

Week 1: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: What is AI?

- Section 1.1, "What is AI?" from AI:AMA (**4th ed**)

Lesson 2: The state of the art in AI

- Section 1.3: "The History of AI" from from AI:AMA (**4th ed**)
- Section 1.4: "The State of the Art" from AI:AMA (**4th ed**)
- Section 1.5: "Risks and Benefits of AI", from AI:AMA (**4th ed**)

Lesson 3: Intro to AI Search

- Section 3.1: "Problem-Solving Agents", from AI:AMA (**4th ed**)

Lesson 4: Intro to Logic

- Section 7.1: "Knowledge-Based Agents", from AI:AMA (**4th ed**)

Lesson 5: Intro to Machine Learning

- Section 19.1: "Forms of Learning" from AI:AMA(**4th ed**)
 - *ref Section 18.1, from AI:AMA (3rd ed)*

Lesson 6: Contemporary AI

- [Drones with soon decide who to kill](#)
- [The everyday ethical challenges of self-driving cars](#)
- [Mastering the game of Go with deep neural networks and tree search](#)

Study Session:

Wednesday, 13 Sept 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Addt'l Tools:

[Harvard CS50 Introduction to AI with Python](#)

What is AI?

Thinking Acting Humanly Rationally Matrix

	Thinking	Acting
Humanly	Input/Output matches human behaviour <ul style="list-style-type: none">• Cognitive science - bringing together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind• the GPS program solving problems as a human would and following a similar thought process• self-preservation• neural networks• problem-solving and decision making like how humans would.	Turing Test <ul style="list-style-type: none">• Turing Test: A computer passes the test if a human interrogator, after posing written questions, cannot tell whether the responses are from a person or a computer.• Requires:<ul style="list-style-type: none">• Natural Language Processing (NLP)• knowledge representation<ul style="list-style-type: none">• storing what the machine knows• automated reasoning<ul style="list-style-type: none">• new conclusions• machine learning<ul style="list-style-type: none">• adaptation, extrapolation• Total Turing Test - Additionally requires interaction with objects in the real world.<ul style="list-style-type: none">• computer vision• robotics• Performing actions that require intelligence
Rationally	Logic (with Probability for uncertainty) <ul style="list-style-type: none">• syllogisms - patterns for argument structures• logic, which requires knowledge of the world that is certain• probability, which allows for rigorous reasoning about uncertain information <p>(!) Thinking rationally does not in itself generate intelligent behaviour; for this we need a theory of how to translate thinking into rational action.</p>	Logic & Probability with an Agent <ul style="list-style-type: none">• rational agent - acts so as to achieve the best (expected) outcome• standard model - a paradigm which models agents as doing the right thing, which is defined as achieving an objective that we provide to the agent.• in some environments we may wish to model limited rationality as full rationality, may not be computationally possible or optimal.

tionally -
thought
d reasoning
s used

ionally -
n behaviour and
phasizing the
ment

Search Problems (CS50 Intro to AI)

Vocabulary taken from CS50 Intro to AI with Python, Lecture 0

agent	entity that perceives its environment and acts upon that environment
state	a configuration of the agent and its environment
initial state	the state in which the agent begins
actions	choices that can be made in a state <i>Actions(s)</i> returns the set of actions that can be executed in state s
transition model	a description of what state results from performing any applicable action in any state <i>Results(s, a)</i> returns the state resulting from performing action a in state s
state space	the state of all states reachable from the initial state by any sequence of actions
goal test	way to determine whether a given state is a goal state
path cost	numerical cost associated with a given path
node	a data structure that keeps track of: <ul style="list-style-type: none">• a state• a parent (node that generated this node)• an action (action applied to parent to get to this node)• a path cost (from the initial state to this node)
stack	Last In First Out (LIFO) data type
queue	First In First Out (FIFO) data type
solution	a sequence of actions that leads from the initial state to a goal state

Search Problems (Problem-Solving Agents)

Solving problems by Searching

The goal: Planning ahead, to formulate a sequence of actions which form a path to the goal state

Such an agent is called a problem-solving agent and the computational process they undertake is called a search.

Informed algorithms - the agent can estimate how far it is from the goal (**Week 3**)

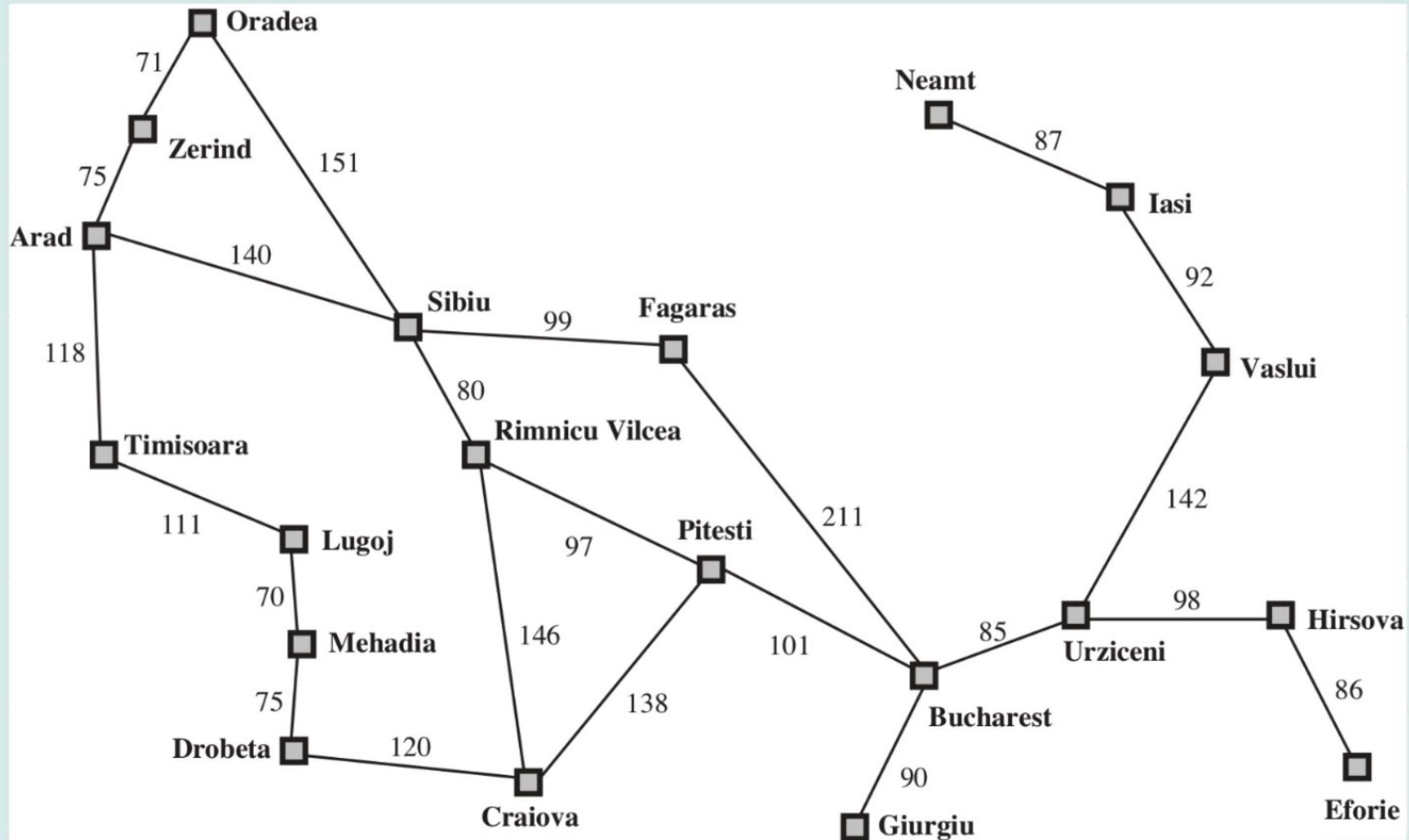
Uninformed algorithms - no such estimate is available (**Week 2**)

We assume that our agents have access to information about the world (e.g. a map), then with that information **the agent follows a 4-step process:**

1. Goal formulation
 - a. e.g. get to Bucharest
2. Problem formulation
 - a. a description of states and actions necessary to reach the goal
3. Search
 - a. before taking any action, the agent simulates sequences of actions in its model, until it finds a sequence of actions that reaches a goal
-> this sequence is called a solution.
4. Execution - the agent executes the actions in the solution, one at a time.

In a fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions (i.e. we can ignore percepts along the way, in what is called an **open-loop approach**).

If the model is incorrect then we should use a **closed-loop approach** where we monitor the percepts.



Formal definition of a search problem:

1. state space - set of possible states that the environment can be in
2. initial state - where we start
3. goal states (may be a single state or a set of states)
4. actions - which are available to the agent
5. transition model - describes what each action does
6. action cost function

PATH = sequence of actions

SOLUTION = path from initial state to goal

OPTIMAL SOLUTION = has the lowest cost among all solutions

abstraction - we need to think about how to formulate the problem abstracting irrelevant aspects

Lesson 4: Intro to Logic

"If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned."

- 1 - mythical = immortal
- 2 - NOT mythical = mortal
- 3 - immortal OR mammal = horned

Therefore if the unicorn is immortal or a mammal (or both), then it is horned.

- 4 - horned = magical

Given statements 1 and 2, the unicorn is either immortal or a mammal. Therefore statement 3 is true: as the unicorn is either immortal or a mammal, the unicorn is horned. Statement 4 is also true: The unicorn is horned, therefore it's magical. As we can't say for certain which statement 1 or 2 is true, only statement 3 and 4 hold: The unicorn is horned. The unicorn is magical (since it is horned).

mythical = immortal
NOT mythical = mortal
immortal OR mammal = horned
horned = magical

Logical Agents

Logical Agents can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.

Note: The problem solving agents discussed before did not know general facts, only specific information about states, transitions, etc.

Knowledge-Based Agents can accept new tasks in the form of explicitly defined goals and adapt to a new environment.

knowledge-based agents are supported by **logic (Weeks 4 and 5)**

Propositional logic - Illustrates all the basic concepts of logic, with well-developed inference technologies

First-order logic - Is more expressive

Knowledge-based agents, a few key concepts:

- sentence - an assertion about the world in a knowledge representation language
- knowledge base - a set of sentences which represent some assertion about the world
- axiom - a sentence taken as given without being derived from other sentences
- interacting with the knowledge base:
 - TELLing - adding new sentences to the knowledge base
 - ASKing - querying what is known
 - these operations may involve inference, whereby new sentences are derived from old
- the knowledge base is queried (via ASK) to determine what action to take
- importantly, we only need to specify the agent's knowledge base and goals, in order to determine its behavior
 - the implementation level is not important
 - as a result, we can build knowledge-based agents using the declarative approach, by TELLing it what it needs to know; we don't have to use the procedural approach which encodes behaviours directly.

Learning from Examples

We assume little prior knowledge on the part of the agent, instead the agent starts from scratch and learns from the data.

here, we focus on **induction** - going from a specific set of observations to a general rule

deduction (which is what knowledge-based agents use) is the opposite. In deduction, we start with premises and if the premises are correct then the conclusion is guaranteed to be correct.

Problems:

- **classification** - the output is a finite (countable) set of values
- **regression** - the output is a real number (uncountable)

Types of learning depend on the type of feedback:

- **supervised learning**
 - observing input-output pairs, which are labelled
 - learning a function which maps input to output
- **unsupervised learning**
 - agent learns patterns without explicit feedback/labels
 - e.g. clustering (Week 6)
- **reinforcement learning**
 - the agent learns from a series of reinforcements (rewards and punishments)
 - the agent decided which of the actions prior to the reinforcement were most responsible for it and alters its actions to aim towards more rewards in the future

AI Considerations

Possible issues/factors to consider:

- **value alignment problem** - when the true preferences we have are not aligned with the objectives we have set out for AI
 - it is not possible to predict all the ways in which a machine may "misbehave"
 - "King Midas problem" - machines give us exactly what we have asked for
- we may wish to **build uncertainty into the objectives**
- the "**gorilla problem**"
- **lethal and autonomous weapons**
- **surveillance and persuasion**
- **biased decision making**
- **impact on employment**
- **safety-critical applications**
 - the need for comparable technical standards
- **cyber security**

Drones with soon decide who to kill

- Do you think autonomous drones will use logic, machine learning, or AI search? Or a combination of these?
- Do you think that an AI system should be allowed to make the final decision to fire a weapon?

AI-generated music
<https://openai.com/blog/jukebox/>

The everyday ethical challenges of self-driving cars

- Do you think self-driving cars will use logic, machine learning, or AI search? Or some combination of these?
- What would be the most common ethical or legal problems encountered by self-driving cars?

Mastering the game of Go with deep neural networks and tree search

- Is AlphaGo using logic, machine learning, or AI search? Or some combination of these?
- Did you find the paper interesting or exciting?
- Do you think results from this paper could be used in areas other than game playing?
- Do you think it is good science? For example, could it be reproduced by another lab?

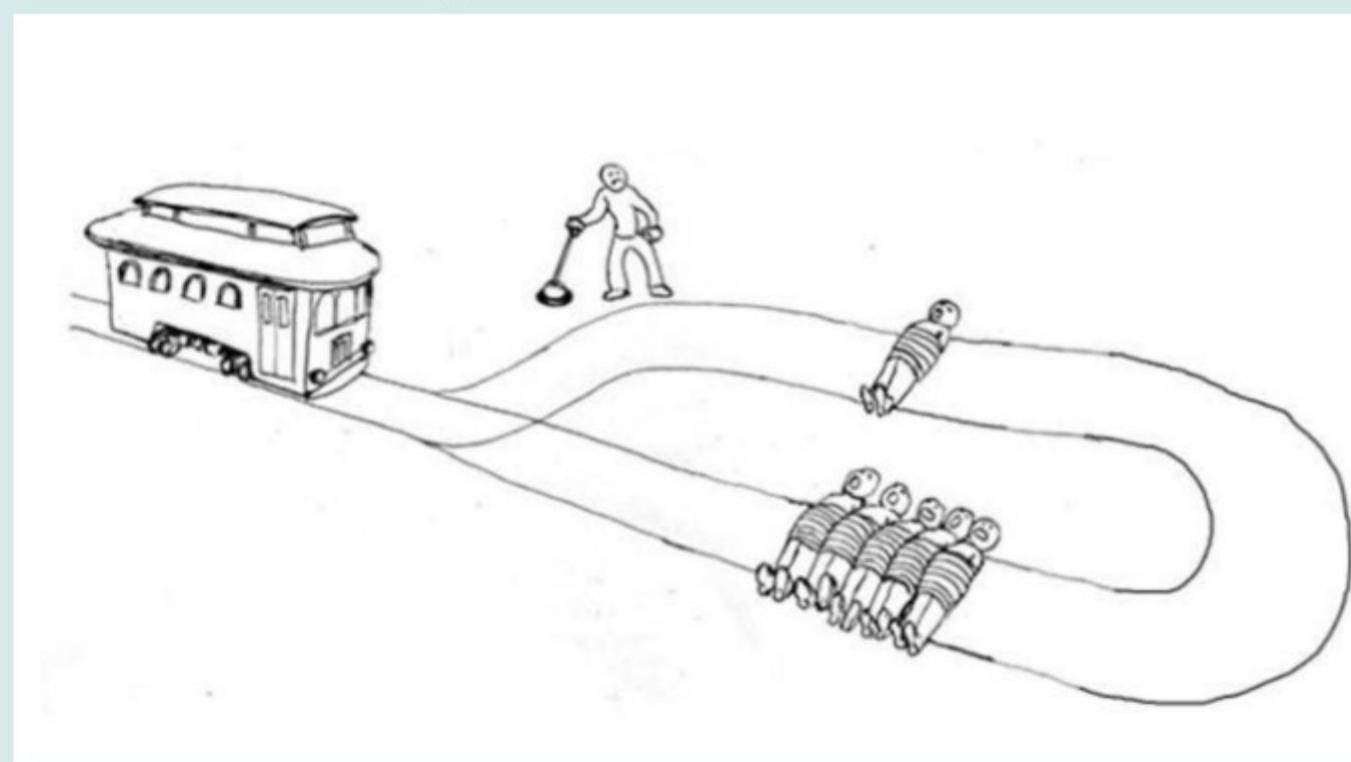
Eurovision song created by AI
https://www.youtube.com/watch?v=4MKAf6YX_7M

"No need to panic - artificial intelligence has yet to create a doomsday machine"

- "the best chess players in the world are not humans working or AI working along but human-computer teams"
- Loebner Prize - annual competition for a chatbot that could pass the Turing Test.
- Riedl's Lovelace 2.0 test of AI's capacity for creativity

"The trolley dilemma: would you kill one person to save five?"

- the problem allows us to think through the consequences of an action and consider whether its moral value is determined by its outcome
- Foot's argument - there is a distinction between killing and letting die (active v. passive)
- given people's responses to versions of the trolley problem, it appears that moral theories which judge the permissibility of an action based on its consequences alone (e.g. consequentialism or utilitarianism) do not explain why some actions are seen as permissible while others aren't.
- neuroscience research discovered that different parts of the brain are activated in different versions of the trolley dilemma - logical vs. emotional reasoning.



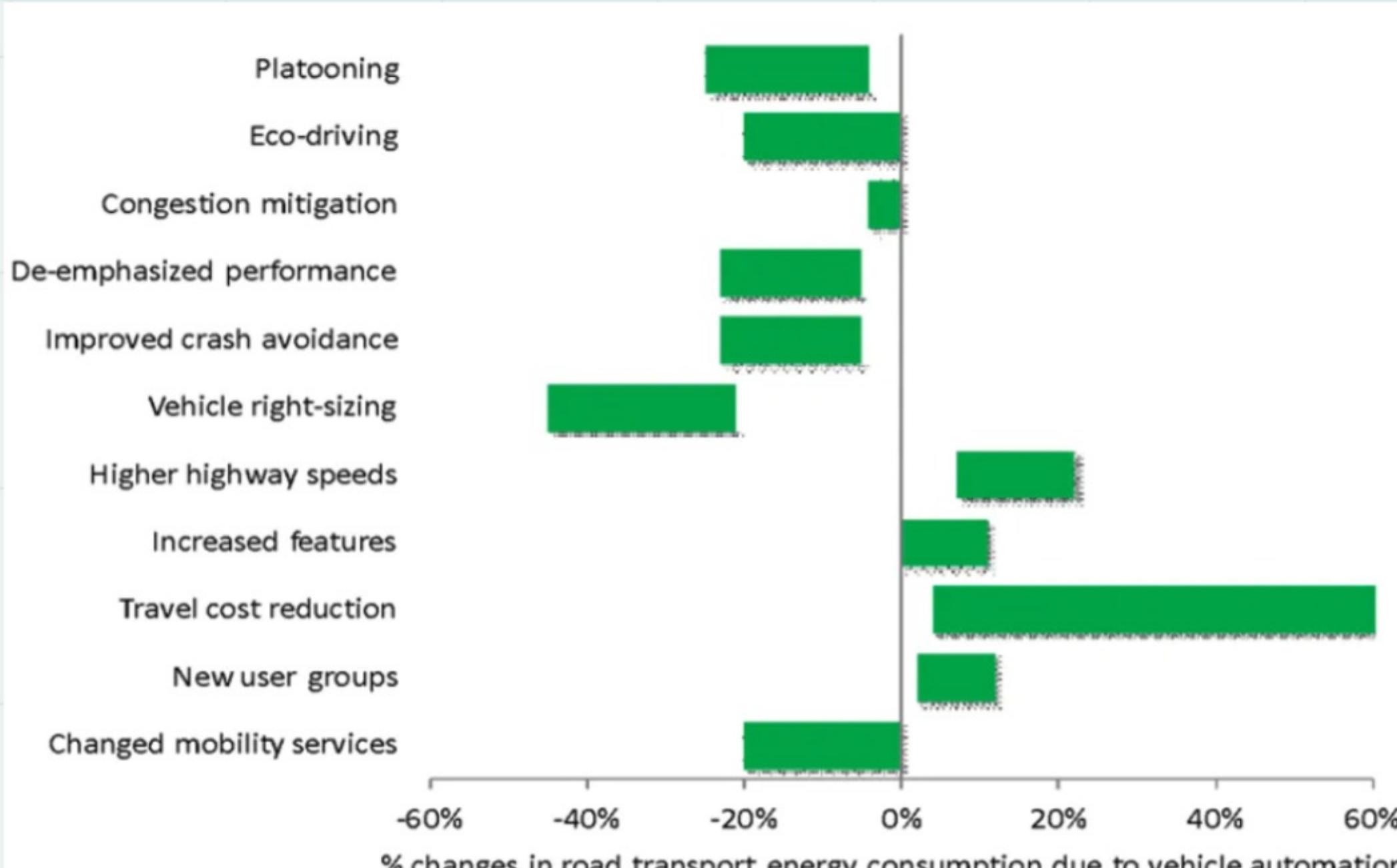
"Will self-driving cars reduce energy use and make travel better for the environment?"

For:

- coordination between automated vehicles to drive in a "platoon" to reduce air resistance and energy consumption
- smoothing traffic flow to reduce congestion and energy use
- automated "ecodriving" for more efficient fuel use
- higher vehicle safety => fewer heavy safety features => lighter cars and less energy use
- car-sharing and on-demand culture
- per-mile costs more visible to the user
- matching car size to trip aim
- alternative forms of fuel

Against:

- increase in demand for car journeys (e.g. instead of a train)
- increase in demand from new groups (those who can't drive themselves)



Canvas Questions / Activities

Competition Activity? Not sure how important this was.

"Competitions encourage researchers to build systems that are as good as possible at a particular task. The way a system works is less important than its performance at the task. Which of the four definitions of AI most closely matches this scenario?"

Acting Rationally.
Because we don't care how we got there, we just care that the solution is performant.

Week 2: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: Representing problems as AI search problems

- Section 3.1: "Problem-Solving Agents" from AI:AMA (**4th ed**)
- Section 3.2: "Example Problems" from AI:AMA (**4th ed**)

Lesson 2: Tree Search and Graph Search

- Section 3.3: "Searching for Solutions" from AI:AMA (**4th ed**)

Lesson 3: Uniformed search strategies

- Section 3.4.1: "Breadth-first search" from AI:AMA (**4th ed**)
- Section 3.4.4: "Depth-limited and iterative deepening search", from AI:AMA (**4th ed**)
 - *ref Sections 3.4.4 and 3.4.5 from AI:AMA (3rd ed)*
- Section 3.4.6, "Comparing uniformed search strategies" from AI:AMA (**4th ed**)
 - *ref Section 3.4.7 from AI:AMA (3rd ed)*

Study Session:

Thursday, 14 Sept 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Week 2: Learning Outcomes

- Represent search problems as states, actions, and goals.
- Implement the simplest search algorithms, including Depth-First Search, Breadth-First Search and Iterative Deepening
- Estimate the size and nature of the searched performed by uninformed search algorithms.

Assignments

In Canvas go to Assignments and under "Undated Assignments" Sample Paper 2 should be available to take. Give it a try before the Week 4 review.

The screenshot shows the Canvas assignments interface. The left sidebar includes links for Home, Units, Discussions, Reading List, Announcements, Assignments (which is selected), Panopto Recordings, and Kortext. The main area displays two sections: 'Upcoming assignments' and 'Undated assignments'. In the 'Upcoming assignments' section, there are two entries: 'Formative Assessment - Practice Exam (Groups 1-6)' due on Oct 4 at 5:00, and 'Artificial Intelligence and Machine Learning Exam - (Groups 1-6) [001 1.0]' due on Oct 27 at 5:00. In the 'Undated assignments' section, there are two entries: 'Sample Paper 1 (Groups 1-6)' worth 100 pts, and 'Sample Paper 2 (Groups 1-6)' worth 100 pts. The 'Sample Paper 2' entry is circled in green.

Basics of Search Problems

A **SEARCH PROBLEM** is formally defined as:

- **State Space** - a set of all possible states s reachable from the initial state by any sequence of actions
- **Initial State**
- A set of one or more **Goal States**
- The **Actions** available to the agent that can be executed in the given state.
 - *Action(s)* given state s
 - Absolute Movements: Right →, Left ←, Up ↑, Down ↓
 - Not all actions are available in all circumstances
- **Transition Model** - Maps a state and an action to a resulting state. This describes what each action *does*.
 - *Result(s,a)* given state s and action a applied to it
- **Action Cost Function** - A function that applies a cost to each action.
 - One per path (or move in a game)
 - Distance
 - Time

You will also generally need...

- **Goal Test** - How do you know when you've reached your goal?
- **Optimal Solution** - The solution with the lowest possible path cost.
 - For now we are assuming all costs are positive.

Examples of applications of search problems:

- Route-finding problems
- Touring problems
 - e.g. travelling salesman problem
- Very Large Scale Integration (VLSI) layout
 - positioning components on a chip to minimise the area
 - cell layout
 - channel routing - finding a specific route for each wire through the gaps between the cells
- Robot navigation
- Automatic assembly sequencing
- Protein design

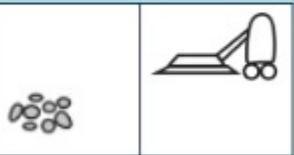
Key examples highlighted in the reading:

- Vacuum world
- 8-puzzle
- 8 queens problem
- Knuth's problem

Vacuum World

The vacuum world problem is **deterministic**, but variations can include **uncertain** (not fully observable or not deterministic) elements such as randomly appearing dirt or an unreliable suction mechanism.

Note: This is a example of a **graph search** (not a tree).

- **Initial State:** Any state can be designated as initial. I'm picking this one arbitrarily.

- **Actions:** Left, Right, Suck, NoOp
- **Action cost:** +1 per action, 0 for NoOp
- **Goal States** - Where everything is clean. In this case there are (2) possibilities.


- **Transitional Models**

Initial State

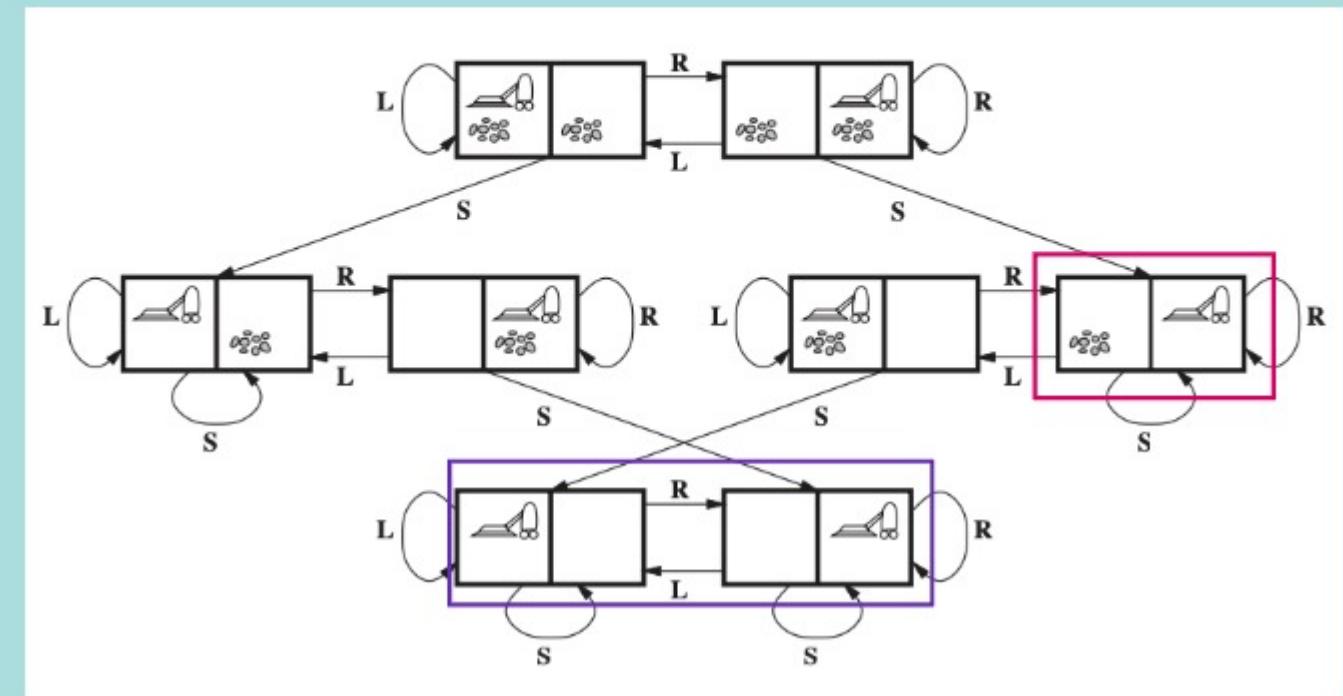

$$\text{Result} \left(\text{Initial State}, \text{Left} \right) = \text{Result} \left(\text{Initial State}, \text{Suck} \right)$$

Total Cost = 1

$$\text{Result} \left(\text{Initial State}, \text{Suck} \right) = \text{Goal State}$$

Total Cost = 2

State Space



The state is determined both by the agent location and the dirt locations.

AI search problems Quiz

Question 1 KEYWORDS: transition model 1 pts

以下哪项陈述描述了转换模型?

Which of the following statements describes the transition model?

- Maps from a state to a state and an action
- Maps from a state to a state
- Maps from a state and an action to another state
从一个状态和一个动作映射到另一个状态
- Maps from a state to an action
- Maps from an action to a state

Question 2 KEYWORDS: incremental formulation; initial state 1 pts

如果对某个问题使用增量公式,您将如何描述初始状态?

If an incremental formulation is used for some problem, how would you describe the initial state?

- Empty .
初始状态将被描述为空。
- Leftmost
- Deconstructed
- Bottom
- Top

Question 3 KEYWORDS: goal states; route-finding problem 1 pts

寻路问题有多少个目标状态?

How many goal states does the route-finding problem have?

- For a problem with n cities, there are $2n$ goal states
- One for each road connected to the destination city
- One
- One for each location

Question 5 KEYWORDS: solution definition 1 pts

Which of the following statements best describes a solution?

- A goal state
- A sequence of actions that reaches the initial state starting from a goal state
- A sequence of actions that reaches a goal state from the initial state
从初始状态达到目标状态的一系列动作
- A special state that must be in all paths from the initial state to any goal state

Question 4 KEYWORDS: goal states; 8-puzzle 1 pts

8 拼图有多少个目标状态?

How many goal states does the 8-puzzle have?

- 1.8 googols
- One
- Eight – one for each tile
- 36 – the sum of the tiles

Question 6 KEYWORDS: optimal solution definition 1 pts

以下哪项陈述最能描述最佳解决方案?

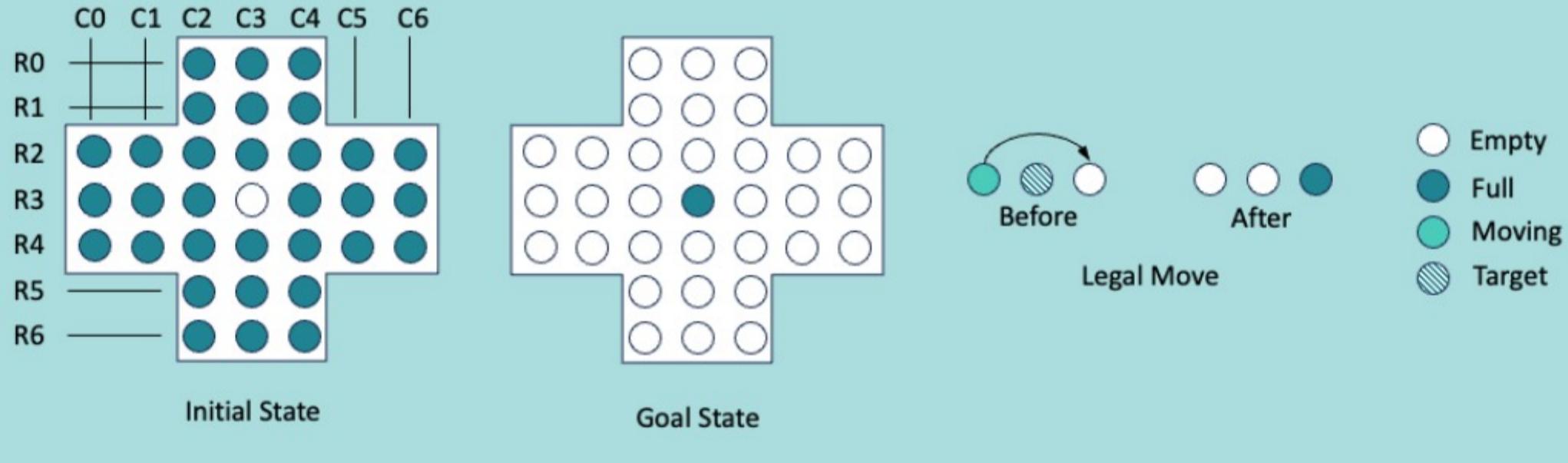
Which of the following statements best describes an optimal solution?

- A solution with the largest possible path cost
- A solution with the smallest number of actions
- A solution with the largest possible step cost
- A solution with the smallest possible path cost

Activity 1: English Peg Solitaire

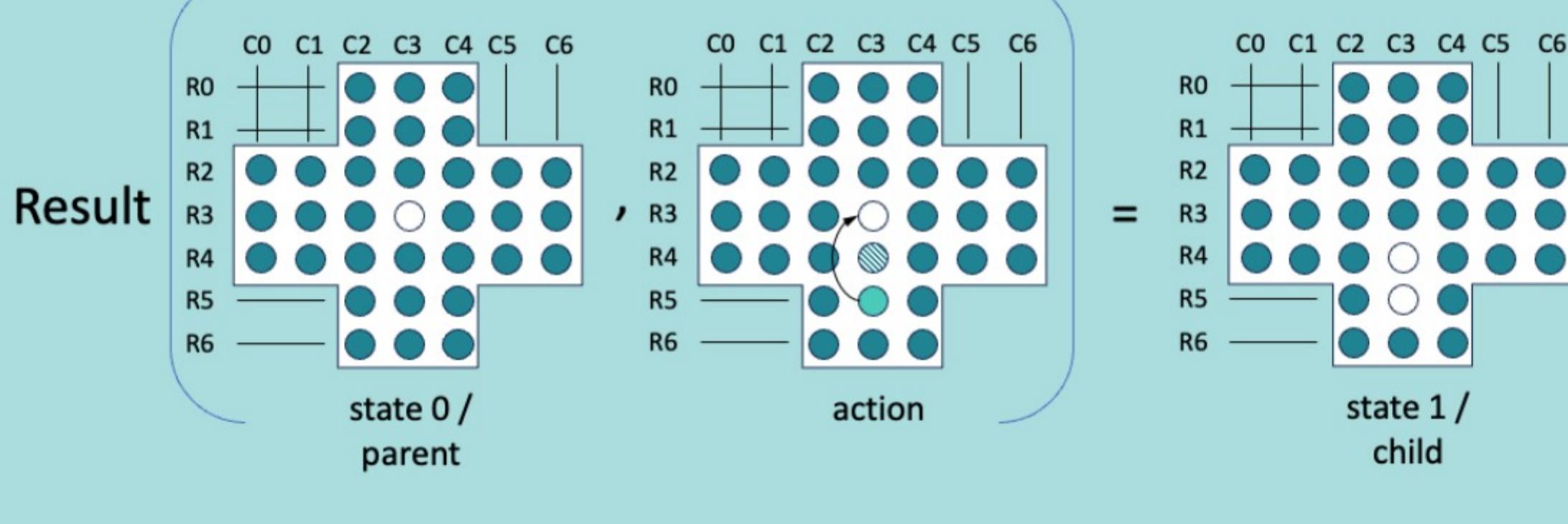
Find out how to play English Peg Solitaire (the game is described on Wikipedia, for example). Formulate it as an AI search problem, starting by defining a state. Then define the actions that can be performed, and the transition model.

- Describe the initial state and the goal test in terms of your definition of the state.
- Does it matter what the step costs are in this game, and if so, can you suggest sensible step costs?
- Can you think of an alternative way of defining a state?
- If so, how do the two formulations compare? Is one clearly better than the other?
- Write out your formulation of the game informally – there is no need to use mathematical notation. You can use the formulation of the 8-puzzle in the textbook as a guide.



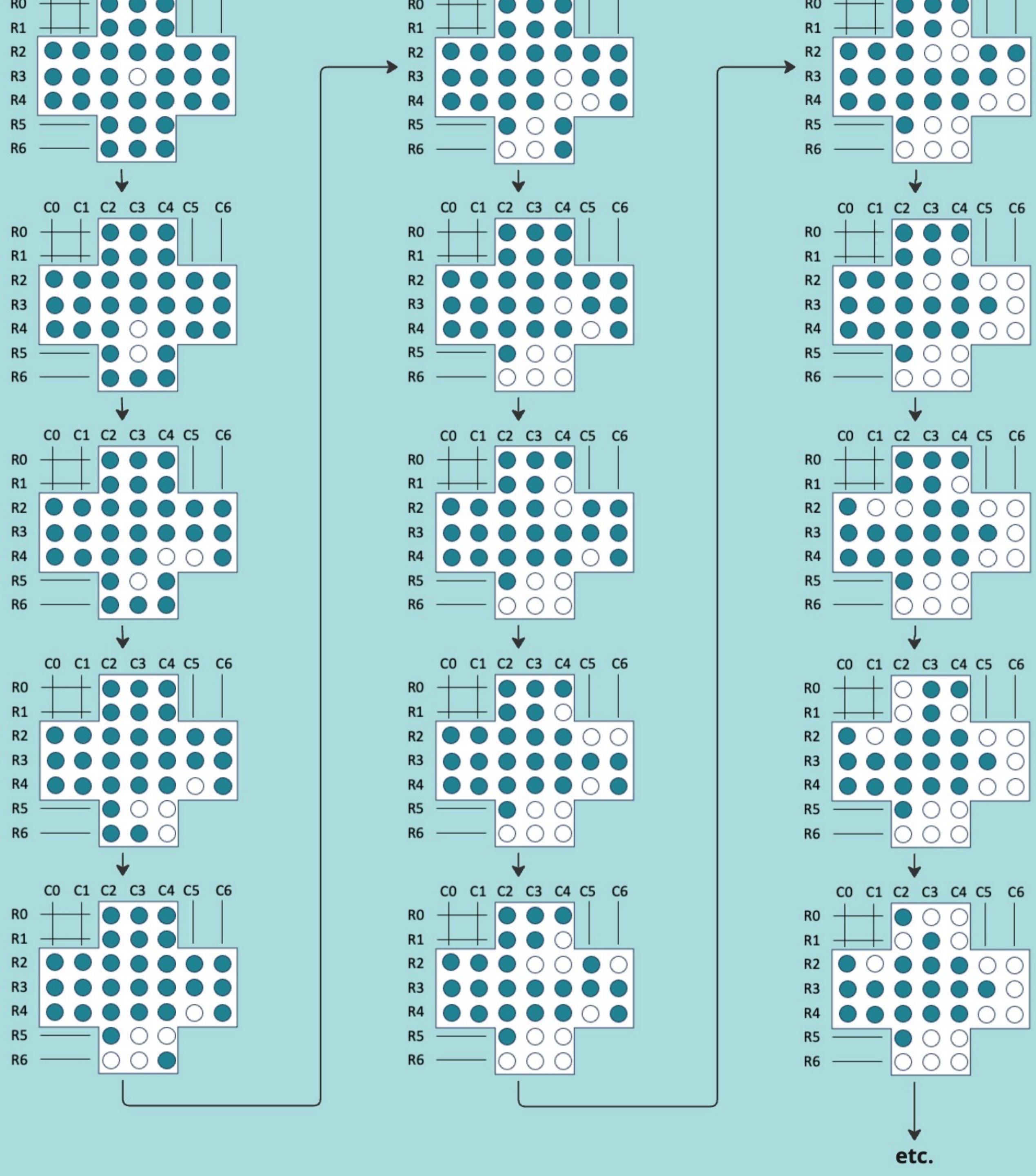
Step Costs = 1? Does it really matter? How many different ways can you solve this successfully?

Transitional Model



State diagrams can be defined by the resultant states after each move. Are there alternative ways to defining the states?

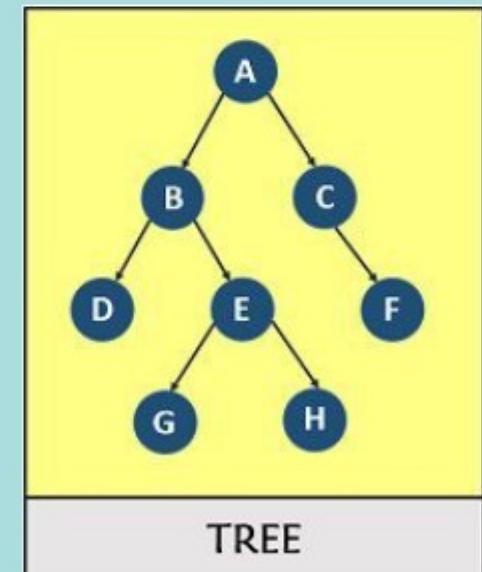
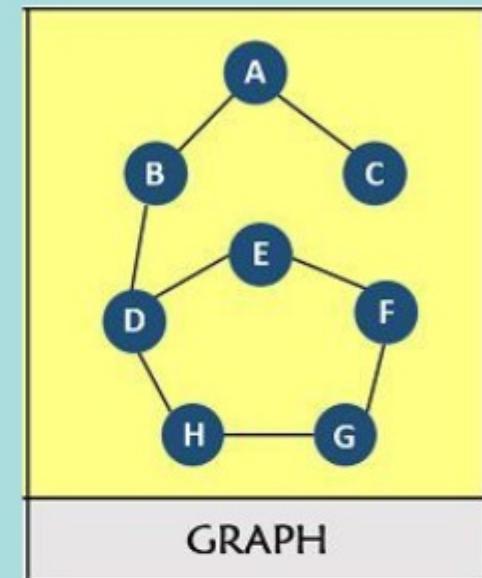
State Space - Each of these states are sequential nodes on the OPTIMAL PATH.



