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**
	
	• 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
		
		• **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:**
		  • Natural Language Processing (NLP)
		  • knowledge representation
		  • storing what the machine knows
		  • automated reasoning
		  • new conclusions
		  • machine learning
		  • adaptation, extrapolation
		• **Total Turing Test** - Additionally requires interaction with objects in the real world.
		  • computer vision
		  • robotics
		• Performing actions that require intelligence

Rationally	**Logic (with Probability for uncertainty)** • syllogisms - patterns for argument structures • logic, which requires knowledge of the world that is certain • probability, which allows for rigorous reasoning about uncertain information (!) Thinking rationally does not in itself generate intelligent behaviour; for this we need a theoury of how to translate thinking into rational action.	**Logic & Probability with an Agent** • rational agent - acts so as to achieve the best (expected) outcome • standard model - a paradigm which models agents as doing the right hting, 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 -
haviour and
hasizing 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 *Actions(s)* 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 *Results(s, a)* 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: • 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

Harvard CS50 Intro to AI with Python: Lecture 0

Search Problems (Problem-Solving Agents)

Solving problems by Searching

The goal: Planning ahead, to forumlate 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.

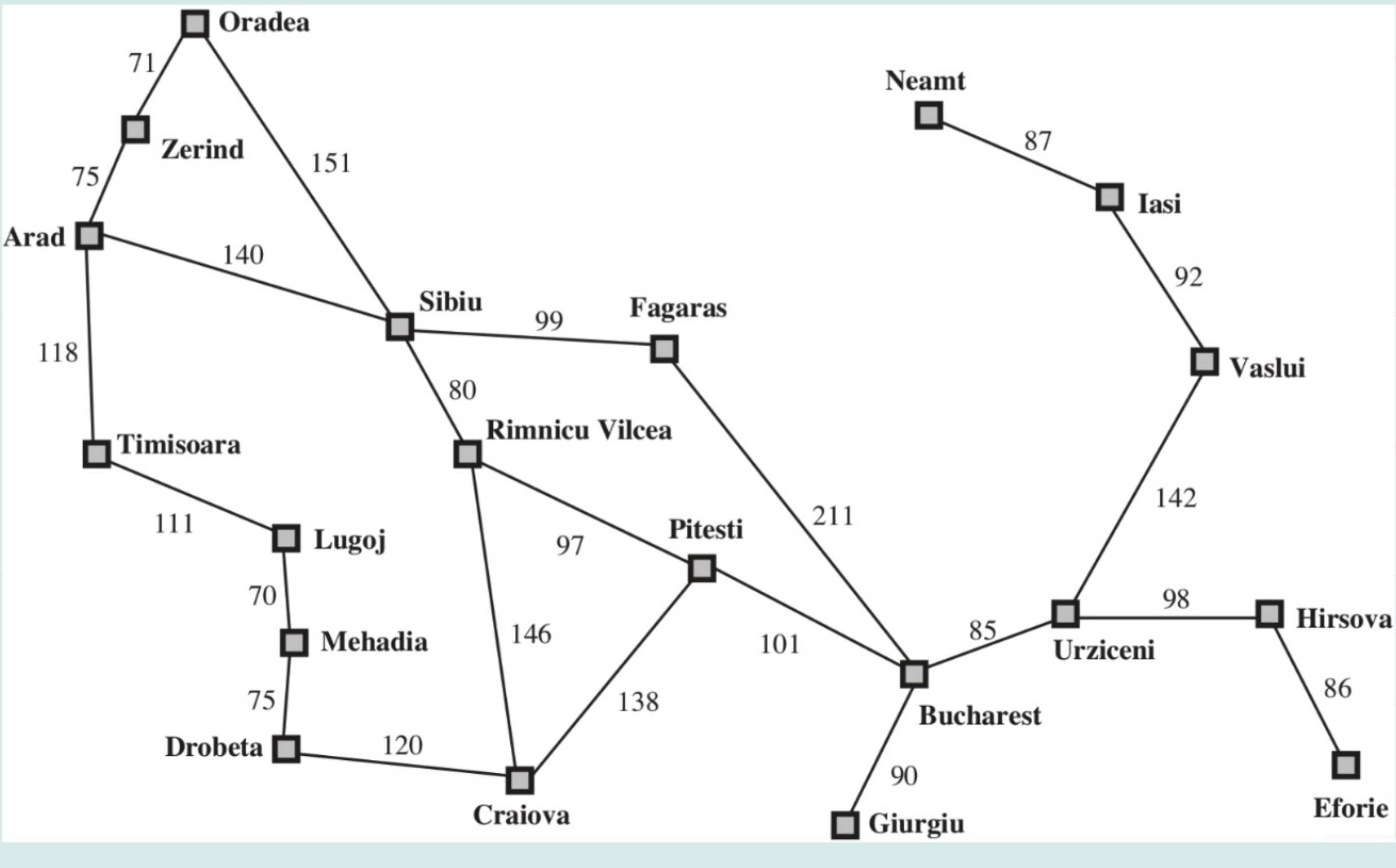
Informed algorithms - the agent can estimate how far it is from the goal (**Week 3**)
Uninformed algorithms - no such estimate is available (**Week 2**)

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 sequesnce of actions that reaches a goal
    -> this sequence is called a solution.
4. Execution - the agent executes the actions int he solution, one at a time.

In a fully observable, deterministic, known environment, teh solution to any problem is a fixed sequence of actions (i.e. we can ignore precepts 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 avialable 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?

[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 htese?
- 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?

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

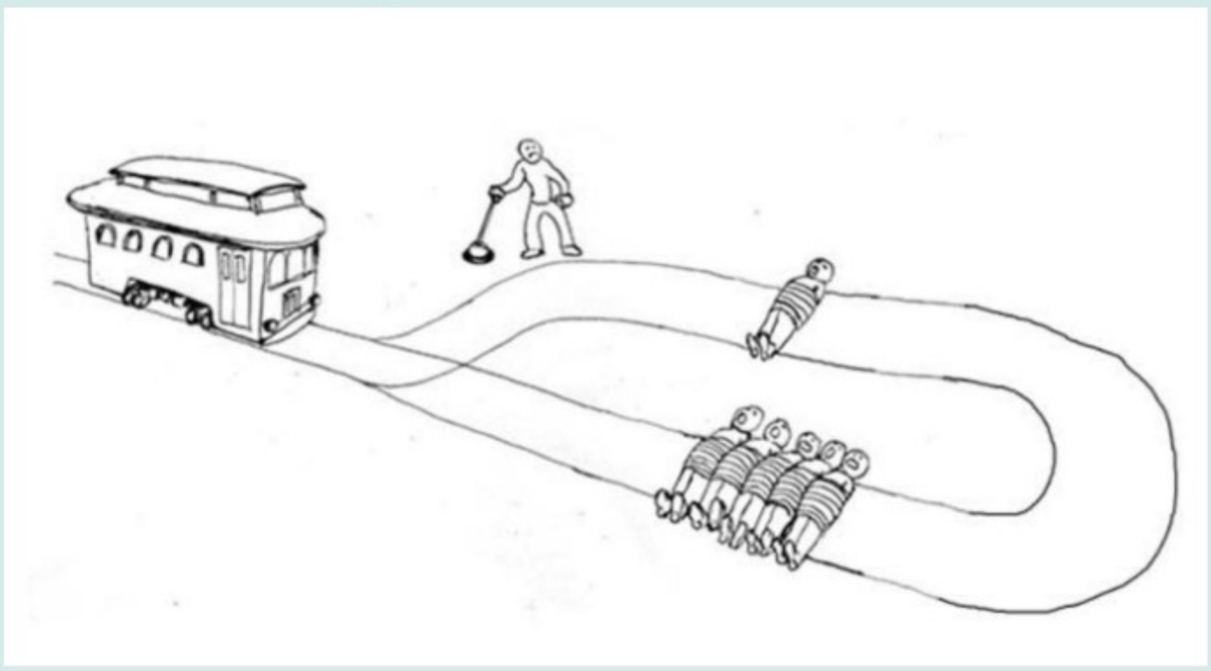
Eurovision song created by AI https://www.youtube.com/watch?v=4MKAf6YX_7M

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

- "the best chess players int he 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 trolly 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 peoples 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.



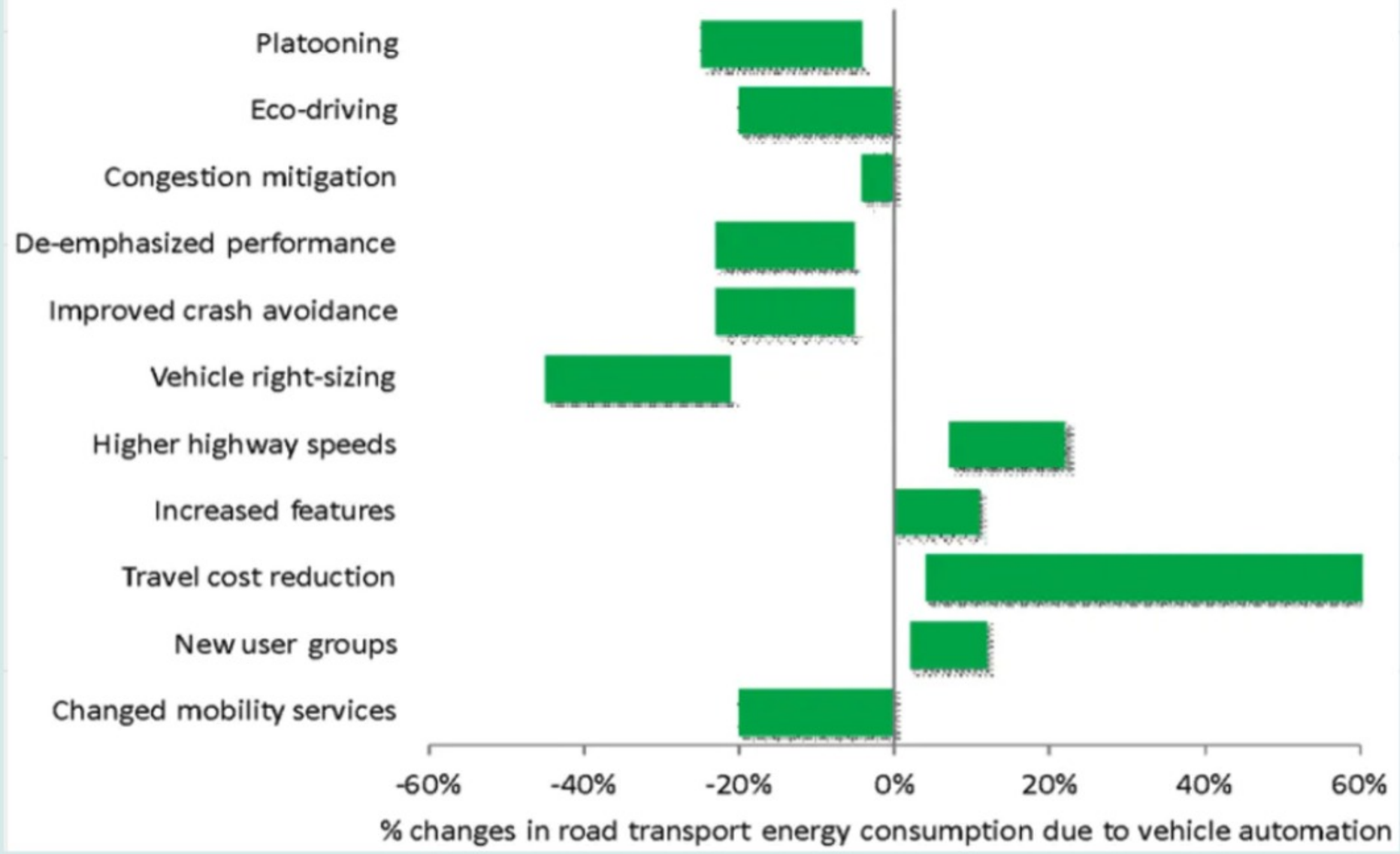
"Will self-driving cars reduce energy use and make travel better for the environment?:

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:

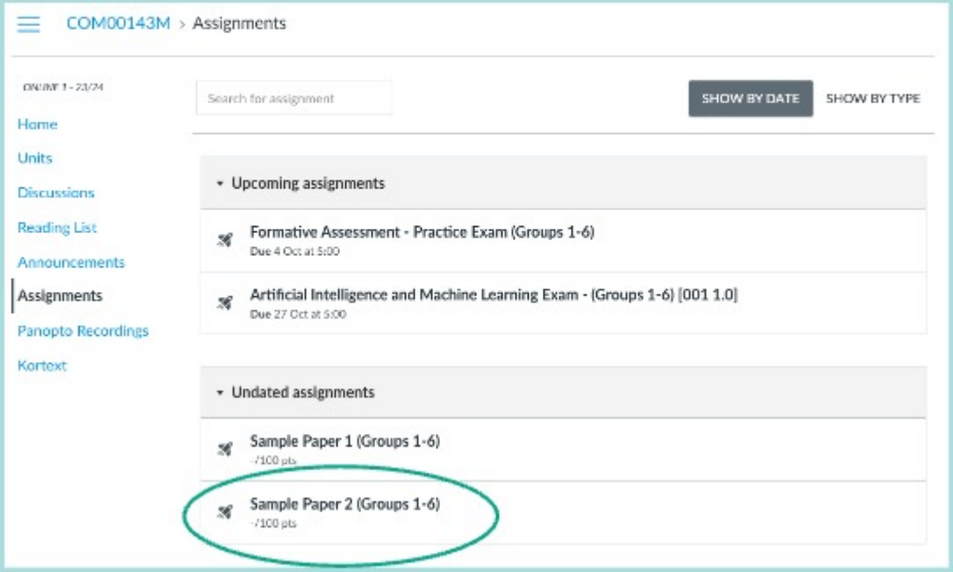
Thursday, 14 Sept 2023, 19:30 UK time
Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

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.



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:

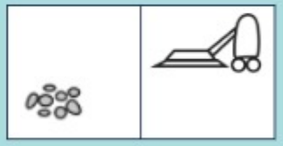
- Vacuum world
- 8-puzzle
- 8 queens problem
- Knuth's problem

Vacuum World

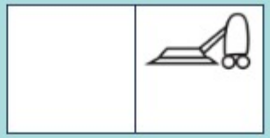
The vacuum world problem is **deterministic**, but variations can include **uncertain** (not fully observable or not deterministic) elements such as randomly appearing dirt or an unreliable suction mechanism.

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.

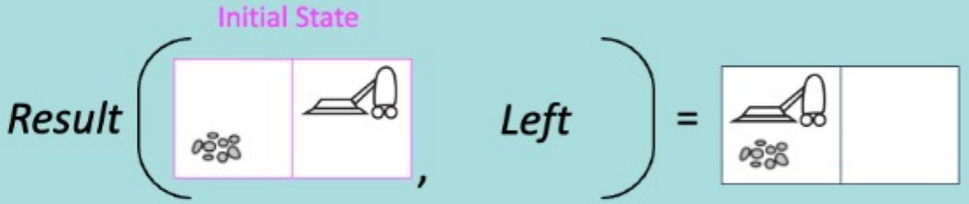


- **Actions:** Left, Right, Suck, NoOp
- **Action cost:** +1 per action, 0 for NoOp
- **Goal States -** Where everything is clean. In this case there are (2) possibilities.



- **Transitional Models**

Initial State

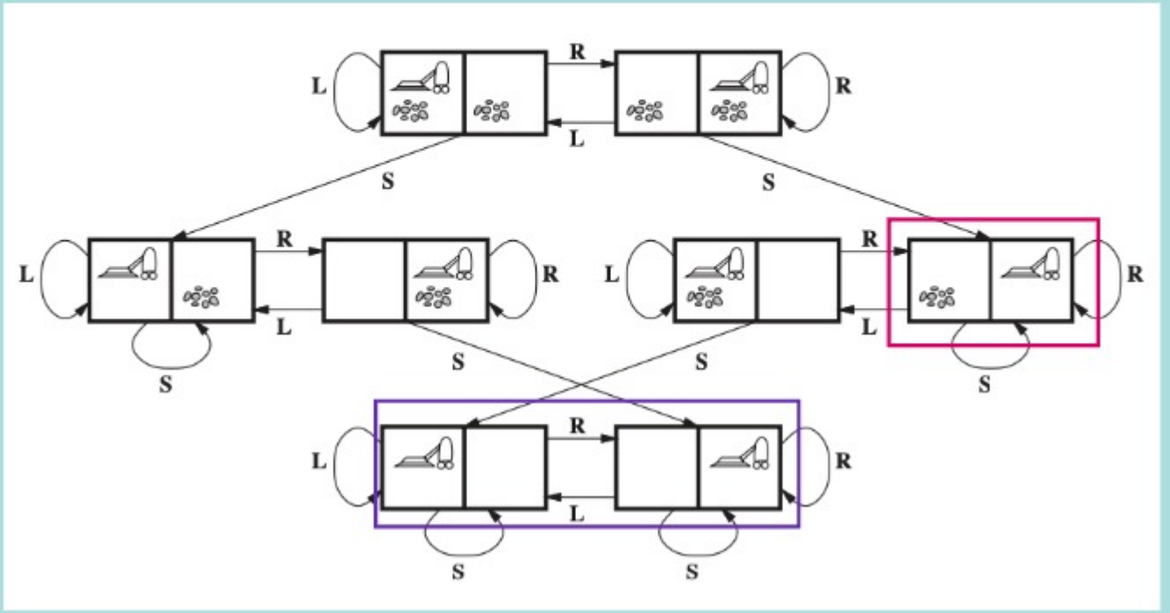


Total Cost = 1

Total Cost = 2

Goal State

State Space



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

AI search problems Quiz

Question 1 — 1 pts

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
- ○ Maps from a state to an action
- ○ Maps from an action to a state

Question 2 — 1 pts

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

- ○ Empty
- ○ Leftmost
- ○ Deconstructed
- ○ Bottom
- ○ Top

Question 3 — 1 pts

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 4 — 1 pts

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 5 — 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
- ○ A special state that must be in all paths from the initial state to any goal state

Question 6 — 1 pts

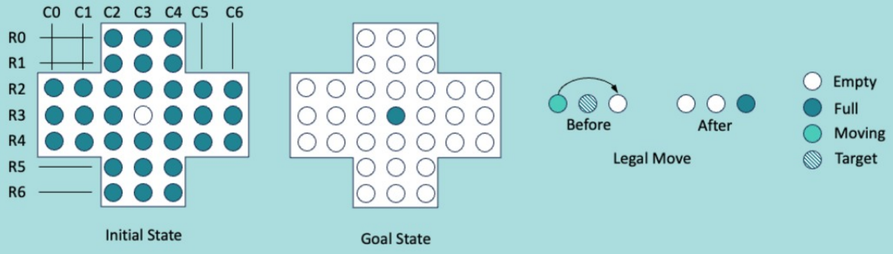
Which of the following statements best describes an optimal solution?

- ○ 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

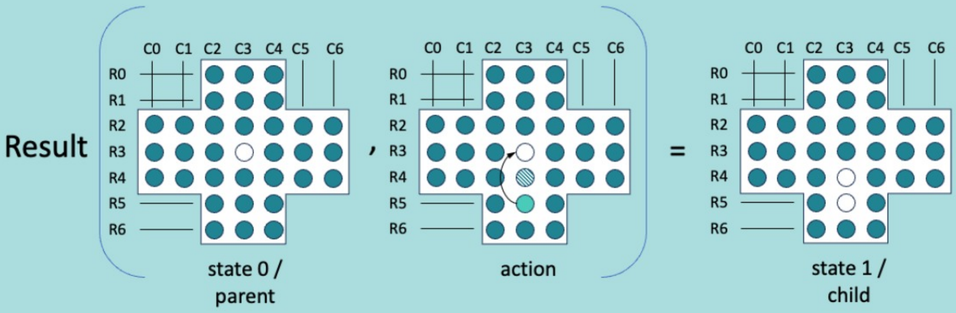
Find out how to play English Peg Solitaire (the game is described on Wikipedia, for example). Formulate it as an AI search problem, starting by defining a state. Then define the actions that can be performed, and the transition model.

