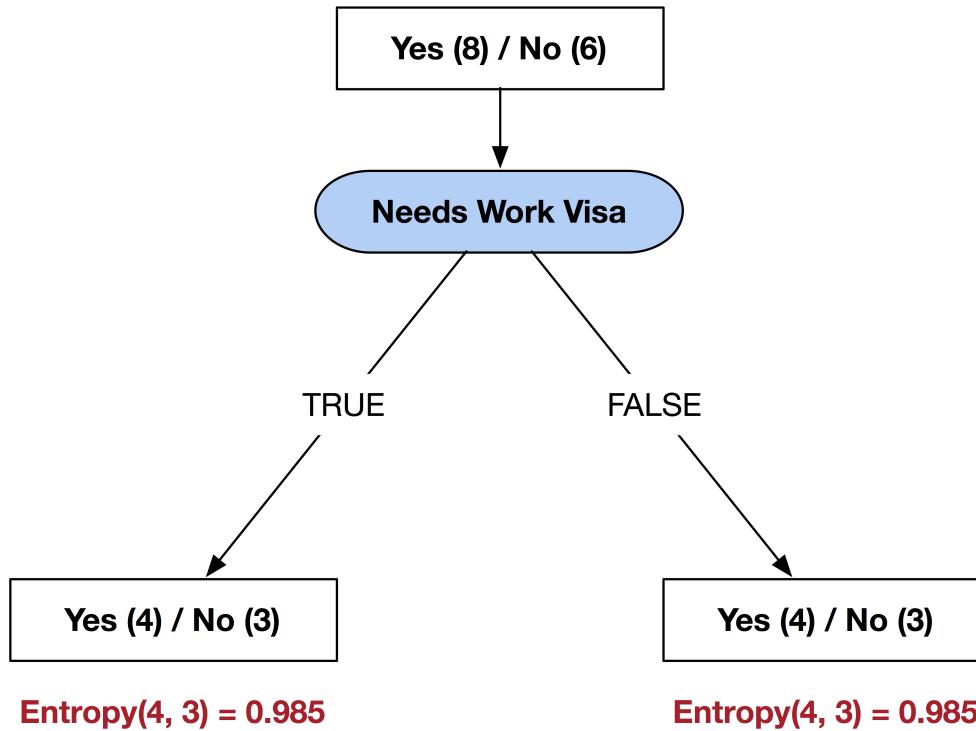
$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



$$\text{Gain}(S, \text{Favorite Language}) = 0.985 - (7/14) \times 0.592 - (7/14) \times 0.863 = 0.258$$

Back to the example

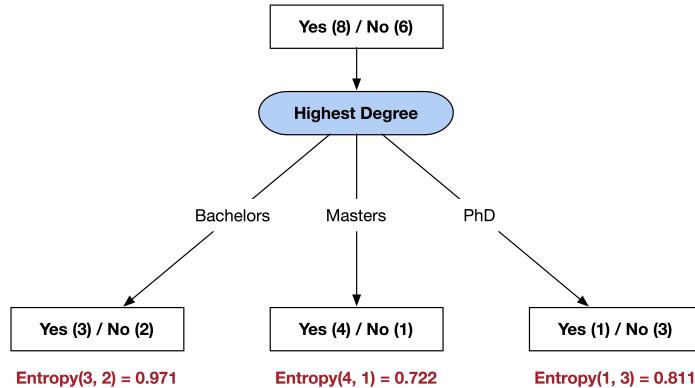
$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



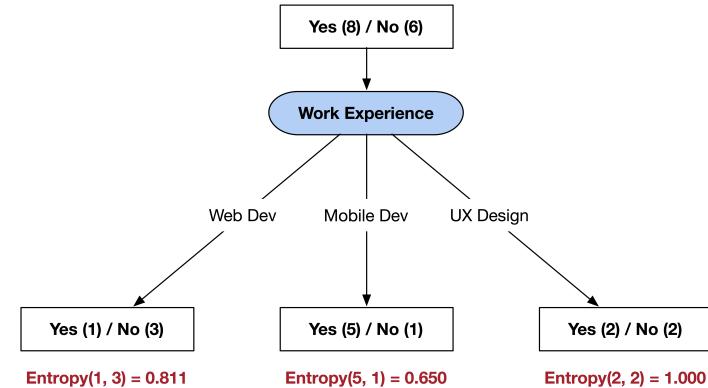
$$\text{Gain}(S, \text{Needs Work Visa}) = 0.985 - (7/14) \times 0.985 - (7/14) \times 0.985 = 0.000$$

Back to the example

$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



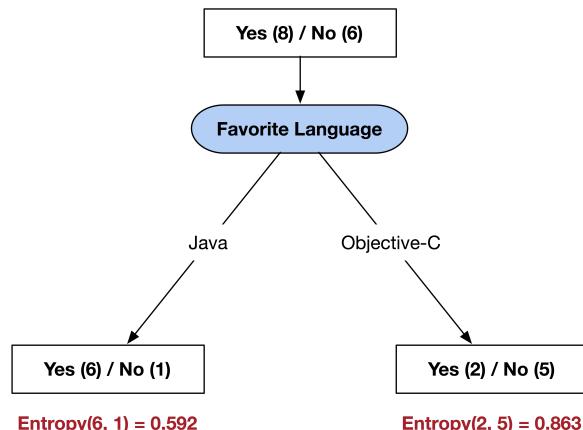
$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



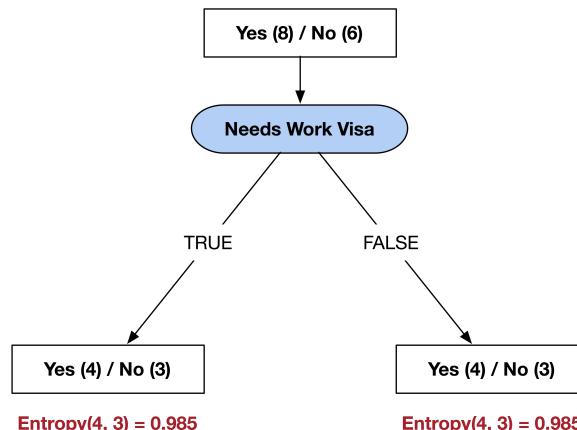
$$\text{Gain}(S, \text{Highest Degree}) = 0.985 - (5/14) \times 0.971 - (5/14) \times 0.722 - (4/14) \times 0.811 = 0.149$$

$$\text{Gain}(S, \text{Work Experience}) = 0.985 - (4/14) \times 0.811 - (6/14) \times 0.650 - (4/14) \times 1.000 = 0.189$$

$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



$$\text{Entropy}(8, 6) = - (8/14) \times \log(8/14) - (6/14) \times \log(6/14) = 0.985$$



$$\text{Gain}(S, \text{Favorite Language}) = 0.985 - (7/14) \times 0.592 - (7/14) \times 0.863 = 0.258$$

$$\text{Gain}(S, \text{Needs Work Visa}) = 0.985 - (7/14) \times 0.985 - (7/14) \times 0.985 = 0.000$$

Back to the example

Feature	Information Gain
Highest Degree	0.149
Work Experience	0.189
Favorite Language	0.258
Needs Work Visa	0.000

At the first split starting from the root, we choose the attribute that has the max gain.

Then, we re-start the same process at each of the children nodes (if node not pure).

Numerical features

Papers Published	Years of Work	Grade Point Average	Needs Work Visa	Hire
0 paper(s)	5 year(s)	3.20	TRUE	yes
5 paper(s)	1 year(s)	3.64	FALSE	yes
4 paper(s)	6 year(s)	2.92	TRUE	yes
10 paper(s)	4 year(s)	4.00	TRUE	yes
12 paper(s)	3 year(s)	3.21	TRUE	no
0 paper(s)	8 year(s)	3.37	TRUE	no
0 paper(s)	5 year(s)	4.00	FALSE	yes
8 paper(s)	3 year(s)	2.59	FALSE	no
0 paper(s)	7 year(s)	3.70	FALSE	yes
4 paper(s)	7 year(s)	3.78	TRUE	no
2 paper(s)	9 year(s)	4.00	FALSE	yes
9 paper(s)	4 year(s)	4.00	FALSE	no
7 paper(s)	4 year(s)	2.71	TRUE	yes
0 paper(s)	2 year(s)	3.03	FALSE	no

Overfitting the data

Pruning strategies

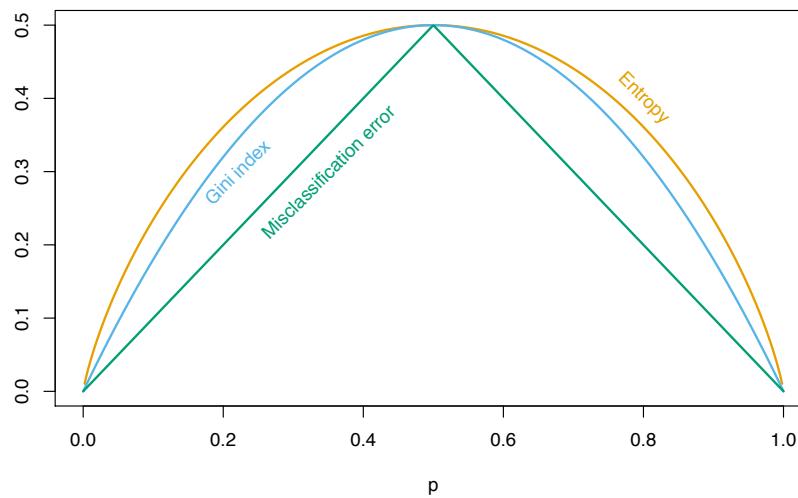
To get suitable tree sizes and avoid overfitting:

- Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training examples. (difficult to know when to stop!).
- Grow a complex tree then to prune it back (Best strategy found).
 1. Use a validation set / Cross validation to evaluate the utility of post-pruning (remove a subtree if the performance of the new tree is no worse than the original tree).
 2. Rule post pruning.

CART

- Adopt same greedy, top-down algorithm.
- Binary splits instead of multiway splits.
- Uses Gini Index instead of information entropy.

$$Gini = 1 - p_{\oplus}^2 - P_{\ominus}^2$$



Practical considerations

1. Consider performing dimensionality reduction beforehand to keep the most discriminative features.
2. Use ensemble methods. E.g., Random Forest, have a great performance.*
3. Balance your dataset before training to prevent the tree from creating a tree biased toward the classes that are dominant.
 - Under-sampling: reduce the majority class
 - Over-sampling: Synthetic data generation for the minority class (e.g., SMOTE, and ADASYN).

**An Empirical Comparison of Supervised Learning Algorithms Rich by Caruana and Alexandru Niculescu-Mizil. ICML 2006.*

Tree classifiers: Pros & Cons

- + Intuitive, interpretable (but...).
 - + Can be turned into rules.
 - + Well-suited for categorical data.
 - + Simple to build.
 - + No need to scale the data.
-
- Unstable (change in an example may lead to a different tree).
 - Univariate (split one attribute at a time, does not combine features).
 - A choice at some node depends on the previous choices.
 - Need to balance the data.

Credit

- The elements of statistical learning. Data mining, inference, and prediction. 10th Edition 2009. T. Hastie, R. Tibshirani, J. Friedman.
- Machine Learning 1997. Tom Mitchell.

Artificial Intelligence

Machine Learning

Naive Bayes

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

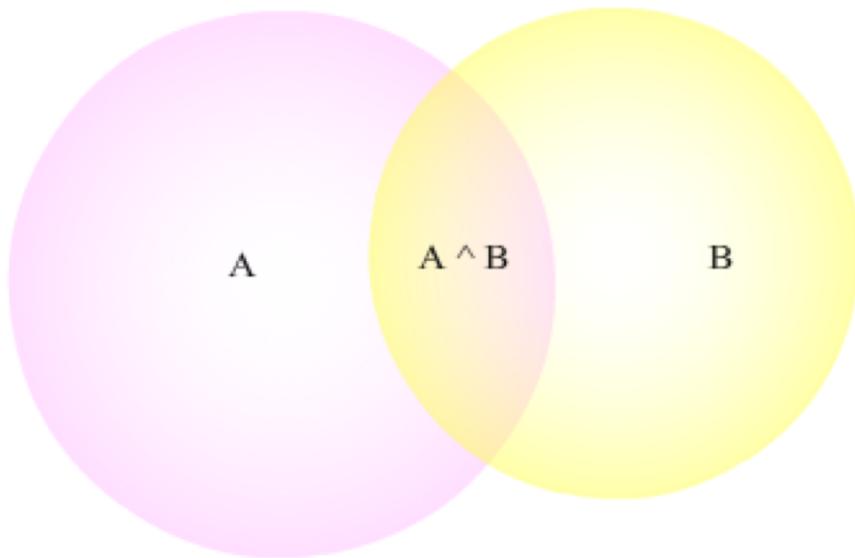
Diagram illustrating the components of the Naive Bayes formula:

- $P(A|B)$ is labeled "Posterior" with a downward arrow.
- $P(B|A)$ is labeled "Likelihood" with a leftward arrow.
- $P(A)$ is labeled "Prior" with a leftward arrow.
- $P(B)$ is labeled "evidence" with a curved arrow pointing to it.

Outline

1. Generative models
2. Naive Bayes Classifier.
3. Setting
4. Example
5. Estimating probabilities

Conditional Probability



$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

$$p(A \cap B) = p(A|B) * p(B)$$

Bayes Rule

Writing $p(A \wedge B)$ in two different ways:

$$p(A \wedge B) = p(B|A) * p(A)$$

$$p(A \wedge B) = p(A|B) * p(B)$$

Bayes Rule

Writing $p(A \wedge B)$ in two different ways:

$$p(A \wedge B) = p(B|A) * p(A)$$

$$p(A \wedge B) = p(A|B) * p(B)$$

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)}$$

Bayes Rule

Writing $p(A \wedge B)$ in two different ways:

$$p(A \wedge B) = p(B|A) * p(A)$$

$$p(A \wedge B) = p(A|B) * p(B)$$

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)}$$

$p(A|B)$ is called posterior (posterior distribution on A given B .)

$p(A)$ is called prior.

$p(B)$ is called evidence.

$p(B|A)$ is called likelihood.

Bayes Rule

	A	not A	Sum
B	$P(A \text{ and } B)$	$P(\text{not } A \text{ and } B)$	$P(B)$
Not B	$P(A \text{ and not } B)$	$P(\text{not } A \text{ and not } B)$	$P(\text{not } B)$
	$P(A)$	$P(\text{not } A)$	1

- This table divides the sample space into 4 mutually exclusive events.
- The probability in the margins are called marginals and are calculated by summing across the rows and columns.

Bayes Rule

	A	not A	Sum
B	P(A and B)	P(not A and B)	P(B)
Not B	P(A and not B)	P(not A and not B)	P(not B)
	P(A)	P(not A)	1

- This table divides the sample space into 4 mutually exclusive events.
- The probability in the margins are called marginals and are calculated by summing across the rows and columns.

Another form:

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

Example of Using Bayes Rule

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

A: patient has cancer.

B: patient has a positive lab test.

Example of Using Bayes Rule

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

A: patient has cancer.

B: patient has a positive lab test.

$$p(A) = 0.008$$

$$p(\neg A) = 0.992$$

Example of Using Bayes Rule

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

A: patient has cancer.

B: patient has a positive lab test.

$$p(A) = 0.008$$

$$p(B|A) = 0.98$$

$$p(\neg A) = 0.992$$

$$p(\neg B|\neg A) = 0.02$$

Example of Using Bayes Rule

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

A: patient has cancer.

B: patient has a positive lab test.

$$p(A) = 0.008$$

$$p(B|A) = 0.98$$

$$p(B|\neg A) = 0.03$$

$$p(\neg A) = 0.992$$

$$p(\neg B|A) = 0.02$$

$$p(\neg B|\neg A) = 0.97$$

Example of Using Bayes Rule

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B|A) * p(A) + p(B|\neg A) * p(\neg A)}$$

A: patient has cancer.

B: patient has a positive lab test.

$$p(A) = 0.008$$

$$p(B|A) = 0.98$$

$$p(B|\neg A) = 0.03$$

$$p(\neg A) = 0.992$$

$$p(\neg B|A) = 0.02$$

$$p(\neg B|\neg A) = 0.97$$

$$p(A|B) = \frac{0.98 * 0.008}{0.98 * 0.008 + 0.02 * 0.992} = 0.21$$

Why probabilities?

Why are we bringing here Bayes rule?

Why probabilities?

Why are we bringing here Bayes rule?

Recall Classification framework:

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{Y}$.

Task: Learn a classification function: $f : \mathbb{R}^d \rightarrow \mathbb{Y}$

Why probabilities?

Why are we bringing here a Bayesian framework?

Recall Classification framework:

Given: Training data: $(x_1, y_1), \dots, (x_n, y_n)$ / $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{Y}$.

Task: Learn a classification function: $f : \mathbb{R}^d \rightarrow \mathbb{Y}$

Learn a mapping from x to y .

We would like to find this mapping $f(x) = y$ through $p(y|x)$!

Discriminative Algorithms

- **Discriminative Algorithms:**
 - Idea: model $p(y|x)$, conditional distribution of y given x .
 - In Discriminative Algorithms: find a decision boundary that separates positive from negative example.
 - To predict a new example, check on which side of the decision boundary it falls.
 - Model $p(y|x)$ directly.

Generative Algorithms

- **Generative Algorithms** adopt a different approach:
 - Idea: Build a model for what positive examples look like.
Build a different model for what negative example look like.
 - To predict a new example, match it with each of the models
and see which match is best.
 - Model $p(x|y)$ and $p(y)$!
 - Use Bayes rule to obtain $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$.

Generative Algorithms

- **Generative Algorithms** adopt a different approach:
 - Idea: Build a model for what positive examples look like.
Build a different model for what negative example look like.
 - To predict a new example, match it with each of the models
and see which match is best.
 - Model $p(x|y)$ and $p(y)$!
 - Use Bayes rule to obtain $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$.
 - To make a prediction:

$$\operatorname{argmax}_y p(y|x) = \operatorname{argmax}_y \frac{p(x|y)p(y)}{p(x)}$$

$$\operatorname{argmax}_y p(y|x) \approx \operatorname{argmax}_y p(x|y)p(y)$$

Naive Bayes Classifier

- Probabilistic model.
- Highly practical method.
- Application domains to natural language text documents.
- Naive because of the strong independence assumption it makes (not realistic).
- Simple model.
- Strong method can be comparable to decision trees and neural networks in some cases.

Setting

- A training data (x_i, y_i) , x_i is a feature vector and y_i is a discrete label.
- d features, and n examples.
- Example: consider document classification, each example is a documents, each feature represents the presence or absence of a particular word in the document.
- We have a training set.
- A new example with feature values $x_{new} = (a_1, a_2, \dots, a_d)$.
- We want to predict the label y_{new} of the new example.

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Use Bayes rule to obtain:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} \frac{p(a_1, a_2, \dots, a_d|y) * p(y)}{p(a_1, a_2, \dots, a_d)}$$

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Use Bayes rule to obtain:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} \frac{p(a_1, a_2, \dots, a_d|y) * p(y)}{p(a_1, a_2, \dots, a_d)}$$

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(a_1, a_2, \dots, a_d|y) * p(y)$$

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Use Bayes rule to obtain:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} \frac{p(a_1, a_2, \dots, a_d|y) * p(y)}{p(a_1, a_2, \dots, a_d)}$$

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(\textcolor{red}{a_1, a_2, \dots, a_d|y}) * \textcolor{blue}{p(y)}$$

Can we estimate these two terms from the training data?

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Use Bayes rule to obtain:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} \frac{p(a_1, a_2, \dots, a_d|y) * p(y)}{p(a_1, a_2, \dots, a_d)}$$

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(a_1, a_2, \dots, a_d|y) * p(y)$$

Can we estimate these two terms from the training data?

1. $p(y)$ can be easy to estimate: count the frequency with which each label y occurs in the training data.

Setting

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y|a_1, a_2, \dots, a_d)$$

Use Bayes rule to obtain:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} \frac{p(a_1, a_2, \dots, a_d|y) * p(y)}{p(a_1, a_2, \dots, a_d)}$$

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(a_1, a_2, \dots, a_d|y) * p(y)$$

Can we estimate these two terms from the training data?

1. $p(y)$ can be easy to estimate: count the frequency with which each label y .
2. $p(a_1, a_2, \dots, a_d|y)$ is not easy to estimate unless we have a very very large sample. (We need to see every example many times to get reliable estimates)

Naive Bayes Classifier

Makes a simplifying assumption that the feature values are conditionally independent given the label.

Given the label of the example, the probability of observing the conjunction a_1, a_2, \dots, a_d is the product of the probabilities for the individual features:

$$p(a_1, a_2, \dots, a_d | y) = \prod_j p(a_j | y)$$

Naive Bayes Classifier

Makes a simplifying assumption that the feature values are conditionally independent given the label.

Given the label of the example, the probability of observing the conjunction a_1, a_2, \dots, a_d is the product of the probabilities for the individual features:

$$p(a_1, a_2, \dots, a_d | y) = \prod_j p(a_j | y)$$

Naive Bayes Classifier:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y) \prod_j p(a_j | y)$$

Naive Bayes Classifier

Makes a simplifying assumption that the feature values are conditionally independent given the label.

Given the label of the example, the probability of observing the conjunction a_1, a_2, \dots, a_d is the product of the probabilities for the individual features:

$$p(a_1, a_2, \dots, a_d | y) = \prod_j p(a_j | y)$$

Naive Bayes Classifier:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y) \prod_j p(a_j | y)$$

Can we estimate these two terms from the training data?

Naive Bayes Classifier

Makes a simplifying assumption that the feature values are conditionally independent given the label.

Given the label of the example, the probability of observing the conjunction a_1, a_2, \dots, a_d is the product of the probabilities for the individual features:

$$p(a_1, a_2, \dots, a_d | y) = \prod_j p(a_j | y)$$

Naive Bayes Classifier:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y) \prod_j p(a_j | y)$$

Can we estimate these two terms from the training data?

Yes!

Algorithm

Learning: Based on the frequency counts in the dataset:

1. Estimate all $p(y)$, $\forall y \in \mathbb{Y}$.
2. Estimate all $p(a_j|y)$ $\forall y \in \mathbb{Y}$, $\forall a_i$.

Classification: For a new example, use:

$$y_{new} = \operatorname{argmax}_{y \in \mathbb{Y}} p(y) \prod_j p(a_j|y)$$

Note: No model per se or hyperplane, just count the frequencies of various data combinations within the training examples.

Example

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Bachelors	Mobile Dev	Objective-C	TRUE	yes
Masters	Web Dev	Java	FALSE	yes
Masters	Mobile Dev	Java	TRUE	yes
PhD	Mobile Dev	Objective-C	TRUE	yes
PhD	Web Dev	Objective-C	TRUE	no
Bachelors	UX Design	Objective-C	TRUE	no
Bachelors	Mobile Dev	Java	FALSE	yes
PhD	Web Dev	Objective-C	FALSE	no
Bachelors	UX Design	Java	FALSE	yes
Masters	UX Design	Objective-C	TRUE	no
Masters	UX Design	Java	FALSE	yes
PhD	Mobile Dev	Java	FALSE	no
Masters	Mobile Dev	Java	TRUE	yes
Bachelors	Web Dev	Objective-C	FALSE	no

Highest Degree	Work Experience	Favorite Language	Needs Work Visa	Hire
Masters	UX Design	Java	TRUE	?

Can we predict the class of the new example?

Example

$$y_{new} = \operatorname{argmax}_{y \in \{yes, no\}} p(y) * p(Masters|y) * p(UX\ Design|y) * p(Java|y) * p(TRUE|y)$$

$$p(yes) = 8/14 = 0.572$$

$$p(no) = 6/14 = 0.428$$

Conditional probabilities:

$$p(masters|yes) = 4/8 \quad p(masters|no) = 1/6$$

$$p(UX\ Design|yes) = 2/8 \quad p(UX\ Design|no) = 2/6$$

$$p(Java|yes) = 6/8 \quad p(Java|no) = 1/6$$

$$p(TRUE|yes) = 4/8 \quad p(TRUE|no) = 3/6$$

$$p(yes) * p(Masters|yes) * p(UX\ Design|yes) * p(Java|yes) * p(TRUE|yes) = 0.026$$

$$p(no) * p(Masters|no) * p(UX\ Design|no) * p(Java|no) * p(TRUE|no) = 0.002$$

$$y_{new} = yes$$

Estimating probabilities

m-estimate of the probability:

$$p(a_j|y) = \frac{n_c + m * p}{n_y + m}$$

where:

n_y : total number of examples for which the class is y .

n_c : total number of examples for which the class is y and feature $x_j = a_j$.

m : called *equivalent sample size*

Estimating probabilities

m-estimate of the probability:

$$p(a_j|y) = \frac{n_c + m * p}{n_y + m}$$

where:

n_y : total number of examples for which the class is y .

n_c : total number of examples for which the class is y and feature $x_j = a_j$.

m : called *equivalent sample size*

Intuition:

Augment the sample size by m virtual examples, distributed according to prior p (prior estimate of each value).

If prior is unknown, assume uniform prior: if a feature has k values, we can set $p = \frac{1}{k}$.

Text Classification

Learning to classify text. Why?

- Learn which news articles are of interest
- Learn to classify web pages by topic
- Classify Spam from non Spam emails
- Naive Bayes is among most effective algorithms
- What attributes shall we use to represent text documents?

Text Classification

- Given a document (corpus), define an attribute for each word position in the document.
- The value of the attribute is the English word in that position.
- To reduce the number of probabilities that needs to be estimated, besides NB independence assumption, we assume that: The probability of a given word w_k occurrence is independent of the word position within the text. That is:

$$p(x_1 = w_k | c_j), p(x_2 = w_k | c_j), \dots$$

estimated by:

$$p(w_k | c_j)$$

Text Classification

- m-estimate of the probabilities:

$$p(w_k|c_j) = \frac{n_k + 1}{n_j + |Vocabulary|}$$

where:

n_j : total #word positions in all training examples of class c_j .

n_k : number of times the word w_k is found in among these n_j word positions.

- The following function learns the probabilities $P(w_k/c_j)$ describing the probability that a randomly drawn word from a document with class c_j is the English word w_k . It also learn the class priors $P(c_j)$.

Naive Bayes for text

1. Naive Bayes is a linear classifier.
2. Incredibly simple and easy to implement.
3. Works wonderful for text.

Credit

- Machine Learning. Tom Mitchell 1997.

Artificial Intelligence

Machine Learning Ensemble Methods



Outline

- 1. Majority voting**
- 2. Boosting**
- 3. Bagging**
- 4. Random Forests**
- 5. Conclusion**

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = $0.5 - \epsilon$** .

Majority Voting

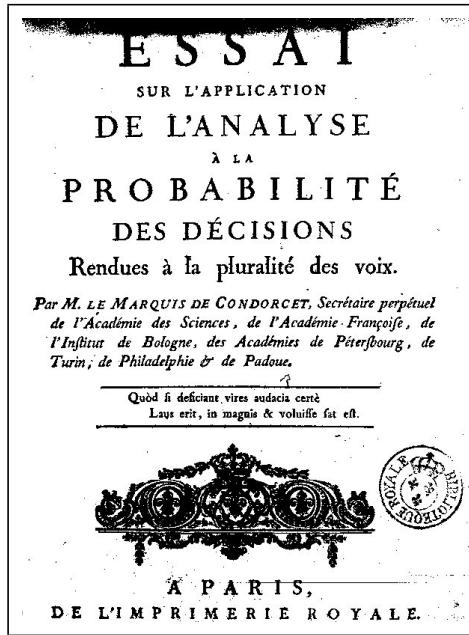
- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = $0.5-\epsilon$** .
- Combine: make a decision based on majority vote?

Majority Voting

- A randomly chosen hyperplane has an expected **error of 0.5**.
- Many random hyperplanes combined by **majority vote** will still be random.
- Suppose we have m classifiers, performing slightly better than random, that is **error = $0.5-\epsilon$** .
- Combine: make a decision based on majority vote?
- What if we combined these m **slightly-better-than-random** classifiers? Would majority vote be a good choice?

Condorcet's Jury Theorem

Marquis de Condorcet Application of Analysis to the Probability of Majority Decisions. 1785.

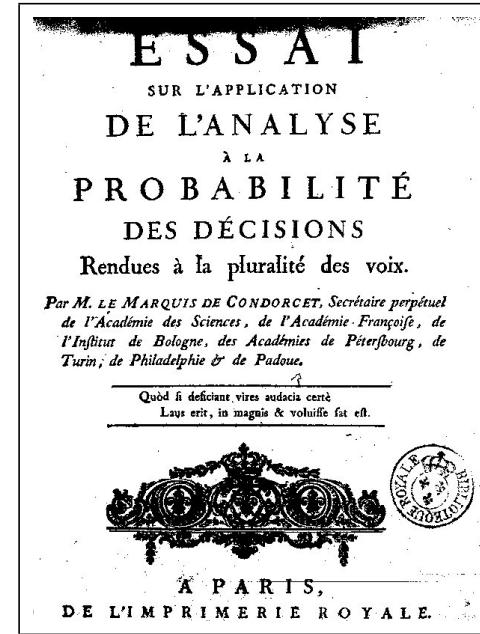


Assumptions:

1. Each individual makes the right choice with a probability p .
2. The votes are independent.

Condorcet's Jury Theorem

Marquis de Condorcet Application of Analysis to the Probability of Majority Decisions. 1785.



Assumptions:

1. Each individual makes the right choice with a probability p .
2. The votes are independent.

If $p > 0.5$, then adding more voters increases the probability that the majority decision is correct. if $p < 0.5$, then adding more voters makes things worse.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

**How do we produce independent weak learners
using the same training data?**

Ensemble Methods

- An **Ensemble Method** combines the predictions of many individual classifiers by majority voting.
- Such individual classifiers, called **weak learners**, are required to perform slightly better than random.

**How do we produce independent weak learners
using the same training data?**

- Use a **strategy** to obtain relatively independent weak learners!
- Different methods:
 1. Boosting
 2. Bagging
 3. Random Forests

Boosting

- First ensemble method.
- One of the most powerful Machine Learning methods.
- Popular algorithm: AdaBoost.M1 (Freund and Shapire 1997).
- “Best off-the-shelf classifier in the world” Breiman (CART’s inventor), 1998.
- Simple algorithm.
- Weak learners can be trees, perceptrons, decision stumps, etc.
- **Idea:**

Train the weak learners on weighted training examples.

Boosting

Boosting

- The predictions from all of the G_m , $m \in \{1, \dots, M\}$ are combined with a weighted majority voting.
- α_m is the contribution of each weak learner G_m .
- Computed by the boosting algorithm to give a weighted importance to the classifiers in the sequence.
- The decision of a highly-performing classifier in the sequence should weight more than less important classifiers in the sequence.
- This is captured in:

$$G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right]$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^n \mathbf{1}\{y_i \neq G(x_i)\}}{n}$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^n \mathbf{1}\{y_i \neq G(x_i)\}}{n}$$

The error rate on each weak learner:

$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

Boosting

The error rate on the training sample:

$$err := \frac{\sum_{i=1}^n \mathbf{1}\{y_i \neq G(x_i)\}}{n}$$

The error rate on each weak learner:

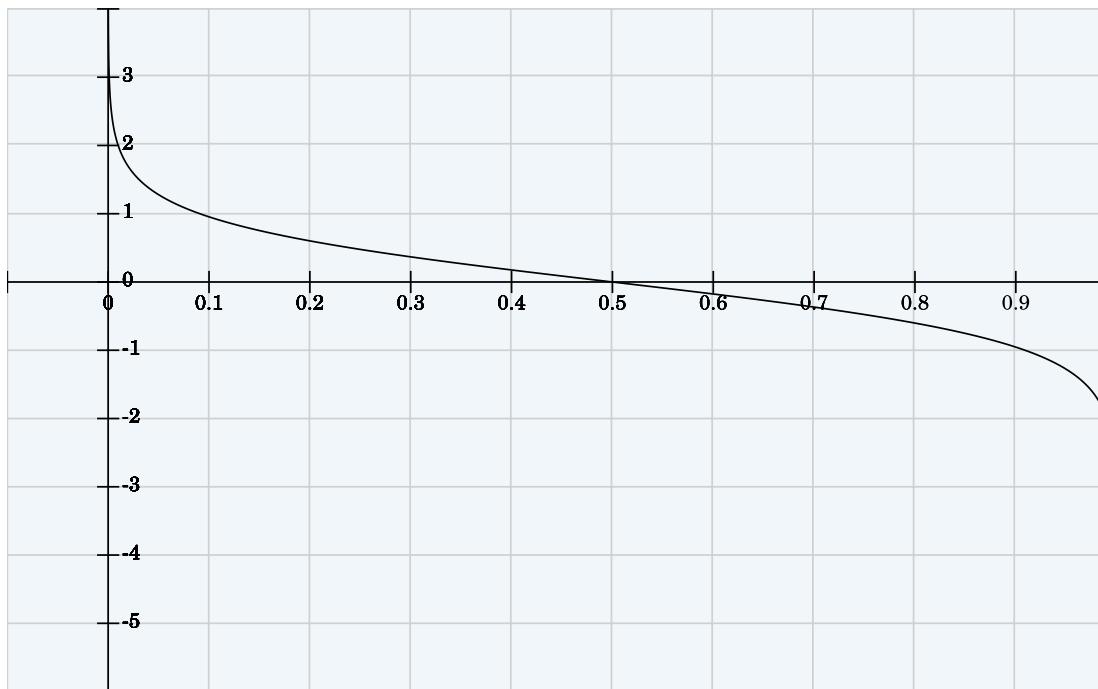
$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

Intuition:

- Give large weights for hard examples.
- Give small weights for easy examples.