Following this model will solve the puzzle (i.e. final result = goal state) after 32 moves (cost = 32). [miro](#)

Tree vs. Graph Data Structures

	Graph	Tree
Definition	A non-linear data structure that consists of a group of vertices/nodes connected through edges	A type of graph. A non-linear data structure that consists of a collection of nodes and edges that simulates a hierarchical tree structure, with a root and subtrees of children with a parent node.
Organization	Network	Hierarchical
Root	Does not have a unique node called a root.	Must have a root node.
Loops	Can have loops	Acyclic / No loops
Connections	Can be connected or unconnected . If there is a path between every pair of distinct vertices of the graph it is considered connected; otherwise it is unconnected.	Always connected : There is a path between every pair of distinct vertices of the graph.
Direction	Can be directed or undirected . Undirected edges are indicated by a straight line, they do not have a direction which indicates a two-way relationship. Directed graphs will use an arrow to specify a direction (typically one-way).	Always directed from parent to child.
Weighting	Can be weighted or unweighted . In weighted graphs the connections between nodes have a weight assigned to them. Unweighted graphs have no value assigned therefore they are all assumed to be equal with a weight of 1.	??? I believe trees are all unweighted but I'm not sure.
Complexity	More complex	Less complex



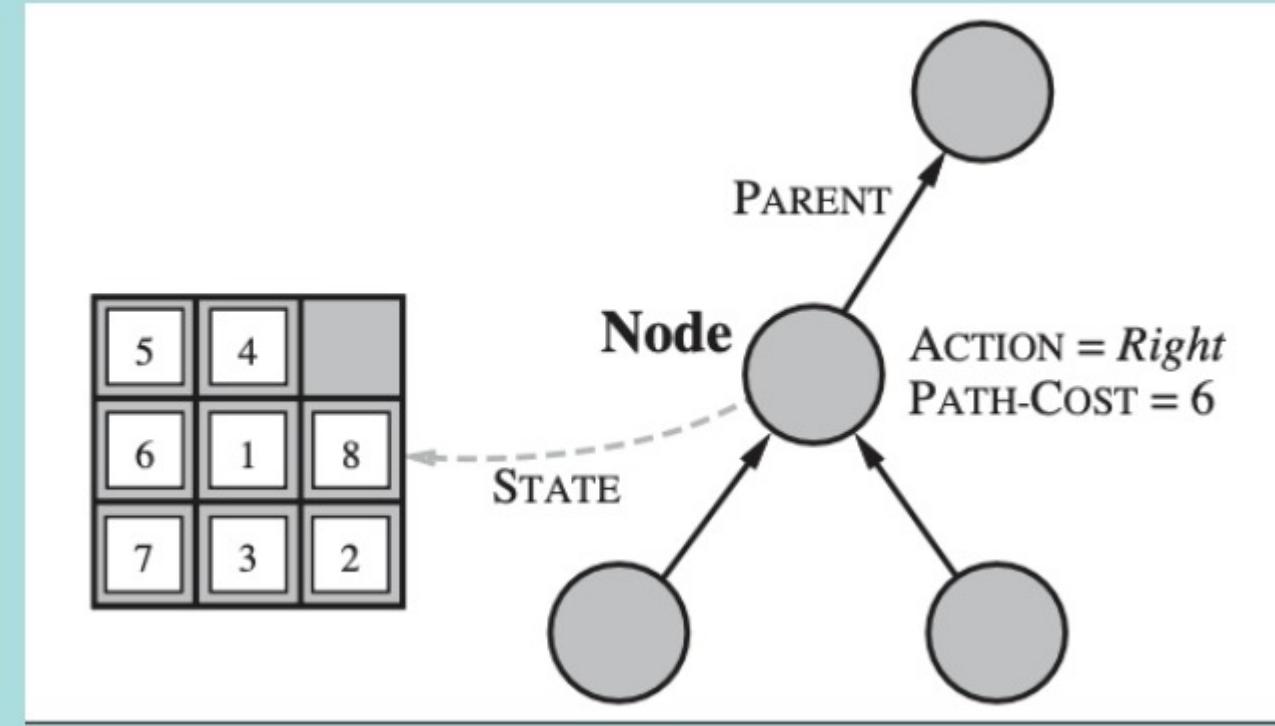
Infrastructure of Search Algorithms

Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node n we have a structure that contains four components:

- $n.State$: the state in the state space to which the node corresponds
- $n.Parent$: the node in the search tree that generated this node
- $n.Action$: the action that was applied to the parent to generate the node
- $n.Path-Cost$: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

Two different nodes **can** contain the same state if that state was generated by two different search paths.

- **State space** - Describes the (possibly infinite) set of states in the world and the actions that allow transitions from one state to another.
- **Search tree** - Describes the paths between these states towards a goal.
- Note: A search tree may have multiple paths to a given state, but each *node* in the tree has a unique path back to the root.
- **Repeated states** can give rise to loopy paths, which in turn make the complete search tree infinite.
- **Loopy Paths** are a special case of redundant paths, which exist whenever there is more than one way of getting from one state to another.



Considerations when selecting an algorithm:

- **Completeness** - Is the algorithm guaranteed to find a solution when there is one?
- **Time Complexity** - Big O notation
- **Space Complexity** - memory requirements
- **Optimality** - does it find the optimal solution (with regard to path costs)?
- **State Space / Tree Complexity** - Expressed in terms of:
 - b , the **branching factor**, or max number of successors of any node
 - d , the **depth** of the shallowest goal node (i.e. the number of steps along the path from the root)
 - m , the **maximum length** of any path in the state space

Tree Search vs. Graph Search

The difference between **Tree Search** and **Graph Search** is **-NOT-** that tree search works on trees and graph search works on graphs. Both can work on either. The key difference is that graph search uses a data structure called the **explored set (closed list)**, which remembers every expanded node. This prevents re-visiting already visited states, or adding already generated states to the frontier, which would result in duplicates.

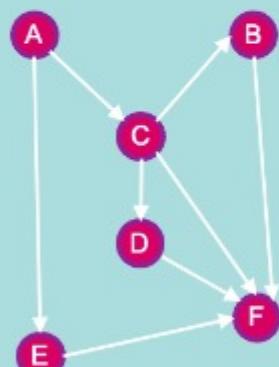
frontier - The set of all leaf nodes available for expansion at any given point

explored set - A closed list used by Graph-Search to keep track of the nodes that have been expanded/explored.

Function: Tree-Search(*problem, strategy*)

Approach:

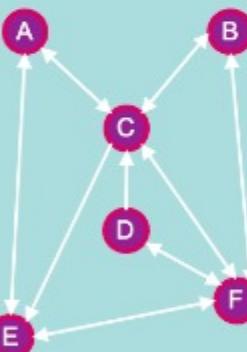
- Start with a **frontier** that contains the initial state of the *problem*.
- LOOP DO:**
 - IF** the frontier is empty, **THEN return** failure (no solution).
 - Choose a frontier node for expansion using *strategy* and remove from the frontier.
 - IF** the node contains the goal state, **THEN return** the solution.
 - ELSE** expand node, add resulting nodes to the frontier.



Function: Graph-Search(*problem, strategy*)

Approach:

- Start with a **frontier** that contains the initial state of the *problem*.
- Start with an empty **explored set**.
- LOOP DO:**
 - IF** the **frontier** is empty, **THEN return** failure (no solution).
 - Choose a frontier node for expansion using *strategy* and remove from the **frontier**.
 - IF** the node contains the goal state, **THEN return** the solution.
 - ELSE** Add the node to the **explored set**.
 - Expand** node and add resulting nodes to the frontier only if not already in the frontier or the explored set.



The frontier is a dataset. The strategy that is drive which node is selected next.

queue - A linear data structure that operates on a **First In Last Out (FIFO)** basis; pops the oldest element of the queue.

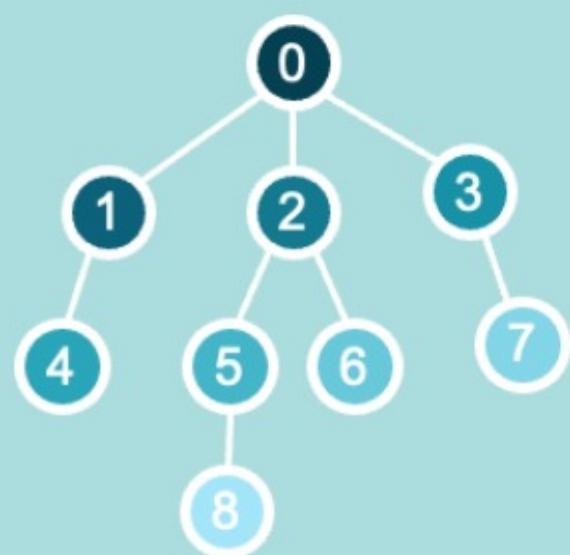
stack - A linear data structure that operates on a **Last In First Out (LIFO)** basis; pops the newest element of the queue.

priority queue - Elements are popped based on the priority given by an ordering function (i.e. uniform cost search).

Breadth-First Search

BFS is an instance of graph search that always expands the shallowest node in the frontier, i.e. the frontier operates as a **Queue (FIFO)**.

Note: In BFS the goal test is applied to each node when it is generated, rather than when it is selected for expansion.

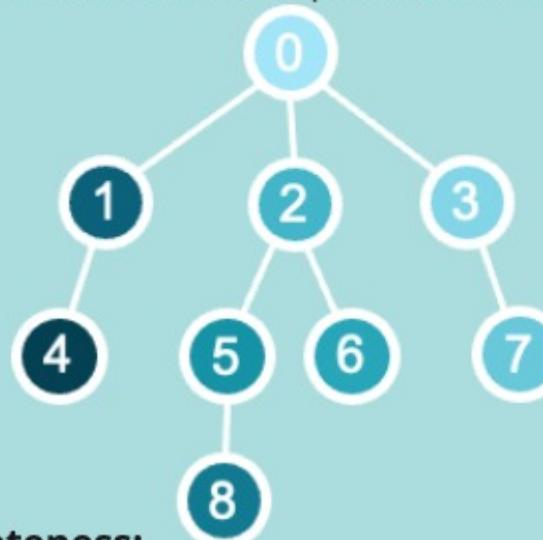


```
At node 0 in the BFS  
Adding node 1 to the queue in the BFS  
Adding node 2 to the queue in the BFS  
Adding node 3 to the queue in the BFS  
Done processing node 0  
At node 1 in the BFS  
Adding node 4 to the queue in the BFS  
Done processing node 1  
At node 2 in the BFS  
Adding node 5 to the queue in the BFS  
Adding node 6 to the queue in the BFS  
Done processing node 2  
At node 3 in the BFS  
Adding node 7 to the queue in the BFS  
Done processing node 3  
At node 4 in the BFS  
Done processing node 4  
At node 5 in the BFS  
Adding node 8 to the queue in the BFS  
Done processing node 5  
At node 6 in the BFS  
Done processing node 6  
At node 7 in the BFS  
Done processing node 7  
At node 8 in the BFS  
Done processing node 8  
Finished with the BFS from start node 0
```

- **Complete**; it is guaranteed to reach the goal state provided that the branching factor is finite.
- **Not necessarily optimal** as the shallowest node is not always the optimal one. **Optimal only if** the path cost is a non-decreasing function of the depth of the node (typically when **all actions have the same cost**).
- **Memory requirements** are a bigger problem than execution time.

Depth-First Search

DFS can be implemented through tree search, graph search, or recursively. It is a search algorithm that always expands the deepest node in the frontier, i.e. the frontier operates as a **Stack (LIFO)**.



```
At node 0 in the DFS  
Going to node 1...At node 1 in the DFS  
Going to node 4...At node 4 in the DFS  
Done processing node 4  
Done processing node 1  
Going to node 2...At node 2 in the DFS  
Going to node 5...At node 5 in the DFS  
Going to node 8...At node 8 in the DFS  
Done processing node 8  
Done processing node 5  
Going to node 6...At node 6 in the DFS  
Done processing node 6  
Done processing node 2  
Going to node 3...At node 3 in the DFS  
Going to node 7...At node 7 in the DFS  
Done processing node 7  
Done processing node 3  
Done processing node 0
```

Completeness:

- tree-search based: Not complete due to loops
- graph-search based: Complete because eventually every node is expanded

Non-optimal:

Time complexity:

- tree-search based: $O(b^m)$, where m is the maximum depth of any node
- graph-search based: bounded by the size of the state space (which may be infinite)

Space complexity:

- tree-search based: $O(bm)$ - this is because we only ever need to remember a single path from the root to a leaf node
- graph-search based: no advantage over BFS in space complexity

(!) The key advantage of DFS over BFS is in its space complexity

Depth-Limited Search

- An instance of **Depth-First Search** that includes a recursive function that calls itself on each of its children, incorporating a depth limit.
- Essentially DFS but with a **depth limit**, L , and all nodes at depth L are treated as if they have no successors.
 - DFS can be seen as a special case with $L=\text{infinity}$
- The aim is to overcome the failure of DFS in infinite state spaces, this solves the infinite-path problem.
- We can set limits based on our knowledge of the problem, such as the diameter of the state space.

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    else if limit = 0 then return cutoff
    else
        cutoff_occurred? ← false
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            result ← RECURSIVE-DLS(child, problem, limit - 1)
            if result = cutoff then cutoff_occurred? ← true
            else if result ≠ failure then return result
        if cutoff_occurred? then return cutoff else return failure
```

- **Incomplete if $L < d$**
- **Non-optimal if $L > d$**
- **Time Complexity:** $O(b^L)$
- **Space Complexity:** $O(bL)$

Iterative Deepening Search

- A version of **Depth-Limited Search** but gradually increasing the depth limit L until a goal is found.
- It combines the benefits of breadth-first and depth-first search.
- It may seem wasteful as states are generated multiple times, but the states which are generated multiple times are closer to the root.
- **In general this approach is the preferred uninformed search method when the search space is large and the depth of the solution is known.**

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to  $\infty$  do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

- **Complete**; like BFS, so long as the branching factor is finite.
- **Optimal if** the path cost is a non-decreasing function of the depth of the node (typically when **all actions have the same cost**).
- **Time Complexity:** $O(b^d)$
- **Space Complexity:** $O(bd)$

Backtracking Search

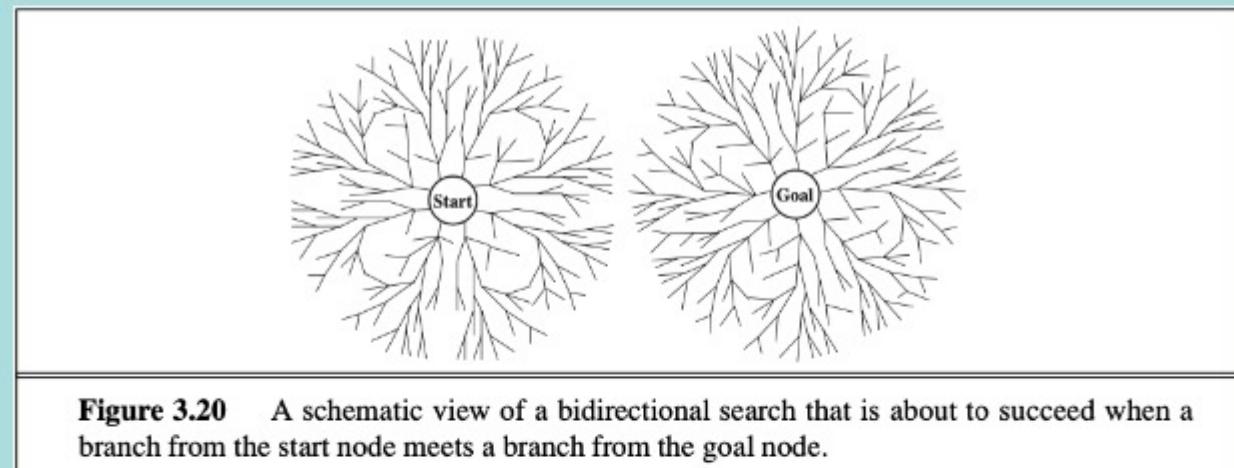
- An instance of **Depth-First Search** which uses even less memory.
- Here only one successor is generated at a time (rather than all successors) so only $O(m)$ of memory is required.
- Additionally we can generate state descriptions through modifications rather than copying which reduces the memory requirements to just one state description and $O(m)$ actions.

```
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  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, value)
      if inferences  $\neq$  failure then
        add inferences to assignment
        result  $\leftarrow$  BACKTRACK(assignment, csp)
        if result  $\neq$  failure then
          return result
      remove {var = value} and inferences from assignment
  return failure
```

Bidirectional Search

- Running two simultaneous searches, one forward from the initial state, and the other backwards from the goal, **hoping that they meet in the middle**.
- Instead of a goal test, we check whether the frontiers of the two searches intersect...if they do then a solution is found.
- The solution need not be optimal (there could be a shortcut)
- But space complexity is a significant downside.
- Helpful when there is only one goal state but difficult to use when the goal is an abstract description.



- **Complete** if b is finite and if both directions use breadth-first search
- **Optimal** if step costs are identical and if both directions use breadth-first search
- **Time Complexity:** $O(b^{d/2})$
- **Space Complexity:** $O(b^{d/2})$

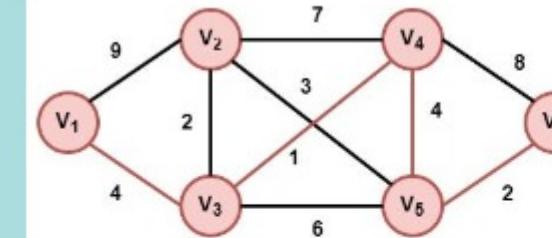
Uniform-Cost Search

- An instance of **Breadth-First Search** that is optimal with any step-cost function (not just when all step costs are equal).
- The frontier is a **priority queue** dataset and instead of expanding the shallowest node, expands the node n with the lowest path cost $g(n)$.
- Goal test is applied when the node is selected for expansion, to avoid selecting a sub-optimal path.
- A test is added in case a better path is found to a node which is currently in the frontier.
- **Optimal** when a node is selected for expansion, the optimal path to the node has been found. Step costs are non-negative so paths never get shorter as nodes are added. Uniform search expands nodes in the order of their optimal path cost.
- If C^* is the cost of the optimal solution and each action costs at least 3, then the worst case time and space complexity is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- When all costs are equal, the **time and space complexity** are $O(b^{d+1})$; in this case the search is very similar to BFS but with the goal test applied at expansion, so in this case uniform-cost search performs additional work compared to BFS in expanding the nodes.

```

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
            frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
            replace that frontier node with child
    end for
end loop

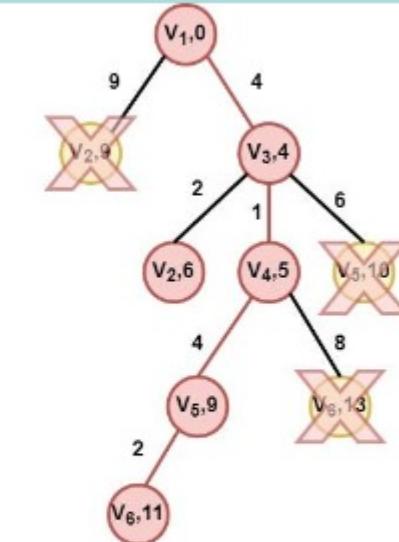
```



Pace	Opened	Closed
0	$\{(V_1,0)\}$	$\{-\}$
1	$\{(V_2,9),(V_3,4)\}$	$\{(V_1,0)\}$
2	$\{(V_2,6),(V_4,5),(V_5,10)\}$	$\{(V_1,0),(V_3,4)\}$
3	$\{(V_2,6),(V_5,13),(V_5,9)\}$	$\{(V_1,0),(V_3,4),(V_4,5)\}$
4	$\{(V_6,13),(V_5,9)\}$	$\{(V_1,0),(V_3,4),(V_4,5),(V_2,6)\}$
5	$\{(V_6,11)\}$	$\{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)\}$
6	$\{-\}$	$\{(V_1,0),(V_3,4),(V_4,5),(V_2,6),(V_5,9)\}$

Path: $V_1 - V_3 - V_4 - V_5 - V_6$

Total Cost: 11



```

Enter the number of vertices
6
Enter the weighted Adjacency Matrix for the graph
0 9 4 0 0 0
9 0 2 7 3 0
4 2 0 1 6 0
0 7 1 0 4 8
0 3 6 4 0 2
0 0 0 8 2 0
Enter the source (start at 1)
1
Enter the destination
6
|The eval Node is 1
The eval Node is 3
The eval Node is 4
The eval Node is 2
The eval Node is 5
The eval Node is 6
The Path between 1 and 6 is
1      3      4      5      6
The Distance between source 1 and destination 6 is 11

```

Uninformed Search Strategies (aka blind searches) - Summary

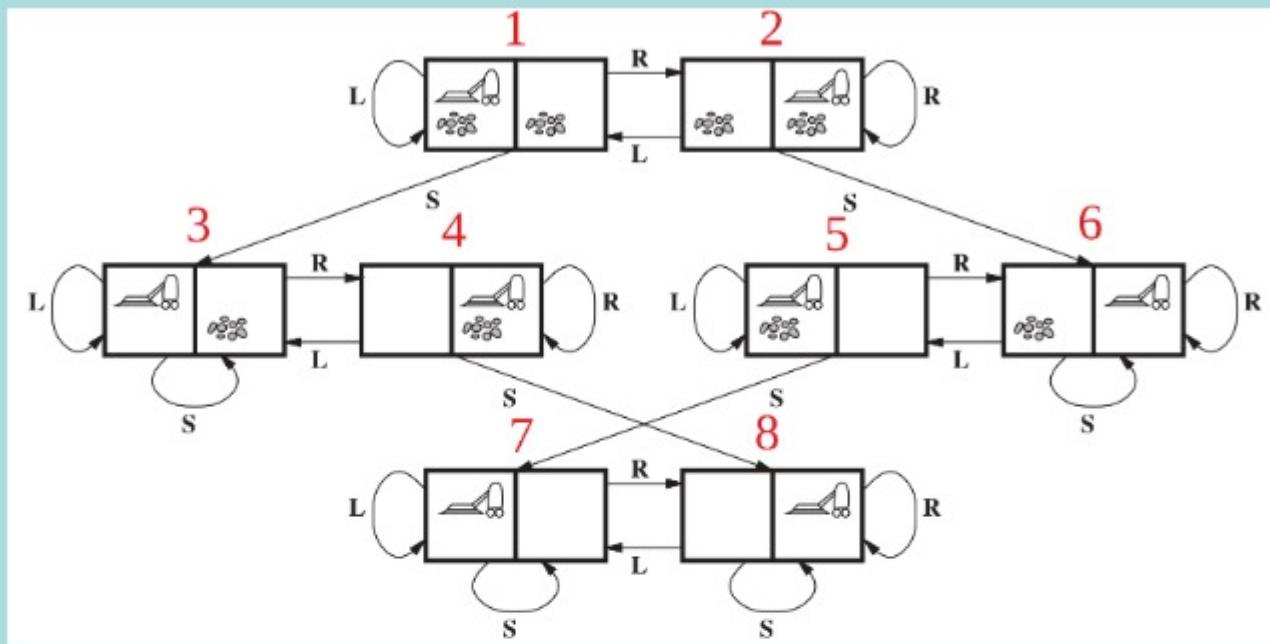
Uninformed search methods have access only to the problem definition. The basic algorithms are as follows:

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; ℓ is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

- **Breadth-First search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.
- **Uniform-Cost search** expands the node with the lowest path cost, $g(n)$, and is optimal for general step costs.
- **Depth-First search** expands the deepest unexpanded node first. It is neither complete nor optimal, but has linear space complexity. Depth-limited search adds a depth bound.
- **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.
- **Bidirectional search** can enormously reduce time complexity, but it is not applicable and may require too much space.

Activity 2: Implementing Graph Based version of Breadth-First Search



Code distributed by the tutors

```

2 import java.util.*;
3
4 class VacuumBFS {
5     // Constant arrays to look up the result of applying an action to a given state.
6     // For example, looking up state 1 in the actionRight array gives state 2
7     // which is the result of moving right from state 1 (see the diagram in unit 2.8)
8     static int[] actionLeft= {1,1,3,3,5,5,7,7};
9     static int[] actionRight={2,2,4,4,6,6,8,8};
10    static int[] actionSuck= {3,6,3,8,7,6,7,8};
11
12    // Returns true when s is a goal state.
13    static boolean goalTest(int s) {
14        return (s==7 || s==8);
15    }
16
17    // Returns true if there is a solution, i.e. a path from state 1 to 7 or 8.
18    static boolean BFS() {
19        int initialState=1;
20        if(goalTest(initialState)) {
21            return true;
22        }
23
24        // frontier initially contains just 1.
25        ArrayList<Integer> frontier=new ArrayList<Integer>();
26        frontier.add(1);
27        // frontier will be used as a FIFO queue to implement BFS.
28
29        // explored is initially empty.
30        ArrayList<Integer> explored=new ArrayList<Integer>();
31
32        while(true) {
33            if(frontier.size()==0) {
34                // There are no more nodes to expand, and we did not find a goal.
35                return false;
36            }
37
38            // Remove the first element of frontier
39            int node=frontier.remove(0);
40            explored.add(node);
41
42            // Expand 'node'
43            // Make children array containing the three states that
44            // result from applying the three actions to the current state.
45            int[] children = {actionLeft[node-1], actionRight[node-1], actionSuck[node-1]};
46
47            System.out.println("Expanding node: "+node+" to: "+Arrays.toString(children));
48
49            for(int child : children) {
50                if(goalTest(child)) {
51                    System.out.println("Goal state found: "+child);
52                    return true;
53                }
54                if( !frontier.contains(child) && !explored.contains(child) ) {
55                    // This state has not been seen before.
56                    // Add it to the end of the frontier list.
57                    frontier.add(child);
58                }
59            }
60        }
61
62    public static void main(String args []) {
63        boolean solutionFound=BFS();
64        System.out.println("Solution found: "+solutionFound);
65    }
66 }

```

```

Expanding node: 1 to: [1, 2, 3]
Expanding node: 2 to: [1, 2, 6]
Expanding node: 3 to: [3, 4, 3]
Expanding node: 6 to: [5, 6, 6]
Expanding node: 4 to: [3, 4, 8]
Goal state found: 8
Solution found: true
|

```

State	1	2	3	4	5	6	7	8
Dirt Left	True	True	False	False	True	True	False	False
Dirt Right	True	True	True	True	False	False	False	False
Vacuum Position	Left	Right	Left	Right	Left	Right	Left	Right
Action – Left	1	1	3	3	5	5	7	7
Action – Right	2	2	4	4	6	6	8	8
Action – Suck	3	6	3	8	7	6	7	8

Shamelessly stolen from Tom Harrison as it's much simpler
but appears to take 1 step longer.

```

1 import java.util.ArrayList;
2 import java.util.LinkedList;
3 import java.util.Queue;
4 public class BFSVacuumWorld {
5     public static void main(String[] args) {
6         // Define the neighbours for each state.
7         // The neighbours are defined as [leftMove, rightMove, suckMove]
8         int[][][] neighbors = { { { 1, 2, 3 }, // State 1
9             { 1, 6 }, // State 2
10            { 3, 4, 3 }, // State 3
11            { 3, 4, 8 }, // State 4
12            { 5, 6, 7 }, // State 5
13            { 5, 6, 6 }, // State 6
14            { 7, 8, 7 }, // State 7
15            { 7, 8, 8 } // State 8
16        };
17        // Initialise the frontier queue as a LinkedList with the first state
18        Queue<Integer> frontier = new LinkedList<Integer>();
19        frontier.add(1);
20        // Initialise the explored nodes as an empty ArrayList
21        ArrayList<Integer> explored = new ArrayList<Integer>();
22        // Breadth-First Search Loop
23        while (!frontier.isEmpty()) {
24            int currentState = frontier.poll();
25            System.out.println("Expanding State: " + currentState);
26            // Mark the current state as explored
27            explored.add(currentState);
28            // Check if goal state before expanding node
29            if (currentState == 7 || currentState == 8) {
30                System.out.println("Goal State " + currentState + " Reached!");
31                return;
32            }
33            // Expand the current state
34            for (int neighbor : neighbors[currentState - 1]) {
35                if (!explored.contains(neighbor) && !
36                    frontier.contains(neighbor)) {
37                    frontier.add(neighbor);
38                    System.out.println(" Child Node: " + neighbor);
39                }
40            }
41        }
42    }
43 }

```

```

Expanding State: 1
Child Node: 2
Child Node: 3
Expanding State: 2
Child Node: 6
Expanding State: 3
Child Node: 4
Expanding State: 6
Child Node: 5
Expanding State: 4
Child Node: 8
Expanding State: 5
Child Node: 7
Expanding State: 8
Goal State 8Reached!

```

Week 3: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: Informed Search Strategies

- Section 3.5: "Informed (Heuristic) Search" from AI:AMA (**4th ed**)
- Section 3.5.1: "Greedy best-first search" from AI:AMA (**4th ed**)
- Section 3.5.2: "A* search" from AI:AMA (**4th ed**)

Lesson 2: Design of Heuristics

- Section 3.6: "Heuristic Functions" from AI:AMA (**4th ed**)
- Section 3.6.1: "Effect of heuristic accuracy on performance" from AI:AMA (**4th ed**)
- Section 3.6.2: "Generating admissible heuristics from relaxed problems" from AI:AMA (**4th ed**)

Week 3 Further Reading:

- Section 3.5.5: "Memory-bounded search" from AI:AMA (**4th ed**)
 - *ref Section 3.5.3 from AI:AMA (3rd Edition)*

Study Session:

Thursday, 21 Sept 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Week 3: Learning Outcomes

- Design heuristics for informed search algorithms
- Implement informed search algorithms, including Greedy Best-First Search and A* Search
- Estimate the size and nature of the searches performed by informed search algorithms.

Assignments

In Canvas go to Assignments and under "Undated Assignments" Sample Paper 2 should be available to take. Give it a try before the Week 4 review.

The screenshot shows the Canvas assignments interface. At the top, there's a navigation bar with 'COM00143M > Assignments'. Below this, a sidebar on the left lists 'Home', 'Units', 'Discussions', 'Reading List', 'Announcements', 'Assignments' (which is selected), 'Panopto Recordings', and 'Kortext'. The main area is titled 'UNDATED - 23/24'. It has two sections: 'Upcoming assignments' and 'Undated assignments'. Under 'Upcoming assignments', there are two entries: 'Formative Assessment - Practice Exam (Groups 1-6)' due on Oct 4 at 5:00 and 'Artificial Intelligence and Machine Learning Exam - (Groups 1-6) [001 1.0]' due on Oct 27 at 5:00. Under 'Undated assignments', there are two entries: 'Sample Paper 1 (Groups 1-6)' worth 100 pts and 'Sample Paper 2 (Groups 1-6)' also worth 100 pts. The 'Sample Paper 2' entry is highlighted with a green oval.

Informed Search Strategies (aka heuristic search)

Unlike uniformed search, in informed search we use problem-specific knowledge to find solutions more efficiently:

- $f(n)$ - **evaluation function** - this is the function on the basis of which we expand nodes
- $h(n)$ - **heuristic function** - estimated cost of the cheapest path from the state at node n to a goal state
- $g(n)$ - **cost function** - the cost incurred in reaching node n

Best-First Search

- BFS defines a class of algorithms that use a **priority queue** based on an evaluation function $f(n)$ to expand the nodes.
- **NOT that it uses the the cost function $g(n)$ (Ali was wrong). It can, but that's not required. It uses a $f(n)$ to be defined by each approach.**

Greedy Best-First Search

- An instance of BFS where the priority queue is driven by a **heuristic function** $f(n) = h(n)$. Only the node with the lowest $h(n)$ value is selected for expansion, **it doesn't take into account the cost** of getting to that node.
- Not optimal, but is often efficient.
- Time and space complexity: $O(b^m)$ where m = maximum depth of the search space, b = branching factor
 - Can be reduced with a good heuristic function, how much this helps is problem and heuristic dependent.
- Completeness:
 - tree-search based: not complete (even in finite spaces) due to the possibility of infinite loops
 - graph-search based: complete in finite spaces, not in infinite ones

A* search

- An instance of BFS where the priority queue is driven by both the cost to reach the node $g(n)$ and a heuristic function $h(n)$.
- $f(n) = g(n) + h(n)$ = estimated cheapest solution through n
- A* is **complete**. It is **optimal**. And it is **optimally efficient**. However it will often hold a large number of nodes in memory.

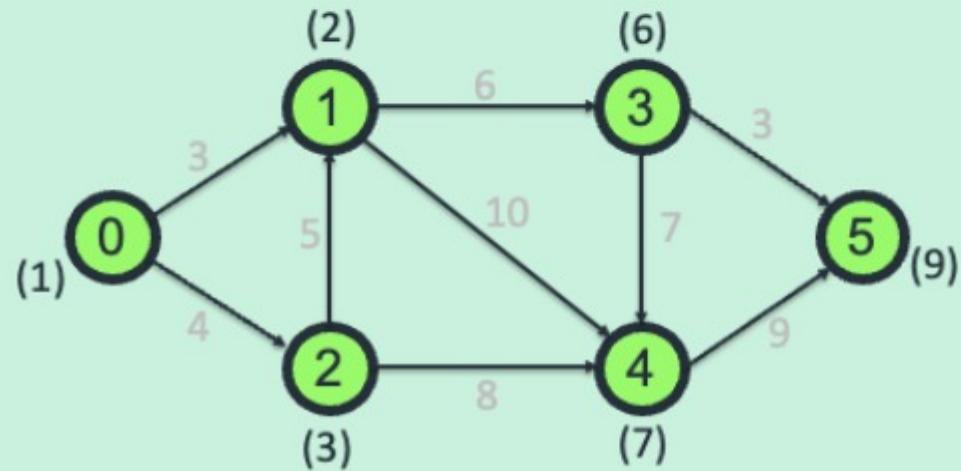
admissible heuristic: One that never overestimates the cost to reach the goal. They are by nature optimistic because they think the cost of solving the problem is less than it actually is. This also implies that $f(n)$ never overestimates the path cost.

consistency (monotonicity): A form of the triangle inequality, $h(n) \leq c(n, a, n') + h(n')$. Intuitively, it means that the estimated cost (heuristic) for going from n directly to the goal must be smaller than the actual cost of going from n to n' plus the heuristic cost estimate of going from n' to the goal.

- **consistency implies admissibility**

Greedy Best-First Search

Key idea: We expand the node which is closest to the goal, i.e. $f(n) = h(n)$. We don't take into account the cost of getting to that node.



Adjacency Matrix

	0	1	2	3	4	5
0:	0	3	4	0	0	0
1:	0	0	0	6	10	0
2:	0	5	0	0	8	0
3:	0	0	0	0	7	3
4:	0	0	0	0	0	9
5:	0	0	0	0	0	0

Heuristic
0: 1
1: 2
2: 3
3: 6
4: 7
5: 9

Not used with GBFS

Completeness:

- tree-search based: Not complete (even in finite spaces) due to the possibility of infinite loops.
- graph-search based: Complete in finite spaces but not infinite ones

Non-optimal: It may not find the optimal path if intermediate steps do not have the lowest $f(n)$.

Time and Space complexity:

- $O(b^m)$, where m is the maximum depth of the search space.
- But the complexity can be reduced with a good heuristic function

Start: Explored Nodes = []
Frontier = [0(5)]
0 is selected for expansion

0 → Explored Nodes = [0]
Frontier = [1(2), 2(3)]
1 is selected for expansion

1 → Explored Nodes = [0, 1]
Frontier = [2(3), 3(6), 4(7)]
3 is selected for expansion because 2 is not accessible from 1 (one-way arrow)

3 → Explored Nodes = [0, 1, 3]
Frontier = [2(3), 4(7), 4(7), 5(3)]
5 is selected for expansion because 2 is not accessible from 3

End: Explored Nodes = [0, 1, 3, 5]
5 is the Goal

```
Console X
<terminated> GBFS [Java Application] /Users/firef...
End: Explored Nodes = [A, B, D, E, G, F]
0 --> 1 --> 3 --> 5
H is the Goal
miro
```

A* Search

- While GBFS aims, at each step, to get as close as possible to the goal. A* search tries to minimize the total estimated solution cost as $f(n)=h(n)+g(n)$
- So we organize our priority queue based on the **SUM** of the heuristic function and the actual cost of getting to node n .
- This is similar to uniform cost search but we use $g+h$ instead of g .

Completeness

- Completeness requires that there be a finite number of nodes with a cost less than or equal to C^* , where C^* is the cost of the optimal solution path.
 - This is satisfied if all step costs exceed some threshold **e** and if **b**, the branching factor, is finite.

Optimality:

- tree-search based: optimal if the heuristic is **admissible** => it never over estimates the cost of reaching the goal.
 - this also implies that $f(n)$ never overestimates the path cost
- graph-search based: optimal if the heuristic is **consistent**.

Optimally Efficient:

- A* is optimally efficient for any given **consistent** heuristic.
 - this means that no other optimal algorithm is guaranteed to expand fewer nodes than A* (except possibly for tie-breaking rules)
 - this is because A* is guaranteed to expand all nodes with $f(n) < C^*$, otherwise there would be a risk of missing the optimal solution.

Time & Space Complexity:

- Because A* keeps all generated nodes in memory, it usually runs out of space long before it runs out of time.
- Absolute error of the heuristic: $\delta = h^* - h$
- Relative error of the heuristic: $e = (\delta / h^*) \cdot 100\%$
- Complexity results depend on the assumptions made about the state space:
 - consider a state space that is a tree with reversible actions, with a single goal, then the time complexity of A* is $O(b^d e d)$ where d is the solution depth and e is the relative error.
- When the state space has many goal states, particularly near-optimal goal states, then the search process can be led astray from the optimal path and there is an extra cost proportional to the number of goals whose cost is within a factor e of the optimal cost.
- In the graph search version there can be exponentially many states with $f(n) < C^*$ even if the absolute error is bounded by a constant.
- Overall the complexity of A* makes it impractical to insist on finding an optimal solution.

Contours

- We can draw contours in the state space, to indicate all nodes whose $f(n)$ is less than or equal to some pre-specified number.
- A* search expands all nodes for which $f(n) < C^*$ where C^* is the cost of the optimal solution path.
- It may also expand some nodes on the goal contour itself, i.e. for which $f(n)=C^*$, before selecting a goal node.
- A* expands no nodes with $f(n) > C^*$. We can say that those nodes/subtrees are pruned.
- The fact that it is guaranteed to expand all nodes with $f(n) < C^*$ is why it is guaranteed to be optimal and the fact that it does not expand nodes with $f(n) > C$ makes it optimally efficient.

e.g $C^* = 420$

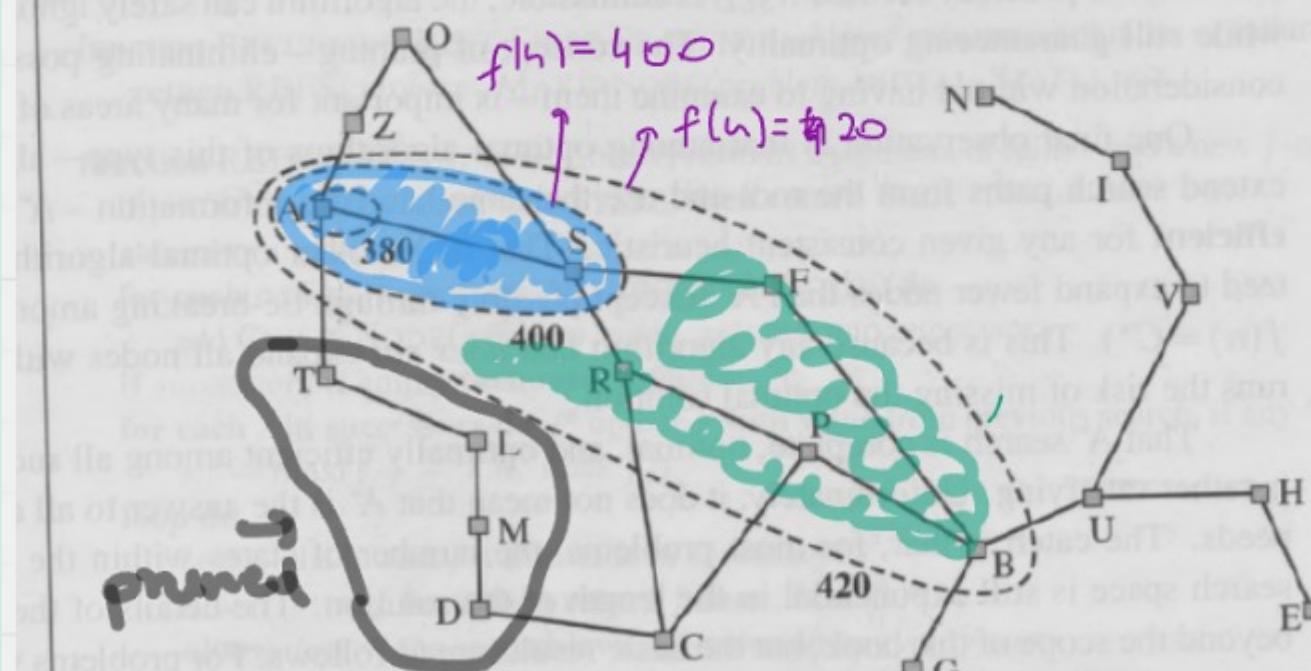


Figure 3.25 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have f -costs less than or equal to the contour value.

3.4 Quiz: Informed Search Strategies

KEYWORDS: greedy-best-first search; node expansion;

Question 1 iterative deepening

1 pts

为什么贪婪最佳优先搜索比迭代深化扩展的节点更少?选择两个原因

Why might Greedy Best-First Search expand fewer nodes than Iterative Deepening? Select two reasons.

It always produces optimal solutions. 错误 - 贪婪最佳优先搜索不一定会产生最佳解决方案。

It uses an admissible heuristic, whereas Iterative Deepening uses an inadmissible heuristic.

Greedy Best-First Search has a transition model, whereas Iterative Deepening does not.

The heuristic helps Greedy Best-First Search to select nodes closer to the goal.

启发式帮助贪婪最佳优先搜索选择更接近目标的节点。

It does not re-generate nodes, and Iterative Deepening does.

它不会重新生成节点,而迭代加深会重新生成节点。

KEYWORDS: admissible heuristic

1 pts

启发式的可接受性意味着什么?

What does it mean for a heuristic to be admissible?

启发式方法永远不会高估达到目标状态的成本。

The heuristic never produces an over-estimate of the number of actions to reach a goal.

The heuristic never produces an over-estimate of the cost to reach a goal state.

The heuristic always produces an over-estimate of the cost to reach a goal state.

The heuristic is a reasonably accurate estimate of the cost to reach a goal state.

KEYWORDS: admissible heuristic

1 pts

假设我有一个总是返回 0 的启发式函数。它可以接受吗?

Suppose I have a heuristic function that always returns 0. Is it admissible?

No - heuristics must somehow estimate the cost of reaching a goal state, and this function does not.

Yes - it never over-estimates the cost of reaching a goal state. 它永远不会高估达到目标状态的成本。

Yes - it never under-estimates the cost of reaching a goal state.

No - an admissible heuristic cannot return the same value for two drastically different states.

KEYWORDS: breadth-first search; iterative deepening; special class; step cost 1; greedy best-first search; optimal solutions; heuristic

Question 4

1 pts

Breadth-First Search and Iterative Deepening produce optimal solutions for a special class of problems where the step cost is always 1. Does Greedy Best-First Search also produce optimal solutions on the same class of problems? 广度优先搜索和迭代深化为一类特殊问题提供最优解,其中步骤成本始终为

1。贪婪最佳优先搜索是否也为同一类问题产生最优解?

Yes - it will do the same search as Breadth-First Search on problems with that property.

No - the heuristic may guide the search to a sub-optimal solution.

不。启发式可能会引导搜索到次优解决方案。
Yes - the heuristic will guide it towards a goal so it will expand a subset of the nodes that Breadth-First Search would expand.

Question 6 KEYWORDS: contour definition

1 pts

什么是轮廓?

What is a contour?

A solution with the smallest possible path cost

A boundary in the state space enclosing all reachable states

A heuristic that employs potential fields

A boundary in the state space that encloses all states with f-cost less than or equal to the contour value.

状态空间中的边界,包围 f 成本小于或等于轮廓值的所有状态。

KEYWORDS: perfect heuristic properties; over-estimate; under-estimate; goal

Question 5

1 pts

Suppose my heuristic $h(n)$ for the Romania problem is the exact minimal cost to reach Bucharest, computed in advance. In other words, $h(n)$ never over-estimates or under-estimates the cost of reaching the goal. It is the perfect heuristic. What are the properties of this heuristic?

Select all correct answers.

假设我针对罗马尼亚问题的启发式 $h(n)$ 是预先计算的到达布加勒斯特的确切最小成本。换句话说, $h(n)$ 永远不会高估或低估达到目标的成本。这是完美的启发式方法。这种启发法有什么特点?

Admissible

Composite

Useful for reducing search

Based on a relaxation

Consistent

可受理

持续的

Question 7 KEYWORDS: inadmissible heuristic; algorithms; optimal solution

1 pts

If the heuristic is inadmissible, which of these algorithms is guaranteed to produce the optimal solution?
如果启发式方法不可接受,那么哪些算法可以保证产生最佳解决方案?

Both Greedy Best-First Search and A* Search

Greedy Best-First Search but not A* Search

Neither Greedy Best-First Search nor A* Search

既不是贪婪最佳优先搜索也不是A*搜索

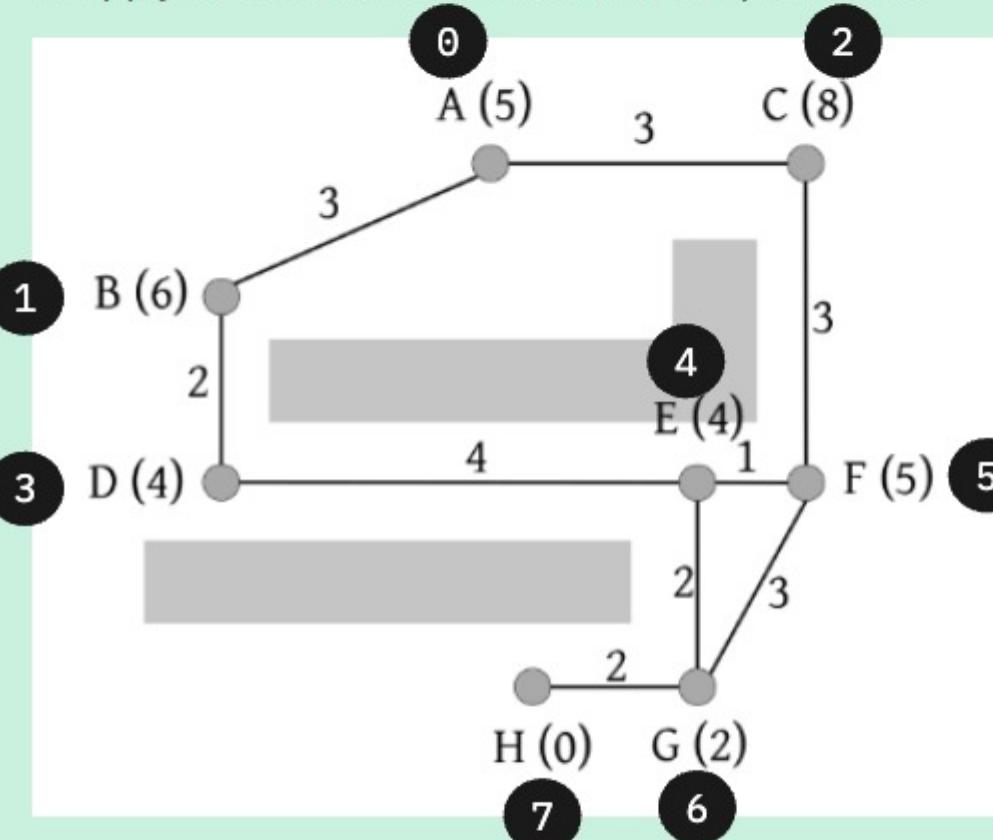
A* Search but not Greedy Best-First Search

启发法不可接受,则贪婪最佳优先搜索和 A* 搜索都不能保证产生最优解决方案。不可接受的启发式方法可能会导致这两种搜索算法出现次优或不正确的解决方案。

Activity 1

- The task is to **get the robot from A to H** on the other side of a room, avoiding two large obstacles on the way.
- The heuristic is the **Manhattan Distance**; calculated by taking the sum of the distances between the x and y coordinates.
 - The manhattan distance is both **consistent** and **admissible**.

- Apply Greedy Best First Search (Graph Based) to this problem. Is this optimal?
- Apply A* Search (Graph Based) to this problem, note the frontier in each step and the f-cost.
- Apply A* Search (Tree Based) to this problem.



States and Heuristic Values	
State	$h(n)$
A	5
B	6
C	8
D	4
E	4
F	5
G	2
H	0

Greedy Best First Search (Graph Based)

- priority queue using $h(n)$

Start: Explored Nodes = []
 Frontier = [A(5)]
 A is selected for expansion
 A → Explored Nodes = [A]
 Frontier = [B(6), C(8)]
 B is selected for expansion
 B → Explored Nodes = [A, B]
 Frontier = [C(8), D(4)]
 D is selected for expansion
 D → Explored Nodes = [A, B, D]
 Frontier = [C(8), E(4)]
 E is selected for expansion
 E → Explored Nodes = [A, B, D, E]
 Frontier = [C(8), F(5), G(2)]
 G is selected for expansion
 G → Explored Nodes = [A, B, D, E, G]
 Frontier = [C(8), F(5), H(0)]
 H is selected for expansion
 End: Explored Nodes = [A, B, D, E, G, H]
 H is the Goal