- 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.



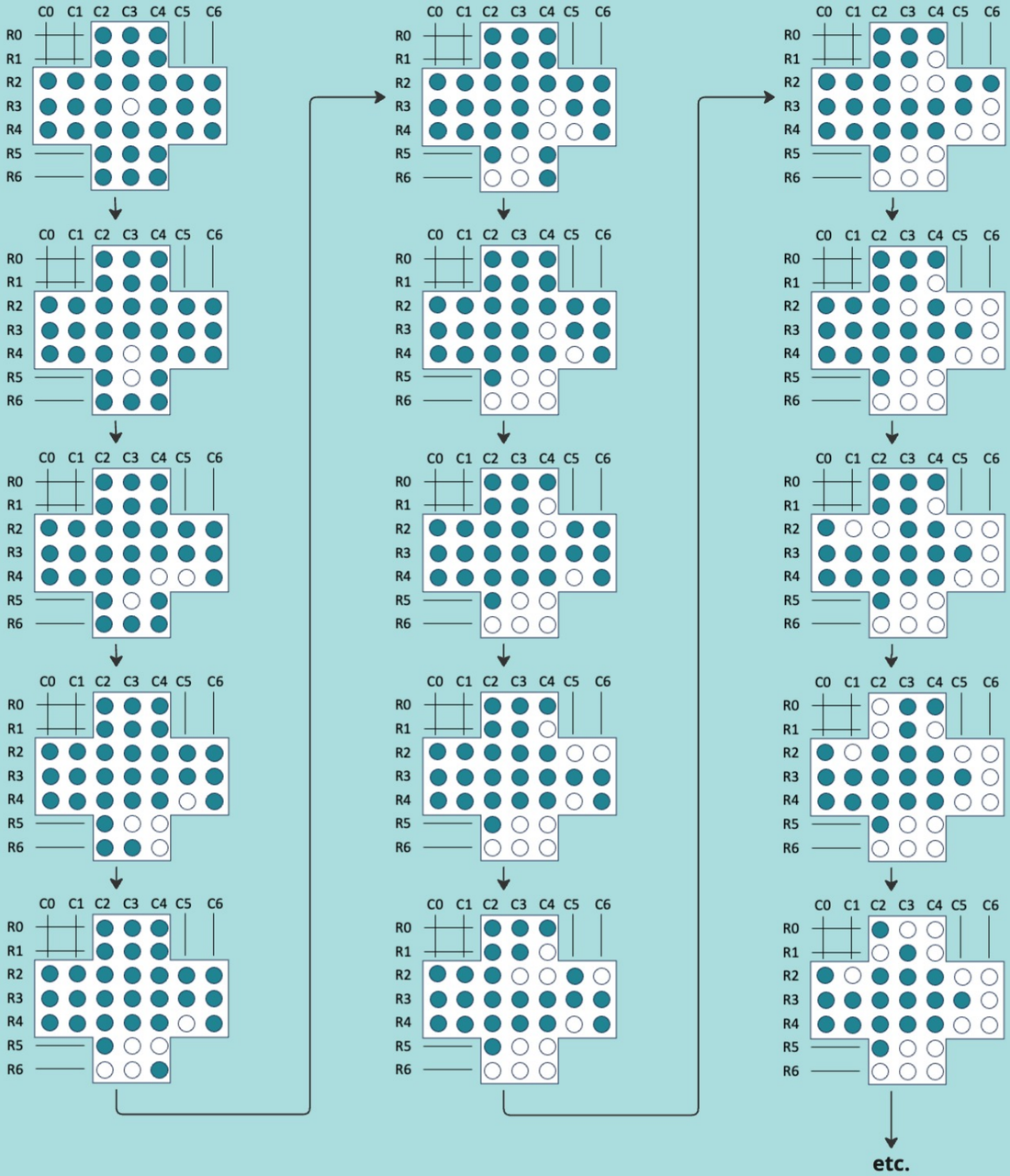
Step Costs = 1? Does it really matter? How many different ways can you solve this successfully?

Transitional Model



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

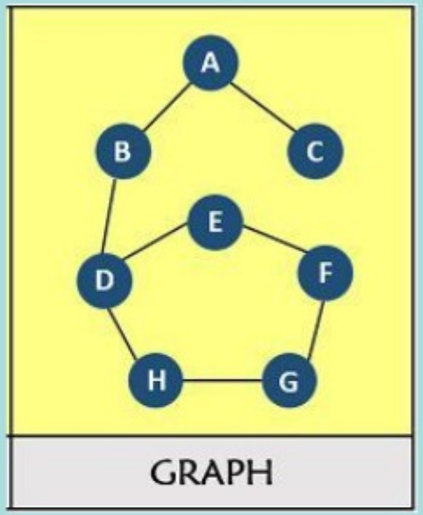
State Space - Each of these states are sequential nodes on the OPTIMAL PATH.



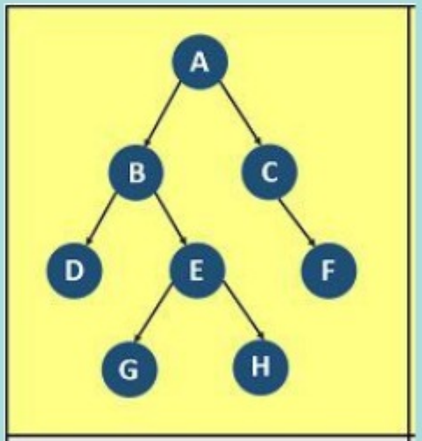
Following this model will solve the puzzle (i.e. final result = goal state) after 32 moves (cost = 32).

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



GRAPH



TREE

Infrastructure of Search Algorithms

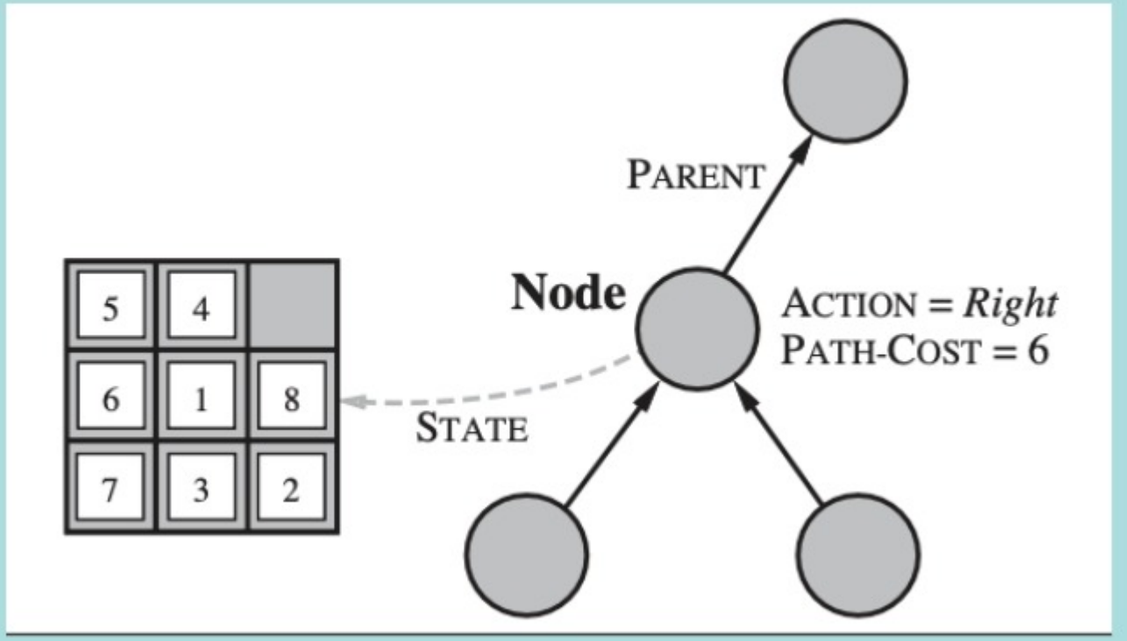
Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node *n* we have a structure that contains four components:

- ***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

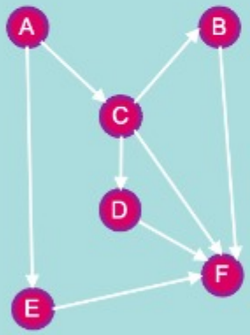
The difference between **Tree Search** and **Graph Search** is **-NOT-** that tree search works on trees and graph search works on graphs. Both can work on either. The key difference is that graph search uses a data structure called the **explored set (closed list),** which remembers every expanded node. 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:

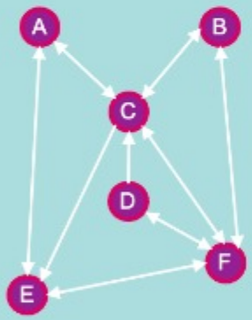
- Start with a **frontier** that contains the initial state of the *problem*.
- **LOOP DO**:
  - **IF** the frontier is empty, **THEN return** failure (no solution).
  - Choose a frontier node for expansion using *strategy* and remove from the frontier.
  - **IF** the node contains the goal state, **THEN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



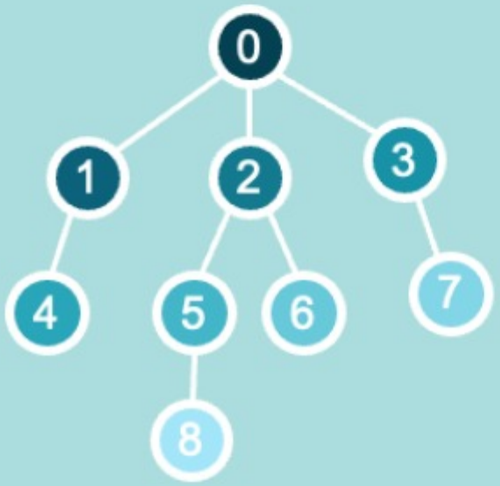
The frontier is a dataset. The *strategy* that is drive which node is selected next.

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.

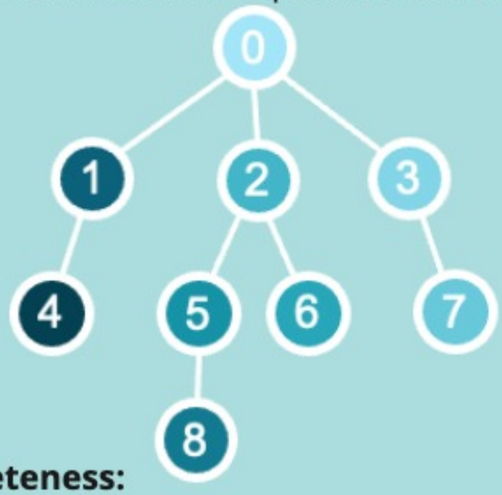



```
At node 0 in the BFS
Adding node 1 to the queue in the BFS
Adding node 2 to the queue in the BFS
Adding node 3 to the queue in the BFS
Done processing node 0
At node 1 in the BFS
Adding node 4 to the queue in the BFS
Done processing node 1
At node 2 in the BFS
Adding node 5 to the queue in the BFS
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 is 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: Possible to find a solution which is not the shortest path.

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*=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 ∞ 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 ← 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 ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

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.

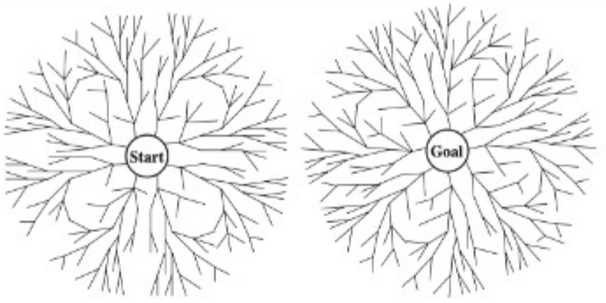


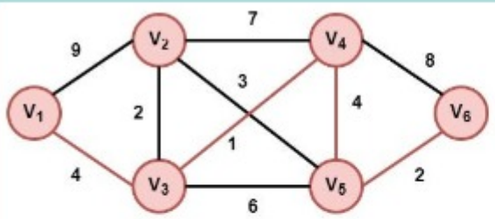
Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

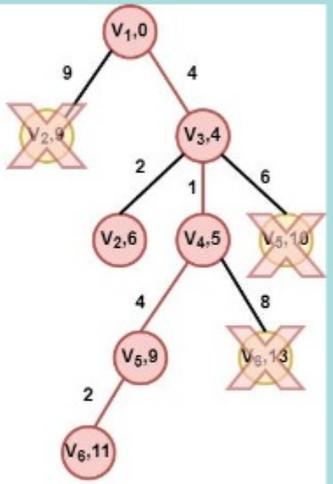
- **Complete** if *b* is finite and if both directions use breadth-first search
- **Optimal** if step costs are identical and if both directions use breadth-first search
- **Time Complexity:** O(b^(d/2))
- **Space Complexity:** O(b^(d/2))

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
```





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_6,13),(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)

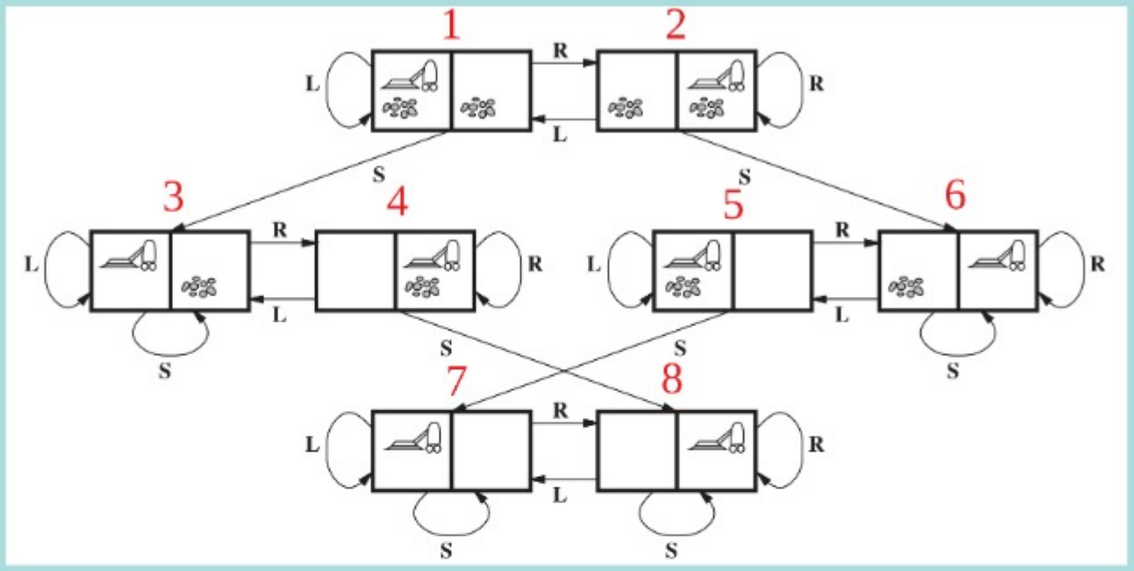
Uniformed search methods have access only to the problem definition. The basic algorithms are as follows:

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



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

Code distributed by the tutors

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

```
import java.util.*;

class VacuumBFS {
  //  Constant arrays to look up the result of applying an action to a given state.
  //  For example, looking up state 1 in the actionRight array gives state 2
  //  which is the result of moving right from state 1 (see the diagram in unit 2.8)
  static int[] actionLeft= {1,1,3,3,5,5,7,7};
  static int[] actionRight={2,2,4,4,6,6,8,8};
  static int[] actionSuck= {3,6,3,8,7,6,7,8};

  //  Returns true when s is a goal state.
  static boolean goalTest(int s) {
    return (s==7 || s==8);
  }

  //  Returns true if there is a solution, i.e. a path from state 1 to 7 or 8.
  static boolean BFS() {
    int initialState=1;
    if(goalTest(initialState)) {
      return true;
    }

    //  frontier initially contains just 1.
    ArrayList<Integer> frontier=new ArrayList<Integer>();
    frontier.add(1);
    //  frontier will be used as a FIFO queue to implement BFS.

    //  explored is initially empty.
    ArrayList<Integer> explored=new ArrayList<Integer>();

    while(true) {
      if(frontier.size()==0) {
        //  There are no more nodes to expand, and we did not find a goal.
        return false;
      }

      //  Remove the first element of frontier
      int node=frontier.remove(0);
      explored.add(node);

      //  Expand 'node'
      //  Make children array containing the three states that
      //  result from applying the three actions to the current state.
      int[] children = {actionLeft[node-1], actionRight[node-1], actionSuck[node-1]};

      System.out.println("Expanding node: "+node+" to: "+Arrays.toString(children));

      for(int child : children) {
        if(goalTest(child)) {
          System.out.println("Goal state found: "+child);
          return true;
        }
        if( !frontier.contains(child) && !explored.contains(child) ) {
          //  This state has not been seen before.
          //  Add it to the end of the frontier list.
          frontier.add(child);
        }
      }
    }
  }

  public static void main(String args []) {
    boolean solutionFound=BFS();
    System.out.println("Solution found: "+solutionFound);
  }
}
```

```
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
```

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
public class BFSVacuumWorld {
    public static void main(String[] args) {
        // Define the neighbours for each state.
        // The neighbours are defined as [leftMove, rightMove, suckMove]
        int[][] neighbors = { { 1, 2, 3 }, // State 1
                { 1, 2, 6 }, // State 2
                { 3, 4, 3 }, // State 3
                { 3, 4, 8 }, // State 4
                { 5, 6, 7 }, // State 5
                { 5, 6, 6 }, // State 6
                { 7, 8, 7 }, // State 7
                { 7, 8, 8 } // State 8
        };
        // Initialise the frontier queue as a LinkedList with the first state
        Queue<Integer> frontier = new LinkedList<>();
        frontier.add(1);
        // Initialise the explored nodes as an empty ArrayList
        ArrayList<Integer> explored = new ArrayList<>();
        // Breadth-First Search Loop
        while (!frontier.isEmpty()) {
            int currentState = frontier.poll();
            System.out.println("Expanding State: " + currentState);
            // Mark the current state as explored
            explored.add(currentState);
            // Check if goal state before expanding node
            if (currentState == 7 || currentState == 8) {
                System.out.println("Goal State " + currentState + "Reached!");
                        return;
            }
            // Expand the current state
            for (int neighbor : neighbors[currentState - 1]) {
                if (!explored.contains(neighbor) && !
                        frontier.contains(neighbor)) {
                    frontier.add(neighbor);
                    System.out.println(" Child Node: " + neighbor);
                }
            }
        }
    }
}
```

```
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:

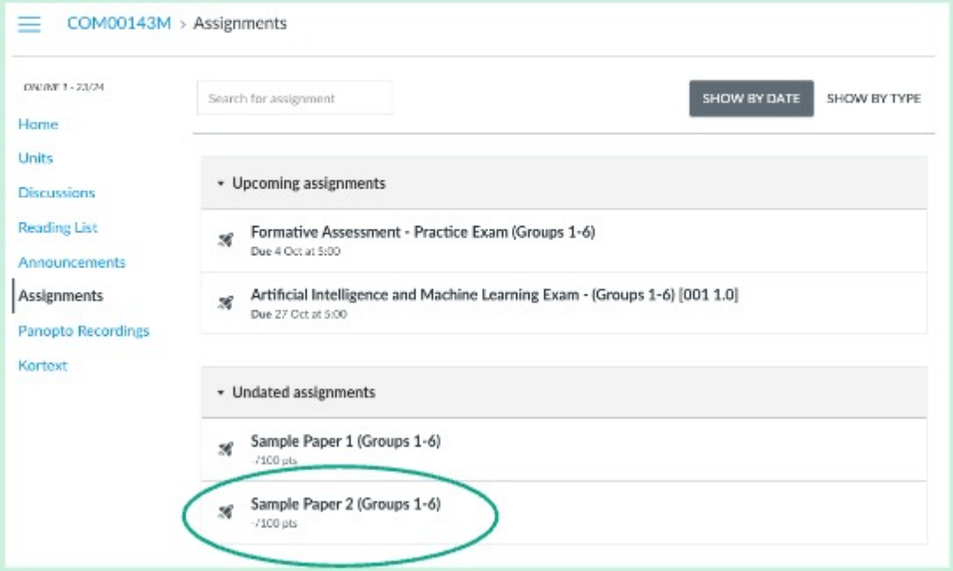
Thursday, 21 Sept 2023, 19:30 UK time
Studies held on the YO-MCS Discord Channel #aiml-chat, please reach out to an invite to the channel

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.