Boosting

For each weak learner m , we associate an error err_m .



$$\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)
 - (a) Fit a classifier $G_m(x)$ to training data using the weights w_i .

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)
 - (a) Fit a classifier $G_m(x)$ to training data using the weights w_i .
 - (b) Compute

$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)
 - (a) Fit a classifier $G_m(x)$ to training data using the weights w_i .
 - (b) Compute

$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

- (c) Compute

$$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)
 - (a) Fit a classifier $G_m(x)$ to training data using the weights w_i .
 - (b) Compute

$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

- (c) Compute

$$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$$

- (d) $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot \mathbf{1}(y_i \neq G_m(x_i))]$ for $i = 1, \dots, n.$

AdaBoost

1. Initialize the example weights $w_i = \frac{1}{n}, i = 1, \dots, n.$
2. For $m = 1$ to M (number of weak learners)
 - (a) Fit a classifier $G_m(x)$ to training data using the weights w_i .
 - (b) Compute

$$err_m := \frac{\sum_{i=1}^n w_i \mathbf{1}\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^n w_i}$$

- (c) Compute

$$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right)$$

- (d) $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot \mathbf{1}(y_i \neq G_m(x_i))]$ for $i = 1, \dots, n.$

3. Output

$$G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x) \right]$$

Digression: Decision Stumps

This is an example of very weak classifier

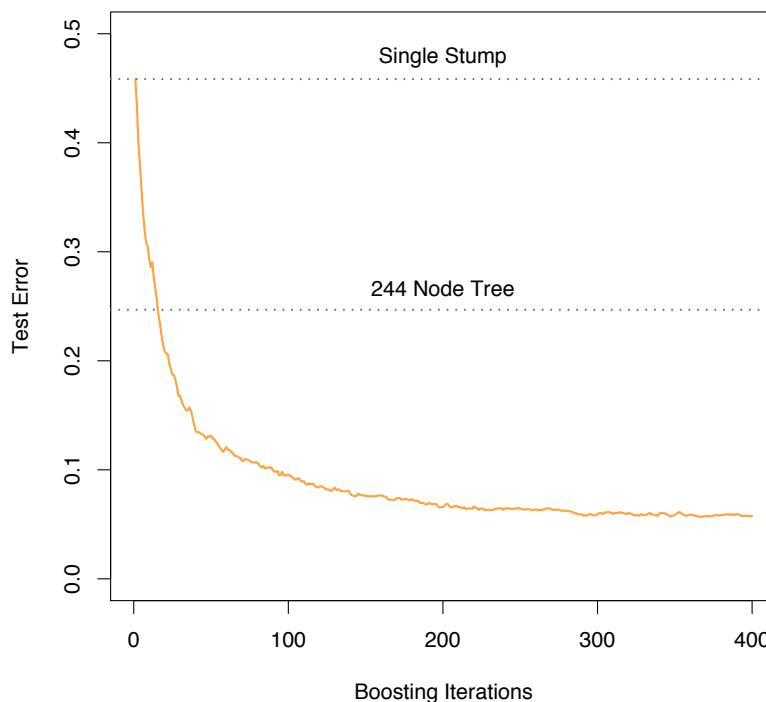
A simple 2-terminal node decision tree for binary classification.

$$f(x_i|j, t) = \begin{cases} +1 & \text{if } x_{ij} > t \\ -1 & \text{otherwise} \end{cases}$$

Where $j \in \{1, \dots, d\}$.

Example: A dataset with 10 features, 2,000 examples training and 10,000 testing.

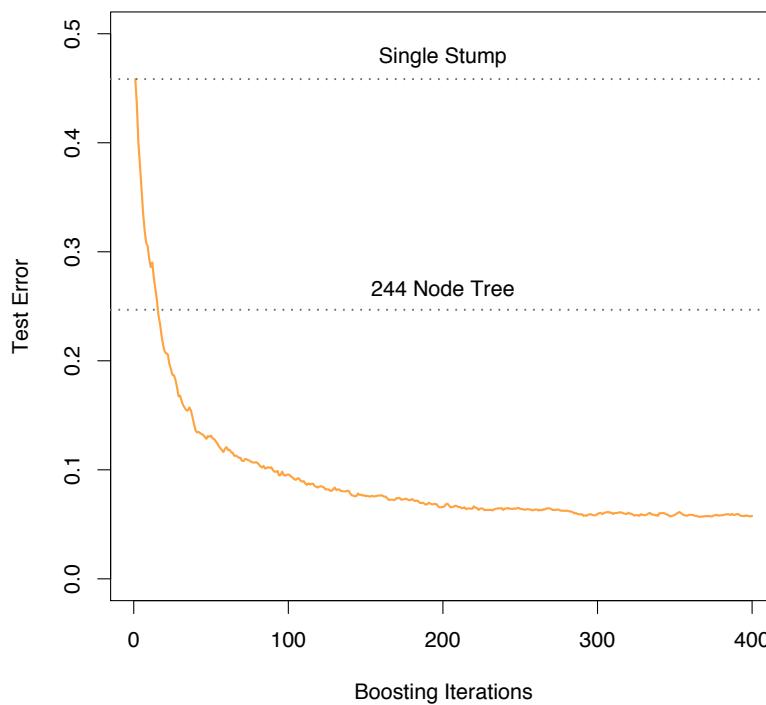
AdaBoost Performance



Error rates:

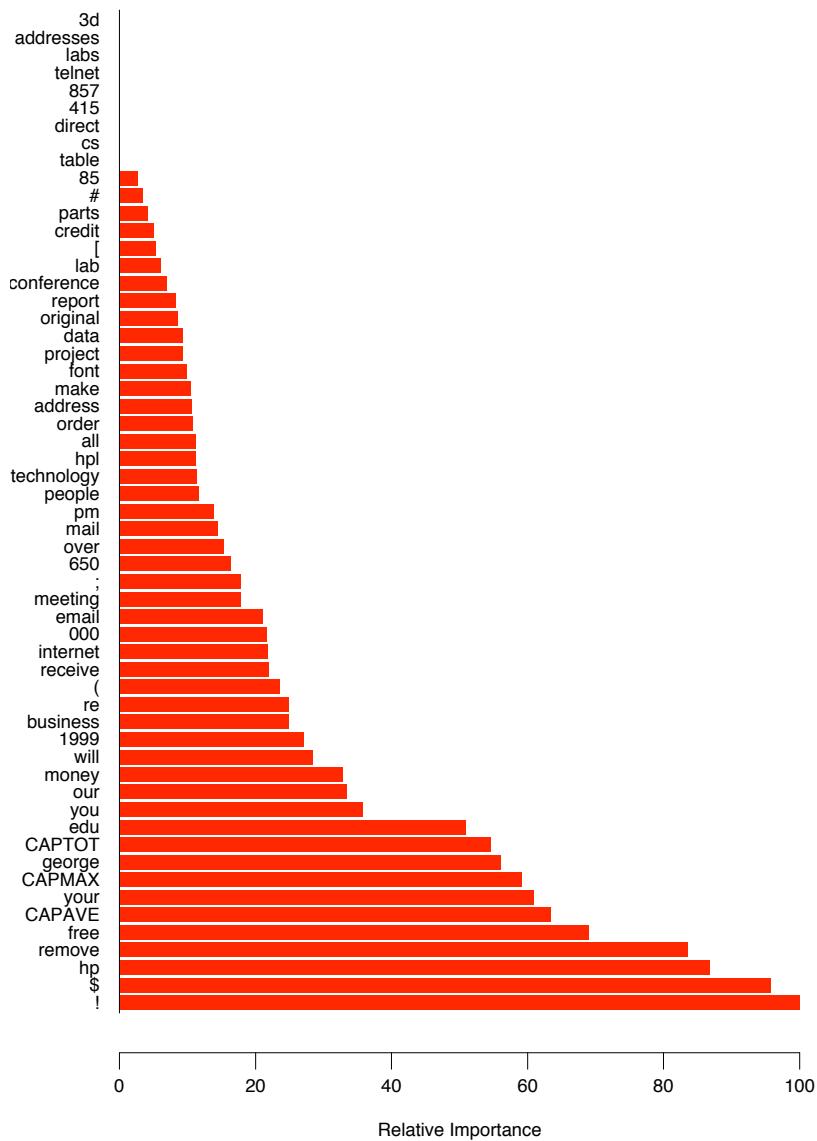
- Random: 50%.
- Stump: 45.8%.
- Large classification tree: 24.7%.
- AdaBoost with stumps: 5.8% after 400 iterations!

AdaBoost Performance



AdaBoost with Decision stumps lead to a form of:
feature selection

AdaBoost-Decision Stumps



Bagging & Bootstrapping

- Bootstrap is a re-sampling technique \equiv sampling from the empirical distribution.
- Aims to improve the quality of estimators.
- Bagging and Boosting are based on bootstrapping.
- Both use re-sampling to generate weak learners for classification.
- **Strategy:** Randomly distort data by re-sampling.
- Train weak learners on re-sampled training sets.
- **Bootstrap aggregation \equiv Bagging.**

Bagging

Training

For $b = 1, \dots, B$

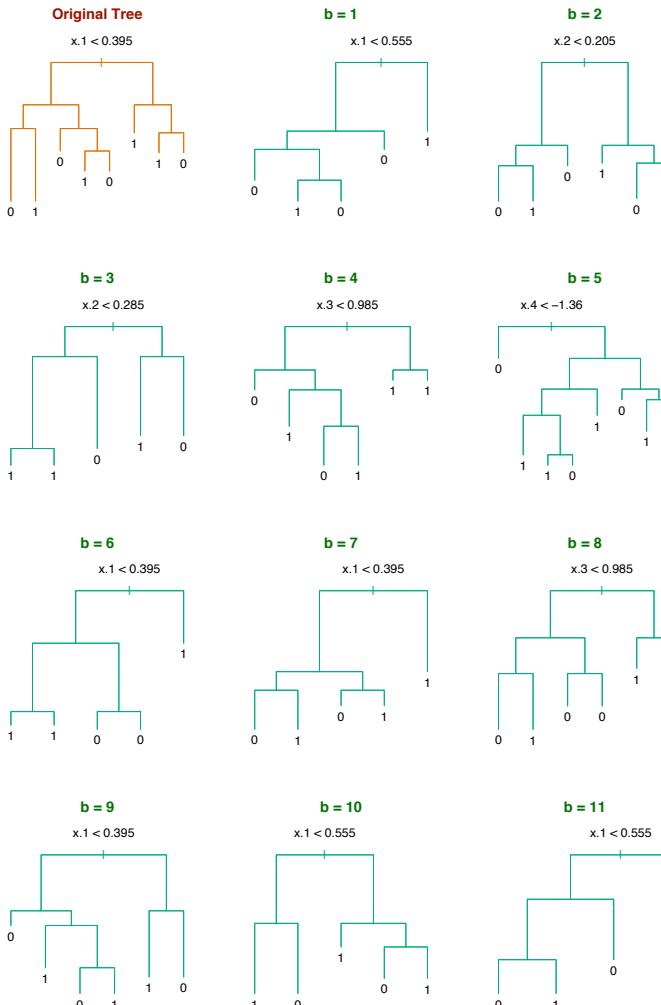
1. Draw a bootstrap sample \mathcal{B}_b of size ℓ from training data.
2. Train a classifier f_b on \mathcal{B}_b .

Classification: Classify by majority vote among the B trees:

$$f_{avg} := \frac{1}{B} \sum_{b=1}^B f_b(x)$$

Bagging

Bagging works well for trees:



Random Forests

1. Random forests: modifies bagging with trees to reduce correlation between trees.
2. Tree training optimizes each split over all dimensions.
3. But for Random forests, **choose a different subset of dimensions at each split**. Number of dimensions chosen m .
4. Optimal split is chosen within the subset.
5. The subset is chosen at random out of all dimensions $1, \dots, d$.
6. Recommended $m = \sqrt{d}$ or smaller.

Credit

- “An Introduction to Statistical Learning, with applications in R” (Springer, 2013)” with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Artificial Intelligence

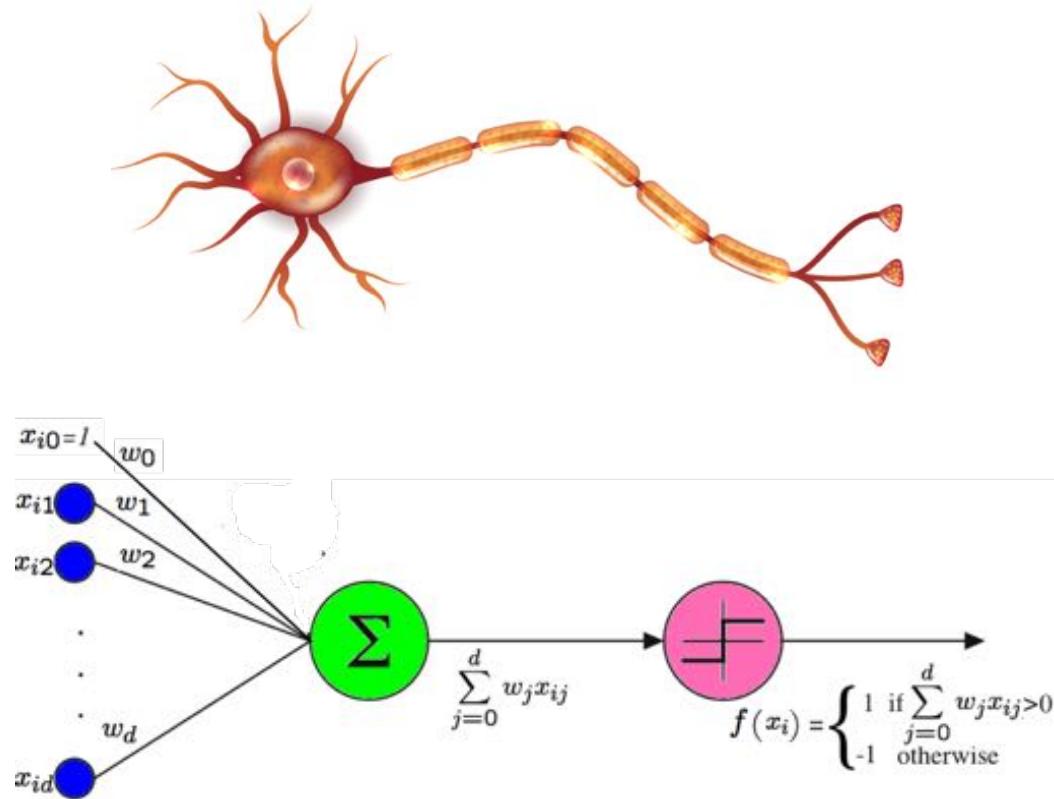
Machine Learning Neural Networks



Neural Networks

- Algorithms that try to mimic how the brain functions.
- Worked extremely well to recognize:
 1. handwritten characters (LeCun et al. 1989),
 2. spoken words (Lang et al. 1990),
 3. faces (Cottrell 1990)
- Extensively studied in the 1990's with a moderate success.
- Now back with lots of success with deep learning thanks to the algorithmic and computational progress.
- The first algorithm used was the Perceptron (Resenblatt 1959).

Perceptron



Given n examples and d features.

$$f(x_i) = \text{sign}\left(\sum_{j=0}^d w_j x_{ij}\right)$$

Perceptron expressiveness

- Consider the perceptron with the step function.
- Idea: Iterative method that starts with a random hyperplane and adjust it using your training data.
- It can represent Boolean functions such as AND, OR, NOT but not the XOR function.
- It produces a linear separator in the input space.

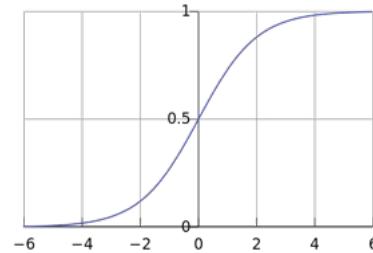
From perceptron to MLP

- The perceptron works perfectly if data is linearly separable. If not, it will not converge.
- Neural networks use the ability of the perceptrons to represent elementary functions and combine them in a network of layers of elementary questions.
- However, a cascade of linear functions is still linear,
- and we want networks that represent highly non-linear functions.

From perceptron to MLP

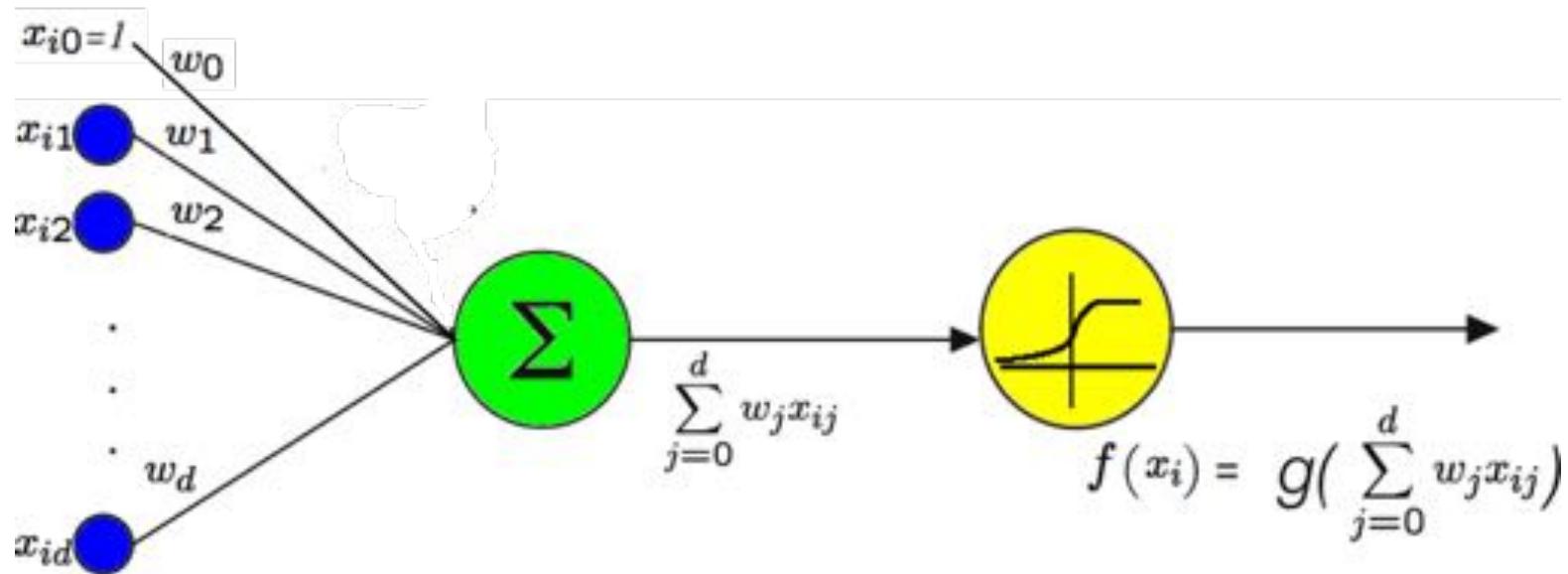
- Also, perceptron used a **threshold function**, which is undifferentiable and not suitable for gradient descent in case data is not linearly separable.
- We want a function whose output is a linear function of the inputs.
- One possibility is to use the sigmoid function:

$$g(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}$$



$$g(z) \rightarrow 1 \text{ when } z \rightarrow +\infty \quad g(z) \rightarrow 0 \text{ when } z \rightarrow -\infty$$

Perceptron with Sigmoid



Given n examples and d features.

For an example x_i (the i^{th} line in the matrix of examples)

$$f(x_i) = \frac{1}{1 + e^{-\sum_{j=0}^d w_j x_{ij}}}$$

The XOR example

Let's try to create a MLP for the XOR function using elementary perceptrons.

The XOR example

Let's try to create a NN for the XOR function using elementary perceptrons.

First observe:

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g(10) = 0.99995$$

$$g(-10) = 0.00004$$

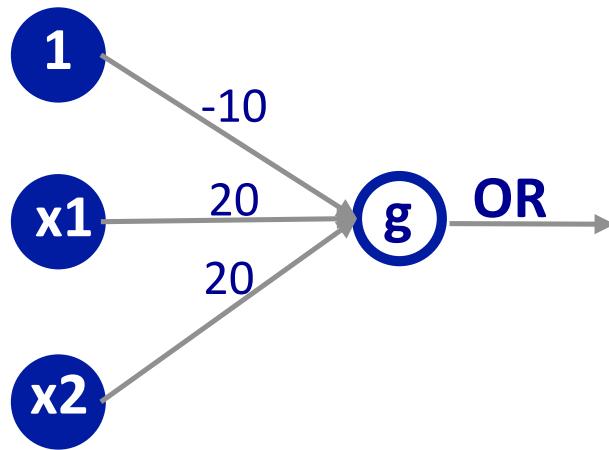
Let's consider that: For $z \geq 10$, $g(z) \rightarrow 1$. For $z \leq -10$, $g(z) \rightarrow 0$.

The XOR example

First what is the perceptron of the OR?

The **XOR** example

x_1	x_2	$x_1 \text{ OR } x_2$	$g(z)$
0	0	0	$g(w_0 + w_1x_1 + w_2x_2) = g(-10)$
0	1	1	$g(10)$
1	0	1	$g(10)$
1	1	1	$g(10)$

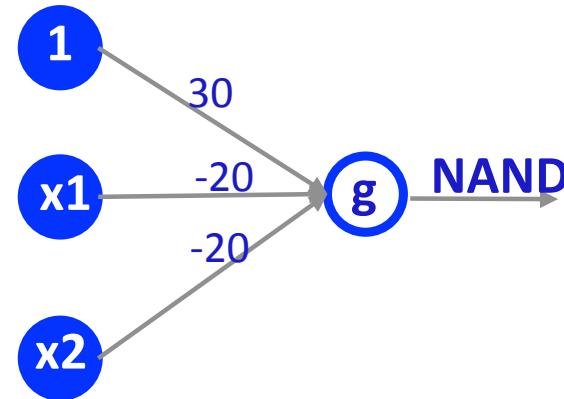
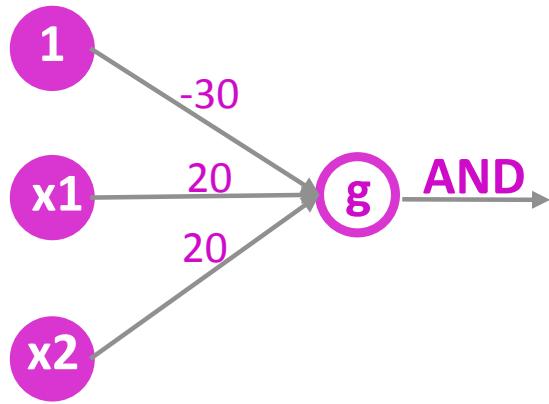


The **XOR** example

Similarly, we obtain the perceptrons for the AND and NAND:

The XOR example

Similarly, we obtain the perceptrons for the AND and NAND:



Note: how the weights in the NAND are the inverse weights of the AND.

The **XOR** example

Let's try to create a NN for the XOR function using elementary perceptrons.

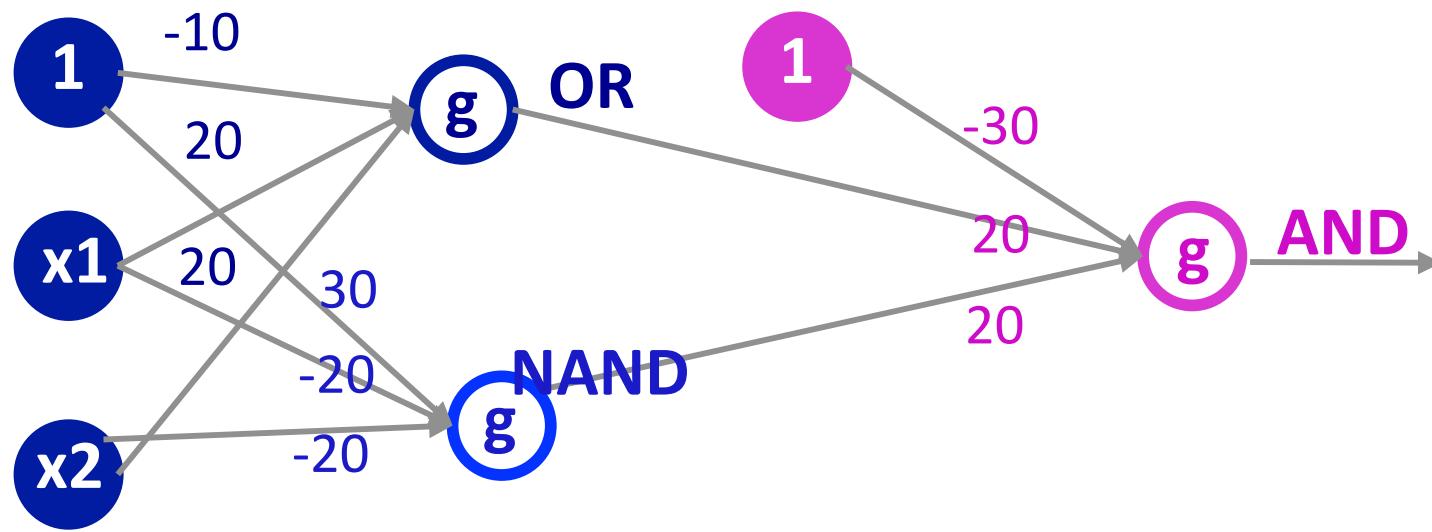
x_1	x_2	$x_1 \text{ XOR } x_2$	$(x_1 \text{ OR } x_2) \text{ AND } (x_1 \text{ NAND } x_2)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

The XOR example

Let's put them together...

The XOR example

Let's put them together...

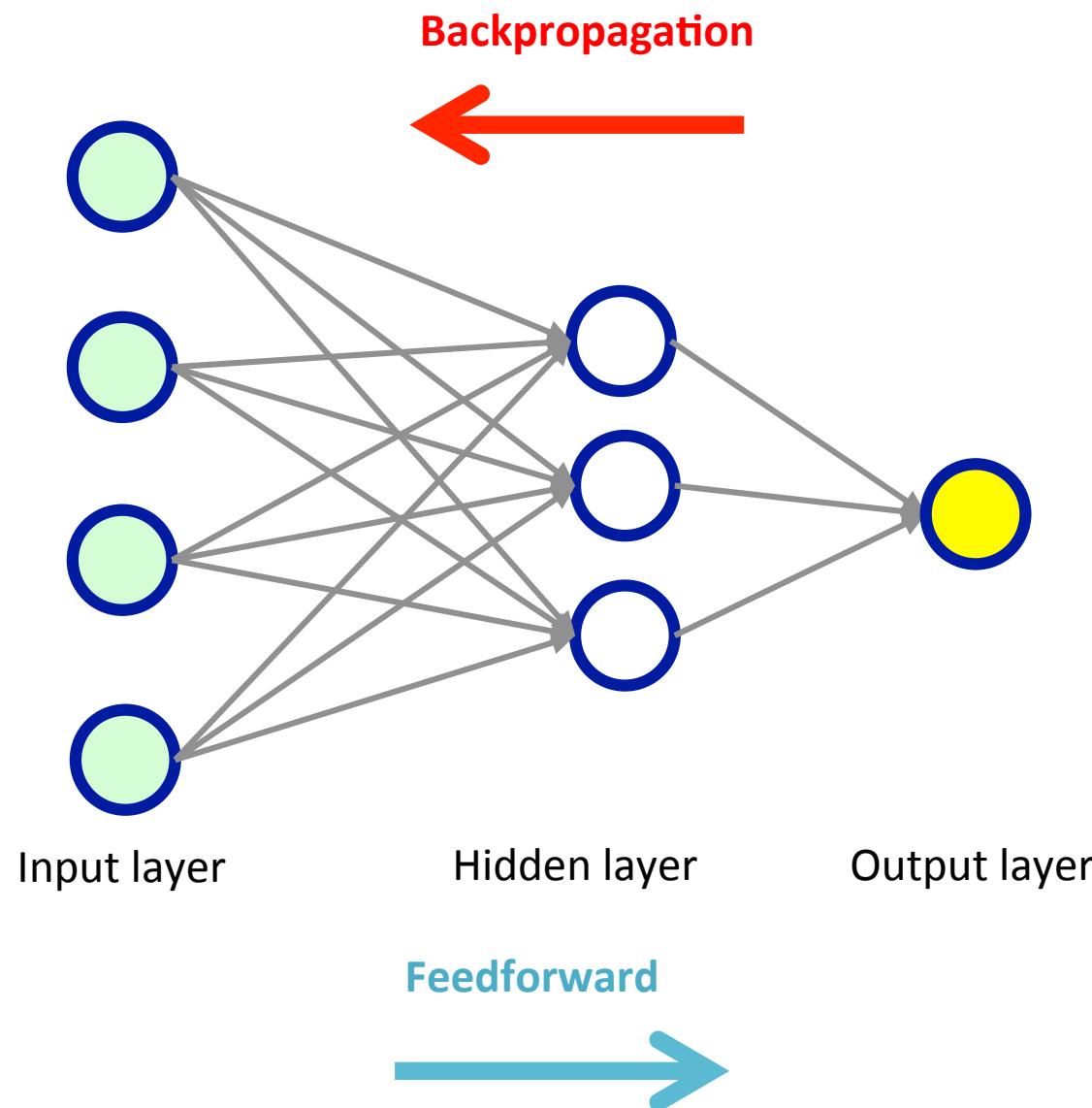


XOR as a combination of 3 basic perceptrons.

Backpropagation algorithm

- Note: Feedforward NN (as opposed to recurrent networks) have no connections that loop.
- Learn the weights for a multilayer network.
- Backpropagation stands for “backward propagation of errors”.
- Given a network with a fixed architecture (neurons and interconnections).
- Use Gradient descent to minimize the squared error between the network output value o and the ground truth y .
- We suppose multiple output k .
- Challenge: Search in all possible weight values for all neurons in the network.

Feedforward-Backpropagation



Backpropagation rules

- We consider k outputs
- For an example e defined by (x, y) , the error on training example e , summed over all output neurons in the network is:

$$E_e(w) = \frac{1}{2} \sum_k (y_k - o_k)^2$$

- Remember, gradient descent iterates through all the training examples one at a time, descending the gradient of the error w.r.t. this example.