```
<terminated> GBFS [Java Application] /Users/firefly/.p2/pod
0 --> 1 --> 3 --> 4 --> 6 --> 7
A --> B --> D --> E --> G --> H
```

Not Optimal

A* Search (Graph Based)

- priority queue using $f(n) = g(n) + h(n)$

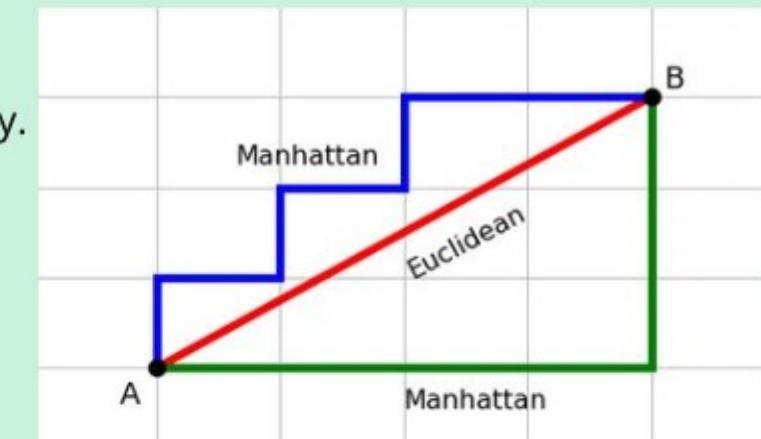
Start: Explored Nodes = []
 Frontier = [A]
 A is selected for expansion
 A → Explored Nodes = [A]
 Frontier = [B(6+3 = 9), C(3+8=11)]
 B is selected for expansion
 B → Explored Nodes = [A, B]
 Frontier = [C(11), D(2+3+(4)=9)]
 D is selected for expansion
 D → Explored Nodes = [A, B, D]
 Frontier = [C(11), E(3+2+4+(4)=13)]
 C is selected for expansion
 C → Explored Nodes = [A, B, D, C]
 Frontier = [E(13), F(3+3+(5)=11)]
 F is selected for expansion
 F → Explored Nodes = [A, B, D, C, F]
 Frontier = [E(13), E(3+3+1+(4)=11),
 G(3+3+3+(2)=11)]
 G is arbitrarily selected for expansion
 G → Explored Nodes = [A, B, D, C, F, G]
 Frontier = [E(13), E(11), E(3+3+3+2+(4)=16),
 H(3+3+3+2+(0)=11)]
 H is arbitrarily selected for expansion
 End: Explored Nodes = [A, B, D, C, F, G, H]
 H is the Goal

A* Search (Tree Based)

- priority queue using $f(n) = g(n) + h(n)$

Start: Frontier = [A]
 A is selected for expansion
 A → Frontier = [B(6+(3 = 9), C(3+(8)=11)]
 B is selected for expansion
 B → Frontier = [C(11), D(2+3+(4)=9),
 A(3+3+(5)=11)]
 D is selected for expansion
 D → Frontier = [C(11), A(11), B(3+2+2+(6)=13),
 E(3+2+4+(4)=13)]
 C is arbitrarily selected for expansion
 C → Frontier = [A(11), B(13), E(13),
 A(3+3+(8)=14), F(3+3+(5)=11)]
 F is arbitrarily selected for expansion
 F → Frontier = [A(11), B(13), E(13), A(14),
 E(3+3+1+(4)=11), C(3+3+3+(8)=17),
 G(3+3+3+(2)=11)]
 G is arbitrarily selected for expansion
 G → Frontier = [A(11), B(13), E(13), A(14),
 E(11), C(17), H(3+3+3+2+(0)=11),
 F(3+3+3+3+(5)=17), E(3+3+3+2+(4)=15)]
 H is arbitrarily selected for expansion
 End: Explored Nodes = [A, B, D, C, F, G, H]
 H is the Goal

Unlikely traversal, would more likely have been messier.



Solution provided by York is highly convenient, it can find the same path as the graph based version, but that is dependent on arbitrary selections that don't follow a discernible pattern...FIFO one time, LIFO the next. It is just as likely to be significantly longer, visiting A multiple times, and potentially getting stuck in a loop.

miro

Heuristics

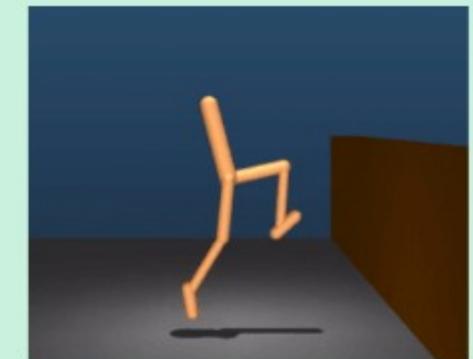
+

=

Designing a Heuristic:

A Heuristic is just a rule of thumb to let you guesstimate how close you are to the goal without looking at all of the options.

Why do we need Heuristics? This diagram gives a good reason -- the "learning to walk" AI has a more relaxed version of the problem where only the basics are included. This allows the problem to be solved more easily before it is subsequently scaled up which would increase the space and time complexity.



Simple Heuristic:

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

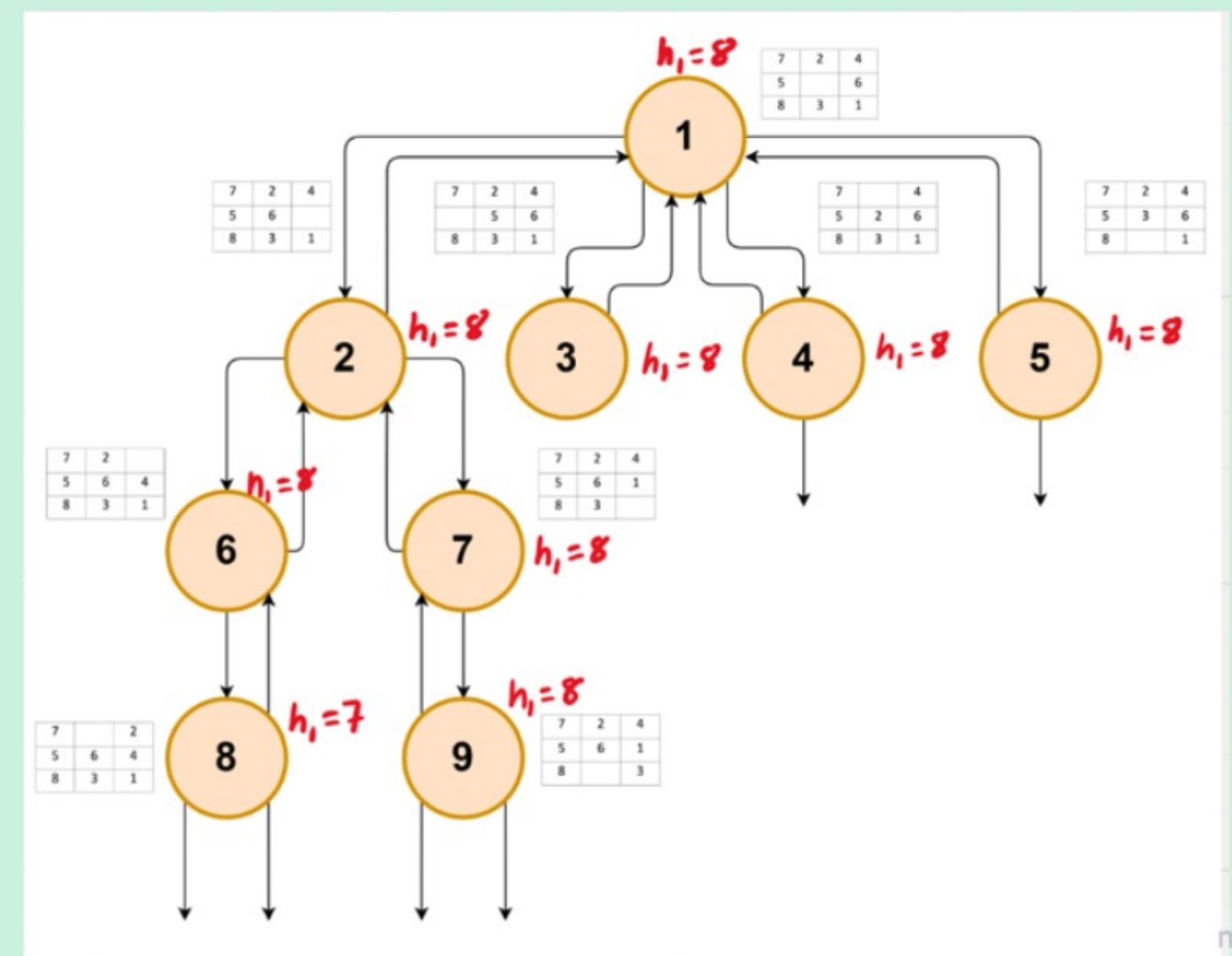
Goal State

There are multiple ways of relaxing the 8 puzzle.

One simple method is to use the "misplaced tiles" approach. This simply means "count the number of tiles which are not in the place they need to be for the goal state." I am going to give this heuristic the name h_1 . In the start state on the left, this is all of them, therefore giving us a value of $h(n) = 8$

From the state state, I can move the blank tile around, exploring new states and draw a graph of those states as I find them. All have a path cost of 1. There are 4 possible states adjacent to state 1. As it turns out, all of these also have $h(n) = 8$, so it comes down to my algorithm to choose which to explore next. I pick left first (just preference) and explored state 6.

That also has $h(n) = 8$, but now I am faced with options of "move the piece back and move to a state with $h(n) = 8$ or move 2 blocks and go to state 8 which has $h(n) = 7$. This is closer to my goal and so it is more appealing.



Comparing Heuristics

- The **effective branching factor** is a useful measure of the accuracy of a heuristic in guiding the search towards a goal. It is an estimate of the real branching factor (not worst case) of the search tree. Combined with the depth, it can be a powerful statistic.
- Alternatively we can use **domination**, where h_2 is always larger than or equal to h_1 , and since they are both admissible then h_2 must be more accurate than h_1 . Greater accuracy would imply A* performs better with the larger heuristic.
 - This is not typically useful as it can be impossible to prove.
- **relaxation** - where some of the rules of the real problem are removed to create a relaxed problem and a heuristic is derived from that.

Read *Artificial Intelligence: A Modern Approach* Chapter 3 Section 3.6.2. They discuss three ways of relaxing the rules of the 8-puzzle, and two of those relaxations give us the heuristics h_1 and h_2 .

Have a look at the other relaxation ("a tile can move from square A to square B if B is blank"). In this relaxation, the pieces are allowed to move any distance (and any direction) across the board in one step, as long as they move into the blank space.

Try solving this relaxation of the 8-puzzle a few times from different initial states. A heuristic (let's call it h_3) can be derived from this relaxation as well. How does h_3 compare to h_1 ? Is it always the same, or is it different sometimes?

Click the button below for the answer.

h_3 is almost the same as h_1 , the misplaced tiles heuristic. In h_1 , the heuristic value is just the number of tiles that are in the wrong square, because each tile can be moved directly (in one step) to the correct square. In h_3 , a tile can be moved in one step to the correct place when the correct place is blank. To play out the relaxed game, there are two types of move:

1. If the blank space is in the location where tile X should be, move X to the blank space.
2. If the blank space is in the same location as it is in the goal state, then move a misplaced tile to the blank space.

Move type 1 makes obvious progress towards the goal, and move type 2 creates a state where you are guaranteed to be able to apply move type 1 at least once. In a particular state, if it turns out that only moves of type 1 are needed, then $h_3=h_1$. Otherwise, if some moves of type 2 are needed, then $h_3>h_1$. Therefore h_3 is a stronger heuristic than h_1 .

Activity 2: Monkey Banana Problem

A 1 metre tall monkey is in a room where some bananas are suspended from the ceiling, which is 4 metres high. He would like to get the bananas. The room contains three stackable, movable, climbable crates that are 1 metre tall. All the crates are on the floor to begin with.

The problem is formulated as follows. Let's call the crates A, B, and C. For each crate, it is either on the floor or on top of another crate. A crate can be moved if, and only if, there is no other crate on top of it.

There are four kinds of actions:

- Put a crate on the floor.
- Put a crate on top of another crate.
- Climb a stack of crates.
- Take the bananas.

The monkey is an excellent climber and can climb any stack of crates, however high, in one action. One optimal solution to this problem is: put B onto A, put C onto B, climb onto C, take the bananas.

York have relaxed the problem to the point that crates can move straight from their current location to the goal location without taking any steps in between. They are correct (as they are the examiners) but Keith explored a heuristic using manhattan distance and followed the rule that agents could only move one block at a time.

Formulate the Problem:

Initial State: The problem doesn't define this, so an assumption is being made of the worst case scenario that all of the crates are in the wrong location.

Goal State: The solution makes the assumption that the monkey is not already at the top of the stack -and- that it has the bananas.

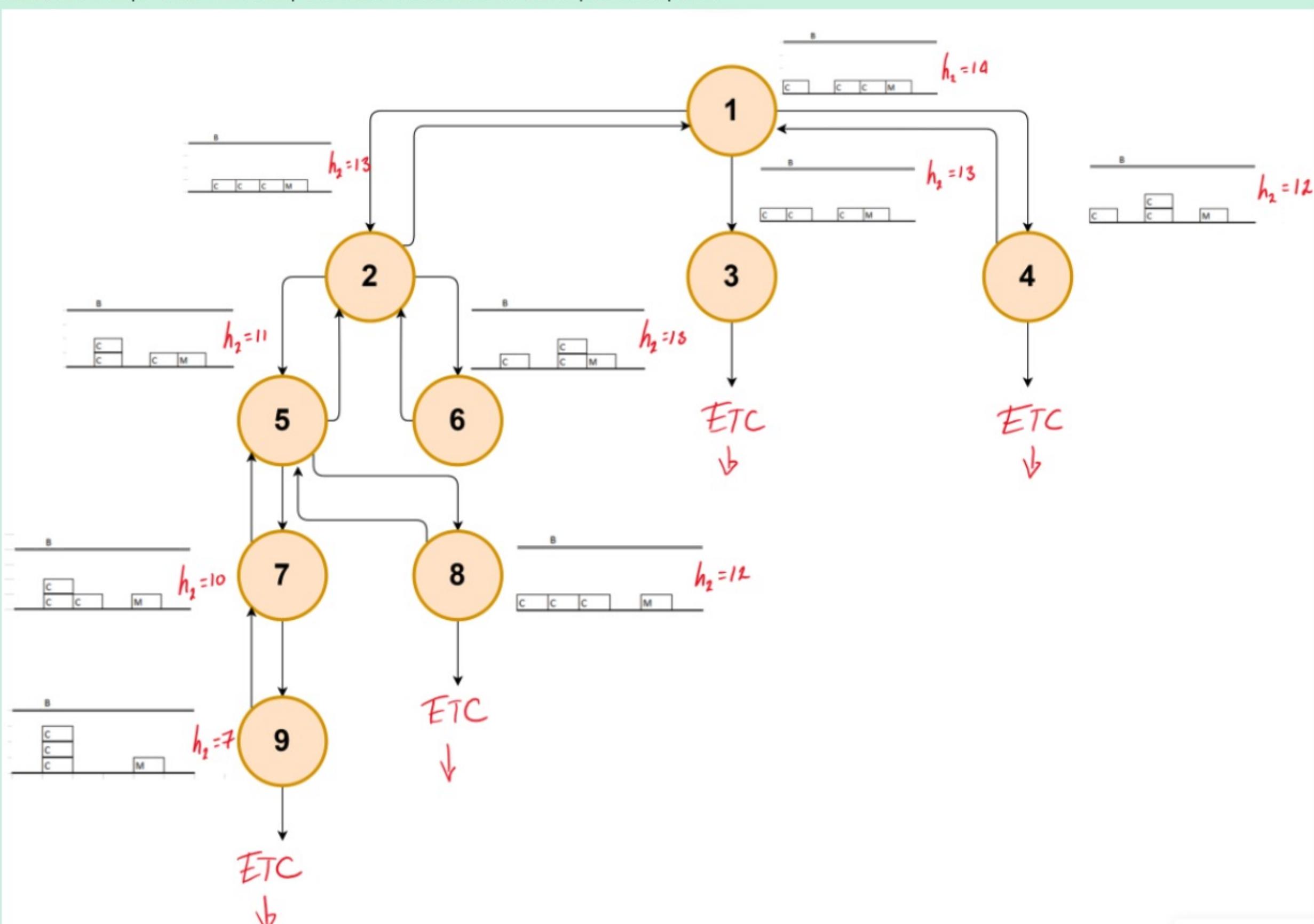
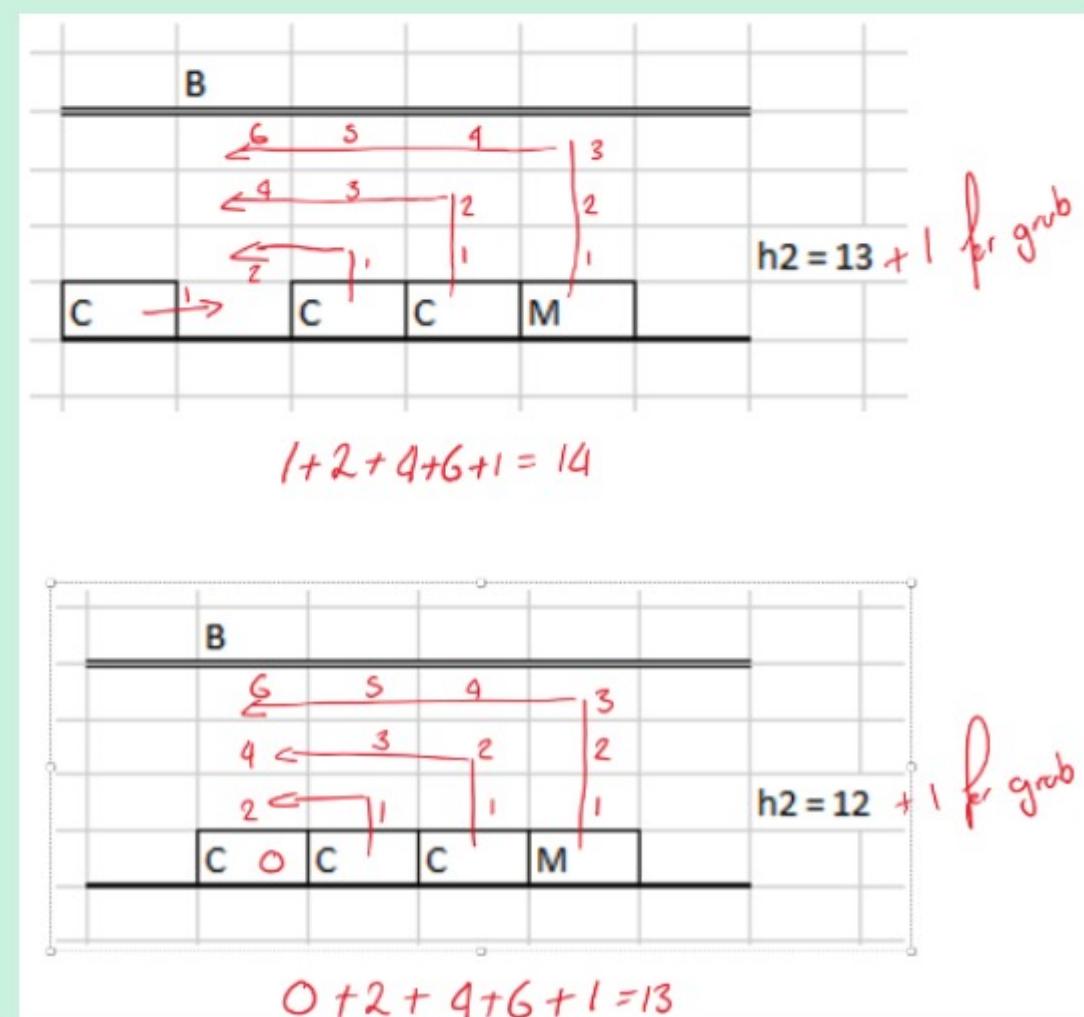
States: The state is determined by the position of the crates, the position of the monkey, and whether or not the monkey has grabbed the bananas. The state space is difficult to estimate without defining the width of the floor.

Actions: When on the ground, a crate has the possible actions of: Left, Right, and Stack. Once stacked, a crate cannot move. When on the ground the monkey has the possible actions of Left, Right, Climb. When on the stack, the monkey cannot move left or right, but has the possible action of "Grab". The agents can only move one state at a time.

Transitional Model: The actions have their expected effects, moving the agents in the respective direction in the room, except where when they are on the stack.

Goal Test: Checks to see if the monkey can reach up and grab the bananas (Cost = 1).

Path Cost: Each step costs 1, so the path cost is the number of steps in the path.



Maze Finding

States: The state is determined by the agent's location in the maze. In this simplistic maze there are 37 states. This is simply adding up the number of possible positions the agent could be in. Larger environments could have n states.

Initial State: The agent is in the bottom left hand corner of the maze, shown at cell reference A14

Actions: Each state has up to 4 possible actions, Up, Down, Left, Right. Depending on their positions in the maze, not all states have all possible actions. The agent can only move on estate at a time.

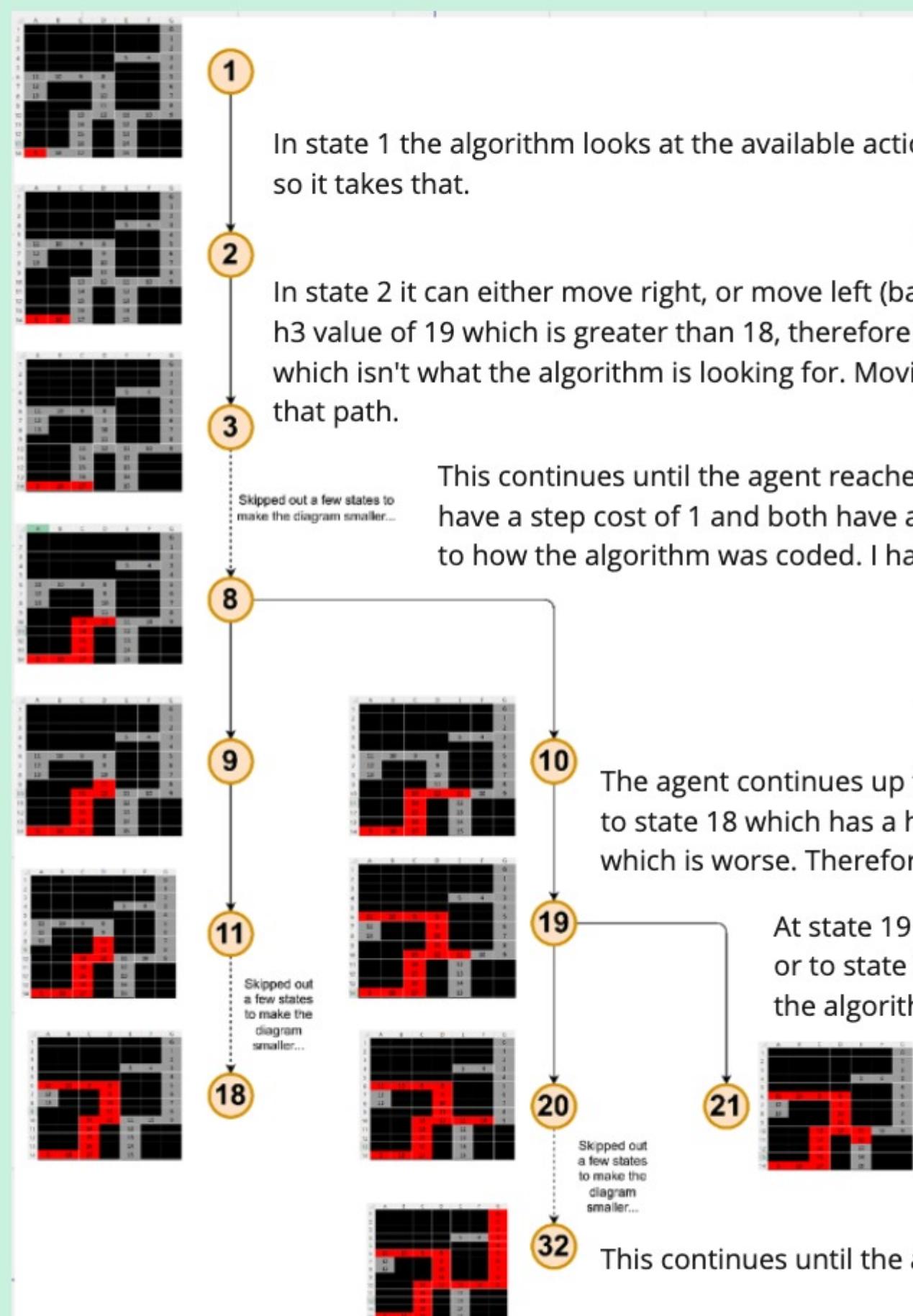
Transitional Model: The actions have their expected effects, moving the agents in the respective direction in the maze, except where that movement is blocked by a wall or the edge of the maze.

Goal Test: Checks to see if the agent is in the cell in the top left corner of the maze, shown as cell reference G1.

	A	B	C	D	E	F	G
1							G
2							1
3							2
4					5	4	3
5							4
6	11	10	9	8			5
7	12			9			6
8	13			10			7
9				11			8
10			13	12	11	10	9
11			14		12		
12			15		13		
13			16		14		
14	S	18	17		15		

Path Cost: Each step costs 1, so the path cost is the number of steps in the path.

Heuristic h_3 : Agent distance from goal, represented here by the Manhattan Distance.





Reading list prompts

For further reading this week I have suggested a section of the core text that discusses some more search algorithms based on A* Search. Memory usage is the Achilles heel of A* Search in practice, but there are several ways of mitigating the problem. The section discusses a number of them, the simplest of which is a combination of Iterative Deepening with A* Search. These algorithms extend the reach of informed search strategies to solve much larger problems where A* Search would run out of memory.

Iterative Deepening with A* Search (IDA*):

- IDA* is a kind of iterative deepening depth-first search (DFS) that adopts the A* search's idea of a heuristic function to assess the remaining cost to reach the goal.
 - $g(n)$ = actual cost
 - $h(n)$ = estimated cost from the current node to the goal state
 - **f(n) = actual cost + estimated cost**; f(n) is called the **f-score or f-cost**
 - A lower F-score indicates that the node is closer to the goal state and will be expanded before a node with a higher F-score.
 - Iterative-deepening A* search (IDA*) is to A* what iterative-deepening search is to depth-first.
- IDA* is a **graph traversal**.
- IDA* gives us the benefits of A* without the requirement to keep all reached states in memory, at a cost of visiting some states multiple times.
 - It's a **DFS algorithm** so it uses less memory than the A*.
 - **Addresses the space complexity issue of A* Search.**
 - It is a very important and commonly used algorithm for problems that do not fit in memory.
 - It is efficient in handling large numbers of states and large branch factors.
- Standard Iterative Deepening vs. IDA:
 - In standard iterative deepening the cutoff is the depth, d , which is increased by one each iteration.
 - In IDA* the cutoff is the f -cost ($g + h$).
 - At each iteration, the cutoff value is the smallest f -cost of any node that exceeded the cutoff on the previous iteration.
 - Each iteration exhaustively searches an **f -contour**, finds a node just beyond that contour, and uses that node's f -cost as the next contour.
- IDA* heuristic is **admissible** because it never overestimates the cost of reaching the goal.
- IDA* is **optimal** if a solution exists.
- IDA* works very well in problems where each path's f -cost is an integer (like the 8-puzzle).
- For a problem where every node has a different f -cost, each new contour might contain only one new node, and the number of iterations could be equal to the number of states
- IDA* can be **slower** than A* or BFS because it explores and repeats the explored node iteratively.
- IDA* takes **more time and power than A***.

How does IDA* work?

1. **Initialization:** Set the root node as the current node, and find the f-score.
2. **Set threshold:** Set the cost limit as a **threshold** for a **node** i.e the **maximum f-cost** allowed for that node for further explorations.
3. **Node Expansion:** Expand the current node to its children and find f-scores.
4. **Pruning:** If for any **node** the **f-cost > threshold**, prune that node because it's considered too expensive for that node. and store it in the **explored node list**.
5. **Return Path:** If the **Goal node** is found then return the **path** from the start node Goal node.
6. **Update the Threshold:** If the Goal node is not found then **repeat from step 2** by changing the threshold with the minimum pruned value from the **visited node list**. And Continue it until you reach the goal node.

Week 4: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: Principles of logic and propositional logic

- Section 7.2: "The Wumpus World" from AI:AMA (**4th ed**)
- Section 7.3: "Logic" from AI:AMA (**4th ed**)
- Section 7.4: "Propositional Logic" from AI:AMA (**4th ed**)

Lesson 2: Inference in propositional logic

- Section 7.5.2: "Proof by resolution" from AI:AMA (**4th ed**)
- Section 7.5.3: "Horn clauses and definite clauses", from AI:AMA (**4th ed**)
- Section 7.5.4, "Forward and backward chaining" from AI:AMA (**4th ed**)

Week 4 Further Reading:

- Section 8: "First-Order Logic" from AI:AMA (**4th ed**)

Study Session:

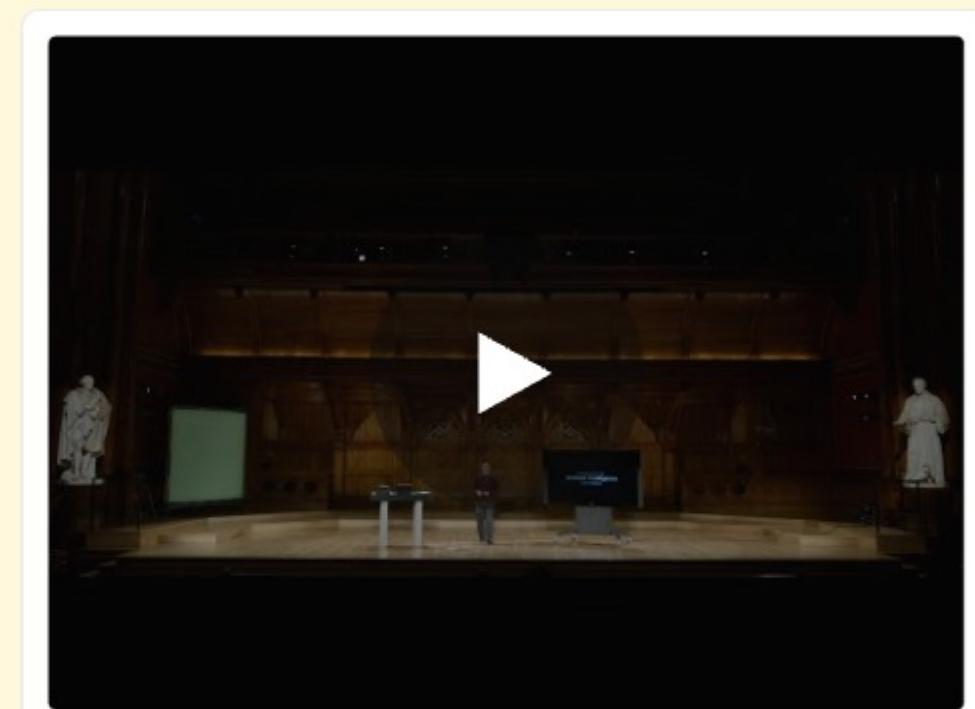
Thursday, 5 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Week 4: Learning Outcomes

- Represent scenarios in propositional logic, and be able to perform rearrangements of logic sentences into conjunctive normal form.
- Apply reasoning algorithms for propositional logic with Horn clauses: forward and backward chaining.
- Review the Sample Paper 2 Quiz

CS50
Lecture 1
has good
overlap
with this
week



Knowledge - Lecture 1 - CS50's Introduction to Artificial Int...



| Updated 26-09-2023 @ 14:39 GMT-07:00



Propositional Logic - Facts and the relationship between facts.

Propositional Symbols: P, Q, R

Literal: A propositional symbol or its opposite; e.g. P, $\neg P$.

P	Q	$\neg P$	$\neg Q$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

Logical connectives:

- negation, "not": \neg
- and: \wedge
- or: \vee
- implication, "if/then" e.g. if P is true then Q is true: \rightarrow or \Rightarrow
- bi-conditional, iff = "if and only if": \leftrightarrow or \Leftrightarrow
- entailment (aka semantically entails), "means": \models (**double turnstile**)
 - $P \models Q$ = in every model, every possible world, in which P is true, Q is also true.
- inference (aka syntactically entails), "proves": \vdash (**turnstile**)
 - $\vdash P$ = given P
 - $P \vdash Q$ = from P, I know that Q -or- Q is provable from P
 - $KB \vdash_i S$ = S is inferred from KB using algorithm i

Knowledge Based Agents: Agents that reason by operating on internal representations of knowledge

Sentence: An assertion about the world in a knowledge representation language

Knowledge Base (KB):

- A set of sentences in propositional logic that our AI (knowledge-based agent) knows are true.
- A store of knowledge that the system has.

Inference: The process of deriving new sentences from old ones.

Inference Algorithm (i):

- A procedure for deriving a sentence from the Knowledge Base.
- If an inference algorithm i can derive sentence S from the KB, we write:
 $KB \vdash_i S$
- The algorithm is **sound** if it derives only sentences that are entailed by KB, i.e. it is always **correct**.
- The algorithm is **complete** if it can derive any sentence that is entailed by KB, it knows everything.
- **(!) If the KB is true in the real world then any sentence S derived from KB by a sound inference procedure is also true in the real world.**

Grounding:

- Grounding captures the connection between logical reasoning processes and the real environment in which the agent exists.
- The agents sensors create a connection between the world and the KB.
- The meaning and truth of percept sentences are defined by the processes of sensing and sentence construction that produce them.
- Learning is fallible. Our observation of a particular fact may not generalize.

Truth Tables

Model: A precise representation of a possible world which allows us to decide (in a completely precise and algorithmic way) whether sentences are true or false.

Syntax: Describes the structure of sentences in the logic.

Semantics: Defines what sentences in the language mean.

Tautology: A sentence that always holds true, e.g. $p \vee \neg p$ (the law of the excluded middle).

- Not to be confused with $p \wedge \neg p$ which is an empty set and always resolves to false.

Shamelessly stolen from skeletman because they have a MUCH better grasp on this than I do.

We can assign a **truth value** to each sentence. Each connective has an associated rule for how true values must behave. If these rules are satisfied then the assignment is called a **valuation**. We can visualise these rules with a **truth table**.

In practice, a sentence is a tautology if it has a 1 in every row of its column in a truth table.

Figure 2.7: The truth table for *Pierce's Law*

Pierce's Law: $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$

p	q	$p \Rightarrow q$	$(p \Rightarrow q) \Rightarrow p$	$((p \Rightarrow q) \Rightarrow p) \Rightarrow p$
0	0	1	0	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

Model Checking: Enumerates all possible models to check that a statement s is true in all models in which KB is true, i.e. that: $M(\text{KB}) \subseteq M(s)$

These become the proofs for the subsequent laws

Figure 2.2: Truth tables for logical connectives

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

p	$\neg p$
0	1
1	0

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

p	q	$p \Leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2.4: The truth table for $\neg(p \Rightarrow \neg q)$

p	q	$\neg q$	$p \Rightarrow \neg q$	$\neg(p \Rightarrow \neg q)$
0	0	1	1	0
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

Notice that the truth table for $\neg(p \Rightarrow \neg q)$ is the same as for $p \wedge q$.

Figure 2.5: The truth table for $\neg p \Rightarrow q$

p	q	$\neg p$	$\neg p \Rightarrow q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

Notice that the truth table for $\neg p \Rightarrow q$ is the same as for $p \vee q$.

Inference Rules Cheat Sheet

Rule of Thumb for Conversion to Conjunctive Normal Form (CNF):

- Eliminate bi-conditionals
- Eliminate implications
- Move \neg inwards using De Morgan's Laws
- Use distributive law to distribute ors (\vee) wherever possible.

Example:

$$(p \vee q) \Rightarrow r$$

$$\neg(p \vee q) \vee r \quad \text{implication elimination}$$

$$(\neg p \vee \neg q) \vee r \quad \text{De Morgan's Law}$$

$$(\neg p \vee r) \wedge (\neg q \vee r) \quad \text{distributive law}$$

From Canvas 4.6 Lesson 2:

For the exam you may need to use the [implication elimination](#) rule to convert an implication into a disjunction. You do *not* need to be able to convert all propositional logic formulas into CNF.

<p>And Elimination the elimination of <i>and</i></p> $\frac{p \wedge q}{p} \quad \begin{array}{l} \text{if } p \text{ and } q \text{ are true} \\ \therefore \\ p \text{ is true} \end{array}$	<p>Modus Ponens the application of <i>implication</i></p> $\frac{p \Rightarrow q}{\frac{p}{q}} \quad \begin{array}{l} \text{if } p \text{ implies } q \\ \text{and } p \text{ is true} \\ \therefore \\ q \text{ is true} \end{array}$	<p>Associative Laws removal of parentheses</p> $p \wedge q \wedge r \equiv ((p \wedge q) \wedge r) \equiv (p \wedge (q \wedge r))$ $p \vee q \vee r \equiv ((p \vee q) \vee r) \equiv (p \vee (q \vee r))$
<p>Biconditional Elimination the application of <i>biconditionals</i></p> $\frac{\frac{p \Leftrightarrow q}{(p \Rightarrow q) \wedge (q \Rightarrow p)}}{\frac{\therefore}{\begin{array}{l} \text{if } p \text{ if and only if } q \\ \text{either } p \text{ or } q \text{ is true} \end{array}}} \quad \begin{array}{l} \text{if } p \text{ if and only if } q \\ \therefore \\ \text{either } p \text{ or } q \text{ is true} \end{array}$	<p>Hypothetical Syllogism</p> $\frac{\frac{p \Rightarrow q}{q \Rightarrow r}}{\frac{p \Rightarrow r}{\therefore p \text{ implies } r}} \quad \begin{array}{l} \text{if } p \text{ implies } q \\ \text{and } q \text{ implies } r \\ \therefore \\ p \text{ implies } r \end{array}$	<p>Resolution Inference Rule</p> $\frac{p \vee q}{\frac{\neg p \vee r}{\therefore q \vee r}} \quad \begin{array}{l} p \vee q \\ \neg p \vee r \\ \therefore q \vee r \end{array}$
<p>Implication Elimination turning <i>implication</i> into <i>or</i></p> $\frac{\frac{p \Rightarrow q}{\neg p \vee q}}{\frac{\therefore}{\begin{array}{l} \text{if } p \text{ implies } q \\ \therefore \\ \text{either } p \text{ or } q \text{ is true} \end{array}}} \quad \begin{array}{l} \text{if } p \text{ implies } q \\ \therefore \\ \text{either } p \text{ or } q \text{ is true} \end{array}$	<p>De Morgan's Laws turning <i>and</i> into <i>or</i></p> $\frac{\frac{\neg(p \wedge q)}{\neg p \vee \neg q}}{\frac{\neg(p \vee q)}{\neg p \wedge \neg q}} \quad \begin{array}{l} \neg(p \wedge q) \\ \neg p \vee \neg q \\ \neg(p \vee q) \\ \neg p \wedge \neg q \end{array}$	<p>Unit Resolution Rule</p> $\frac{p \vee q}{\frac{\neg p}{\frac{\therefore}{\begin{array}{l} \text{if } p \text{ or } q \text{ is true} \\ \text{and } p \text{ is false} \\ \therefore \\ q \text{ is true} \end{array}}}} \quad \begin{array}{l} p \vee q \\ \neg p \\ \therefore \\ q \text{ is true} \end{array}$
<p>Contraposition</p> $(p \Rightarrow q) \equiv (\neg p \Rightarrow \neg q)$	<p>Distributive Laws applying algebra to logic</p> $\frac{(p \wedge (q \vee r))}{(p \wedge q) \vee (p \wedge r)}$ $\frac{(p \vee (q \wedge r))}{(p \vee q) \wedge (p \vee r)}$	<p>Double Negation Elimination the removal of <i>not</i></p> $\frac{\neg(\neg p)}{p} \quad \begin{array}{l} \neg(\neg p) \\ \therefore \\ p \text{ is true} \end{array} \quad \begin{array}{l} \text{if } p \text{ is not false} \\ \therefore \\ p \text{ is true} \end{array}$
<p>Idempotent Laws</p> $p \vee p \equiv p$ $p \wedge p \equiv p$		<p>Commutative Laws</p> $(p \wedge q) \equiv (q \wedge p)$ $(p \vee q) \equiv (q \vee p)$

Activity 1

If the unicorn is mythical, then it is immortal.
If it is not mythical, then it is mortal and a mammal.
If it is either immortal or a mammal, then it is horned.
If it is horned it is magical.

Ali K's Solution:

S1: Mythical $\Rightarrow \neg$ Mortal
S2: \neg Mythical \Rightarrow Mortal \wedge Mammal
S3: \neg Mortal \vee Mammal \Rightarrow Horned
S4 Horned \Rightarrow Magical

S1: Mythical $\Rightarrow \neg$ Mortal

Using **contraposition**: $\neg(\neg\text{Mortal}) \Rightarrow \neg\text{Mythical}$

double negation elimination: Mortal $\Rightarrow \neg\text{Mythical}$

S2: \neg Mythical \Rightarrow Mortal \wedge Mammal

Using **hypothetical syllogism**: Mortal \Rightarrow Mortal \wedge Mammal

Using **implication elimination**: $\neg\text{Mortal} \vee (\text{Mortal} \wedge \text{Mammal})$

Using **distributive law**: $(\neg\text{Mortal} \vee \text{Mortal}) \wedge (\neg\text{Mortal} \vee \text{Mammal})$

Eliminate the **tautology**: $\neg\text{Mortal} \vee \text{Mammal}$

S3: $\neg\text{Mortal} \vee \text{Mammal} \Rightarrow \text{Horned}$

Using **modus ponens** conclude: Horned

S4: Horned \Rightarrow Magical

Using **modus ponens** conclude: Magical

Note: Can't prove Mythical

York Solution:

S1: Mythical \Rightarrow Immortal
S2: \neg Mythical $\Rightarrow \neg$ Immortal \wedge Mammal
S3: Immortal \vee Mammal \Rightarrow Horned
S4 Horned \Rightarrow Magical

(1) Suppose Mythical is true. Then Immortal must be true.

(2) Suppose Mythical is false. Then Immortal must be false, and Mammal must be true.

Then the third implication comes into play. If the unicorn is immortal or a mammal then it is horned. In case (1), Immortal is true. In case (2), Mammal is true. In both cases, Horned must be true as well. Horned has been proved to be true in all models where all the sentences are true.

Then we use the final sentence, Horned \Rightarrow Magical, to prove that Magical is also true in all models where all sentences are true.

The result is that Horned and Magical can be proved to be true, but not Mythical.

4.4 Quiz: Principles of logic and propositional logic

Question 1 **KEYWORDS:** proposition symbols; models 1 pts

Suppose there are four proposition symbols A, B, C, D . How many models are there?

12
 16
 8
 2

A	B	C	D
TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	FALSE
TRUE	TRUE	FALSE	TRUE
TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE
TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

$$2^4 = 16$$

Question 2 **KEYWORDS:** proposition symbols; models satisfying criteria 1 pts

Using the same four proposition symbols A, B, C, D , how many models are there of the formula $A \vee \neg C$? Or, to put it another way, how many models satisfy $A \vee \neg C$?

8
 16
 12
 2

A	$\neg C$	$A \vee \neg C$	$A \wedge \neg C$
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE

Question 3 **KEYWORDS:** proposition symbols; models satisfying criteria 1 pts

Using the same four proposition symbols A, B, C, D , how many models are there of the formula $A \wedge \neg C$? Or, to put it another way, how many models satisfy $A \wedge \neg C$?

8
 4
 16
 12

A	$\neg A$	$A \vee \neg A$	$A \wedge \neg A$
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE

Question 4 **KEYWORDS:** proposition symbols; models satisfying criteria 1 pts

Using the same four proposition symbols A, B, C, D , how many models are there of the formula $A \vee \neg A$? Or, to put it another way, how many models satisfy $A \vee \neg A$?

16
 4
 8
 12

A	B	C	D	$\neg A$	$\neg B$	$\neg C$	$\neg D$
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

Question 5 **KEYWORDS:** proposition symbols; models satisfying criteria 1 pts

Using the same four proposition symbols A, B, C, D , how many models are there of the formula $A \wedge \neg A$? Or, to put it another way, how many models satisfy $A \wedge \neg A$?

2
 0
 4
 16

Question 6 **KEYWORDS:** proposition symbols; models satisfying criteria 1 pts

Using the same four proposition symbols A, B, C, D , how many literals are there?

3
 4
 8
 16

Question 7 **KEYWORDS:** converting proposition symbols to propositional logic; CNF 1 pts

Suppose we have the proposition symbols L, Q, D (meaning: "looks like a duck", "quacks like a duck", and "is a duck"). The sentence "If it looks like a duck, and quacks like a duck, then it is a duck" can be written in more than one way using the three proposition symbols. Select all the formulas that correctly encode the sentence.

$L \vee Q \Rightarrow D$
 $L \wedge Q \Rightarrow D$
 $\neg L \vee \neg Q \vee D$
 $L \vee Q \vee D$

$L \wedge Q \Rightarrow D$
\therefore
$\neg(L \wedge Q) \vee D$
Implication Elimination
\therefore
$\neg L \vee \neg Q \vee D$
De Morgan's Law

Question 8 **KEYWORDS:** entailment uses; entailment usage 1 pts

Entailment \models in propositional logic is about one sentence logically following from another. For two sentences S_1, S_2 , if $S_1 \models S_2$ then all the models that satisfy S_1 also satisfy S_2 . Which of the following uses of entailment are correct?

Select all the correct ones.

- $A \models B$
 $\models A$
 $A \wedge B \models B$
 $A \wedge \neg A \models B$
 $A \Leftrightarrow B \models A \vee B$
 $A \Leftrightarrow B \models \neg A \vee B$

Question 9 **KEYWORDS:** propositional logic; sentences; entailment; sound inference 1 pts

For two sentences in propositional logic S_1, S_2 , how is entailment related to inference performed by a sound inference algorithm?

- If $S_1 \models S_2$ then S_2 is inferred from S_1 (i.e. $S_1 \vdash S_2$)
 If $S_1 \models S_2$ then S_2 may be inferred from S_1 (i.e. $S_1 \vdash S_2$)
 If $S_2 \models S_1$ then S_2 is inferred from S_1 (i.e. $S_1 \vdash S_2$)
 If $S_2 \models S_1$ then S_2 may be inferred from S_1 (i.e. $S_1 \vdash S_2$)

The others have a model where the left-hand side is satisfied and the right-hand side is not.

For example, for answer A, any model containing $A = \text{false}$ satisfies the left-hand side and not the right-hand side.

Need help with this one

A **sound** inference algorithm is always correct, it is not always complete. Therefore S_2 *may* be inferred from S_1 but that is not guaranteed.

Propositional Theorem Proving

Proof methods divide into roughly two kinds:

- Application by inference rules
 - Legitimate (sound) generation of new sentences from old
 - Proof = a sequence of inference rule applications; can use inference rules as operators in a standard search algorithm.
 - Typically require transformation of sentences into conjunctive normal form.
- Model Checking
 - Truth table enumeration (always exponential in n)
 - improved backtracking (DPLL, see week 5)
 - heuristic search in model space (sound but incomplete)

Satisfiability

- A sentence is satisfiable **if it is true (it is satisfied by) some model.**
- The problem of determining the satisfiability in proposition logic was the first problem proven to be NP-complete.
- **p is valid iff $\neg p$ is unsatisfiable** (i.e. true in all models, if and only if, there is no model in which $\neg p$ is true.)
- This allows us to check for entailment using a **proof by contradiction** (see Resolution).

Logical Equivalence

- **Logical Equivalence** - Two sentences p and q are logically equivalent if they are true in the same set of models.
- Any two sentences are equivalent if and only if one of them entails the other:
- $p \equiv q \text{ iff } p \vDash q \text{ and } q \vDash p$

Validity

- A sentence is valid if it is true in all models.
- **Tautology** - A sentence that always holds true, a valid sentence.
 - e.g. $p \vee \neg p$ (the law of the excluded middle).
 - *Not to be confused with $p \wedge \neg p$ which is an empty set and always resolves to false.*

Deduction Theorem

- The deduction theorem states that for any sentences p and q , p entails q if and only if the implication $p \Rightarrow q$ is valid (**true in all models**).
- $p \vDash q \text{ iff } (p \Rightarrow q) \text{ is valid}$
- The deduction theorem stats that every valid implication sentence describes a legitimate inference.

{B, $\neg B \vee C$, $\neg B$ }

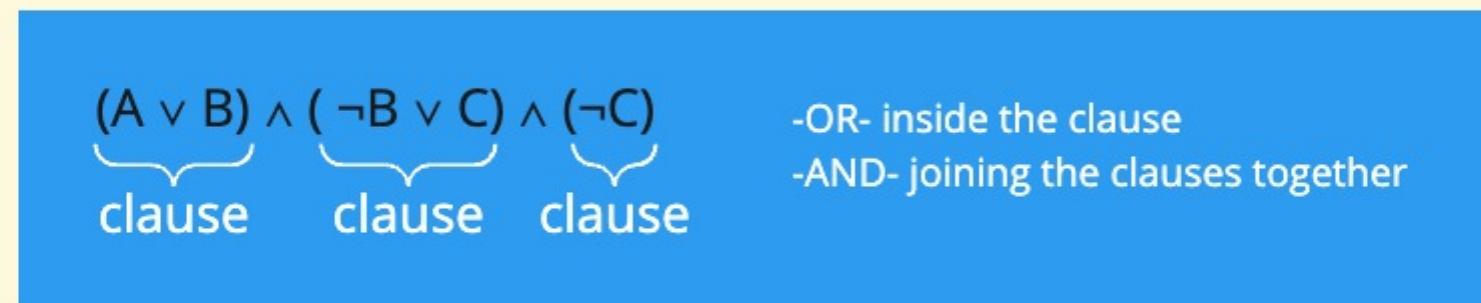
{C, $\neg C$ }
miro

CNF and Knowledge Base Algorithms

Conjunctive Normal Form (CNF):

Every sentence in propositional logic is logically equivalent to a conjunction of clauses (a clause is a disjunction of literals, i.e. a combination of statements joined by the -OR- operator). In this context a single atomic sentence is seen as a unit clause.

That is, we can express any sentence in propositional logic as a set of clauses joined together using an -AND- operator, as in the following example:



Algorithms:

CNF is important because it can be used as inputs into algorithms that drive knowledge based systems (systems that will work logically through its knowledge base to infer whether an event will happen or if something is true).

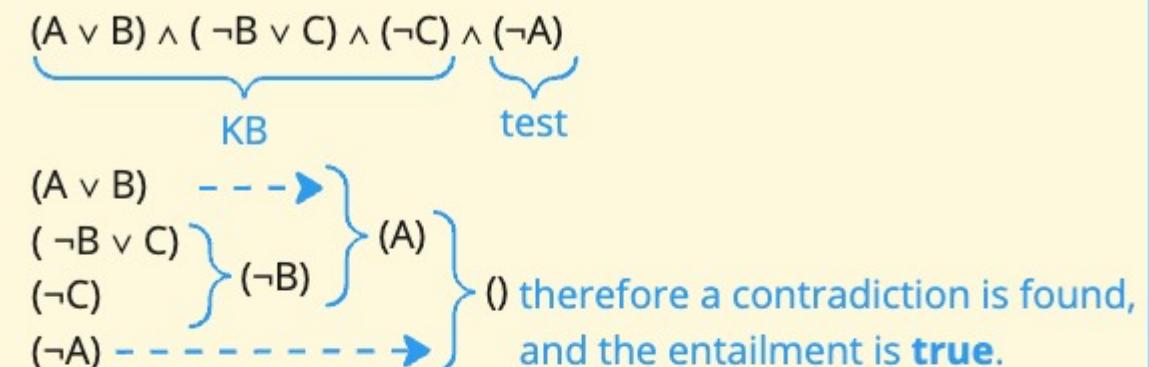
- **Resolution**
 - PL-RESOLUTION(KB, α)
- **Forward Chaining**
 - PL-FC-ENTAILS(KB, α)
- **Backwards Chaining**

PL-RESOLUTION algorithm

General:

- Propositional Logic Resolution = PL-RESOLUTION
- **PL-RESOLUTION(KB, α)** takes as inputs a knowledge base and a sentence, p , in propositional logic.
- The resolution algorithm works on the basis of check whether (KB and $\neg A$) is unsatisfiable, in which case $KB \models A$.
- This process was manually detailed in the **proof by contradiction**.

Proof by Contradiction: Does $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A)$ entail A ?



To determine if $KB \models A$:

The basics:

- Check if $(KB \wedge \neg A)$ is a contradiction?
 - Empty Clause is $A \wedge \neg A$
 - If so, then $KB \models A$
 - Otherwise, no entailment.

General pseudocode (from CS50 lecture 1):

- Convert $(KB \wedge \neg A)$ to Conjunctive Normal Form
- Keep checking to see if we can use resolution to produce a new clause.
 - If we ever produce the empty clause (equivalent to False), we have a contradiction, and $KB \models A$.
 - Otherwise, if we can't add new clauses, no entailment.

Algorithm pseudocode (from the text):

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
    inputs:  $KB$ , the knowledge base, a sentence in propositional logic
             $\alpha$ , the query, a sentence in propositional logic
    clauses  $\leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg \alpha$ 
    new  $\leftarrow \{ \}$ 
    while true do
        for each pair of clauses  $C_i, C_j$  in clauses do
            resolvents  $\leftarrow$  PL-RESOLVE( $C_i, C_j$ )
            if resolvents contains the empty clause then return true
            new  $\leftarrow$  new  $\cup$  resolvents
        if new  $\subseteq$  clauses then return false
        clauses  $\leftarrow$  clauses  $\cup$  new
```