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

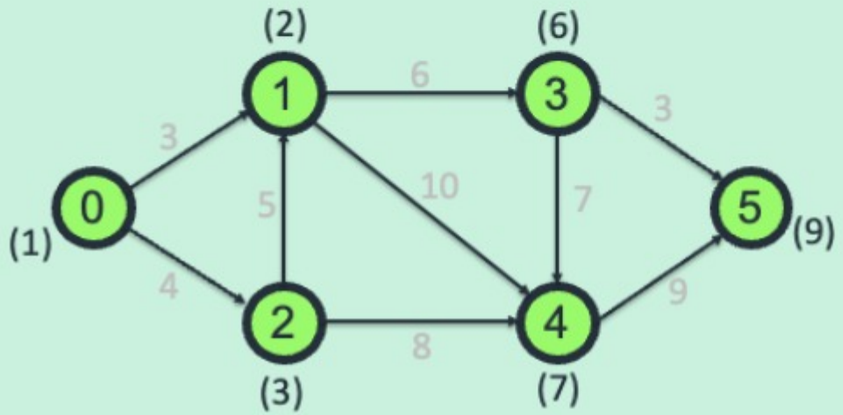
- An instance of BFS where the priority queue is driven by both the cost to reach the node *g(n)* and a heuristic function *h(n)*.
- *f(n) = g(n) + h(n)* = estimated cheapest solution through n
- A* is **complete**. It is **optimal**. And it is **optimally efficient**. However it will often hold a large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

Not used with GBFS

Heuristic

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

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

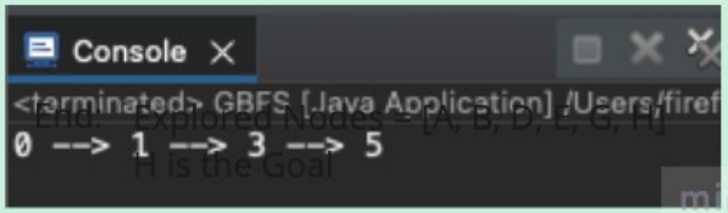
Completeness:

- 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



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 becuase 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* = (h* - h)/h*
- 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^ed) 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.

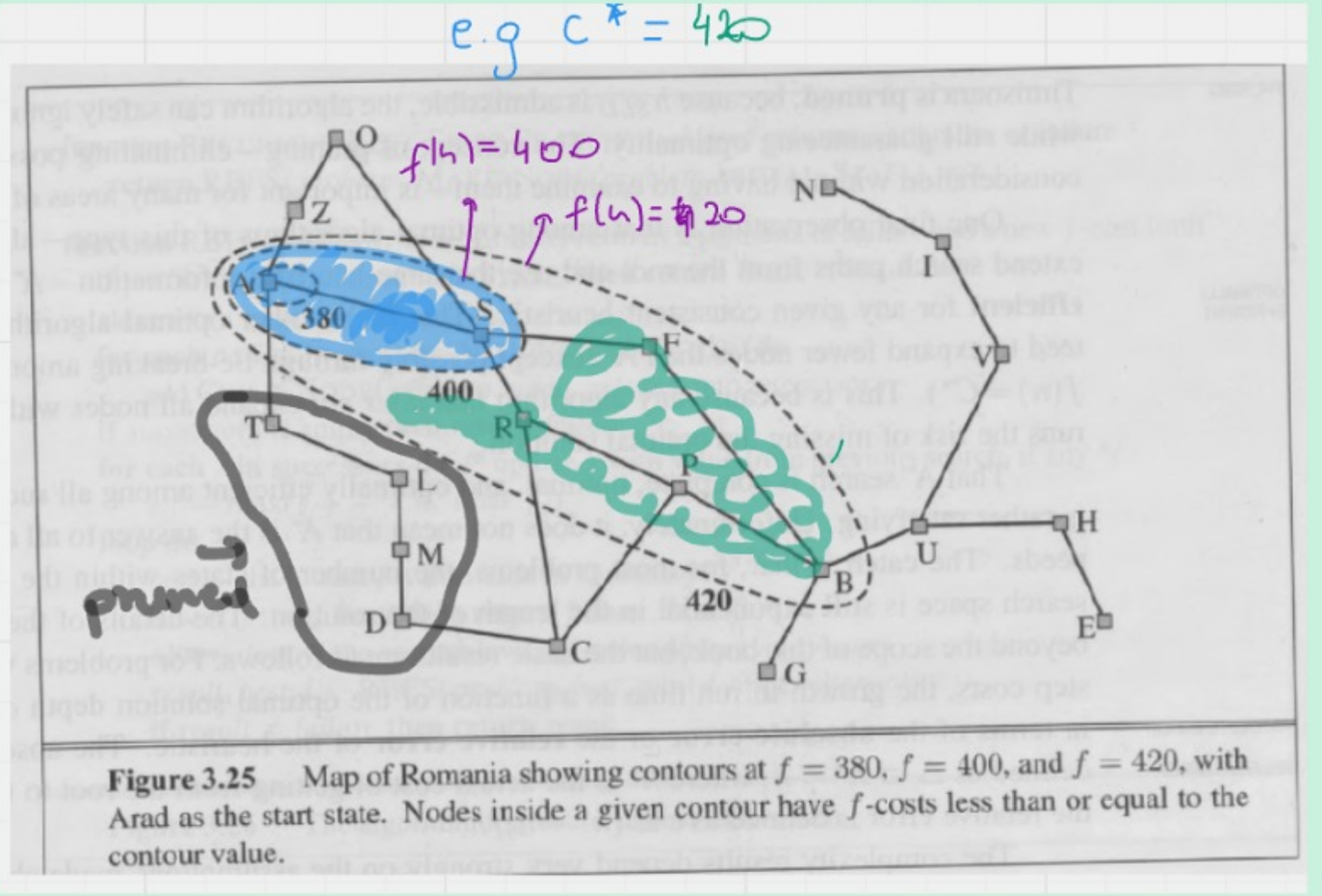


Figure 3.25 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.

3.4 Quiz: Informed Search Strategies

Question 1 — 1 pts

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

- [x] It always produces optimal solutions.
- [ ] 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.
- [x] The heuristic helps Greedy Best-First Search to select nodes closer to the goal.
- [ ] It does not re-generate nodes and Iterative Deepening does.

Question 2 — 1 pts

What does it mean for a heuristic to be admissible?

- (●) 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.

Question 3 — 1 pts

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.
- ( ) 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.

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?

- ( ) 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.
- ( ) 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 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.

- [x] Admissible
- [ ] Composite
- [ ] Useful for reducing search
- [ ] Based on a relaxation
- [x] Consistent

Question 6 — 1 pts

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.

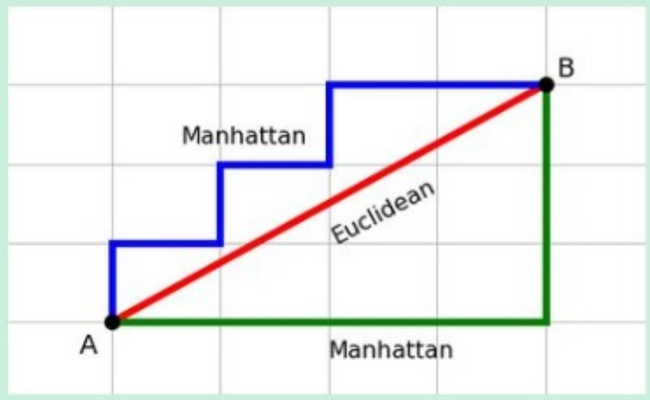
Question 7 — 1 pts

If the heuristic is inadmissible, which of these algorithms is guaranteed to produce the optimal solution?

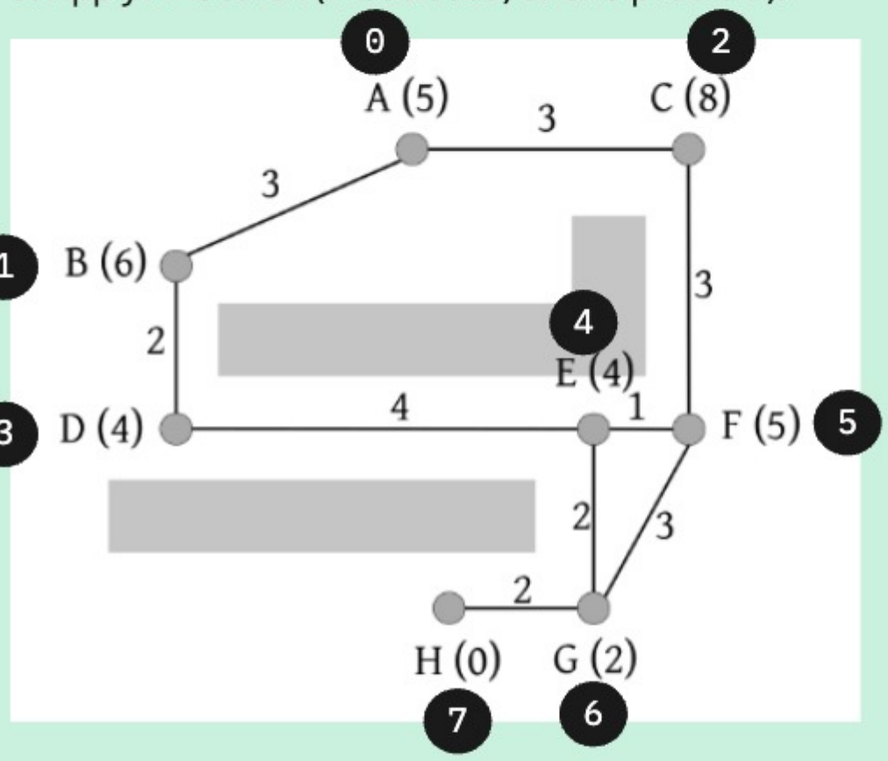
- ( ) Both Greedy Best-First Search and A* Search
- ( ) Greedy Best-First Search but not A* Search
- (●) Neither Greedy Best-First Search nor A* Search
- ( ) A* Search but not Greedy Best-First Search

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**.



1. Apply Greedy Best First Search (Graph Based) to this problem. Is this optimal?
2. Apply A* Search (Graph Based) to this problem, note the frontier in each step and the f-cost.
3. 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

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.

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

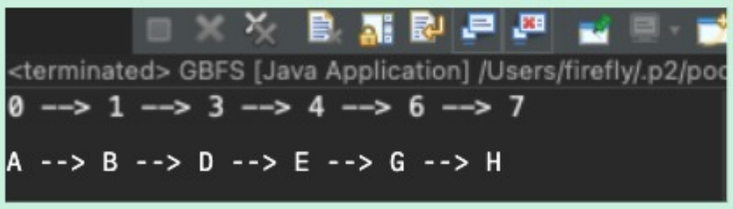
B → Explored Nodes = [A, B]
Frontier = [C(8), D(4)]
D is selected for expansion

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



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

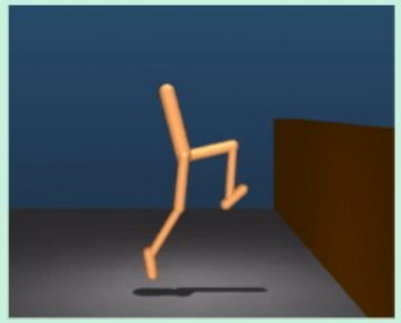
End: Explored Nodes = [A, B, D, C, F, G, H]
H is the Goal

Unlikely traversal, would more likely have been messier.

Heuristics

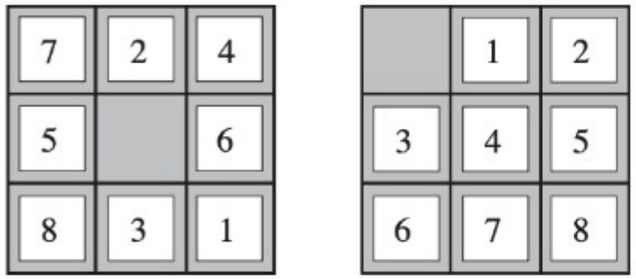
Designing a Heuristic:

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:

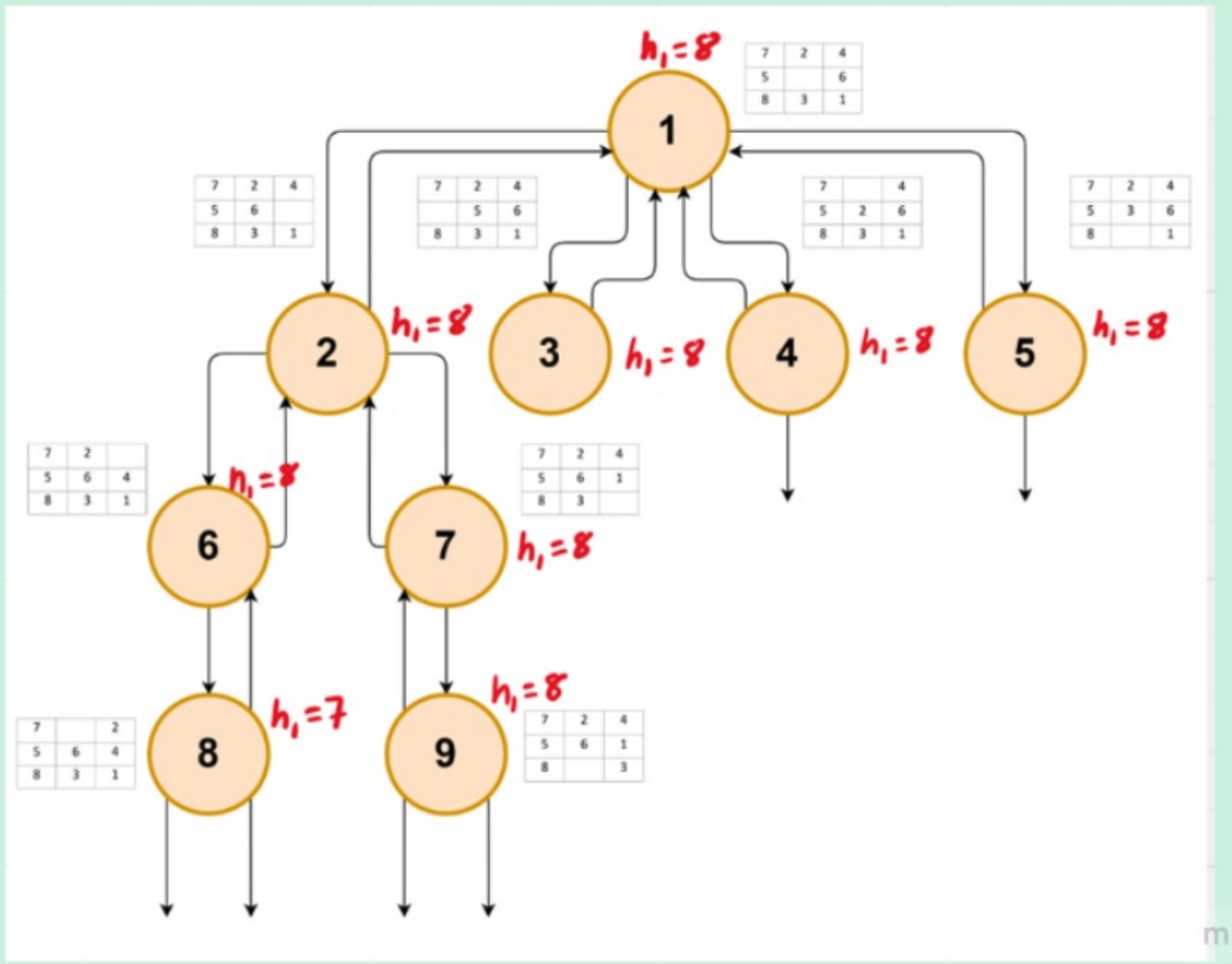


There are multiple ways of relaxing the 8 puzzle.

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 h1. 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 h2 is always larger than or equal to h1, and since they are both admissible then h2 must be more accurate than h1. 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.

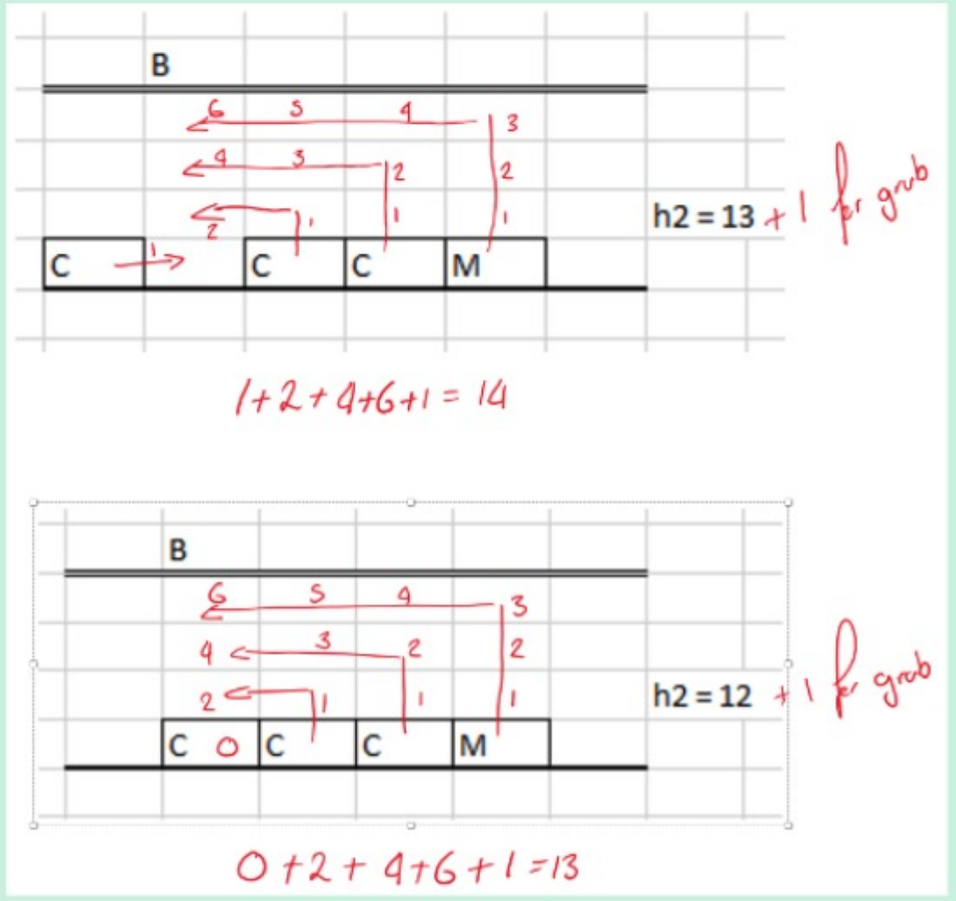
States: The state is determined by the position of the crates, the position of the monkey, and whether or not the monkey has grabbed the bananas. The state space is difficult to estimate without defining the width of the floor.