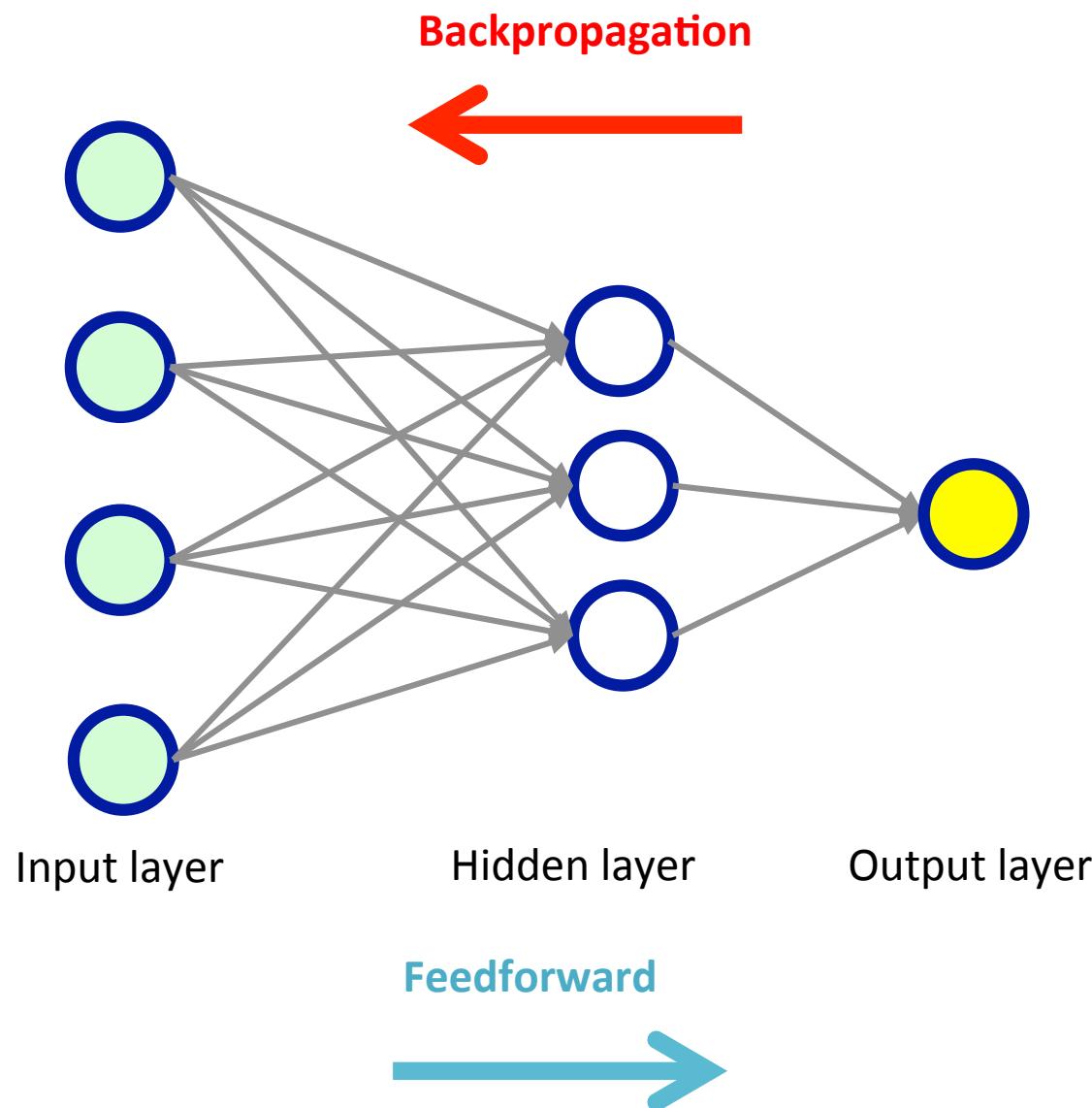
$$\Delta w_{ij} = -\alpha \frac{\partial E_e(w)}{\partial w_{ij}}$$

Backpropagation rules

Notations:

- x_{ij} : the i^{th} input to neuron j .
- w_{ij} : the weight associated with the i^{th} input to neuron j .
- $z_j = \sum w_{ij}x_j$, weighted sum of inputs for neuron j .
- o_j : output computed by neuron j .
- g is the sigmoid function.
- *outputs*: the set of neurons in the output layer.
- $Succ(j)$: the set of neurons whose immediate inputs include the output of neuron j .

Backpropagation notations



Backpropagation rules

$$\frac{\partial E_e}{\partial w_{ij}} = \frac{\partial E_e}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E_e}{\partial z_j} x_{ij}$$

$$\Delta w_{ij} = -\alpha \frac{\partial E_e}{\partial z_j} x_{ij}$$

We consider two cases in calculating $\frac{\partial E_e}{\partial z_j}$ (let's abandon the index e):

- **Case 1: Neuron j is an output neuron**
- **Case 2: Neuron j is a hidden neuron**

Backpropagation rules

- Case 1: Neuron j is an output neuron

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j}$$

Backpropagation rules

- Case 1: Neuron j is an output neuron

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j}$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_k (y_k - o_k)^2$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (y_j - o_j)^2$$

$$\frac{\partial E}{\partial o_j} = \frac{1}{2} 2 (y_j - o_j) \frac{\partial (y_j - o_j)}{\partial o_j}$$

$$\frac{\partial E}{\partial o_j} = -(y_j - o_j)$$

Backpropagation rules

- Case 1: Neuron j is an output neuron

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j}$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_k (y_k - o_k)^2$$

We have: $o_j = g(z_j)$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (y_j - o_j)^2$$

$$\frac{\partial o_j}{\partial z_j} = \frac{\partial g(z_j)}{\partial z_j}$$

$$\frac{\partial E}{\partial o_j} = \frac{1}{2} 2 (y_j - o_j) \frac{\partial (y_j - o_j)}{\partial o_j}$$

$$\frac{\partial o_j}{\partial z_j} = o_j(1 - o_j)$$

$$\frac{\partial E}{\partial o_j} = -(y_j - o_j)$$

Backpropagation rules

$$\frac{\partial E}{\partial z_j} = -(y_j - o_j)o_j(1 - o_j)$$

$$\Delta w_{ij} = \alpha(y_j - o_j)o_j(1 - o_j)x_{ij}$$

We will note

$$\delta_j = -\frac{\partial E}{\partial z_j}$$

$$\Delta w_{ij} = \alpha \delta_j x_{ij}$$

Backpropagation rules

- Case 2: Neuron j is a hidden neuron

$$\frac{\partial E}{\partial z_j} = \sum_{k \in \text{succ}\{j\}} \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial z_j} = \sum_{k \in \text{succ}\{j\}} -\delta_k \frac{\partial z_k}{\partial z_j}$$

$$\frac{\partial E}{\partial z_j} = \sum_{k \in \text{succ}\{j\}} -\delta_k \frac{\partial z_k}{\partial o_j} \frac{\partial o_j}{\partial z_j}$$

$$\frac{\partial E}{\partial z_j} = \sum_{k \in \text{succ}\{j\}} -\delta_k w_{jk} \frac{\partial o_j}{\partial z_j}$$

$$\frac{\partial E}{\partial z_j} = \sum_{k \in \text{succ}\{j\}} -\delta_k w_{jk} o_j (1 - o_j)$$

$$\delta_j = -\frac{\partial E}{\partial z_j} = o_j (1 - o_j) \sum_{k \in \text{succ}\{j\}} \delta_k w_{jk}$$

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i, n_h, n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i, n_h, n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)
 - i. **Feed forward:** Propagate example x through the network and compute the output o_j from every neuron.

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)
 - i. **Feed forward:** Propagate example x through the network and compute the output o_j from every neuron.
 - ii. **Propagate backward:** Propagate the errors backward.

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)
 - i. **Feed forward:** Propagate example x through the network and compute the output o_j from every neuron.
 - ii. **Propagate backward:** Propagate the errors backward.

Case 1 For each output neuron k , calculate its error

$$\delta_k = o_k(1 - o_k)(y_k - o_k)$$

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)
 - i. **Feed forward:** Propagate example x through the network and compute the output o_j from every neuron.
 - ii. **Propagate backward:** Propagate the errors backward.

Case 1 For each output neuron k , calculate its error

$$\delta_k = o_k(1 - o_k)(y_k - o_k)$$

Case 2 For each hidden neuron h , calculate its error

$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{Succ}(h)} w_{hk} \delta_k$$

Backpropagation algorithm

Input: training examples (x, y) , learning rate α (e.g., $\alpha = 0.1$), n_i , n_h and n_o .

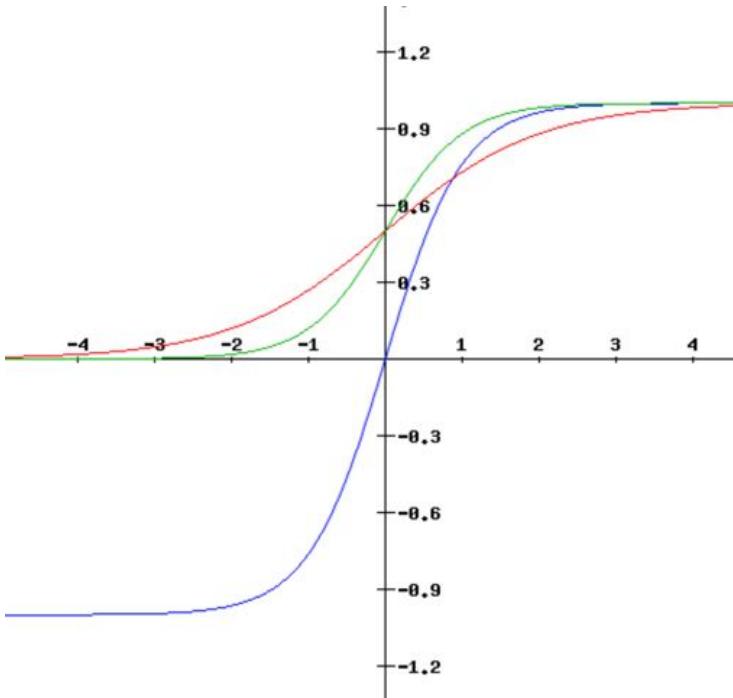
Output: a neural network with one input layer, one hidden layer and one output layer with n_i , n_h and n_o number of neurons respectively and all its weights.

1. Create_feedforward_network (n_i , n_h , n_o)
2. Initialize all weights to a small random number (e.g., in $[-0.2, 0.2]$)
3. Repeat until convergence
 - (a) For each training example (x, y)
 - i. **Feed forward:** Propagate example x through the network and compute the output o_j from every neuron.
 - ii. **Propagate backward:** Propagate the errors backward.
Case 1 For each output neuron k , calculate its error
$$\delta_k = o_k(1 - o_k)(y_k - o_k)$$

Case 2 For each hidden neuron h , calculate its error
$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{Succ}(h)} w_{hk} \delta_k$$
 - iii. Update each weight $w_{ij} \leftarrow w_{ij} + \alpha \delta_j x_{ij}$

Observations

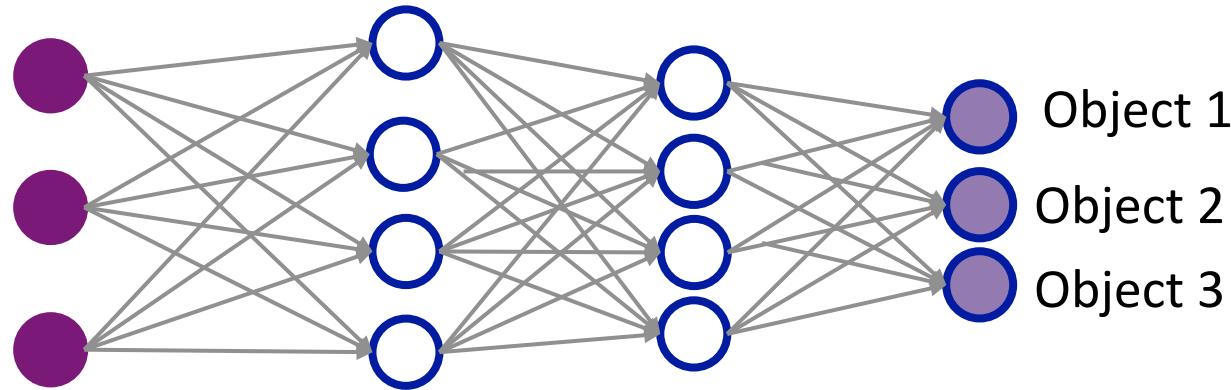
- Convergence: small changes in the weights
- There are other activation functions. Hyperbolic tangent function, is practically better for NN as its outputs range from -1 to 1.



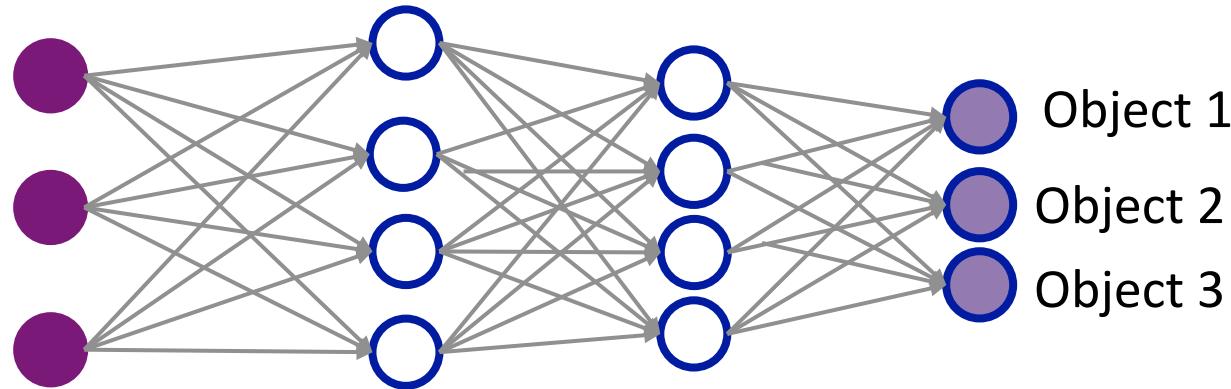
$$g(x) = \text{sigmoid}(x) = \frac{e^{kx}}{1+e^{kx}} \text{ for } k = 1, k = 2, \text{ etc.}$$

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \text{ (It is a rescaling of the logistic sigmoid function!).}$$

Multi class case etc.



Multi class case etc.

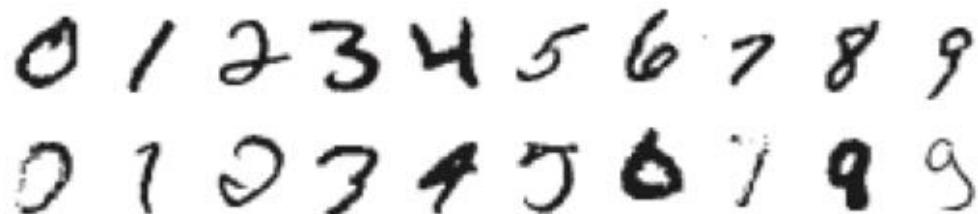


- Nowadays, networks with more than two layers, *a.k.a.* deep networks, have proven to be very effective in many domains.
- Examples of deep networks: restricted Boltzman machines, convolutional NN, auto encoders, etc.

MNIST database

<http://yann.lecun.com/exdb/mnist/>

- The MNIST database of handwritten digits
- Training set of 60,000 examples, test set of 10,000 examples
- Vectors in \mathbb{R}^{784} (28x28 images)
- Labels are the digits they represent
- Various methods have been tested with this training set and test set



- Linear models: 7% - 12% error
- KNN: 0.5%- 5% error
- Neural networks: 0.35% - 4.7% error
- Convolutional NN: 0.23% - 1.7% error

Demo: Tensorflow

<http://playground.tensorflow.org/>

- Open source software to play with neural networks in your browser.
- The dots are colored orange or blue for positive and negative examples.
- It's possible to choose the activation function, architecture, rate etc.
- Very well done! Let's check it out!

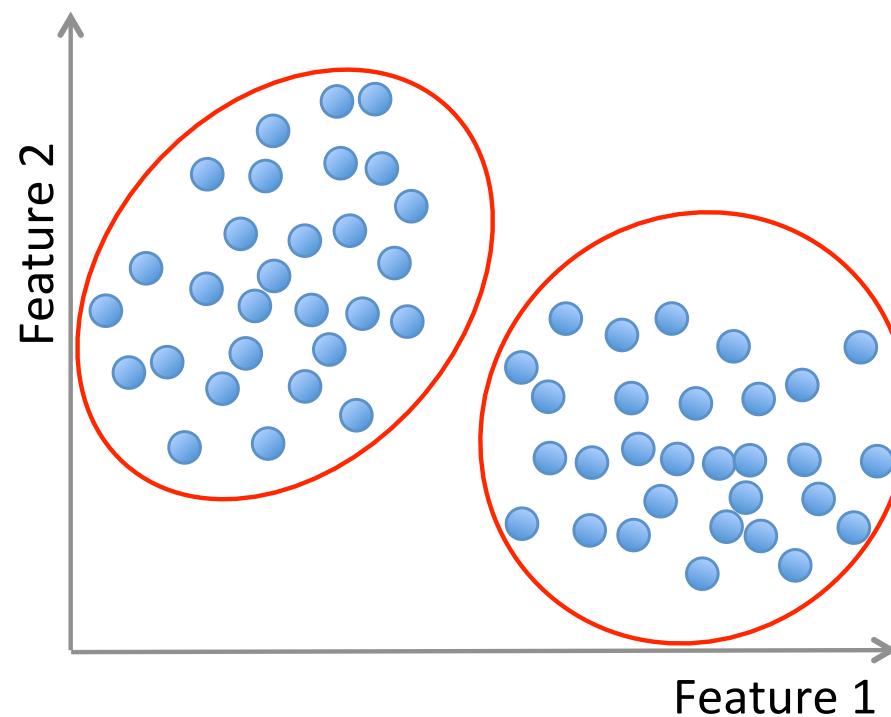
Credit

- Machine Learning 1997. T. Mitchell.
- Andrew Ng's lecture notes.
- <http://playground.tensorflow.org/>

Artificial Intelligence

Machine Learning

Unsupervised learning



Unsupervised Learning

Training data: “examples” x .

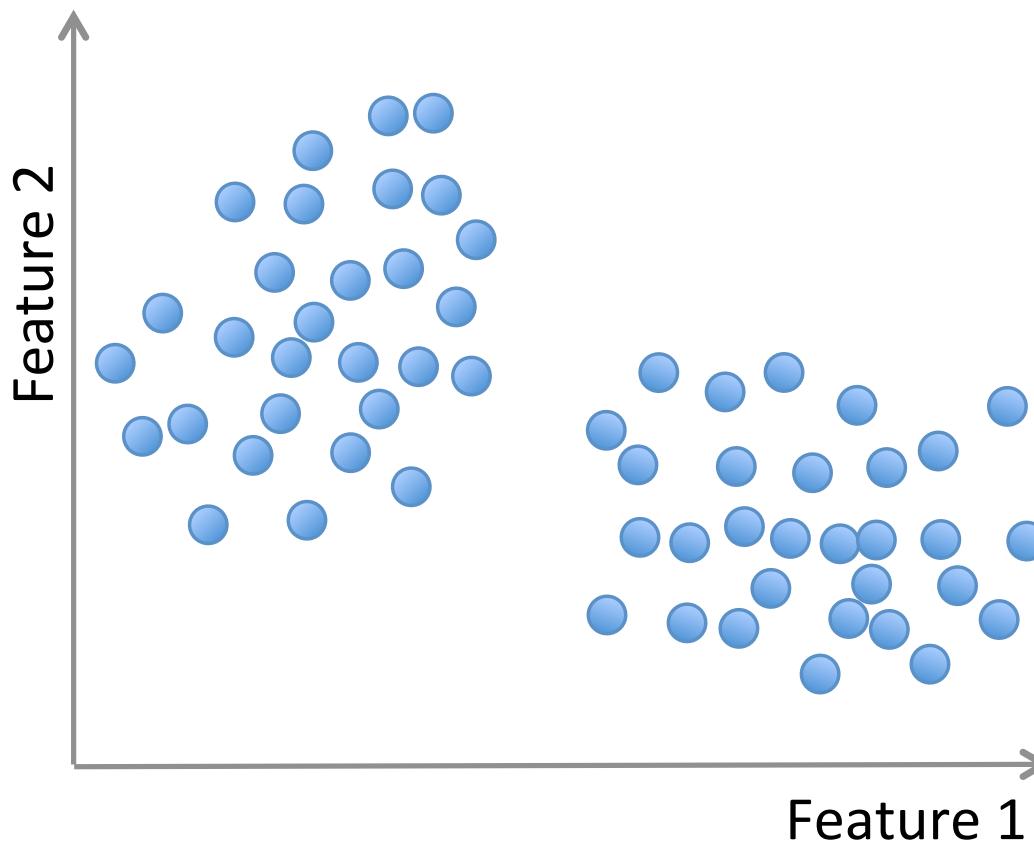
$$x_1, \dots, x_n, \quad x_i \in X \subset \mathbb{R}^n$$

- **Clustering/segmentation:**

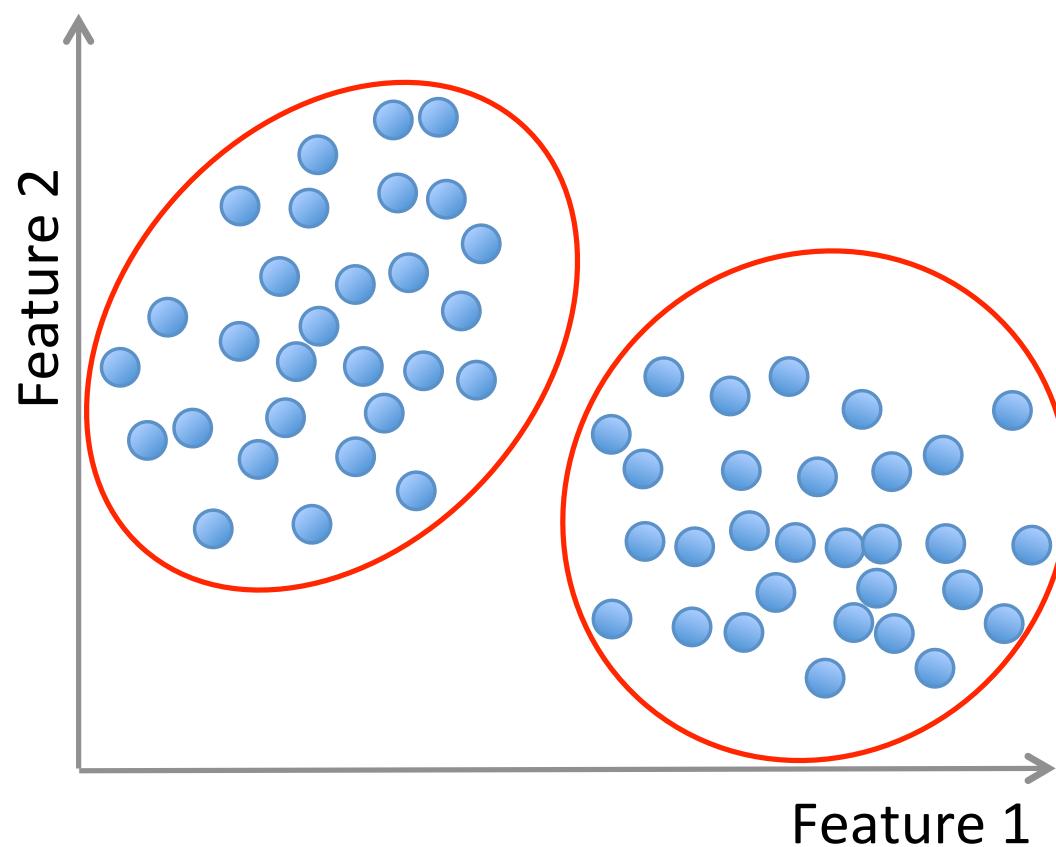
$$f : \mathbb{R}^d \longrightarrow \{C_1, \dots C_k\} \text{ (set of clusters).}$$

Example: Find clusters in the population, fruits, species.

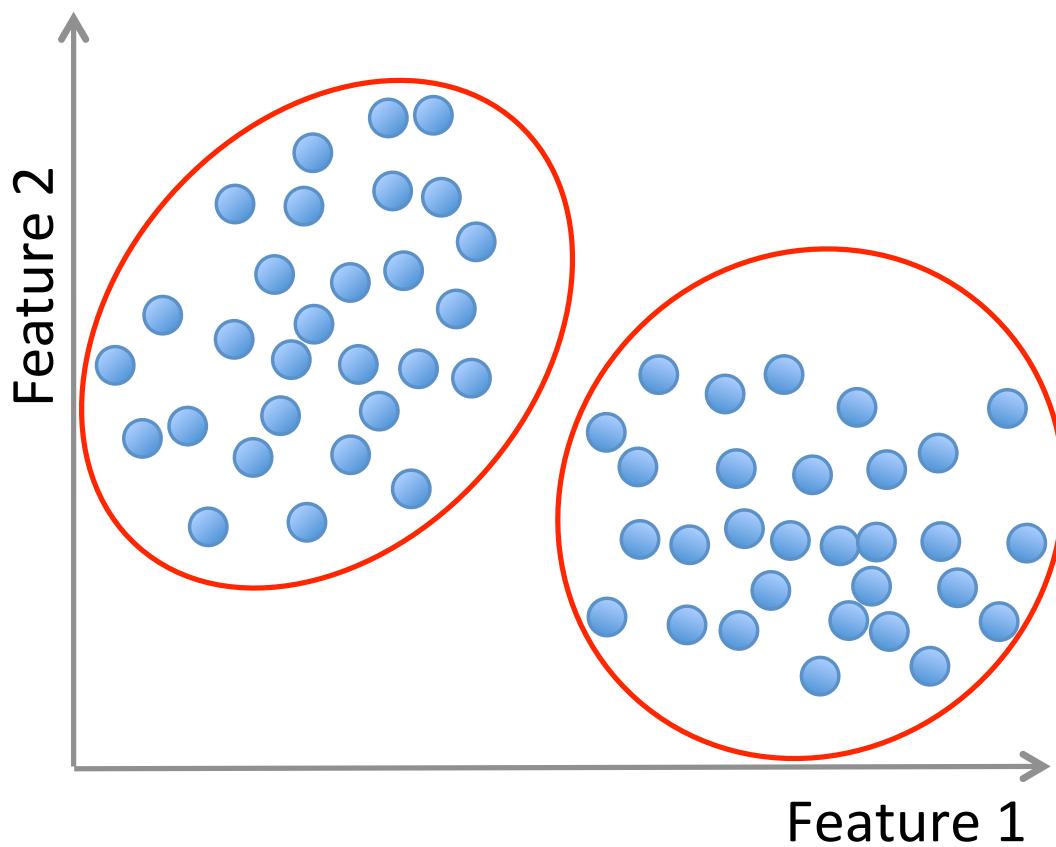
Unsupervised learning



Unsupervised learning



Unsupervised learning

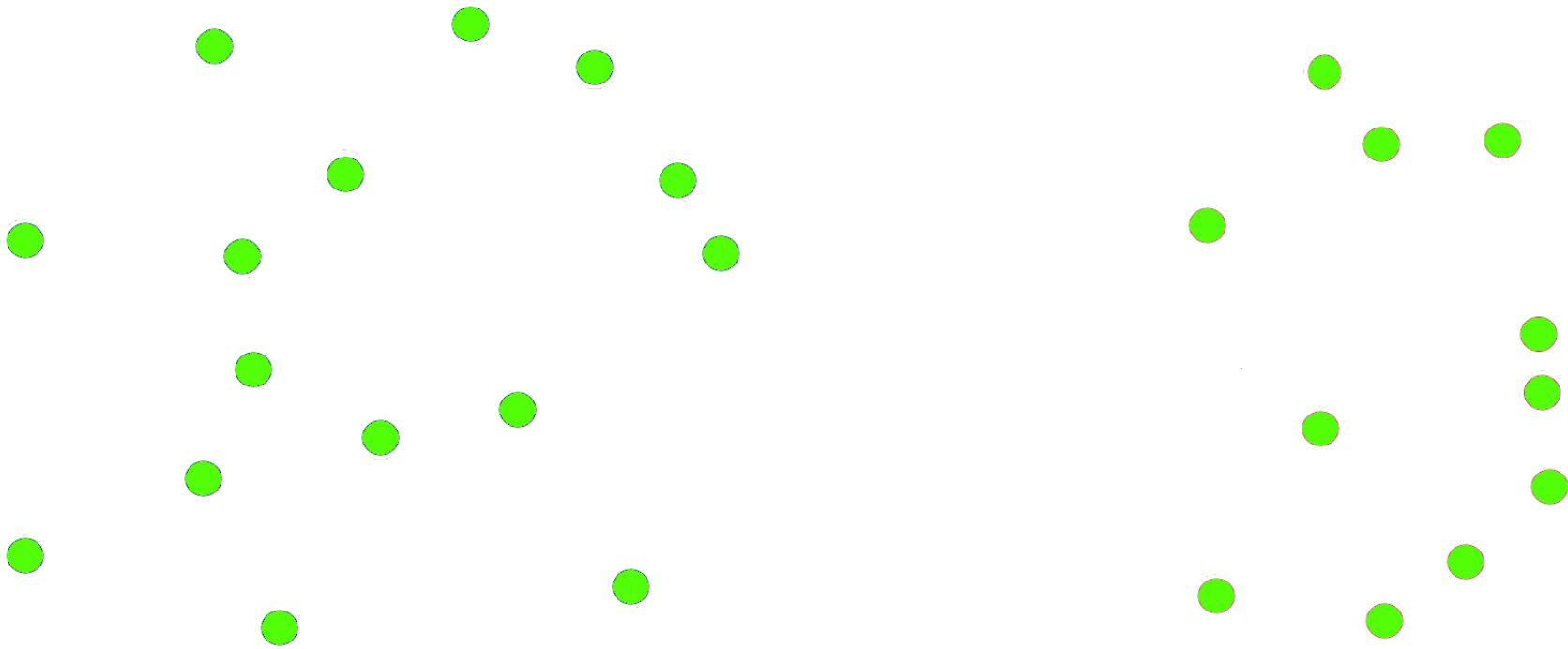


Methods: K-means, gaussian mixtures, hierarchical agglomerative clustering, spectral clustering, DBScan, etc.

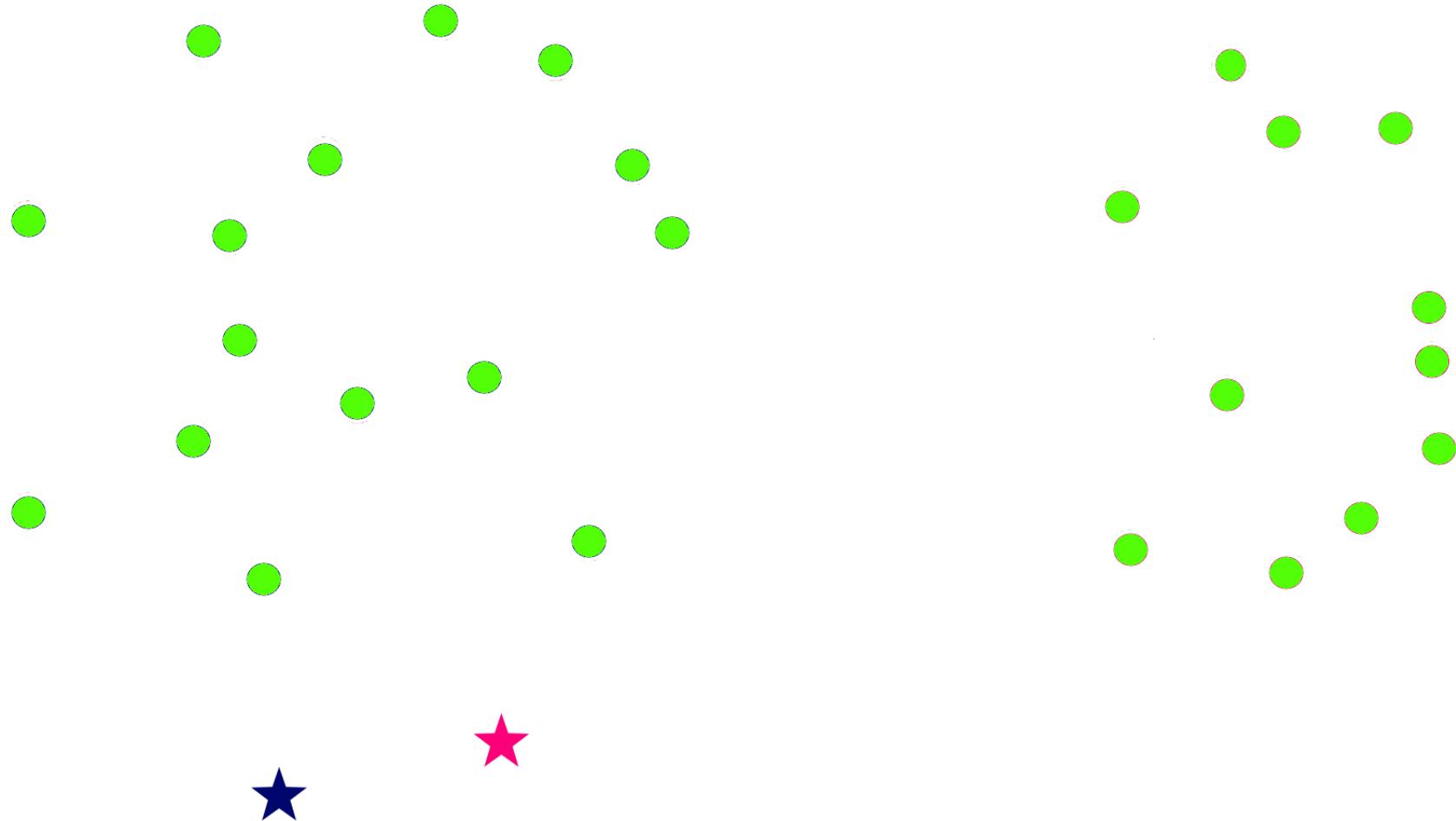
Clustering examples

- Clustering of the population by their demographics.
- Clustering of geographic objects (mineral deposits, houses, etc.)
- Clustering of stars
- Audio signal separation. Example?
- Image segmentation. Example?