Properties:

- The PL-RESOLUTION algorithm is **complete**.
 - Here completeness means refutation completeness, **resolution can refute or confirm the truth of any sentence with respect to the KB**.
- Space & Time Complexity: ???

Horn Clauses and Forward Chaining

Definite Clause

- A disjunction of literals of which exactly one is positive.
 - $(\neg p \vee q \vee \neg r)$

Why do we care?

- Real-world knowledge bases often contain only horn clauses.
- Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
 - e.g. $(\neg p \vee \neg q \vee r)$ can be written as $(p \wedge q) \Rightarrow r$
- Inference with horn clauses can be done through **forward chaining** and **backwards chaining**.
- Deciding entailment with horn clauses can be done in time that is linear in the size of the KB.

PL-FC-ENTAILS(KB, α)

- The **forward-chaining algorithm** determines if a single proposition symbol α (i.e. the query) is entailed by a knowledge base of definite clauses.
- Forward Chaining is an example of a general concept of data-driven reasoning:
 - We start from known facts.
 - If all of the premises of a given implication are known, we add the conclusion to the set of known facts.
 - This process continues until q is added or until no further inferences can be made.
 - The point where no further inferences are possible is called a **fixed point**.
- **Time complexity = $O(n)$** .
- Is **sound and complete for horn clause based KB**, i.e. it only derives sentences entailed by KB and every atomic sentence which is entailed by KB will be derived.
- Forward chaining can be neatly represented on a graph where multiple links joined by an arc indicate a conjunction (AND) while multiple links without an arc indicate a disjunction (OR).

Horn Clause

- A disjunction of literals in which at most one (i.e. **zero or one**) is positive (this implies a superset of definite clauses).
- e.g. $(\neg p \vee \neg q)$ is a horn clause but not a definite clause
- Horn clauses are **closed** under resolution: resolving two horn clauses gives a horn clause.

What we want to prove $P \Rightarrow Q$

$L \wedge M \Rightarrow P$

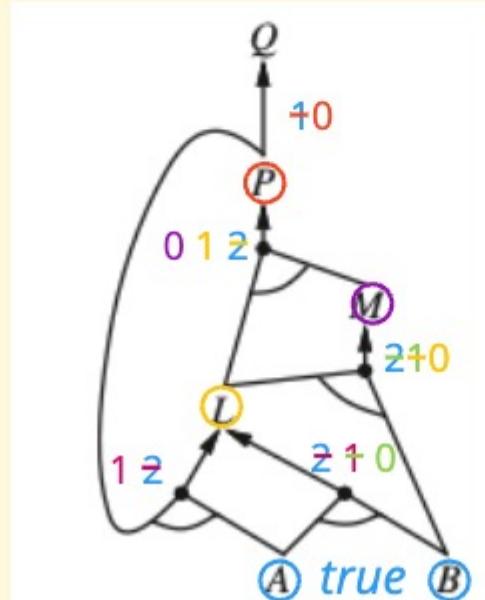
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



1. Count the literals in the premise.
2. Look at all the implications in which A is a premise, decrease the count of literals by 1.
3. Repeat for B
4. Because that clause went to 0 we have proved L is true. Repeat for L.
5. M is true. Repeat for M.
6. P is true, repeat for P.
7. Q is true.

Backward Chaining

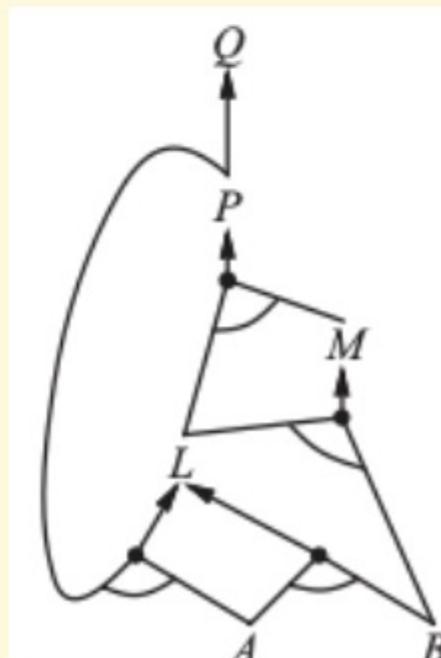
General:

- The second major family of logical inference algorithms uses backwards chaining over **definite clauses**.
- These algorithms work backwards from the query, q , chaining through rules to find known facts that support the proof.
 - If q is known to be true, we stop here.
 - Otherwise, we find those implications in the knowledge base whose conclusion is q . If all the premises of at least ONE of those implications can be proven to be true (by backward chaining) then q is true.
- Similar to FC but we work backwards from q .
- FC is data-driven reasoning, BC is **goal-directed reasoning**.
- BC is essentially the AND-OR-GRAH-SEARCH algorithm in the text.

Properties:

- Like FC, an efficient implementation of BC runs in **linear time; time complexity = $O(n)$** .
- **Space Complexity** of BC can be **much less** than linear in the size of the KB
Often the cost of BC is much less than linear in the size of the knowledge base because the process focuses only on relevant facts.

<https://www.youtube.com/watch?v=EZjs6w2YFRM>



TT-ENTAILS(KB, α)

General:

- Inference by enumeration.
- TT-ENTAILS is a truth-table enumeration algorithm for deciding propositional entailment (TT stands for Truth Table).
- PL-True? returns *true* if a sentence holds within the model.
- The variable model represents a partial model--an assignment to some of the symbols.
- The keyword "**and**" is used here as a logical operation on its two arguments, returning *true* or *false*.

Properties:

- For n symbols, **time complexity** is $O(2^n)$.
- **Space complexity** is $O(n)$.

```
function TT-ENTAILS?(KB,  $\alpha$ ) returns true or false
```

```
inputs: KB, the knowledge base, a sentence in propositional logic  
 $\alpha$ , the query, a sentence in propositional logic
```

```
symbols  $\leftarrow$  a list of the proposition symbols in KB and  $\alpha$ 
```

```
return TT-CHECK-ALL(KB,  $\alpha$ , symbols, {})
```

```
function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false
```

```
if EMPTY?(symbols) then
```

```
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)
```

```
    else return true // when KB is false, always return true
```

```
else do
```

```
    P  $\leftarrow$  FIRST(symbols)
```

```
    rest  $\leftarrow$  REST(symbols)
```

```
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = true})
```

```
        and
```

```
        TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = false}))
```

First Order Logic

Propositional Logic:

- Facts and the relationship between facts.
 - Facts are binary in nature, they either hold or do not hold (true or false).
- **Declarative** in nature; semantics are based on a truth between sentences and possible worlds.
- **Context Independent** - knowledge and inference are separate and inference is entirely domain independent.
- **Compositionality or Compositional Semantics** - the meaning of a sentence is a function of the meaning of its parts.
- Not concise.

First Order Logic

- Objects and the relationship between objects.
- Has the ability to concisely represent **more** information.
- A more expressive logic that uses the syntax of natural language
 - **Objects** - people, houses, numbers, theories, colors, etc.
 - **Relations** - these can be properties such as red, round, prime, etc.; or more general *n*-ary relations such as bigger than, inside, part of, owns, etc.
 - **Functions** - father of, best friend, third inning of, one more than, beginning of, etc.

First Order Logic Symbols

- **Constant Symbols** - Representations of objects.
- **Predicate Symbols** - Representations of properties that can resolve to true or false with regards to the constants.
- \forall = **for all**
- \exists = **there exists**

Optional Activity 2

Selections:

A: 2.4GHz CPU

B: 3.0GHz CPU

C: 16GB RAM

D: 32GB RAM

E: 500GB SSD

F: 1TB SSD

G: Discrete GPU

H: CPU/Memory run Hot

I: GPU/Disk run Hot

J: Slim Case

K: Large Case

Rules:

(A \vee B)

(C \vee D)

(E \vee F)

(J \vee K)

G \Rightarrow B

B \Rightarrow H

A \wedge D \Rightarrow H

G \Rightarrow I

F \Rightarrow I

H \wedge I \Rightarrow K

D \wedge F \Rightarrow K

Horn Clauses (max 1 positive):

(\neg A \vee \neg B)

(\neg C \vee \neg D)

(\neg E \vee \neg F)

(\neg J \vee \neg K)

(\neg G \vee B)

(\neg B \vee H)

(\neg (A \wedge D) \vee H) \equiv (\neg A \vee \neg D \vee H)

(\neg G \vee I)

(\neg F \vee I)

(\neg (H \wedge I) \vee K) \equiv (\neg H \vee \neg I \vee K)

(\neg (D \wedge F) \vee K) \equiv (\neg D \vee \neg F \vee K)

```
// The customer may choose at most one CPU, memory, disk, case
fc.addClause(new int[]{-A, -B});
fc.addClause(new int[]{-C, -D});
fc.addClause(new int[]{-E, -F});
fc.addClause(new int[]{-J, -K});
fc.addClause(new int[]{-G, B});
fc.addClause(new int[]{-B, H});
fc.addClause(new int[]{-A, -D, H});
fc.addClause(new int[]{-G, I});
fc.addClause(new int[]{-F, I});
fc.addClause(new int[]{-H, -I, K});
fc.addClause(new int[]{-D, -F, K});
```

```
// New instance of Forward Chaining
ForwardChainingConfiguration fc=new ForwardChainingConfiguration();

// No symbols set to true:
setupConfig(fc);
System.out.println(fc.forwardChaining(11));

// Test scenario 1:
fc.resetClauses();
setupConfig(fc);

// Add in the choices that the customer has already made
fc.addClause(new int[]{B});
fc.addClause(new int[]{J});

// To test if an additional choice is an option, set it to true and
// if there is a model.
fc.addClause(new int[]{G});

System.out.println(fc.forwardChaining(11));
```

Model:

Variable 1 = false
Variable 2 = false
Variable 3 = false
Variable 4 = false
Variable 5 = false
Variable 6 = false
Variable 7 = false
Variable 8 = false
Variable 9 = false
Variable 10 = false
Variable 11 = false
true

Inferred 2 with clause [2]
Inferred 10 with clause [10]
Inferred 7 with clause [7]
Inferred 8 with clause [-2, 8]
Inferred 9 with clause [-7, 9]
Inferred 11 with clause [-8, -9, 11]
No models satisfy all clauses simultaneously. False goal clause: [-10, -11]

false
Inferred 1 with clause [1]
Inferred 4 with clause [4]
Inferred 10 with clause [10]
Inferred 6 with clause [6]
Inferred 8 with clause [-1, -4, 8]
Inferred 9 with clause [-6, 9]
Inferred 11 with clause [-8, -9, 11]
No models satisfy all clauses simultaneously. False goal clause: [-10, -11]

Week 5: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: The satisfiability problem in propositional logic

- Section 7.3: "Logic" from AI:AMA (**4th ed**)
- Section 7.4: "Propositional Logic" from AI:AMA (**4th ed**)
- Section 7.5: "Propositional Theorem Proving" from AI:AMA (**4th ed**)

Lesson 2: A complete, backtracking algorithm for SAT

- Section 7.6: "Effective propositional model checking" from AI:AMA (**4th ed**)
- Section 7.6.1: "A complete backtracking algorithm" from AI:AMA (**4th ed**)
- Section 7.6.2: "Local search algorithms", from AI:AMA (**4th ed**)

Lesson 3: An incomplete, randomised algorithm for SAT

- Section 7.6.2: "Local search algorithms", from AI:AMA (**4th ed**)

Lesson 4: Scalability of SAT

- [Solving and verifying the boolean pythagorean triples problem](#)

Week 5 Further Reading:

- Chapter 34: "NP Completeness", from Introduction to Algorithms (3rd ed)
- Section 7.6.3: "The landscape of random SAT problems", from AI:AMA (**4th ed**)

Study Session:

Thursday, 12 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

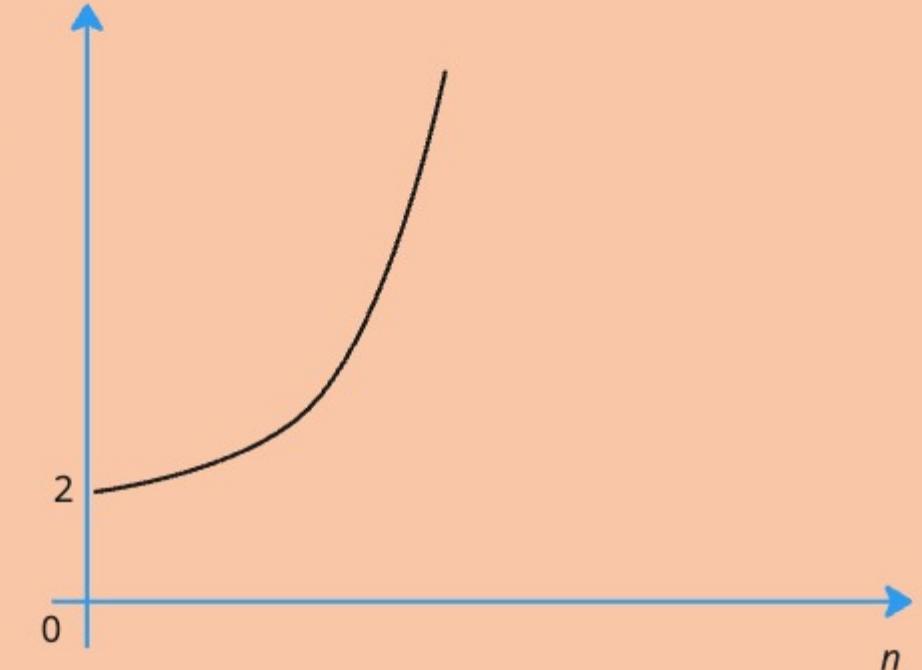
Satisfiability - General

Satisfiability - A formula / sentence in propositional logic is true or false with respect to a model.

Satisfiability (SAT) problem - To find a model which satisfies a formula, or to prove that there is no such model.

- **A formula is satisfiable if at least one model satisfies the formula.** Since all propositional logic sentences can be transformed into Conjunctive Normal Form (CNF), this means that a formula is satisfiable if all of the clauses in CNF are satisfied simultaneously.
- No need for horn clauses.
- If there is more than one such model then an algorithm for SAT can return any one of them to prove satisfiability.

Time Complexity: SAT belongs to a class of problems called **NP-complete**. If there are n propositional symbols, then a **complete algorithm for SAT takes $O(2^n)$ time**.



Validity vs. Satisfiability:

Validity and satisfiability are connected:

- A is **valid** iff $\neg A$ is **unsatisfiable**
- Contra-positively, A is **satisfiable** iff $\neg A$ is **not valid**
- $A \models B$ if and only if the sentence $(A \wedge \neg B)$ is **unsatisfiable**.

Motivation:

- Proving entailment: Recall that proving $(KB \models a)$ is equivalent to proving the unsatisfiability of $(KB \wedge \neg a)$
- Practical applications:
 - product configuration
 - hardware verification
 - software verification
 - planning
 - scheduling
 - proofs in mathematics

What is NP-completeness? (From the Optional Reading)

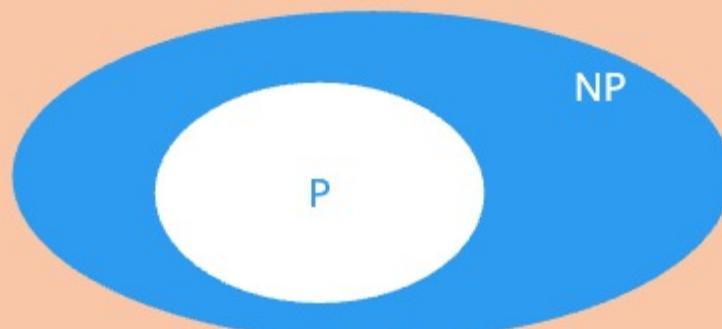
Classes of problems:

- **P** - A complexity class that represents the set of all decision problems that can be solved in polynomial time
- **NP** - A complexity class that represents the set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified (i.e. we were given a "certificate") in polynomial time. **NP stands for nondeterministic polynomial time.**
- **NPC (NP-Complete)** - A complexity class of problems whose status is unknown: no polynomial-time algorithm has yet been discovered for an NP-complete problem, but neither has anyone been able to prove that no polynomial-time algorithm can exist for any of them.

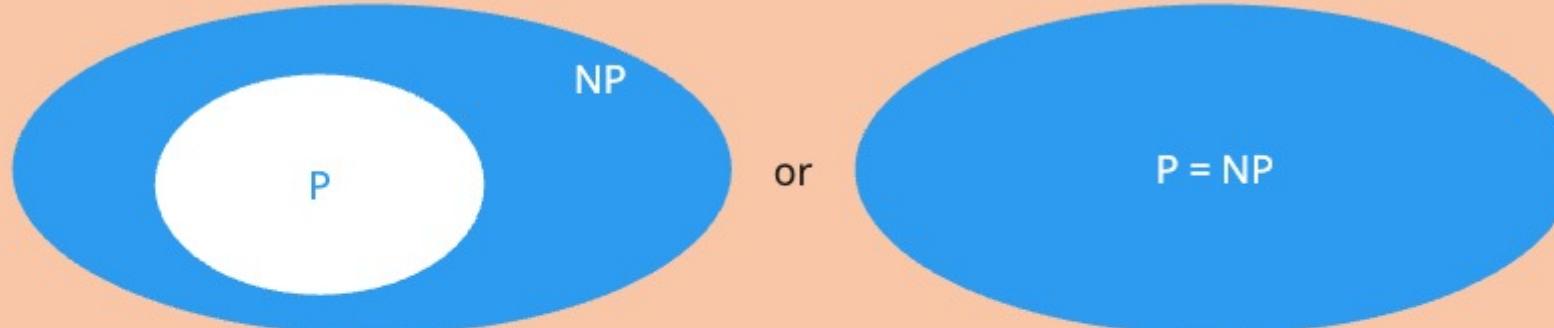
If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time algorithm.

In order to show that a problem is NP-complete we usually try to show that it reduces to a different problem, which is known to be NP-complete. This, in turn, requires us to find a "first" NP-complete problem.

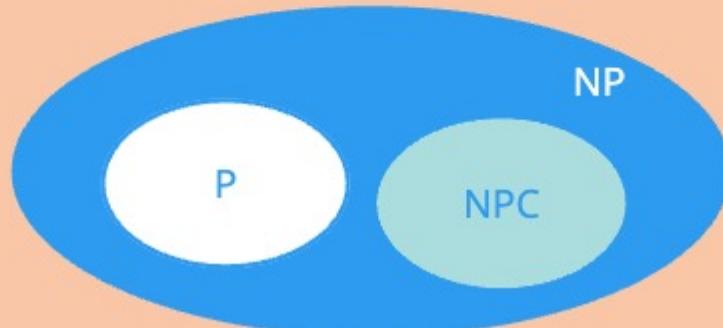
What we know (P is a subset of NP since we can solve the problems in P time without the need for a "certificate"):



What we want to know: if P is a proper subset of NP?



According to CLRS, p. 1070, Fig. 34.6, most theoretical computer scientists view the relationships among P, NP, and NPC as follows, with there being no intersection between P and NPC:



SAT Algorithms

SAT for decision problems:

- A SAT solver is an algorithm for establishing satisfiability.
- SAT takes the boolean logic formula as an input and returns if it finds a combination of variables that can satisfy it (**SAT**), or if it can demonstrate that no such combination exists (**UNSAT**).
- SAT algorithms have been researched for decades and are now very efficient (even though their time complexity is exponential)).
- Note: In the sudoku example in canvas the representation of at-most-one is quite simple with **space complexity** of $O(n^2)$ for n propositional symbols.

3 steps to solving decision problems using SAT:

1. Represent the problem as a propositional logic formula.
2. Solve the problem using a SAT algorithm.
3. If the SAT algorithm has found a model, translate the model into a solution to the original decision problem.

Restrictions:

- In many SAT problems, you must encode the constraint that exactly one of n propositional variables is true. This is usually broken down into two constraints: (1) "**at least one**" variable is true and (2) "**at most one**" variable is true.
- For example: If we have 2 propositions, A and B:
 - In order to impose the restriction that **at least one (ALO)** of them is true we would impose $(A \vee B)$. *This requires just one restriction.*
 - In order to impose the restriction that **at most one (AMO)** of them is true, we would impose $(\neg A \vee \neg B)$.

Exercise A: Given $X = \{ A, B, C, D, E \}$, what is the CNF clause for the **at least one** restriction?

$$(A \vee B \vee C \vee D \vee E)$$

Exercise B: Given $X = \{ A, B, C, D, E \}$, what is the CNF for **at most one** restriction?

Number of Clauses for AMO

$$\left\{ \frac{n * (n-1)}{2} = \frac{5 * 4}{2} = 10 \text{ clauses} \right.$$

CNF

$$\left\{ (\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg A \vee \neg D) \wedge (\neg A \vee \neg E) \wedge (\neg B \vee \neg C) \wedge \right. \\ \left. (\neg B \vee \neg D) \wedge (\neg B \vee \neg E) \wedge (\neg C \vee \neg D) \wedge (\neg C \vee \neg E) \wedge (\neg D \vee \neg E)$$

not CNF

$$(A \wedge B \wedge C \wedge D \wedge E) \vee (A \wedge B \wedge C \wedge D \wedge \neg E) \vee \dots \vee (A \wedge B \wedge C \wedge \neg D \wedge \neg E)$$
$$\left. \left(\neg A \vee \neg B \right) \vee \left(\neg A \vee \neg C \right) \vee \left(\neg A \vee \neg D \right) \vee \left(\neg A \vee \neg E \right) \vee \dots \right\} \frac{(n-1)n}{2} = \frac{n^2 - n}{2} = O(n^2)$$
$$\dots \vee (\neg B \vee \neg C) \vee (\neg B \vee \neg D) \vee (\neg B \vee \neg E) + \dots$$

$A \rightarrow 4$
 $B \rightarrow 3$
 $C \rightarrow 2$
 $D \rightarrow 1$

$$1+2+3+4 = 10$$
$$1+2+3+2 \cdot (n-1) = 1+2+3+2 \cdot 4 = 15$$

From previous year's Miro board
I'm not sure I understand what they're doing here or if it's important

5.4 Quiz: The satisfiability problem in propositional logic

Question 1 KEYWORDS: propositional satisfiability; model 1 pts

In the context of propositional satisfiability, what is a model?

- An assignment of every proposition symbol to true or false
- A 3D representation of something, usually of a smaller size than the real thing.
- A representation of a real-world scenario as a propositional logic formula.
- A logic sentence that is entailed by a formula

If every propositional symbol is given a value, then any sentence of propositional logic can be evaluated to true or false.

Question 3 KEYWORDS: unsatisfiable; models 1 pts

If a formula F is unsatisfiable, how many models satisfy F ?

- 1
- 0
- 2^n where n is the number of proposition symbols.
- At least 1

As a follow up to Q2: therefore a formula is unsatisfiable if zero models satisfy the formula.

Question 5 KEYWORDS: proposition symbols; at-most-one; at-least-one; CNF clauses 1 pts

Suppose you have four proposition symbols and you want exactly one of them to be true. Using the representations of at-most-one and at-least-one given in this lesson, how many CNF clauses will there be?

- 12
- 8
- 7
- 16

- $n = 4$ therefore "at most one" includes $4(3)/2 = 6$ clauses,
- "at least one" always only includes 1 clause
- Therefore there will be a total of $6+1=7$

Question 7 KEYWORDS: propositional logic; SAT algorithm; SAT usage 1 pts

Which of the following problem scenarios could sensibly be represented in propositional logic and tackled using a SAT algorithm?

Select all that apply.

- Finding a sequence of actions to assemble an object from parts.
- Playing chess.
- Recognising pedestrians from a 3D point cloud produced by a sensor in a self-driving car.
- Finding a way through a maze where the layout of the maze is known in advance.
- Deciding how to assign tasks to workers and machines in a factory.

Question 2 KEYWORDS: satisfiable; models 1 pts

If a formula F is satisfiable, how many models satisfy F ?

- 1
- 0
- 2^n where n is the number of proposition symbols
- At least 1

A formula is satisfiable if at least one model satisfies the formula.

Question 4 KEYWORDS: satisfiability; entailment; sentence 1 pts

How is satisfiability related to entailment for a formula F and a sentence α ?

- $F \models \alpha$ is equivalent to satisfiability of $F \wedge \alpha$
- $F \models \alpha$ is equivalent to unsatisfiability of $F \wedge \alpha$
- $F \models \alpha$ is equivalent to satisfiability of $F \wedge \neg\alpha$
- $F \models \alpha$ is equivalent to unsatisfiability of $F \wedge \neg\alpha$

By definition: $A \models B$ if and only if the sentence $(A \wedge \neg B)$ is unsatisfiable.

Question 6 KEYWORDS: SAT computational complexity; n 1 pts

What is the computational complexity of solving SAT for a formula with n proposition symbols?

- Linear time - $O(n)$
- Quadratic time - $O(n^2)$
- Exponential time - $O(2^n)$
- Doubly exponential time - $O(2^{2^n})$

The complexity of solving SAT is exponentially related to the total number of models.

(1) is probably a sensible choice unless there is a lot of arithmetic involved.

(4) could be solved with propositional logic although A* might be better

(5) is the most obvious choice because the decision to assign a particular task to a machine can be represented with one propositional symbol.

A good rule of thumb for these types of questions is to remember is that SAT algorithms are used very often in manufacturing environments.

DPLL - A backtracking algorithm for SAT

Definitions:

- **DPLL** - A **backtracking based algorithm for solving SAT problems** named after the initial authors: Davis, Putnam, Logemann, and Loveland
- **sentence (*s*)** - A sentence in CNF that is provided as input for DPLL to solve. Is this, or is this not, **satisfiable**?
 - In some examples outside the book they use the term *formula* for *sentence*, but I'm going to stick to the book's nomenclature.
 - A given *sentence* resolves to TRUE if **all** clauses in the *sentence* are true.
 - A given *sentence* resolves to FALSE if there is **any** in the *sentence* that is false.
 - Otherwise the *sentence* is unassigned and the algorithm continues.
- **literal (*l*)** - A propositional symbol or its opposite; e.g. P, $\neg P$. (reminder from week 4)
- **clauses (*c*)** - The individual clauses from *s* contained in a set; e.g. $\{(A \vee B \vee \neg C \vee D), (\neg A \vee B \vee \neg C), (\neg B \vee \neg C), (E), (E \vee C)\}$
- **propositional symbols (*p*)** - A list of propositional symbols in *s*; e.g. $\{A, B, C, D, E\}$
- **models (*m*)** - a set of models with **partial assignments**
 - In the partial models, a given *literal* can be set to a value of TRUE ($m(l) = 1$), FALSE ($m(l) = 0$) or UNASSIGNED.
 - In the partial models, a given *clause* resolves to true if **any** *literal* contained in that *clause* is TRUE (for *m*, *c* is true if for any $l \in c$, *l* is true).
 - In the partial models, a given *clause* resolves to false if **all** *literals* contained in that *clause* are FALSE (for *m*, *c* is false if for each $l \in c$, *l* is false).
 - Otherwise the *clause* is UNASSIGNED.

General:

- DPLL constructs a model that satisfies the formula by building it step-by-step, assigning one *symbol* at a time to TRUE or FALSE.
- This is essentially a recursive, depth-first enumeration of all possible models, analogous to backtracking search.
- The algorithm uses the **tree-search** version of **Depth-First Search (DFS)** and each node of the tree has at most two children (corresponding to TRUE/FALSE assignments).
- The frontier is **not** kept in memory, instead every recursive call to the DPLL function corresponds to a node in the search tree.
- When the algorithm can't extend the partial model it reverses earlier decisions and tries a different path.
- Three main actions: **Decision, Unit propagation, backtracking**.

Heuristics:

The DPLL algorithm uses a number of heuristics to improve efficiency:

- **Early Termination** - The algorithm detects whether a *sentence* must be TRUE or FALSE even based on a partial assignment, without examining entire subtrees of the search space.
 - e.g.: $(B \vee S) \wedge (A \vee C)$ **will evaluate to TRUE if A is TRUE**, irrespective of the values of B or C.
- **Pure Symbol Heuristic**
 - A pure symbol is a symbol that always appears with the same "sign" in all *clauses*.
 - e.g.: C in $(A \vee C) \wedge (B \vee C)$
 - e.g.: $\neg C$ in $(A \vee \neg C) \wedge (B \vee \neg C)$
 - If a *sentence* has a model, then it (also) has a model with the pure symbol assigned so that the literal is True, since this cannot turn sentences False, it can only turn them to True or leave their truth value unchanged.
 - **(!) A symbol can become pure even if it initially is not. This happens if, after assigning values to (some) other symbols, the only remaining clauses, which are not already true, contain the symbol in pure form.**
 - for example: $(A \vee B \vee C) \wedge (\neg A \vee C) \wedge (C \vee D)$, if we assign A=False, then $(\neg A \vee C)$ is already True and C is a pure symbol in the remaining clauses.
- **Unit Clause Heuristic**
 - **unit clause:**
 - a *clause* with just one literal; e.g.: (A) , or, $(\neg A)$.
 - In DPLL a given *clause* is ALSO a **unit clause** if there is exactly one *literal* **unassigned** and the rest have already been assigned FALSE by the model; in this case, the unassigned literal is called a **unit literal**.
 - for example: $(\neg A \vee \neg B \vee \neg C \vee D)$, if we assigned A=True, B=True, C=True, then this becomes a unit clause and D becomes a unit literal.
 - The truth value of the unit clause must be assigned so as to make the literal true.
 - **Unit propagation:** Assigning one unit clause can create another unit clause, and so on, in a cascade. This resembles forward chaining.

Properties:

- **Complete**: Guaranteed to find a solution if one exists; this follows from the fact that the algorithm eventually enumerates all possible models.
- **Time complexity**: $O(2^n)$ where n is the number of proposition symbols. Worst case it has to explore all possible variable assignments (equivalent to traversing a binary tree of depth n). However it often performs much better in practice.
- **Space Complexity** is $O(n)$ as it needs to store the current partial assignment in memory, which can have n variables.

Notes about the Pseudocode in the text:

- The simple implementation of DPLL in the text uses an **arbitrary variable ordering** and **always tries the value of TRUE before FALSE**.
- The pure symbol rule goes **before** the unit clause rule.

```
function DPLL-SATISFIABLE?(s) returns true or false
    inputs: s, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(clauses, symbols, {})

function DPLL(clauses, symbols, model):
    if every clause in clauses is true:
        return true
    if some clause in clauses is false:
        return false
    P, value ← FIND-PURE(symbols)
    if P is non-null then
        P, value ← FIND-UNIT(clauses, P)
    if P is non-null then
        P ← FIRST(symbols); i ← 1
        while P[i] is unassigned:
            P[i] ← value
            i ← i + 1
        return DPLL(clauses, symbols, extend(model, P))
    else:
        return DPLL(clauses, symbols, extend(model, P))

def dpll(clauses, symbols, model):
    """See if the clauses are true in a partial model"""
    unknown_clauses = [] # clauses with an unknown symbol
    for c in clauses:
        val = pl_true(c, model)
        if val is False:
            return False
        if val is not True:
            unknown_clauses.append(c)
    if not unknown_clauses:
        return model
    P, value = find_pure_symbol(symbols, unknown_clauses)
    if P:
        return dpll(clauses, remove_all(P, symbols), extend(model, P))
    P, value = find_unit_clause(clauses, model)
    if P:
        return dpll(clauses, remove_all(P, symbols), extend(model, P))
    raise TypeError("Argument should be of the type symbol")
    P, symbols = symbols[0], symbols[1:]
    return (dpll(clauses, symbols, extend(model, P)),
            dpll(clauses, symbols, extend(model, P)))
```

Figure 7.17 The DPLL logic. The ideas behind the text; each returns a *s* such that $TT\text{-ENTAILS?}(s, TT\text{-ENTAILS?}(DPLL(s)))$

DPLL Example (Sample Paper 2)

Question 11:

Consider the following five sentences in propositional logic:

1. $(A \vee B)$
2. $(A \vee \neg B \vee C)$
3. $(\neg A \vee B)$
4. $(\neg A \vee \neg B \vee C)$
5. $(\neg B \vee \neg C)$

Apply the DPLL algorithm with the following assumptions:

- Proposition symbols are assigned in the order A-B-C
- When assigning symbols, true is assigned first.
- When performing unit propagation, the clauses are always processed in the order given.

For each step of DPLL, you will need to give the assignment made by DPLL followed by the consequences of that assignment, if there are any. Write your answer in the same format as this example.

Step 1: B=true, C=false, A=true, fail

Use the word **fail** to indicate that DPLL discovered a clause that is false in the current model, and **success** to indicate that all clauses are satisfied.

Step 1:

- Initial settings: $m=\{\}$, $p = \{A, B, C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? No.
- Assignment: **A = True (given in problem)**
- Consequence:
 - $\checkmark c1. (\text{True} \vee B) = \text{true}$
 - $\checkmark c2. (\text{True} \vee \neg B \vee C) = \text{true}$
 - $c3. (\neg A \vee B)$
 - $c4. (\neg A \vee \neg B \vee C)$
 - $c5. (\neg B \vee \neg C)$
- $m=\{A=True\}$, $p = \{B, C\}$
- $c = \{B, \neg B \vee C, \neg B \vee \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? **Yes, B is a unit clause.**
- Assignment: **Therefore B = True.**
- Consequence:
 - $\checkmark c3. (\text{True}) = \text{true}$
 - $c4. (\neg B \vee C)$
 - $c5. (\neg B \vee \neg C)$
- $m=\{A=True, B=True\}$ $p = \{C\}$ $c = \{C, \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? **Yes, C is a unit clause.**
- Assignment: **Therefore C = True**
- Consequence: **Fail**

Step 1: A=True, B=True, C= True, fail

Step 2:

- Backtrack to last given/guess: A
- Assignment: **A = False**
- Consequence:
 - $c1. (\neg A \vee B)$
 - $c2. (\neg A \vee \neg B \vee C)$
 - $\checkmark c3. (\text{True} \vee B) = \text{true}$
 - $\checkmark c4. (\text{True} \vee \neg B \vee C) = \text{true}$
 - $c5. (\neg B \vee \neg C)$
- $m=\{A=False\}$,
- $p = \{B, C\}$
- $c = \{B, \neg B \vee C, \neg B \vee \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? **Yes, B is a unit clause.**
- Assignment: **Therefore B = True.**
- Consequence:
 - $\checkmark c1. (\text{True}) = \text{true}$
 - $c2. (\neg B \vee C)$
 - $c5. (\neg B \vee \neg C)$
- $m=\{A=False, B=True\}$ $p = \{C\}$ $c = \{C, \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? **Yes, C is a unit clause.**
- Assignment: **Therefore C = True**
- Consequence: **Fail**

Step 2: A=False, B=True, C= True, fail

DPLL Walk Through

Trace the DPLL algorithm on the resulting set of clauses. Point out pure symbols and unit clauses at each iteration.

$$s = (\neg A \vee B) \wedge (\neg B \vee \neg C) \wedge (A \vee B) \wedge (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D) \wedge (C \vee D)$$

c1 c2 c3 c4 c5 c6 c7 c8

Step 0: Initial settings: $m=\{\}$, $p = \{A, B, C, D\}$

Step 1: Can you terminate early? No.

Step 2: Any pure symbols? Yes. D is pure.

- **Assignment:** $D=True$,
- **Consequence:** $c5, c7$ and $c8$ all become true

Step 3: Any unit clauses? No.

Step 4: Next settings: $m=\{D=True\}$, $p = \{A, B, C\}$

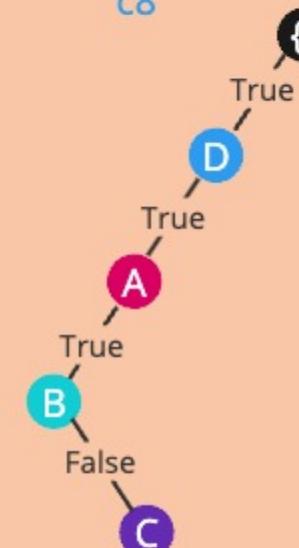
Step 5: Can you terminate early? No.

Step 6: Any pure symbols? No.

Step 7: Any unit clauses? No.

Step 8: Pick a literal and assignment at random.

- **Assignment:** $A=True$
- **Consequence:** $c3$ and $c4$ become true.
remove $\neg A$ where it occurs because it is false.



Step 0 Clauses:

1. $(\neg A \vee B)$
2. $(\neg B \vee \neg C)$
3. $(A \vee B)$
4. $(A \vee C)$
5. $(A \vee D)$
6. $(B \vee C)$
7. $(B \vee D)$
8. $(C \vee D)$

Step 5 Clauses:

1. $(\neg A \vee B)$
2. $(\neg B \vee \neg C)$
3. $(A \vee B)$
4. $(A \vee C)$
5. $(A \vee \text{True})=\text{true}$
6. $(B \vee C)$
7. $(B \vee \text{True})=\text{true}$
8. $(C \vee \text{True})=\text{true}$

Step 9 Clauses:

1. $(\neg A \vee B)$
2. $(\neg B \vee \neg C)$
3. $(\text{True} \vee B)=\text{true}$
4. $(\text{True} \vee C)=\text{true}$
5. $(A \vee \text{True})=\text{true}$
6. $(B \vee C)$
7. $(B \vee \text{True})=\text{true}$
8. $(C \vee \text{True})=\text{true}$

Step 13 & 18 Clauses:

1. $(\neg A \vee \text{True})=\text{true}$
2. $(\neg B \vee \neg C)=\text{true}$
3. $(\text{True} \vee B)=\text{true}$
4. $(\text{True} \vee C)=\text{true}$
5. $(A \vee \text{True})=\text{true}$
6. $(\text{True} \vee C)=\text{true}$
7. $(B \vee \text{True})=\text{true}$
8. $(C \vee \text{True})=\text{true}$

Step 9: Next settings: $m=\{A=True, D=True\}$, $p = \{B, C\}$

Step 10: Can you terminate early? No.

Step 11: No pure symbols.

Step 12: $c1$ is a unit clause

- **Assignment:** $B=True$
- **Consequence:** $c1$ and $c6$ become true. Remove $\neg B$ where it occurs because it is false.

Step 13: Next settings: $m=\{A=True, B=True, D=True\}$, $p = \{C\}$

Step 14: Can you terminate early? No.

Step 15: No pure symbols.

Step 16: $c2$ is a unit clause

- **Assignment:** $C=False$
- **Consequence:** $c2$ becomes true.

Step 17: Next settings: $m=\{A=True, B=True, C=False, D=True\}$

Step 18: return true for a satisfying model.

Lecture

Assumption	Deduction	Clauses Satisfied	Satisfiability
X=true	None	None	Unknown
X=true & Y=true	From clause 3 – deduced that Z= true	1,2,3	Fail so backtrack one <u>assumption</u> and instead of Y=true Set Y=false
X=true & Y=false	From clause 2 deduced that Z= false	1,2,3,4	Success

$X \vee \neg Z$
 $Y \vee \neg Z$
 $\neg X \vee \neg Y \vee Z$
 $\neg Y \vee \neg Z$

Q11

Assumption	Deduction	Clauses Satisfied	Satisfiability
A=True	From clause 3 – deduced that B=true From clause 4 - Deduce that C= true	1,2,3,4	Fail in clause 5 so backtrack one Assumption and instead of A=true Set A=false
A=False	From clause 1 – deduced that B=true From clause 2 - Deduce that C= true	1,2, 3,4	Fail in clause 5

1. $A \vee B$
2. $A \vee \neg B \vee C$
3. $\neg A \vee B$
4. $\neg A \vee \neg B \vee C$
5. $\neg B \vee \neg C$

WalkSAT - An incomplete, randomised algorithm for SAT

General Idea:

- We start with a random model: the truth value of each symbol is assigned randomly.
- We make small iterative changes (one symbol at a time):
 - Pick a clause at random from clauses which are not satisfied by the current model
 - With probability p , select a symbol from the chosen clause at random and flip its value.
 - With probability $(1-p)$ select a symbol from the chosen clause, on the basis of which symbol will maximize the number of satisfied clauses (this can be seen as a "greedy" aspect of WalkSAT).

Properties:

- The input is in CNF.
- The algorithm can revisit the same model multiple times (we can have loops)
- WalkSAT has no memory - it does not remember which model it has seen in the past and only stores the current model
- It is **incomplete** - Not guaranteed to find a solution if one exists.
- It is useful for finding a satisfiable solution but cannot be used to show unsatisfiability - if given an unsatisfiable formula it will run forever or until a pre-specified limit is reached (this can be imposed using the max_flips parameter in the pseudocode).
- Notice that if we set max_flips to infinity and $p>0$ then the algorithm will run forever when given an unsatisfiable formula.
- It is an example of a local search algorithm.
- Complementary to DPLL: some SAT instances which are difficult for DPLL are easy for WalkSAT and vice versa.

Note: Local search methods, of which WalkSAT is an instance, search in the space of full models (i.e. all symbols have a truth value assigned) as opposed to DPLL which searches in the space of partial models.

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic
        p, the probability of choosing to do a "random walk" move, typically around 0.5
        max_flips, number of value flips allowed before giving up

model  $\leftarrow$  a random assignment of true/false to the symbols in clauses
for each i = 1 to max_flips do
    if model satisfies clauses then return model
    clause  $\leftarrow$  a randomly selected clause from clauses that is false in model
    if RANDOM(0, 1)  $\leq p$  then
        flip the value in model of a randomly selected symbol from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure
```

Figure 7.18 The WALKSAT algorithm for checking satisfiability by randomly flipping the values of variables. Many versions of the algorithm exist.

WalkSAT Exercise

Consider the following SAT instance in CNF:

1. $A \vee B \vee \neg C \vee D$
2. $\neg A \vee B \vee \neg C$
3. $\neg B \vee \neg C$
4. E
5. $E \vee C$

Initial Model (randomly assigned): $A = \text{True}$, $B = \text{False}$, $C = \text{False}$, $D = \text{False}$, $E = \text{False}$

Solve it using WalkSAT with $p = 1/2$

Model: $A = \text{True}$, $B = \text{False}$, $C = \text{False}$, $D = \text{False}$, $E = \text{False}$

- Evaluate clauses:
 - Satisfied Clauses: 1, 2, 3
 - Unsatisfied Clauses: 4, 5
- Pick an unsatisfied clause at random: 5. $E \vee C$
- Random or Greedy ($p=0.5$)? Random, C
- Flip selected symbol: $C = \text{True}$

Model: $A = \text{True}$, $B = \text{False}$, $C = \text{True}$, $D = \text{False}$, $E = \text{False}$

- Evaluate clauses:
 - Satisfied Clauses: 1, 3, 5
 - Unsatisfied Clauses: 2, 4
- Pick an unsatisfied clause at random: 2. $\neg A \vee B \vee \neg C$
- Random or Greedy ($p=0.5$)? Greedy
 - If we flip $A = \text{False}$, then 2, 3, & 5 are satisfied
 - If we flip $B = \text{True}$, then 1, 2, & 5 are satisfied
 - If we flip $C = \text{False}$, then 1, 2, & 3 are satisfied
- Flip selected symbol
 - $B = \text{True}$

How do we choose between the Greedy options here?

Model: $A = \text{True}$, $B = \text{True}$, $C = \text{True}$, $D = \text{False}$, $E = \text{False}$

- Evaluate clauses:
 - Satisfied Clauses: 1, 2, 5
 - Unsatisfied Clauses: 3, 4
- Pick an unsatisfied clause at random: 4. E
- Random or Greedy ($p=0.5$)? Random, E
- Flip selected symbol: $E = \text{True}$

Model: $A = \text{True}$, $B = \text{True}$, $C = \text{True}$, $D = \text{False}$, $E = \text{True}$

- Evaluate clauses:
 - Satisfied Clauses: 1, 2, 4, 5
 - Unsatisfied Clause: 3
- Only one unsatisfied clause to choose: 3. $\neg B \vee \neg C$
- Random or Greedy ($p=0.5$)? Greedy.
 - If we flip $B = \text{False}$, then 1, 3, 4, & 5 are satisfied
 - If we flip $C = \text{False}$, then 1, 2, 3, 4, 5 are satisfied
- Solution found

Activity 1

Given $s = (X \vee Y \vee Z) \wedge (X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee Y \vee Z) \wedge (\neg X \vee \neg Y \vee \neg Z) \wedge (\neg Y \vee Z) \wedge (Y \vee \neg Z)$

(1) Solve the following CNF formula using DPLL, use symbol order X-Y-Z:

Step 1:

- Initial settings: $m=\{\}$, $p = \{X, Y, Z\}$

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? No.

- Assignment:** $X = \text{True}$ (Decision Point)

- Outcome:**

- ✓ c1. $(\text{True} \vee Y \vee Z) = \text{true}$

- ✓ c2. $(\text{True} \vee \neg Y \vee \neg Z) = \text{true}$

- c3. $(\neg X \vee Y \vee Z)$

- c4. $(\neg X \vee \neg Y \vee \neg Z)$

- c5. $(\neg Y \vee Z)$

- c6. $(Y \vee \neg Z)$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? No.

- Therefore: **none**

Step 2:

- Assignment:** $Y = \text{True}$ (Decision Point)

- Outcome:**

- ✓ c3. $(\text{True} \vee Z) = \text{true}$

- c4. $(\neg Y \vee \neg Z)$

- c5. $(\neg Y \vee Z)$

- ✓ c6. $(\text{True} \vee \neg Z) = \text{true}$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? Yes. c4 => $Z = \text{False}$

- Assignment:** $Z = \text{False}$

- Outcome:**

- c4. $(\text{True}) = \text{true}$

- c5. $(Z) = \text{empty set} = \text{fail}$

- Consequence:** Fail.

Step 3:

- Backtrack to last decision:** Y

- Assignment:** $Y = \text{False}$

- Outcome:**

- c3. $(Y \vee Z)$

- ✓ c4. $(\text{True} \vee \neg Z) = \text{true}$

- ✓ c5. $(\text{True} \vee Z) = \text{true}$

- c6. $(Y \vee \neg Z)$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? Yes. c3 => $Z = \text{True}$

- Assignment:** $Z = \text{True}$

- Outcome:**

- c3. $(\text{True}) = \text{true}$

- c6. $(\neg Z) = \text{empty set} = \text{fail}$

Step 4:

- Backtrack to last decision:** X

- Assignment:** $X = \text{False}$

- Outcome:**

- c1. $(\neg X \vee Y \vee Z)$

- c2. $(\neg X \vee \neg Y \vee \neg Z)$

- ✓ c3. $(\text{True} \vee Y \vee Z) = \text{true}$

- ✓ c4. $(\text{True} \vee \neg Y \vee \neg Z) = \text{true}$

- c5. $(\neg Y \vee Z)$

- c6. $(Y \vee \neg Z)$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? No.

- Therefore: **none**

Step 5:

- Assignment:** $Y = \text{True}$ (Decision Point)

- Outcome:**

- ✓ c1. $(\text{True} \vee Z) = \text{true}$

- c2. $(\neg Y \vee \neg Z)$

- c5. $(\neg Y \vee Z)$

- ✓ c6. $(\text{True} \vee \neg Z) = \text{true}$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? Yes. c2 => $Z = \text{False}$

- Assignment:** $Z = \text{False}$

- Outcome:**

- ✓ c2. $(\text{True}) = \text{true}$

- c5. $(Z) = \text{empty set} = \text{fail}$

- Consequence:** Fail.

Step 6:

- Backtrack to last decision:** Y

- Assignment:** $Y = \text{False}$ (Decision Point)

- Outcome:**

- c1. $(Y \vee \neg Z)$

- ✓ c2. $(\text{True} \vee \neg Z) = \text{true}$

- ✓ c5. $(\text{True} \vee Z) = \text{true}$

- c6. $(Y \vee \neg Z)$

- Consequence:**

- Can you terminate early? No.

- Any pure symbols? No.

- Any unit clauses? Yes. c1 => $Z = \text{True}$

- Assignment:** $Z = \text{True}$

- Outcome:**

- ✓ c1. $(\text{True}) = \text{true}$

- c5. $(\neg Z) = \text{empty set} = \text{fail}$

- Consequence:** Fail.

(2) Delete one of the clauses that has three literals in it. Execute WalkSAT on the remaining 5 clauses, using $p=0.5$. You can use coin flips to get the random initial model, and also to select either greedy or random moves for each iteration.

Given $s = (X \vee Y \vee Z) \wedge (X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee Y \vee Z) \wedge (\neg X \vee \neg Y \vee \neg Z) \wedge (\neg Y \vee Z) \wedge (Y \vee \neg Z)$

Step 1:

Model: $X = \text{False}$, $Y = \text{True}$, $Z = \text{True}$ (flipping a coin)

X v Y v Z	X v -Y v -Z	-X v -Y v Z	-Y v Z	Y v -Z
TRUE	FALSE	TRUE	TRUE	TRUE

- Evaluate clauses:
 - Satisfied Clauses: 1, 3, 4, 5
 - Unsatisfied Clauses: 2
- Pick an unsatisfied clause: 2. $(X \vee \neg Y \vee \neg Z)$
- Random or Greedy ($p=0.5$)? Random (coin flip) Z.
- Flip selected symbol: $Z = \text{False}$

Step 2:

Model: $X = \text{False}$, $Y = \text{True}$, $Z = \text{False}$

X v Y v Z	X v -Y v -Z	-X v -Y v Z	-Y v Z	Y v -Z
TRUE	TRUE	TRUE	FALSE	TRUE

- Evaluate clauses:
 - Satisfied Clauses: 1, 2, 3, 5
 - Unsatisfied Clauses: 4
- Pick an unsatisfied clause: 4. $\neg Y \vee Z$
- Random or Greedy ($p=0.5$)? Greedy
 - If we flip $Y = \text{False}$, then 2, 3, 4, 5 are satisfied

X v Y v Z	X v -Y v -Z	-X v -Y v Z	-Y v Z	Y v -Z
FALSE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

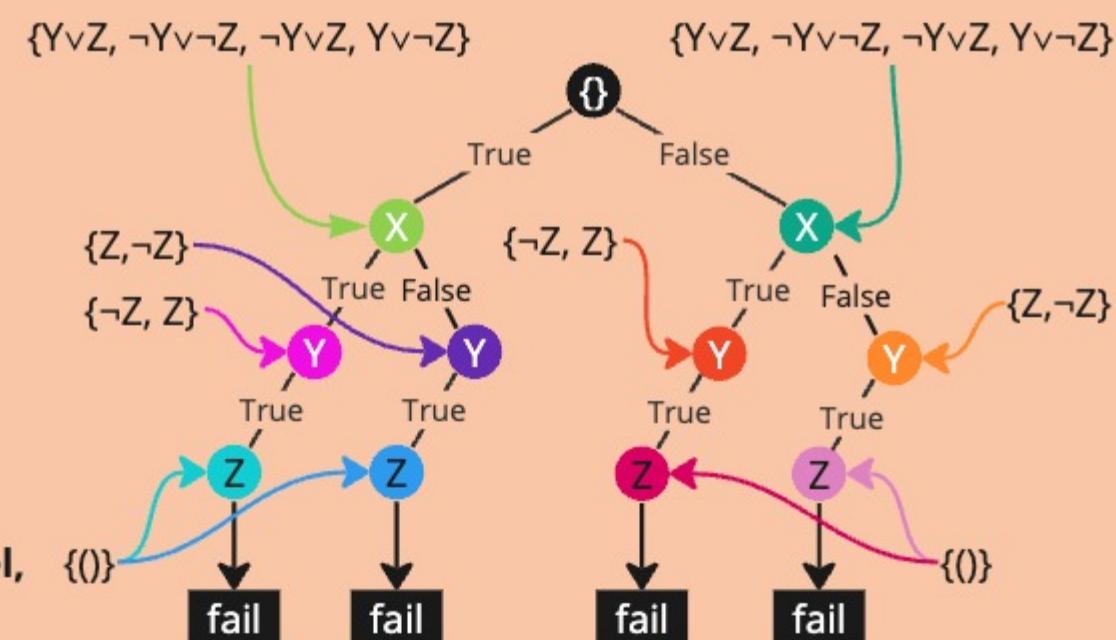
- Flip random symbol, Y
 - $Y = \text{False}$

Step 3:

Model: $X = \text{False}$, $Y = \text{False}$, $Z = \text{False}$

X v Y v Z	X v -Y v -Z	-X v -Y v Z	-Y v Z	Y v -Z
FALSE	TRUE	TRUE	TRUE	TRUE

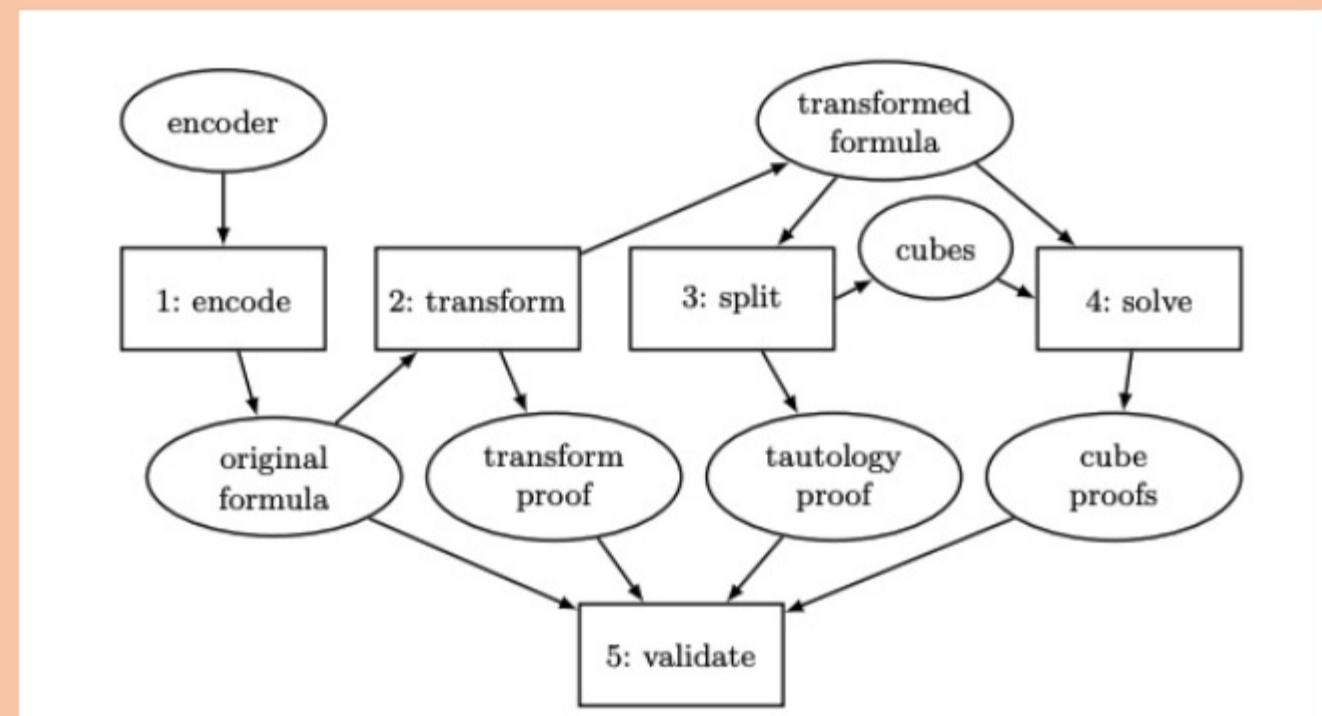
- Evaluate clauses:
 - Satisfied Clauses: 2, 3, 4, 5
 - Unsatisfied Clauses: 1
- Pick an unsatisfied clause: 1. $X \vee \neg Y \vee Z$
- Random or Greedy ($p=0.5$)? Greedy
 - If we flip $X = \text{True}$, then 1, 2, 3, 4, 5 are satisfied
 - If we flip $Y = \text{True}$, then 1, 2, 3, 5 are satisfied
 - If we flip $Z = \text{True}$, then 1, 3, 4, 5 are satisfied
- Solution Found



Scalability of SAT / Cube & Conquer: Addt'l Reading

Boolean Pythagorean Triples problem:

- Open problem in Ramsey Theory
- Can the set $N = \{1, 2, \dots\}$ of natural numbers be divided into two parts such that no part contains a triple (a, b, c) with $a^2 + b^2 = c^2$.
- Framework for solving consists of 5 phases:
 - **Encode** - Make sure that the representation of the problem as SAT instance is valid.
 - **Transform** - Reformulate the problem to reduce the computation costs later.
 - **Split** - Partition the transformed formula into many sub-problems. This uses a global heuristic in contrast to the local heuristic used by C&C.
 - **Solve** - Take the transformed formula and cube files as input and produce a proof of unsatisfiability of the transformed formula.
 - For easy problems (it can be solved within a day on a single core), a single cube file can be used and the incremental SAT solver will emit a single proof.
 - Hard problems will require two levels of splitting which allows for parallel solving.
 - boolean Pythagorean triples is very hard and requires 2 level splitting with a total runtime of 35,000 CPU hours or 4 CPU years.
 - **Validation** - check whether the proofs emitted in the prior phases are a valid refutation for the original formula.



The conclusions section of the paper gives an approximate amount of time for solving the problem. When parallelised, it took approximately two days using 800 CPUs. Suppose it was executed sequentially, taking 35,000 hours on one CPU. Approximately how many models is the solver dealing with per second?

Other Key Concepts:

Cube and Conquer (C&C) paradigm:

- A hybrid SAT method for hard problems that employs both look-ahead and Conflict Driven Clause Learning (CDCL) solvers.
- The most effective method to solve most hard combinatorial problems via SAT technology.
- **Step 1:** A look-ahead SAT solver partitions the problem into many sub-problems that are expressed as "cubes" (conjunctions) of the decisions (literals set to true).
- **Step 2:** The cubes are solved using a CDCL solver, also known as the "conquer" solver.
- Look ahead heuristics focus on global decisions while CDCL heuristics focus on local decisions. The idea behind C&C is to partition the problem until a short refutation arises. C&C together can solve problems faster than either pure approach.

Resolution Asymmetric Tautology (RAT) Clauses:

- A redundancy property of clauses that is computable in polynomial time and preserves satisfiability; all preprocessing, in-processing, and solving techniques in state-of-the-art SAT solvers can be expressed in terms of the addition and removal of RAT clauses.

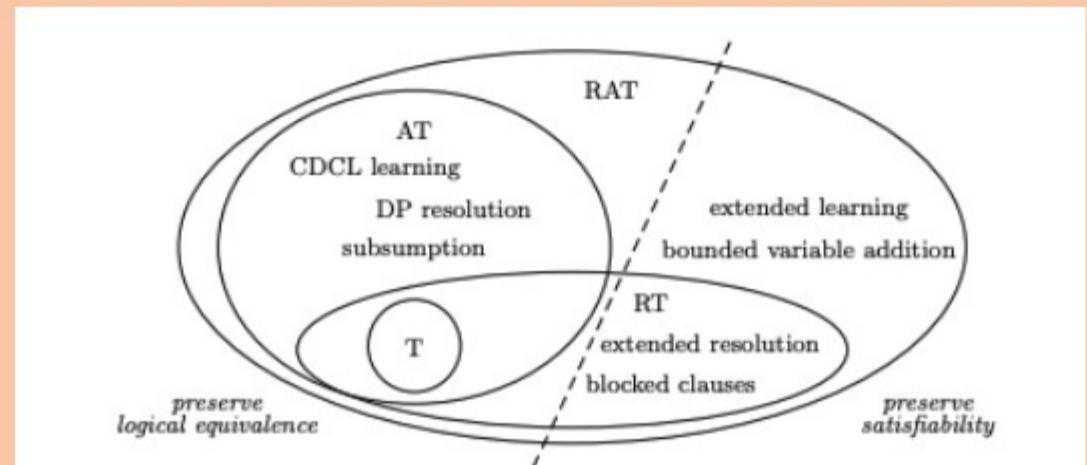


Fig. 1. Relationships between clause redundancy properties that can be computed in polynomial time. Techniques shown in an area denote the cheapest check one can apply to verify a proof trace from a SAT solver that uses that technique. All techniques used in state-of-the-art SAT solvers can be expressed as a sequence of RAT clauses [21]. The dashed line separates techniques that preserve logical equivalence and those that preserve satisfiability.

Deletion Resolution Asymmetric Tautology (DRAT) proof:

- A DRAT proof for a sentence s is a sequence of additions and deletions of clauses, starting with s , such that each addition is the addition of a RAT clause with respect to the current formula .
- A DRAT refutation is a DRAT proof containing \perp .
- DRAT refutations are correct proofs of unsatisfiability.

Week 6: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated.

Lesson 1: Supervised, unsupervised, and reinforcement learning

- Section 19: "Learning from Examples" from AI:AMA (**4th ed**)
 - *ref Section 18, from AI:AMA (3rd ed)*
- Section 19.1: "Forms of Learning" from AI:AMA (**4th ed**)
 - *ref Section 18.1, from AI:AMA (3rd ed)*

Lesson 2: Clustering with K-Means

- Section 4.8: "Clustering", from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed
- Section 4.8: "Iterative Distance-Based Clustering", from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed

Lesson 3: Principles and pitfalls of supervised learning

- Section 19.2: "Supervised Learning" from AI:AMA (**4th ed**)
 - *ref Section 18.2, from AI:AMA (3rd ed)*

Week 6 Further Reading:

- Section 4.7: "Instance-Based Learning", from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed

Study Session:

Wednesday, 18 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Lesson 1: Learning

Types of Learning

Types of learning depend on the type of feedback:

- **Supervised Learning**

- Observing input-output pairs, which are labelled.
- Learning a function which maps input to output (attributes to label prediction), the function is called the hypothesis.

- **Unsupervised Learning**

- Agent learns patterns without explicit feedback/labels.
- The most common example is clustering.

- **Reinforcement Learning**

- Usually the learning is embedded in an agent who operates in an environment.
- Agent learns from a series of reinforcements (rewards and punishments) resulting from its actions.
- The agent decides which of the actions prior to the reinforcement were most responsible for it and alters its actions to aim towards more rewards in the future.

Representation of States:

- **Atomic:**

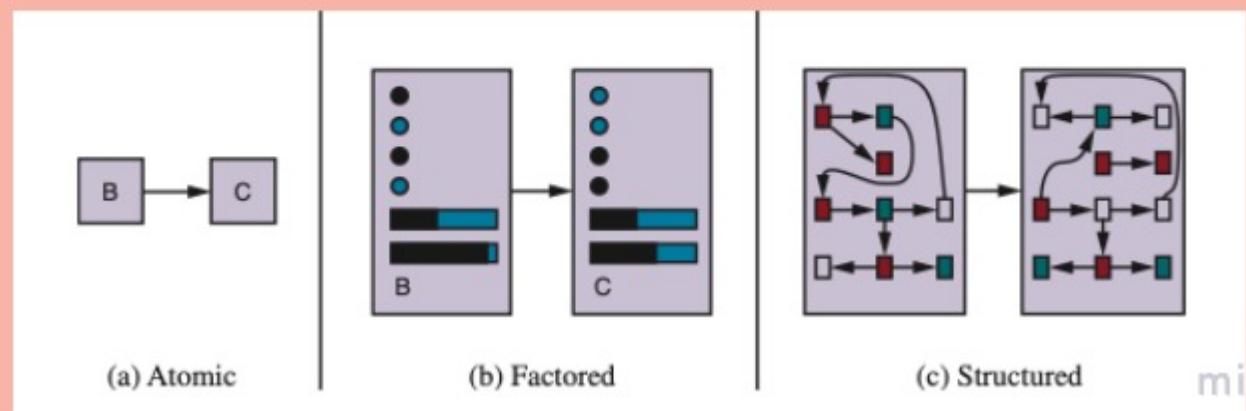
- Each state of the world is indivisible and has no internal structure; a black box.
- e.g. AI search problems and game playing typically have atomic states - the general search algorithm sees the state as one indivisible thing. Only problem-specific functions (such as heuristics) can access individual values within the state.

- **Factored:**

- Each example provided to the ML algorithm is represented as a sequence of vector attribute (feature) values.
- We can think of a model in propositional logic as being a type of factored representation.
- e.g. constraint satisfaction algorithms, propositional logic, planning, Bayesian networks.

- **Structured:**

- A state includes objects, each of which may have attributes as well as relationships to other objects.
- e.g. Natural Language Processing (NLP)



6.4 Quiz: Supervised, unsupervised, and reinforcement learning

Question 1

1 pts

What do supervised learning methods learn?

- A function that maps from attributes to attributes.
- A relation over the set of attributes.
- A function that maps from attributes to predicted labels.
- A function that maps from labels to predicted attribute values.

Question 2

1 pts

Both supervised and unsupervised types of learning need a set of examples to be provided. Which types of learning need labelled examples?

- Both types of learning need a set of labelled examples.
- Supervised learning needs labelled examples, whereas unsupervised learning works on unlabelled examples.
- Unsupervised learning needs labelled examples, whereas supervised learning works on unlabelled examples.
- Both types of learning work on unlabelled examples.

Question 3

1 pts

Which of the following is a common task in unsupervised learning?

- Binning
- Clustering
- Packing
- Correlating

Question 4

1 pts

How are examples represented for supervised and unsupervised machine learning methods?

- A factory representation is used, where the examples are generated on-demand to avoid the overhead of storing them all.
- A layered representation is used, where the most important attributes are stored in the top layer.
- An atomic representation is used, allowing the machine learning algorithm to calculate the distance between examples but not access the individual attributes.
- A factored representation is used, allowing the machine learning algorithm to access each attribute of every example.

Question 5

1 pts

Which of the following sentences best describes how reinforcement learning systems learn?

- Clustering similar examples in the set of examples they are given.
- Passively observing their environment for a long period of time.
- Taking actions in their environment and learning from the good or bad outcomes of those actions.
- Constructing a function, called a hypothesis, that predicts the label given a set of labelled examples.

Quiz Keywords: Supervised learning (1, 2, 4); unsupervised learning (2, 3, 4); labels (2); labelled set of examples (2); tasks (3); binning (3); clustering (3); packing (3); correlating (3); example representation (4); reinforcement learning (5)

Lesson 2: Unsupervised Learning - Clustering with K-Means

- All unsupervised learning methods learn from the distribution of examples in the dataset.
- We do not have labels, so we are not aiming to calculate $P(Y|x)$ but instead $P(x)$. We aim to learn if some examples are more likely than others.

Clustering - Clustering techniques apply when there is no class to be predicted but rather when the instances are to be divided into natural groups.

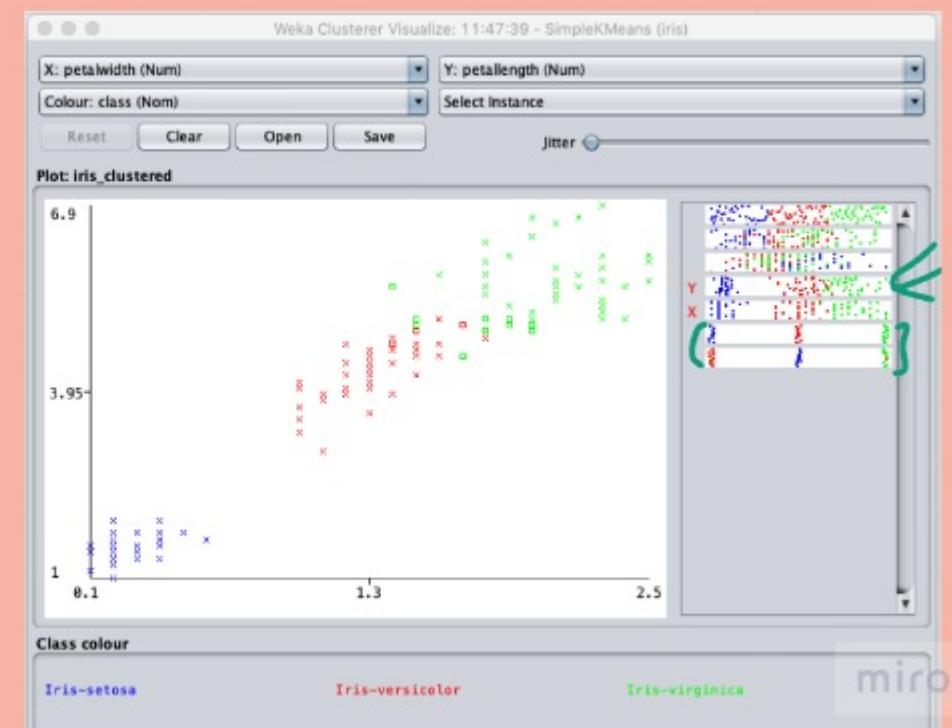
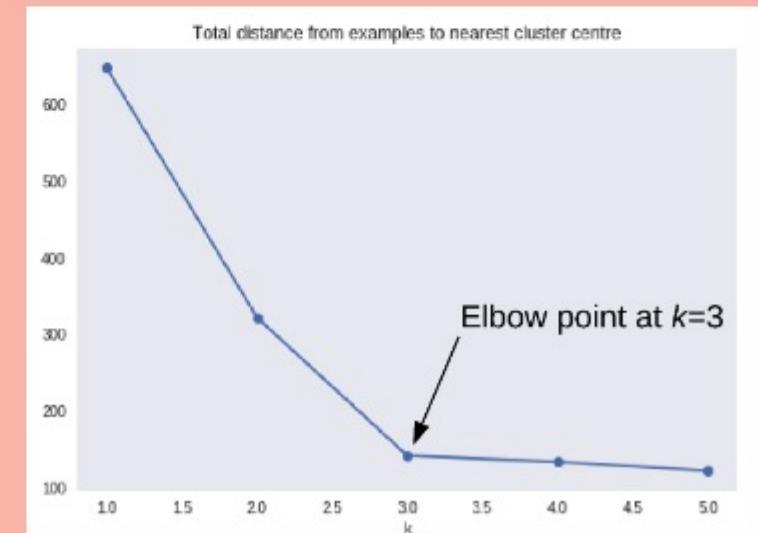
K-Means Algorithm (iterative distance-based clustering):

1. Choose K cluster centers at random.
2. Allocate each point in the data to the cluster whose center is closest to that point given some distance measure (e.g. Euclidean distance).
3. Calculate new cluster centers as the average of the coordinates of points belonging to each cluster.
4. Repeat steps 2-3 until convergence (i.e. until the centers stop changing with each iteration).

Properties:

- The clustering method can be proven to choose the cluster center which minimizes the total squared distance from each of the cluster's points to its center, but the minimum is a local one: there is no guarantee that it is the global minimum.
- The final cluster allocation is sensitive to the initial (random) allocation of clusters.
- To overcome the limitations and to increase the chance of finding a global minimum, the algorithm is often run several times for a given value of K , with different initial choices, and the result is chosen which minimizes the total distance.

Additionally, we need a method for choosing K . This can be done by doing the above (i.e. multiple evaluations for a given K), for a number of values of K , starting with $K=1$. We then plot the minimum total distance for each value of K and look for an elbow point.



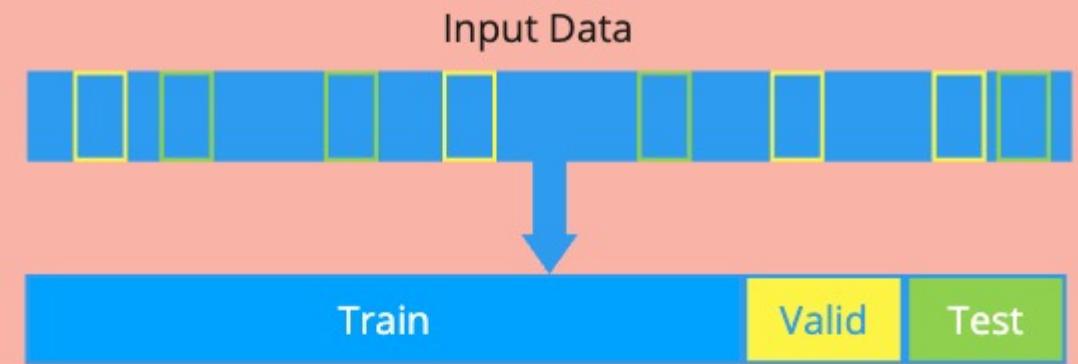
Activity: K-means clustering on real data



Lesson 3: Supervised Learning

Two types of problems:

1. **Classification** - predicting some finite number of distinct classes; e.g. cat vs. dog, integer values
2. **Regression** - predicting a real number; e.g. stock prices, floating point values



Hypotheses:

The task of supervised learning is...

*Given a **training set** of N example input-output pairs*

$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$,

where each pair was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f , given that x and y can be any value (they need not be numbers) and the function, h , is a hypothesis.

The aim is to search through the space of hypotheses (**hypothesis space**) to find one that will perform well. In some cases f will be stochastic, in which case our aim is to learn the probability distribution $P(y|x)$.

We select the hypothesis based on the **training set** and then measure the accuracy of a hypothesis on the basis of a **test set**, which has a set of examples distinct from the training set. (Actually, in practice, we would want to use 3 sets: training, validation, and test; where the validation set is used for model selection and the test set is used for model assessment).

Desirable properties of a hypotheses:

- **Generalizability** - if the hypothesis correctly predicts the value of y for previously unseen examples; the problem of choosing a hypothesis which is too specific to the training data is called overfitting.
 - h generalizes well if it accurately predicts the outputs of the test set.
- **Consistency** - When the hypothesis agrees with all of the data.

Choosing a hypothesis:

- We may have more than one hypothesis consistent with the data, in which case we want to choose the simplest one (Occam's razor).
- **There is a tradeoff between the expressiveness of a hypothesis space and the computational complexity of finding a good hypothesis within it.**
 - Turing Machines - Very expressive space
 - The alternative is to restrict the search to the space of less expressive hypotheses (which are less complex to search along).
- Another trade-off: a more complex hypotheses may fit the training better, but generalize less well to unseen examples; this is called overfitting.
- **bias-variance tradeoff** - a choice between a more complex, low-bias hypotheses that fit the training data well and simpler, low variance hypotheses that may generalise better.
- Supervised learning can be done by choosing a hypothesis which is most likely given the data:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}} P(h|data).$$

By Bayes' rule this is equivalent to

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}} P(data|h) P(h).$$

- We can choose the prior $P(h)$ such that complex hypotheses are unlikely.

Realizability:

- A learning problem is realisable if the **hypothesis space** contains the true function; but the true function is not known so we can't always tell whether the given learning problem is realisable.

Potential Issues:

- Attempting to learn using an inappropriate hypothesis space.
- Using the entire data set for training and failing to test whether the hypothesis generalizes well.
- Learning a very complex hypothesis that is accurate on the training set but less accurate on the test set, and probably also less accurate when deployed in the real world.
- **Overfitting** - We say a function is overfitting the data when it pays too much attention to the particular dataset it is trained on causing it to perform poorly on unseen data.

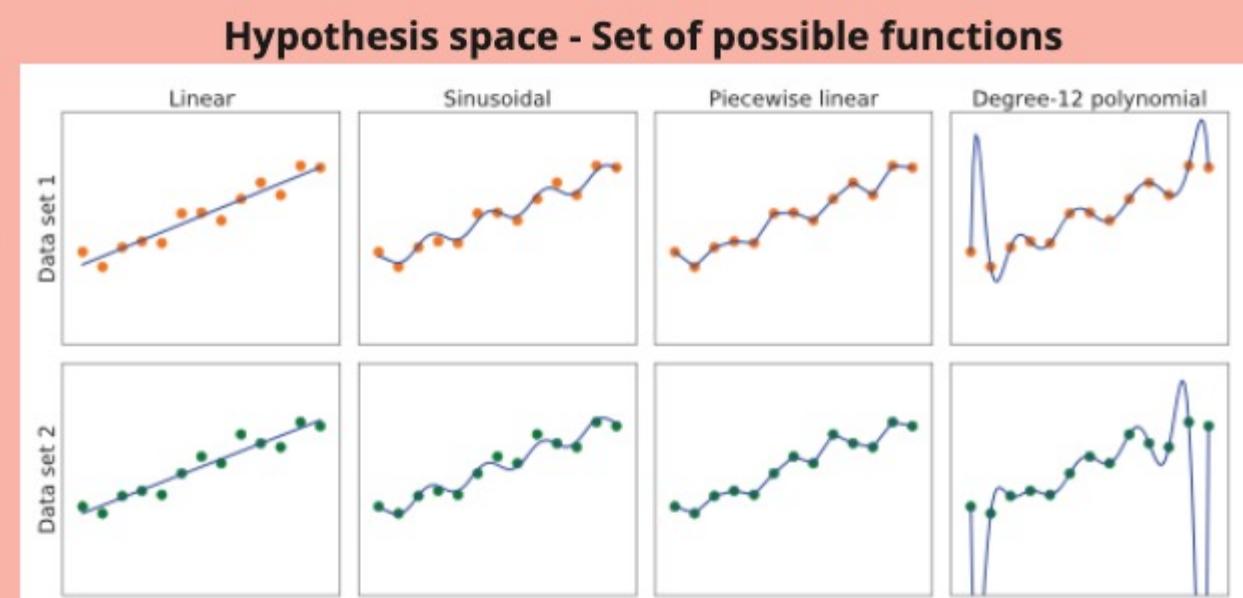
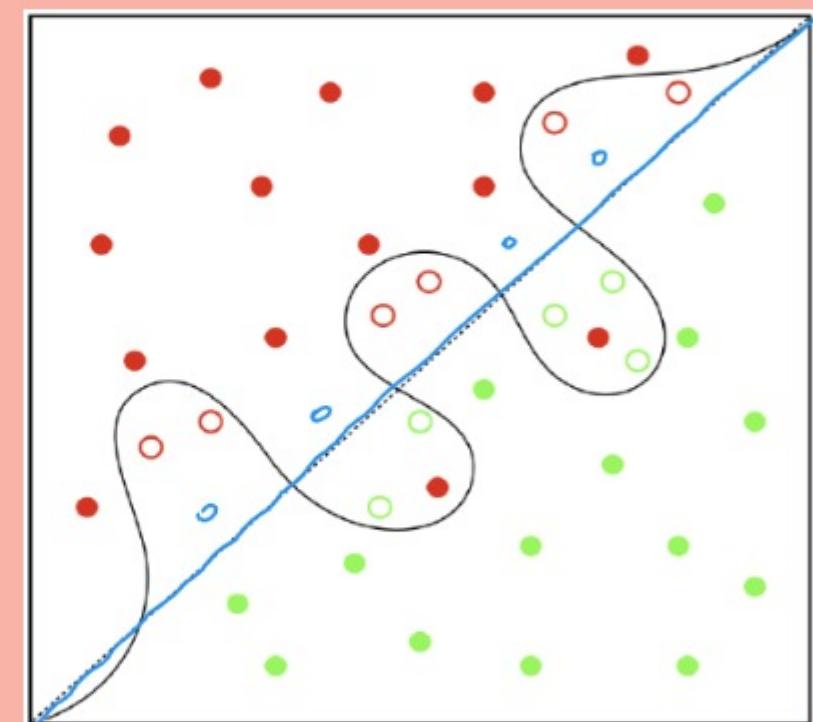


Figure 19.1 Finding hypotheses to fit data. **Top row:** four plots of best-fit functions from four different hypothesis spaces trained on data set 1. **Bottom row:** the same four functions, but trained on a slightly different data set (sampled from the same $f(x)$ function).



Reinforcement Learning

- The overall aim of any reinforcement learning method is to maximize the accumulated rewards over time.
- Reinforcement learning is useful when there are no labelled examples to learn from, but it can also be used as an alternative to supervised learning to avoid up-front cost of assembling a large set of training examples.
- Engineering the reward function is not necessarily straight-forward, and it is essential to the success of reinforcement learning.
- A reinforcement learning system learns by taking actions in an environment and gaining feedback from the environment (i.e. evaluating the outcomes of the actions).

Instance-Based Learning

- In instance-based learning, the training examples are stored verbatim and a distance function is used to determine which member of the training set is closest to any instance whose class is unknown. Once the nearest instance has been located, its class is predicted for the unknown.
- Can be distance based or nearest neighbors. Distance based is simple and effective, but slow; nearest neighbors can be found more efficiently.
- Advantage = New examples can be added to the training set at any time.

Euclidean Distance Function:

$$d((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$

Nearest Neighbor Classifiers:

- Uses proximity or similarity to classify unclassified attributes.
- Search distance is a critical factor in the accuracy of the classifier.
- A dataset that is too small the results become noisy with a high probability of overfitting.
- A dataset that is too large has a high probability of mis-classification.
- Model induction is relatively inexpensive, however deduction is expensive.
- **k-nearest-neighbor** method uses more than one nearest neighbor and the majority class of the closest **k** neighbors (**or the distance weighted average**) is assigned to the new instance.

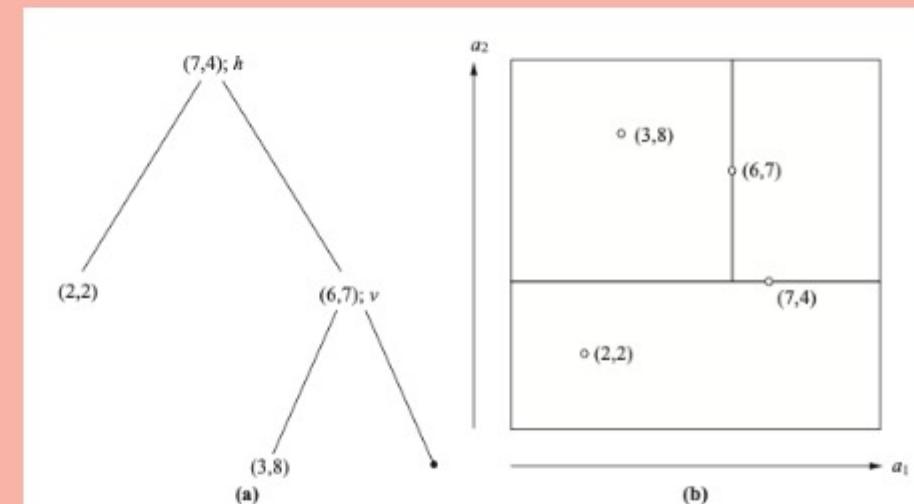


FIGURE 4.12

A kD-tree for four training instances: (a) the tree and (b) instances and splits.

kD-tree:

- A binary tree that divides the input space with a hyperplane and then splits each partition again, recursively.
- The data structure is called a kD-tree because it stores a set of points in **k**-dimensional space where **k**=number of attributes.

6.9 Quiz: K-means clustering, supervised learning

KEYWORDS: relationship h and f ; supervised learning

Question 1 1 pts

How are the two functions h and f related in supervised learning?

- f is an approximation of h .
- h is an approximation of f .
- h and f are the same function.
- h and f are inverses of each other.

Question 2 1 pts

What is **overfitting**?

- Learning a hypothesis that is too specific to the test set and is less accurate on the training set.
- Learning a hypothesis that is too specific to the entire data set (including both training and test sets), and is less accurate when deployed.
- Learning a hypothesis that is too specific to the training set and is less accurate on the test set.
- Learning a hypothesis using the entire data set and failing to test it.

Question 3 1 pts

What does it mean for a hypothesis to **generalize well**?

- The hypothesis is accurate for unseen examples (i.e. examples not in the training set).
- The hypothesis is accurate for examples in the training set
- The hypothesis is inaccurate for examples in the test set, but accurate when deployed in the real world.
- The hypothesis is inaccurate for examples in the training set, but accurate for examples in the test set.

Question 6 4 pts

Given the following set of eight unlabelled examples:

x	y
1	1
1	2
2	1
2	2
3	3
3	4
4	3
4	4

Suppose you are applying k-means clustering to find two clusters (named C1 and C2), and the coordinates of the initial cluster centres are as follows:

C1: $x=2.5$, $y=1.5$

C2: $x=3.5$, $y=3$

Perform **one step** of the k-means algorithm, and enter the new cluster centres into the four boxes below:

C1 x =

C1 y =

C2 x =

C2 y =

Hint: It may help to draw a rough scatterplot of the eight examples.

KEYWORDS: hypothesis space definition

1 pts

What is the **hypothesis space**?

- Given two hypotheses h_1 and h_2 , the hypothesis space is the proportion of times that h_1 and h_2 predict a different label for examples in the test set (informally, the space between the two hypotheses).
- For a supervised learning task, the hypothesis space is the set of all possible hypotheses that may be selected.
- Given a hypothesis h , the hypothesis space of h is the space of all training and test examples where h is accurate.
- The hypothesis space is the space of all possible examples, including all training, test and other unseen examples.

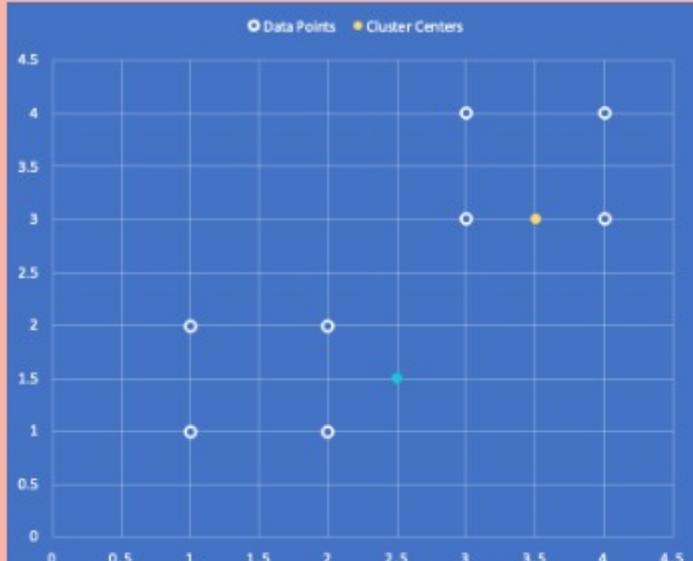
KEYWORDS: supervised learning; training set; test set split

Question 5

1 pts

In supervised learning, is it useful to split the data set into training and test sets, and if so, why is it useful?

- It is not useful – we should use the entire data set for training to maximise the accuracy of the hypothesis.
- It is useful because reducing the number of training examples helps to prevent overfitting.
- It is useful because the test set is used to measure generalization.
- It is useful because the test set is used for a second training phase where the choice of hypothesis is improved.



Data points		Distance From		Allocation
x	y	Cluster 1	Cluster 2	
1	1	1.58	3.20	Cluster 1
1	2	1.58	2.69	Cluster 1
2	1	0.71	2.50	Cluster 1
2	2	0.71	1.80	Cluster 1
3	3	1.58	0.50	Cluster 2
3	4	2.55	1.12	Cluster 2
4	3	2.12	0.50	Cluster 2
4	4	2.92	1.12	Cluster 2

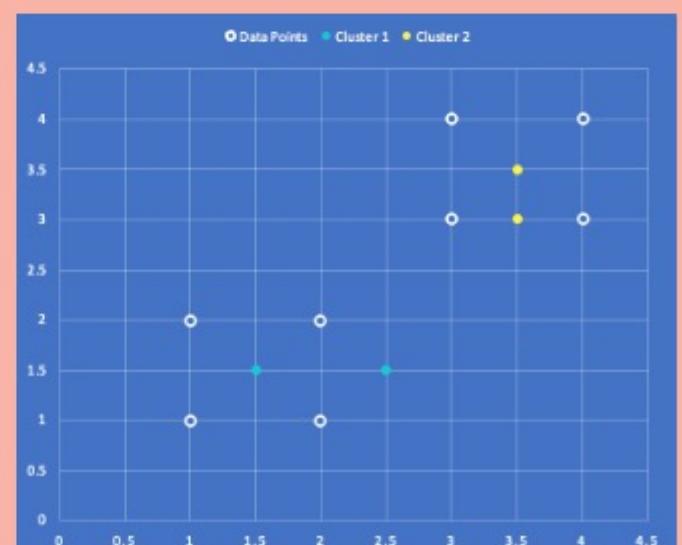
Step 1: There are (2) visible clusters so use $k=2$ for the number of clusters. Typically these would be chosen at random but here we have been provided with their specific locations:

$$1. \quad C1 = (2.5, 1.5), C2 = (3.5, 3)$$

Step 2: Allocate each point in the data to the cluster whose center is closest to that point given some distance measure; Euclidean distance: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Step 3: Calculate new cluster centers as the average of the coordinates belonging to each cluster.

x	y	Cluster 1 Original
2.5	1.5	Cluster 1 New
1.5	1.5	Cluster 1 New
3.5	3	Cluster 2 Original
3.5	3.5	Cluster 2 New



Week 7: Reading

Please note the primary textbook for this cohort is "AI: A Modern Approach" **4th edition** (AI:AMA) but that canvas may not be updated. Waiting to hear back from the tutors as there isn't a direct cross-over to the 4th edition for some of this reading.

Lesson 1:

- Section 1.2: "Simple Examples" from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed
- Section 3.2: "Linear Models" from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed
- Section 4.6: "Linear Models" from Data Mining: Practical Machine Learning Tools and Techniques, 4th ed

Lesson 2:

- Section X.X: "Artificial Neural Networks" from AI:AMA (**4th ed**) ??
 - *ref Section 18.7, from AI:AMA (3rd ed)*
- Section X.X: "Neural network structures" from AI:AMA (**4th ed**) ??
 - *ref Section 18.7.1 from AI:AMA (3rd ed)*
- Section X.X: "Single-layer feed-forward neural networks (perceptrons)" from AI:AMA (**4th ed**)
 - *ref Section 18.7.2 from AI:AMA (3rd ed)*
- Section X.X: "Multilayer feed-forward neural networks" from AI:AMA (**4th ed**) ??
 - *ref Section 18.7.3 from AI:AMA (3rd ed)*

Week 7 Further Reading:

- Section 19.6: "Regression and Classification" from AI:AMA (**4th ed**)
 - *ref Section 18.7, from AI:AMA (3rd ed)*
- Section X.X: "Learning in multilayer networks" from AI:AMA (**4th ed**) ??
 - *ref Section 18.7.4, from AI:AMA (3rd ed)*
- Section X.X: "Learning neural network structures" from AI:AMA (**4th ed**) ??
 - *ref Section 18.7.5, from AI:AMA (3rd ed)*

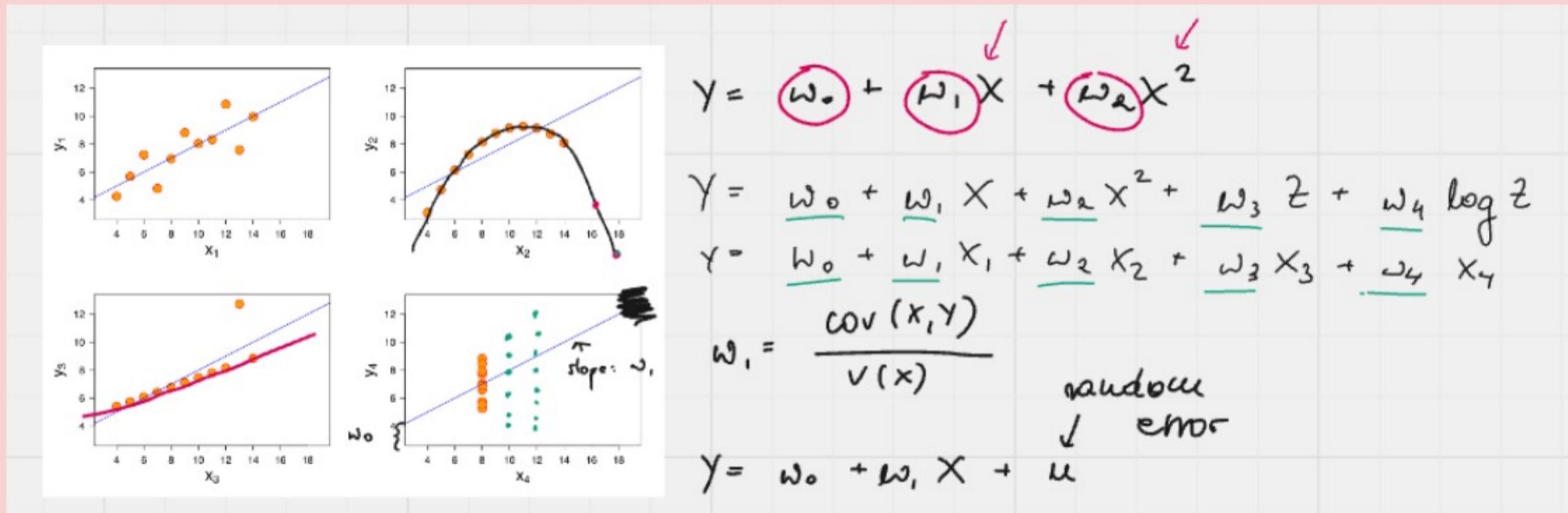
Study Session:

Monday, 23 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Linear Regression

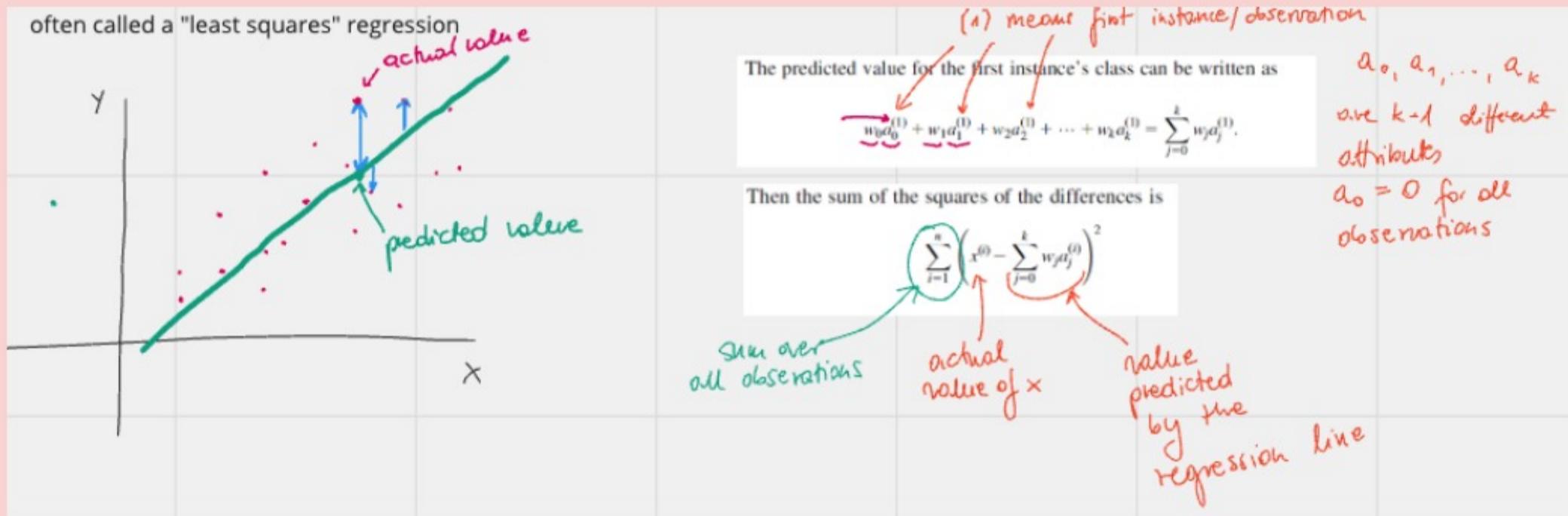
- **Linear Regression (Numeric Prediction)** - Express the class as a linear combination of the attributes with weights calculated from the training data.
- Linear regression focuses on the regression task; i.e. predicting a real number.
- We can think of it as corresponding to finding a hypothesis in the hypothesis space.
- $y = w_0 + w_1 X$ where w_0 and w_1 are weights which we need to find values for.
- We can have a multivariate regression, whereby X contains lots of different attributes.
 - $y = w_0 + w_1 X_1 + w_2 X_2 + w_3 X_3 \dots$ and so on
- In general, some problems are better suited to a linear regression than others (see the Anscombe Quartet)



- Note that the model in 2 could still be modeled via a linear regression but we would also need to include a squared term in there. This is still an example of a linear regression because the hypothesis is linear in parameters.

How do we choose the values of the weights (w_0, w_1, \dots)?

- We do so in a way which minimises the sum of squared residuals - this is why a linear regression is often called a "least squares" regression.



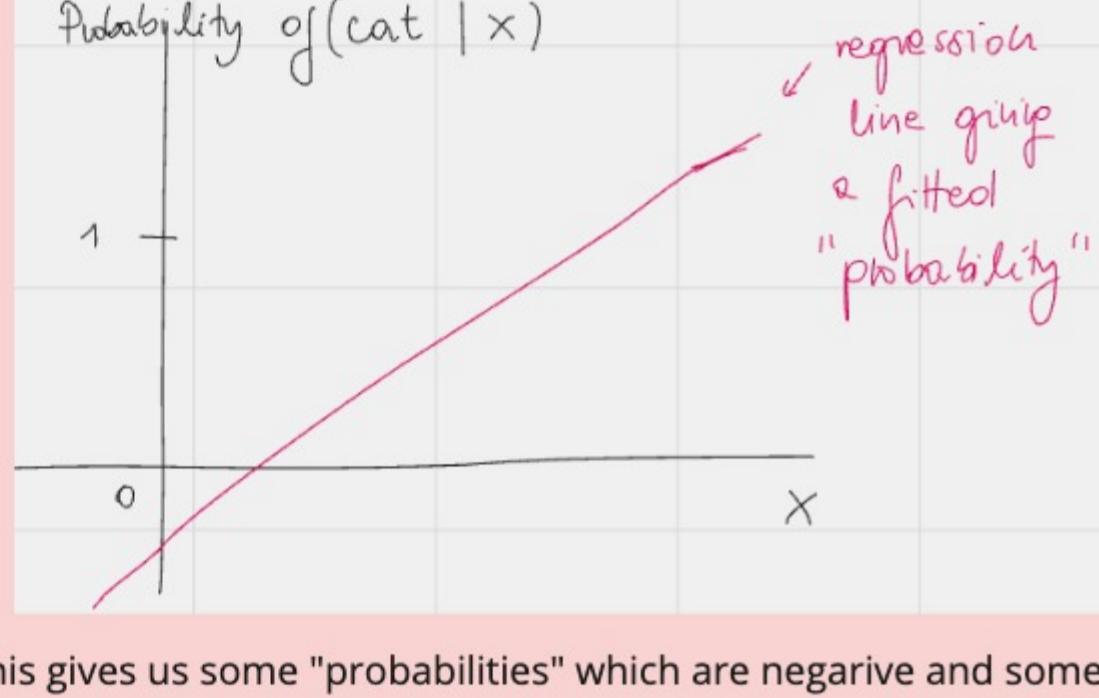
Why the sum of squares?

- The squares make each of the values in the summation non-negative.
- The squares punish larger deviations more.
 - $(0 - 1)^2 = 1$
 - $(3 - 0)^2 = 9$
 - $9^2 - 8^2 = 81 - 64$

In order to be able to fit a linear regression we need to have more examples (observations) than attributes - so if we have a sample of n observations and we are interested in k attributes, we need $n > k$, or in other words at least $n = k + 1$ to account for k attributes + the constant term.

Logistic Regression and Classification in General

- Classification learning** - the model is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples. AKA Supervised Learning.
- Logistic Regression (Linear Classification)** - a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on. Only instead of approximating the 0 and 1 values directly, thereby risking illegitimate probability values when the target is overshot, logistic regression builds a linear model based on a transformed target.
- We use **logistic regression** in classification problems, when we are trying to predict the membership in one of two classes - e.g. cat vs dog.
 - There is an extension of this model to problems in which we aim to predict more than 2 classes, using the **softmax** function, but this is not a part of the course.
- We could, in theory, use linear regression to estimate the probability of belonging to a class.



- The problem is that this gives us some "probabilities" which are negative and some which are greater than 1, so it is not a perfect solution.
- A **logistic regression** replaces the target variable y (which can only be 0 or 1) with the **log-odds** of y , and effectively performs a linear regression on those.

We can convert a probability into an **odds ratio**:

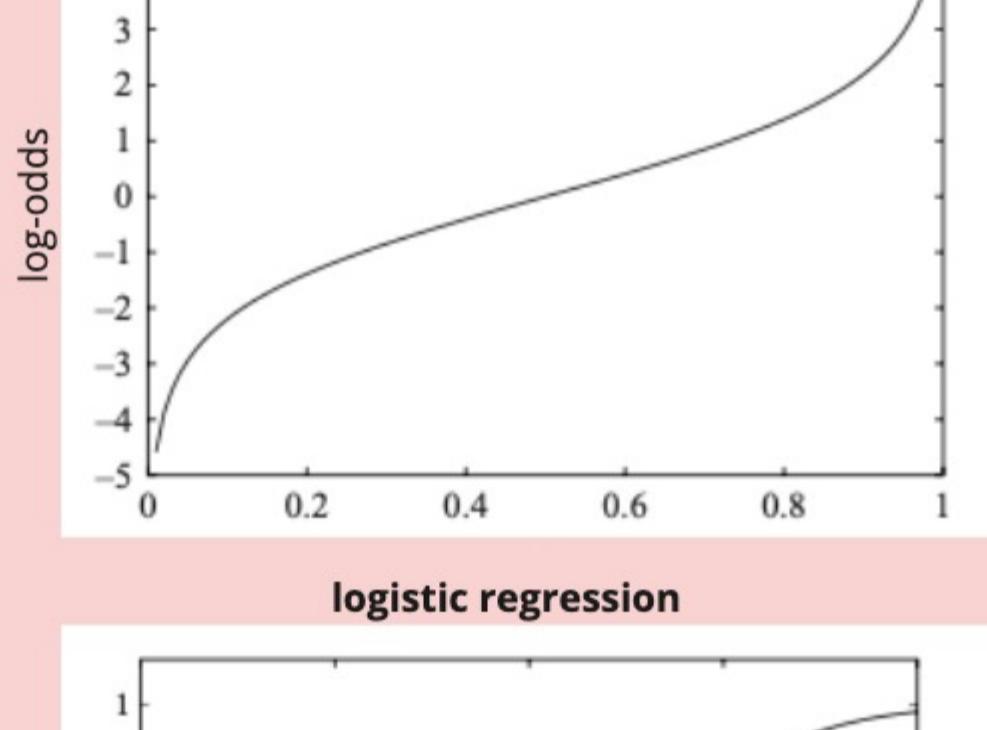
$$\text{odds}_i = \frac{p_i}{1-p_i}$$

logit = natural log (\ln) of an odds (base e not base 10)

$$\text{log-odds: } \ln(\text{odds}) = \text{logit}(P) = \ln\left(\frac{P}{1-P}\right)$$

We effectively estimate this model:

$$\ln\left(\frac{P}{1-P}\right) = -w_0 - w_1 a_1 - \dots - w_k a_k$$



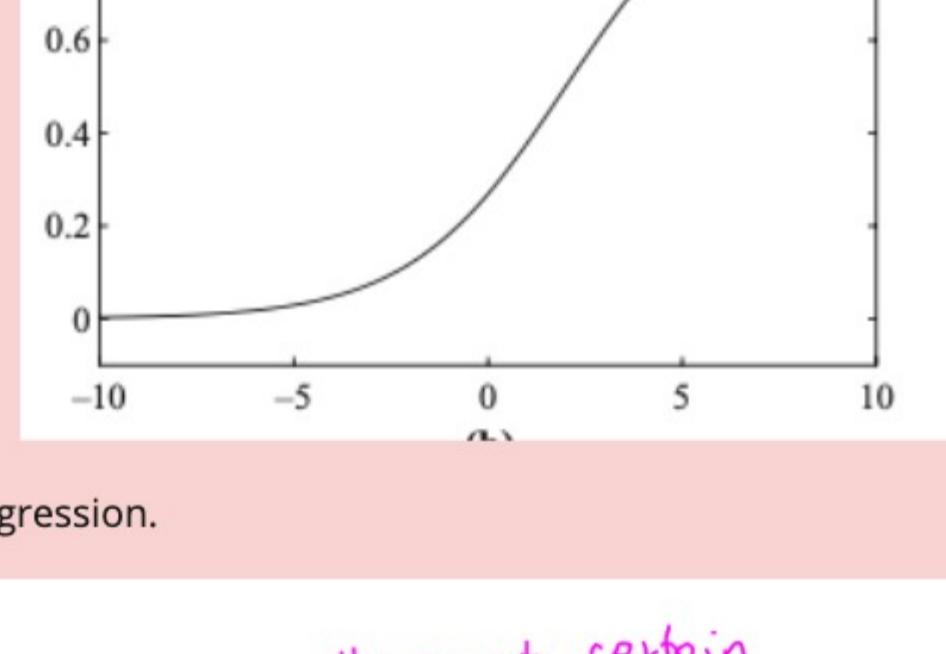
logistic regression

- Given that we effectively model the log-odds, how do we obtain the predicted probabilities?** We use the following transformation of the above formula which ensures that the probabilities are between 0 and 1:

$$\Pr[1|a_1, a_2, \dots, a_k] = 1/(1 + \exp(-w_0 - w_1 a_1 - \dots - w_k a_k)),$$

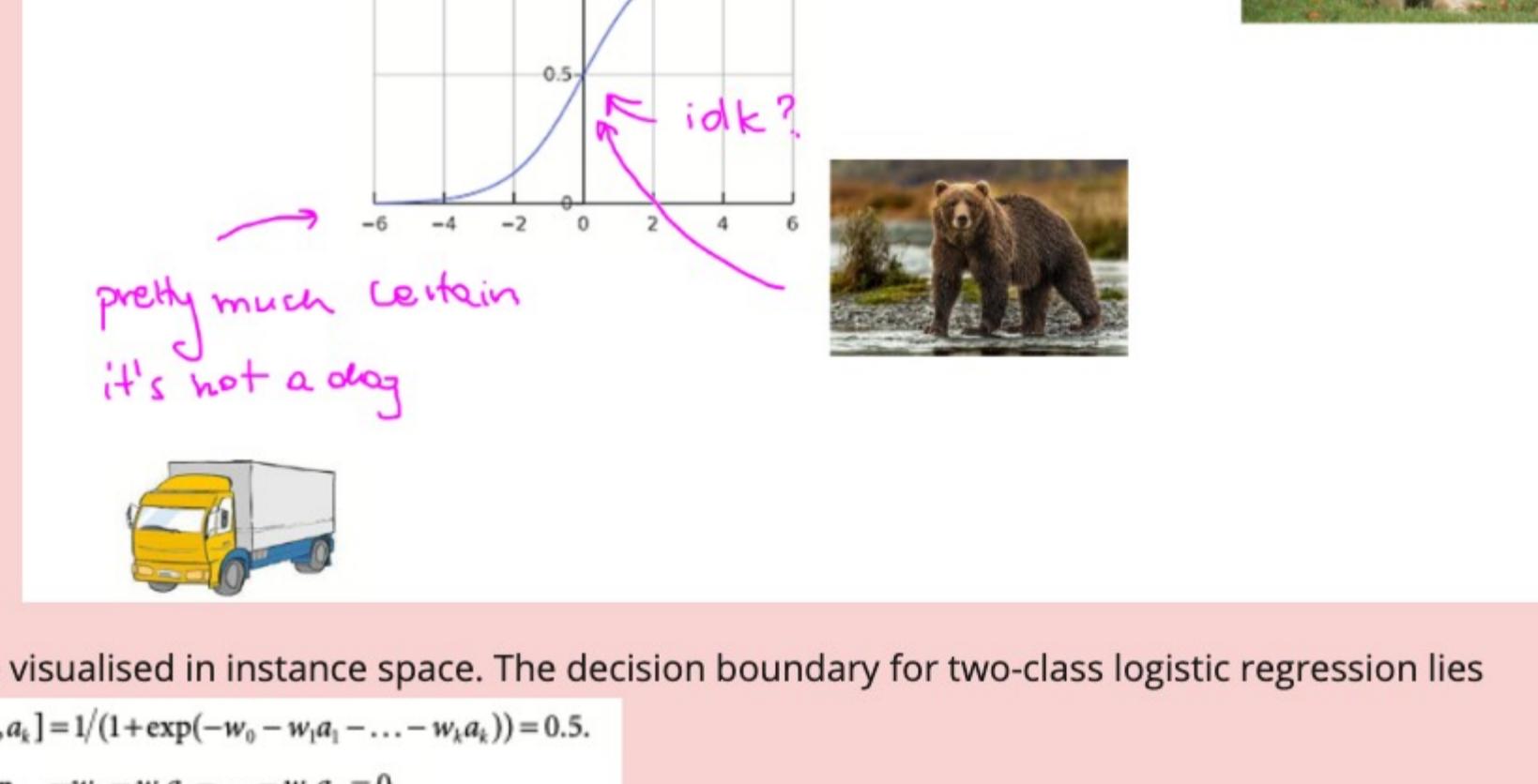
which is the same as...

$$P(y=1 | a_1, \dots, a_k) = 1 / (1 + e^{-t}) \text{ where } t = w_0 + w_1 a_1 + \dots + w_k a_k$$



- Why choose the logit function? Couldn't we find a simpler choice?** An alternative that we could use is the **hard threshold function**: $H(z) = 1$ if $z \geq 0$, and 0 otherwise

But this creates a hard decision boundary, unlike the soft decision boundary of the logistic regression.



- The use of linear functions for classification can easily be visualised in instance space. The decision boundary for two-class logistic regression lies where the prediction probability is 0.5, i.e.:

$$\Pr[1|a_1, a_2, \dots, a_k] = 1/(1 + \exp(-w_0 - w_1 a_1 - \dots - w_k a_k)) = 0.5.$$

$$\text{This occurs when } -w_0 - w_1 a_1 - \dots - w_k a_k = 0.$$



$$P(y=x | a_1, a_2) = \frac{1}{1+e^{-(w_0 + w_1 a_1 + w_2 a_2)}}$$

Decision: Predict x if $P(y=x | a_1, a_2) \geq 0.5$

Boundary between x and o

$$P(y=x | a_1, a_2) = 0.5$$

$$\frac{1}{1+e^{-(w_0 + w_1 a_1 + w_2 a_2)}} = 0.5$$

$$e^{-(w_0 + w_1 a_1 + w_2 a_2)} = 1$$

$$\text{LHS} = 1 / (1+1) = 1/2 = 0.5$$

- In general, we find the parameters of the logistic regression (the weights w_0, \dots, w_k) through one of two methods:
 - Using maximum likelihood
 - Starting with arbitrary values for the weights and then using gradient ascent to iteratively improve the goodness of fit.

Perceptron Training Rule:

- There is also a third way of classifying data, provided that the data are linearly separable. This method is called the **perceptron training rule**.
- The general idea is that we are trying to go through all instances (examples/observations) and update the weights (w_0, \dots, w_k) iteratively until all examples are correctly classified.
- Each iteration goes through all the training instances. If a misclassified instance is encountered, the parameters of the hyperplane are changed so that the misclassified instance moves closer to the hyperplane, or maybe even across the hyperplane to the correct side.
- The algorithm converges in a finite number of iterations if the data are linearly separable, but otherwise will not terminate.
- The resulting **hyperplane** is called the **perceptron**.

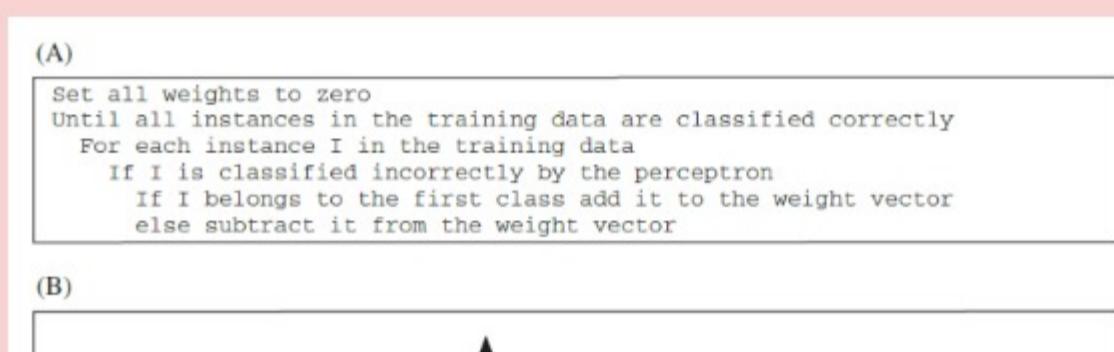
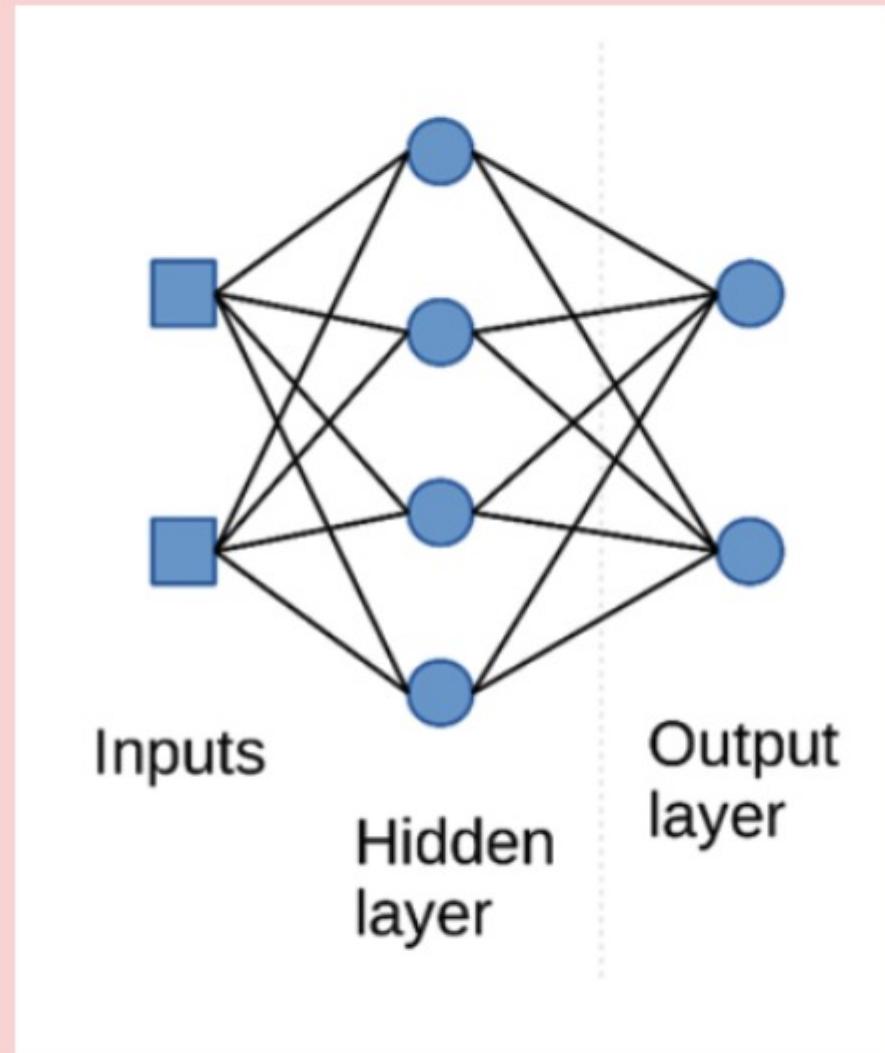


FIGURE 4.11

The perceptron: (A) learning rule; (B) representation as a neural network.

Activity

How many weights does this network have in total:



Hidden Layer

- per neuron: 2 inputs => $2 + 1$ weights = 3 weights per neuron
- total for hidden layer: $4 \text{ neurons} * 3 \text{ weights/neuron} = \mathbf{12 \text{ weights}}$

Output Layer

- per neuron: $4 + 1$ weights = 5 weights per neuron
- total for output layer: $2 \text{ neurons} * 5 \text{ weights/neuron} = \mathbf{10 \text{ weights}}$

Network Total:

- $12 + 10 = \mathbf{22 \text{ weights}}$

Neural Network Structures

Artificial neural networks are composed of **neurons**, **nodes**, or **units**, connected by **directed links**.

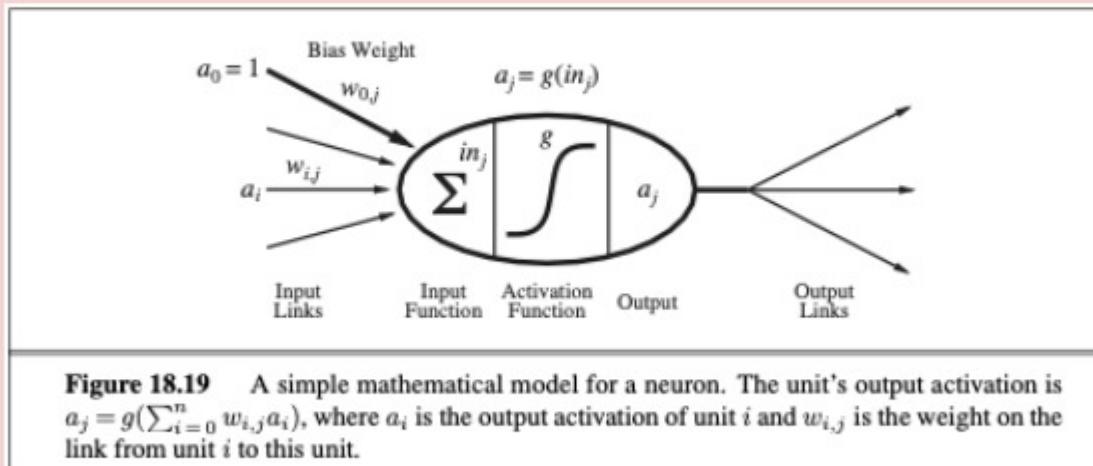


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

- A **link** from i to j serves to propagate the activation a_i from i to j .
- Each link also has a numeric **weight** $w_{i,j}$ associated with it which determines the strength and sign of the connection.
- It then applies an **activation function** g to this sum to derive the output.

inputs (linear function)

$$in_j = \sum_{i=0}^n w_{i,j} a_i .$$

outputs (activation function)

$$a_j = g(in_j) = g\left(\sum_{i=0}^n w_{i,j} a_i\right)$$

Activation Functions:

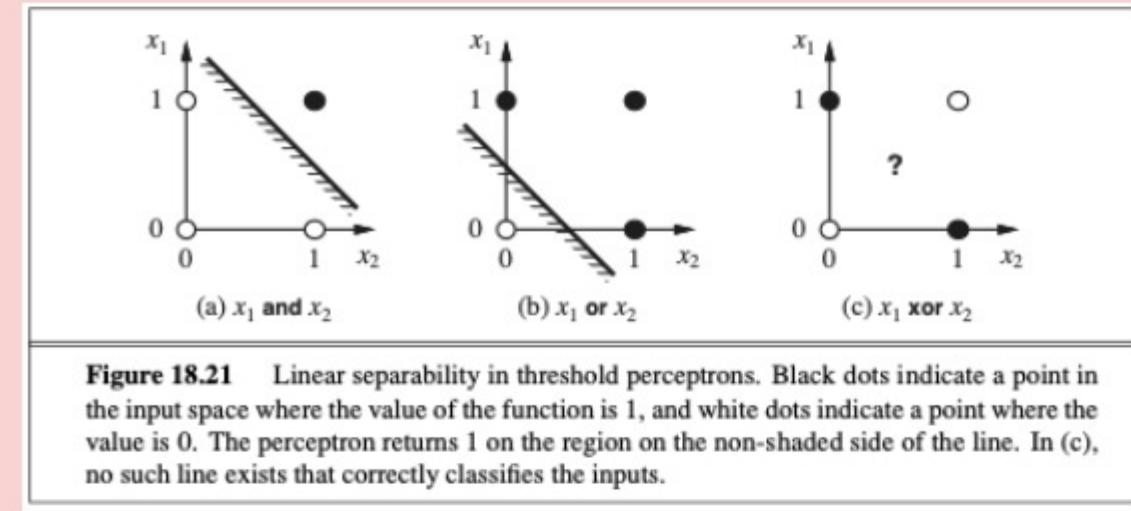
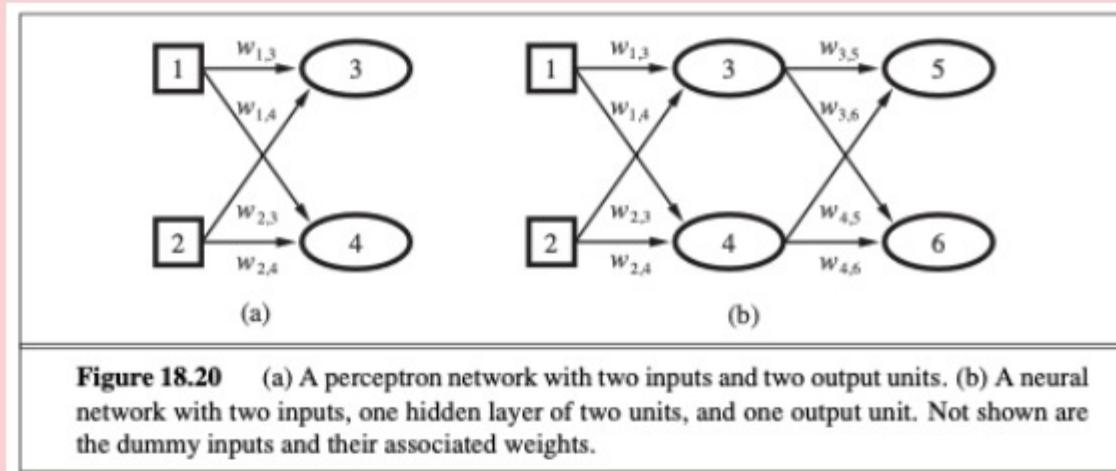
- Popular **activation functions** include:
 - logistic or **sigmoid function** - $\sigma(x) = 1/(1+e^{-x})$
 - **ReLU** (Rectified Linear Unit) - $\text{ReLU}(x) = \max(0, x)$
 - **softplus** - $\text{softplus}(x) = \log(1+e^x)$
 - A smooth version of ReLU; the derivative of the softplus function is the sigmoid function.
 - **tanh** (hyperbolic tangent) -
 - The range of tanh is $(-1, +1)$. It is a scaled and shifted version of the sigmoid.
- If the activation function has a **hard threshold** the unit is called a **perceptron** or a **logistic function**.
- It is important that the activation function(s) be non-linear because if all activation functions are linear then the neural network boils down to good ole' linear regression.

Types of Neural Networks:

- A **feed-forward network** has connections only in one direction. Every node receives input from "upstream" nodes and delivers output to "downstream nodes". There are no loops. It is usually arranged in **layers** and will have one or more layers of **hidden units** that are not directly connected to the outputs of the network.
 - **Single-layer feed-forward neural networks (perceptrons)**
 - **Multilayer feed-forward neural networks**
- A **recurrent network** feeds its outputs back into its own inputs. The activation levels of the network form a dynamic system that could be stable or could chaotic.

Single-Layer Feed-Forward Neural Networks

- A single-layer neural network is also called a perceptron network.
- Each neuron in a single-layer feed-forward network acts independently -- if the network has m outputs, it is equivalent to m distinct neural networks, each with one neuron and one output.
- When the activation function is a hard threshold or a logistic function, we effectively have m independent classifiers.
- We can think of this as m separate networks. This is because each weight affects only one of the outputs, leading to m separate training processes.



Note: The simple single-layer feed forward neural network cannot, for example, represent the XOR operation on two inputs. This is because the XOR operation is not separable, so the perceptron can't learn it. This does not mean that perceptrons are completely useless. In fact, they can represent some complex Boolean functions compactly; e.g. the majority function which outputs a 1 if more than half of the inputs are 1. Trying to model the same function with e.g. a decision tree would require exponentially many nodes to represent the function.

Multi-Layer Feed-Forward Neural Networks

- Given the apparent limitations of single-layer neural networks, how can we represent more complex functions? The answer is to add at least one hidden layer to the neural network, making it a **multi-layer neural network**.
- The beauty of multilayer neural networks is that with a sufficiently large hidden layer, **it is possible to represent any continuous function of the units with arbitrary accuracy**, and with two hidden layers, even more discontinuous functions can be represented.
- The proof is complex, the key point is that the number of hidden units grows exponentially with the number of inputs.
- In general, we can think of neural networks as a tool for doing non-linear regression.
- We are still trying to find a hypothesis to model the data, specifically, we think of it as finding a function $hw(x)$ parameterized by the weights w that best fits the data.
- The actual fitting of the model is performed by trying to minimize a chosen loss function using gradient descent.
- The formal process of finding weights requires back-propagation, whereby we calculate the error in the output layer (the difference between the actual values and predictions) and we propagate the error to the previous layers of the network, throughout the network. In the process we update the weights in each layer.
- This process can take a significant amount of time, highlighting one of the disadvantages of neural networks: **the time required to train them can be much longer than for other methods**.

How do we choose the optimal network structure?

- We use cross validation, which involves trying several different structures and seeing which works best.

Risk of Overfitting?

- Since neural networks have a large number of parameters, there is a potential for overfitting (fitting too complex a model, that does not generalize well).
- However, the book claims that very large networks do generalize well as long as the weights are kept small.
- There are various approaches to reduce the risk of overfitting such as starting with a fully connected network and removing connections from it in such a way that identifies the optimal selection of connections which can be dropped.

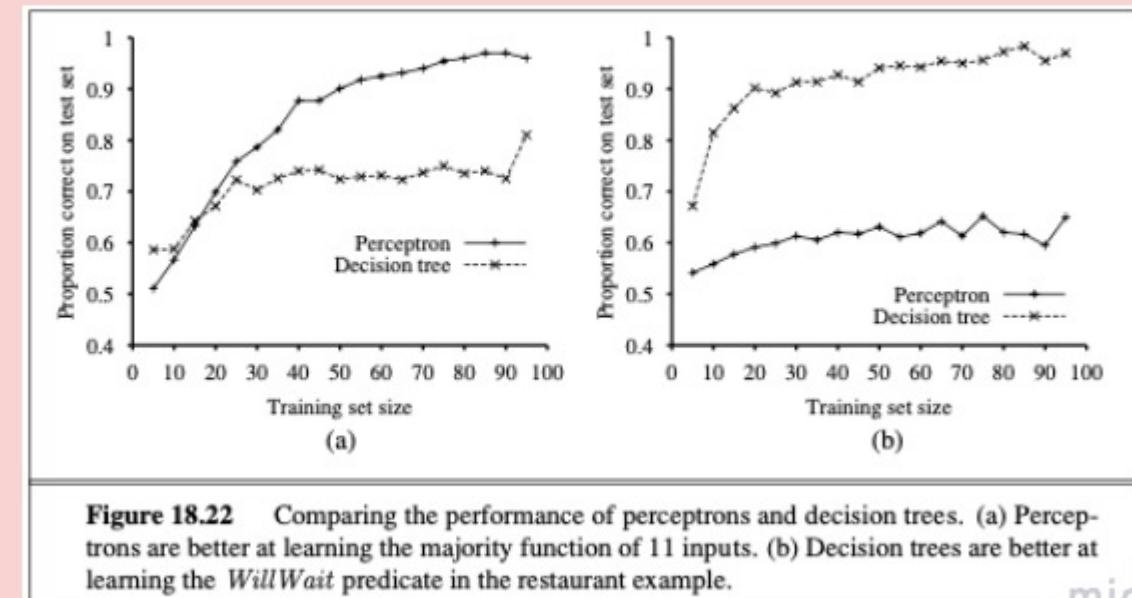
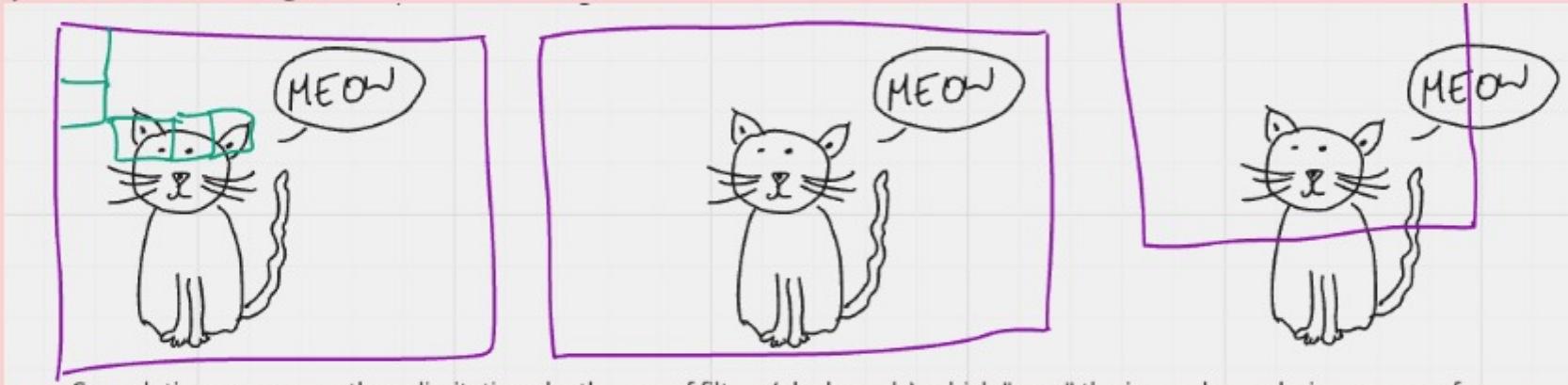


Figure 18.22 Comparing the performance of perceptrons and decision trees. (a) Perceptrons are better at learning the majority function of 11 inputs. (b) Decision trees are better at learning the *WillWait* predicate in the restaurant example.

Convolutional Neural Networks (CNNs)

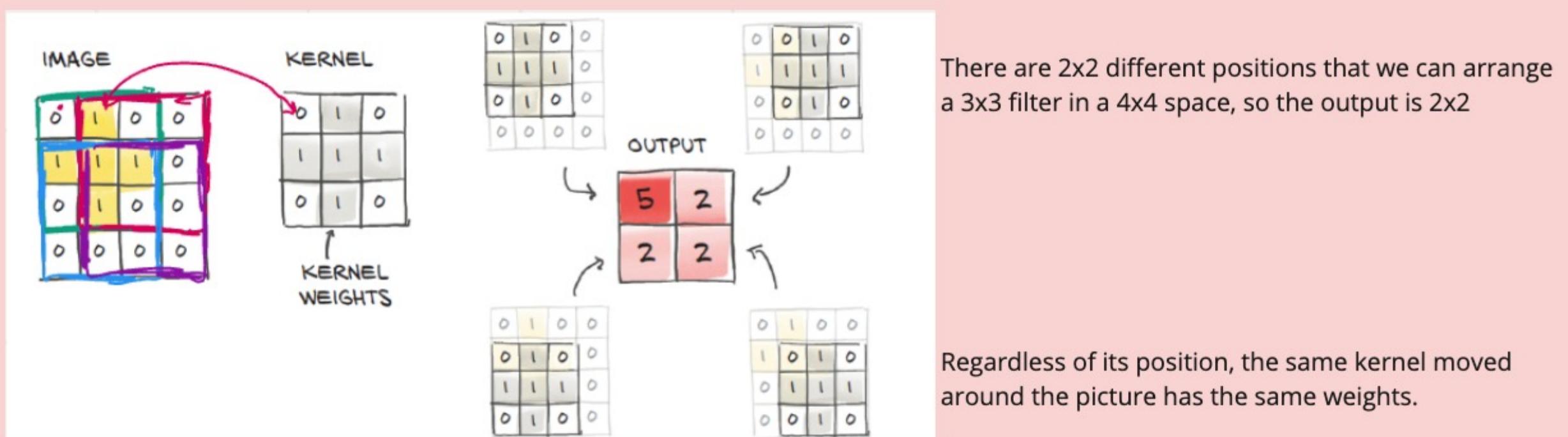
- CNNs are most commonly used in processing image data.
- With a picture that has $P \times P$ pixels and 3 RGB color channels, we could (in theory) build a network based on $P \times P \times 3$ attributes and train it as usual.
- This approach has two problems:
 - 1) The large number of weights that we would need to find, especially if each input is connected to each neuron in the hidden layer.
 - 2) The lack of provision for local patterns and relationships, independently of where the local pattern is in the picture, and what else is in the picture.
- For example: We would like our classifier to recognize a cat regardless of what the background is, and whether or not the picture has the cat at the center or in other parts of the image.



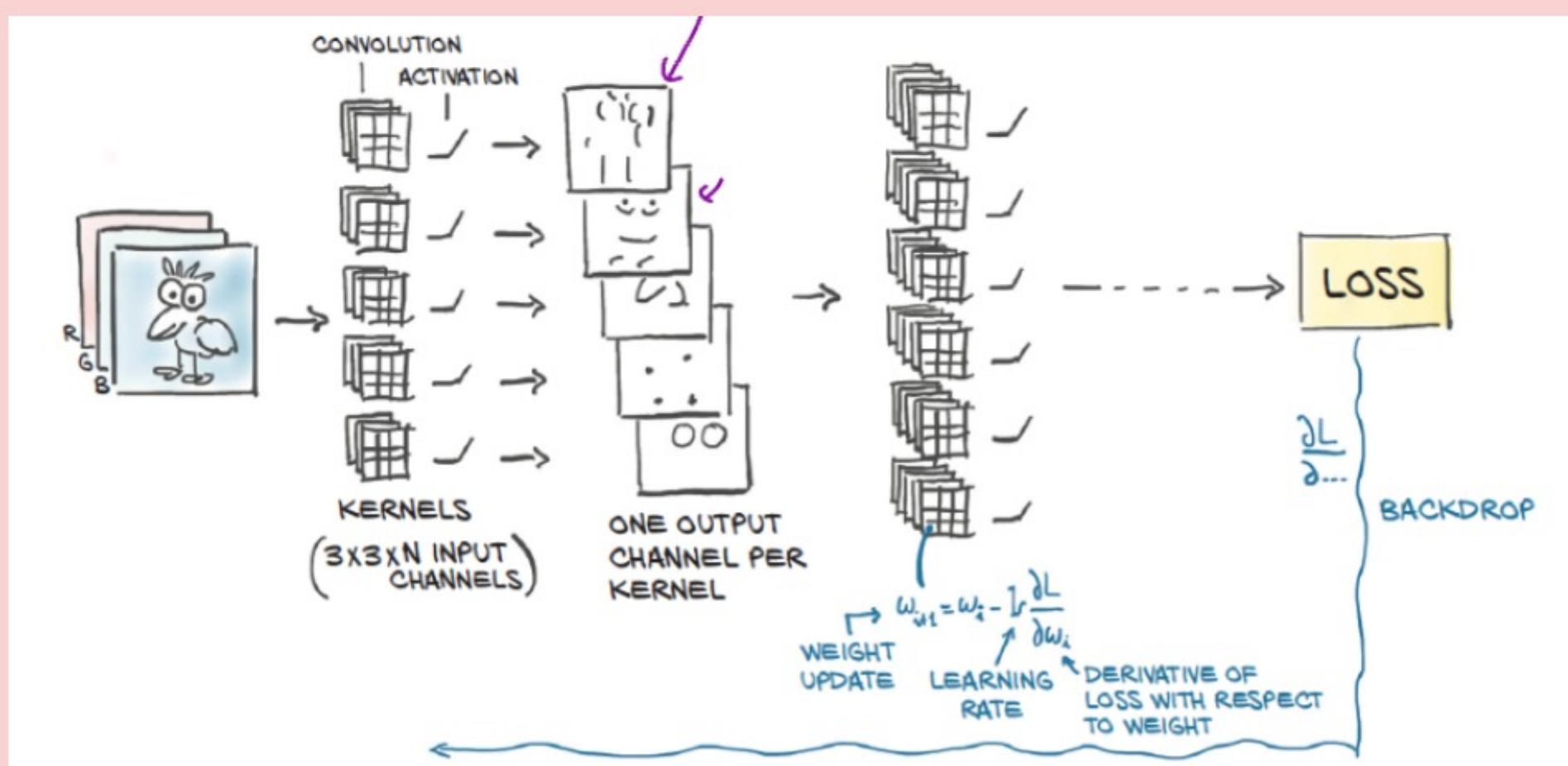
- Convolutions overcome these limitations by the use of filters (aka kernels) which scan the image by analyzing groups of nearby pixels, to detect spacial patterns (e.g. vertical lines, horizontal lines, etc.).

The benefits of using filters/kernels:

- translation invariance** - they help us recognize a pattern regardless of where it is in the picture.
- locality** - processing local pixels together, to spot local patterns.
- reduction in the number of weights** that we need to train since copies of the same filter use the same weights.



- We will normally use multiple kernels/filters. The reasoning is that each of the filters detects a different type of patterns (i.e. vertical lines, horizontal lines, etc.).
- If we have n such filters, then the 3rd dimension of the output changes from 1 to n . For example, in the above picture where the output is 2×2 for a single kernel, with n kernels the output would be $2 \times 2 \times n$.



CNN Example - Calculating the number of weights

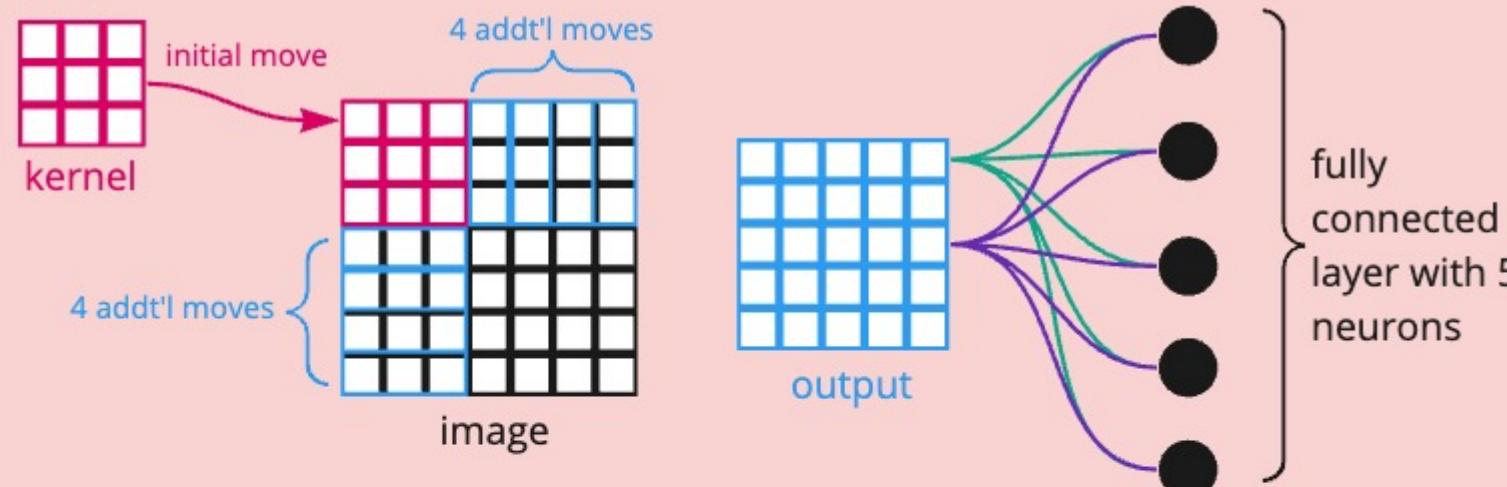
Problem: Suppose we have a $7 \times 7 \times 1$ image to start with, and a convolution layer with one $3 \times 3 \times 1$ kernel/filter. This is followed by a fully connected layer with 5 neurons. How many weights do we have in total?

Solution:

Since the weights for a given kernel are the same, regardless of where we move it within the image, the input weights for the first convolutional layer depend on the kernel size. **A $3 \times 3 \times 1$ kernel will have $3 \times 3 \times 1 = 9$ weights for the inputs + 1 bias term, for a total of 10 weights for that kernel.**

- This does not depend on the size of the input image, it only depends on the size of the kernel/filter.

However, the size of the **output** of that kernel does depend on the size of the image, as it corresponds to the number of locations in which we can place the kernel within the image. **There are $5 \times 5 \times 1$ different ways to place a $3 \times 3 \times 1$ kernel in a $7 \times 7 \times 1$ image, so the output of the convolutional layer is $5 \times 5 \times 1$.**



Each of the neurons in the fully connected layer is a function of the $5 \times 5 \times 1$ cells of the output of the convolutional layer + it also contains 1 bias term per each kernel.

So each neuron has $5 \times 5 \times 1 = 25$ inputs, +1 bias = 26 wts/neuron.

That's 26 weights/neuron * 5 neurons = 130 wts/output layer

$$\text{Total weights} = 10 \text{ weights} + 130 \text{ weights} = 140 \text{ weights}$$

convolutional layer output layer

7.6 Quiz: Linear Models and Neural Networks

Question 1	1 pts
The mathematical model of a neuron has two parts. Name the two parts.	
<input type="radio"/> A non-linear function and an activation function. <input type="radio"/> A rectifier function and a hard threshold function. <input checked="" type="radio"/> A linear function and an activation function. <input type="radio"/> A linear function and a logistic function. <input type="radio"/> A non-linear function and a hard threshold function.	

KEYWORDS: parts of neuron; mathematical model

A neuron consists of a linear function of the inputs, followed by an activation function. The activation function could be logistic, but it is not always.

Question 2	1 pts
A logistic regression function is equivalent to which type of neuron?	
<input type="radio"/> A neuron that combines a rectifier function with a logistic function. <input checked="" type="radio"/> A neuron with the logistic function as its activation function. <input type="radio"/> A neuron with the rectifier function as its activation function. <input type="radio"/> A neuron with the hard threshold function as its activation function. <input type="radio"/> A neuron with the softplus function as its activation function.	

KEYWORDS: logistic regression function

A neuron consists of a linear function of the inputs, followed by an activation function. The activation function could be logistic, but it is not always.

Question 3	1 pts
Suppose the linear part of a neuron produces the value 5. For each of the three activation functions (hard threshold, logistic, rectifier), what would be the output of the neuron? The answers are for the three activation functions in order, and are rounded to the nearest integer.	
<input type="radio"/> 5, 5, 1. <input type="radio"/> 0, 1, 1. <input checked="" type="radio"/> 1, 1, 5. <input type="radio"/> 1, 0, 1. <input type="radio"/> 0, 1, 5.	

KEYWORDS: linear part of neuron; activation functions; hard threshold; logistic; rectifier; neuron output

When given a positive number:

- hard threshold = 1
- logistic = very close to 1
- rectifier returns value given, in this case 5

Question 4	1 pts
Q4. Suppose the linear part of a neuron produces the value -4. For each of the three activation functions (hard threshold, logistic, rectifier), what would be the output of the neuron? The answers are for the three activation functions in order, and are rounded to the nearest integer.	
<input type="radio"/> 0, 0, -4. <input type="radio"/> 0, 1, -4. <input type="radio"/> 0, 1, 0. <input checked="" type="radio"/> 0, 0, 0. <input type="radio"/> 1, 0, 1.	

KEYWORDS: linear part of neuron; activation functions; hard threshold; logistic; rectifier; neuron output

When given a negative number:

- hard threshold = 0
- logistic = very close to 0
- rectifier returns 0

Question 5	1 pts
Suppose we have a multilayer neural network with 10 inputs, two fully-connected hidden layers of 10 neurons each, and one fully-connected output layer with 5 neurons. How many weights does it have in total?	
A model answer for this question will be released towards the end of the week.	
<input type="radio"/> 35 <input type="radio"/> 625 <input checked="" type="radio"/> 275 <input type="radio"/> 250	

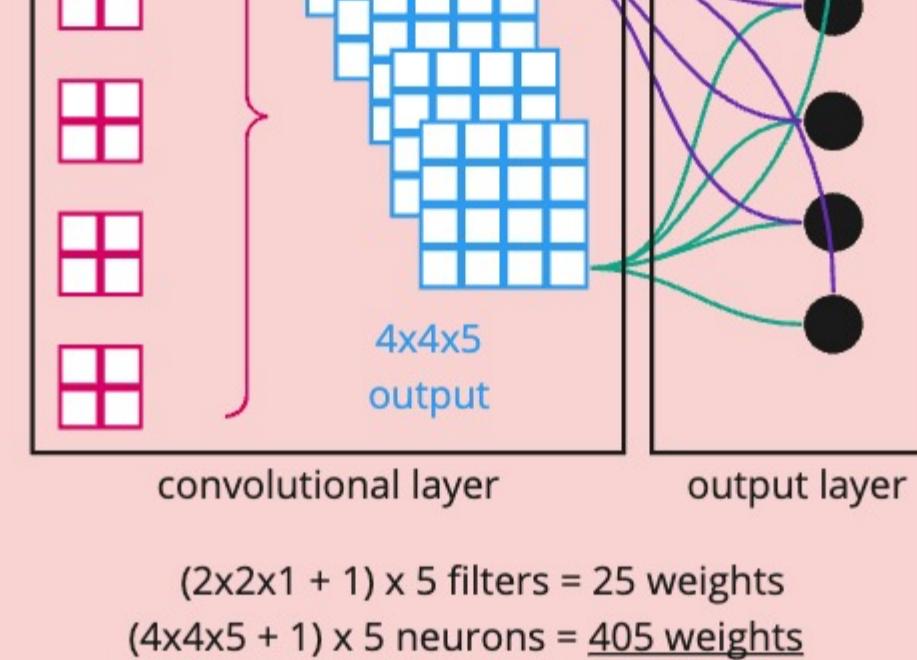
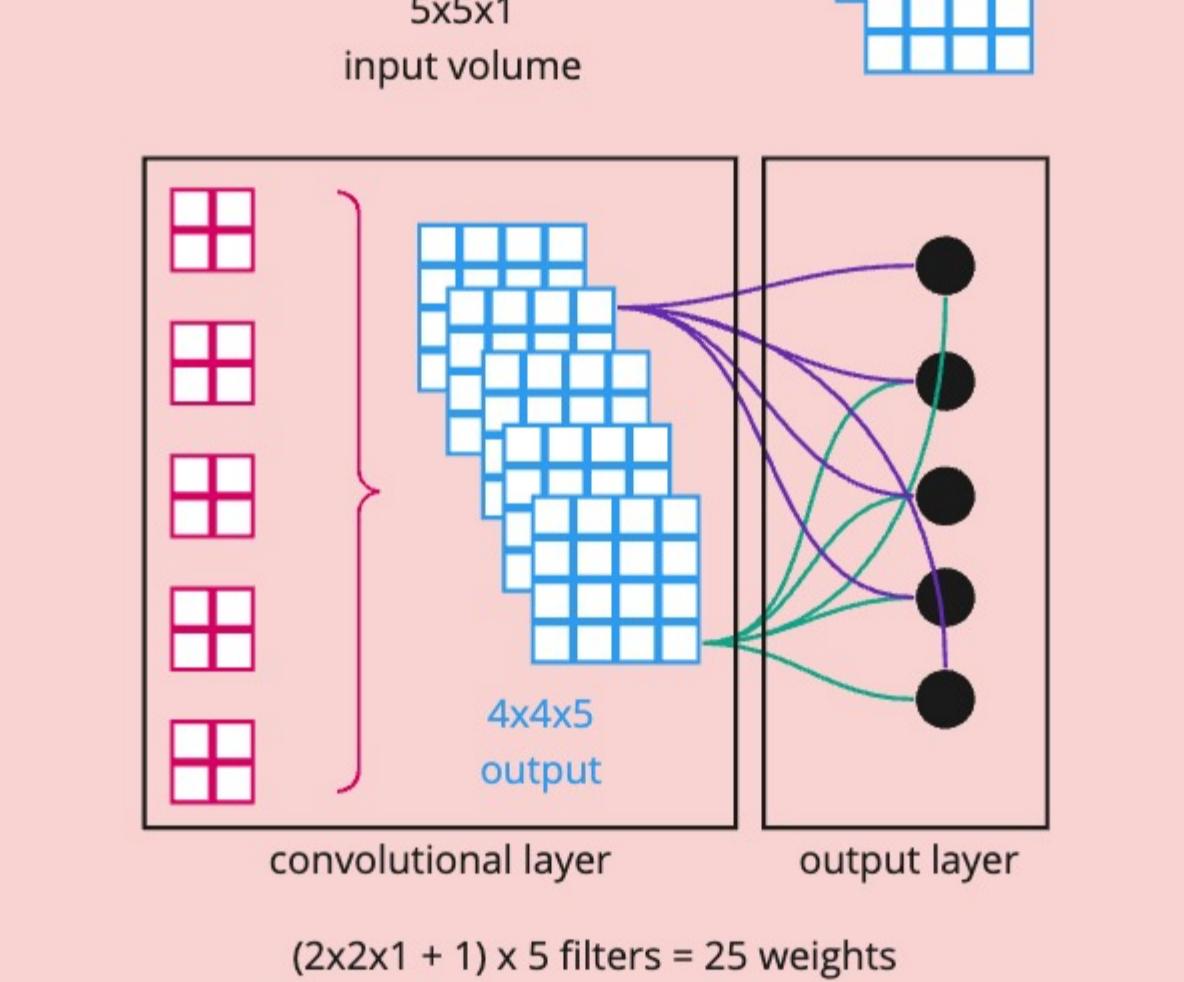
KEYWORDS: multilayer neural network; inputs; neurons; fully-connected; hidden layers; output layers; weights

1st hidden layer: $(10 + 1) \times 10 = 110$ weights

2nd hidden layer: $(10 + 1) \times 10 = 110$ weights

output layer: $(10 + 1) \times 5 = 55$ weights

total: **275 weights**



Question 6	1 pts
Suppose we have a convolutional neural network with a 5x5x1 input volume, followed by one convolutional layer with 5 filters that have a 2x2x1 receptive field, followed by one fully connected output layer with 5 neurons. How many weights does the network have in total?	
A model answer for this question will be released towards the end of the week.	
<input checked="" type="radio"/> 430 <input type="radio"/> 2500 <input type="radio"/> 2485 <input type="radio"/> 805 <input type="radio"/> 530	

KEYWORDS: convolutional neural network; input volume; convolutional layer; filters; receptive field; fully connected output layer; neurons; weights

Question 7	1 pts
Suppose we have the following convolutional neural network architecture:	
Input volume: 15x15x1 (i.e. width = 15, height = 15, and depth = 1)	
First convolutional layer: 5 filters, with receptive field 3x3x1	
Second convolutional layer: 5 filters, with receptive field 2x2x5	
Output layer: fully connected with 10 neurons.	
How many weights are there in each layer?	
First convolutional layer:	<input type="text" value="50"/>
Second convolutional layer:	<input type="text" value="105"/>
Output layer:	<input type="text" value="7210"/>
A model answer for this question will be released towards the end of the week.	

1st convolutional layer:

$(3 \times 3 \times 1 + 1) \times 5 = 50$ weights

output of 1st convolutional layer:

$3 \times 3 \times 1$ into $15 \times 15 \times 5 = 13 \times 13 \times 5$

$(13 \times 13 \times 5) \times 5$ filters = $13 \times 13 \times 5$ volume

2nd convolutional layer:

$(2 \times 2 \times 5 + 1) \times 5 = 105$ weights

output of 2nd convolutional layer:

$2 \times 2 \times 5$ into $13 \times 13 \times 5 = 12 \times 12 \times 5$

$(12 \times 12 \times 5) \times 5$ filters = $12 \times 12 \times 5$

output layer:

$(12 \times 12 \times 5 + 1) \times 10$ neurons = **7210 weights**

Total:

$50 + 105 + 7210 = 7365$ weights

Formative Assessment - Practice Exam

The following is a walk through for the formative offered to groups 1-6, **the solutions are an attempt by Ali K but are not guaranteed to be correct.** Please provide feedback on any solutions you disagree with. Should be a great discussion!

COM00143M > Assignments

ONLINE 1 - 23/24

Search for assignment

SHOW BY DATE SHOW BY TYPE

Home

Units

Discussions

Reading List

Announcements

Assignments

Panopto Recordings

Kortext

Upcoming assignments

Formative Assessment - Practice Exam (Groups 1-6) Due 4 Oct at 5:00

Artificial Intelligence and Machine Learning Exam - [Groups 1-6] [001 1.0] Due 27 Oct at 5:00

Undated assignments

Sample Paper 1 (Groups 1-6) /100 pts

Sample Paper 2 (Groups 1-6) /100 pts

Question 1

0 pts

In many practical applications of artificial intelligence, it is possible for people to be harmed or even killed by a deployed AI system. Depending on the application, there can also be financial losses or damage to property. Suppose you are designing an autonomous robot to deliver medical supplies around a hospital building. The robot will share the corridors and lifts of the building with many other users (for example patients, medical staff, and trolleys). Identify one ethical and/or legal issue for delivery robots, and reflect critically on the following questions:

1. How important is the ethical and/or legal issue that you chose? For example, could it prevent the deployment of an AI system, or cause significant harm or loss when an AI system is deployed?
2. Are there potential solutions to the ethical and/or legal issue? How plausible are the potential solutions?

There is no need to include references in your answer.

Your answer is limited to 250 words. Any part of your answer beyond the word limit will not be marked.

KEYWORDS: AI Ethical Issues; AI Legal Issues

Study Session:

Thursday, 19 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

Question 2

0 pts

A FIFO (first-in-first-out) queue is used to store the frontier set of which search algorithm?

- A* Search Priority queue
- Breadth-First Search ✓ Correct per the answer key
- Depth-First Depth-Limited Search LIFO stack
- Tree Search Can be but not required
- Graph Search Can be but not required
- Depth-First Search LIFO stack
- Iterative Deepening LIFO stack
- Greedy Best-First Search Priority queue

KEYWORDS: FIFO; LIFO; Priority Queue; Algorithms

Question 4

✓ Correct per the answer key

0 pts

Suppose you are applying the Tree Search version of Greedy Best-First Search to a search problem. Using an admissible heuristic would guarantee:

- Optimality of the first solution found. GBFS is not optimal
- Termination whenever the set of states is finite. Not complete even in finite spaces
- Consistency of the heuristic. Consistency implies admissibility, but not the other way around
- None of the above. ✓ Correct per the answer key

KEYWORDS: Greedy Best-First Search; tree search; admissible heuristic

Question 3

0 pts

Which of the following search algorithms require a priority queue to store the frontier set? Select two algorithms.

- Iterative Deepening LIFO stack
- Tree Search Can be but not required
- Depth-First Search LIFO stack
- Greedy Best-First Search ✓ Correct per the answer key
- A* Search ✓ Correct per the answer key
- Breadth-First Search FIFO queue
- Depth-First Depth-Limited Search LIFO stack
- Graph Search Can be but not required

KEYWORDS: FIFO, LIFO, Priority Queue; Algorithms

Question 5

0 pts

Give the time and space complexity of the Tree Search version of Depth-First Search (DFS), in terms of the branching factor b and the maximum depth m of the search tree.

Write your answer using big-O notation, and if you need to use powers then write them in TeX notation. For example, n squared would be written as n^2 . If the algorithm has worst-case complexity n squared, it would be written as $O(n^2)$.

Time complexity: $O(b^m)$

Space complexity: $O(bm)$

✓ Correct per the answer key

✓ Correct per the answer key

OTH
2 SE
FRO
SLID

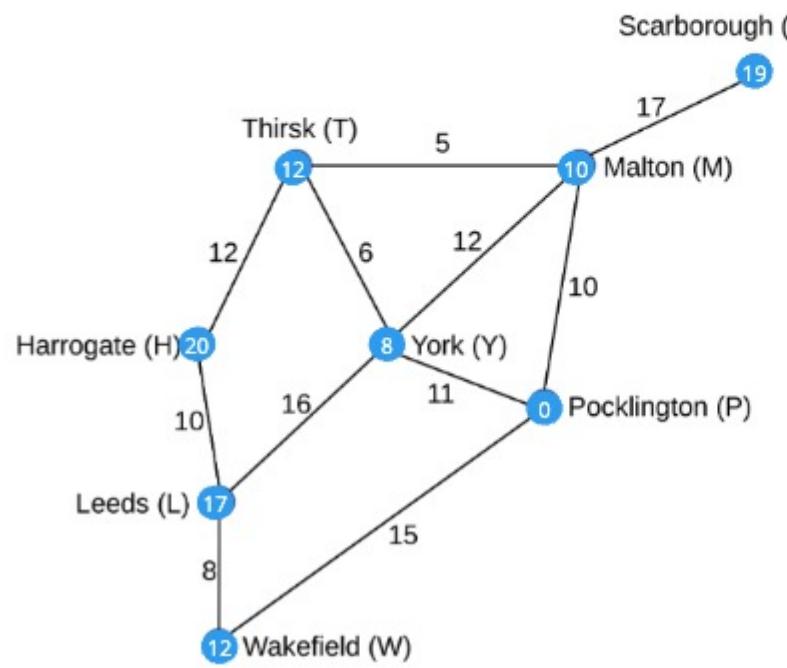
KEYWORDS: Space complexity; time complexity; depth-first search; branching factor; maximum depth

miro

Question 6

0 pts

Consider the following map where each connection between two cities is labelled with the distance by road.



City	Euclidean distance to Pocklington
Scarborough	19
Malton	10
York	8
Thirsk	12
Harrogate	20
Leeds	17
Wakefield	12

REMEMBER - These are the distances to use when looking for next node.

KEYWORDS: Greedy Best First Search; Graph Search

This question is about applying the Graph Search version of **Greedy Best-First Search (GBFS)** to find a path from Harrogate to Pocklington. One step of the algorithm consists of selecting a node from the frontier, potentially expanding it and adding any relevant child nodes to the frontier.

The initial frontier will contain only Harrogate (H). Execute GBFS for 4 steps. For each step, specify which node is chosen to be expanded, and give the new frontier as a sequence of states (represented by letters) in alphabetical order. For example, W is expanded and the new frontier set is LP.

Step 1

✓ Correct per the answer key

Node selected: New frontier in alphabetical order: **Step 2**Node selected: New frontier in alphabetical order: **Step 3**Node selected: New frontier in alphabetical order: **Step 4**Node selected: New frontier in alphabetical order:

Question 7

0 pts

What is the meaning of this symbol: \models

- Inference
- Sentence
- Knowledge base
- Entailment
- Negation

✓ Correct per the answer key

Question 8

0 pts

What is the meaning of this symbol: \neg

- Inference
- Knowledge base
- Entailment
- Negation
- Sentence

✓ Correct per the answer key

Question 9

0 pts

In propositional logic, suppose there are five proposition symbols A, B, C, D, E . How many models are there?

$$2^5 = 32$$

✓ Correct per the answer key

KEYWORDS: Propositional logic symbols

KEYWORDS: Propositional logic number of models.

B \wedge \neg D	▼
FALSE	
TRUE	
FALSE	
TRUE	
FALSE	
FALSE	
FALSE	
FALSE	
TRUE	
FALSE	
TRUE	
FALSE	
FALSE	
FALSE	

Question 10

0 pts

In propositional logic, using four proposition symbols A, B, C, D , how many models are there of the formula $B \wedge \neg D$? Or, to put it another way, how many models satisfy $B \wedge \neg D$?

4

✓ Correct per the answer key

Question 11

0 pts

In propositional logic, using four proposition symbols A, B, C, D , how many models are there of the formula $C \vee A \vee \neg C$? Or, to put it another way, how many models satisfy $C \vee A \vee \neg C$?

16

$C \vee \neg C = \text{all models}$ ✓ Correct per the answer key

Question 12

0 pts

In propositional logic, using four proposition symbols A, B, C, D , how many models are there of the formula $A \wedge B \wedge \neg A$? Or, to put it another way, how many models satisfy $A \wedge B \wedge \neg A$?

0

$A \wedge \neg A = () \text{ which is false}$ ✓ Correct per the answer key

[Truth Table Generator \(loigic.net\)](http://loigic.net)

KEYWORDS: Propositional logic number of models.



Question 13

0 pts

Suppose we have two sentences S_1, S_2 in propositional logic, and suppose $S_1 \models S_2$ and $S_2 \models S_1$. Which of the following statements are true?

1. A sound inference algorithm will infer S_1 from S_2 .
- 2. A sound inference algorithm may infer S_1 from S_2 .**
3. A sound inference algorithm will not infer S_1 from S_2 .
4. A sound inference algorithm will infer S_2 from S_1 .
- 5. A sound inference algorithm may infer S_2 from S_1 .**
6. A sound inference algorithm will not infer S_2 from S_1 .

A **sound** inference algorithm is always correct but it is not always complete.
Therefore *may* is appropriate here.
Will would be for complete.

- 1 and 6.
- Only 1.

 2 and 5. ✓ Correct per the answer key

- 3 and 6.
- 3 and 4.
- 3 and 5.
- Only 2.
- 2 and 6.
- 1 and 5.
- 2 and 4.
- Only 6.
- Only 5.
- 1 and 4.
- Only 4.
- Only 3.

KEYWORDS: propositional logic; sentences; sound inference algorithm

Question 14

0 pts

Consider the following logical statement:

If the battery is OK, and the screen is not cracked, then the phone works.

Suppose we have the following three proposition symbols:

B: The battery is OK.

S: The screen is cracked.

W: The phone works.

Write the logical statement in propositional logic using the three proposition symbols B, S, and W. It should be written as a clause in conjunctive normal form. Use a minus sign for negation (for example, $\neg A$ means the negation of A). Write disjunction ('or') using the letter 'v'. For example, the clause $\neg B \vee \neg S$ would be written as $\neg B \vee S$.

$\neg B \vee S \vee W$

✓ Correct per the answer key

KEYWORDS: CNF; propositional logic; Conjunctive Normal Form;

$$B \wedge \neg S \Rightarrow W$$

$$\neg(B \wedge \neg S) \vee W \quad \text{implication elimination}$$

$$\neg B \vee \neg(\neg S) \vee W \quad \text{De Morgan's Law}$$

$$\neg B \vee S \vee W \quad \text{double negation elimination}$$

Question 15

0 pts

Consider the following three sentences in propositional logic.

1. $A \vee B$
2. $A \vee \neg B$
3. $A \vee B \vee \neg C$

Apply one step of the WalkSAT algorithm with the following assumptions:

- The probability p of making a random walk move is set to 0.
- The starting assignment is $A=\text{false}$, $B=\text{false}$, $C=\text{true}$.
- The algorithm chooses clause 3.

For each proposition symbol in clause 3, how many clauses would be satisfied in total if the symbol were flipped?

A: ✓ Correct per the answer key

B: ✓ Correct per the answer key

C: ✓ Correct per the answer key

Which proposition symbol is flipped? ✓ Correct per the answer key

Model: A = False, B = False, C = True

- Evaluate clauses:

- Satisfied Clauses: 2
- Unsatisfied Clauses: 1, 3

A	$\neg T$	B	$\neg T$	C	$\neg T$	$A \vee B$	\neg	$A \vee \neg B$	\neg	$A \vee B \vee \neg C$	\neg
FALSE		FALSE		TRUE		FALSE		TRUE		FALSE	

- Pick a clause at random from unsatisfied clauses:

- Given: Algorithm chooses 3: $A \vee B \vee \neg C$

- Random or Greedy:

- **Probability of random walk = 0 => event will not happen. This algorithm is ALWAYS greedy.**

- Greedy Choice:

- If we flip $A = \text{True}$, then 1, 2, & 3 are satisfied

A	$\neg T$	B	$\neg T$	C	$\neg T$	$A \vee B$	\neg	$A \vee \neg B$	\neg	$A \vee B \vee \neg C$	\neg
TRUE		FALSE		TRUE		TRUE		TRUE		TRUE	

- If we flip $B = \text{True}$, then 1 & 3 is satisfied

A	$\neg T$	B	$\neg T$	C	$\neg T$	$A \vee B$	\neg	$A \vee \neg B$	\neg	$A \vee B \vee \neg C$	\neg
FALSE		TRUE		TRUE		TRUE		FALSE		TRUE	

- If we flip $C = \text{False}$, then 2 & 3 is satisfied

A	$\neg T$	B	$\neg T$	C	$\neg T$	$A \vee B$	\neg	$A \vee \neg B$	\neg	$A \vee B \vee \neg C$	\neg
FALSE		FALSE		FALSE		FALSE		TRUE		TRUE	

- Flip selected symbol

- A = True

KEYWORDS: WALKSAT; propositional logic

START STATE

A - FALSE

B - FALSE

C - TRUE

A FLIPPED

A - TRUE

B - FALSE

C - TRUE

1 SATISFIED

2 SATISFIED

3 SATISFIED

TOTAL 3

B FLIPPED

A - FALSE

B - TRUE

C - TRUE

1 SATISFIED

2 NOT SATISFIED

3 SATISFIED

TOTAL 2

C FLIPPED

A - FALSE

B - FALSE

C - FALSE

1 NOT SATISFIED

2 SATISFIED

3 SATISFIED

TOTAL 2

'A' IS THE PROPOSITION THAT IS FLIPPED AS IT MAXIMISES THE NUMBER OF SATISFIED CLAUSES.

The following represent questions included in Paper 1 that were not included in the Formative

Question 16	✓ Correct per the answer key	2 pts
The mathematical model of a neuron has two parts. Name the two parts.		
<ul style="list-style-type: none"><input type="radio"/> An arbitrary non-linear function and an activation function.<input type="radio"/> A rectifier function and a hard threshold function.<input type="radio"/> An arbitrary non-linear function and a logistic function.<input checked="" type="radio"/> A linear function and an activation function.<input type="radio"/> A linear function and a logistic function.<input type="radio"/> An arbitrary non-linear function and a hard threshold function.		

KEYWORDS: parts of a mathematical model of a neuron

The Input Layer with Weights is a Linear Function. The other part is the Activation Function as shown below.

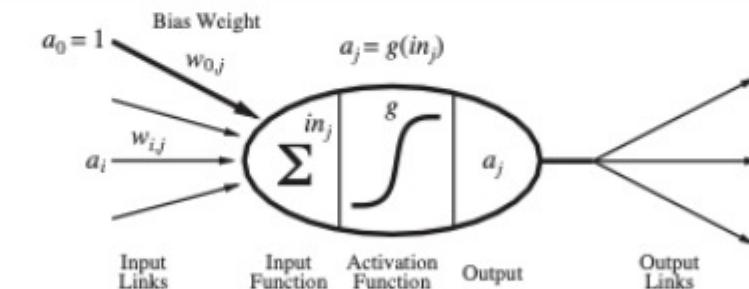


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^n w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

KEYWORDS: types of machine learning; machine learning types; supervised learning; unsupervised learning; reinforcement learning; deep learning

Question 17	✓ Correct per the answer key	2 pts
Suppose you have a data set containing examples with multiple attributes and no labels. Which type of machine learning would be appropriate?		
<ul style="list-style-type: none"><input type="radio"/> Supervised learning.<input checked="" type="radio"/> Unsupervised learning.<input type="radio"/> Reinforcement learning.<input type="radio"/> Deep learning.		

SUPERVISED: For labelled data; uses learning algorithms for prediction and classification or regression. Optimises decision making.

UNSUPERVISED: No need for labels; can identify inherent patterns; can reduce dimensionality (reducing random variables whilst preserving essential features); anomaly / outlier detection; adaptable to various data types

REINFORCEMENT: Has a decision making agent and a reward function; useful for self drive vehicles; learns by taking actions and receiving feedback

DEEP: Deep neural networks can learn hierarchical representations of data; examples of use include natural language processing and speech recognition.

Question 18**✓ Correct per the answer key**

3 pts

Given a set of unlabelled examples, the k-means clustering algorithm:

- Finds the best possible cluster centres: minimising the total Euclidean distance from each example to its nearest cluster centre.
- Finds the best possible cluster centres: minimising the total Manhattan distance from each example to its nearest cluster centre.
- Finds a set of cluster centres that may not be the best possible, but cannot be improved by the k-means update step.
- Finds a set of cluster centres that are located where the examples are most dense.
- Finds the best possible cluster centres: dividing the set of examples into clusters of equal size (or as close to equal as possible).

Question 19**✓ Correct per the answer key**

3 pts

When using a multilayer neural network to predict the probability that an example belongs to a class, which activation function would be suitable for the final layer of the neural network?

- The hard threshold function.
- The logistic function.
- The rectifier function.

KEYWORDS: k-means clustering algorithm; unlabelled examples

KEYWORDS: multilayer neural network; probability; activation function; final layer; hard threshold function; logistic function; rectifier function

The **hard threshold function** acts as a binary switch, turning on or off based on the sign of the input. Returns 1 if it is ≥ 0 or 0 if it is negative.

The **logistic function** squashes its input to the range $[0, 1]$, which can be interpreted as a probability score. It's often used in binary classification problems because it models the probability that a given input point belongs to the positive class.

The **rectifier function** will return the positive input value or 0 if the input value is negative.

Question 20

8 pts

Suppose you are given the following set of five unlabelled examples, each with two attributes x and y:

x	y
3	1
4	3
4	4
5	3
5	4

Data points		Distance From		Allocation
x	y	Cluster 1	Cluster 2	
3	1	1.00	2.00	Cluster A
4	3	2.83	1.00	Cluster B
4	4	3.61	1.41	Cluster B
5	3	3.61	2.00	Cluster B
5	4	4.24	2.24	Cluster B

Execute the first update step of the k-means clustering algorithm with $k=2$. The initial cluster centres (named A and B) are given in the table below.

Cluster centre	x	y
A	2	1
B	3	3

Cluster Centers	
x	y
2	1
3	1
3	3
4.5	3.5

Cluster A Original
Cluster A New
Cluster B Original
Cluster B New

Enter the updated x and y positions of cluster centres A and B in the boxes below.

Cluster A

x:

✓ Correct per the answer key

y:

✓ Correct per the answer key

Cluster B

x:

✓ Correct per the answer key

y:

✓ Correct per the answer key

KEYWORDS: k-means clustering algorithm



Question 21

9 pts

Suppose you have a convolutional neural network with an $8 \times 8 \times 3$ input volume representing a colour image. The first layer is a convolutional layer consisting of 10 filters that have a $3 \times 3 \times 3$ receptive field. Calculate the number of neurons and the total number of weights in the first layer.

Neurons: **360**

✓ Correct per the answer key

Weights: **280**

✓ Correct per the answer key

The output of the first layer is a volume. Give its dimensions.

Output volume dimensions: **$6 \times 6 \times 10$**

✓ Correct per the answer key

KEYWORDS: convolutional neural network (CNN); filters; receptive field; neurons; weights; first layer

$3 \times 3 \times 3$ filter has 27 weights + 1 bias = 28 weights/filter
 $\times 10$ filters = 280 weights

$3 \times 3 \times 3$ filter on $8 \times 8 \times 3$ input = $6 \times 6 \times 1$ output layer $\times 10$.
= $6 \times 6 \times 10$ output layer volume???

In the context of a convolutional layer with no padding and a stride of 1, the formula to calculate the output width (W_{out}) given the input width (W_{in}) and filter width (F) is:

$$W_{\text{out}} = W_{\text{in}} - F + 1$$

CHAT GPT EXPLANATION

In your case:

- Input width (W_{in}) is 8 (from the input volume).
- Filter width (F) is 3 (from the $3 \times 3 \times 3$ filter).

Plugging these values into the formula:

$$W_{\text{out}} = 8 - 3 + 1 = 6$$

So, with an input width of 8, a filter width of 3, no padding, and a stride of 1, the output width (W_{out}) after the convolutional operation is 6.

Similarly, the output height (H_{out}) would also be 6. Therefore, the output volume is $6 \times 6 \times 10$, resulting in 360 neurons in total.

Question 22

3 pts

Suppose we have a convolutional neural network. The first layer is a convolutional layer, and it has an output volume with dimensions $8 \times 8 \times 8$. Suppose the second layer is a fully connected layer with 5 neurons. How many weights does the second layer have?

2565

✓ Correct per the answer key

KEYWORDS: convolutional neural network; weights

each neuron has $8 \times 8 \times 8 = 512$ inputs + 1 bias = 513 weights

513 weights $\times 5$ neurons = 2565 weights/output layer

Sample Paper 2 Walk Through

In Canvas go to Assignments and under "Undated Assignments" Sample Paper 2 should be available to take. Give it a try before the Week 4 review.

The screenshot shows the 'Assignments' page in Canvas. The left sidebar includes links for Home, Units, Discussions, Reading List, Announcements, Assignments (which is selected), and Panopto Recordings. The main content area has tabs for 'SHOW BY DATE' and 'SHOW BY TYPE'. Under 'Upcoming assignments', there are two entries: 'Formative Assessment - Practice Exam (Groups 1-6)' due on Oct 4 at 5:00, and 'Artificial Intelligence and Machine Learning Exam - (Groups 1-6) [001 1.0]' due on Oct 27 at 5:00. Below this, under 'Undated assignments', there are two entries: 'Sample Paper 1 (Groups 1-6) /100 pts' and 'Sample Paper 2 (Groups 1-6) /100 pts'. The 'Sample Paper 2' entry is circled in green.

Assignment	Due Date	Points
Formative Assessment - Practice Exam (Groups 1-6)	Due 4 Oct at 5:00	
Artificial Intelligence and Machine Learning Exam - (Groups 1-6) [001 1.0]	Due 27 Oct at 5:00	
Sample Paper 1 (Groups 1-6)		/100 pts
Sample Paper 2 (Groups 1-6)		/100 pts

Study Session:

Thursday, 19 Oct 2023, 19:30 UK time

Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

The following is a walk through for the Paper 2 offered to groups 1-6, **the solutions are an attempt by Ali K but are not guaranteed to be correct**. Please provide feedback on any solutions you disagree with. Should be a great discussion!

KEYWORDS: AI Ethical Issues; AI Legal Issues

Question 1	10 pts
<p>In many practical applications of artificial intelligence, it is possible for people to be harmed or even killed by a deployed AI system. Depending on the application, there can also be financial losses or damage to property.</p> <p>Suppose you are implementing a system to approve or deny small loans using a machine learning model. It will be based on a supervised machine learning model, and trained on past approval decisions made by experienced staff. The system will take the customer's credit history, postal code, and salary as input.</p> <p>Identify one ethical and/or legal issue in this scenario, and reflect critically on the following questions:</p> <ol style="list-style-type: none">1. How important is the ethical and/or legal issue that you chose? For example, could it prevent the deployment of an AI system, or cause significant harm or loss when an AI system is deployed?2. Are there potential solutions to the ethical and/or legal issue? How plausible are the potential solutions? <p>There is no need to include references in your answer.</p> <p>Your answer is limited to 250 words. Any part of your answer beyond the word limit will not be marked.</p>	

DISCUSSION POINT: WHAT ARE SOME COMMON ETHICAL / LEGAL ISSUES THAT COULD BE DISCUSSED?

Question 2

3 pts

KEYWORDS: recognising algorithms; identifying algorithms

Consider the following description of a general search algorithm.

1. **function XXXXX-Search(*problem*) returns** a solution, or failure
2. initialize the frontier using the initial state of *problem*
3. initialize the explored set to be empty
4. **loop do**
5. **if** the frontier is empty **then return** failure
6. choose a leaf node and remove it from the frontier
7. **if** the node contains a goal state **then return** the corresponding solution
8. add the node to the explored set
9. expand the chosen node, adding the resulting nodes to the frontier, only if not in the frontier or explored set

Identify this algorithm.

- Depth-First Search
- A* Search
- Greedy Best-First Search
- Graph Search
- Breadth-First Search
- Depth-First Depth-Limited Search
- Tree Search
- Iterative Deepening

It is the Generic Graph Search because it has both the frontier and explored sets, and does not specify any particular way of selecting nodes from the frontier.

DISCUSSION POINT: HOW COULD WE RECOGNISE THE OTHER ALGORITHMS?

Question 3

6 pts

A LIFO (last-in-first-out) stack is used to store the frontier set of which search algorithms? Select three of the following answers.

Depth-First Search

A* Search

Priority queue

Greedy Best-First Search

Priority queue

Graph Search

Can be but not required

Breadth-First Search

FIFO queue

Depth-First Depth-Limited Search

Tree Search

Can be but not required

Iterative Deepening

KEYWORDS: FIFO, LIFO, Priority Queue; Algorithms

Question 4

3 pts

Suppose you were applying the Graph Search version of A* Search to a search problem. Using an **admissible** heuristic would guarantee that:

- All nodes expanded by the search algorithm will have the same value of $f(n)$. **Not with A***
- The first solution found is an optimal solution. **Only optimal if the heuristic is consistent.**
- The heuristic is also consistent. **Consistency implies admissibility, but not the other way around**
- None of the above.

Question 5

3 pts

Write the evaluation function of A* Search in terms of the path cost $g(n)$ and the heuristic $h(n)$.

$$f(n) = g(n) + h(n)$$

KEYWORDS: A* Search; admissible heuristic

KEYWORDS: A* Search evaluation function; path cost; heuristic

Question 6

18 pts

The Port of Bridlington is upgrading its infrastructure with the latest autonomous crane. The crane can be positioned over land or over a ship. Also, the crane can be holding the container or not holding the container. The set of states is shown in the table below, and the value of the heuristic function is given for each state.

State	Crane position	Holding container?	Container position	Heuristic value $h(n)$
A	Land	No	Land	0
B	Land	No	Ship	20
C	Land	Yes	Land	3
D	Ship	No	Land	5
E	Ship	No	Ship	16
F	Ship	Yes	Ship	9

The task is to offload a container from a ship while minimising the amount of energy that is used. The actions are:

- Pick up the container, with a cost of 10.
- Put down the container, with a cost of 3.
- Move the crane from land to ship or from ship to land. Moving the crane while not holding anything has a cost of 3. Moving the crane while holding the container has a cost of 7.

The task is to move the container from the ship onto land. The starting state is B, and the goal state is A.

Apply the **Tree Search version of A* Search** to solve this problem. One step of the algorithm consists of selecting a node from the frontier, potentially expanding it and adding any relevant child nodes to the frontier.

The initial frontier set will contain only B. Execute A* Search for 3 steps. For each step, specify which node is chosen to be expanded, and give the new frontier set as a sequence of states in alphabetical order. For example, C is expanded and the new frontier set is AEF.

Step 1

Node selected: | **B**

New frontier set in alphabetical order: **E**

Step 2

Node selected: **E**

New frontier set in alphabetical order: **BF**

Step 3

Node selected: **F**

New frontier set in alphabetical order: **BCE**

N.B We do not reach the goal state in the 3 moves.

KEYWORDS: A* Tree Search; Tree Search Version of A*

Working it out...

	CRANE	HOLD	CONTAINER	HEURISTIC	PICK UP	PUT DOWN	MOVE CRANE	POSSIBLE NEXT MOVES	FRONTIER	STEP
A [GOAL]	L	N	L	0	10	N/A	3			
B [START]	L	N	S	20	10	N/A	3	E	E	1
C	L	Y	L	3	N/A	3	7			
D	S	N	L	5	10	N/A	3			
E	S	N	S	16	10	N/A	3	B,F	B,F	2
F	S	Y	S	9	N/A	3	7	C,E	B,C,E	3

IMPORTANT

- As this is TREE SEARCH - remember to include the backwards steps in the Possible Next Moves. If it was GRAPH SEARCH we would keep track of these explored nodes so they would not need to be included.
- Also remember to keep any unexplored nodes in the Frontier until they are explored.

Question 7

2 pts

What is the meaning of this symbol: \vdash

- Inference
- Negation \neg
- Entailment \models
- Knowledge base KB
- Sentence s

KEYWORDS: Propositional logic symbols.

[List of logic symbols - Wikipedia](#)

SOME COMMON SYMBOLS:

\wedge (*conjunction*),
 \vee (*disjunction*),
 \neg (*negation*),
 \rightarrow (*implication*),
 \leftrightarrow (*biconditional*)
 \models (*entailment*)

Question 8

2 pts

In propositional logic, suppose there are four proposition symbols A, B, C, D . How many models are there?

$$2^4 = 16$$

KEYWORDS: Propositional logic number of models.

FORMULA:

2^n where n is the number of symbols

[Truth Table Generator \(loighic.net\)](#)

Question 9

10 pts

Consider the following four sentences in propositional logic.

1. $A \vee B$
2. $A \vee \neg B \vee C$
3. $\neg A \vee B$
4. $\neg A \vee \neg B \vee C$

Suppose you are using the WalkSAT algorithm with the following assumptions:

- The probability p of making a random walk move is set to 0.5
- The starting assignment is $A=\text{false}$, $B=\text{false}$, $C=\text{false}$.

Which of the following assignments can be reached by executing **one step** of WalkSAT? Select all the assignments that can be reached.

A=false, B=false, C=false

A=false, B=false, C=true

A=false, B=true, C=false

A=false, B=true, C=true

A=true, B=false, C=false

A=true, B=false, C=true

A=true, B=true, C=false

A=true, B=true, C=true

Model: $A = \text{False}$, $B = \text{False}$, $C = \text{False}$

- Evaluate clauses:
 - Satisfied Clauses: 2, 3, & 4
 - Unsatisfied Clauses: 1
- Pick a clause at random from unsatisfied clauses:
 - 1: $A \vee B$
- Random or Greedy:
 - Probability of random walk = 0.5
 - Greedy Choice:
 - If we flip $A = \text{True}$, then 1, 2, & 4 are satisfied
 - If we flip $B = \text{True}$, then 1, 2, & 4 are satisfied
 - If we flip $C = \text{True}$, then 2, 3, & 4 are satisfied
- Could flip any of the symbols giving us:
 - $A = \text{True}$, $B = \text{False}$, $C = \text{False}$ (A flipped)
 - $A = \text{False}$, $B = \text{True}$, $C = \text{False}$ (B flipped)
 - $A = \text{False}$, $B = \text{False}$, $C = \text{True}$ (C flipped)

KEYWORDS: WALKSAT; propositional logic

Question 10

6 pts

Consider the following logical statement:

If the battery is bad, and the screen is cracked, then the laptop is broken.

Suppose we have the following three proposition symbols:

B: The battery is OK.

S: The screen is cracked.

L: The laptop is OK.

Write the logical statement in propositional logic using the three proposition symbols B, S, and L. It should be written as a clause in conjunctive normal form. Use a minus sign for negation (for example, $\neg A$ means the negation of A). Write disjunction ('or') using the letter 'v'. For example, the clause $\neg B \vee \neg S$ would be written as $\neg B \vee S$.

B $\vee \neg S \vee \neg L$

$$\neg B \wedge S \Rightarrow \neg L$$

$\neg(\neg B \wedge S) \vee \neg L$ implication elimination

$\neg(\neg B) \vee \neg S \vee \neg L$ De Morgan's Law

$B \vee \neg S \vee \neg L$ double negation elimination

KEYWORDS: CNF; Conjunctive Normal Form; Implication Elimination; De Morgan's Laws; Double Negation Elimination

Implication Elimination
turning *implication* into *or*

$$\begin{array}{c} p \Rightarrow q \\ \hline \neg p \vee q \end{array}$$

if p implies q
 \therefore
either p or q is true

De Morgan's Laws
turning *and* into *or*

$$\begin{array}{c} \neg(p \wedge q) \\ \hline \neg p \vee \neg q \end{array}$$
$$\begin{array}{c} \neg(p \vee q) \\ \hline \neg p \wedge \neg q \end{array}$$

Double Negation Elimination
the removal of *not*

$$\begin{array}{c} \neg(\neg p) \\ \hline p \end{array}$$

if p is not false
 \therefore
 p is true

Question 11

Consider the following five sentences in propositional logic.

1. $A \vee B$
2. $A \vee \neg B \vee C$
3. $\neg A \vee B$
4. $\neg A \vee \neg B \vee C$
5. $\neg B \vee \neg C$

Apply two steps of the DPLL algorithm with the following assumptions:

- Proposition symbols are assigned in the order A, B, C.
- When assigning a proposition symbol, the value *true* is assigned before *false*.
- When performing unit propagation, the clauses are always processed in the order given above.

For each step of DPLL, you will need to give the assignment made by DPLL followed by the clauses that are still present. If there are no clauses left, write *fail*. If there are any, write your answer in the same format as this example:

Step 1: B=true, C=false, A=true, fail

Use the word *fail* to indicate that DPLL discovered a clause that is false in the current model, i.e. all clauses are satisfied.

Step 1: A=True, B=True, C=True, fail

Step 2: A=False, B=True, C=True, fail

These answers match the sample solutions provided.

See week 5 for full walkthrough

KEYWORDS: Propositional logic; DPLL algorithm

Step 1:

- Initial settings: $m = \{\}$, $p = \{A, B, C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? No.
- Assignment: A = True (given in problem)
- Consequence:

- $\checkmark e1. (\text{True} \vee B) = \text{true}$
- $\checkmark e2. (\text{True} \vee \neg B \vee C) = \text{true}$
- c3. ($\neg A \vee B$)
- c4. ($\neg A \vee \neg B \vee C$)
- c5. ($\neg B \vee \neg C$)

- $m = \{A=\text{True}\}$, $p = \{B, C\}$
- $c = \{B, \neg B \vee C, \neg B \vee \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? Yes, B is a unit clause.
- Assignment: Therefore B = True.

Consequence:

- $\checkmark e3. (\text{True}) = \text{true}$
- c4. ($\neg B \vee C$)
- c5. ($\neg B \vee \neg C$)

- $m = \{A=\text{True}, B=\text{True}\}$ $p = \{C\}$ $c = \{C, \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? Yes, C is a unit clause.
- Assignment: Therefore C = True
- Consequence: Fail

Step 2:

- Backtrack to last given/guess: A
- Assignment: A = False
- Consequence:
 - c1. ($\neg A \vee B$)
 - c2. ($\neg A \vee \neg B \vee C$)
 - $\checkmark e3. (\text{True} \vee B) = \text{true}$
 - $\checkmark e4. (\text{True} \vee \neg B \vee C) = \text{true}$
 - c5. ($\neg B \vee \neg C$)
- $m = \{A=\text{False}\}$,
- $p = \{B, C\}$
- $c = \{B, \neg B \vee C, \neg B \vee \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? Yes, B is a unit clause.
- Assignment: Therefore B = True.
- Consequence:
 - $\checkmark e1. (\text{True}) = \text{true}$
 - c2. ($\neg B \vee C$)
 - c5. ($\neg B \vee \neg C$)
- $m = \{A=\text{False}, B=\text{True}\}$ $p = \{C\}$ $c = \{C, \neg C\}$
- Can you terminate early? No.
- Any pure symbols? No.
- Any unit clauses? Yes, C is a unit clause.
- Assignment: Therefore C = True
- Consequence: Fail

Question 12

2 pts

A logistic regression function is equivalent to which type of neuron?

- A neuron that combines a rectifier function with a logistic function.
- A neuron with the logistic function as its activation function.
- A neuron with the rectifier function as its activation function.
- A neuron with the hard threshold function as its activation function.
- A neuron with the softplus function as its activation function.

KEYWORDS: Logistic regression function; neurons; rectifier; hard threshold; softplus function.

See Question 13 for details of the others
(except softplus!)

Question 13

4 pts

Suppose the linear part of a neuron produces the value 6. For each of three activation functions (hard threshold, logistic, rectifier), what is the output of the neuron? The answers are for the three activation functions (hard threshold, logistic, rectifier) in that order, and are rounded to the nearest integer.

- 6, 6, 1. Hard Threshold will produce an output of 0 or 1, so this can be discounted.
- 1, 6, 1. Logistic will produce an output of between 0 and 1, so this can be discounted.
- 0, 1, 1. Hard Threshold will produce an output of 1 for positive value 6, so this can be discounted.
- 1, 0, 1. Logistic produces result of 1 (as shown on the right), so this can be discounted.
- 1, 1, 6. CORRECT ANSWER
- 0, 1, 6. Hard Threshold will produce an output of 1 for positive value 6, so this can be discounted.
- 1, 1, 1. The Rectifier output does not match the input of 6, so this can be discounted.

Question 14

2 pts

Reinforcement learning systems learn by:

- Clustering similar examples in a set of unlabelled examples.
- Passively observing their environment and generalising from the set of observations.
- Taking actions in their environment and learning from the good or bad outcomes of those actions.
- Generalising from a set of labelled examples to construct a function that maps from an example to a prediction of the label.

KEYWORDS: Hard threshold; logistic; rectifier; neurons; activation function; neuron output.

HARD THRESHOLD - SEE Q.15 FOR MORE DETAILS.

LOGISTIC FORMULA:

$$\text{Output} = \frac{1}{1+e^{-x}}$$

Substituting X for 6. the result is 0.9975 so rounded to nearest integer, this is 1.

In the **RECTIFIER** (or rectified linear unit (ReLU)) activation function, the output is equal to the input if the input is positive, and zero otherwise.

N.B I believe 0 outputs should be treated as positive.

KEYWORDS: Reinforcement learning systems.

These two answers match the sample solutions provided.

Question 15

12 pts

In the data set below there are four examples with two attributes (x and y), and each example is labelled 0 or 1.

x	y	Label
1	1	0
1	2	0
2	1	0
2	2	1

FULL CALCULATION TABLE:

LINE	X	Y	LABEL	A ($X + Y - 4.5$)	B ($X + Y - 3.5$)	C ($-X - Y + 3.5$)	D ($X + Y - 2.5$)
[1]	1	1	0	0*	0	1	0
[2]	1	2	0	0	0*	1	0
[3]	2	1	0	0	0	1*	1
[4]	2	2	1	0	1	1	1*
MATCHES LABEL FOR ALL LINES	N/A	N/A	N/A	N	Y	N	N

Professor Fuzzbrain wants to predict the label from x and y using a single neuron with a hard threshold activation function. He has made four attempts at this task, and they are shown in hypotheses A to D below. In each case, *threshold* represents the hard threshold function that returns 0 for negative inputs and 1 for positive inputs.

For each hypothesis, work out whether it correctly predicts the label for all four of the examples. If it does, enter 'y' into the box, otherwise enter 'n'.

The hard threshold activation function, also known as the step function, is one of the simplest activation functions used in neural networks. It is a binary function that operates as follows: [CHAT GPT EXPLANATION](#)

- If the input is greater than or equal to a certain threshold value, the function outputs 1.
- If the input is less than the threshold, the function outputs 0.

*= See example calculations below.

KEYWORDS: hard threshold activation function; neurons

A. $f(x,y) = \text{threshold}(x + y - 4.5)$

B. $f(x,y) = \text{threshold}(x + y - 3.5)$

C. $f(x,y) = \text{threshold}(-x - y + 3.5)$

D. $f(x,y) = \text{threshold}(x + y - 2.5)$

These answers match the sample solutions provided.

Example calculations:

For A, line 1: $1 + 1 - 4.5 = -2.5$ so 0 as it's negative

For B, line 2: $1 + 2 - 3.5 = -0.5$ so 0 as it's negative

For C, line 3: $-2 * -1 + 3.5 = 5.5$ so 1 as it's positive

For D, line 4: $2 + 2 - 2.5 = 1.5$ so 1 as it's positive

TIPS: To avoid confusion, I suggest doing a table like the above. Also you may not need to work out every equation. i.e. if you find one that does not match the label, you can conclude that the answer is n (e.g. equation C, the first one doesn't match). Also, perhaps work backwards e.g. do line 4 first.)

miro

Question 16

9 pts

Suppose you have a convolutional neural network with an $9 \times 9 \times 3$ input volume representing a colour image. The first layer is a convolutional layer consisting of 10 filters that have a $4 \times 4 \times 3$ receptive field. Calculate the number of neurons and the total number of weights in the first layer.

Neurons:

Weights:

The output of the first layer is a volume. Give its dimensions.

Output volume dimensions:

These answers match the sample solutions provided.

KEYWORDS: convolutional neural network (CNN);
input volume; filters; receptive field; neurons; weights

$4 \text{ (w)} \times 4 \text{ (h)} \times 3 \text{ (d)}$ filter has $48 \text{ weights} + 1 \text{ bias} = 49 \text{ weights/filter}$
 $\times 10 \text{ filters} = 490 \text{ weights}$

For the neuron / dimension questions, we do:

$$\begin{aligned}\text{Width Out} &= \text{Width In (9)} - \text{Filter Width (4)} + 1 \\ &= 6\end{aligned}$$

$$\begin{aligned}\text{Height Out} &= \text{Height In (9)} - \text{Filter Height (4)} + 1 \\ &= 6\end{aligned}$$

$$\begin{aligned}\text{Depth Out} &= \text{Depth In (3)} - \text{Filter Depth (3)} + 1 \\ &= 10\end{aligned}$$

Dimensions: $6 \times 6 \times 10$

Neurons: $6 \times 6 \times 10 = 360$

'Stolen' from Discord

Feature	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Definition	Learning from labelled data	Learning from unlabelled data	Learning by interacting with environment
Goal	Predict output	Find patterns/structures	Optimal action selection
Examples	Predicting house prices	Customer segmentation	Learning to play chess
Training Data	Labelled	Unlabelled	Actions and Rewards
Key Concepts	Bias-Variance Tradeoff, Underfitting, Overfitting	Clustering, Dimensionality Reduction	Exploration vs. Exploitation, Reward Shaping
Algorithms	SVM, Decision Trees, Neural Networks	K-Means, Hierarchical Clustering, PCA	Q-Learning, Policy Gradients
Feedback	Direct (labels)	None	Rewards/Penalties
Model Evaluation	Accuracy, Precision, Recall	Silhouette Score, Davies–Bouldin index	Cumulative Reward, Average Reward
Challenges	Risk of overfitting	Determining cluster number	Defining a suitable reward function
Use-Cases	Spam detection, Image classification	Market basket analysis, Recommender systems	Game AI, Autonomous vehicles
Application Domains	Medicine, Finance	Marketing, Social Science	Robotics, Gaming
Model Example	Linear Regression	DBSCAN	Deep Q-Networks
Advanced Concepts	Ensemble Learning, Neural Network Architectures	Anomaly Detection, Association Rule Learning	Multi-Agent Learning, Inverse RL

Comparison Chart -
Supervised Learning,
Unsupervised Learning;
Reinforcement Learning
(thanks to Silver)

Question 1 KEYWORDS: chatbot definition of AI

1 / 1 pts

A chatbot is an AI system that holds conversations with people. In some cases, chatbots learn behaviour from the people they are talking to.

In 2016, Microsoft launched a chatbot called Tay, which talked to people through Twitter. Tay was immediately sabotaged by a group of people who taught it to say racist things.

Which definition of AI would a learning chatbot like Tay fall under?

Acting rationally

Thinking humanly

Acting humanly

Correct. The aim of a chatbot is to make people feel like they are talking to another human, so 'acting humanly' is the right answer.

Thinking rationally

KEYWORDS: satellite navigation systems definition of AI; satnav

1 / 1 pts

Satellite navigation systems advise drivers on the best route to take through busy cities.

They deal with certain knowledge (the road network) and likely knowledge (such as predictions of congestion in some areas at particular times), and should always produce the route that is expected to be the best.

Which definition of AI would best match a satellite navigation system?

Thinking humanly

Acting rationally

Correct. Satellite navigation systems should find the very best route given the information they have. It does not matter how they find it, or whether the route is similar to one that a human might have found, so the aim is to 'act rationally'.

Thinking rationally

Acting humanly

KEYWORDS: satellite navigation systems acting humanly problems; satnav

Question 3

1 / 1 pts

Suppose a satellite navigation system were "acting humanly".

What problems might there be?

Poor quality routes

Correct. Poor quality routes is the right answer, humans do not necessarily find good routes through a busy city. If you answered that it might criticise the driver: the user interface of a satellite navigation system is only able to give directions, not to criticise the driver! The AI component is only responsible for finding routes, not interacting with the driver directly.

Excessive use of the CPU

Criticism of the driver

Lack of 'common sense' background information

Question 4

KEYWORDS: expert systems definition of AI

1 / 1 pts

Expert systems use rules expressed in logic to make decisions or recommendations (such as configuring a computer system).

In most expert systems the rules are perfectly correct, and the algorithm for applying them is also correct.

Suppose that an expert system can produce good decisions, but not necessarily the best possible decisions. Which definition of AI would best match an expert system like this?

Thinking humanly

Thinking rationally

Correct. Since the system is not providing the best possible answer, and it is using well-founded (correct) rules to reach its conclusions, it is 'thinking rationally'.

Acting humanly

Acting rationally

KEYWORDS: satellite navigation systems definition of AI; satnav

Question 2

1 / 1 pts

Satellite navigation systems advise drivers on the best route to take through busy cities.

They deal with certain knowledge (the road network) and likely knowledge (such as predictions of congestion in some areas at particular times), and should always produce the route that is expected to be the best.

Which definition of AI would best match a satellite navigation system?

Thinking humanly

Acting rationally

Correct. Satellite navigation systems should find the very best route given the information they have. It does not matter how they find it, or whether the route is similar to one that a human might have found, so the aim is to 'act rationally'.

Thinking rationally

Acting humanly

KEYWORDS: expert systems; best possible decision; definition of AI

Question 5

1 / 1 pts

Suppose the expert system from question 4 always produced the best possible decision every time. As well as "thinking rationally," which other definition of AI would be appropriate?

Acting capriciously

Acting humanly

Acting randomly

Acting rationally

Correct. In this case, it is 'acting rationally' as well as 'thinking rationally', because it is finding the best possible answer.

Acting theatrically

KEYWORDS: satellite navigation systems acting humanly problems; satnav

Question 3

1 / 1 pts

Suppose a satellite navigation system were "acting humanly".

What problems might there be?

Poor quality routes

Correct. Poor quality routes is the right answer, humans do not necessarily find good routes through a busy city. If you answered that it might criticise the driver: the user interface of a satellite navigation system is only able to give directions, not to criticise the driver! The AI component is only responsible for finding routes, not interacting with the driver directly.

Excessive use of the CPU

Criticism of the driver

Lack of 'common sense' background information

KEYWORDS: general problem solver; GPS; AI definition

Question 6

1 / 1 pts

The "General Problem Solver" (GPS) by Newell, Shaw, and Simon was an early AI system, developed around 1961.

GPS could solve simple problems by applying a sequence of reasoning steps. They compared traces of its reasoning steps to traces of human subjects solving the same problems.

Which definition of AI would best describe their research methods?

Thinking rationally

Thinking humanly

Correct. When an AI system is designed to perform reasoning steps that are similar to human reasoning, then 'thinking humanly' is the most closely-matched definition.

Acting rationally

Acting humanly

Q.1 DISCUSSION POINT: WHAT ARE SOME COMMON ETHICAL / LEGAL ISSUES THAT COULD BE DISCUSSED? DOES ANYBODY WANT TO SHARE THEIR FORMATIVE ANSWER AND THE FEEDBACK RECEIVED?

we discussed BIAS and Legal aspects as 2 potential areas for cosnid

Q.5 PAPER 1 DISCUSSION POINT: ALGORITHM TIME / SPACE COMPLEXITY

Q.2 PAPER 2 DISCUSSION POINT: HOW CAN WE RECOGNISE OTHER ALGORITHMS

WHAT OTHER QUESTIONS MIGHT COME UP THAT ARE NOT NECESSARILY COVERED IN PAPERS 1 / 2. WHAT DO YOU THINK MIGHT BE IN THE EXAM?

- ACTING RATIONALLY VS ACTING HUMANLY?
- REGRESSION?
- WEEK 4 Q.8?
- FORWARD AND BACKWARD CHAINING?

PAPER 1 - SUPPLIED MODEL ANSWER:

An autonomous delivery robot could fail to recognise a person when moving in a shared area, for example by misclassifying the person as an inanimate object. In the worst case this could lead to injury of an already ill patient, by the robot failing to stop or deciding to collide with the person instead of an obstacle such as a bin – so it would be a serious ethical concern. If the risk is not suitably managed, then it could prevent the deployment of the autonomous robot.

A potential solution is to use the largest possible set of training examples that include a wide variety of images of people, including different ethnicities, clothing, height, weight, lighting conditions etc. A computer vision system trained on a large and sufficiently diverse set of examples could potentially be more accurate than humans. A system that is more accurate than humans could potentially be argued to be sufficiently safe.

Another aspect of this issue is bias (for example racial bias). For example, if the AI system is able to recognise white people more accurately than black people, and thus the robot presents more of a danger to black people, then it would not be acceptable for social reasons even if on average it were safer than (for example) a robot controlled by a human. The issue of bias can potentially be solved with a suitably diverse set of training instances.

PAPER 1 - CHAT GPT SUMMARY OF ALI'S ANSWER

The deployment of autonomous robots for medical supply delivery raises significant concerns about privacy and security. These robots, equipped with visual sensors, may capture sensitive data such as medical records and personal information. Given the strict regulations governing medical data, especially in countries like the USA with HIPPA laws, safeguarding this information is crucial. To address this issue, a viable solution involves storing data locally, encrypting all gathered data (including videos) to a high standard, and implementing automatic purging of local data after a specified time, potentially 24 hours. Although this limits the manufacturer's access to training data, it enhances the safety and security of both individuals and the data encountered by the robot.

ALI'S FEEDBACK

You have presented ethical issues that have been shown in answers of Q1 for instance, an autonomous robot using visual sensors to navigate will be capturing sensitive data; that data must be kept safe and the systems storing it must be secure. You could also expand this by misclassifying the person as an inanimate object. You have offered solutions, for example, store data locally, to encrypt all data gathered by the device (including videos) to an advanced standard, and to purge the local data storage after a fixed amount of time. You might also think about whether the robot can accurately categorise, recognise, and avoid employees and patients, as well as deliver the proper medical supplies or equipment to the right staff or patients. You could provide more solutions for example, use the largest possible set of training examples that include a wide variety of images of people, including different ethnicities, clothing, height, weight, lighting conditions, etc to avoid misclassifying a person as an object.

PAPER 1 - CHAT GPT SUMMARY OF JOHN'S ANSWER

The use of delivery robots in hospitals sharing space with humans can lead to accidents, especially during busy times. Although robots can be trained to avoid collisions, humans cannot be trained to avoid robots. Accidents could result in damage to the robot or injuries to humans, leading to potential legal implications such as lawsuits and claims for compensation. Establishing blame for such accidents could involve complex and costly court proceedings, involving designers, manufacturers, and injured parties. There is no obvious solution to this problem, and while AI delivery robots have significant societal benefits, legal and ethical issues like these must be addressed.

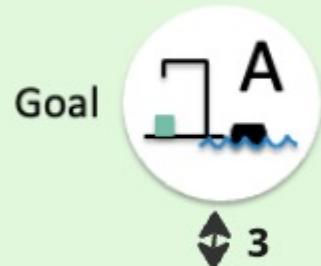
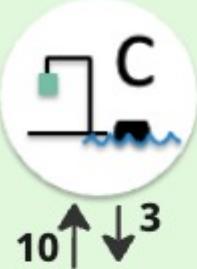
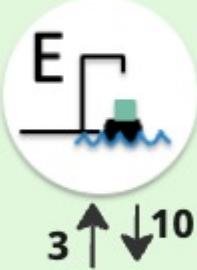
JOHN'S FEEDBACK

Examples are used appropriately to justify the ethical and or legal issue chosen and the solutions are sound. Solutions are critically presented using a wide range of discussion to support the ideas presented. Shows depth and sound reasoning that identifies the pertinent points. The answer is insightful and shows sound thinking and argument that we are looking for. Where this could be improved is to deepen the discussion and provide a wider range of alternatives and arguments to consider but this is insightful in considering the human factor and the need to allow for this.

PAPER 2 - SUPPLIED MODEL ANSWER

The historical training data is likely to contain biased decisions, and there is a risk that the system could learn the same biases. The attributes include the area where the customer lives (via the postal code) which is predictive of ethnicity and religion in some cases. Suppose that staff making loan decisions had unconsciously discriminated against ethnic minority customers, being more likely to refuse loans to them, and as a result the 'reject' decisions are more likely for postal codes belonging to neighbourhoods with a significant ethnic minority population. Decisions made by the system could then discriminate in a way that is unethical and possibly even illegal depending on local laws (e.g. the Equality Act 2010 in the UK). This would be a very serious issue and would need to be suitably managed.

One potential solution would be to remove any attributes that are predictive of ethnicity, religion or other protected characteristics, for example removing the postal code and making decisions based only on credit history and current salary. However this is likely to reduce the accuracy of the system because it has fewer attributes to use to make the prediction. Postal codes predict the value of a person's main asset (i.e. their house). Another possibility would be to investigate the training data for biases, or to monitor the system for bias when in use. Both approaches would be plausible ways of managing the risk of bias.



A* Search (Tree Based)

- priority queue using $f(n) = g(n) + h(n)$

Start: Frontier = [B]

B is selected for expansion

B → Frontier = [E(3+16=19)]

E is selected for expansion

E → Frontier = [B(6+20=26), F(10+9=19)]

F is selected for expansion

F → Frontier = [B(26), E(16+16=32), C(7+3=10)]

C is selected for expansion

C → Frontier = [B(26), E(32), F(7+9=16), A(3+0=3)]

A is selected for expansion

End: Explored Nodes = [B, E, F, C, A]

A is the Goal

State	Crane position	Holding container?	Container position	Heuristic value $h(n)$
A	Land	No	Land	0
B	Land	No	Ship	20
C	Land	Yes	Land	3
D	Ship	No	Land	5
E	Ship	No	Ship	16
F	Ship	Yes	Ship	9