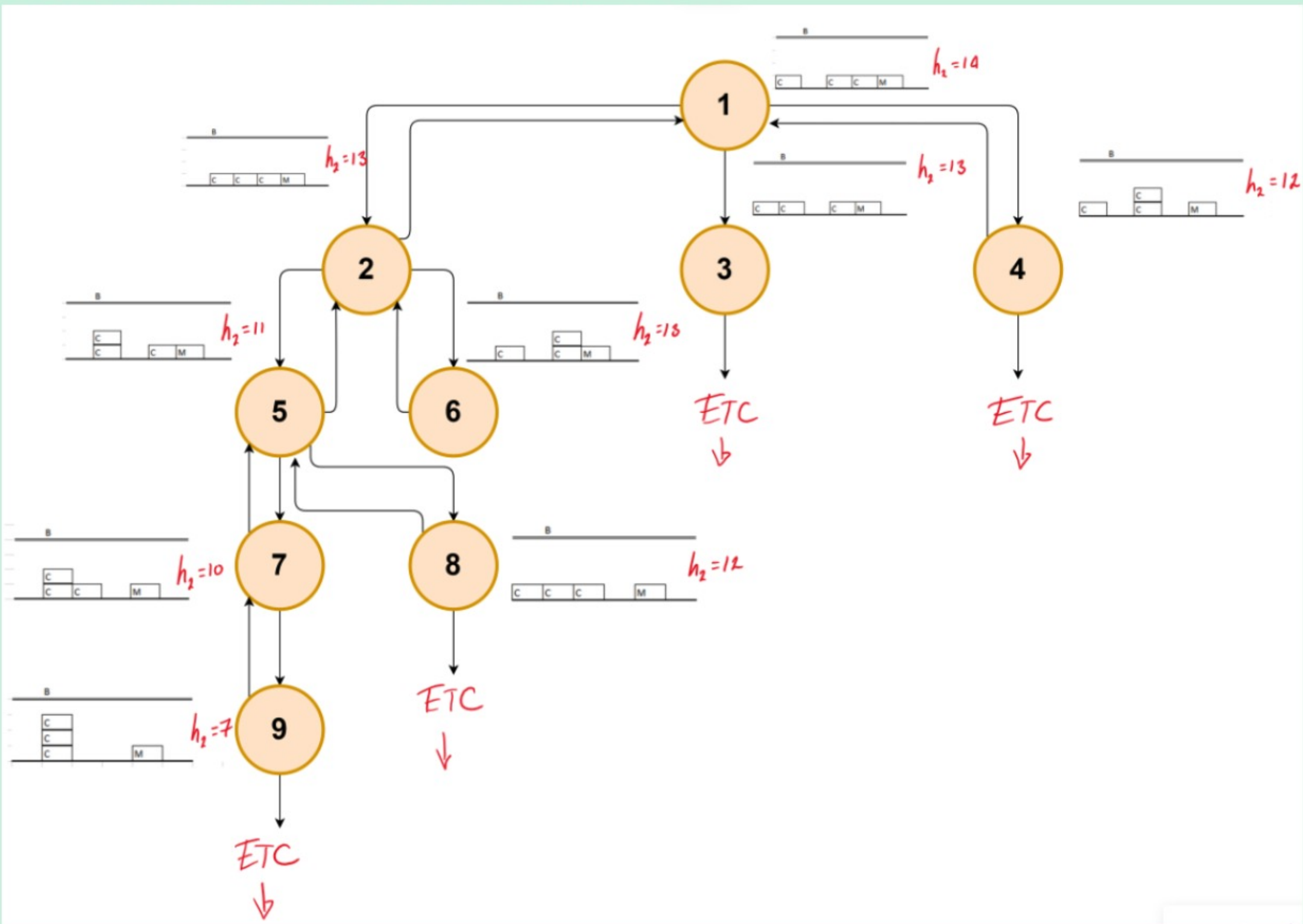
Actions: When on the ground, a crate has the possible actions of: Left, Right, and Stack. Once stacked, a crate cannot move. When on the ground the monkey has the possible actions of Left, Right, Climb. When on the stack, the monkey cannot move left or right, but has the possible action of "Grab". The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

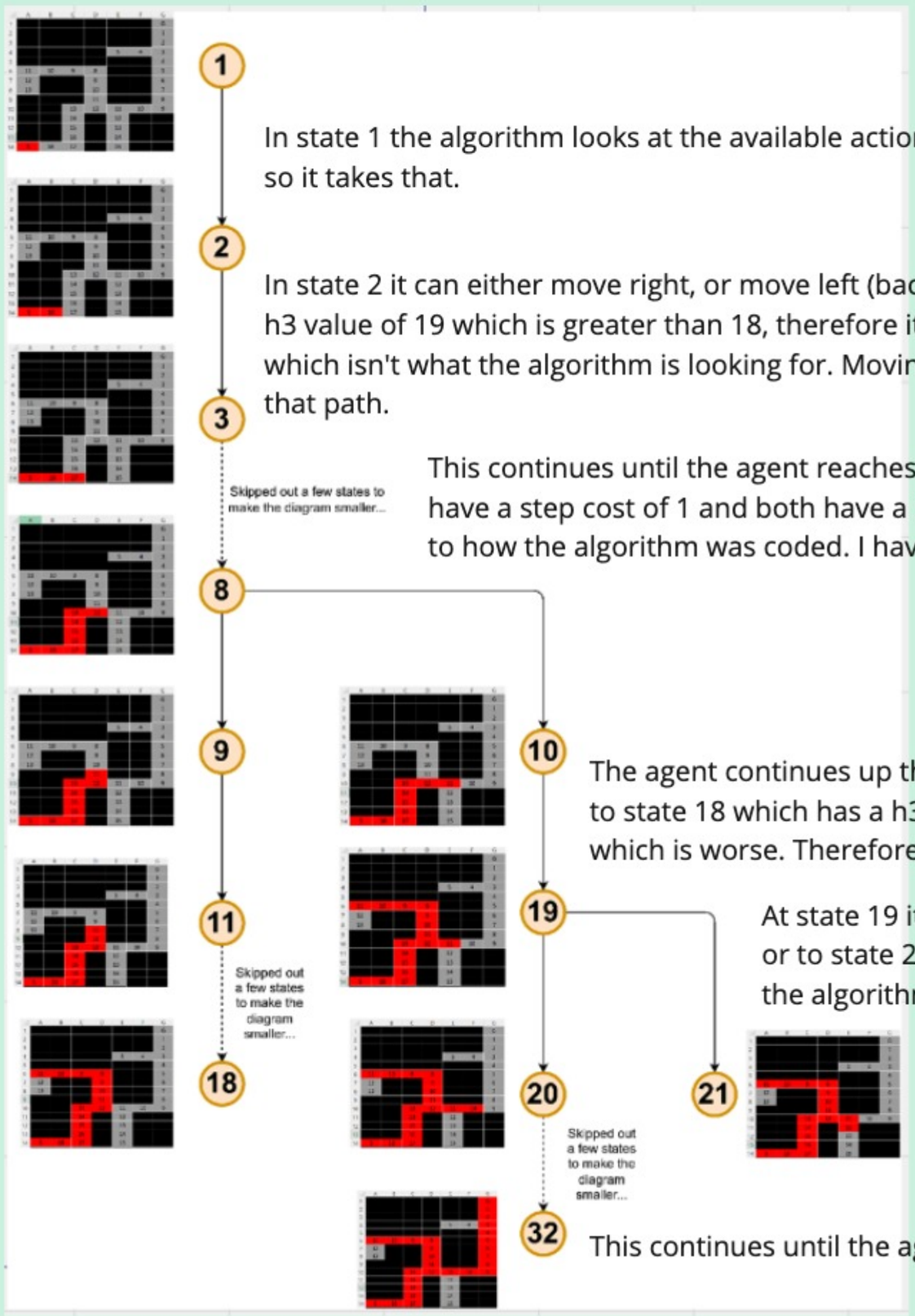
Transitional Model: The actions have their expected effects, moving the agents in the respective direction in the maze, except where that movement is blocked by a wall or the edge of the maze.

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 h3 = Agent distance from goal, represented here by the Manhattan Distance.



In state 1 the algorithm looks at the available actions, as it turns out there is only 1 so it takes that.

In state 2 it can either move right, or move left (back where it was). Moving left has a h3 value of 19 which is greater than 18, therefore it is moving away from the goal which isn't what the algorithm is looking for. Moving right has an h3 of 17 so it takes that path.

This continues until the agent reaches state 8. The agent is faced with a choice, both have a step cost of 1 and both have a h3 of 11, therefore the route decision is down to how the algorithm was coded. I have chosen left first.

The agent continues up the path and h3 is decreasing, therefore this seems a logical choice, until it gets to state 18 which has a h3 of 11 (the same as it was earlier in the maze). The next state has a h3 of 12 which is worse. Therefore, the algorithm backtracks to state 10.

At state 19 it is again faced with a choice: move to state 20 which has a h3 of 10, or to state 21 which has a h3 of 12. H3 of 10 suggests closer to the goal and so the algorithm takes that path.

This continues until the agent reaches the goal.

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**)

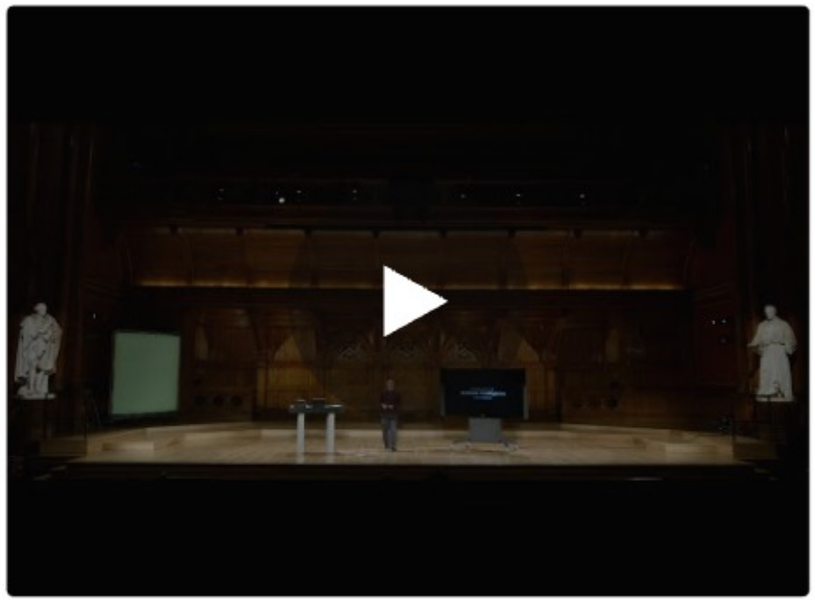
Study Session:

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

Propositional Logic - Facts and the relationship between facts.

Propositional Symbols: P, Q, R

P	Q	¬P	¬Q	P ^ Q	P v Q	P ⇒ Q	P ⇔ 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": ¬
- and: **^**
- or: **v**
- implication, "if/then" e.g. if P is true then Q is true: → or **⇒**
- bi-conditional, iff = "if and only if": ↔ or **⇔**
- entailment (aka semantically entails), "means": ⊨ **(double turnstile)**
  - P ⊨ Q = in every model, every possible world, in which P is true, Q is also true.
- inference (aka syntactically entails), "proves": ⊢ **(turnstile)**
  - ⊢ P = given P
  - P ⊢ Q = from P, I know that Q -or- Q is provable from P
  - KB ⊢ *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 i… by KB, it knows everything.
- **(!) If the KB is true in the real world then any sentence S d… from KB by a sound inference procedure is also true in th… world.**

Grounding:

- Grounding captures the connection between logical reasonin… processes and the real environment in which the agent exists
- The agents sensors create a connection between the world a…
- The meaning and truth of percept sentences are defined by t… processes of sensing and sentence construction that produce…
- Learning is fallible. Our observation of a particular fact may n… generalize.

Lite… prop… sym… its c…

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: $((p \implies q) \implies p) \implies p$

p	q	$p \implies q$	$(p \implies q) \implies p$	$((p \implies q) \implies p) \implies p$
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(KB) ⊆ M(*s*)

These become the proofs for the subsequent laws

Figure 2.2: Truth tables for logical connectives

p	q	$p \implies q$
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	$p \iff q$
0	0	1
0	1	0
1	0	0
1	1	1

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

p	q	$\neg q$	$p \implies \neg q$	$\neg(p \implies \neg q)$
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 $\neg(p \implies \neg q)$ is the same as for $p \wedge q$.

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

p	q	$\neg p$	$\neg p \implies q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

Notice that the truth table for $\neg p \implies q$ 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 ¬ inwards using De Morgan's Laws
- Use distributive law to distribute ors (v) wherever possible.

Example:

$(p \vee q) \Rightarrow r$

$\neg(p \vee q) \vee r$ implication elimination

$(\neg p \vee \neg q) \vee r$ De Morgan's Law

$(\neg p \vee r) \wedge (\neg q \vee r)$ 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.

And Elimination
the elimination of *and*

$$\frac{p \wedge q}{p}$$

if p and q are true
$\therefore$
p is true

Modus Ponens
the application of *implication*

$$\frac{p \Rightarrow q \quad p}{q}$$

if p implies q
and p is true
$\therefore$
q is true

Associative Laws
removal of parentheses

$p \wedge q \wedge r \equiv ((p \wedge q) \wedge r) \equiv (p \wedge (q \wedge r))$

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

Biconditional Elimination
the application of *biconditionals*

$$\frac{p \Leftrightarrow q}{(p \Rightarrow q) \wedge (q \Rightarrow p)}$$

if p if and only if q
$\therefore$
either p or q is true

Hypothetical Syllogism

$$\frac{p \Rightarrow q \quad q \Rightarrow r}{p \Rightarrow r}$$

if p implies q
and q implies r
$\therefore$
p implies r

Resolution Inference Rule

$$\frac{p \vee q \quad \neg p \vee r}{q \vee r}$$

Implication Elimination
turning *implication* into *or*

$$\frac{p \Rightarrow q}{\neg p \vee q}$$

if p implies q
$\therefore$
either p or q is true

De Morgan's Laws
turning *and* into *or*

$$\frac{\neg(p \wedge q)}{\neg p \vee \neg q}$$

$$\frac{\neg(p \vee q)}{\neg p \wedge \neg q}$$

Unit Resolution Rule

$$\frac{p \vee q \quad \neg p}{q}$$

if p or q is true
and p is false
$\therefore$
q is true

Contraposition

$$(p \Rightarrow q) \equiv (\neg p \Rightarrow \neg q)$$

Distributive Laws
applying algebra to logic

$$\frac{(p \wedge (q \vee r))}{(p \wedge q) \vee (p \wedge r)}$$

$$\frac{(p \vee (q \wedge r))}{(p \vee q) \wedge (p \vee r)}$$

Double Negation Elimination
the removal of *not*

$$\frac{\neg(\neg p)}{p}$$

if p is not false
$\therefore$
p is true

Idempotent Laws

$$p \vee p \equiv p$$

$$p \vee p \equiv p$$

Commutative Laws

$$(p \wedge q) \equiv (q \wedge p)$$

$$(p \vee q) \equiv (q \vee p)$$

4.4 Quiz: Principles of logic and propositional logic

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 1 1 pts

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

- 12
- (•) 16
- 8
- 2

Question 2 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	¬C	A v ¬C	A ^ ¬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	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 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

Question 4 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	¬A	A v ¬A	A ^ ¬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
FALSE	TRUE	TRUE	FALSE

Question 5 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

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

Question 6 1 pts

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

- 3
- 4
- (•) 8
- 16

Question 7 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 = looks like a duck
Q = quacks like a duck
D = is a duck

L ^ Q ⇒ D

∴

¬(L ^ Q) v D **Implication Elimination**

∴

¬L v ¬Q v D **De Morgan's Law**

Question 8 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.

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

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 = false satisfies the left-hand side and not the right-hand side.

Need help with this one

Question 9 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. $S_1 \vdash S_2$)

A **sound** inference algorithm is always correct, it is not always complete. Therefore S2 *may* be inferred from S1 but that is not guaranteed.

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 ⇒ ¬Mortal
S2: ¬Mythical ⇒ Mortal ∧ Mammal
S3: ¬Mortal ∨ Mammal ⇒ Horned
S4 Horned ⇒ Magical

S1: Mythical ⇒ ¬Mortal
Using **contraposition**: ¬(¬Mortal) ⇒ ¬Mythical
double negation elimination: Mortal ⇒ ¬Mythical

S2: ¬Mythical ⇒ Mortal ∧ Mammal
Using **hypothetical syllogism**: Mortal ⇒ Mortal ∧ Mammal
Using **implication elimination**: ¬Mortal ∨ (Mortal ∧ Mammal)
Using **distributive law**: (¬Mortal ∨ Mortal) ∧ (¬Mortal ∨ Mammal)
Eliminate the **tautology**: ¬Mortal ∨ Mammal

S3: ¬Mortal ∨ Mammal ⇒ Horned
Using **modus ponens** conclude: Horned

S4: Horned ⇒ Magical
Using **modus ponens** conclude: Magical

Note: Can't prove Mythical

York Solution:

S1: Mythical ⇒ Immortal
S2: ¬Mythical ⇒ ¬Immortal ∧ Mammal
S3: Immortal ∨ Mammal ⇒ Horned
S4 Horned ⇒ 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⇒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.

Propositional Theorem Proving

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*** **≡** ***q*** **iff** ***p*** **⊨** ***q*** **and** ***q*** **⊨** ***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*** ∨ ***¬p*** (the law of the excluded middle).
        - *Not to be confused with p ∧ ¬p which is an empty set and always resolves to false.*

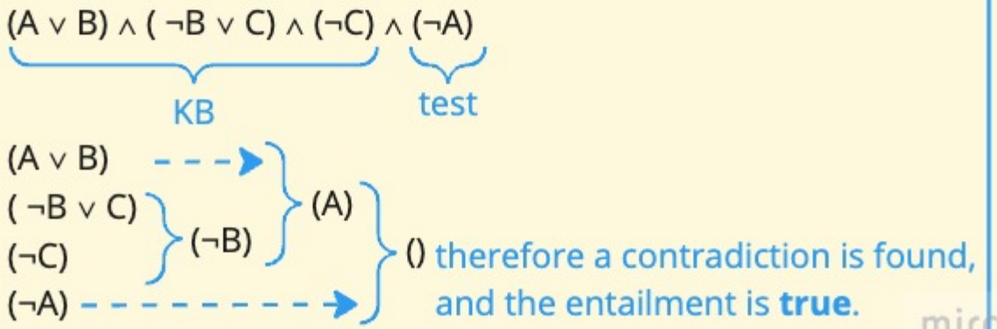
Deduction Theorem

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

Satisfiability

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

Proof by Contradiction: Does (A ∨ B) ∧ (¬B ∨ C) ∧ (¬C) entail A?



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:

$$\underbrace{(A \vee B)}_{\text{clause}} \wedge \underbrace{(\neg B \vee C)}_{\text{clause}} \wedge \underbrace{(\neg C)}_{\text{clause}}$$

-OR- inside the clause
-AND- joining the clauses together

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

- 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 ¬A) is unsatisfiable, in which case KB ⊨ A.
- This process was manually detailed in the proof by contradiction above.