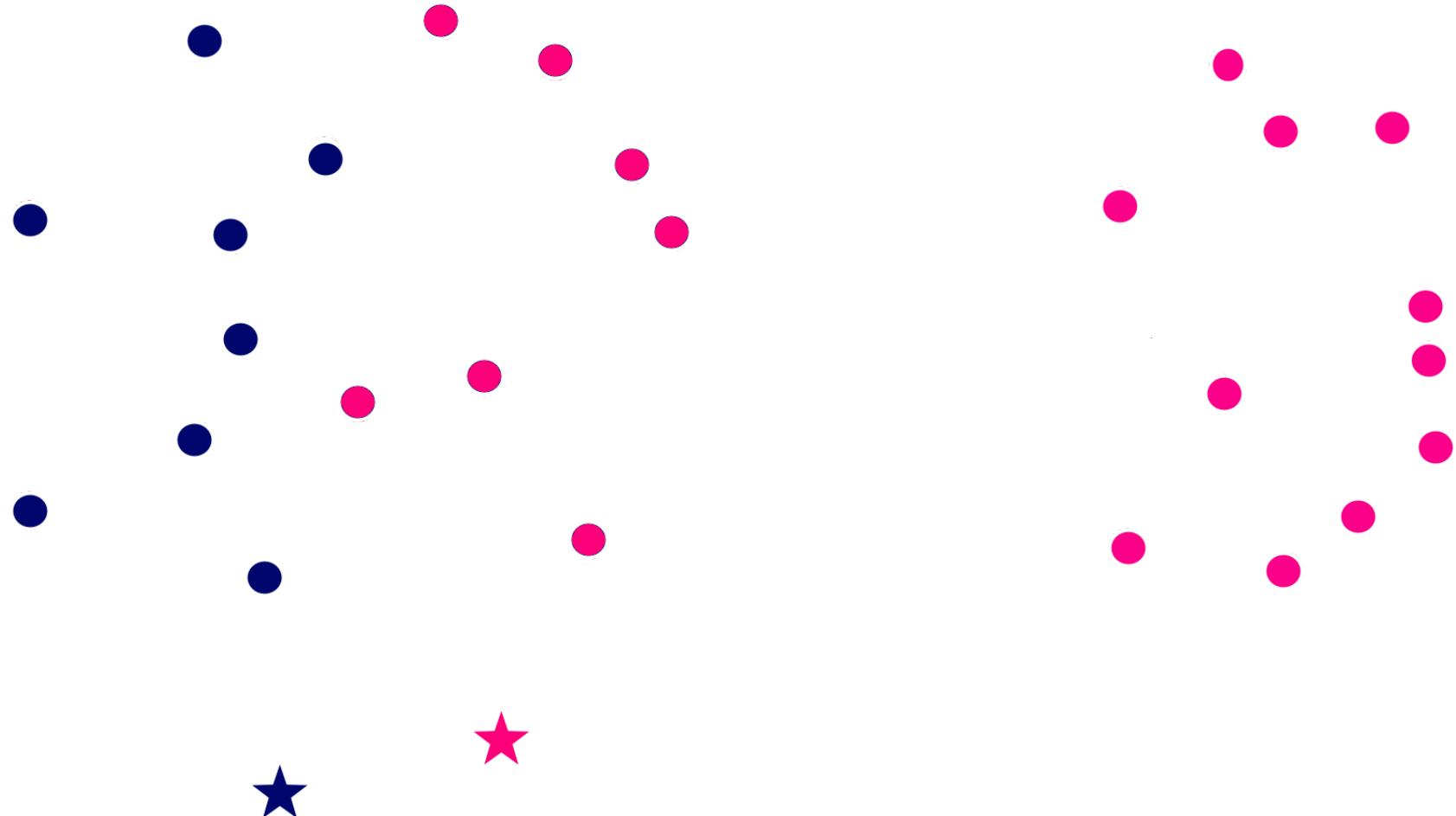
K-Means: example



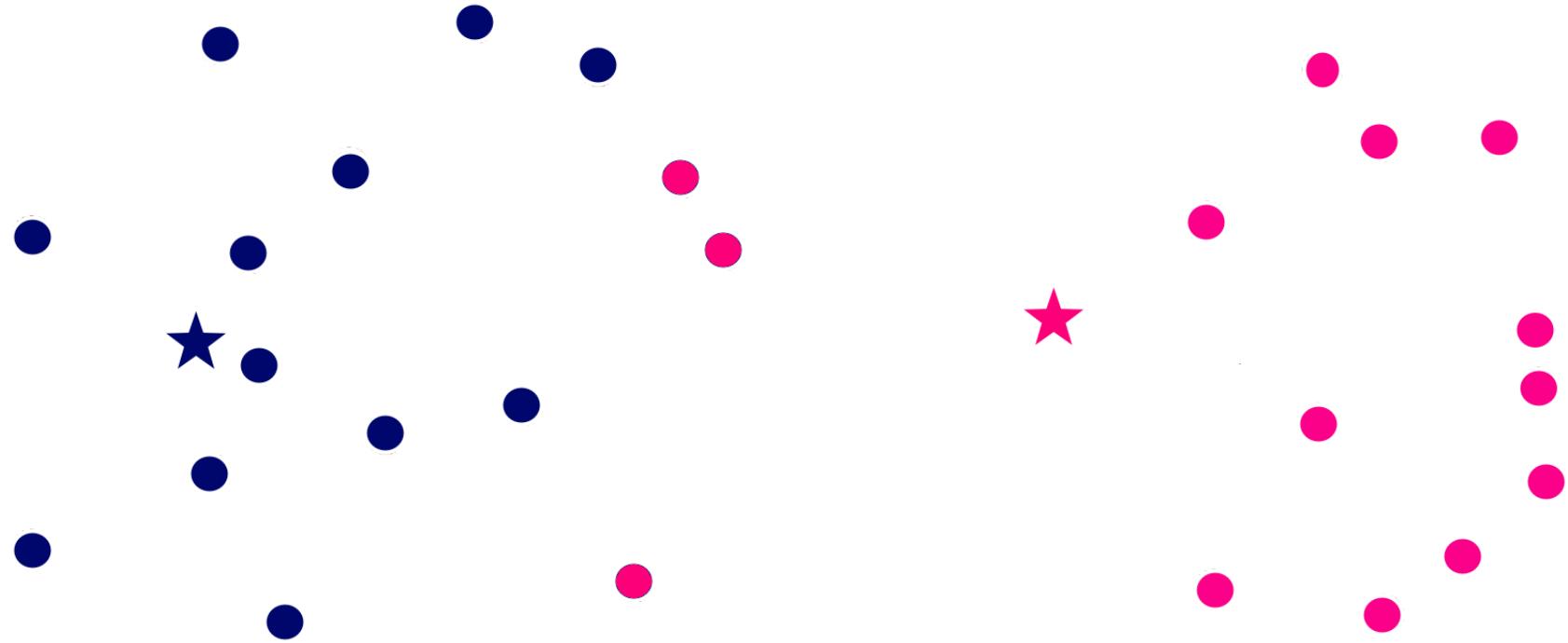
K-Means: example



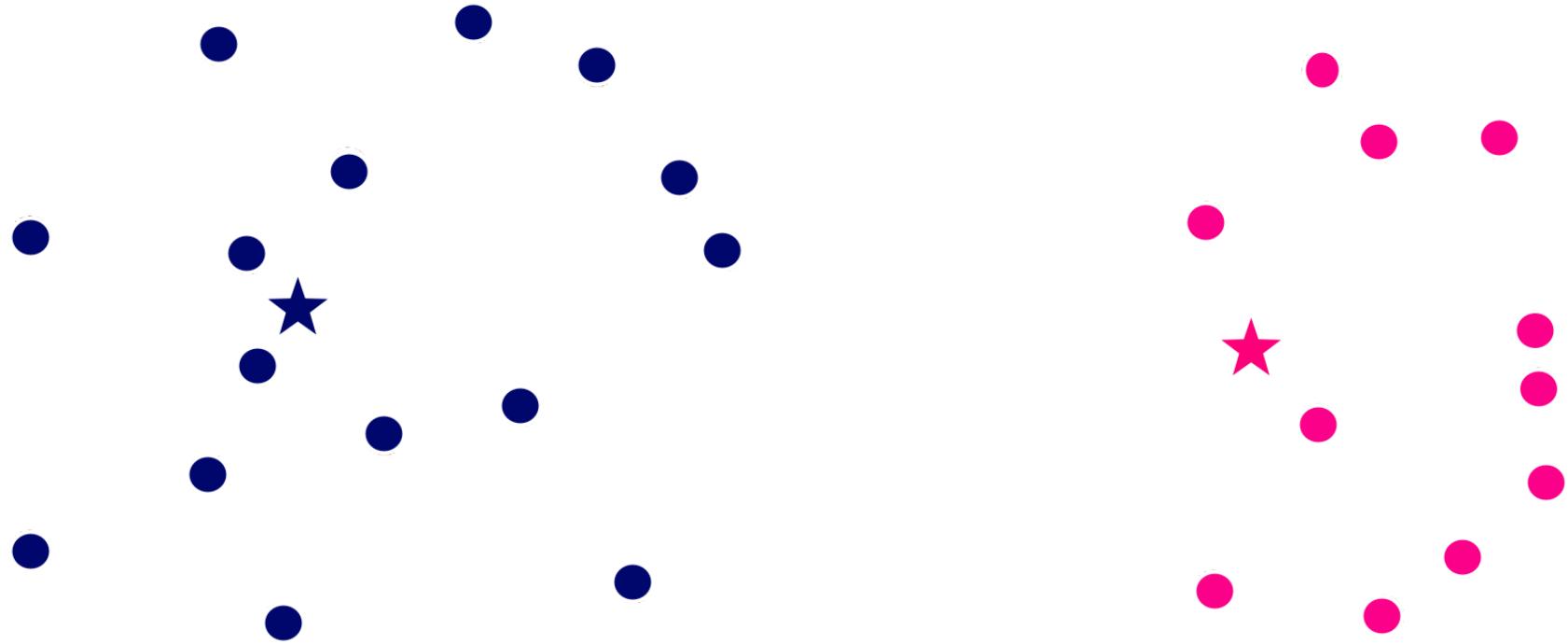
K-Means: example



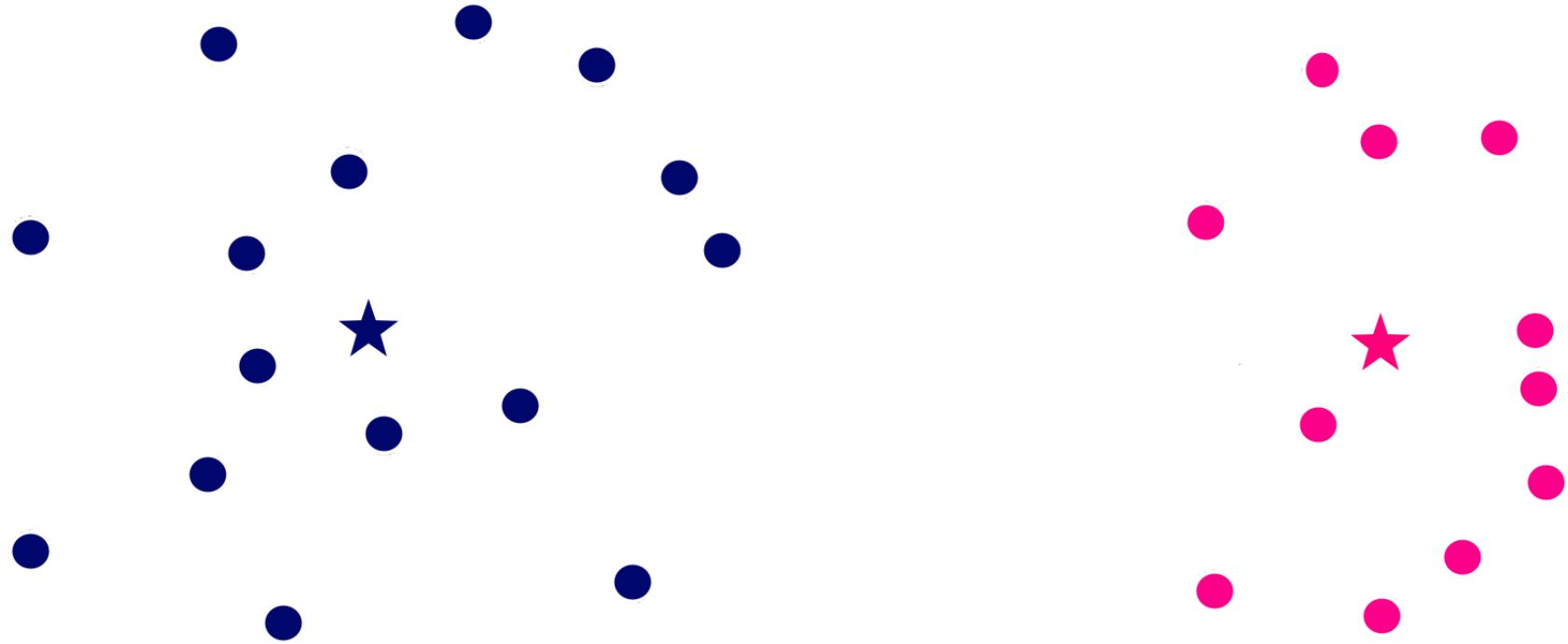
K-Means: example



K-Means: example



K-Means: example



Clustering: K-Means

- **Goal:** Assign each example (x_1, \dots, x_n) to one of the k clusters $\{\mathcal{C}_1, \dots, \mathcal{C}_k\}$.

Clustering: K-Means

- **Goal:** Assign each example (x_1, \dots, x_n) to one of the k clusters $\{\mathcal{C}_1, \dots, \mathcal{C}_k\}$.
- μ_j is the mean of all examples in the j^{th} cluster.

Clustering: K-Means

- **Goal:** Assign each example (x_1, \dots, x_n) to one of the k clusters $\{\mathcal{C}_1, \dots, \mathcal{C}_k\}$.
- μ_j is the mean of all examples in the j^{th} cluster.
- **Minimize:**

$$J = \sum_{j=1}^k \sum_{x_i \in \mathcal{C}_j} ||x_i - \mu_j||^2$$

Clustering: K-Means

Algorithm K-Means:

Initialize randomly μ_1, \dots, μ_k .

Clustering: K-Means

Algorithm K-Means:

Initialize randomly μ_1, \dots, μ_k .

Repeat

Assign each point x_i to the cluster with the closest μ_j .

Clustering: K-Means

Algorithm K-Means:

Initialize randomly μ_1, \dots, μ_k .

Repeat

Assign each point x_i to the cluster with the closest μ_j .

Calculate the new mean for each cluster as follows:

$$\mu_j = \frac{1}{|\mathcal{C}_j|} \sum_{x_i \in \mathcal{C}_j} x_i$$

Until convergence*.

Clustering: K-Means

Algorithm K-Means:

Initialize randomly μ_1, \dots, μ_k .

Repeat

Assign each point x_i to the cluster with the closest μ_j .

Calculate the new mean for each cluster as follows:

$$\mu_j = \frac{1}{|\mathcal{C}_j|} \sum_{x_i \in \mathcal{C}_j} x_i$$

Until convergence*.

*Convergence: Means no change in the clusters OR maximum number of iterations reached.

K-Means: pros and cons

- + Easy to implement

BUT...

- Need to know K
- Suffer from the curse of dimensionality
- No theoretical foundation

K-Means: questions

1. How to set k to optimally cluster the data?
2. How to evaluate your model?
3. How to cluster non circular shapes?

K-Means: question 1

How to set k to optimally cluster the data?

G-means algorithm (Hamerly and Elkan, NIPS 2003)

1. Initialize k to be a small number
2. Run k-means with those cluster centers, and store the resulting centers as C
3. Assign each point to its nearest cluster
4. Determine if the points in each cluster fit a Gaussian distribution (Anderson-Darling test).
5. For each cluster, if the points seem to be normally distributed, keep the cluster center. Otherwise, replace it with two cluster centers.
6. Repeat this algorithm from step 2. until no more cluster centers are created.

K-Means: question 2

How to evaluate your model?

- Not trivial (as compared to counting the number of errors in classification).
- Internal evaluation: using same data. high intra-cluster similarity (documents within a cluster are similar) and low inter-cluster similarity. E.g., Davies-Bouldin index that takes into account both the distance inside the clusters and the distance between clusters. The lower the value of the index, the wider is the separation between different clusters, and the more tightly the points within each cluster are located together.
- External evaluation: use of ground truth of external data. E.g., mutual information, entropy, adjusted random index, etc.

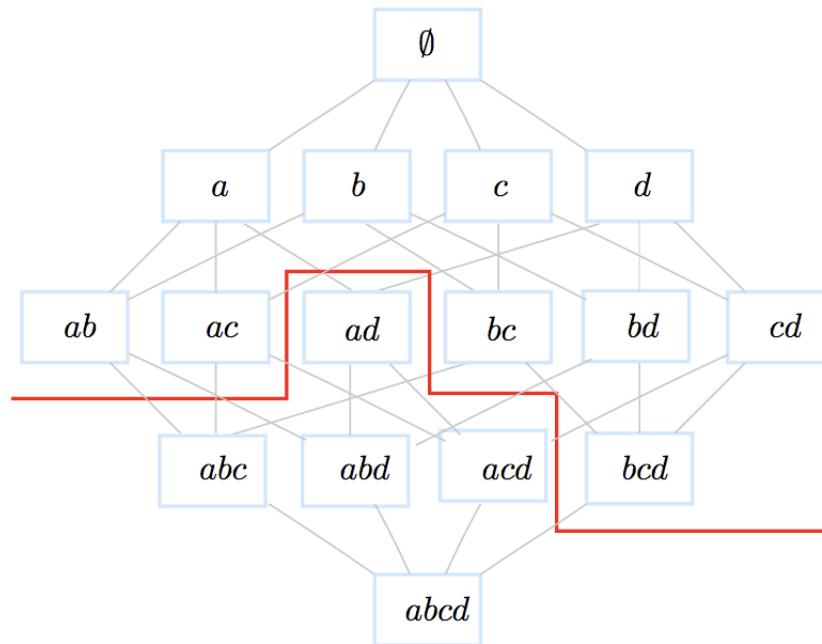
K-Means: question 3

How to cluster non circular shapes?

There are other methods: spectral clustering, DBSCAN, BIRCH, etc. that handle other shapes.

Artificial Intelligence

Machine Learning Association Rules



Outline

1. Introduction
2. A two-step process
3. Applications
4. Definitions and examples
5. Frequent patterns: Apriori algorithm
6. Example
7. Representation of \mathcal{D}
8. Definitions cont'd
9. Association rules algorithm
10. Example
11. A probabilistic framework Association Rules
12. Support-confidence cons
13. Quantitative association rules

Introduction

- Unsupervised task.
- R. Agrawal, T. Imielinski and A.N. Swami Mining Association Rules between sets of items in large databases. Proceedings of SIGMOD 1993.
- Highly cited work because of its wide applicability.

Applications

- Market Basket Analysis: cross-selling (ex. Amazon), product placement, affinity promotion, customer behavior analysis
- Collaborative filtering
- Web organization
- Symptoms-diseases associations
- Supervised classification

A two-step process

Given a transaction dataset \mathcal{D}

1. Mining **frequent** patterns in \mathcal{D}
2. Generation of **strong** association rules

Example:

$\{Bread, Butter\}$ is a frequent pattern (itemset)

$Bread \rightarrow Butter$ is a strong rule

Definitions

- **Item**: an object belonging to $\mathcal{I} = \{x_1, x_2, \dots, x_m\}$.
- **Itemset**: any subset of \mathcal{I} .
- **k -itemset**: an itemset of cardinality k .
- We define a total order $(\mathcal{I}, <)$ on the items.
- $\mathcal{P}(\mathcal{I})$ is a **lattice** with $\perp = \emptyset$ and $\top = \mathcal{I}$.
- **Transaction**: itemset identified by a unique identifier **tid**.
- \mathcal{T} : the set of all transactions ids. **Tidset**: a subset of \mathcal{T} .
- **Transaction dataset**: $\mathcal{D} = \{(tid, X_{tid}) / tid \in \mathcal{T}, X_{tid} \subseteq \mathcal{I}\}$

Example

item	name
a	coffee
b	milk
c	butter
d	bread

\mathcal{D}	
tid	transaction
1	a b
2	a c
3	c d
4	b c d
5	a b c d

$\mathcal{I} =$

$\mathcal{T} =$

$\mathcal{D} =$

Example

item	name
a	coffee
b	milk
c	butter
d	bread

\mathcal{D}	
tid	transaction
1	$a\ b$
2	$a\ c$
3	$c\ d$
4	$b\ c\ d$
5	$a\ b\ c\ d$

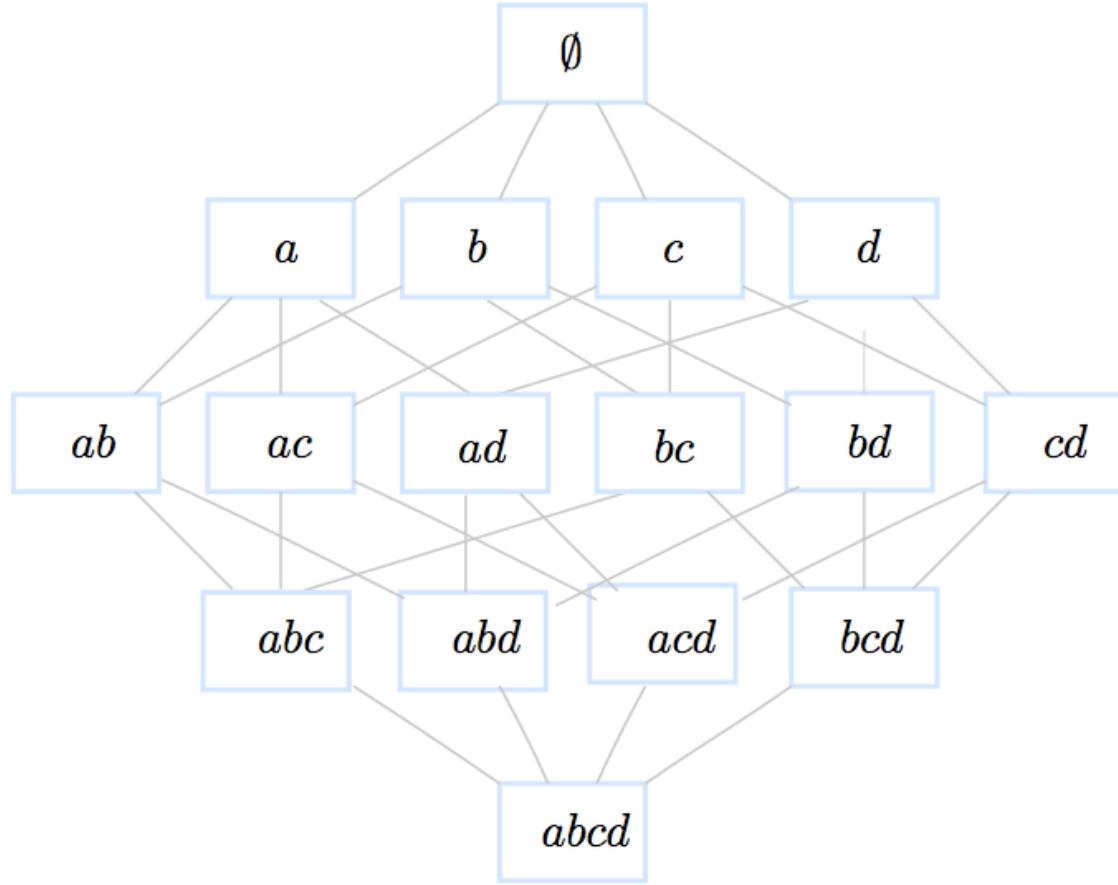
$$\mathcal{I} = \{a, b, c, d\}$$

$$\mathcal{T} = \{1, 2, 3, 4, 5\}$$

$$\mathcal{D} = \{(1, ab), (2, ac), (3, cd), (4, bcd), (5, abcd)\}$$

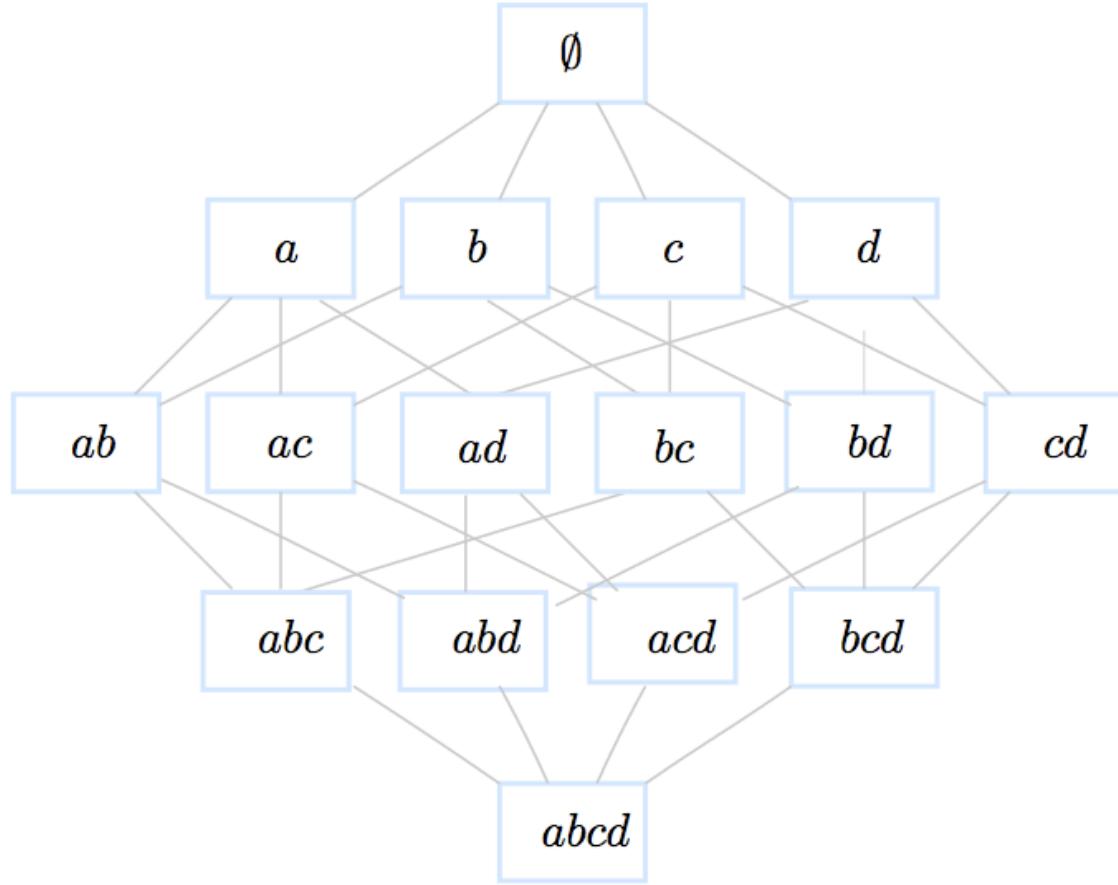
E.g., $\{b, c\}$ is a 2-itemset, for writing simplification we will give up the braces and write bc . $\{3, 4, 5\}$ is a tidset similarly let's abandon the braces here too and write 345.

Example



Lattice of itemsets of size ...

Example



Lattice of itemsets of size $2^{|\mathcal{I}|} = 16$.

Definitions cont'd

- **Mapping t :**

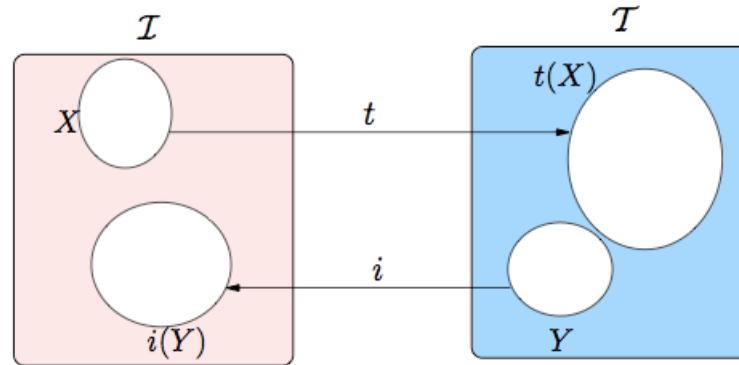
$$t : \mathcal{P}(\mathcal{I}) \rightarrow \mathcal{P}(\mathcal{T})$$

$$X \mapsto t(X) = \{tid \in \mathcal{T} | \exists X_{tid}, (tid, X_{tid}) \in \mathcal{D} \wedge X \subseteq X_{tid}\}$$

- **Mapping i :**

$$i : \mathcal{P}(\mathcal{T}) \rightarrow \mathcal{P}(\mathcal{I})$$

$$Y \mapsto i(Y) = \{x \in \mathcal{I} | \forall (tid, X_{tid}) \in \mathcal{D}, tid \in Y \Rightarrow x \in X_{tid}\}$$

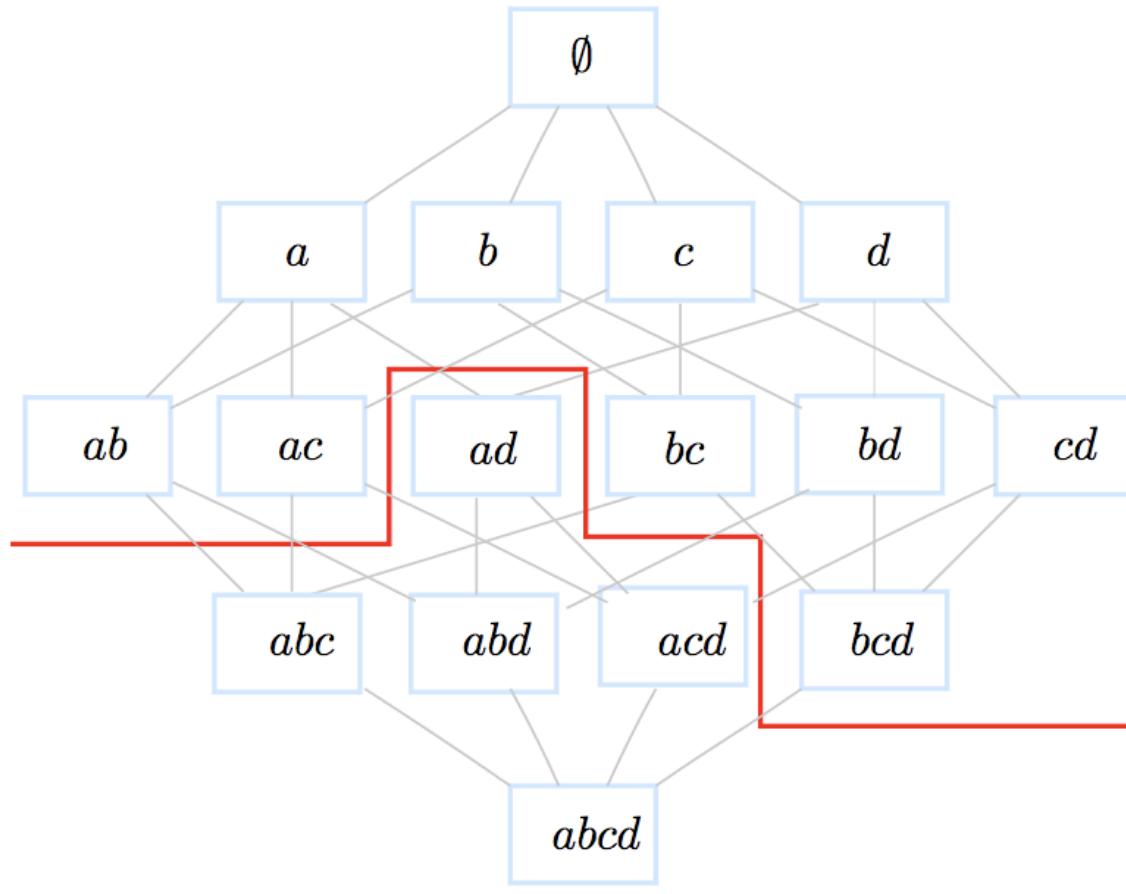


Definitions cont'd

- **Frequency:** $freq(X) = |\{(tid, X_{tid}) \in \mathcal{D} / X \subseteq X_{tid}\}| = |t(X)|$
- **Support:** $supp(X) = \frac{|t(X)|}{|\mathcal{D}|}$
- **Frequent itemset:** X is frequent iff $supp(X) \geq \text{MinSupp}$
- **Property (Support downward closure) :** if an itemset is frequent then all its subsets also are frequent.
- **Mining Frequent Itemsets:**

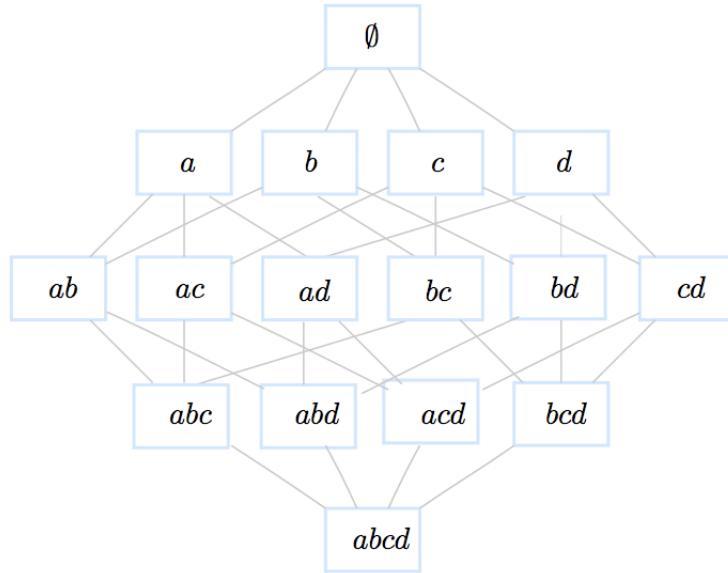
$$\mathcal{F} = \{ X \subseteq \mathcal{I} | \ supp(X) \geq \text{MinSupp} \}$$

Example

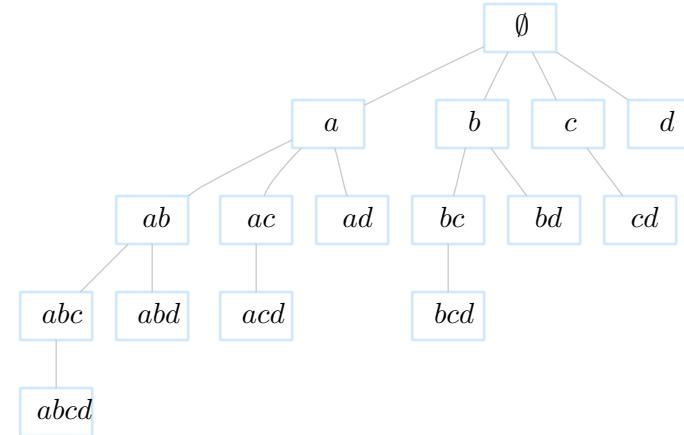


MinSupp=40%

BFS and DFS



Breadth First Search



Depth First Search

Apriori pseudo-algorithm

Level-wise algorithm – lattice explored with a Breath First Search approach (BFS). Start at level 1 in the lattice: $k = 1$

- Generate candidates of size k

$$\mathcal{C}_k = \{(c_k, \text{supp}(c_k)) | \forall X \subset c_k, X \neq \emptyset, \text{support}(X) \geq \text{MinSupp}\}$$

- Scan the dataset to compute the support of each candidate and keep the frequent ones

$$\mathcal{F}_k = \{(l_k, \text{supp}(l_k)) | \text{supp}(l_k) \geq \text{Minsupp}\}$$

- Go to the next level $k = k + 1$ and redo the process.