To determine if KB ⊨ A:

The basics:

- Check if (KB ∧ ¬A) is a contradiction?
  - Empty Clause is A ∧ ¬A
  - If so, then KB ⊨ A
  - Otherwise, no entailment.

Space & Time complexity

General pseudocode (from CS50 lecture 1):

- Convert (KB ∧ ¬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 ⊨ A.
- Otherwise, if we can't add new clauses, no entailment.

Algorithm pseudocode (from the text):

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  while true do
    for each pair of clauses C_i, C_j in clauses do
      resolvents ← PL-RESOLVE(C_i, C_j)
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

- 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.

Horn Clauses and Forward Chaining

Definite Clause

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

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.

Why do we care?

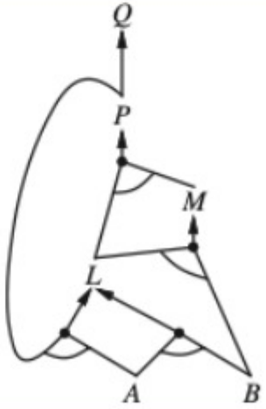
- Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
  - e.g. $(\neg p \vee \neg q \vee r)$ can be written as $(p \wedge q) \Rightarrow r$
- Inference with horn clauses can be done through **forward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

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Horn Clauses

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**.
- Runs in **linear time**
- Is **sound**, i.e. it only derives sentences entailed by KB
- Is complete; every atomic sentence which is entailed by KB will be derived.
- In fact, the set of atomic sentences inferred at the fixed point defines a model of the original KB.
- 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 And-Or graph

Backward Chaining

General:

- The second major family of logical inference algorithms uses backwards chaining over definite clauses.
- These algorithms work backwards from the query, 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 the Forward Chaining (FC) graph but we work backwards from q.
- Like FC, an efficient implementation of BC runs in **linear time**.
- BC is an example of **goal-directed reasoning**. 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.

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.
- ∀ **= for all**
- ∃ **= there exists**

Optional Activity 2

Selections:	Rules:	Horn Clauses (max 1 positive):
A: 2.4GHz CPU	(A ∨ B)	(¬A ∨ ¬B)
B: 3.0GHz CPU	(C ∨ D)	(¬C ∨ ¬D)
C: 16GB RAM	(E ∨ F)	(¬E ∨ ¬F)
D: 32GB RAM	(J ∨ K)	(¬J ∨ ¬K)
E: 500GB SSD	G ⇒ B	(¬G ∨ B)
F: 1TB SSD	B ⇒ H	(¬B ∨ H)
G: Discrete GPU	A ∧ D ⇒ H	(¬(A ∧ D) ∨ H) ≡ (¬A ∨ ¬D ∨ H)
H: CPU/Memory run Hot	G ⇒ I	(¬G ∨ I)
I: GPU/Disk run Hot	F ⇒ I	(¬F ∨ I)
J: Slim Case	H ∧ I ⇒ K	(¬(H ∧ I) ∨ K) ≡ (¬H ∨ ¬I ∨ K)
K: Large Case	D ∧ F ⇒ K	(¬(D ∧ F) ∨ K) ≡ (¬D ∨ ¬F ∨ 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]
false
```

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

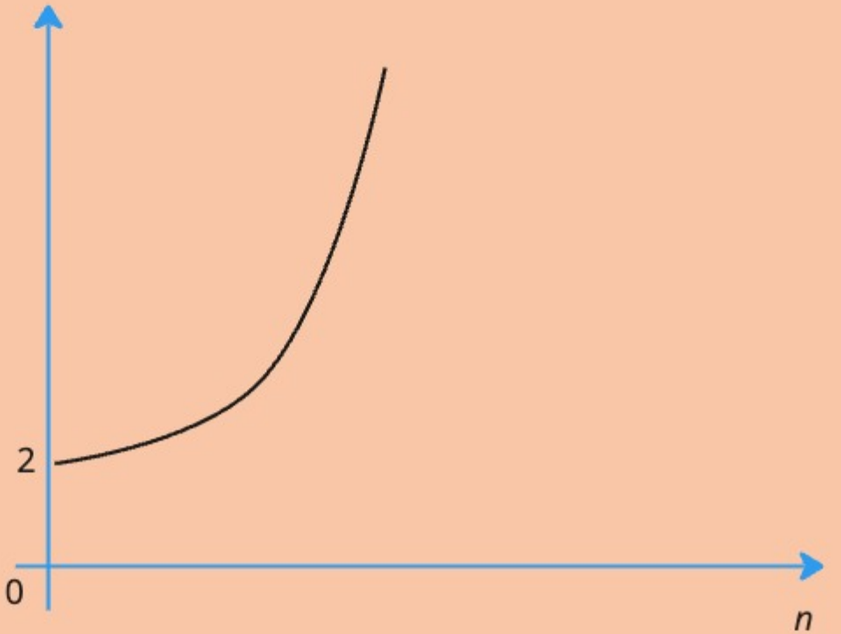
A formula / sentence in propositional logic is true or false with respect to a model.

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.

If there is more than one such model then an algorithm for SAT can return any one of them to prove satisfiability.

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 and satisfiability are connected:

- A is valid iff ¬A is unsatisfiable
- Contra-positively, A is satisfiable iff ¬A is not valid
- A ⊨ B if and only if the sentence (A ∧ ¬B) is unsatisfiable.

Motivation:

- Proving entailment: Recall that proving (KB ⊨ α) is equivalent to proving the unsatisfiability of (KB ∧ ¬α)
- Practical applications:
  - product configuration
  - hardware verification
  - software verification
  - planning
  - scheduling
  - proofs in mathematics

Detour: NP-completeness

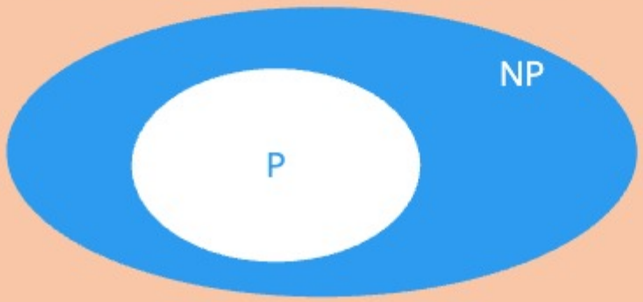
Classes of problems:

- **P** - problems which can be solved in polynomial time
- **NP** - problems which can be verified in polynomial time (if we were given a "certificate")
- **NPC** (NP-complete) - 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.

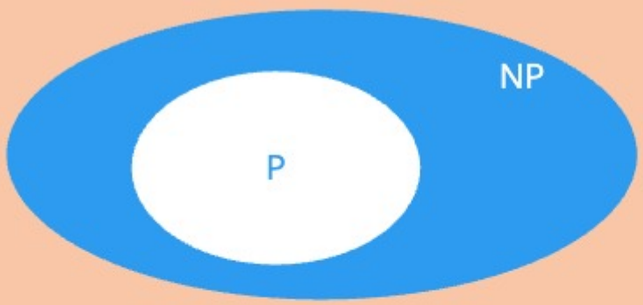
If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time algorithm.

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.

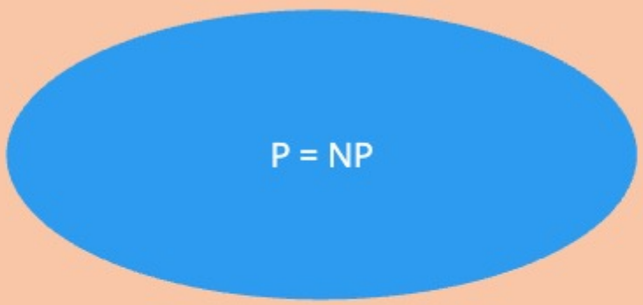
What we know (P is a subset of NP since we can solve the problems in P time without the need for a "certificate"):



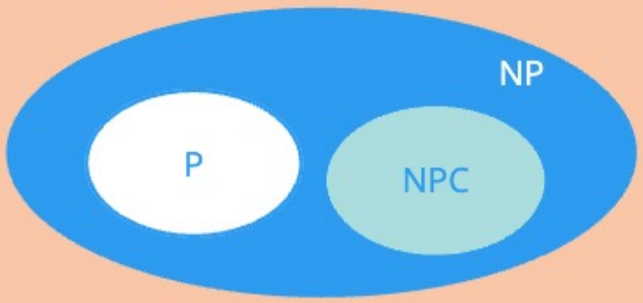
What we want to know: if P is a proper subset of NP?



or



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

Using SAT for decision problems:
SAT algorithms have been researched for decades and are now very efficient (even though their **worst-case time bound is still exponential, O(2^n)**). As a result, some decision problems are solved using SAT.

3 steps to solving decision problems using SAT:

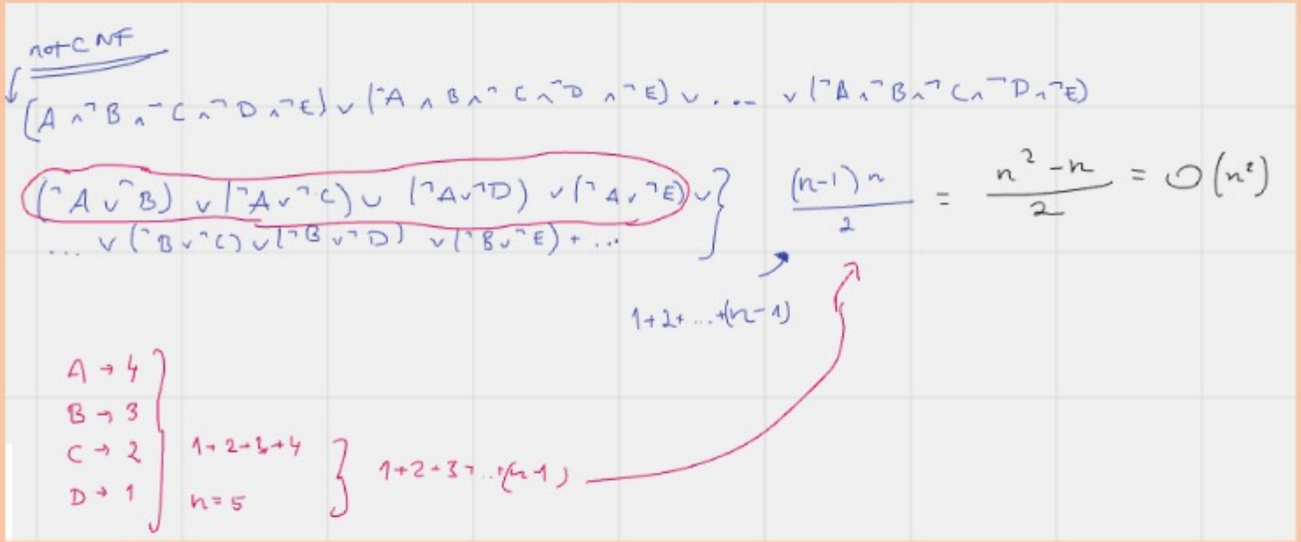
1. Represent the problem as a propositional logic formula.
2. Solve the problem using a SAT algorithm.
3. If the SAT algorithm has found a model, translate the model into a solution to the original decision problem.

Representing a problem using propositional logic may require us to impose restrictions such as **"at most one"** or **"at least one"** - i.e. the sudoku example.

For example: If we have 2 propositions, A and B:

- In order to impose the restriction that at least one of them is true we would impose (A ∨ B). This requires just one restriction.
- In order to impose the restriction that at most one of them is true, we would impose (¬A ∨ ¬B).
    - Note that this restriction can be represented in O(n^2) in size for *n* proposition symbols.

Exercise:
We have 5 propositions: A, B, C, D, E and we want at most one of them to be true.
(a) How can we write this in CNF?
(b) What would change if we also wanted to ensure that at least one of them is true? (A ∨ B ∨ C ∨ D ∨ E)



5.4 Quiz: The satisfiability problem in propositional logic

Question 1 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 2 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

Question 3 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

Question 4 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 $F \wedge \neg\alpha$

A ⊨ B if and only if the sentence (A ∧ ¬B) is unsatisfiable.

Question 5 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

There are (6) pairs of proposition symbols, each producing one clause in the at-most-one. At-least-one always has 1 clause.

Question 6 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.

Question 7 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.

- [x] 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.
- [x] Finding a way through a maze where the layout of the maze is known in advance.
- [x] Deciding how to assign tasks to workers and machines in a factory.

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

DPLL - A backtracking algorithm for SAT

General Idea:

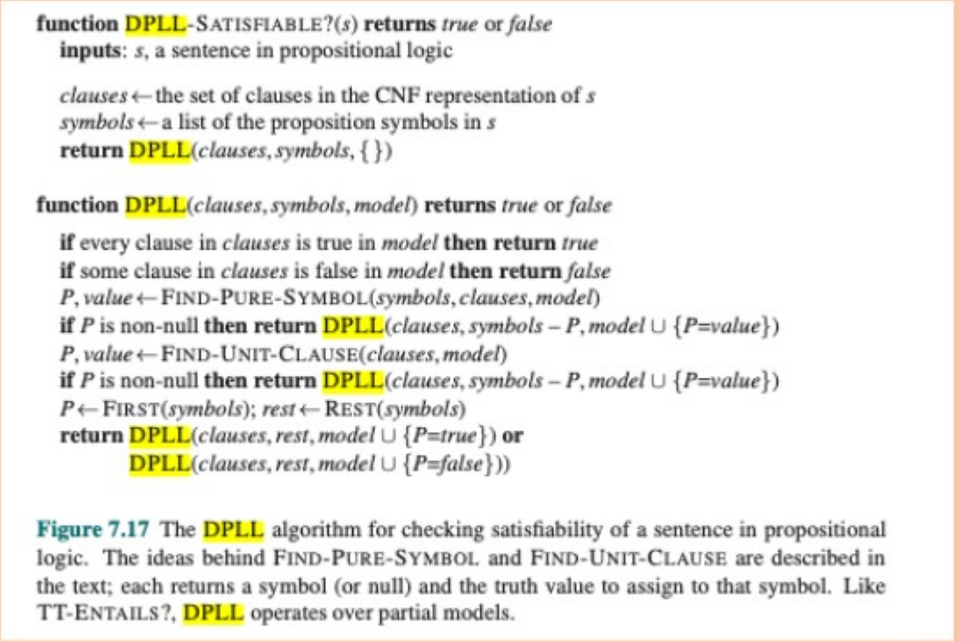
- Named after the initial authors: Davis, Logemann, and Loveland
- We take as input a sentence in CNF
- We construct a model that satisfies the formula by building it step-by-step, assigning one symbol at a time to True or False
- This is essentially a recursive, depth-first enumeration of all possible models, analagous 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.

Properties:

- Input in CNF
- **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 (reasoning as above).

Notes about the Pseudocode:

- We assign the value of True before assigning False.
- The pure symbol rule goes **before** the unit clause rule
- We only assign the truth after examining pure clauses and unit clauses (I think??)



```
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) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

Figure 7.17 The DPLL algorithm for checking satisfiability of a sentence in propositional logic. The ideas behind FIND-PURE-SYMBOL and FIND-UNIT-CLAUSE are described in the text; each returns a symbol (or null) and the truth value to assign to that symbol. Like TT-ENTAILS?, DPLL operates over partial models.

DPLL 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 model. This allows us to avoid examining entire subtrees of the search space.
- Example: $(B \vee S) \wedge (A \vee C)$ **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.
    - for example: C in $(A \vee C) \wedge (B \vee C)$
    - for example: $\neg C$ 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

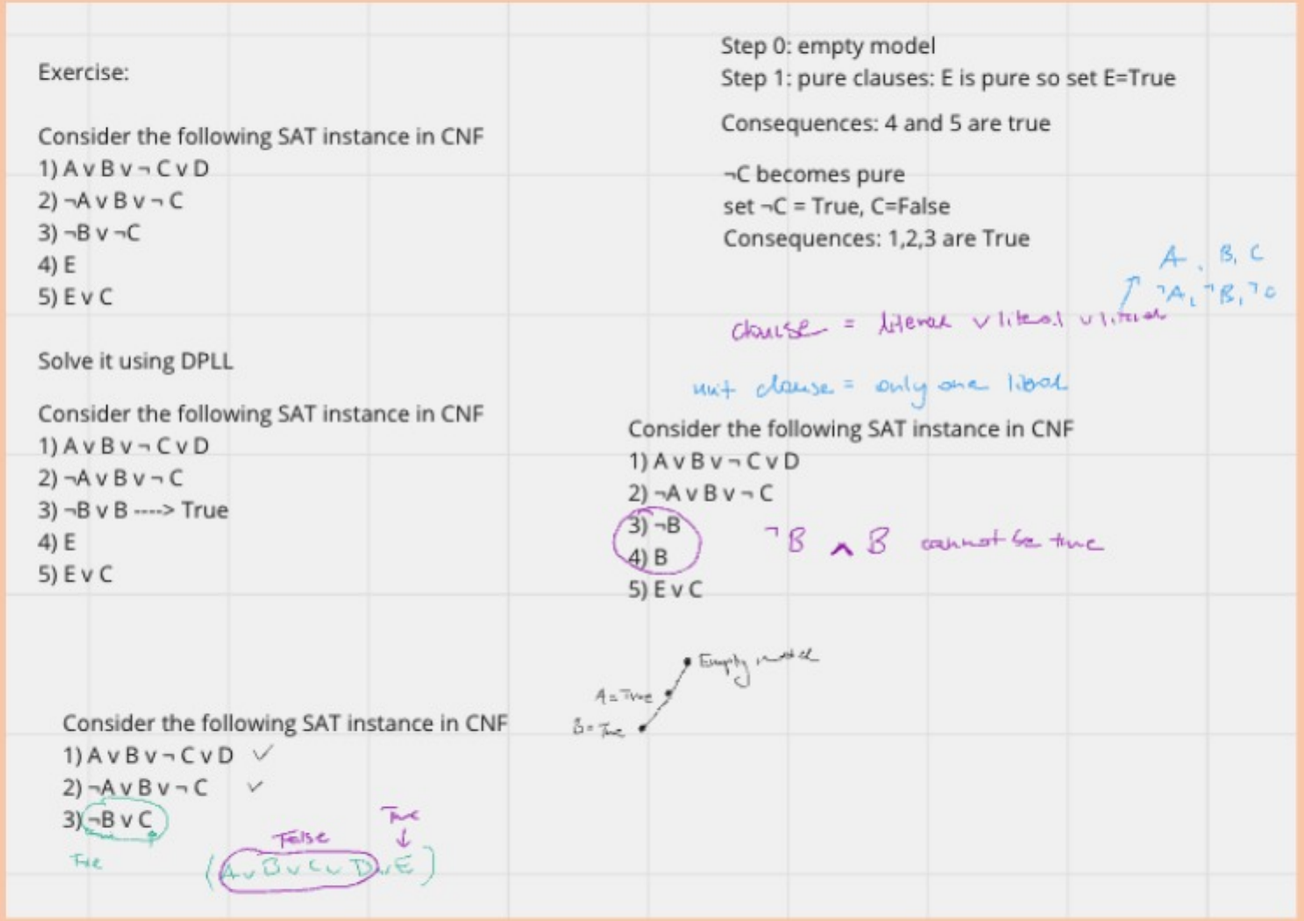
- A unit clause is a clause with just one literal
    - for example: A
    - for example: $\neg A$
- In DPLL a unit clause is also a clause in which all literals but one are already assigned to False by the model.
    - for example: $(\neg A \vee \neg B \vee \neg C \vee D)$, if we assigned A=True, B=True, C=True, then D becomes a unit clause.
- 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.