Example

Minsupp=2/5 (40%)

\mathcal{D}	
tid	transaction
1	a b
2	a c
3	c d
4	b c d
5	a b c d

\mathcal{C}_1	
Itemset	
a	
b	
c	
d	

$\xrightarrow{\text{Scan of } \mathcal{D}}$

\mathcal{C}_1	
Itemset	Support
a	3/5
b	3/5
c	4/5
d	3/5

\mathcal{F}_1	
Itemset	Support
a	3/5
b	3/5
c	4/5
d	3/5

\mathcal{C}_2	
Itemset	
ab	
ac	
ad	
bc	
bd	
cd	

$\xrightarrow{\text{Scan of } \mathcal{D}}$

\mathcal{C}_2	
Itemset	Support
ab	2/5
ac	2/5
ad	1/5
bc	2/5
bd	2/5
cd	3/5

\mathcal{F}_2	
Itemset	Support
ab	2/5
ac	2/5
bc	2/5
bd	2/5
cd	3/5

\mathcal{C}_3	
Itemset	
abc	
bcd	

$\xrightarrow{\text{Scan of } \mathcal{D}}$

\mathcal{C}_3	
Itemset	Support
abc	1/5
bcd	2/5

\mathcal{F}_3	
Itemset	Support
bcd	2/5

Apriori bottleneck

Characteristics of real-life datasets:

1. Billions of transactions,
2. Tens of thousands of items,
3. Tera-bytes of data.

This leads to:

1. Multiple scans of the dataset residing in the disk (costly I/O operations)
2. A **HUGE** number of candidates sets.

Representation of \mathcal{D}

Row-wise

1	a	b		
2	a	c		
3	c	d		
4	b	c	d	
5	a	b	c	d

Column-wise

a	b	c	d
1	1	2	3
2	4	3	4
5	5	4	5

Boolean

	a	b	c	d
1	1	1	0	0
2	1	0	1	0
3	0	0	1	1
4	0	1	1	1
5	1	1	1	1

Definitions cont'd

- Given \mathcal{F} and a Minimum confidence threshold MinConf
- Generate rules:

$$(l - C) \rightarrow C$$

$$conf((l - C) \rightarrow C) = \frac{supp(l)}{supp(l - C)} \geq MinConf$$

- From a $k - itemset$ ($k > 1$), one can generate $2^k - 1$ rules.

Property

Let l be a large (frequent) itemset:

$$\forall C \subset l, C \neq \emptyset, [(l - C) \rightarrow C] \text{ is strong} \Rightarrow \forall \tilde{C} \subset C, \tilde{C} \neq \emptyset, [(l - \tilde{C}) \rightarrow \tilde{C}] \text{ is strong}$$

Example

Minconf=60%

Itemset	Rule#	Rule	Confidence	Strong?
ab	1	$a \rightarrow b$	$2/3 = 66.66\%$	yes
	2	$b \rightarrow a$	$2/3 = 66.66\%$	yes
ac	3	$a \rightarrow c$	$2/3 = 66.66\%$	yes
	4	$c \rightarrow a$	$2/4 = 50.00\%$	no
bc	5	$b \rightarrow c$	$2/3 = 66.66\%$	yes
	6	$c \rightarrow b$	$2/4 = 50.00\%$	no
bd	7	$b \rightarrow d$	$2/3 = 66.66\%$	yes
	8	$d \rightarrow b$	$2/3 = 66.66\%$	yes
cd	9	$c \rightarrow d$	$3/4 = 75.00\%$	yes
	10	$d \rightarrow c$	$3/3 = 100.00\%$	yes

Example

Minconf=60%

Itemset	Rule#	Rule	Confidence	Strong?
bcd	11	$cd \rightarrow b$	$2/3 = 66.66\%$	yes
	12	$bd \rightarrow c$	$2/2 = 100.00\%$	yes
	13	$bc \rightarrow d$	$2/2 = 100.00\%$	yes

Itemset	Rule#	Rule	Confidence	Strong?
bcd	14	$d \rightarrow bc$	$2/3 = 66.66\%$	yes
	15	$c \rightarrow bd$	$2/4 = 50.00\%$	no
	16	$b \rightarrow cd$	$2/3 = 66.66\%$	yes

Probabilistic Interpretation

Brin et al. 97

$$R : A \longrightarrow C$$

- R measures the distribution of A and C in the finite space \mathcal{D} .
- The sets A and C are 2 events
- $P(A)$ and $P(C)$ the probabilities that events A and C happen resp. estimated by the frequency of A and C resp. in \mathcal{D}

$$\text{supp}(A \rightarrow C) = \text{supp}(A \cup C) = P(A \wedge C)$$

$$\text{conf}(A \rightarrow C) = P(C|A) = \frac{P(A \wedge C)}{P(A)}$$

Support-Confidence: cons

- Example (Brin et al. 97)

	<i>coffee</i>	$\overline{\text{coffee}}$	$\sum \text{rows}$
<i>tea</i>	20	5	25
$\overline{\text{tea}}$	70	5	75
$\sum \text{columns}$	90	10	100

$\text{tea} \rightarrow \text{coffee}$ ($\text{supp} = 20\%, \text{conf} = 80\%$)

Strong rule?

Support-Confidence: cons

- Example (Brin et al. 97)

	<i>coffee</i>	$\overline{\text{coffee}}$	$\sum \text{rows}$
<i>tea</i>	20	5	25
$\overline{\text{tea}}$	70	5	75
$\sum \text{columns}$	90	10	100

$\text{tea} \rightarrow \text{coffee}$ ($\text{supp} = 20\%, \text{conf} = 80\%$)

Strong rule? Yes but a misleading one!

$\text{Support}(\text{coffee}) = 90\%$ is a bias that the confidence cannot detect because it ignores $\text{support}(\text{coffee})$.

Other evaluation Measures

- Interest (Piatetsky-Shapiro 91) or Lift (Bayardo et al. 99)

$$Interest(A \rightarrow C) = \frac{P(A \wedge C)}{P(A) \times P(C)} = \frac{supp(A \cup C)}{supp(A) \times supp(C)}$$

Interest is between 0 and $+\infty$:

1. If $Interest(\mathcal{R}) = 1$ then A and C are independent;
2. If $Interest(\mathcal{R}) > 1$ then A and C are positively dependent;
3. If $Interest(\mathcal{R}) < 1$ then A and C are negatively dependent.

$$Interest(A \rightarrow C) = \frac{conf(A \rightarrow C)}{supp(C)} = \frac{conf(C \rightarrow A)}{supp(A)}$$

Other evaluation Measures

	<i>coffee</i>	$\overline{\text{coffee}}$	$\sum \text{rows}$
<i>tea</i>	20	5	25
$\overline{\text{tea}}$	70	5	75
$\sum \text{columns}$	90	10	100

$$\text{Interest}(\text{tea} \rightarrow \text{coffee}) = \frac{P(\text{tea} \wedge \text{coffee})}{P(\text{tea}) \times P(\text{coffee})} = \frac{0.2}{0.25 * 0.9} = 0.89 < 1$$

	<i>coffee</i>	$\overline{\text{coffee}}$
<i>tea</i>	0.89	2
$\overline{\text{tea}}$	1.03	0.66

Multi-dimensional rules

- One-dimensional rules:

$$buy(x, \text{"Bread"}) \longrightarrow buy(x, \text{"Butter"})$$

- Multi-dimensional rules:

$$buy(x, \text{"Pizza"}) \wedge age(x, \text{"Young"}) \longrightarrow buy(x, \text{"Coke"})$$

- Construct k-predicatesets instead of k-itemsets
- How about numerical features?

$$buy(x, \text{"Pizza"}) \wedge age(x, \text{"18 - 22"}) \longrightarrow buy(x, \text{"Coke"})$$

Post-processing of AR

- AR framework may lead to a large number of rules.
- How one can reduce the number of rules?
 1. Use many evaluation measures
 2. Increase minimum support
 3. Increase minimum confidence
 4. use rule templates (define constraints on max rule length, exclude some items, include in the rules specific items)
(Agrawal et al. 1995, Salleb et al. 2007)

Implementations

- **FIMI** Frequent Itemset Mining Implementations Repository
<http://fimi.cs.helsinki.fi/> FIMI'03 and FIMI'04 workshop,
Bayardo, Goethals & Zaki
- **Apriori** <http://www.borgelt.net/apriori.html> developed by
Borgelt
- **Weka** <http://www.cs.waikato.ac.nz/ml/weka/> by Witten &
Frank
- **ARMADA** Data Mining Tool version 1.3.2 in matlab available
at Mathworks, by Malone

FP algorithms

According to the strategy to traverse the search space:

- Breadth First Search (ex: Apriori, AprioriTid, Partition, DIC)
- Depth First Search (ex: Eclat, Clique, Depth project)
- Hybrid (ex: AprioriHybrid, Hybrid, Viper, Kdci)
- Pattern growth, i.e. no candidate generation (ex: Fpgrowth, HMine, Cofi)

Uniform notion of item

- Apriori has been initially designed for **boolean tables** (transactional datasets) thus propositional logic was sufficient to express: items, itemsets and rules.

milk → cereals

- For **relational tables**, one need to extend the notion of items to literals:

item \equiv (*attribute, value*)

An attribute could be:

1. categorical, for ex. (*color, blue*),
2. quantitatif with a few numerical values, for ex. (*#cars, 2*),
3. quantitatif with a large domain values, for ex. (*age, [20, 40]*).

Example

\mathcal{D} : people			
id	age	married?	#cars
1	23	no	1
2	25	yes	1
3	29	no	0
4	34	yes	2
5	38	yes	2

Examples of frequent itemsets	
itemset	support
(age, 20..29)	3
(age, 30..39)	2
(married?, yes)	3
(married?, no)	2
(#cars, 1)	2
(#cars, 2)	2
(age, 30..39), (married?, yes)	2

Examples of rules			
rule	support	confidence	
(age, 30..39) et (married?, yes) \rightarrow (#cars, 2)	40%	100%	
(age, 20..29) \rightarrow (#cars, 1)	60%	66.6%	

Quantitative AR

Question: Mining Quantitative AR is not a simple extension of mining categorical AR. why?

- **Infinite search space:** In Boolean AR, the Arriori property allows to prune the search space efficiently, but we do explore the whole space of hypothesis (lattice of itemsets), which is IMPOSSIBLE for Quantitative AR.
- **The support-confidence tradeoff:** Choosing intervals is quite sensitive to support and confidence.
 - intervals too small, not enough support;
 - intervals too large, not enough confidence.
- What is the difference between supervised and **unsupervised discretization**?

Approaches to mine QARs

- Discretization-based approaches
- Distribution-based approaches
- Optimization-based approaches

Approaches to mine QARs

Discretization-based approaches

- A pre-processing step
- Use equi-depth, equi-width, domain-knowledge
- Lent et al., 1997; Miller and Yang, 1997; Srikant and Agrawal, 1996; Wang et al., 1998
- Discretization combined with clustering or interval merging.
- Problems: univariate, sensitive to outliers, loss of information.

Approaches to mine QARs

Distribution-based approaches

$Sex = \text{female} \rightarrow Height : mean = 168 \wedge Weight : mean = 68$

- Aumann and Lindell, 1999, Webb 2001.
- Restricted form of rules:
 1. A set of categorical attributes on the left-hand side and several distributions on the right-hand side,
 2. A single discretized numeric attribute on the left-hand side and a single distribution on the right-hand side.

Approaches to mine QARs

Optimization-based approaches

- Numerical attributes are optimized during the mining process
- Fukuda et al., 96, Rastogi and Shim 99, Brin et al. 2003.
Techniques inspired from image segmentation.

$$Gain(A \rightarrow B) = Supp(AB) - MinConf * Supp(A)$$

Form of the rules restricted to 1 or 2 numerical attributes.

- Mata et al. 2002 Use genetic algorithms to optimize the support of itemsets with non instantiated intervals.

$$\text{Fitness} = cov - (\psi * ampl) - (\omega * mark) + (\mu * nAtr)$$

Apriori-like algorithm to mine association rules.

Approaches to mine QARs

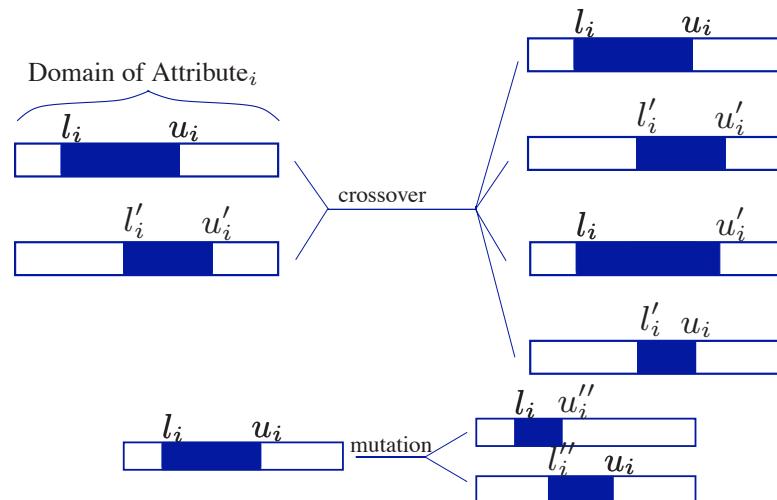
Optimization-based approaches

- Ruckert et al. 2004 use half-spaces to mine such rules like:

$$x_1 > 20 \rightarrow 0.5x_3 + 2.3x_6 \geq 100$$

Cannot handle categorical attributes.

- Salleb et al 2007: QuantMiner Optimize the *Gain* of rules templates using a genetic algorithm.



Approaches to mine QARs

Optimization-based approaches: QuantMiner cont'd.

Example UCI Iris dataset:

$$\begin{array}{ll} \text{Species=} & \Rightarrow \left\{ \begin{array}{ll} \text{PW} \in [l_1, u_1] & \text{SW} \in [l_2, u_2] \\ \text{value} & \text{PL} \in [l_3, u_3] \quad \text{SL} \in [l_4, u_4] \end{array} \right\} \begin{array}{l} \text{supp\%} \\ \text{conf\%} \end{array} \end{array}$$

$$\begin{array}{ll} \text{Species=} & \Rightarrow \left\{ \begin{array}{ll} \text{PW} \in [1, 6] & \text{SW} \in [31, 39] \\ \text{setosa} & \text{PL} \in [10, 19] \quad \text{SL} \in [46, 54] \end{array} \right\} \begin{array}{l} 23\% \\ 70\% \end{array} \end{array}$$

$$\begin{array}{ll} \text{Species=} & \Rightarrow \left\{ \begin{array}{ll} \text{PW} \in [10, 15] & \text{SW} \in [22, 30] \\ \text{versicolor} & \text{PL} \in [35, 47] \quad \text{SL} \in [55, 66] \end{array} \right\} \begin{array}{l} 21\% \\ 64\% \end{array} \end{array}$$

$$\begin{array}{ll} \text{Species=} & \Rightarrow \left\{ \begin{array}{ll} \text{PW} \in [18, 25] & \text{SW} \in [27, 33] \\ \text{virginica} & \text{PL} \in [48, 60] \quad \text{SL} \in [58, 72] \end{array} \right\} \begin{array}{l} 20\% \\ 60\% \end{array} \end{array}$$

QuantMiner

<http://quantminer.github.io/QuantMiner/>

QuantMiner

Attributes

sepal_length	sepal_width	petal_length	petal_width	Iris_class
5	2	3.5	1	versicolor
6	2.2	4	1	versicolor
6.2	2.2	4.5	1.5	versicolor
6	2.2	5	1.5	virginica
4.5	2.3	1.3	0.3	setosa
5.5	2.3	4	1.3	versicolor
6.3	2.3	4.4	1.3	versicolor
5	2.3	3.3	1	versicolor
4.9	2.4	3.3	1	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1	versicolor
5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5			
6.3	2.5			
5.7	2.6			
5.5	2.6			
4.4	2.6			
5.8	2.6			

Data point /example

5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5			
6.3	2.5			
5.7	2.6			
5.5	2.6			
4.4	2.6			
5.8	2.6			

Numerical value

5.6	2.5	3.9	1.1	versicolor
6.3	2.5	4.9	1.5	versicolor
5.5	2.5	4	1.3	versicolor
5.1	2.5	3	1.1	versicolor
4.9	2.5	4.5	1.7	virginica
6.7	2.5	5.8	1.8	virginica
5.7	2.5			
6.3	2.5			
5.7	2.6			
5.5	2.6			
4.4	2.6			
5.8	2.6			

UCI IRIS dataset

QuantMiner interface showing the process of mining rules from the UCI IRIS dataset:

- Choosing rule templates:** Step 2/5. A window titled "Choosing rule templates" shows the user interface for selecting rule templates.
- Mining rules using a genetic algorithm:** Step 4/5. A window titled "Mining rules using a genetic algorithm" displays the progress of rule mining, stating "Pre-computation with the Apriori algorithm: Computing the set of 2 consecutive frequent modalities...FINISH!" and "Computing the number of rules to test...: 6 rules."
- Results:** Step 5/5. A window titled "Results" displays the mined rules. One rule is shown in detail:

Rule 6/6 (total : 6)

6. SUPPORT = 41 (27.33 %) , CONFIDENCE = 82.0 % :

A → B

A: {Class = Iris-versicolor}

B: {petal_length in [3.3; 4.7] } → {petal_width in [1.0; 1.5]}

SUPPORTS :

A and B	41 (27.33 %)
A	50 (33.33 %)
B	41 (27.33 %)
A and (\neg B)	9 (6.0 %)
(\neg A) and B	0 (0.0 %)
(\neg A) and (\neg B)	100 (66.67 %)

CONFIDENCES:

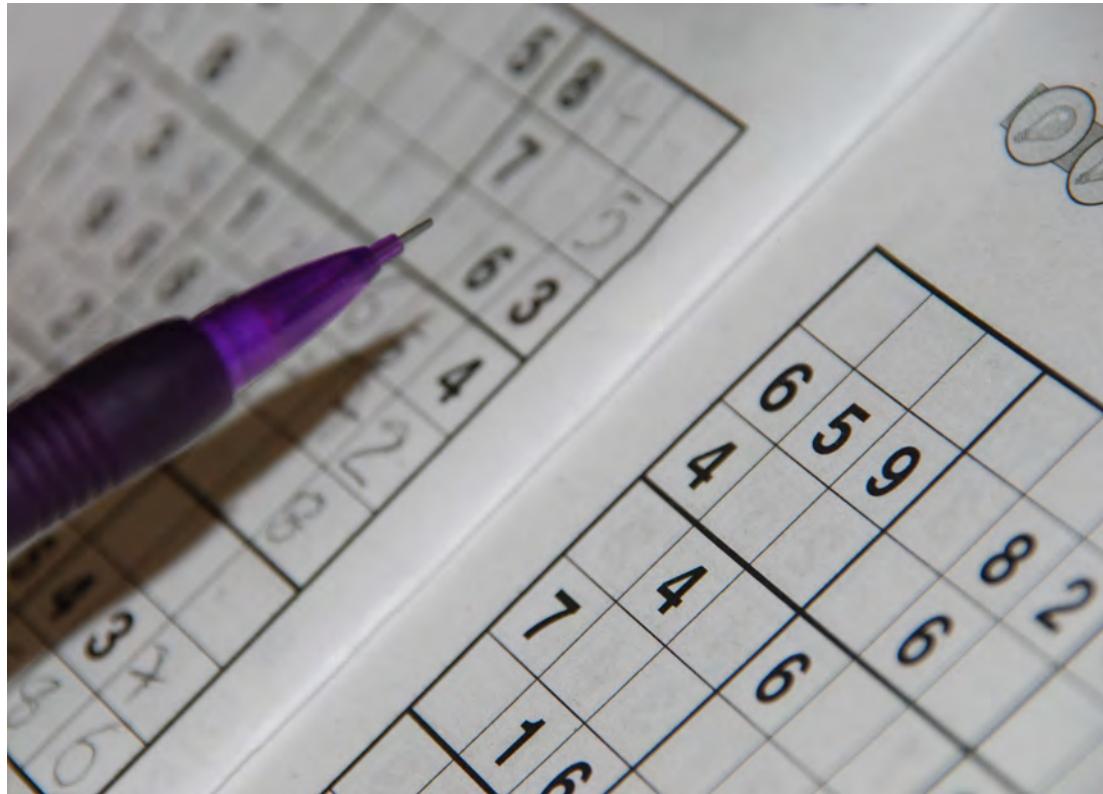
A → B	82.0 %
(\neg A) → B	0.0 %
B → A	100.0 %
(\neg B) → A	8.26 %
A ↔ B	94.0 %

References

- R. Agrawal, T. Imielinski and A.N. Swami “Mining Association Rules between sets of items in large databases”. SIGMOD 1993.
- R. Agrawal, R. Srikant “Fast algorithms for mining association rules ” VLDB 1994.
- B. Goethals “Survey on Frequent Pattern Mining” Technical report, Helsinki Institute for Information Technology, 2003.
- S. Brin et al. “Beyond Market Baskets: Generalizing Association Rules to Correlations”. SIGMOD 1997.
- R. Agrawal et al. “Mining association rules with item constraints”. KDD 1997.
- A. Salleb et al. “QuantMiner: A Genetic Algorithm for Mining Quantitative Association Rules”, IJCAI 2007.
- U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “From Data Mining to Knowledge Discovery: An Overview”. In Advances in Knowledge Discovery and Data Mining, 1996.

Artificial Intelligence

Constraint Satisfaction Problems



Recall

- **Search problems:**
 - Find the **sequence of actions** that leads to the goal.
 - Sequence of actions means a **path** in the search space.
 - Paths come with different costs and depths.
 - We use “rules of thumb” aka **heuristics** to guide the search efficiently.

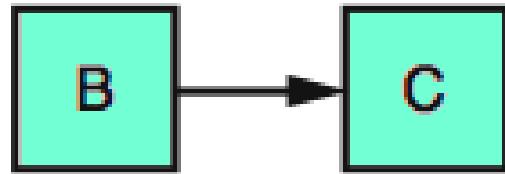
Recall

- **Search problems:**
 - Find the **sequence of actions** that leads to the goal.
 - Sequence of actions means a **path** in the search space.
 - Paths come with different costs and depths.
 - We use “rules of thumb” aka **heuristics** to guide the search efficiently.
- **Constraint satisfaction problems:**
 - A search problem too!
 - We care about the **goal itself**.

CSPs definition

- **Search problems:**

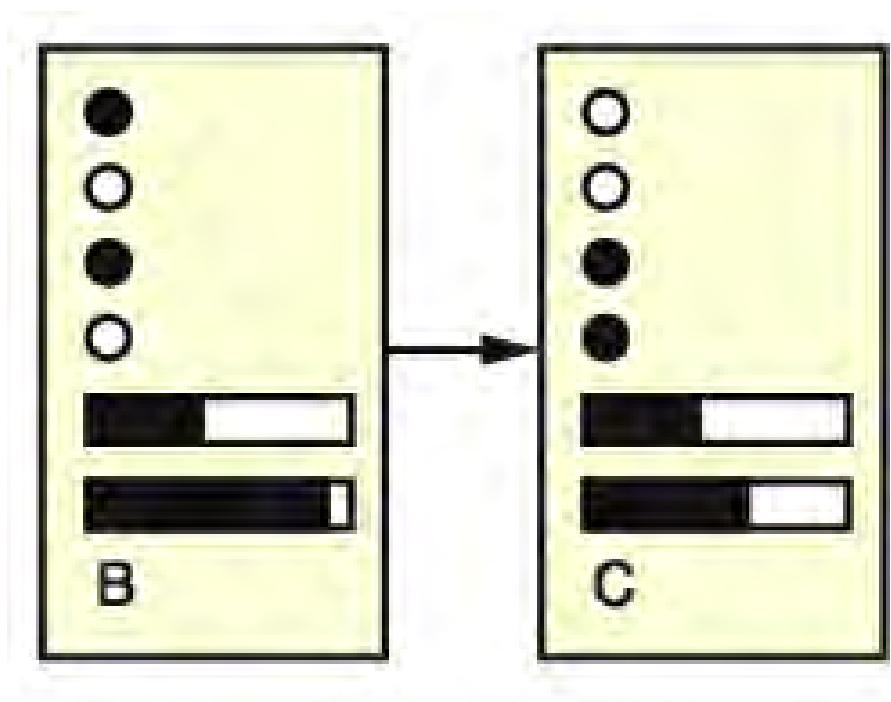
- A state is a **black box**, implemented as some data structure.
Recall [atomic representation](#).
- A goal test is a function over the states.



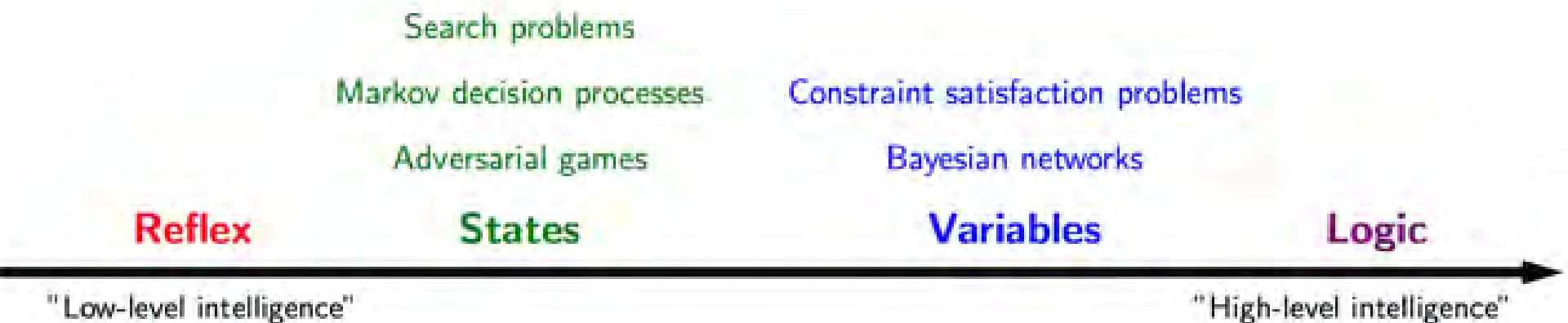
CSPs definition

- **CSPs problems:**

- A state: defined by variables X_i with values from domain D_i . Recall [factored representation](#).
- A goal test is a **set of constraints** specifying **allowable combinations** of values for subsets of variables.



CSPs definition



Credit: Courtesy Percy Liang

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
 - A set of **variables**, $X = \{X_1, X_2, \dots X_n\}$
 - A set of **domains** for each variable: $D = \{D_1, D_2, \dots D_n\}$
 - A set of **constraints** C that specify allowable combinations of values.

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
 - A set of **variables**, $X = \{X_1, X_2, \dots X_n\}$
 - A set of **domains** for each variable: $D = \{D_1, D_2, \dots D_n\}$
 - A set of **constraints** C that specify allowable combinations of values.
- Solving the CSP: **finding the assignment(s)** that **satisfy all constraints**.
- Concepts: problem formalization, backtracking search, arc consistency, etc.

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
 - A set of **variables**, $X = \{X_1, X_2, \dots X_n\}$
 - A set of **domains** for each variable: $D = \{D_1, D_2, \dots D_n\}$
 - A set of **constraints** C that specify allowable combinations of values.
- Solving the CSP: **finding the assignment(s)** that **satisfy all constraints**.
- Concepts: problem formalization, backtracking search, arc consistency, etc.
- We call a solution, a **consistent assignment**.