DPLL Exercise

Consider the following SAT instance in CNF:

1. **A ∨ B ∨ ¬C ∨ D**
2. **¬A ∨ B ∨ ¬C**
3. **¬B ∨ ¬C**
4. **E**
5. **E ∨ C**

Solve it using DPLL



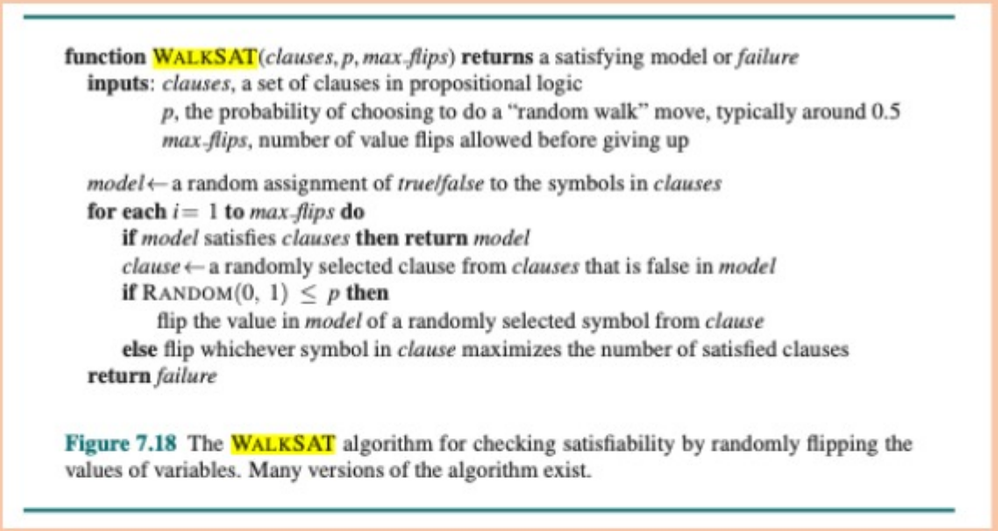
WalkSAT - An incomplete, randomised algorithm for SAT

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 teh 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 reach (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.
- Complimentary to DPLL: some SAT instances which are difficult for DPLL are easy for WalkSAT and vice versa.



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 ← 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 ← 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.

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.

WalkSAT Exercise

Consider the following SAT instance in CNF:

1. **A ∨ B ∨ ¬C ∨ D**
2. **¬A ∨ B ∨ ¬C**
3. **¬B ∨ ¬C**
4. **E**
5. **E ∨ C**

Initial Model (randomly assigned): A = True, B = False, C = False, D = False, E = False

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

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

- Evaluate clauses:
  - Satisfied Clauses: 1, 2, 3
  - Unsatisfied Clauses: 4, 5
- Pick a clause at random from unsatisfied clauses:
  - 5. E ∨ C
- Have a lottery with probability p=1/2 whether we are going to choose a symbol at random or in a greedy fashion.
  - At random: C
- Flip selected symbol
  - C = True

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

- Evaluate clauses:
  - Satisfied Clauses: 1, 3, 5
  - Unsatisfied Clauses: 2, 4
- Pick a clause at random from unsatisfied clauses:
  - 2. ¬A ∨ B ∨ ¬C
- Have a lottery with probability p=1/2 whether we are going to choose a symbol at random or in a greedy fashion.
  - Greedy Choice:
    - If we flip A = False, then 2 & 5 are satisfied
    - If we flip B = True, then 1, 2, & 5 are satisfied
    - If we flip C = False, then 1, 2, & 3 are satisfied
- Flip selected symbol
  - B = True

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

- Evaluate clauses:
  - Satisfied Clauses: 1, 2, 5
  - Unsatisfied Clauses: 3, 4
- Pick a clause at random from unsatisfied clauses:
  - 4. E
- Flip selected symbol
  - E = True

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

- Evaluate clauses:
  - Satisfied Clauses: 1, 2, 4, 5
  - Unsatisfied Clause: 3
- Greedy Choice:
  - If we flip B = False, then 1, 3, 4, & 5 are satisfied
  - If we flip C = False, then 1, 2, 3, 4, 5 are satisfied
- Solution found:

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.

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: 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

Learning

Types of Learning

Types of learning depend on the 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
- **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 decided which of the actions prior to the reinforcement were most responsibile for it and alters its actions to aim towards more rewards in the future.

Factored representation: Each example provided to the ML agorithm is represented as a sequence of attribute (feature) values.

We can think of a model in propositional logic as being a type of factored representation.

This is in contrast to, e.g. AI search problems, in which the states are atomic - 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.

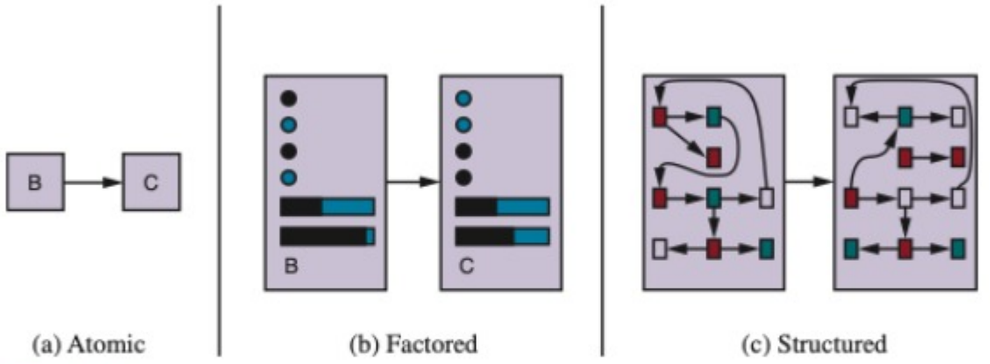
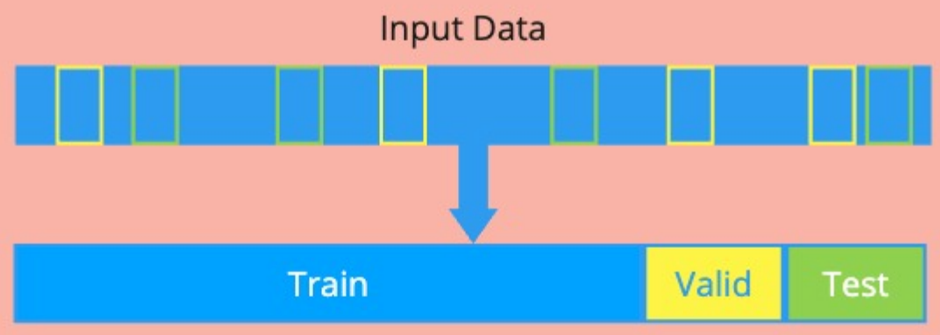


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

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*

(x1, y1), (x2, y2), ... (xn, yn),

*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 general 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.
- **Consistency** - When the hypothesis agrees with all of the data.

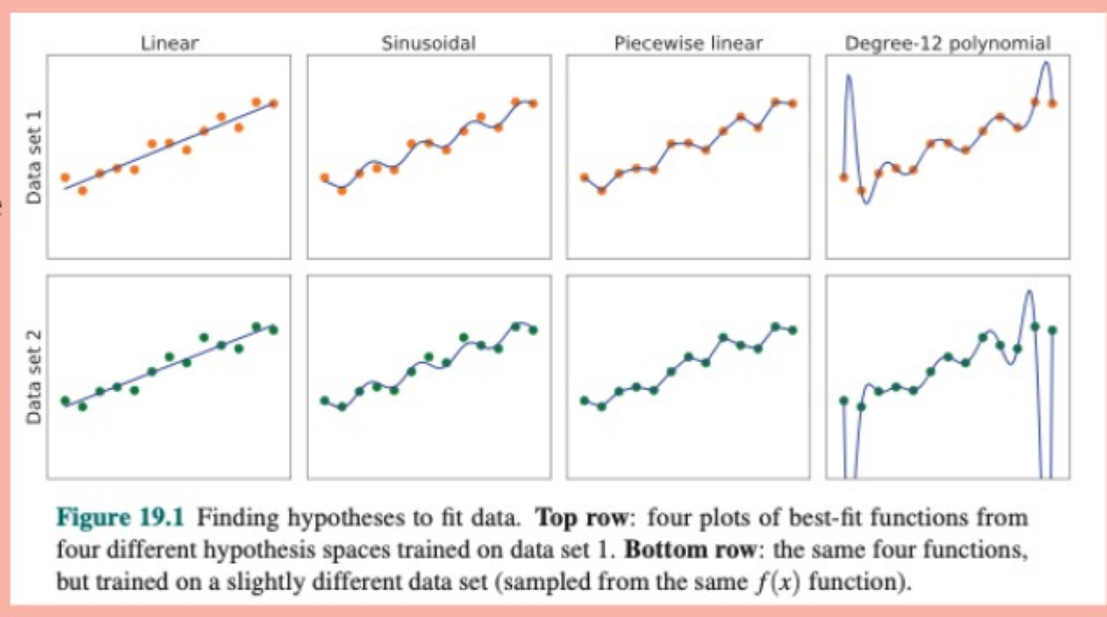


Figure 19.1 Finding hypotheses to fit data. **Top row**: four plots of best-fit functions from four different hypothesis spaces trained on data set 1. **Bottom row**: the same four functions, but trained on a slightly different data set (sampled from the same $f(x)$ function).

Choosing a hypothesis:

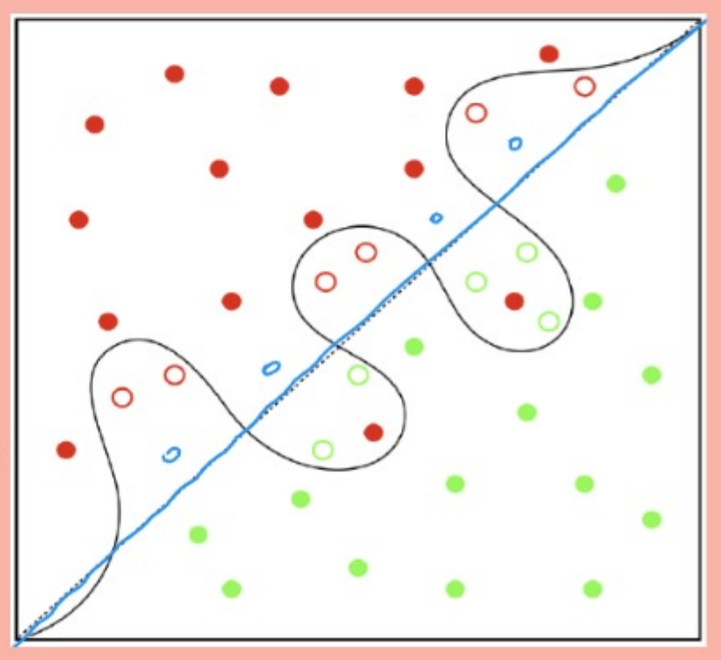
- We may have more than one hypothesis consistent with the data, in which case we want to choose the simplest one (Occam's razor).
- There is a trade-off: more complex hypotheses may fit the training better, but generalise less well to unseen examples; overfitting.
- There is also a trade-off between searching through a space of very expressive hypotheses (e.g. Turing machines), which increases the complexity of finding a good hypothesis; versus restricting the search to the space of less expressive hypotheses (which are less complex to search along).
- **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:

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

By Bayes' rule this is equivalent to

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

  - 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.

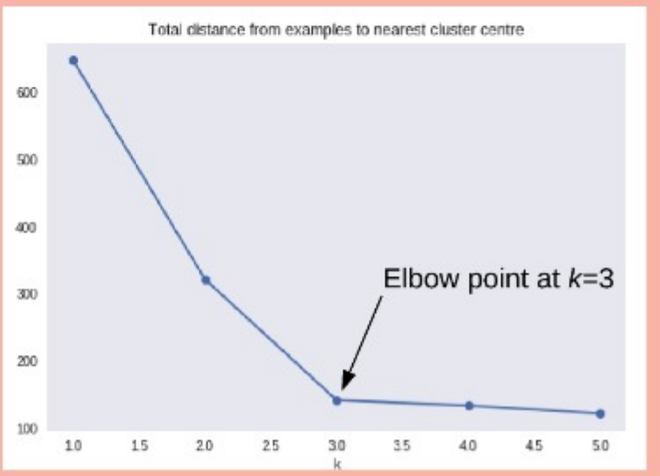
Reinforcement Learning

- 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.

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.

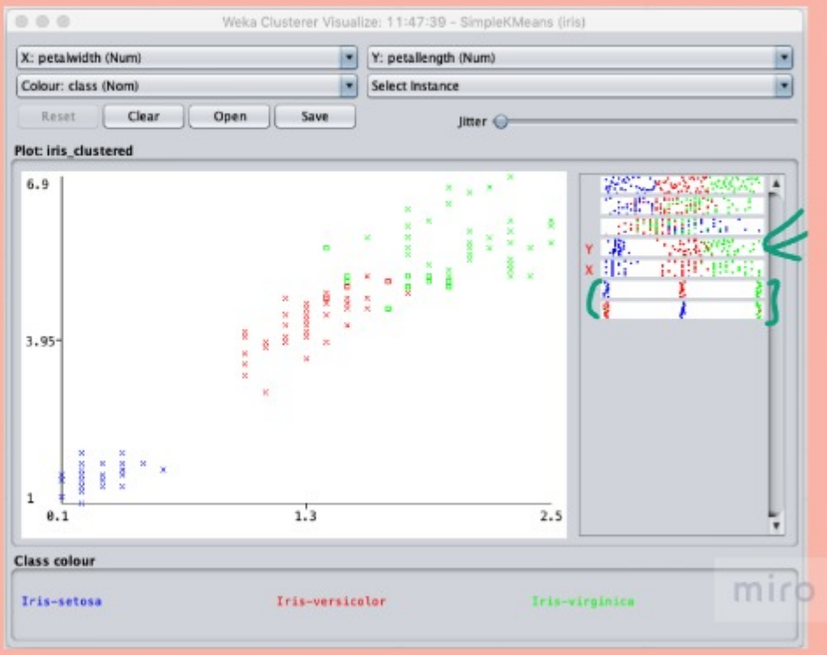


K-Means Algorithm:

1. Choose K cluster centers at random.
2. Allocate each point int he 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:

1. 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.
2. The final cluster allocation is sensitive to the initial (random) allocation of clusters.
3. 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.

Example Activity

Example Activity: suppose we have the following 10 data points, with initial cluster centre assignment of C1=(0.0 , 0.0) and C2 = (10.0, 10.0). What will be the cluster centres after the first iteration?

Observation	X-coordinate	Y-coordinate
1	1.2	3.2
2	2.3	4.5
3	6.7	8.9
4	8.9	6.7
5	4.6	4.6
6	5.1	5.1
7	2.6	7.8
8	10	10
9	2.6	9.5
10	6.7	3.4

$(1.2-0.0)^2 + (3.2-0.0)^2 = 11.68$ ←

$(1.2-10.00)^2 + (3.2-10.0)^2 = 123.68$

$\frac{1.2+2.3+4.6}{3}$

C1

$\frac{3.2+4.5+4.6}{3}$

X-coord. Y-coord. Z-coord. V-coord.

$(1.2-0)^2 + (3.2-0)^2 + (1.0-0)^2 + (3.0-0)^2$

Observation	X-coordinate	Y-coordinate	Z coordinate	V-coordinate
1	1.2	3.2	1.0	3.0
2	2.3	4.5		
3	6.7	8.9		
4	8.9	6.7		
5	4.6	4.6		
6	5.1	5.1		
7	2.6	7.8		
8	10	10		
9	2.6	9.5		
10	6.7	3.4		

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)*

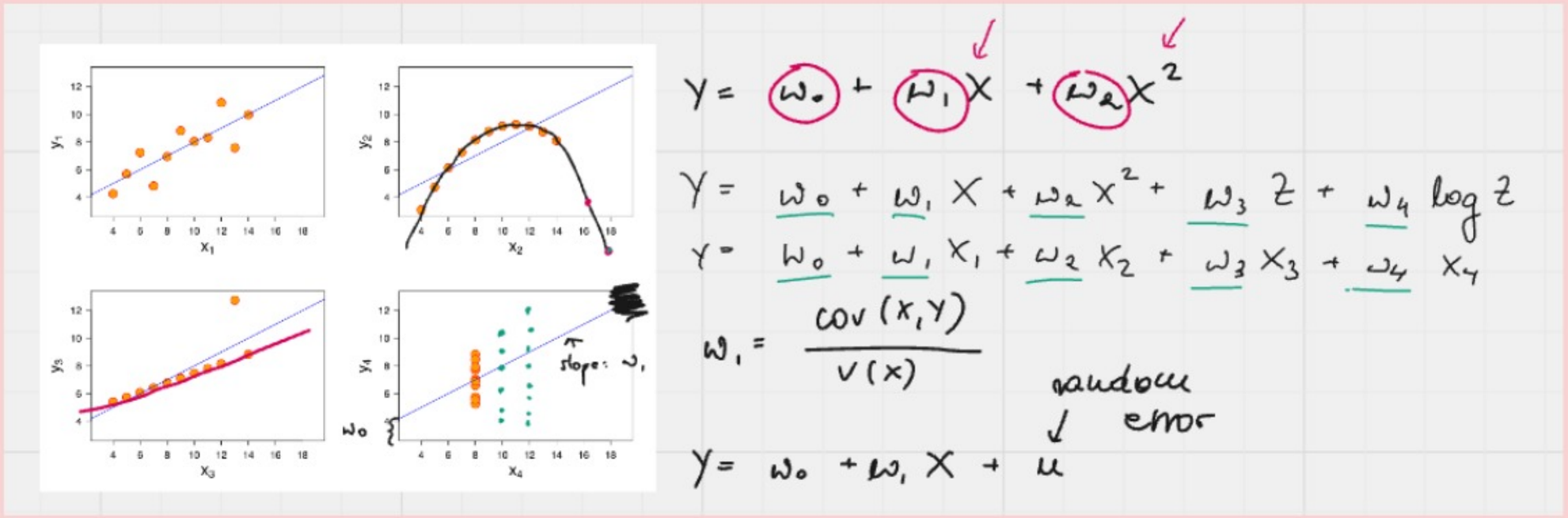
Study Session:

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

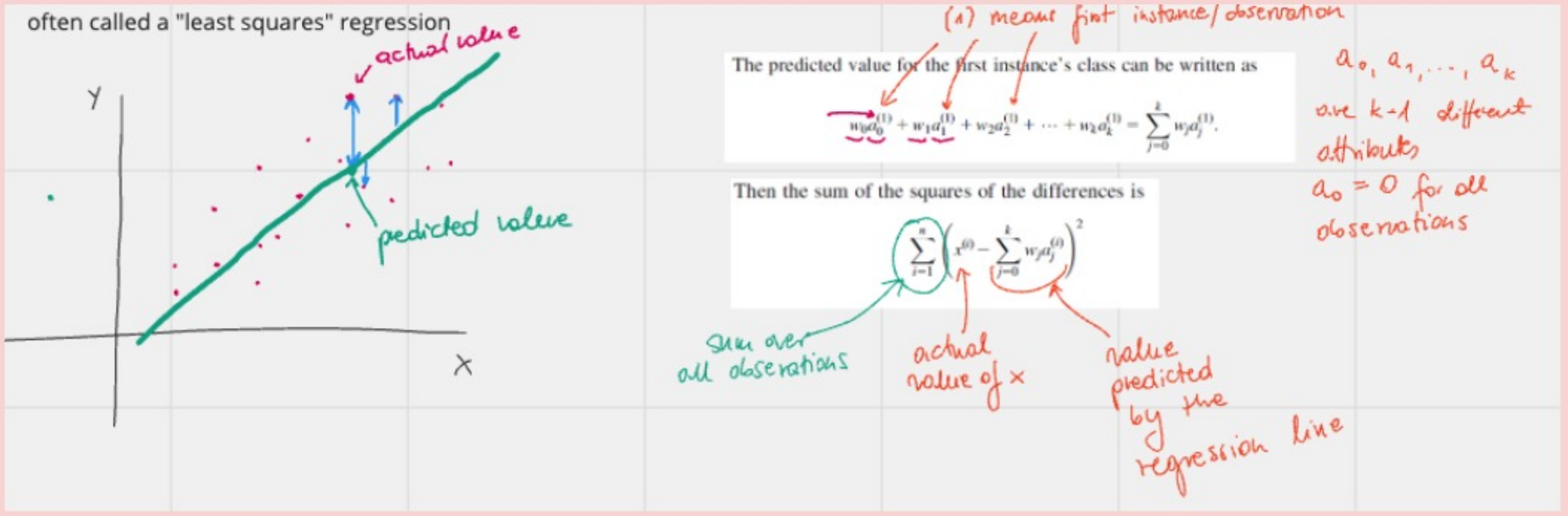
- Linear regression focuses on the regression task; i.e. predicting a real number.
- We can think of it as corresponding to finding a hypothesis in the hypothesis space.
- ***y = w0 + w1X*** where ***w0*** and ***w1*** are weights which we need to find values for.
- We can have a multivariate regression, whereby X contains lots of different attributes.
    - ***y = w0 + w1X1 + w2X2 + w3X3*** ... 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 (w0, w1, ...)?

- 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.



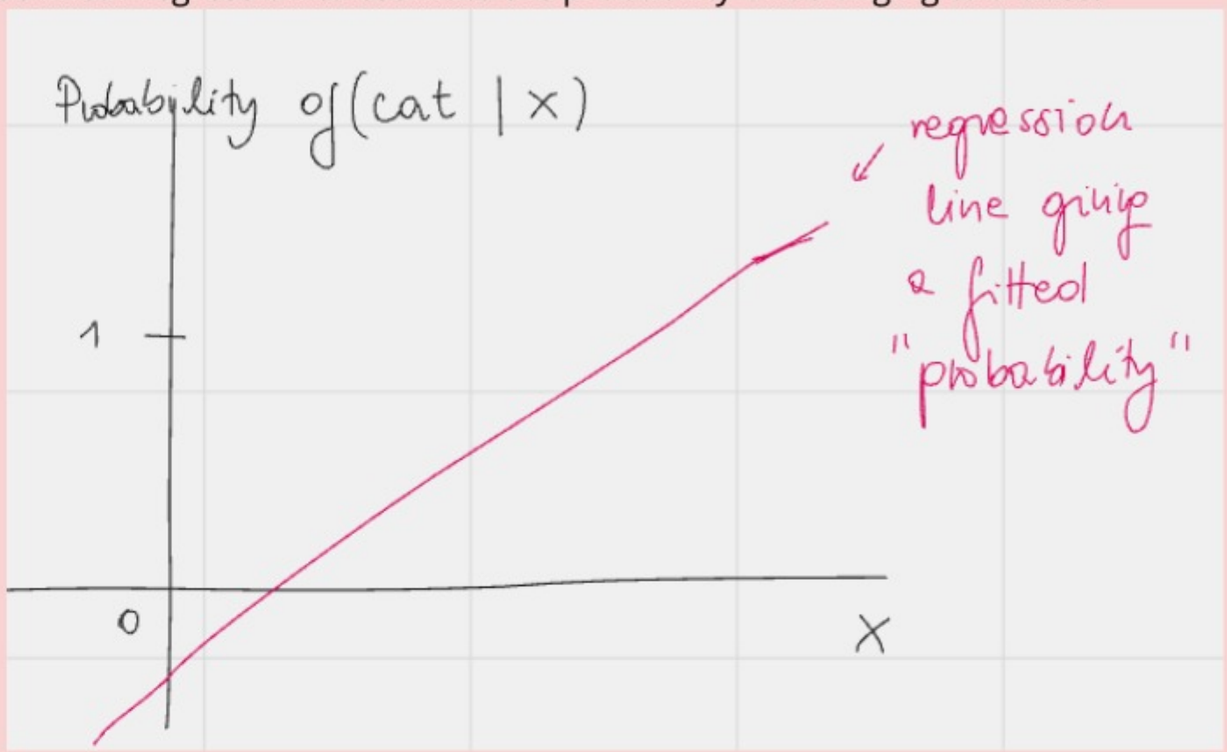
Why the sum of squares?

- 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

- 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 negarive 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.

Neural Network Structures

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

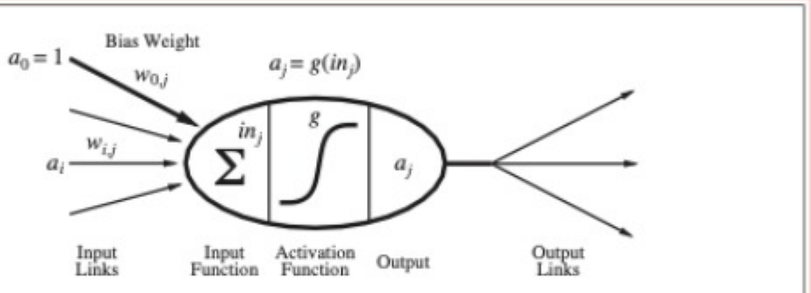


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

- A **link** from ***i*** to ***j*** serves to propagate the activation ***ai*** from ***i*** to ***j***.
- Each link also has a numeric **weight *wi,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** - σ(x)=1/(1+e^−x)
  - **ReLU** (Rectified Linear Unit) - ReLU(x) = max(0, x)
  - **softplus** - softplus(x) = log(1+e^x)
    - A smooth version of ReLU; the derivative of the softplus function is the sigmoid function.
  - **tanh** (hyperbolic tangent) - $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$.
    - 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.

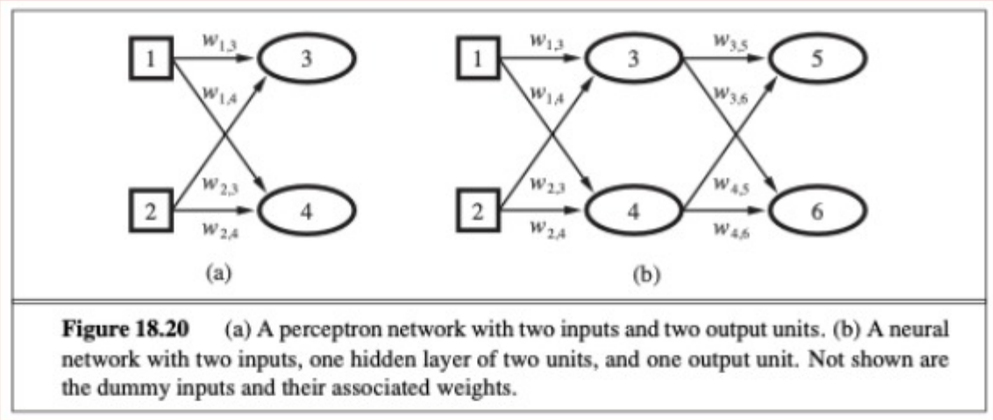


Figure 18.20 (a) A perceptron network with two inputs and two output units. (b) A neural network with two inputs, one hidden layer of two units, and one output unit. Not shown are the dummy inputs and their associated weights.

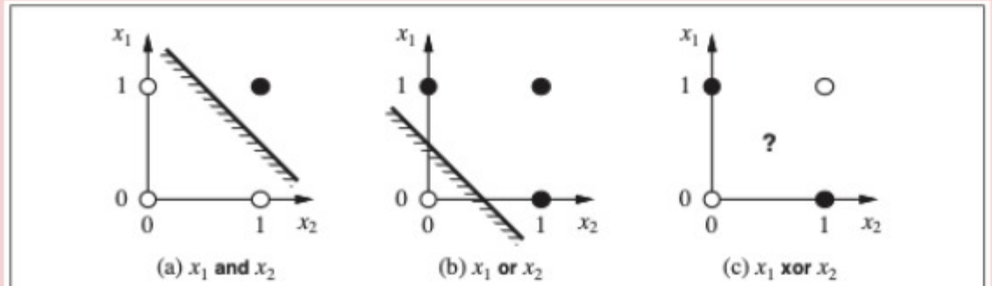


Figure 18.21 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c), no such line exists that correctly classifies the inputs.

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 hte 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 complext 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.

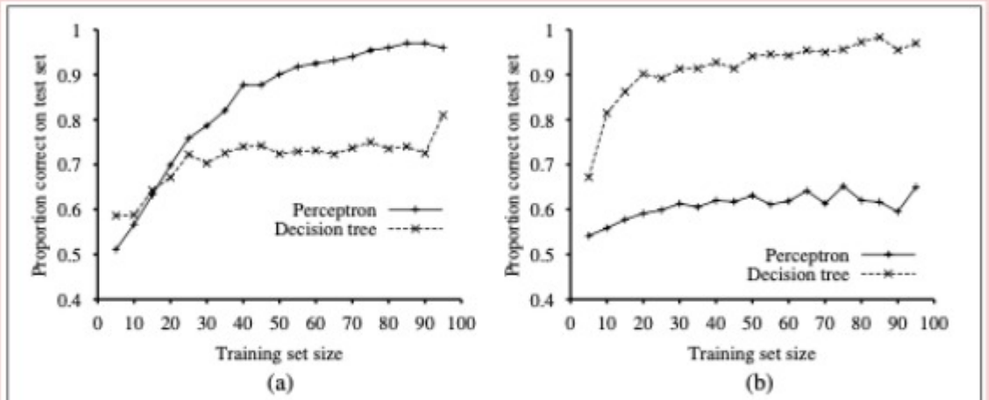


Figure 18.22 Comparing the performance of perceptrons and decision trees. (a) Perceptrons are better at learning the majority function of 11 inputs. (b) Decision trees are better at learning the *WillWait* predicate in the restaurant example.

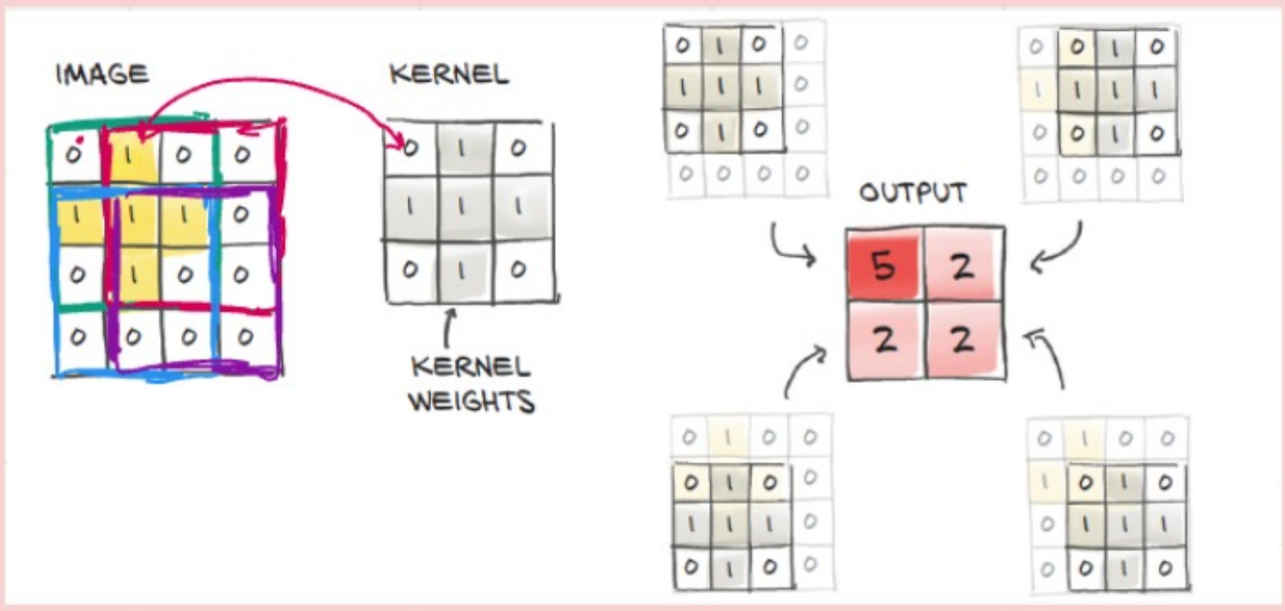
Convolutional Neural Networks (CNNs)

- CNNS are most commonly used in processing image data.
- With a picture that has PxP pixels and 3 RGB color channels, we could (in theory) build a network based on PxPx3 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 patters (e.g. vertical lines, horizontal lines, etc.).

The benefits of using filters/kernels:

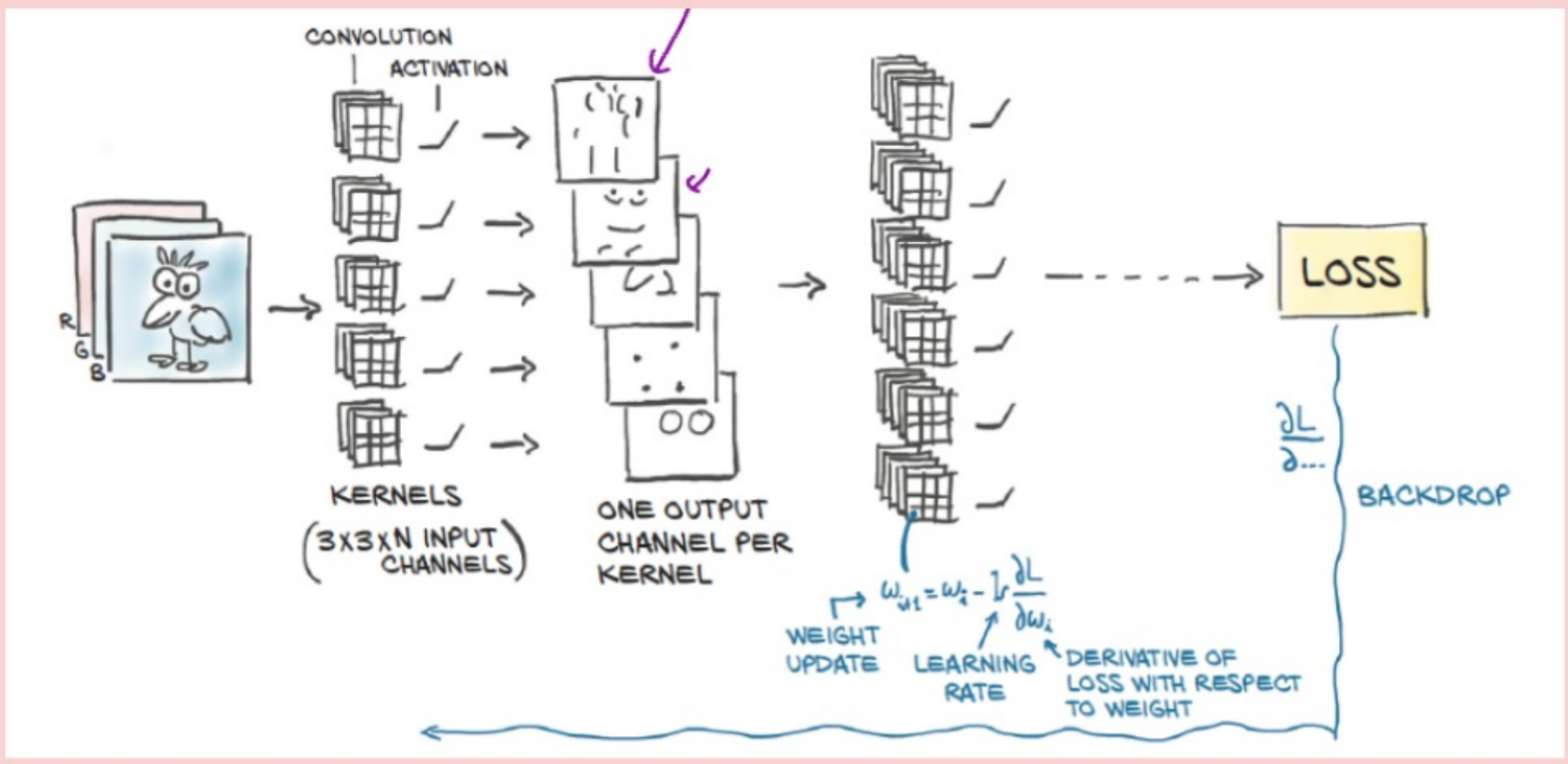
1. **translation invariance** - they help us recognize a pattern regardless of where it is in the picture.
2. **locality** - processing local pixels together, to spot local patterns.
3. **reduction in the number of weights** that we need to train since copies of the same filter use the same weights.



There are 2x2 different positions that we can arrange a 3x3 filter in a 4x4 space, so the output is 2x2

Regardless of its position, the same kernel moved around the picture has 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 2x2 for a single kernel, with ***n*** kernels the output world would be 2 x 2 x ***n***.



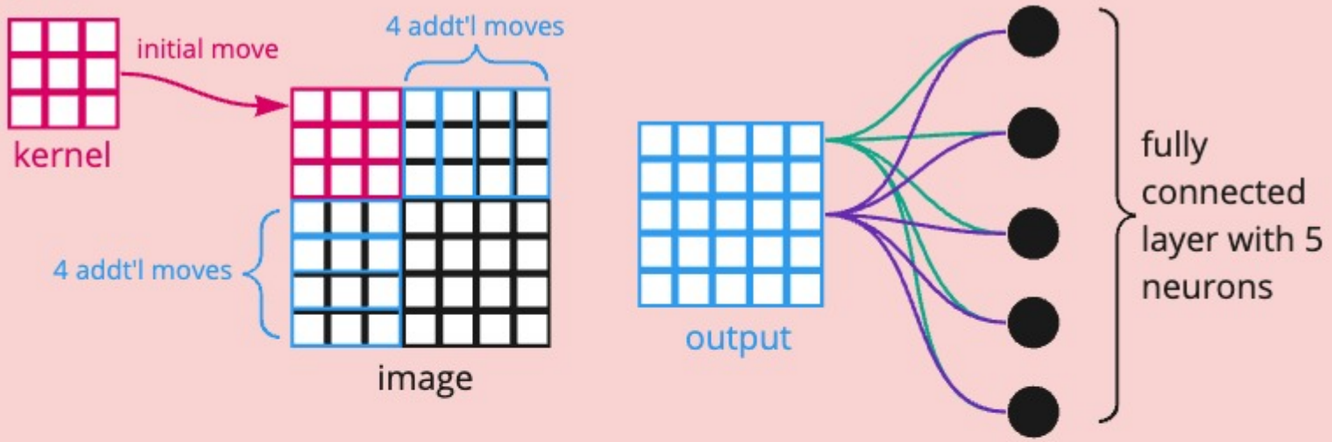
CNN Example - Calculating the number of weights

Problem: Suppose we have a 7x7x1 image to start with, and a convolution layer with one 3x3x1 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 3x3x1 kernel will have 3x3x1 = 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 5x5x1 different ways to place a 3x3x1 kernel in a 7x7x1 image, so the output of the convolutional layer is 5x5x1.**



Each of the neurons in the fully connected layer is a function of the 5x5x1 cells of the output of the convolutional layer + it also contains 1 bias term per each kernel.

So each neuron has 5x5x1 = 25 inputs, +1 bias = 26 wts/neuron.

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

Total weights = 10 weights + 130 weights = 140 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.

- ○ A non-linear function and an activation function.
- ○ A rectifier function and a hard threshold function.
- ◉ A linear function and an activation function.
- ○ A linear function and a logistic function.
- ○ A non-linear function and a hard threshold 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 2 1 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.