Example: Map coloring



Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$

Example: Map coloring



Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$

Domains: $D_i = \{\text{red, green, blue}\}$

Example: Map coloring



Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$

Domains: $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colors;

Example: Map coloring



Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$

Domains: $D_i = \{\text{red, green, blue}\}$

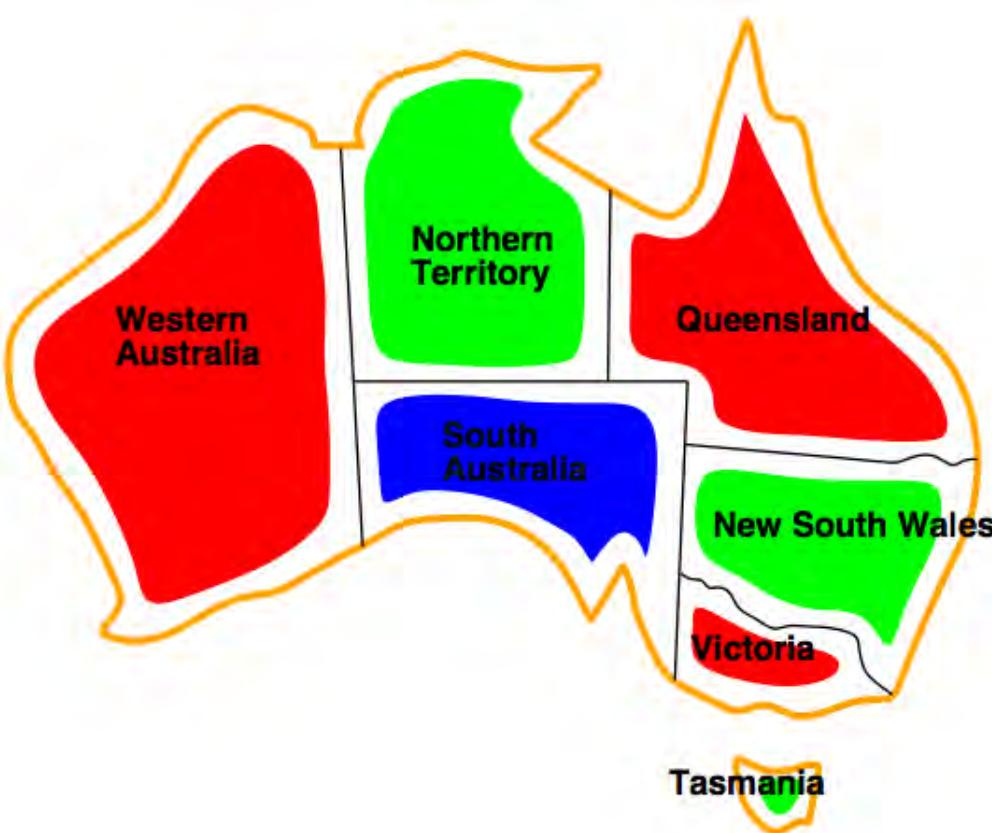
Constraints: adjacent regions must have different colors;

e.g., $\text{WA} \neq \text{NT}$ or $(\text{WA}, \text{NT}) \in \{(\text{red, green}), (\text{red, blue}), \text{etc..}\}$

Example: Map coloring



Example: Map coloring



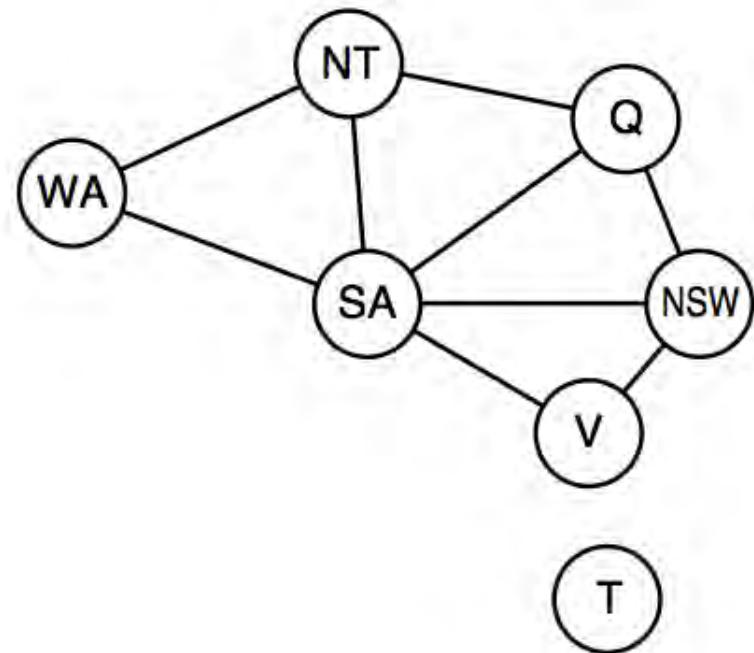
Example:

{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

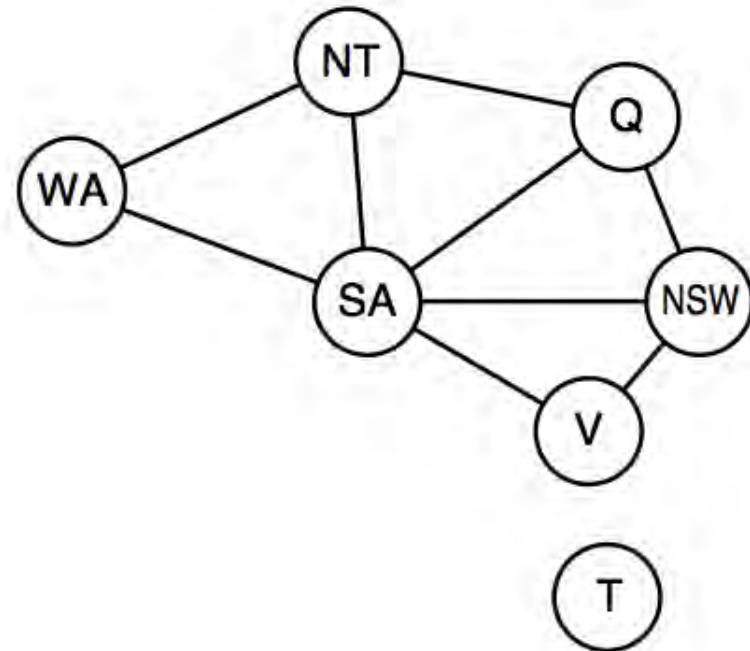
Real-world CSPs

- Assignment problems, e.g., who teaches what class?
- Timetabling problems, e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor planning
- Notice that many real-world problems involve real-valued variables

Constraint graph

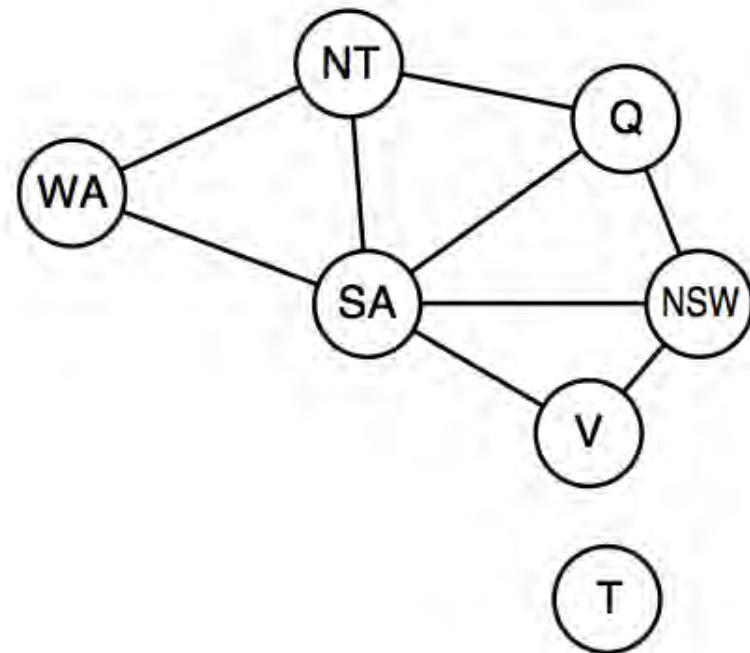


Constraint graph



Binary CSP: each constraint relates at most two variables
Constraint graph: nodes are variables, arcs show constraints

Constraint graph



Binary CSP: each constraint relates at most two variables
Constraint graph: nodes are variables, arcs show constraints

CSP algorithms: use the graph structure to speed up search.
E.g., Tasmania is an independent subproblem!

Varieties of variables

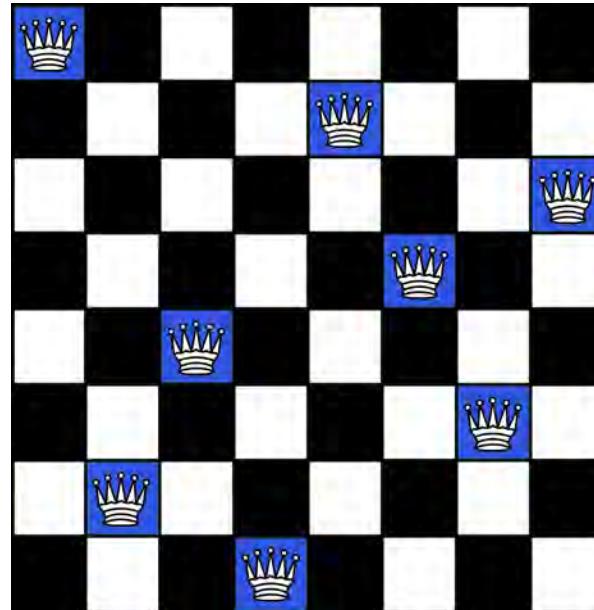
- **Discrete variables:**
 - Finite domains:
 - * assume n variables, d values, then the number of complete assignments is $O(d^n)$.
 - * e.g., map coloring, 8-queens problem
 - Infinite domains (integers, strings, etc.):
 - * need to use a constraint language,
 - * e.g., job scheduling. $T_1 + d \leq T_2$.
- **Continuous variables:**
 - Common in operations research
 - Linear programming problems with linear or non linear equalities

Varieties of constraints

- **Unary constraints:** involve a single variable e.g., SA \neq green
- **Binary constraints:** involve pairs of variables e.g., SA \neq WA
- **Global constraints:** involve 3 or more variables e.g., *Alldiff* that specifies that all variables must have different values (e.g., cryptarithmetic puzzles, Sudoku)
- **Preferences (soft constraints):**
 - Example: red is better than green
 - Often represented by a cost for each variable assignment
 - constrained optimization problems

Example: 8-queen

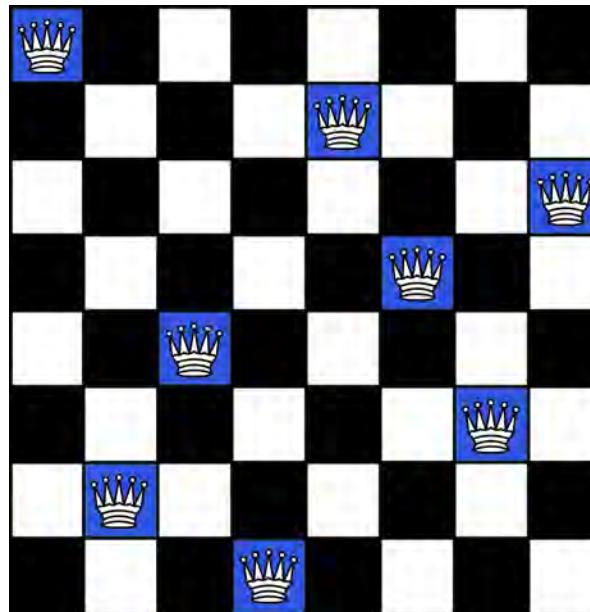
8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.



Problem formalization:

Example: 8-queen

8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.

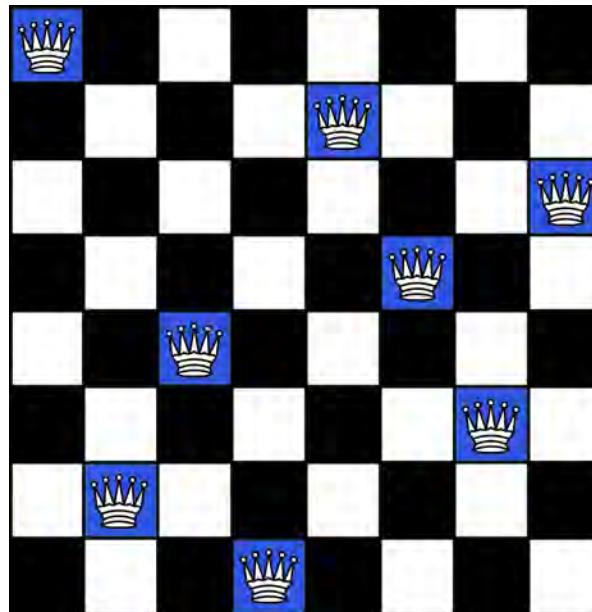


Problem formalization 1:

- One variable per queen, Q_1, Q_2, \dots, Q_8 .
- Each variable could have a value between 1 and 64.
- Solution: $Q_1 = 1, Q_2 = 13, Q_3 = 24, \dots, Q_8 = 60$.

Example: 8-queen

8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.

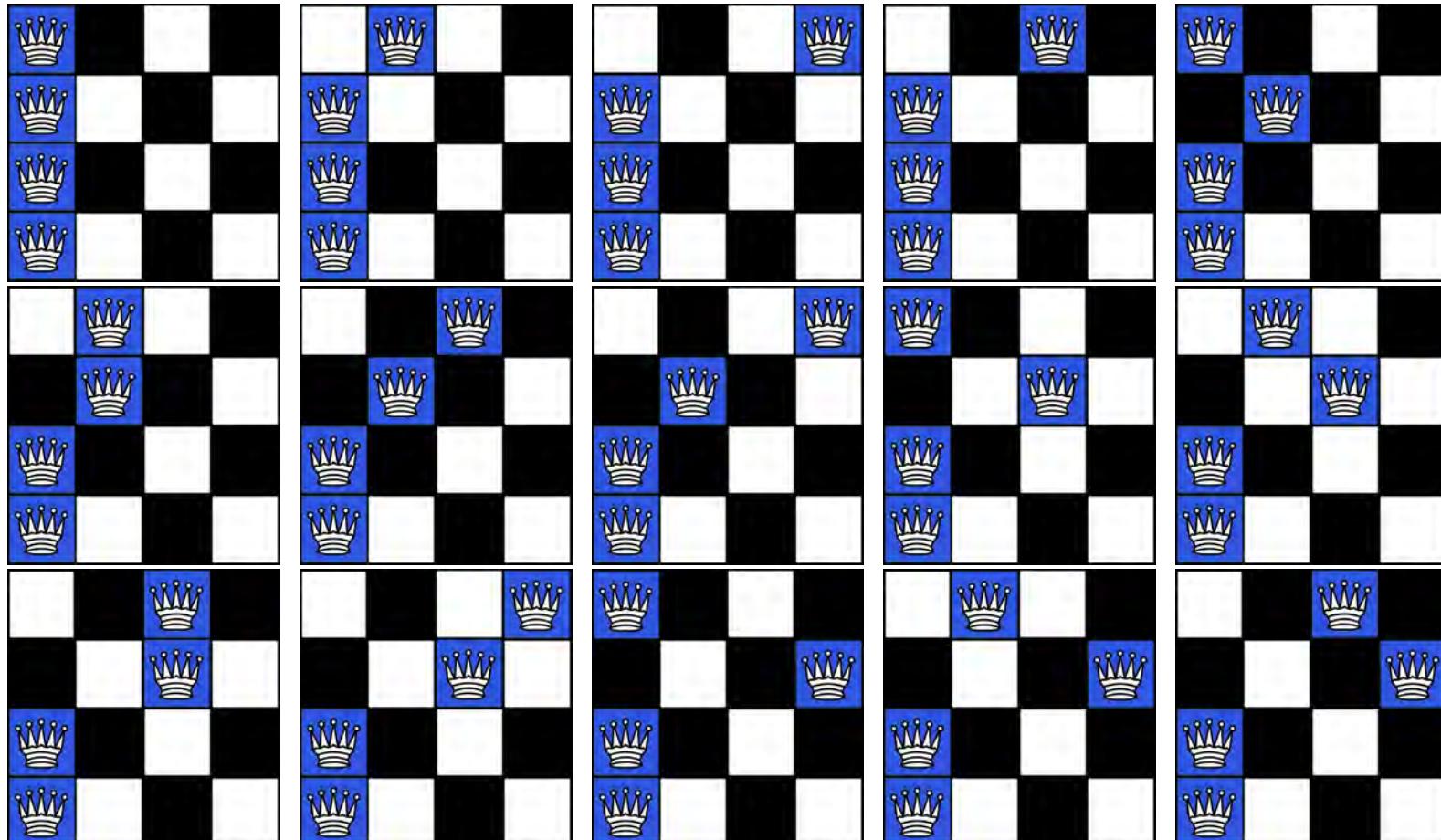


Problem formalization 2:

- One variable per queen, Q_1, Q_2, \dots, Q_8 .
- Each variable could have a value between 1 and 8 (columns).
- Solution: $Q_1 = 1, Q_2 = 7, Q_3 = 5, \dots, Q_8 = 3$.

Brute force?

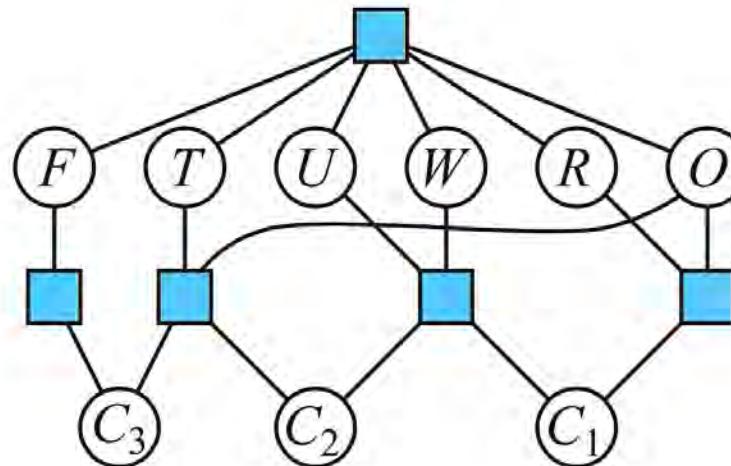
Should we simply generate and test all configurations?



...

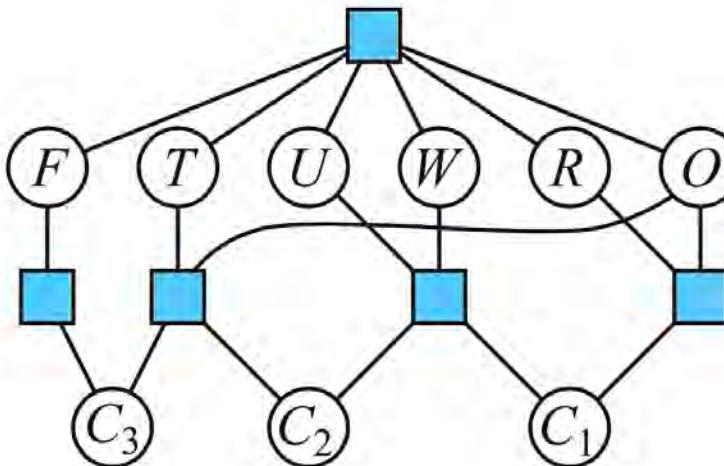
Example Cryptarithmetic

$$\begin{array}{r} T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$



Example Cryptarithmetic

$$\begin{array}{r} T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$



Variables: $X = \{F, T, U, W, R, O, C_1, C_2, C_3\}$

Domain: $D = \{0, 1, 2, \dots, 9\}$

Constraints:

- Alldiff(F, T, U, W, R, O)
- $T \neq 0, F \neq 0$
- $O + O = R + 10 * C_1$
- $C_1 + W + W = U + 10 * C_2$
- $C_2 + T + T = O + 10 * C_3$
- $C_3 = F$

Solving CSPs

IMPORTANT

- **State-space search algorithms:** search!
- **CSP Algorithms:** Algorithm can do two things:
 - **Search:** choose a new variable assignment from many possibilities
 - **Inference:** constraint propagation, use the constraints to spread the word: reduce the number of values for a variable which will reduce the legal values of other variables etc.
- As a preprocessing step, constraint propagation can sometimes solve the problem entirely without search.
- Constraint propagation can be intertwined with search.

Solving CSPs

- **BFS:** Develop the complete tree
- **DFS:** Fine but time consuming
- **BTS: Backtracking search** is the basic uninformed search for CSPs. It's a DFS s.t.
 1. Assign one variable at a time: assignments are commutative. e.g., (WA=red, NT=green) is same as (NT=green, WA=red)
 2. Check constraints on the go: consider values that do not conflict with previous assignments.

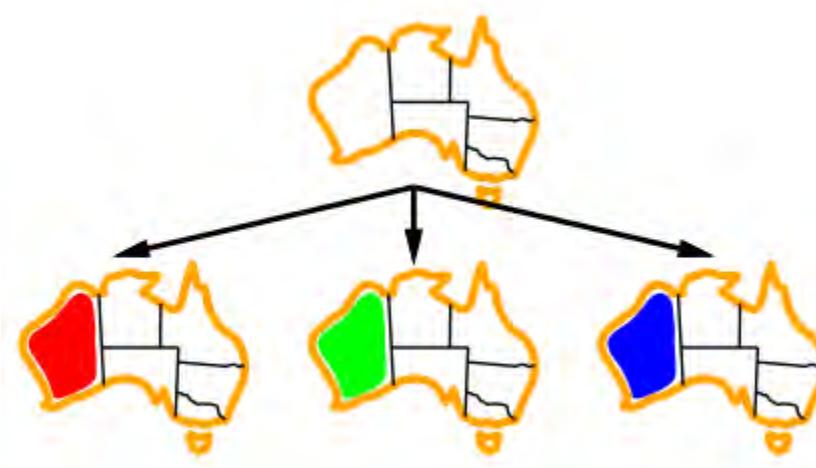
Solving CSPs

- **Initial state:** empty assignment {}
- **States:** are partial assignments
- **Successor function:** assign a value to an unassigned variable
- **Goal test:** the current assignment is complete and satisfies all constraints

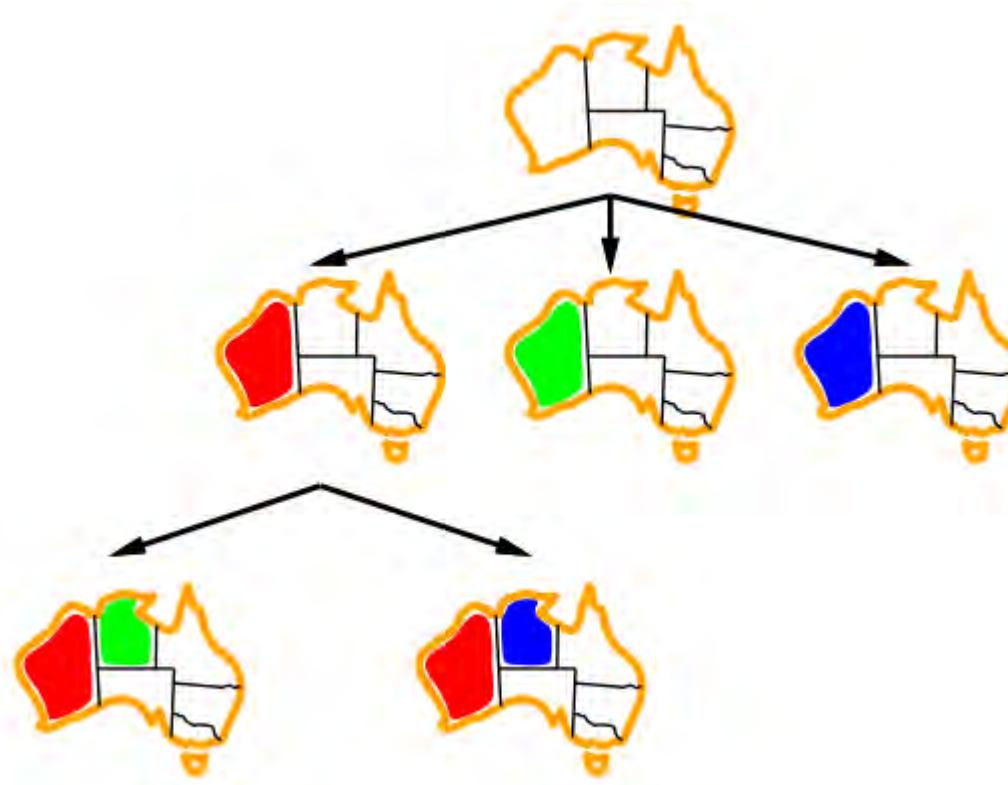
Backtracking search



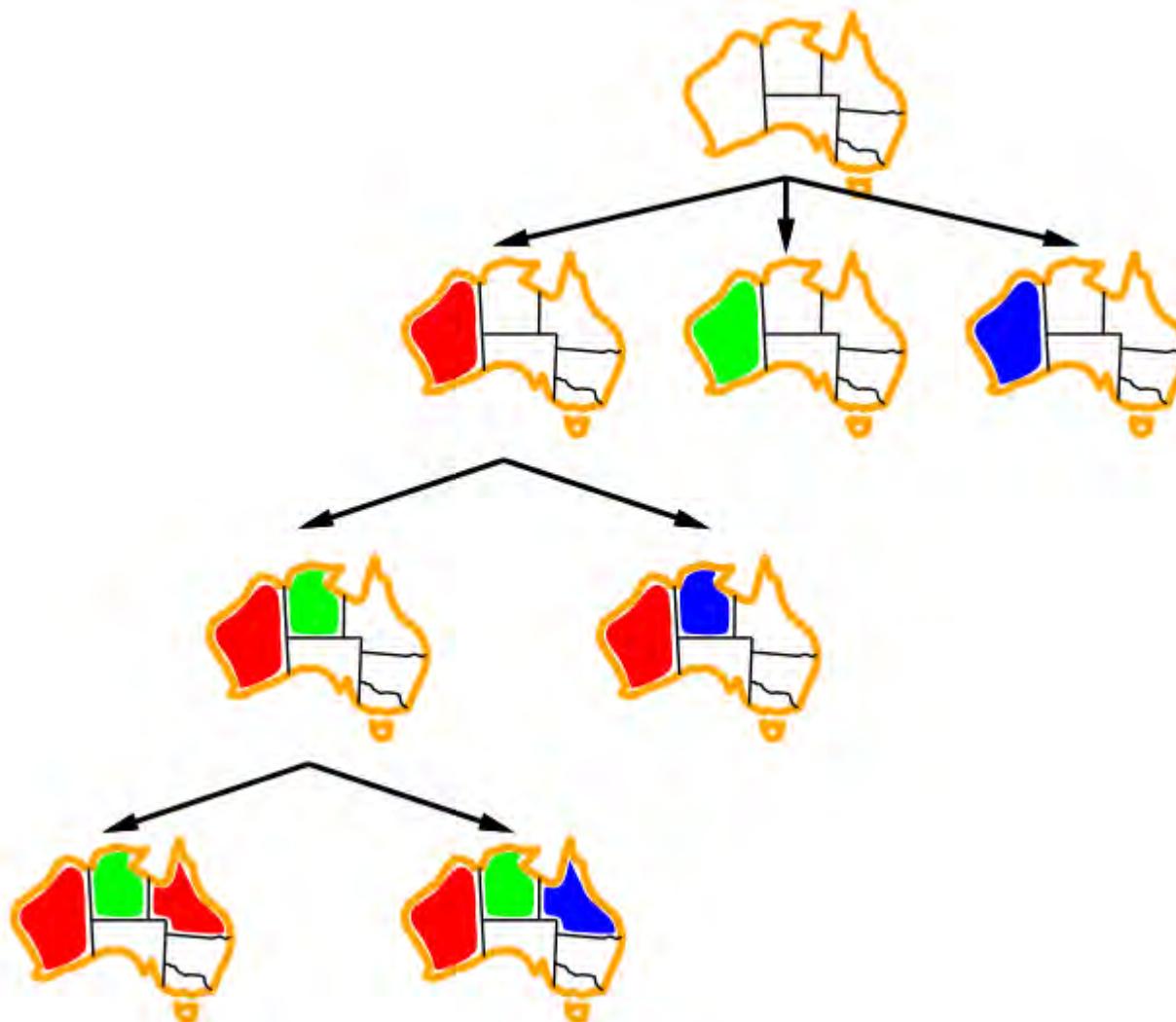
Backtracking search



Backtracking search



Backtracking search



Improving BTS

Heuristics are back!

1. Which variable should be assigned next?

Improving BTS

Heuristics are back!

- 1. Which variable should be assigned next?**
- 2. In what order should its values be tried?**

Improving BTS

Heuristics are back!

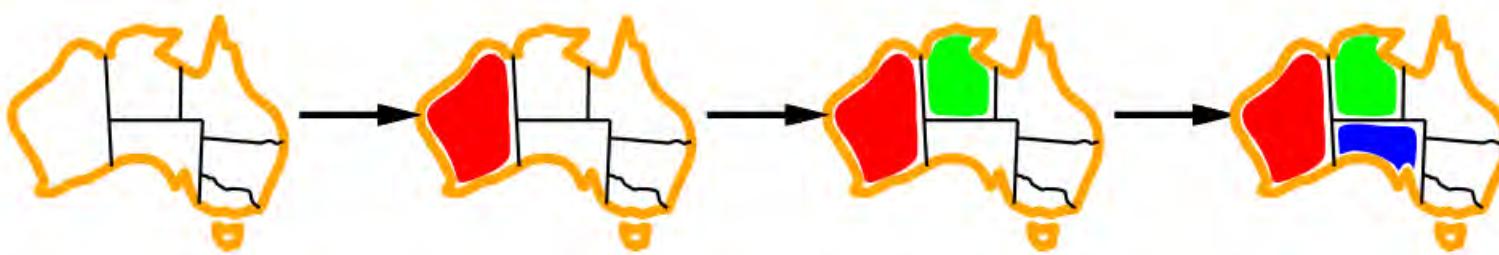
- 1. Which variable should be assigned next?**
- 2. In what order should its values be tried?**
- 3. Can we detect inevitable failure early?**

Minimum Remaining Values

1. Which variable should be assigned next?



- **MRV**: Choose the variable with the fewest legal values in its domain



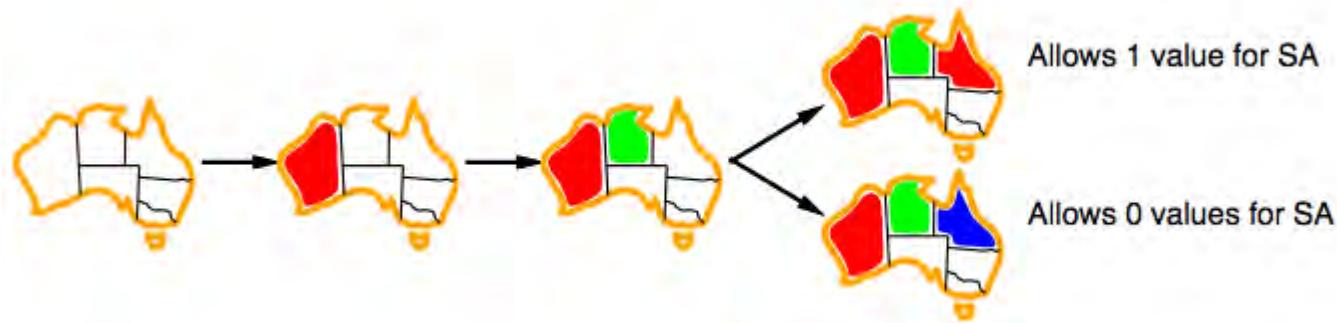
Pick the hardest!

Least constraining value

2. In what order should its values be tried?



- **LCV:** Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables



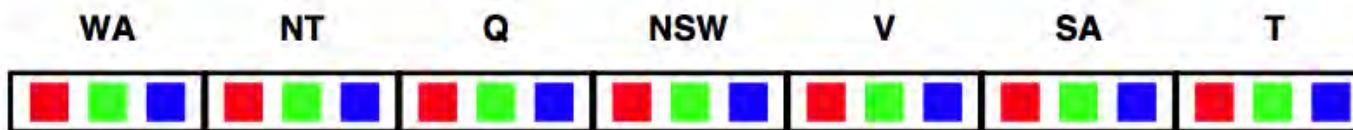
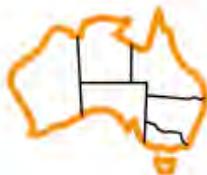
Pick the ones that are likely to work!

Forward checking

3. Can we detect inevitable failure early?



- **FC:** Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.

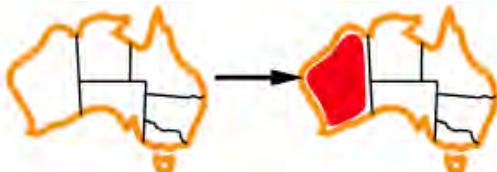


Forward checking

3. Can we detect inevitable failure early?



- **FC:** Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.



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

Forward checking

3. Can we detect inevitable failure early?



- **FC:** Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.



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

Forward checking

3. Can we detect inevitable failure early?



- **FC:** Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.



WA	NT	Q	NSW	V	SA	T
█ red █ green █ blue						
█ red █	█ green █ blue	█ red █ 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 █		█ green █	█ red █	█ blue		█ red █ green █ blue

Backtracking search

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure  
  return BACKTRACK({}, csp)
```

```
function BACKTRACK(assignment, csp)  
  returns a solution, or failure  
    if assignment is complete then return assignment  
    var = SELECT_UNASSIGNED-VARIABLES(csp)  
    for each value in ORDER_DOMAIN_VALUES (var, assignment, csp)  
      if value is consistent with assignment then  
        add {var = value} to assignment  
        result = BACKTRACK(assignment, csp)  
        if result ≠ failure then return result  
        remove {var = value} from assignment  
    return failure
```

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

8		9	5		1	7	3	6
2		7		6	3			
1	6							
				9		4		7
	9		3		7		2	
7		6		8				
						6	3	
			9	3		5		2
5	3	2	6		4	8		9

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

8		9	5		1	7	3	6
2		7		6	3			
1	6							
				9		4		7
	9		3		7		2	
7		6		8				
						6	3	
			9	3		5		2
5	3	2	6		4	8		9

Variables: $V = \{A_1, \dots, A_9, B_1, \dots, B_9, \dots, I_1 \dots I_9\}$, $|V| = 81$.

Domain: $D = \{1, 2, \dots, 9\}$, the filled squares have a single value.

Constraints: 27 constraints

- Alldiff($A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9$)
...
- Alldiff($A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1$)
...
- Alldiff($A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3$)

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

8		9	5		1	7	3	6
2		7		6	3			
1	6							
			9		4			7
	9	3		7		2		
7	6	8						
						6	3	
		9	3		5			2
5	3	2	6		4	8		9

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

8		9	5		1	7	3	6
2		7		6	3			
1	6							
			9		4			7
	9		3	7		2		
7	6		8					
						6	3	
		9	3		5			2
5	3	2	6		4	8		9

- Naked doubles (triples): find two (three) cells in a 3x3 grid that have only the same candidates left, eliminate these two (three) values from all possible assignments in that box.
- Locked pair, Locked triples, etc.

Solving CSPs: Sudoku

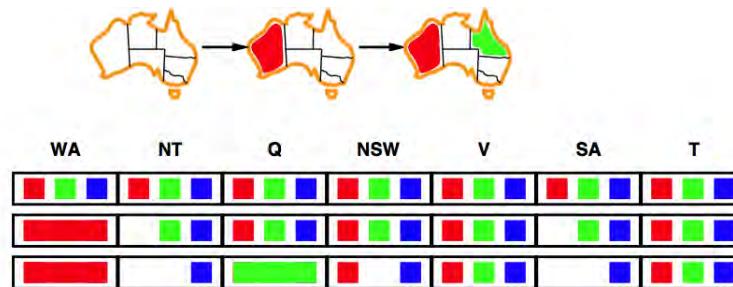
All 3x3 boxes, rows, columns, must contain all digits 1..9.

8		9	5		1	7	3	6
2		7		6	3			
1	6							
			9		4		7	
	9		3	7		2		
7	6		8					
					6	3		
		9	3		5		2	
5	3	2	6		4	8		9

8	4	9	5	2	1	7	3	6
2	5	7	8	6	3	9	1	4
1	6	3	7	4	9	2	5	8
3	2	5	1	9	6	4	8	7
4	9	8	3	5	7	6	2	1
7	1	6	4	8	2	3	9	5
9	8	4	2	7	5	1	6	3
6	7	1	9	3	8	5	4	2
5	3	2	6	1	4	8	7	9

Constraint propagation

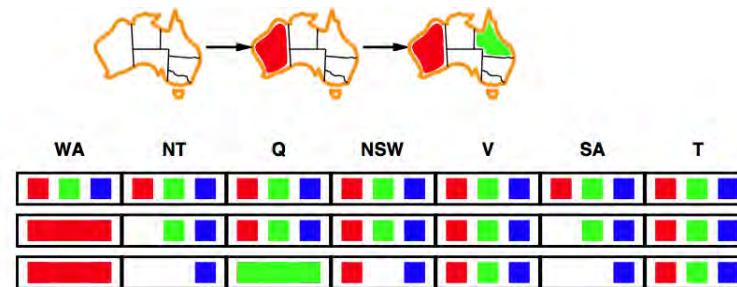
- Forward checking propagates information from assigned to unassigned variables.
- Observe:



- Forward checking does not check interaction between unassigned variables! Here SA and NT! (They both must be blue but can't be blue!).

Constraint propagation

- Forward checking propagates information from assigned to unassigned variables.
- Observe:



- Forward checking does not check interaction between unassigned variables! Here SA and NT! (They both must be blue but can't be blue!).
- Forward checking improves backtracking search but does not look very far in the future, hence does not detect all failures.
- We use constraint propagation, reasoning from constraint to constraint. e.g., arc consistency test.

Types of Consistency

- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $\text{Domain}(X_i)$ satisfy all unary constraints.

Types of Consistency

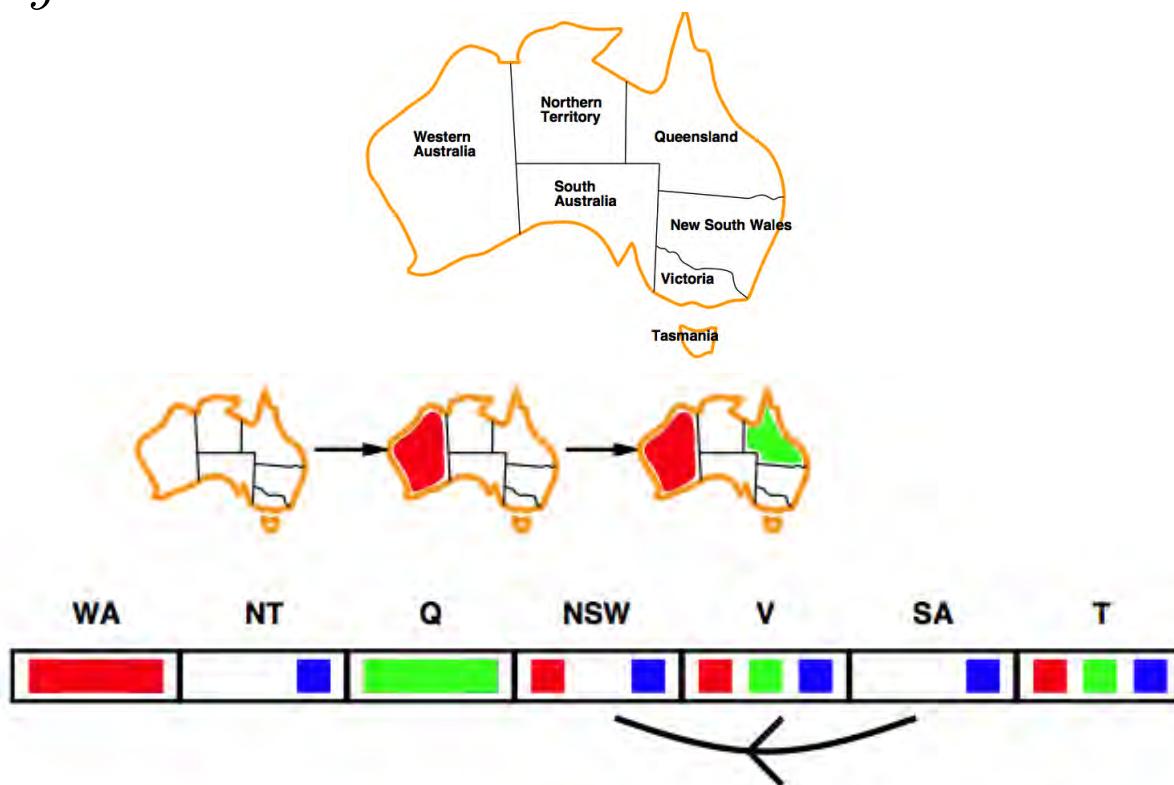
- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $\text{Domain}(X_i)$ satisfy all unary constraints.
- **Arc-consistency** (binary constraints): $X \rightarrow Y$ is arc-consistent if and only if every value x of X is consistent with some value y of Y .

Types of Consistency

- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $\text{Domain}(X_i)$ satisfy all unary constraints.
- **Arc-consistency** (binary constraints): $X \rightarrow Y$ is arc-consistent if and only if every value x of X is consistent with some value y of Y .
- **Path-consistency** (n-ary constraints): generalizes arc-consistency from binary to multiple constraints.
- **Note:** It is always possible to transform all n-ary constraints into binary constraints. Often, CSP solvers are designed to work with binary constraints.

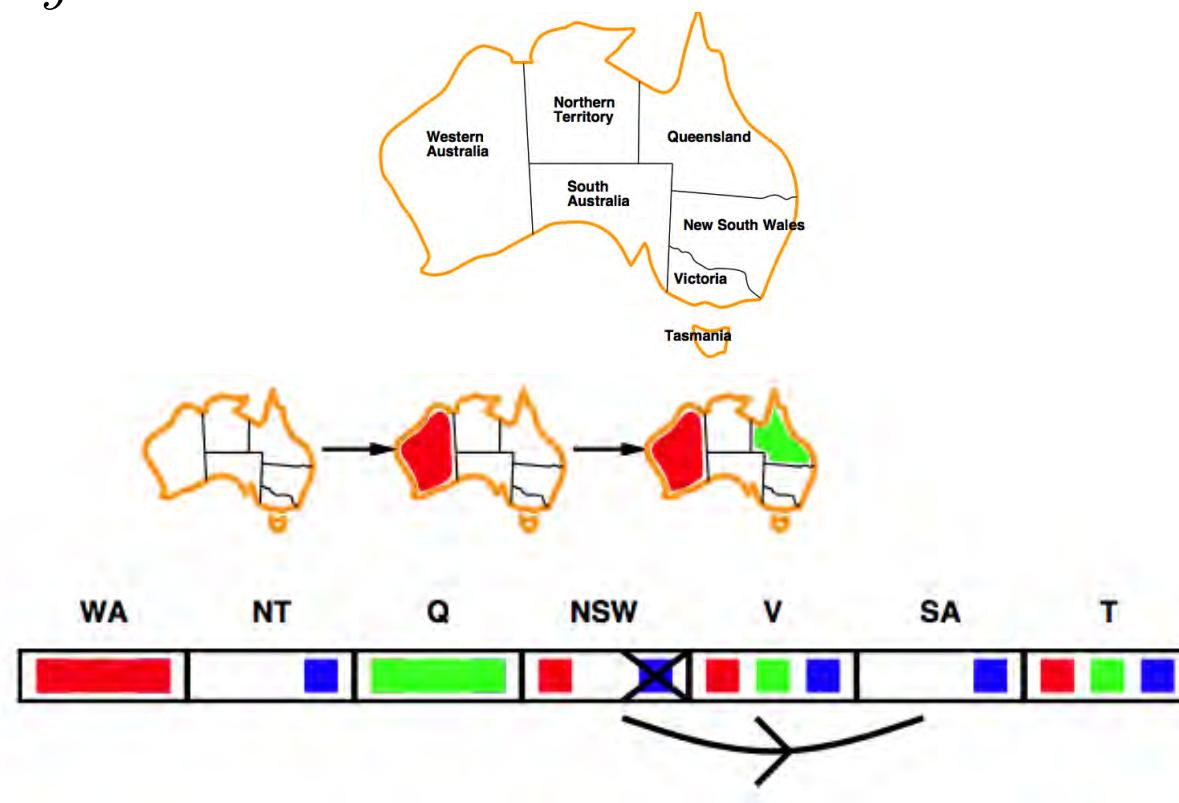
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X , there is some allowed y .



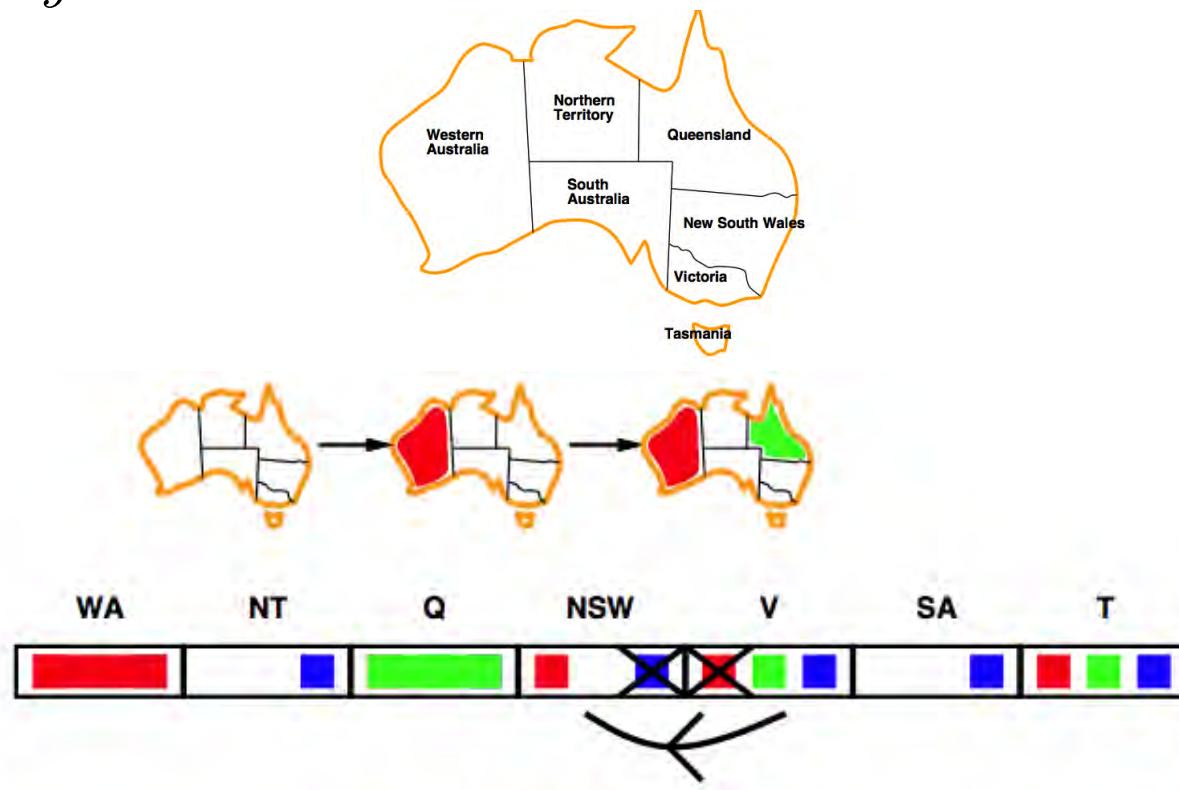
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X , there is some allowed y .



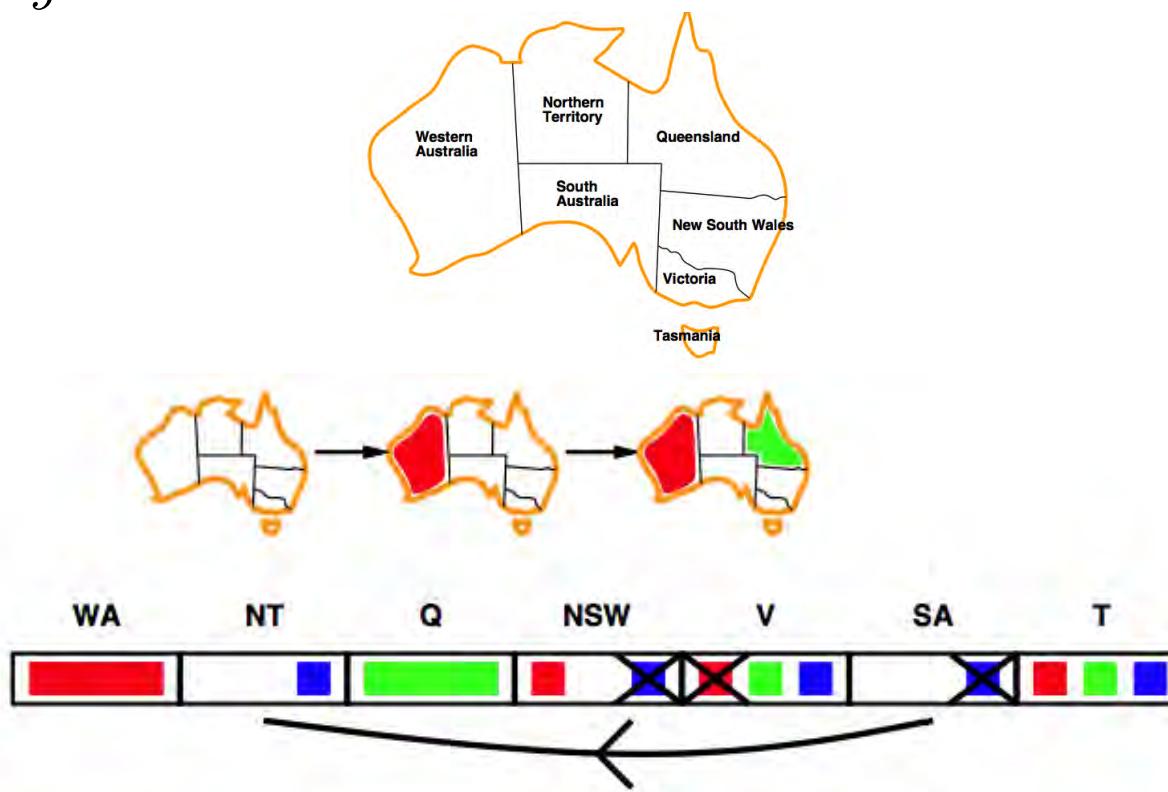
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X , there is some allowed y .



Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X , there is some allowed y .



Arc consistency

Algorithm that makes a CSP arc-consistent!

function AC-3(csp)

returns False if an inconsistency is found, True otherwise

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**

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

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

function REVISE(csp, X_i, X_j)

returns True iff we revise the domain of X_i

revised = False

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

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

 delete x from D_i

 revised = True

return revised

Complexity of AC-3

- Let n be the number of variables, and d be the domain size.
- If every node (variable) is connected to the rest of the variables, then we have $n * (n - 1)$ arcs (constraints) $\rightarrow O(n^2)$
- Each arc can be inserted in the queue d times $\rightarrow O(d)$
- Checking the consistency of an arc costs $\rightarrow O(d^2)$.
- Overall complexity is $O(n^2d^3)$.

Backtracking w/ inference

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK( {}, csp)
```

```
function BACKTRACK(assignment, csp)
```

returns a solution, or failure

```
  if assignment is complete then return assignment
```

```
  var = SELECT-UNASSIGNED-VARIABLES(csp)
```

```
  for each value in ORDER-DOMAIN-VALUES (var, assignment, csp)
```

```
    if value is consistent with assignment then
```

```
      add {var = value} to assignment
```

```
      inferences = INFERENCE(csp, var, value)
```

```
      if inferences ≠ failure then add inferences to assignment
```

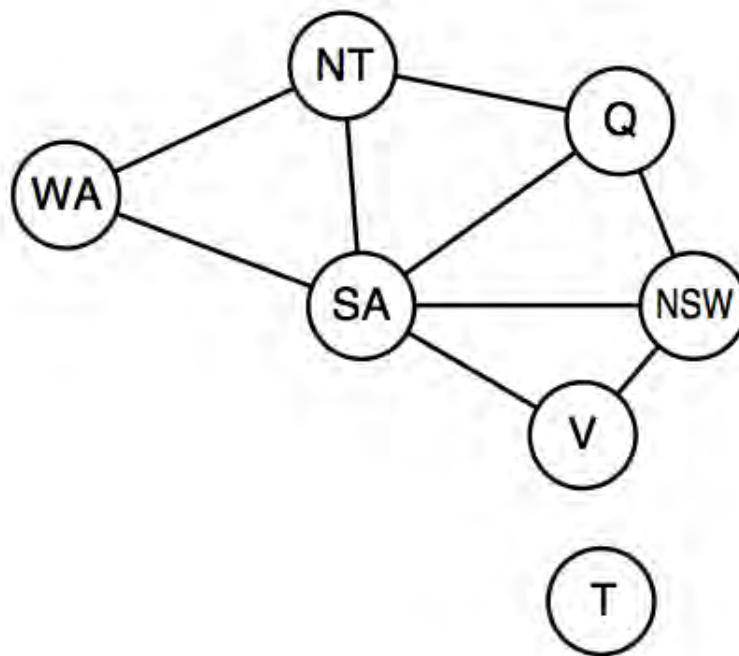
```
      result = BACKTRACK(assignment, csp)
```

```
      if result ≠ failure then return result
```

```
      remove {var = value} and inferences from assignment
```

```
  return failure
```

Problem structure



- Idea: Leverage the problem structure to make the search more efficient.
- Example: Tasmania is an independent problem.
- Identify the connected component of a graph constraint.
- Work on independent subproblems.

Problem structure

Complexity:

- Let d be the size of the domain and n be the number of variables.
- Time complexity for BTS is $O(d^n)$.
- Suppose we decompose into subproblems, with c variables per subproblem.
- Then we have $\frac{n}{c}$ subproblems.
- c variables per subproblem takes $O(d^c)$.
- The total for all subproblems takes $O(\frac{n}{c}d^c)$ in the worst case.

Problem structure

Example:

- Assume $n = 80$, $d = 2$.
- Assume we can decompose into 4 subproblems with $c = 20$.
- Assume processing at 10 million nodes per second.
- Without decomposition of the problem we need:

$$2^{80} = 1.2 \times 10^{24}$$

3.83 million years!

- With decomposition of the problem we need:

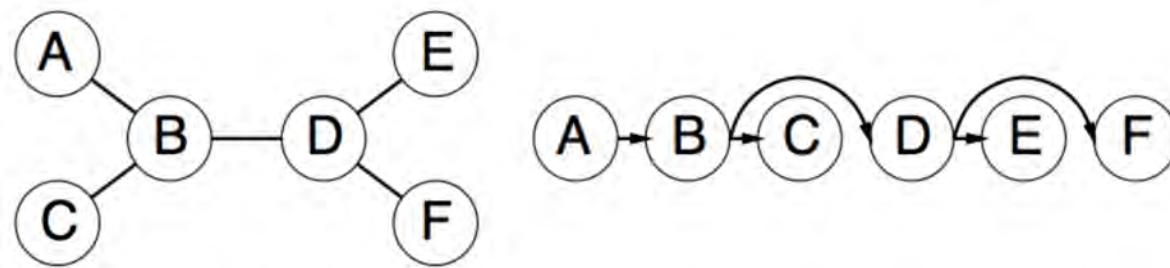
$$4 \times 2^{20} = 4.2 \times 10^6$$

0.4 seconds!

Problem structure

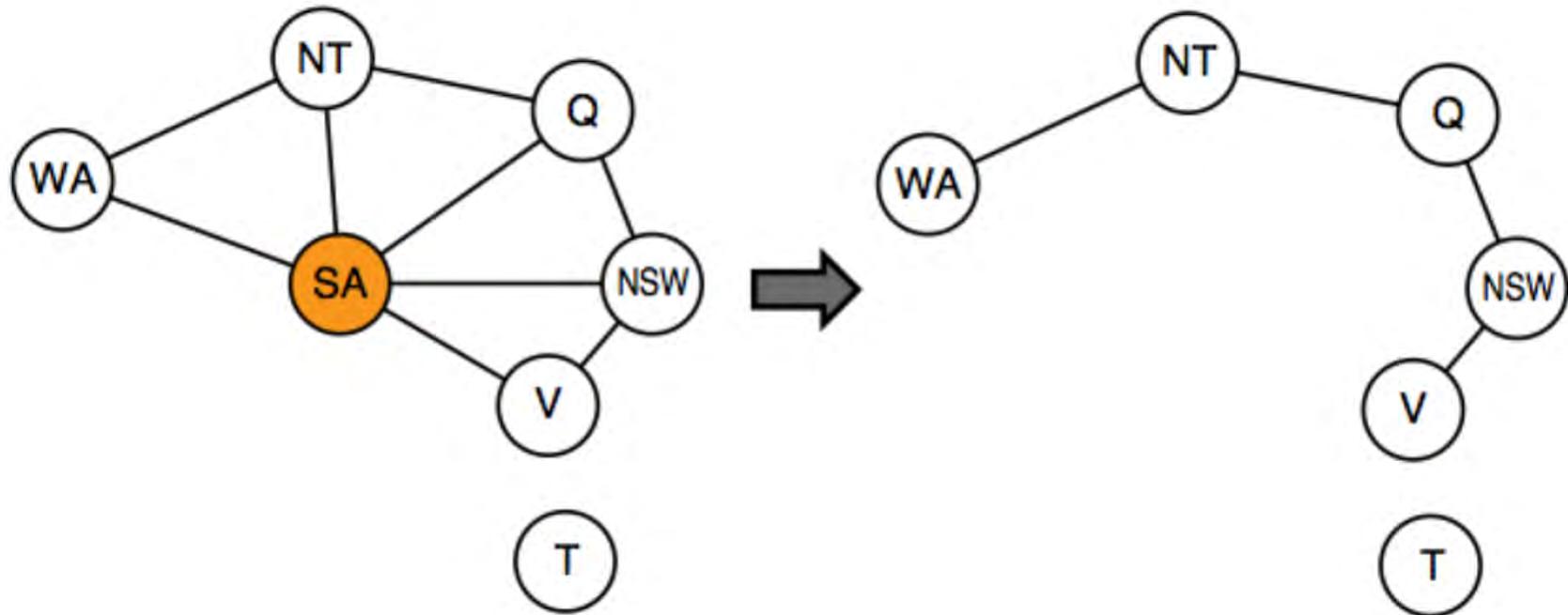
- Turning a problem into independent subproblems is not always possible.
- Can we leverage other graph structures?
- Yes, if the graph is tree-structured or nearly tree-structured.
- A graph is a [tree](#) if any two variables are connected by [only one path](#).
- Idea: use DAC, Directed Arc Consistency
- A CSP is said to be [directed arc-consistent](#) under an ordering X_1, X_2, \dots, X_n IFF every X_i is arc-consistent with each X_j for $j > i$.

Problem structure



- First pick a variable to be the root.
- Do a **topological sorting**: choose an ordering of the variables s.t. each variable appears after its parent in the tree.
- For n nodes, we have $n - 1$ edges.
- Make the tree directed arc-consistent takes $O(n)$
- Each consistency check takes up to $O(d^2)$ (compare d possible values for 2 variables).
- The CSP can be solved in $O(nd^2)$

Nearly tree-structured CSPs



- Assign a variable or a set of variables and prune all the neighbors domains.
- This will turn the constraint graph into a tree :)
- There are other tricks to explore, have fun!

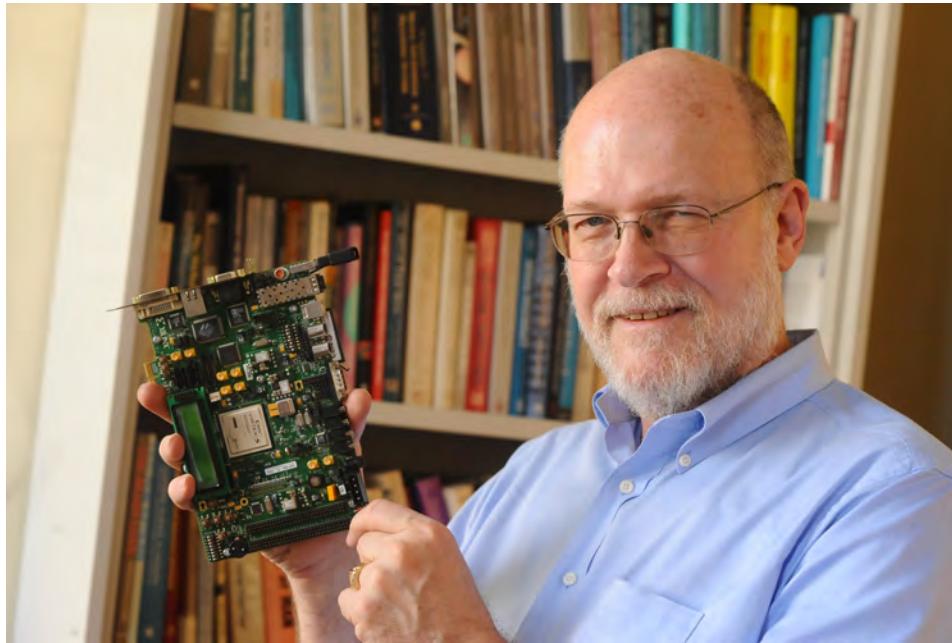
Summary

- CSPs are a special kind of search problems:
 - states defined by values of a fixed set of variables
 - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help
- Forward checking prevents assignments that guarantee later failure

Summary

- Constraint propagation (e.g., arc consistency) is an important mechanism in CSPs.
- It does additional work to constrain values and detect inconsistencies.
- Tree-structured CSPs can be solved in linear time
- Further exploration: How can local search be used for CSPs?
- **The power of CSPs: domain-independent, that is you only need to define the problem and then use a solver that implements CSPs mechanisms.**
- Play with CSP solver? Try <http://aispace.org/constraint/>.

David L. Waltz



**David L. Waltz
28 May 1943 – 22 March 2012**

CCLS founder and leader 2003-2012

David L. Waltz was a computer scientist who made significant contributions in several areas of artificial intelligence, including constraint satisfaction, case-based reasoning and the application of massively parallel computation to AI problems.

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

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

Reinforcement Learning

Introduction

Reinforcement Learning

- Agent interacts and learns from a stochastic environment
- Science of sequential decision making
- Many faces of reinforcement learning
 - Optimal control (Engineering)
 - Dynamic Programming (Operations Research)
 - Reward systems (Neuro-science)
 - Classical/Operant Conditioning (Psychology)

Characteristics of Reinforcement Learning

- No supervisor, only reward *signals*
- Feedback is delayed
- Sequential decisions
- Actions effect observations (non i.i.d.)

Examples

- Automated vehicle control
 - An unmanned helicopter learning to fly and perform stunts
- Game playing
 - Playing backgammon, Atari breakout, Tetris, Tic Tac Toe
- Medical treatment planning
 - Planning a sequence of treatments based on the effect of past treatments
- Chat bots
 - Agent figuring out how to make a conversation

Markov Decision Process (MDP)

Markov Decision Processes (MDP)

- Sequential decisions in round rounds $t = 1, \dots, T$
- Important concepts
 - State
 - Action
 - Reward
- Markov property: Future is independent of the past given the current state

Markov Decision Processes (MDP)

- Starts at some initial state s_1
- In every round t , the agent
 - observes the current state s_t ,
 - take an action a_t , and then
 - observes a reward signal r_t
 - transitions to the next state s_{t+1}
- Markov Property:
 - $\Pr(s_{t+1} = s' | \text{history till time } t) = \Pr(s_t = s' | s_t = s, a_t = a) =: P_{s,a}(s')$
 - $E[r_t | \text{history till time } t] = E[r_t | s_t = s, a_t = a] =: R_{s,a}$

Markov Decision Processes (MDP)

- Goal: Maximize some form of cumulative reward
- Total reward in finite time T
 - maximize $\sum_{t=1}^T r_t$
- Infinite time average reward
 - maximize $\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T r_t$
- **Discounted sum of rewards**
 - maximize $r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{i-1} r_i + \dots$
 - where $\gamma < 1$

Summary

- Markov Decision Process (MDP) is a tuple (S, s_1, A, P, R)
- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix of dimension $S \times A \times S$

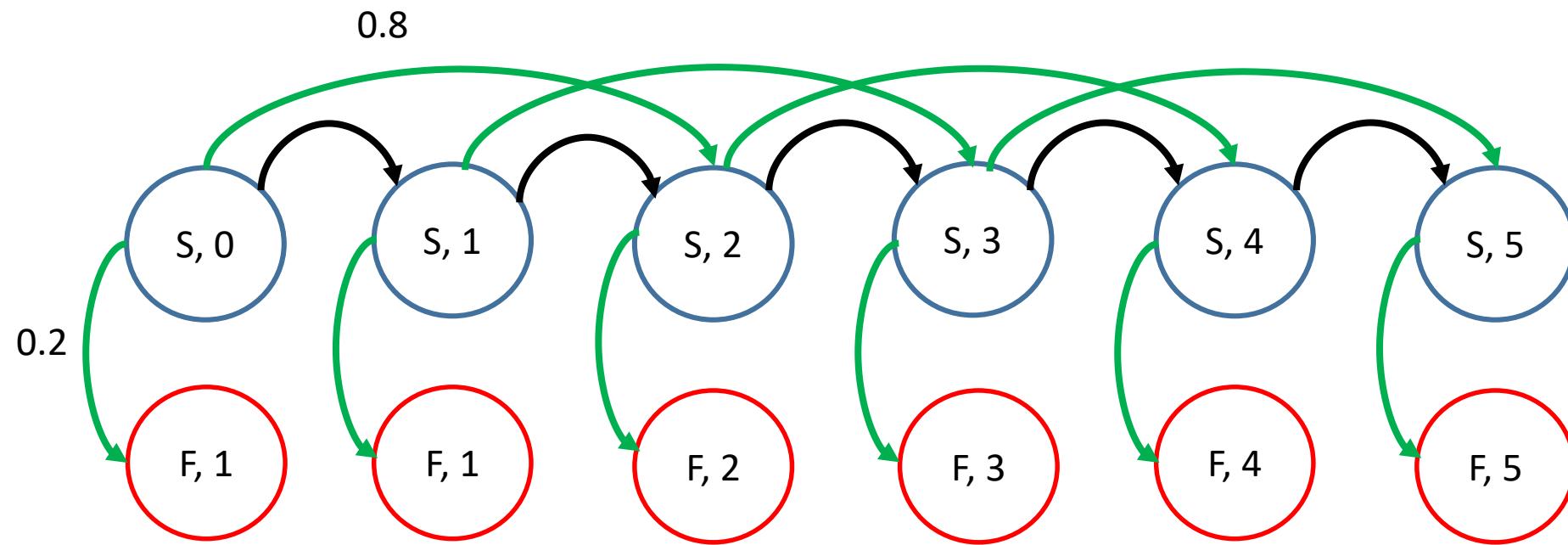
$$P_{s,a}(s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

- R is a reward function

$$R_{s,a} = Ex[r_t | s_t = s, a_t = a]$$

- Goal definition, discount factor $\gamma \in [0,1)$

Example



Markov Decision Processes

Finding an optimal policy: Value functions

Overview

- Markov Decision Process is a tuple (S, s_1, A, P, R)
- P is a state transition probability matrix of dimension $S \times A \times S$

$$P_{s,a}(s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

- R is a reward function

$$R_{s,a} = E[R_{t+1} | s_t = s, a_t = a]$$

- Goal:

Maximize expected discounted reward $E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1]$

where $r_t = R_{s_t, a_t}$, $\gamma \in [0, 1)$ is a discount factor

Policy

- A policy $\pi: S \rightarrow A$ is a mapping from state space to action space
- Following a stationary policy π means taking action $a_t = \pi(s_t)$ at all time steps t
- Theorem
For any discounted MDP, there always exists stationary policy π that is optimal

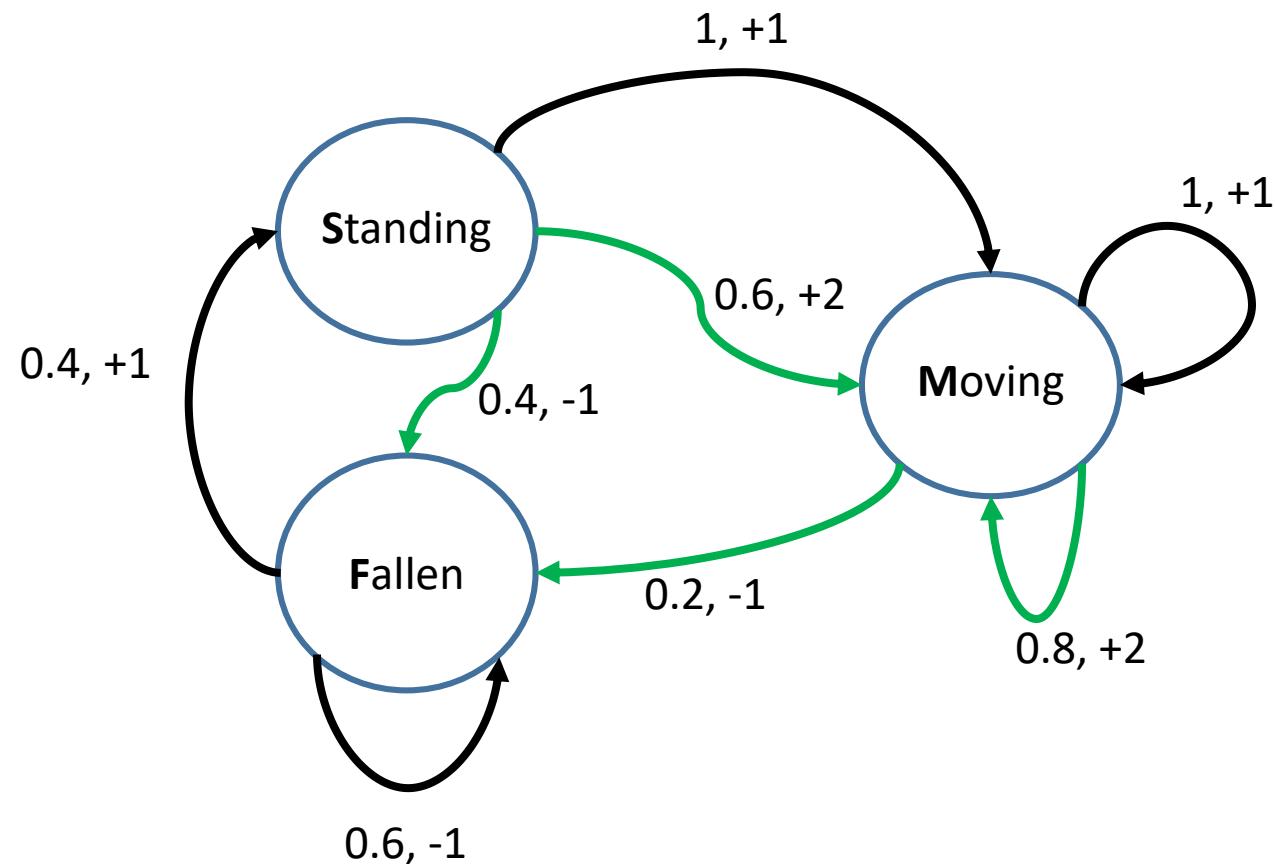
Value function

- Value function $v_\pi(s)$ of a policy π
 - expected reward starting from state s and then following the policy π

$$v_\pi(s) = E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s]$$

where $a_t = \pi(s_t)$, $E[r_t | s_t, a_t] = R_{s_t, a_t}$, $\Pr(\cdot | s_t, a_t) = P_{s_t, a_t}$

Example



Policy: slow action 1 (black) in *Fallen* state, fast action 2 (green) in *Standing* and *Moving* state

Bellman equations

- Value function can be decomposed into immediate reward plus discounted value function of the next state

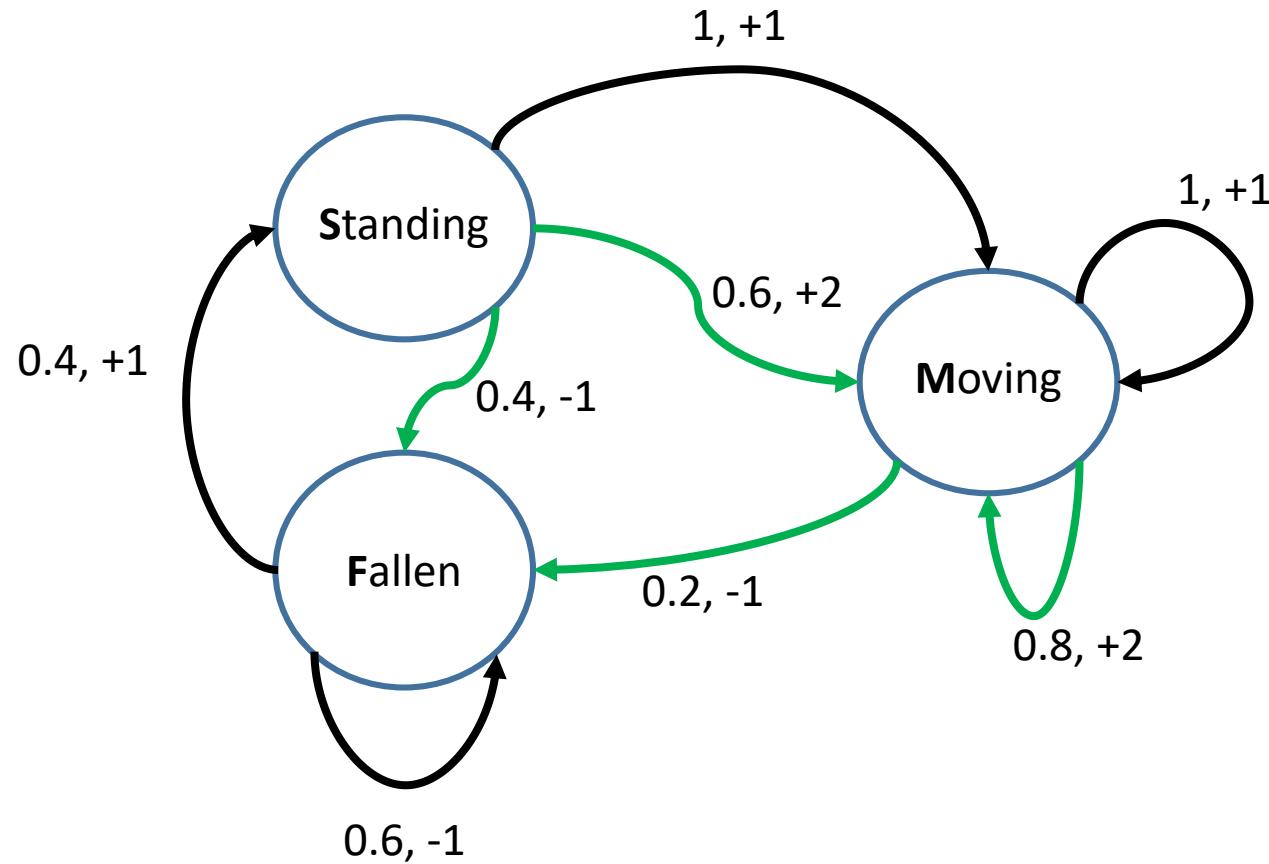
$$v_{\pi}(s) = R_{s,\pi(s)} + \gamma \sum_{s'} P_{s,\pi(s)}(s') v_{\pi}(s')$$

- Compact matrix notation

$$\boldsymbol{v}_{\pi} = \boldsymbol{r}_{\pi} + \gamma \boldsymbol{P}_{\pi} \boldsymbol{v}_{\pi}$$

$$\boldsymbol{v}_{\pi} = (\mathbf{I} - \gamma \boldsymbol{P}_{\pi})^{-1} \boldsymbol{r}_{\pi}$$

Example



Policy: slow action 1 (black) in *Fallen* state, fast action 2 (green) in *Standing* and *Moving* state

Markov Decision Processes

Finding an optimal policy: Iterative methods

Recap

- Value function $v_\pi(s)$ of a policy π

$$v_\pi(s) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | s_1 = s]$$

- Bellman equations

$$\boldsymbol{v}_\pi = \boldsymbol{r}_\pi + \gamma P_\pi \boldsymbol{v}_\pi$$

$$\boldsymbol{v}_\pi = (\mathbf{I} - \gamma P_\pi)^{-1} \boldsymbol{r}_\pi$$

Optimal Policy

- Optimal policy when starting in state s :

$$\operatorname{argmax}_{\pi} v_{\pi}(s)$$

Optimal Policy

- Define partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s$$

- Theorem

- There always exists a policy that is better than all other policies

$$\pi \geq \pi' \text{ for all } \pi'$$

Such a policy is called an optimal policy

- All optimal policies achieve the same value function $v_*(s)$ called the optimal value function

Bellman Optimality Equations

- Optimal value functions are recursively related by Bellman optimality equations

$$v_*(s) = \max_{a \in A} R_{s,a} + \gamma \sum_{s'} P_{s,a}(s') v_*(s')$$

- Matrix notation

$$\boldsymbol{v}_* = \max_{\pi} \boldsymbol{r}_{\pi} + \gamma P_{\pi} \boldsymbol{v}_*$$

- Optimal policy can be computed by solving Bellman equations

Solving the Bellman optimality equations

- No closed form solution in general
- Iterative solution methods
 - Policy iteration
 - Value Iteration

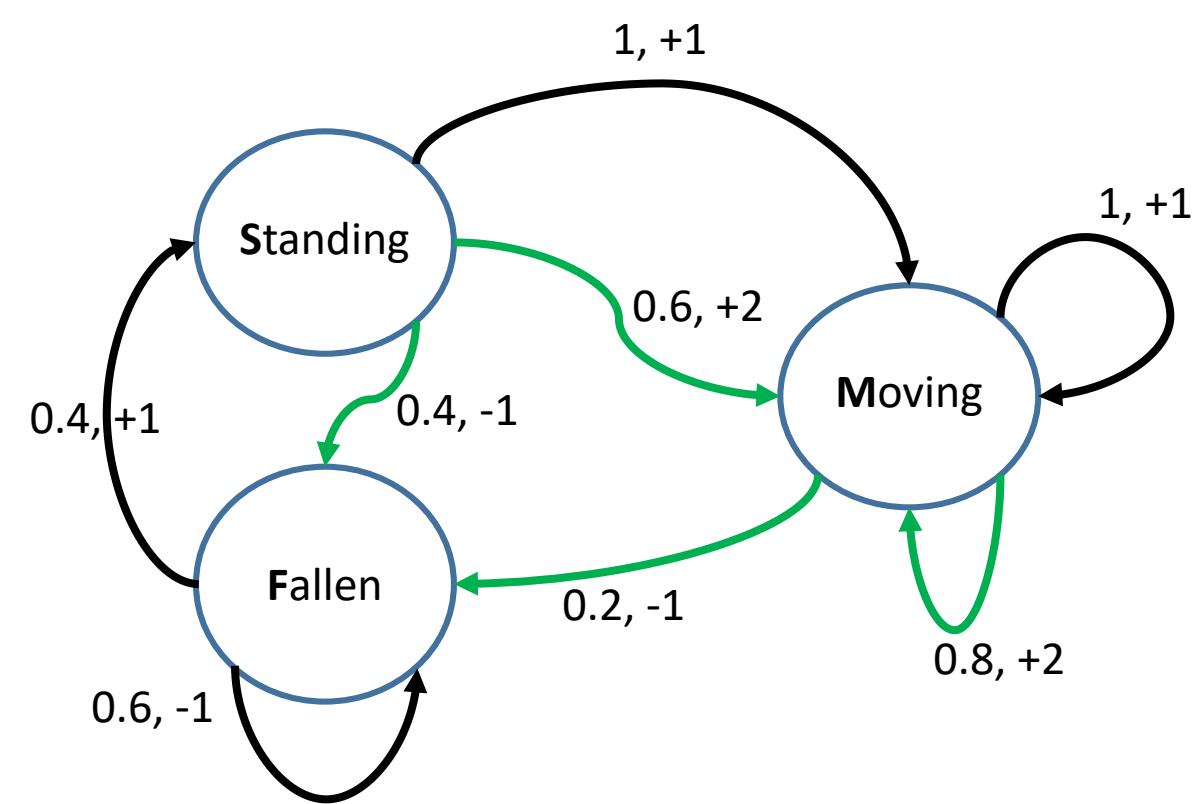
Policy Iteration

- Start with a random policy π

In every iteration,

- Evaluate the policy
 - Compute the value vector for $\mathbf{v}_\pi = (\mathbf{I} - \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi$
- Improve the policy
 - New policy: $\pi'(s) = \arg \max_a R_{s,a} + \gamma P_{s,a} \mathbf{v}_\pi$
 - Stop if no strict improvement ($\mathbf{v}_\pi = \mathbf{v}_{\pi'}$)

$$v_\pi(s) = \max_a R_{s,a} + \gamma P_{s,a} \mathbf{v}_\pi \quad , \forall s$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \gamma = 0.1$$

- Iteration 1

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

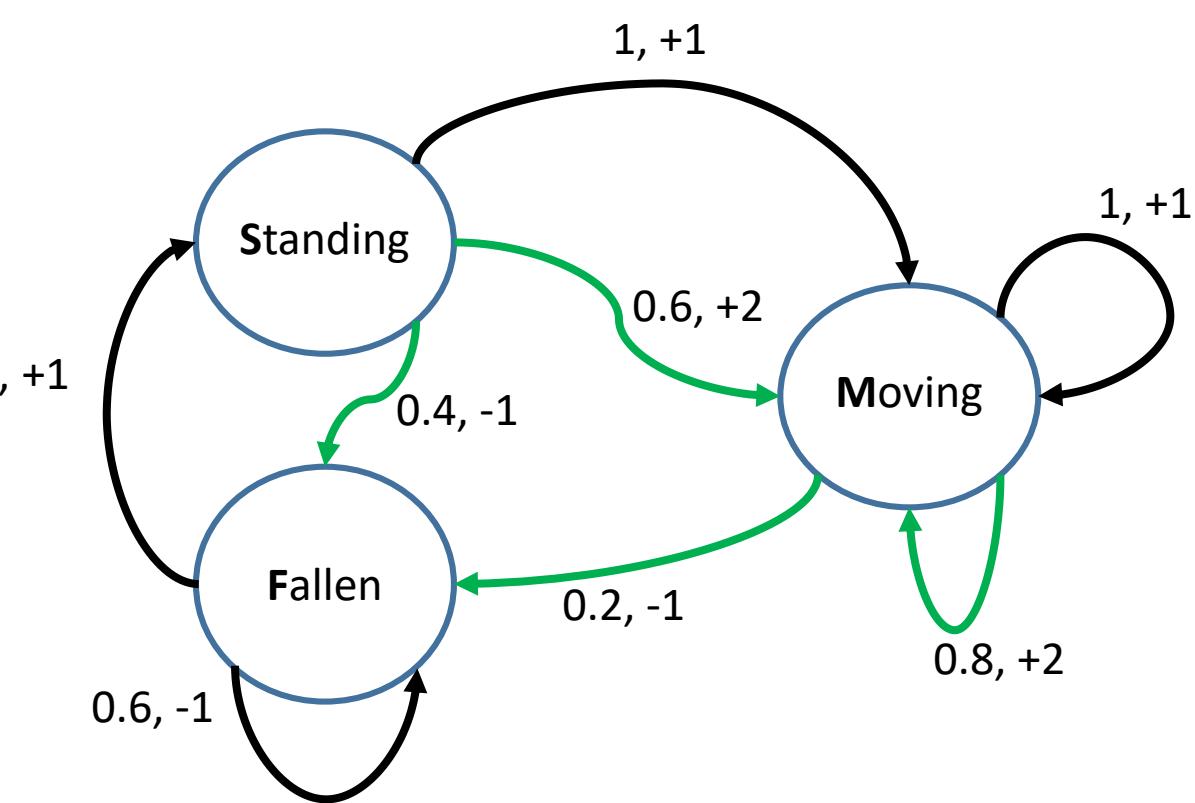
Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State Standing,

Slow Action: = 1.11

Fast Action: 0.8 +

$$0.1 [0.4 \quad 0 \quad 0.6] \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix} = 0.86$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- Iteration 1

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

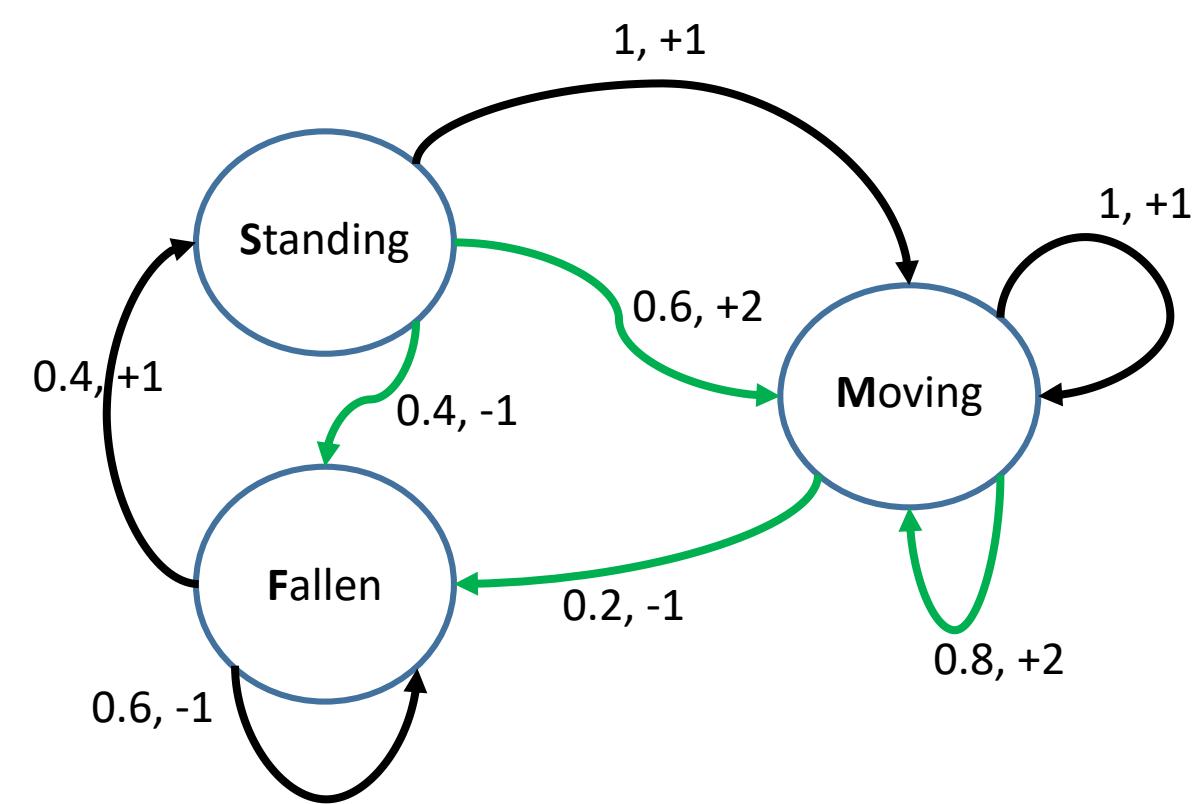
State *Standing*, slow action

State *Moving*

Slow Action: = 1.1111

Fast Action: 1.4 +

$$0.1 [0.2 \quad 0 \quad 0.8] \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix} \simeq 1.48$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- Iteration 1

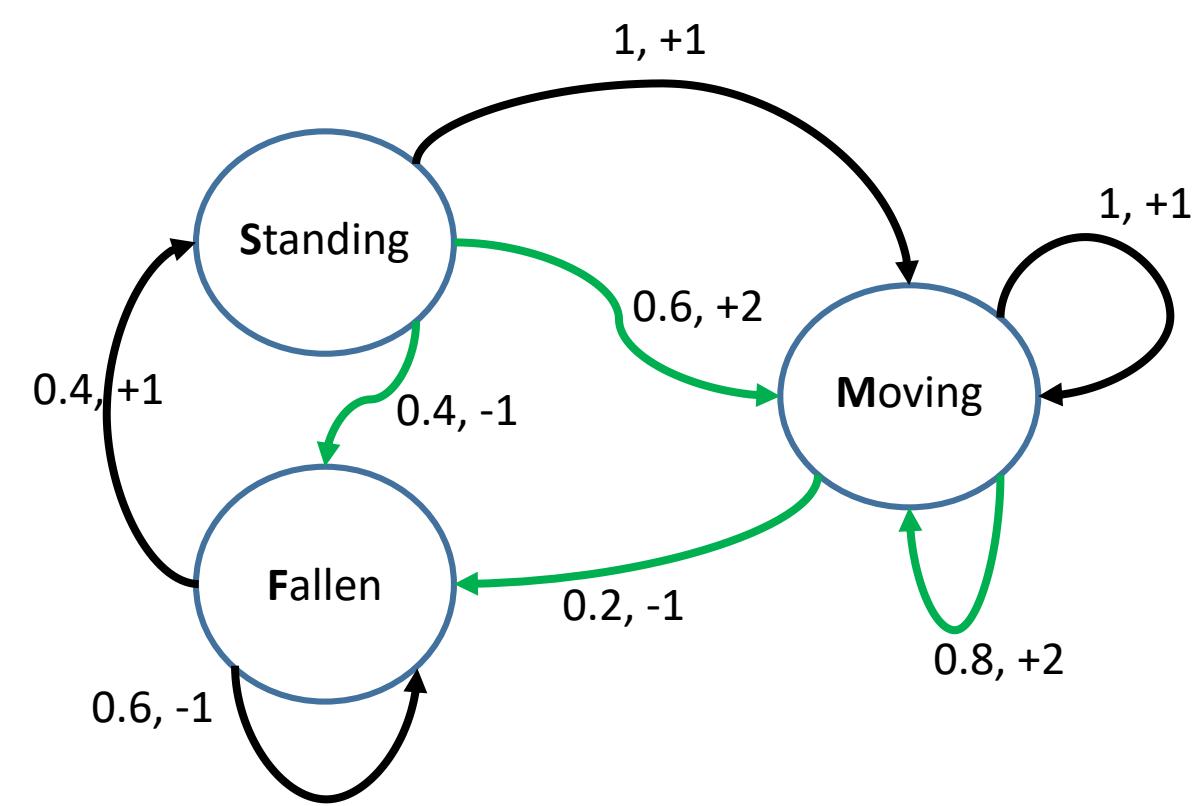
$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State Standing, SLOW action

State Moving, FAST action



New Policy: fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Improve policy:

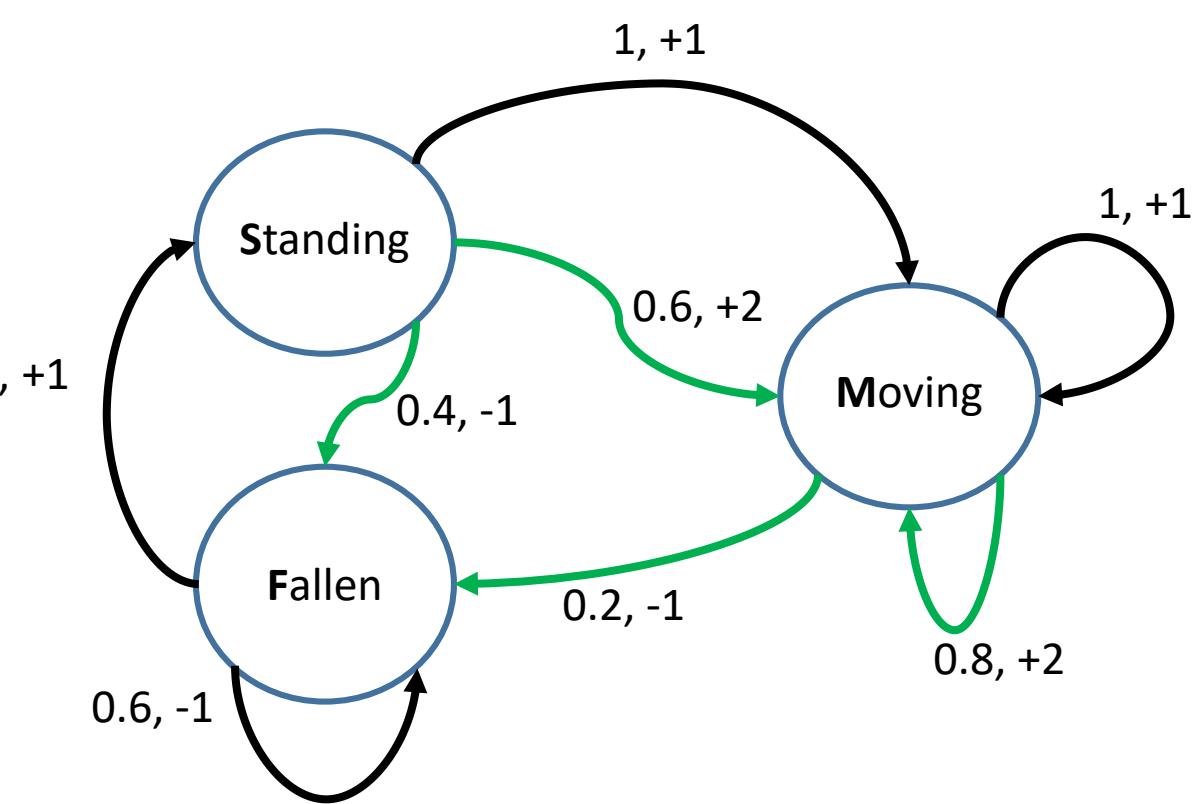
Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State Standing,

Slow Action: $= 1.1518$

Fast Action: $0.8 +$

$$0.1 [0.4 \quad 0 \quad 0.6] \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix} \simeq 0.88$$



New Policy: fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2 Improve policy:

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

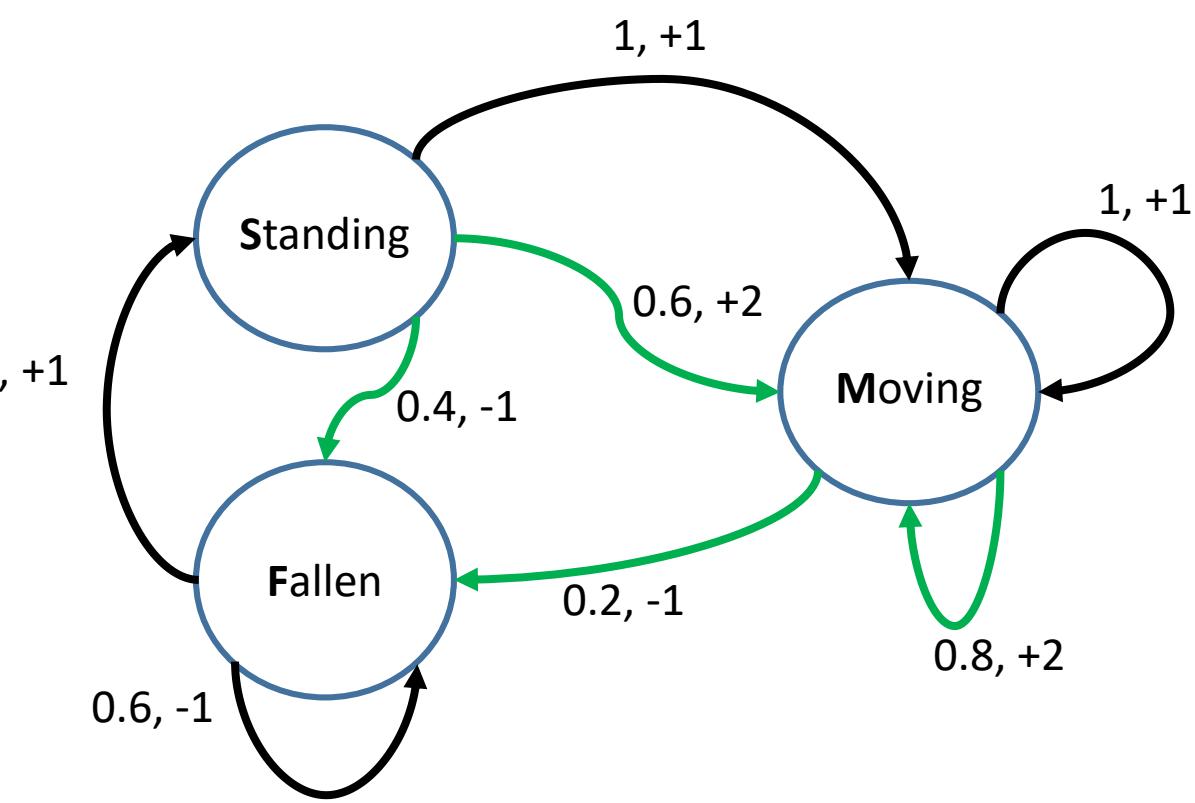
State *Standing*: SLOW action

State *Moving*,

Fast Action: =1.5182

Slow Action: 1 +

$$0.1 [0 \quad 0 \quad 1] \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix} \simeq 1.1518$$



New Policy: fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2 Improve policy:

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Compute $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State *Standing*: SLOW action

State *Moving*, FAST action

New policy is the same as the old policy
STOP!

Value Iteration method

- Finding optimal value function
 - No explicit policy
- In every iteration k , improve the value vector

$$v^{(k+1)}(s) = \max_a R_{s,a} + \gamma P_{s,a} v^{(k)}$$

- Converges to v_*

$$v^{(k)} \rightarrow v_*$$

- Optimal policy given by $\max_a R_{s,a} + \gamma P_{s,a} v_*$

Reinforcement Learning

Algorithms

Model free methods

- Reinforcement learning \equiv MDP with unknown transition model and/or reward distribution
- Model is unknown but agent observes samples
- Learn while optimizing the policy

Formulation

- Starts at some initial state s_1

In every round t , the agent

- observes the current state s_t ,
- take an action a_t , and then
- observes a reward signal r_t , and next state s_{t+1}

$$E[r_t | s_t = s, a_t = a] = R_{s,a}$$

$$\Pr(s_{t+1} = s' | s_t = s, a_t = a) = P_{s,a}(s')$$

$\{R_{s,a}, P_{s,a}\}$ are unknown

Goal

- Find the optimal policy:
Policy that maximizes expected sum of discounted reward

$\{R_{s,a}, P_{s,a}\}$ are unknown

Q-learning

- Uses “Q-values” instead of value function
- $Q(s, a)$: the value of taking action a in state s
- Formally

$$Q(s, a) = R_{s,a} + \gamma E_{s'}[\max_{a'} Q(s', a')]$$

Immediate expected reward plus the best utility from the next state onwards.

- From Bellman optimality equations, an optimal policy π satisfies

$$Q(s, \pi(s)) = R_{s,\pi(s)} + \gamma E_{s'}[Q(s', \pi(s'))] = v_*(s)$$

Q-learning

- Proceeds in discrete rounds $t = 1, 2, \dots$.

In every round t ,

- Choose action greedily using “estimated” Q-values

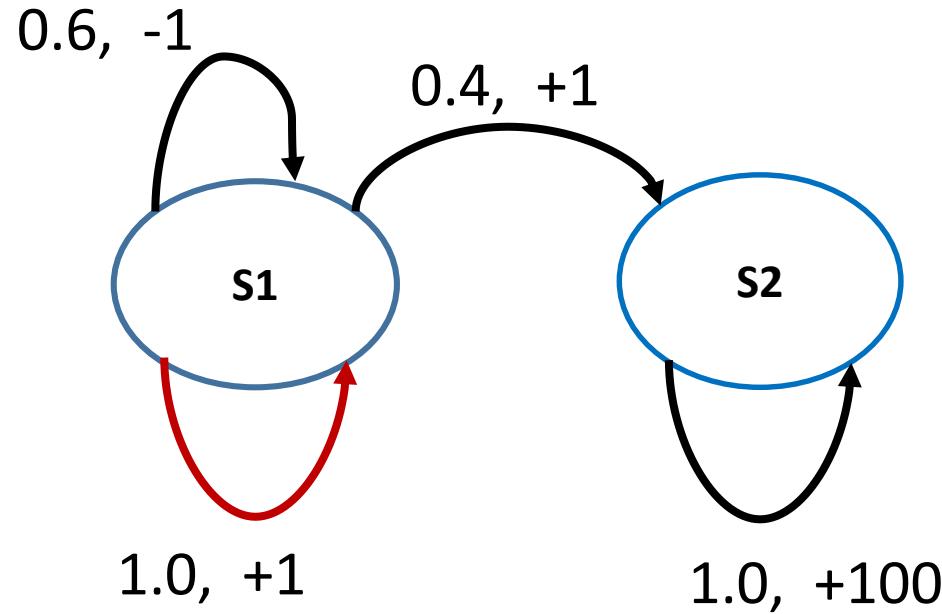
$$a_t = \operatorname{argmax}_a \hat{Q}(s_t, a)$$

- Take action a_t observe reward r_t , next state s_{t+1}
- Update Q-values for s_t, a_t

$$\hat{Q}(s_t, a_t) = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

(Compare to $Q(s, a) = R_{s,a} + \gamma E_{s'}[\max_{a'} Q(s', a')]$)

The need for Exploration



Epsilon Greedy exploration

- With probability $1 - \epsilon$, use greedy action

$$a_t = \operatorname{argmax}_a \hat{Q}(s_t, a)$$

- With probability ϵ , play random action