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.

- ○ 5, 5, 1.
- ○ 0, 1, 1.
- ◉ 1, 1, 5.
- ○ 1, 0, 1.
- ○ 0, 1, 5.

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.

- ○ 0, 0, -4.
- ○ 0, 1, -4.
- ○ 0, 1, 0.
- ◉ 0, 0, 0.
- ○ 1, 0, 1.

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.

- ○ 35
- ○ 625
- ◉ 275
- ○ 250

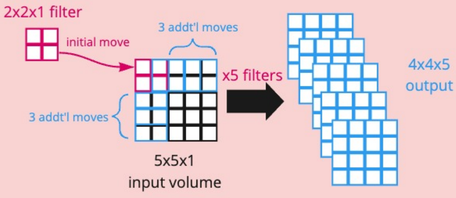
1st hidden layer: (10 + 1) x 10 = 110 weights
2nd hidden layer: (10 + 1) x 10 = 110 weights
output layer: (10 + 1) x 5 = 55 weights
total: **275 weights**

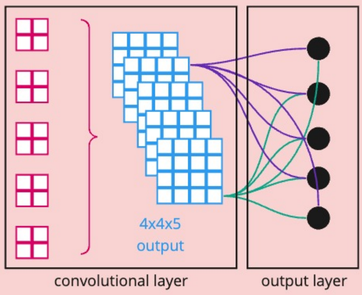
Question 6 1 pts

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.

- ◉ 430
- ○ 2500
- ○ 2485
- ○ 805
- ○ 530





(2x2x1 + 1) x 5 filters = 25 weights
(4x4x5 + 1) x 5 neurons = 405 weights
430 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: 50

Second convolutional layer: 105

Output layer: 7210

A model answer for this question will be released towards the end of the week.

1st convolutional layer:
(3x3x1 + 1) x 5 = **50 weights**

output of 1st convolutional layer:
3x3x1 into 15x15x15 = 13x13x1
(13x13x1) x 5 filters = 13 x 13 x 5 volume

2nd convolutional layer:
(2x2x5 + 1) x 5 = **105 weights**

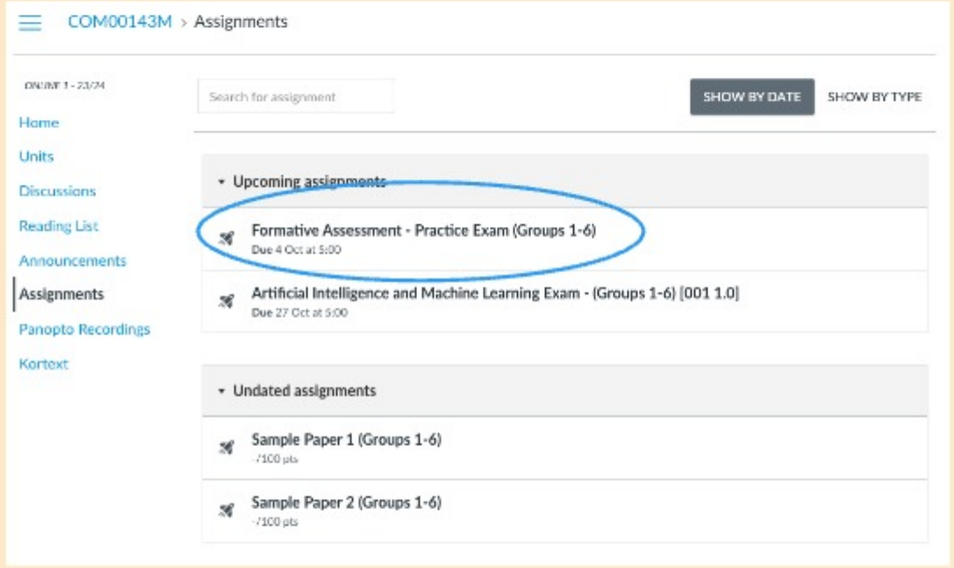
output of 2nd convolutional layer:
2x2x5 into 13x13x5 = 12x12
(12 x12x1) x 5 filters = 12 x 12 x 5

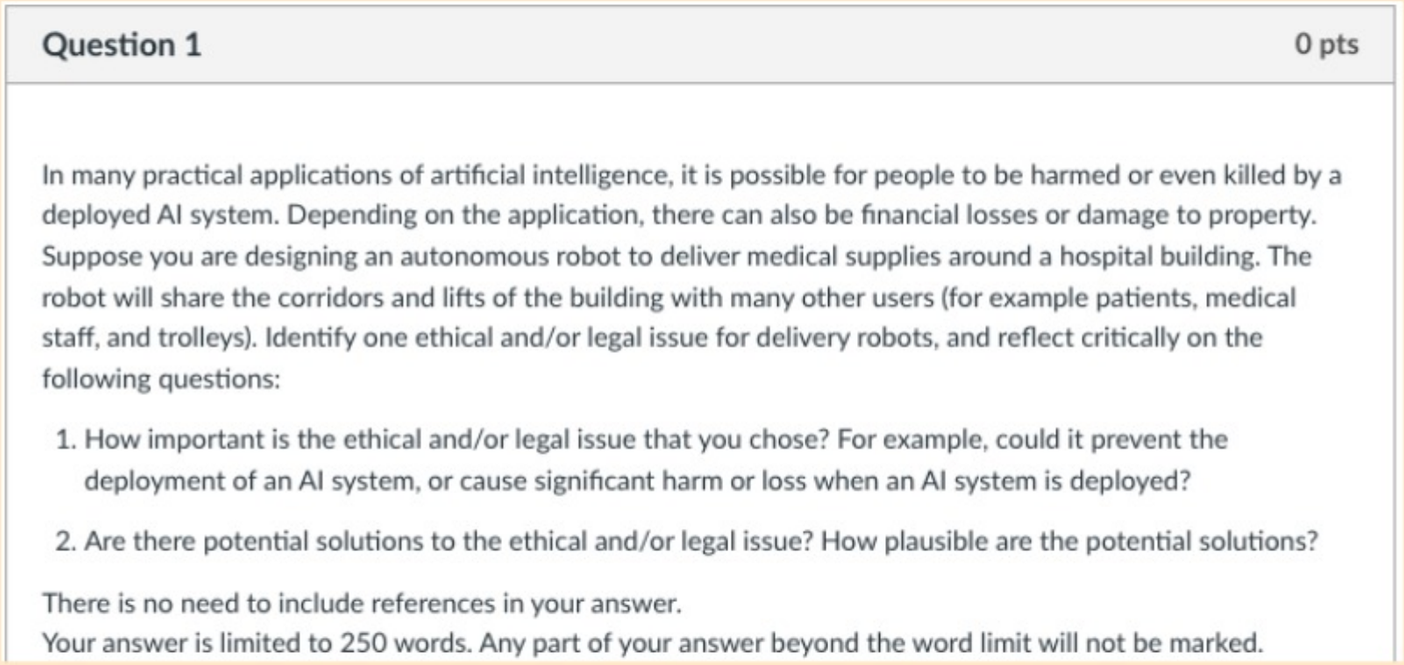
output layer:
(12 x 12 x 5 + 1) x 10 neurons = **7210 weights**

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!





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

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

✓ Correct per the answer key

Question 4 — 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

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.

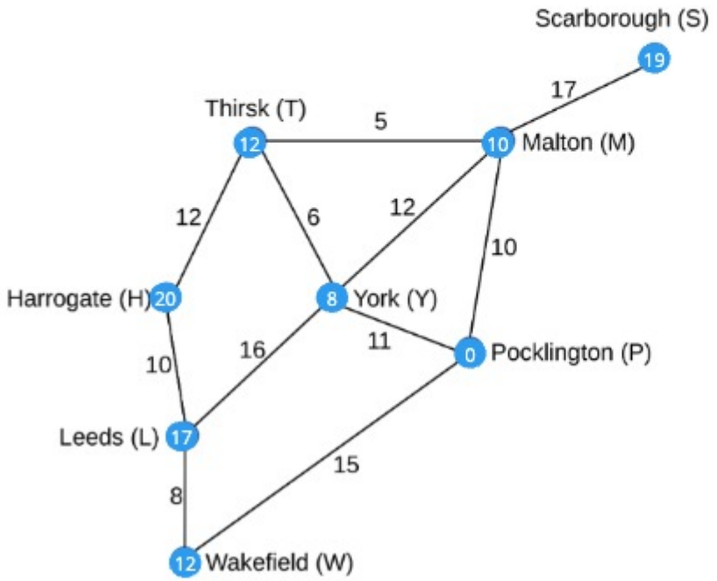
Write your answer using big-O notation, and if you need to use powers then write them in TeX notation. For example, n squared would be written as n^2. If the algorithm has worst-case complexity n squared, it would be written as O(n^2).

Time complexity: O(b^m) ✓ Correct per the answer key

Space complexity: O(bm) ✓ Correct per the answer key

Question 6 0 pts

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



City 1	City 2	Distance by road
Malton	Scarborough	17
Thirsk	Malton	5
York	Malton	12
Pocklington	Malton	10
Thirsk	York	6
York	Pocklington	11
Thirsk	Harrogate	12
Harrogate	Leeds	10
York	Leeds	16
Pocklington	Wakefield	15
Leeds	Wakefield	8

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

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**.

Greedy Best First Search (Graph Based)

- **priority queue using h(n) = euclidean distance to Pocklington**

Start: Explored Nodes = []
Frontier = [H(20)]
H is selected for expansion

H → Explored Nodes = [H]
Frontier = [L(17), T(12)]
T is selected for expansion

T → Explored Nodes = [H, T]
Frontier = [L(17), M(10), Y(8)]
D is selected for expansion

Y → Explored Nodes = [H, T, Y]
Frontier = [L(17), M(10), P(0)]
P is selected for expansion

End: Explored Nodes = [H, T, Y, P]
P is the Goal

✓ Correct per the answer key

Step 1

Node selected: H

New frontier in alphabetical order: L, T

Step 2

Node selected: T

New frontier in alphabetical order: L, M, Y

Step 3

Node selected: Y

New frontier in alphabetical order: L, M, P

Step 4

Node selected: P - Goal

New frontier in alphabetical order: L, M

Question 7 — 0 pts

What is the meaning of this symbol: $\models$

- ○ Inference
- ○ Sentence
- ○ Knowledge base
- ✗ Entailment ✓ Correct per the answer key
- ○ Negation

Question 8 — 0 pts

What is the meaning of this symbol: $\neg$

- ○ Inference
- ○ Knowledge base
- ○ Entailment
- ✗ Negation ✓ Correct per the answer key
- ○ Sentence

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

B ∧ ¬D
FALSE
TRUE
FALSE
TRUE
FALSE
FALSE
FALSE
FALSE
FALSE
TRUE
FALSE
TRUE
FALSE
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 ∨ ¬C = 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 ∧ ¬A = () which is false ✓ Correct per the answer key

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.

- ○ 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.

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, -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 -Bv-S.

¬B ∨ S ∨ W ✓ Correct per the answer key

$B \wedge \neg S \Rightarrow W$

$\neg(B \wedge \neg S) \vee W$ *implication elimination*

$\neg B \vee \neg(\neg S) \vee W$ *De Morgan's Law*

$\neg B \vee S \vee W$ *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=false, B=false, C=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: 3 ✓ Correct per the answer key

B: 2 ✓ Correct per the answer key

C: 2 ✓ Correct per the answer key

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

START STATE
A - F
B - F
C - T

A FLIPPED
A - T
B - F
C - T

1 SATISFIED
2 SATISFIED
3 SATISFIED
TOTAL 3

B FLIPPED
A - F
B - T
C - T

1 SATISFIED
2 NOT SATISFIED
3 SATSIFIED
TOTAL 2

C FLIPPED
A - F
B - F
C - F

1 NOT SATISFIED
2 SATSIFIED
3 SATISFIED
TOTAL 2

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

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

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

A	B	C	A v B	A v ¬B	A v B v ¬C
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE

- Pick a clause at random from unsatisfied clauses:
  - Given: Algorithm chooses 3: A v B v ¬C
- Random or Greedy:
  - Probability of random walk = 0 => event will not happen
  - Greedy Choice:
    - If we flip A = True, then 1, 2, & 3 are satisfied

A	B	C	A v B	A v ¬B	A v B v ¬C
TRUE	FALSE	TRUE	TRUE	TRUE	TRUE

    - If we flip B = True, then 1 & 3 is satisfied

A	B	C	A v B	A v ¬B	A v B v ¬C
FALSE	TRUE	TRUE	TRUE	FALSE	TRUE

    - If we flip C = False, then 2 & 3 is satisfied

A	B	C	A v B	A v ¬B	A v B v ¬C
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

- Flip selected symbol
  - A = True

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

Question 16 — 2 pts

The mathematical model of a neuron has two parts. Name the two parts.

- ○ An arbitrary non-linear function and an activation function.
- ○ A rectifier function and a hard threshold function.
- ○ An arbitrary non-linear function and a logistic function.
- ✗ A linear function and an activation function. ✓ Correct per the answer key
- ○ A linear function and a logistic function.
- ○ An arbitrary non-linear function and a hard threshold function.

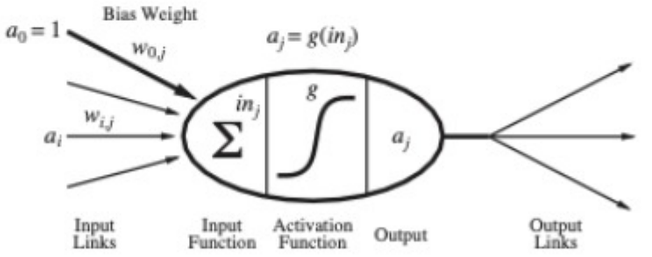


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

Question 17 — 2 pts

Suppose you have a data set containing examples with multiple attributes and no labels. Which type of machine learning would be appropriate?

- ○ Supervised learning.
- ✗ Unsupervised learning. ✓ Correct per the answer key
- ○ Reinforcement learning.
- ○ Deep learning.

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

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. ✓ Correct per the answer key
- ○ The rectifier function.

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

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

Enter the updated **x** and **y** positions of cluster centres A and B in the boxes below.

Cluster A

x:

y:

Cluster B

x:

y:

Question 21 — 9 pts

Suppose you have a convolutional neural network with an 8x8x3 input volume representing a colour image. The first layer is a convolutional layer consisting of 10 filters that have a 3x3x3 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 x 6 x 10 ✓ Correct per the answer key

3x3x3 filter has 27 weights + 1 bias = 28 weights/filter
x 10 filters = 280 weights

3x3x3 filter on 8x8x3 input = 6x6x1 output layer x 10 .
= 6x6x10 output layer volume???

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 8x8x8. Suppose the second layer is a fully connected layer with 5 neurons. How many weights does the second layer have?

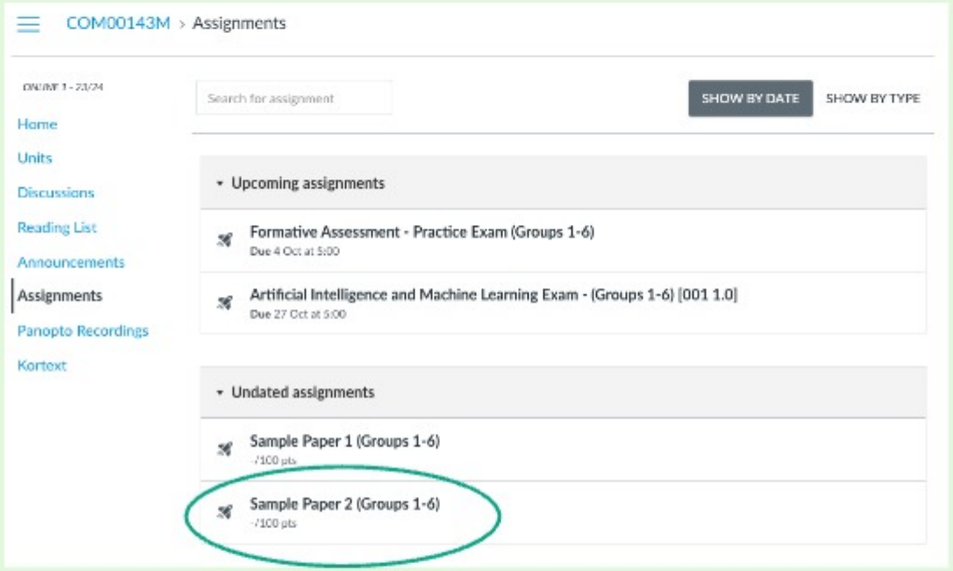
2565 ✓ Correct per the answer key

each neuron has 8x8x8 = 512 inputs + 1 bias = 513 weights

513 weights x 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.



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!

Question 1 10 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 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.

Identify **one** ethical and/or legal issue in this scenario, 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.

Question 2 — 3 pts

Consider the following description of a general search algorithm.

1. **function** XXXXXX-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

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	

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)

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:

New frontier set in alphabetical order:

Step 2

Node selected:

New frontier set in alphabetical order:

Step 3

Node selected:

New frontier set in alphabetical order:

Question 7

2 pts

What is the meaning of this symbol: $\vdash$

- [x] Inference
- [ ] Negation
- [ ] Entailment
- [ ] Knowledge base
- [ ] Sentence

Question 8

2 pts

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

$2^4 = 16$

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=false, B=false, C=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
- [x] A=false, B=false, C=true
- [x] A=false, B=true, C=false
- [ ] A=false, B=true, C=true
- [x] 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 = False, B = False, C = False

- Evaluate clauses:
  - Satisfied Clauses: 2, 3, & 4
  - Unsatisfied Clauses: 1
- Pick a clause at random from unsatisfied clauses:
  - 1: A v B
- Random or Greedy:
  - Probability of random walk = 0.5
  - Greedy Choice:
    - If we flip A = True, then 1, 2, & 4 are satisfied
    - If we flip B = True, then 1, 2, & 4 are satisfied
    - If we flip C = True, then 2, 3, & 4 are satisfied
- Could flip any of the symbols giving us:
  - A = True, B = False, C = False
  - A = False, B = True, C = False
  - A = False, B = False, C = True

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, -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 -Bv-S.

B ∨ ¬S ∨ ¬W

¬B ∧ S ⇒ ¬W
¬(¬B ∧ S) ∨ ¬W *implication elimination*
¬(¬B) ∨ ¬S) ∨ ¬W *De Morgan's Law*
B ∨ ¬S ∨ ¬W *double negation elimination*

Implication Elimination
turning *implication* into *or*

$\frac{p \Rightarrow q}{\neg p \vee q}$

if p implies q
∴
either p or q is true

De Morgan's Laws
turning *and* into *or*

$\frac{\neg(p \wedge q)}{\neg p \vee \neg q}$

$\frac{\neg(p \vee q)}{\neg p \wedge \neg q}$

Double Negation Elimination
the removal of *not*

$\frac{\neg(\neg p)}{p}$

if p is not false
∴
p is true

Question 11 — 8 pts

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 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:

Step 2:

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.

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.
- ○ 1, 6, 1.
- ○ 0, 1, 1.
- ○ 1, 0, 1.
- ○ 1, 1, 6.
- ○ 0, 1, 6.
- ○ 1, 1, 1.

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.

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

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'.

A. f(x,y) = threshold(x + y – 4.5)

B. f(x,y) = threshold(x + y – 3.5)

C. f(x,y) = threshold(-x -y + 3.5)

D. f(x,y) = threshold(x + y – 2.5)

Question 16

9 pts

Suppose you have a convolutional neural network with an 9x9x3 input volume representing a colour image. The first layer is a convolutional layer consisting of 10 filters that have a 4x4x3 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: