COMP30024 AI Notes (2018 Sem 1)

Overview

Textbook:
Artificial Intelligence: A Modern Approach, by S. Russell and P. Norvig, Pearson, ISBN 9781292024202. Available in the ERC Library on reserve, 006.3 RUSS, or in the Co-op Bookshop.
Textbook website: http://aima.cs.berkeley.edu/

Project: *Python*
(Team of 2)
Part A: 30th March
Part B: 11th May

Lectures

Lecture 1 - Introduction

AI Bounds: learn from failures

What is AI?

AI is a science of making computers look more like the ones we see in movies

What sort of behaviour defines intelligent behaviour? (eg. C3PO)

- ❖ Walking
- ❖ Conversation / natural language
- ❖ Identifying other robots
- ❖ Decision making
- ❖ Path finding
- ❖ Social awareness
- ❖ Self awareness
- ❖ Opinion / thinking
- ❖ Inferencing conclusions
- ❖ Emotions (empathy)
- ❖ Long term planning

^ Still better than our current technologies

Approaches to defining AI:

- Thinking like a human
  - **Cognitive modelling**: figure out how we think by *introspection* or *experimentation*.
  - **Self-awareness** is important
  - But, humans feel emotions and don't always think or act rationally.
- Thinking rationally
  - Come up with **laws of thought**:
    - Be able to inference conclusions
  - Rational thinking / study of *logic* by Aristotle
    - "Socrates is man, all men are mortal, therefore Socrates is mortal."
  - But, sometimes inference may not be easy or may not be true
- Acting like a human
  - Turing test + Eliza
- Acting rationally *(focus of this subject)*
  - The rational agent: goal-driven system
  - Because may not have complete information, should have to go beyond strict rational thought in general

Tests for Intelligence: - Turing Test

Suggested major components of AI:

- knowledge
- reasoning
- language understanding
- learning

Problems:

- not reproducible - cannot get exact same results again
- not constructive - does not tell how to make it better
- not amenable to mathematical analysis - not quantifiable

- https://www.scientificamerican.com/article/the-search-for-a-new-test-of-artificial-intelligence/

Eliza Session:
- problem with semantic understanding

State of the art software in this area:

What can be done:

- discover and prove a new math theorem

Lecture 2 (Week 1) - What is AI?

The agent model:

characterise requirements for an agent in terms of its percepts, actions, environment and performance measure.

- ➔ An agent **takes inputs** (perceptions of the environment made by ***sensors***)
  - Video feed, accelerometers, keyboards, GPS
- ➔ Actions which **change the environment** done by ***actuators***
  - Change direction, accelerate, brake, speak, display
  - need to determine the best action, if that action brings us to a state that is closer to our goal.
- ➔ Environment where the **agent exists**
  - city streets, freeways, pedestrians, space,
- ➔ Performance measure of the **desirability of the current state** of the environment
  - Safety, pathfinding, maximizing profits
  - how rational is the behaviour

- ➔ An agent can be evaluated as a function; it does a thing when given a perception
- ➔ Ideally always maximizing its *performance measure*

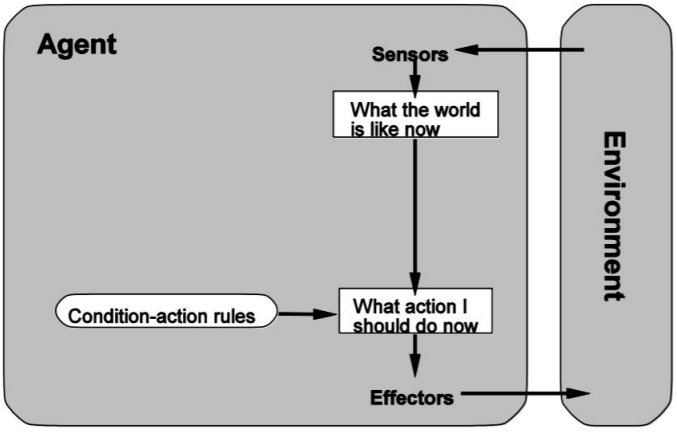
- ➔ An agent should be rational, which means it is ***not***:
  - Omniscient (perceieves everything)
  - Clairvoyant (can tell the future)
  - Successful (wins all the time 😦)

Types of Agents:

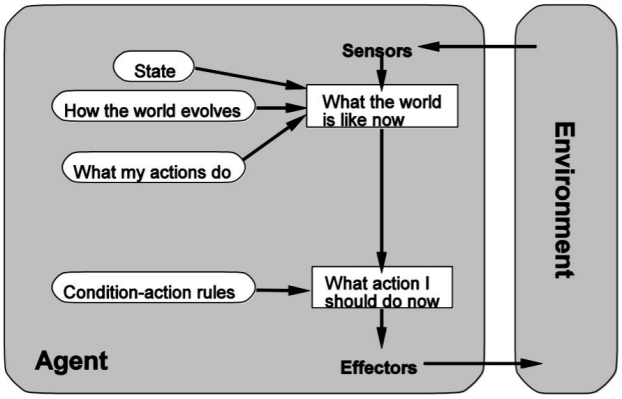
choose and justify choice of agent type for a given problem.

Simple Reflex Agents

- Lives in the moment
- Does actions according to what is happening right now
- Doesn't remember the past
- Probably doesn't like reading books
- Bad at giving directions

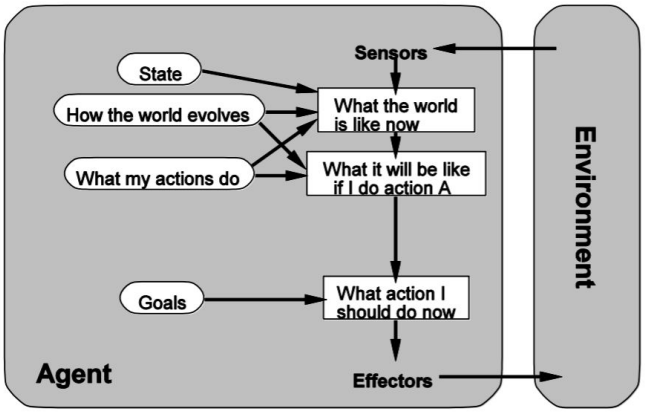


Model-based Reflex Agents



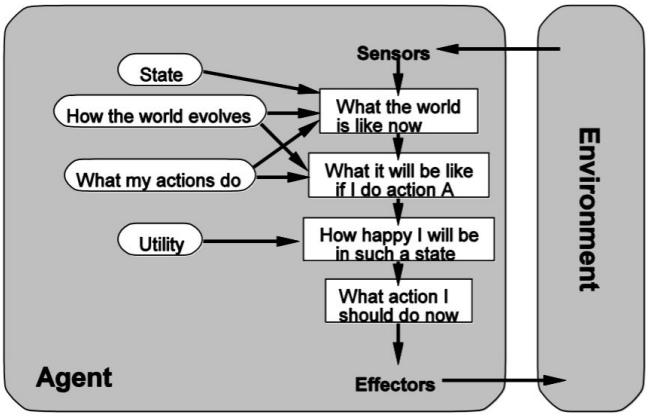
- Takes into account how the environment works
    - world is now fully observable
    - don't have complete information
        - like driving behind a truck, can't see everything but can derive based on truck's movements.
- Uses a lot of memory
- Plans ahead a little
    - E.g. which path should I take to go to the airport

Goal Based Agents



- Takes into account what will happen if agent does a certain thing
    - has model of **current** and **future** states of the world
    - see how each possible action affects distance from goal
- have a **goal** to achieve instead of condition-action rules
- “If I take a left here what will be my remaining path”
- Plans ahead a lot
- Probably likes organizing holidays

Utility Based Agents



- Does the same as a goal based agent, but ***maximizes happiness***
- how we should prioritise our goal?
- “If I take a left here how much time will be remaining on my path and is there a faster way?”
- Probably insecure about it’s happiness

Which agent you use depends on how complex your problem is:

➔ Don't want a utility based agent to select the color red or blue
➔ Don't want a simple reflex agent to decide which path to take to the airport
➔ Based on **environment**
➔ Don't wanna use something too complicated if can use something simpler

Environment Types:

characterise the environment for a given problem.

- Observable (vs Partially Observable)
  - Your perception contains *all relevant information* about the environment
- Deterministic (vs Stochastic)
  - Current state of the world *uniquely determines the next*
  - stochastic: there is some *random* element in env
- Episodic (vs Sequential)
  - Only the current perception is relevant
  - sequential: like basketball, have to look several steps ahead
- Static (vs Dynamic)
  - Environment *doesn't change* while the *agent is thinking/deciding*
- Discrete (vs Continuous)
  - *Finite number* of possible percepts/actions

Examples:

	Solitaire	Backgammon	Internet shopping	Taxi
Observable??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Yes	Semi	No
Discrete??	Yes	Yes	Yes	No

Apparently Backgammon may not be episodic? Never played it so I have no clue.

Lecture 3 - Problem Solving and Search

(Chapter 3, Sections 1 - 4)

Problem Solving Agents

Restricted/(Generic) Form of General Agent

```
# This is offline problem solving:  -- for this subject only offline, no online
#       Acting only with complete knowledge of problem and solution
function SimpleProblemSolvingAgent(percept) returns an action
      static:    seq, an action sequence, initialised to empty
                 state, current world state
                 goal, initialised to null
                 problem, a problem formulation
      state ← UpdateState(state, percept)
      if seq is empty then
            goal ← FormulateGoal(state)
            problem ← FormulateProblem(state, goal)
            seq ← Search(problem)
      action ← Recommendation(seq, state)
      seq ← Remainder(seq, state)
      return action
```

Has a main function which **returns an output**:

Takes a percept

Has: **Action** sequence (initial empty)

State

Goal

Problem

Every other moment you should **Update State**:

If **current state** is empty:

Goal -> find goal from current state

Problem -> find problem from current goal and state

Search -> find solution to problem

Action -> solution from state and search

Return this action

E.g. Structure of how a General Agent would figure out how to get to my house

Formulate goal:

Get to Sam's House

Formulate problem:

State(s): I'm in southbank

Operator/Action(s): Drive down a road

Find Solution:
Sequence of roads: Swanston street! -- can represent as a **graph**.

How to define a Problem:

Single-State Problem Formulation

translate problem into a search problem
Modelling solutions in the real world 1:1 is really difficult
- We must *abstract* the world into a model for problem solving

Defined by *four items*:
Initial State:
Where am I/How am I doing in this environment?
I'm at Southbank
Actions/Operators: *(or successor function S(x))*
What can I do?
I can drive down this road
eg. Elizabeth Street → Victoria Street
Goal Test:
Explicit (there is only 1 state that satisfies the goal):
There is only Sam's place and that's at 108-128 Leicester St
Implicit (there are several states that can satisfy the goal):
You can have *checkmate* in chess in *many different ways*
(still a single state problem)

Path Cost: -- measure to determine how good the solution is
(usually additive and non-negative)
How much does this path via this action take?
It's 1km away from Sam's place this way, or
You need 5 moves in chess to win this

A **solution** is considered **a sequence of actions** leading from the **initial state** to the **goal state**.💪💪

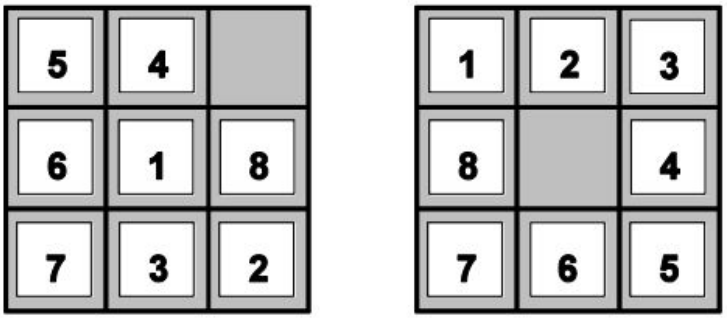
Selecting a State Space:

Because real world is too complex, need to come up with an **abstract** version of the problem.
Abstract state : set of real states
Abstract action : complex combination of real actions
eg. "Melbourne → Ballarat" represents a complex combination of real actions like possible routes, detours, rest stops, etc...
Abstract solution : set of real paths that are solutions in the real world

- each abstract action should be easier than the original problem :)

Example of Modelling Problems:

Simple 8 puzzle game



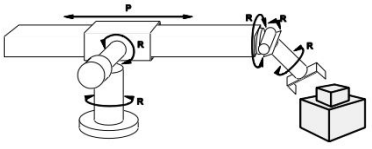
Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

This can be considered an explicit goal, as there is *only one goal state*. (i.e. you can only win one way).

Robotic Assembly



This can be considered an implicit goal as there are *many ways to satisfy the goal*. (i.e. the robot can twist in many ways to satisfy the goal).

states??: real-valued coordinates of
robot joint angles
parts of the object to be assembled

actions??: continuous motions of robot joints

goal test??: complete assembly *with no robot included!*

path cost??: time to execute

General Search Algorithms:

Basic Idea:

- Offline, simulated exploration of state space by generating *new successors of already explored states*.
- Basically, keep trying to find a new path till you reach the goal. If you fail, try again. (wow how optimistic! 👌)

- generate a tree that starts at initial state and hopefully reaches the goal state -- use a DFS-kind of algorithm

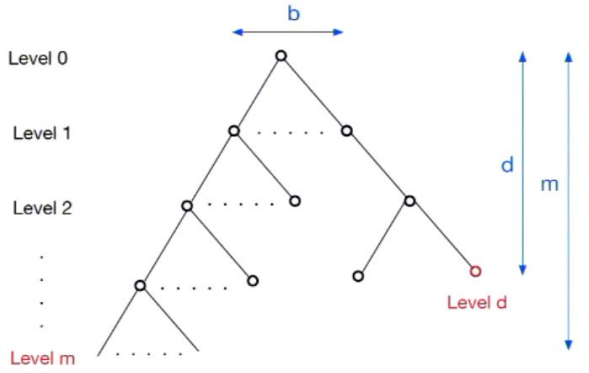
State vs Node:

- A **state is a representation** of a physical configuration
- A **node is a data structure** constituting part of a search tree
- States do not have parents, children, depth or path cost!

Lecture 4 (Week 2) - Searching Algorithm Implementation

Completeness: Does it **always find a solution** if one exists?
Time Complexity: **Number of nodes generated**/expanded
Space Complexity: **Maximum number of nodes** in Memory
Optimality: Does it always find a **least cost solution**?

b — Maximum **branching factor of the search tree**
d — **depth** of the **least-cost solution**
m — **Maximum depth** of the state space (can be infinite)



m: maximum depth
b: maximum branching factor
d: depth of least cost solution

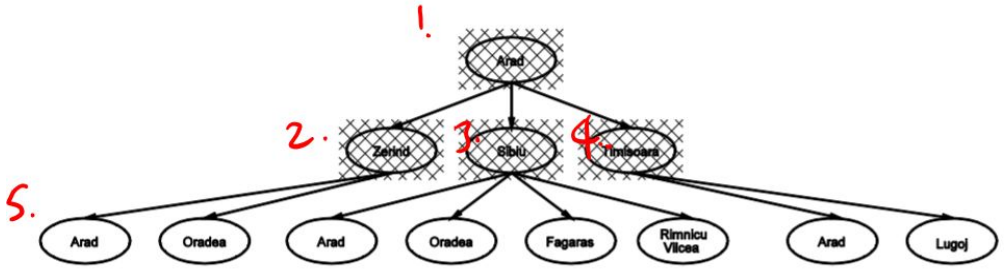
For some stupid reason, these slides aren't in the .pdf... why???

Search Strategies:

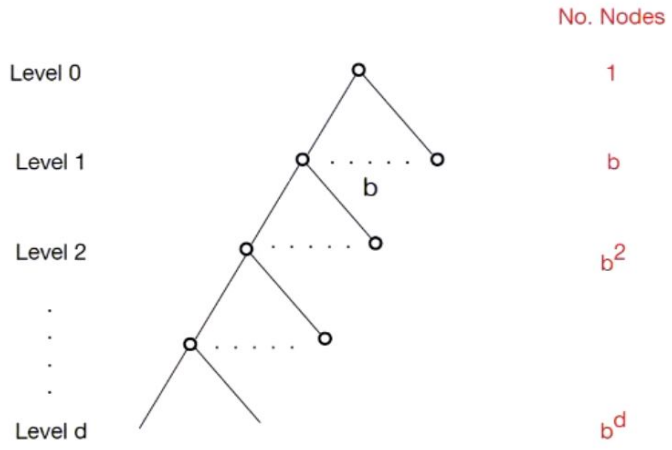
Strategy is defined by picking the **order of node expansion**.

Breadth First Search

Works by Expanding the shallowest node.



- **Complete**: Yes
- **Time**: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$ <- Exponential in d



$$\text{Total} = 1 + b^1 + b^2 + \ldots + b^d$$

$$= \frac{b^{d+1} - 1}{b - 1}$$

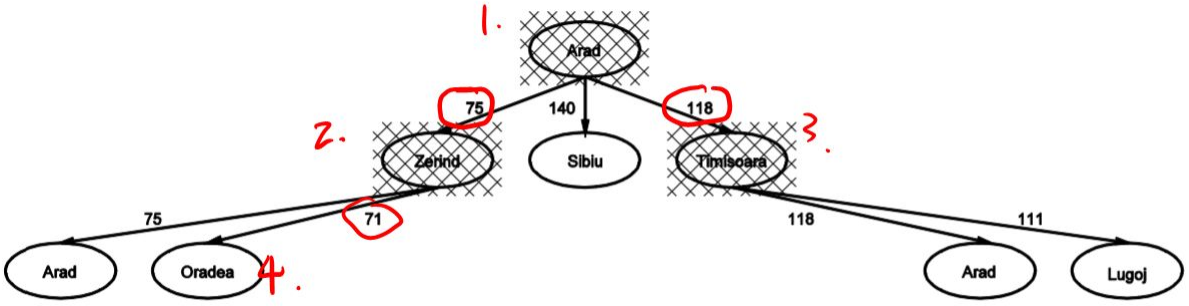
$$= O(b^d)$$

-

- **Space**: $O(b^d)$ <- Keeps every node in memory
  - Can generate nodes at 1MB/sec so 24 hrs of running = 86GB
  - Not very space efficient!
- **Optimal**: Yes (If cost = 1 per step); not optimal in general

Uniform Cost Search

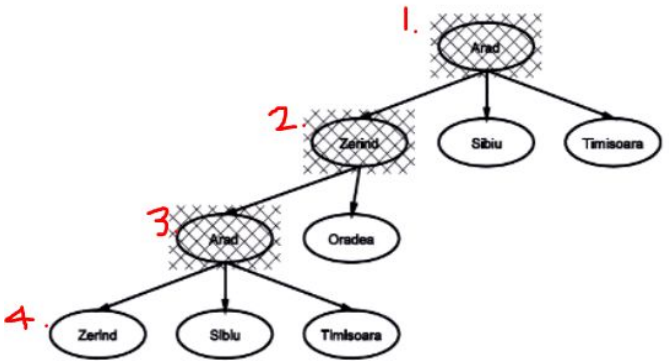
- Expands the least-costing node
- Note: can result in infinite loop

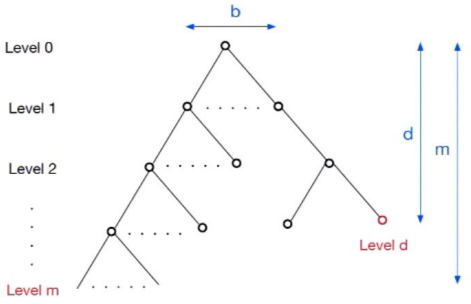


Complete: Yes, if step cost >=

Depth first Search

- Goes deep
- explores a arbitrary child of each node until it hits a leaf, then backtracks to the other children of the last visited node.





b: maximum branching factor
d: depth of least cost solution
m: maximum depth of state step

$b = 10$ $1000\ B/node$

$d = 10$ $10^6\ nodes/sec$

$m = 20$

	BFS		DFS
T	b^d	<<	b^m
	$\frac{10^{10}}{10^6} = 3\ hrs$		$\frac{10^{20}}{10^6} = 10^{14}$
M	b^d	>>	$b \times m$
	$10\ TB$		

Depth limited Search

- Depth first search but limits itself to a certain depth
- E.g. if this is depth of 1 then it's basically breadth first

Iterative Deepening Search

- This starts at the root always and then goes deeper and deeper with each iteration
- So for the first one it may go to depth 1, but on the second it'll go to depth 2 and so on
- Analogous to Breadth First Search (if your depth increases by 1 every iteration)
- Uses only **linear space** and not much more time than other uninformed algorithms

Bidirectional Search

Search simultaneously forwards from the start point and backwards from the goal. Stop when the two searches meet in the middle

Problems:
- generate predecessors
- many goal states
- need efficient check for nodes already visited by other half of search
- what kind of search?

Properties:
- Complete
- Time $O(b^{\frac{d}{2}})$
- Space $O(b^{\frac{d}{2}})$
- Optimal (if done with correct strategy ie. BFS)

^ All the above are good but a bit inefficient

Lecture 5 (Week 3) - Informed Search Algorithms

Chapter 3, Sections 5-6

Heuristics:
In what sort of problems do we need to use heuristics?
- NP-Hard Problems
  - Solutions where you need an approximation of the problem
- "Very complicated problems" (Branching problems)
  - Gives a "good" solution in a "timely manner"
  - May not give an optimal solution

Best First Search

Use an **evaluation function** for each node
- ➔ determine "desirability" (using domain knowledge)
- ➔ "How close does our next node get us to the goal node?"

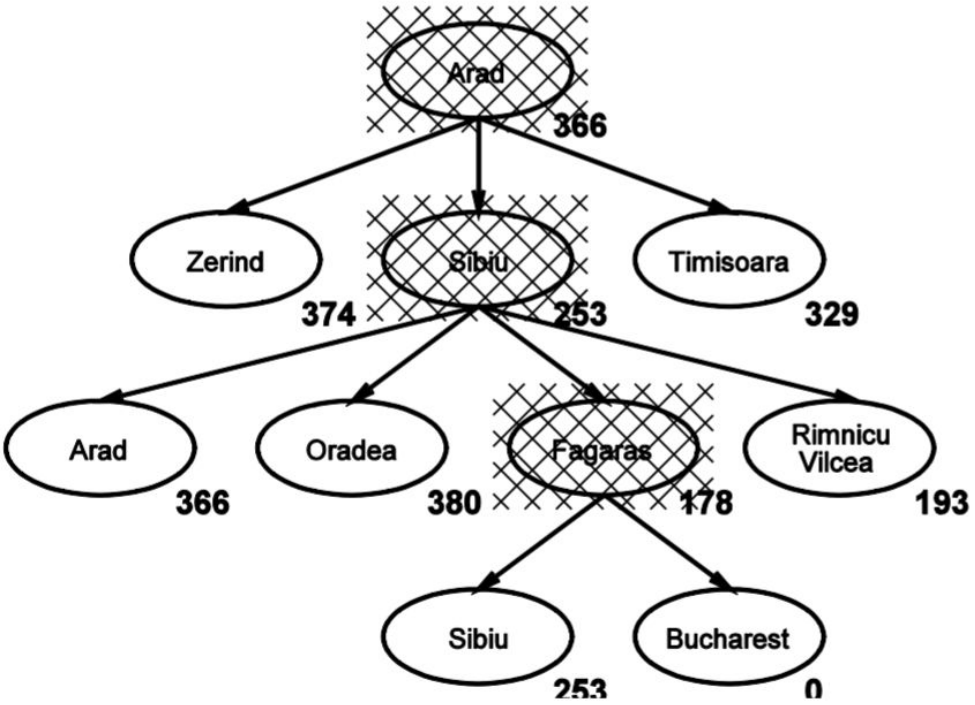
* use Priority Queue: insert successors in decreasing order of desirability

Special cases:
- greedy search
- A* search

Greedy Search

Heuristic: *closest* node to current node (lowest heuristic value)
Estimate cost of *n* to *goal* and then expand that node that *appears to be closest to goal*.
- but not necessary the best cost overall

Greedy search example



Complete?? No – can get stuck in loops, e.g., Iasi to Fagaras
Iasi → Neamt → Iasi → Neamt →
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

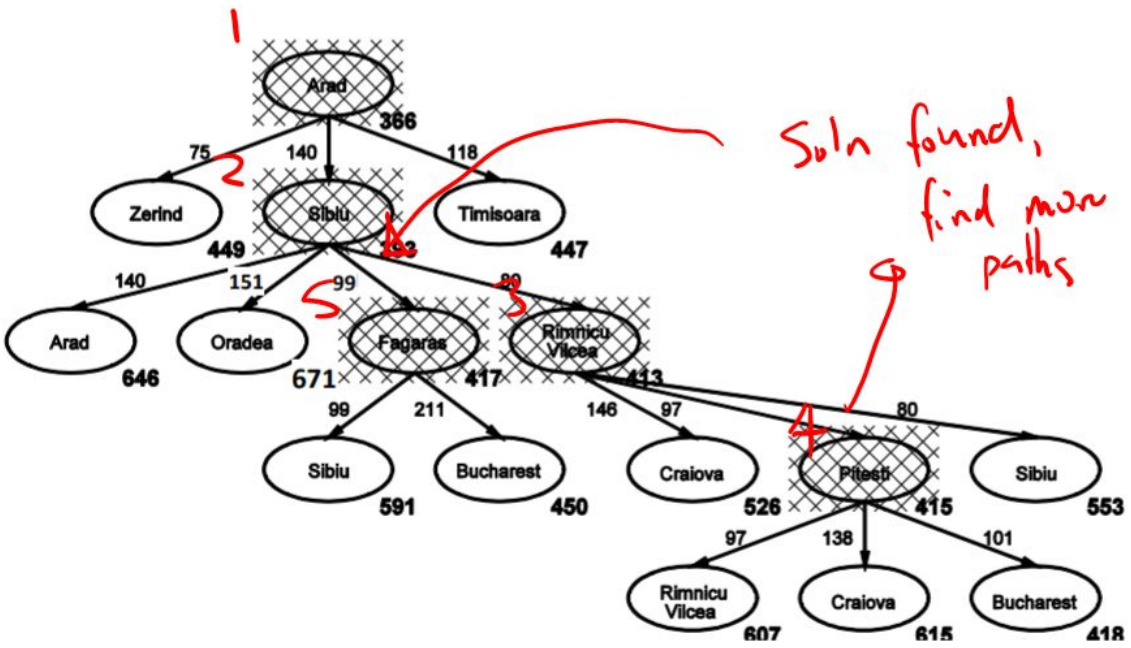
Optimal?? No

A* Search

Combination of uniform-cost search and greedy

True cost: total cost of true path to goal

A* looks for the best solution so it keeps searching even after it finds the goal node.
Example:



A* is optimal

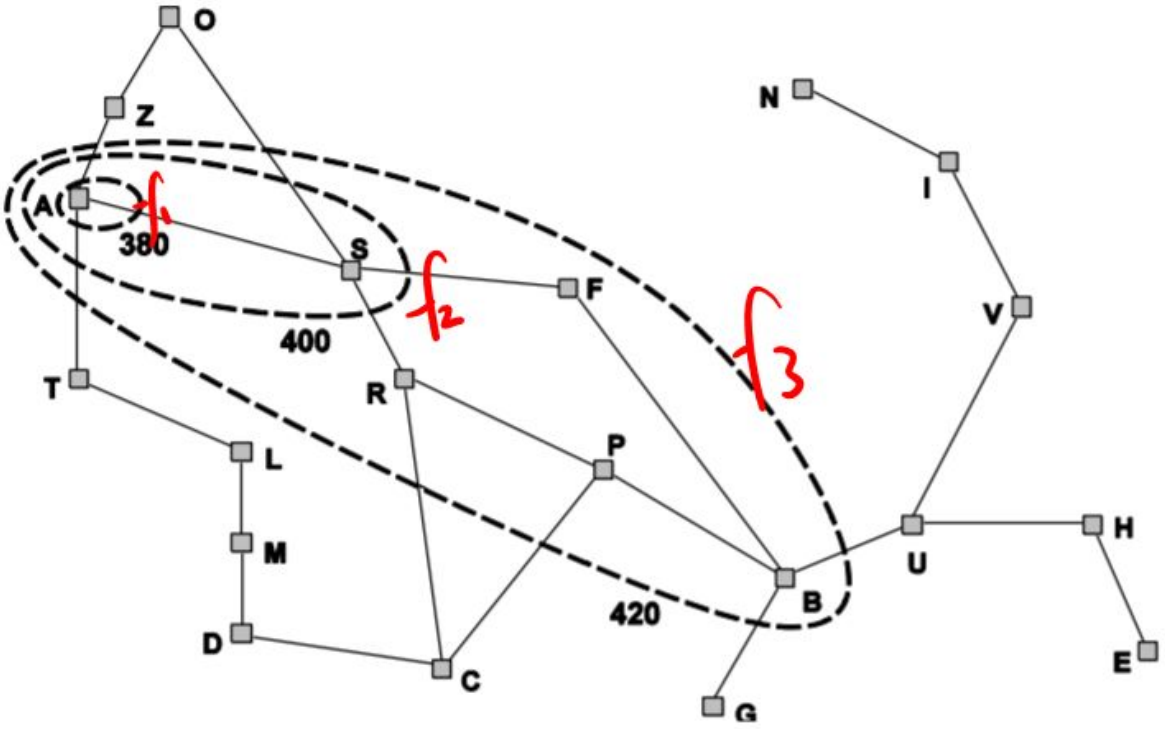
- it works by adds “f-contours” (big circle-y thingoes 🤔) towards the goal

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished



A* changes it's algorithm depending on the heuristic:
h(n) = Nodes to Goal
g(n) = Nodes from root

If *h(n) is very high relative to g(n),* then only h(n) plays a role and A* turns into ***greedy (best-heuristic-value-first) search***.

If *h(n) is 0, and only g(n) plays a role* (as in we've reached the goal), then A* turns into ***uniform-cost search (optimal solution)***

If *h(n) is always lower than (or equal to) the cost of moving from n to the goal* (the true cost), then **A* is guaranteed to find a shortest path**. The lower h(n) is, the more node A* expands, making it slower.

If *h(n) is sometimes greater than the cost of moving from n to the goal*, then **A* is not guaranteed to find a shortest path, but it can run faster.**

If *h(n) is exactly equal to the cost of moving from n to the goal*, then **A* will only follow the best path** and never expand anything else, making it *very fast*

Well-known Heuristic Functions:

1. **Manhattan distance** - allow 4 directions of movement (up, down, left, right)
2. **Diagonal distance** - allow 8 directions of movement (+ diagonal)
3. **Euclidean distance** - allow any direction of movement

Lecture 7 (Week 4) - Games Problems

Games vs Search Problems

“Unpredictable” opponent -> solution is a “oh shit he knows what he’s doing” plan

Limited resources (time limit): may not be able to “lookahead” forever
- much approximate

Methods:
- Algorithm for perfect play (Von Neumann, 1994)
  - (minimax) https://en.wikipedia.org/wiki/Minimax_theorem
- Finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952-57)
- Pruning irrelevant plans to reduce costs (McCarthy, 1956)

Game Types	Deterministic	Chance (*some element of randomness*)
Perfect Information	Chess, checkers, go, othello	Backgammon, Monopoly
Imperfect Information	Mastermind - like that game we played in Declarative Programming, Hangman, Battleship	bridge, poker (*can’t see opponent’s hand*), Scrabble, nuclear war (I think our lecturer is trying to tel us something... RIP)

Represent as a Search Problem

A strategic two-player game is defined as:
- ➔ An initial state
- ➔ Has actions
- ➔ Terminal test (i.e. end states like win, lost draw)
- ➔ Utility function (i.e. a numeric reward or score for an outcome)
  - Chess: Win + 1, Stalemate 0, Loss -1
  - Poker: Cash won or lost $$$$$$$$$

In a zero-sum game with 2 players, each player’s utility for a state are equal and opposite.

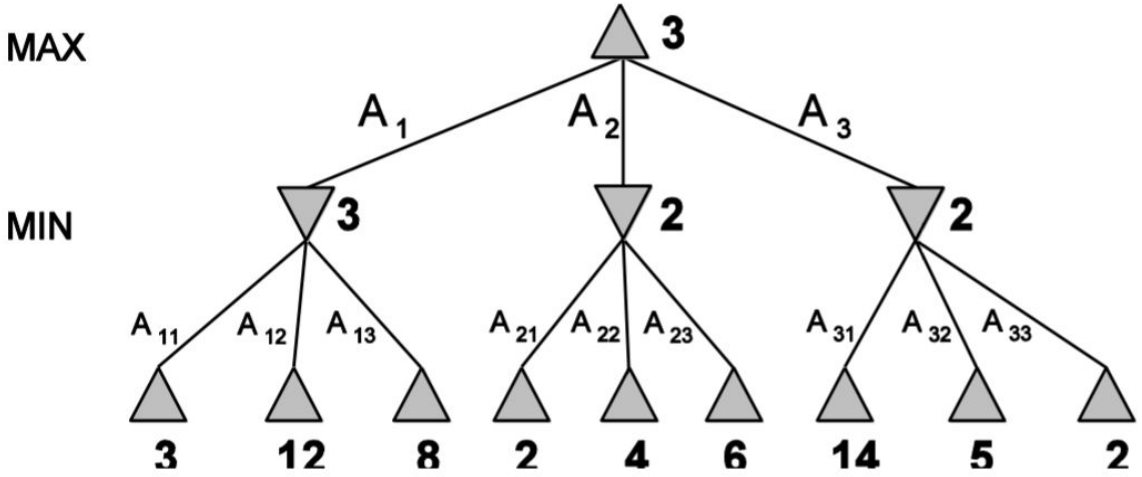
Minimax

*Perfect play for a **deterministic**, **perfect-information** game.*

- ➔ Choose move with *highest minimax value*
- ➔ Assume that the opponent is rational -- they also want to max their returns

Max and Min take turns, Max wants to Maximise their utility value (from the possible values on their turn), Min wants to Minimise their utility value (from the possible values on their turn).

Example: 2-ply (looking 2 steps ahead) game (with perfect play)



Bottom values: ***Utility values***.

- ➔ Remember that the opponent is rational, so instead of choosing A3 and then hoping the opponent chooses A31 (14), they'll chose A33 (2).
- ➔ We should choose A1 so the min value they can choose is A11 (3).

Expand for every node and propagate values back up the tree.

function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**
    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)
  end
  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state*, *game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**
    return UTILITY[*game*](*state*)
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

In terms of properties, what is minimax similar to?
Depth First Search: Because it goes all the way to the *terminal* state

Properties of minimax:

- Complete: Yes, if the tree is finite (chess has specific rules for this)
- Optimal: Yes, against an optimal opponent. Otherwise our utility function becomes invalid
- Time Complexity: $O(b^m)$
- Space Complexity: O(bm) (depth-first exploration)

Not much memory needed but time complexity is not efficient enough

For chess, b = 35, m = 100 for "reasonable" games

- Exact solution completely infeasible

So should be a **zero-sum** game (if one player wins, the other *must* lose)

Resource Limits of minimax

Minimax is extra *thicc* so we need some way to remove irrelevant paths

Standard Approaches:

- Cutoff Test
  - E.g. depth limit (perhaps add quiescence search)
  - Quiescence Search is an algorithm used to evaluate minimax game trees
- Evaluation function
  - = estimated desirability of position
  - Our heuristics
    - importance of pieces

Evaluation Functions:
E.g. here's an evaluation function for chess

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

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens)
etc.

Minimax favours the end state - any exact values don't matter as long as the order of plays results in a good end state.

Ply = Lookahead

MINIMAXCUTOFF is identical to MINIMAXVALUE except

1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Does it work in practice?

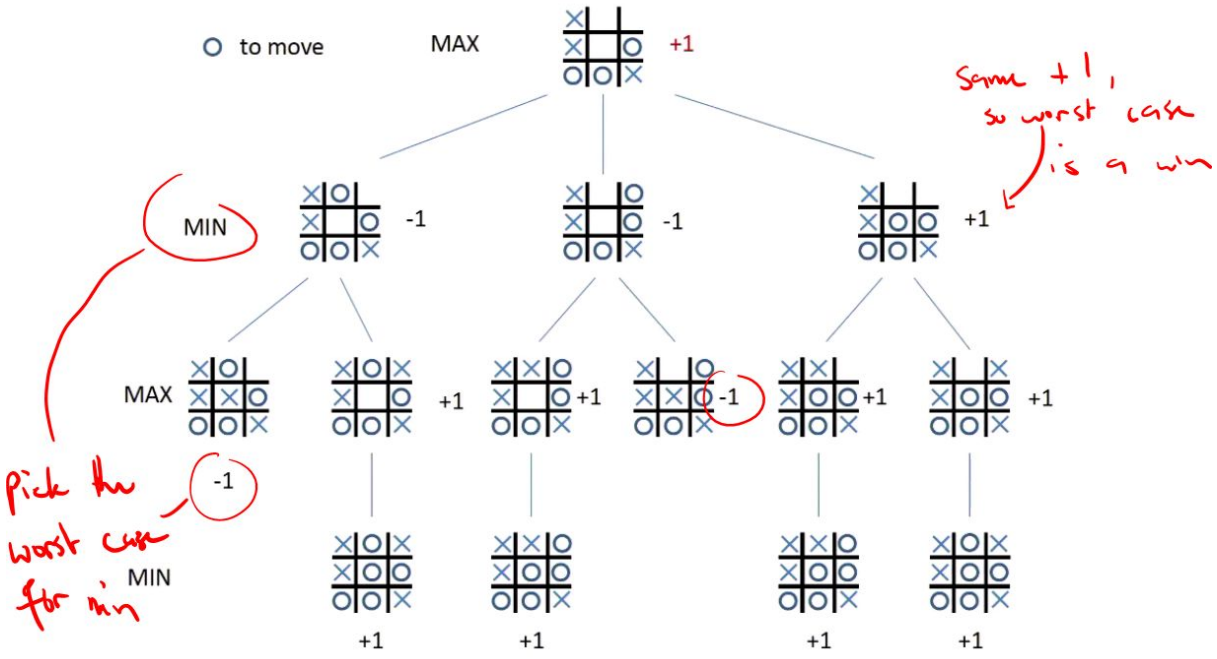
$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply ≈ human novice
8-ply ≈ typical PC, human master
12-ply ≈ IBM's Deep Blue, Kasparov

Lecture 8 (Week 4) - Game Playing

Applying Minimax to Knots and crosses:



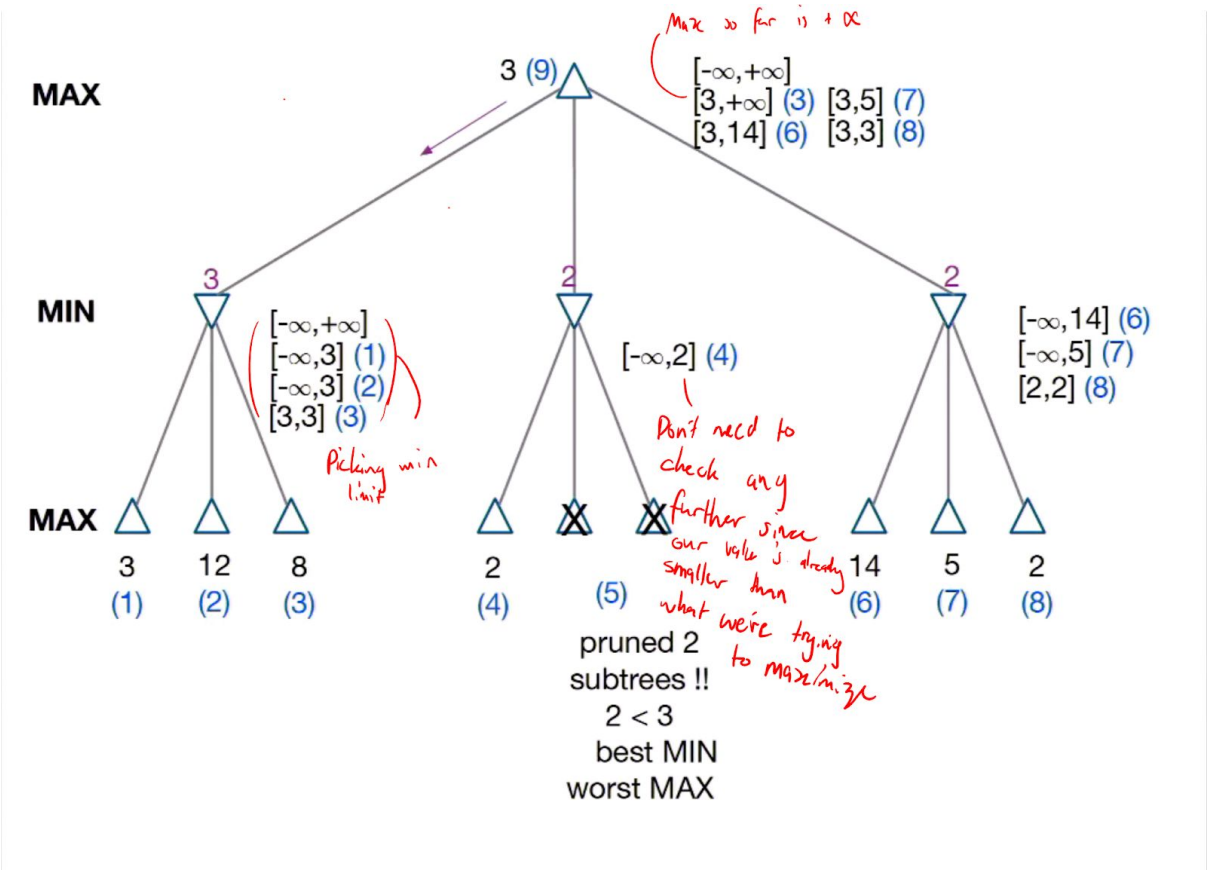
So we'd pick the center square since we want to *maximize our wins* while *minimizing loss*.

Applying Alpha-Beta pruning to an example:

Start at maximum range of values

- our minimum value is -ve infinity and our max value is +ve infinity.
- Keep track of our lowest mins and our highest highs
- If anything is lower than what we currently have, ignore the rest of the tree

Follow the blue numbers for directions. (same example as above)



Start off the same way as minimax, expanding subtrees at n-ply lookahead to see values.
Keep track of what's currently seen as the "lowest" and "highest" values.
Basically in step (4) our current maximum value we can get is 3. In the 2nd subtree, there is *no way we'll get a value higher than 2* so we don't care about that value (since we're trying to maximize).
In the 3rd subtree, we now see our minimum values are way higher so we start to update the origin.
However once we see [2,2] it's over and we go back to the 1st subtree.

Properties of alpha-beta pruning:

Pruning *does not affect the final result*! (lul never using minimax again)
Good move ordering improves effectiveness of pruning

Best case *perfect ordering* time complexity = $O(b^{m/2})$

- Means we can *double our search in the same time*

Called alpha-beta because alpha is our best max value and beta is our worst min value.

The algorithm is literally just ***minimax*** **but you** ***prune values that you don't need to check***.

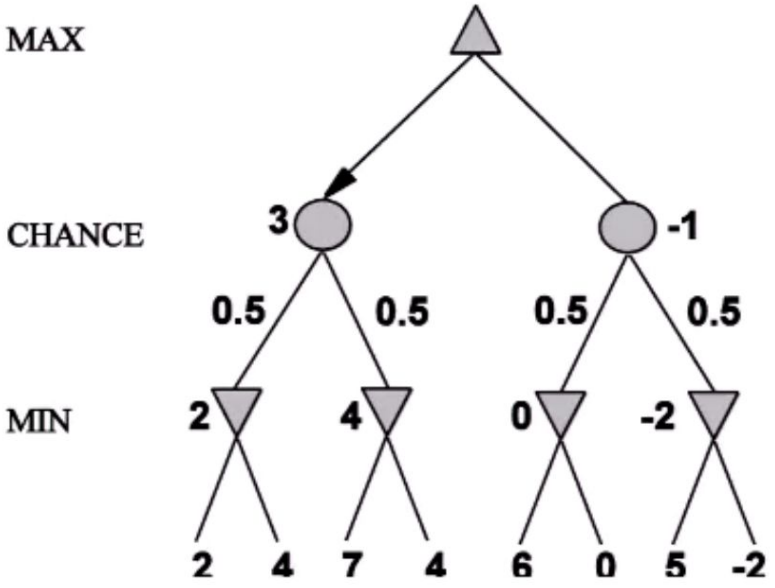
Some other pruning algorithms to check out: Negamax and SSstar pruning. They implement node ordering and some cool stuff.

Drawbacks of alpha-beta pruning:

- It's not feasible to search the entire game tree so a depth limit has to be set
- Calculates all legal moves instead of just "good" ones
- You can improve this by *scoring each node to expand*

Nondeterministic Games

You can actually apply minimax and similar algorithms to *nondeterministic games* by *implementing a chance node* and averaging out the utility value.



Using this *chance node* is called Expectiminimax. (what is this harry potter??? 🪄)
In ***non-deterministic games***, as **depth increases, the probability of reaching a given node shrinks** and **therefore the value of a lookahead is diminished**.

This means in certain games, minimax and alpha-beta pruning is *much less effective*.

Expectiminimax works best when *only going a few ply deep*.
Also, many *non-deterministic games are actually psychological* (poker) so chance is hard to derive.

Lecture 9 (Week 5) - Machine Learning in Game Search

This subject : *Reinforcement learning*

Challenges in game agent design:

- ➔ Handling each stage of the game separately
  - ◆ Compiling a “book” or knowledge base for opening/ending games
  - ◆ just lookup best opening move / best ending move for current state
- ➔ Search Parameters
  - ◆ search control learning
    - look at similar problems
    - can extract some rules that could improve time complexity
  - ◆ eg. when determining order in alpha beta pruning - could half the time complexity
- ➔ Weights in evaluation functions
  - ◆ already have evaluation function, need to find best set of values
- ➔ Finding good features for evaluation functions
  - ◆ don’t have evaluation function’s features, need to come up with these features
  - ◆ not much research in this area

Book Learning

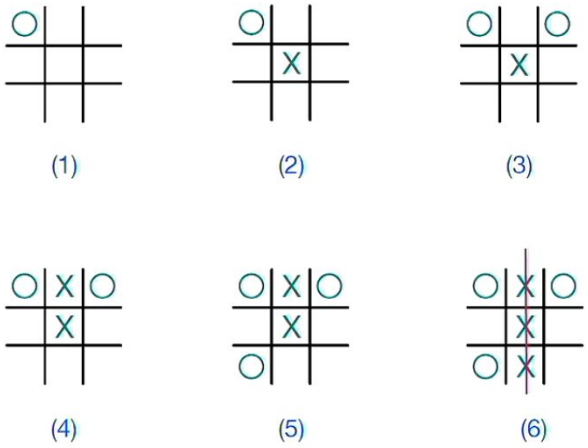
Aim: **Learn a sequence of moves for important positions**
Example 1: Chess

- A ton of opening moves
- For every position seen in an opening game, remember the move taken and the final outcome

Example 2:

- Learn from mistakes
- Identify which moves lead to a loss and whether there was a better alternative

How do we figure out which moves were important?
From the cross' player point of view, which was the **decisive move**?

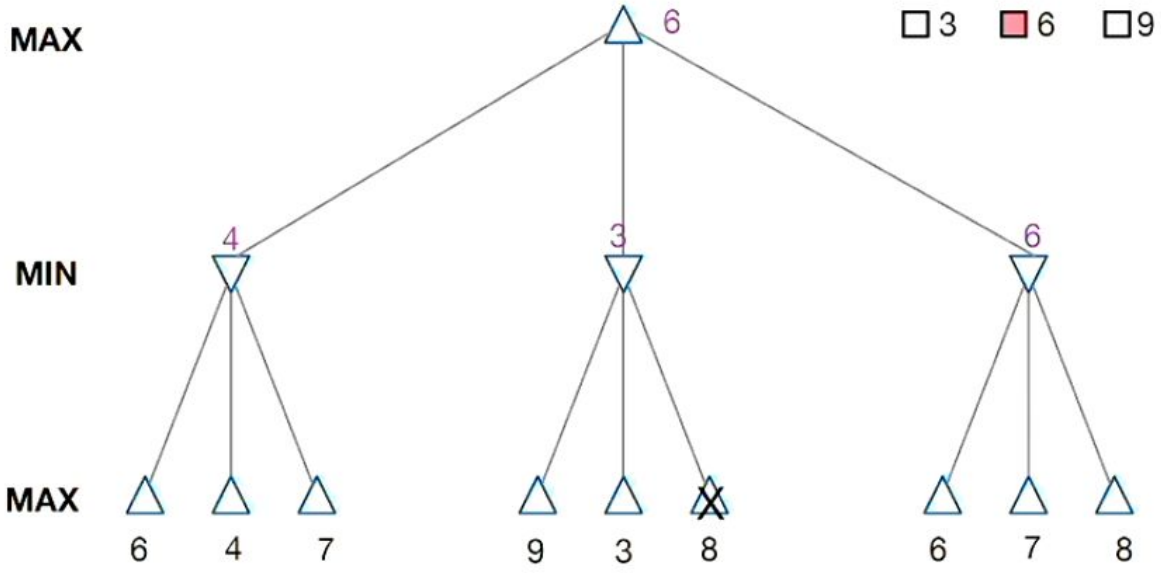


Search Control Learning

Aim: **Learn how to make search more efficient**

Example 1: *Order of move generation affects alpha-beta pruning*

➔ Learn a preferred order for generating possible moves to maximise pruning of subtrees



Example 2: *Can we vary the cut-off depth?*

➔ Learn a classifier to predict which depth we should search to based on current states

Evaluation Functions

For chess we typically have a *linear* weighted sum of features

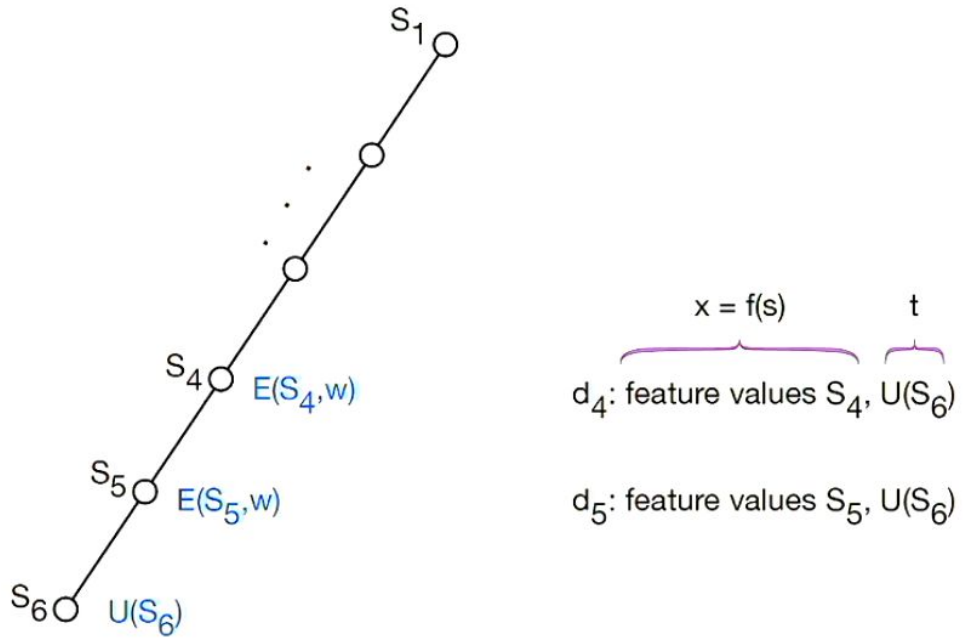
➔ each piece can be a feature

- ➔ how many pieces we lost
- ➔ how many queens are on the board etc...

Aim:
Adjust weights in evaluation function based on experience of their ability to predict the ***true final utility****.*

Using a *linear weighted sum of features*

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$
$$= \sum_{i=1}^{n} w_i f_i(s) = w \cdot f(s)$$



Gradient Descent Learning

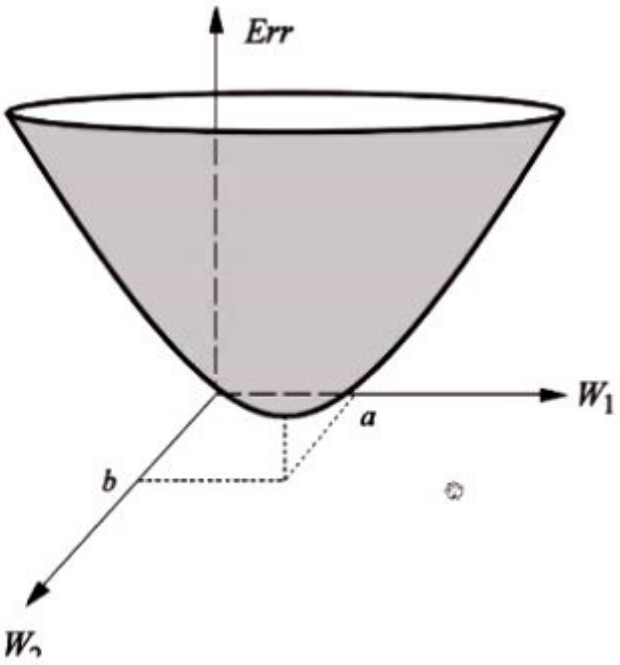
Supervised Learning - have a set of training examples

There was a bunch of maths mumbo-jumbo here but basically we set the machine learning up and then compare it against a model and alter the weights until we have something that closely resembles the true output.

This is done by defining a *error function E* which keeps track of the square of the differences between the *actual output* and the predicted/desired output *t*.

We can then alter the weights until the error is minimized.

A graph of the error function will look like this:



Move in the *steepest downhill direction*.

FYI: Gradient descent approach is the basis of the *back-propagation* algorithms in neural networks for supervised learning.

Weight Update Rule

To calculate the gradients of training data

- just plug in the two values to get the weight value

The minimum point is where a, b is
Also, AlphaGo is absolutely broken:
https://www.youtube.com/watch?v=9xlSy9F5WtE

Lecture 10 (Week 5) - Machine Learning in Game Search

Weight Update Rule

We need to derive the time difference for the error function. Time to get into c a l c u l u s yaaaaaaaaaay (i want to die)

$$\frac{\partial E}{\partial w_i} = ? \qquad E = \frac{1}{2}(t-z)^2 \qquad z = \sum w_i f_i(s)$$

Chain rule: $y = y(u)\ and\ u = u(x)$ $\quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \times \frac{\partial u}{\partial x}$

$$\frac{\partial E}{\partial w_i} = \left(\frac{\partial}{\partial z}\left[\frac{1}{2}(t-z)^2\right]\right)\left(\frac{\partial z}{\partial w_i}\right)$$

$$= \left(\frac{1}{2} \times 2\,(t-z)\,(-1)\right)\left(\frac{\partial z}{\partial w_i}\right)$$

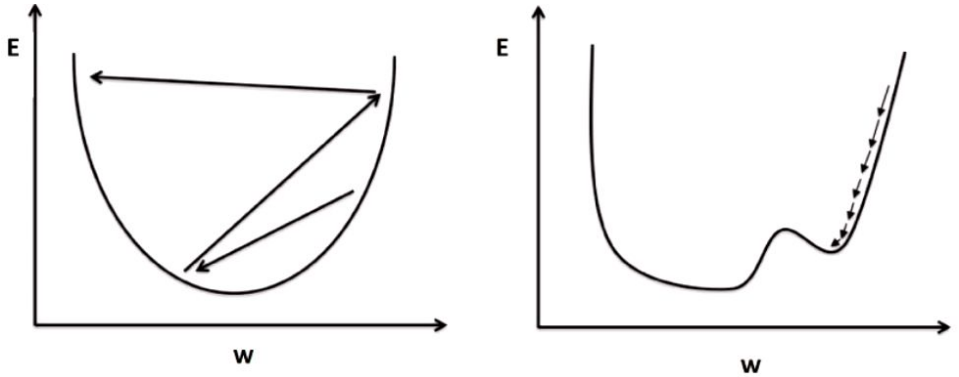
$$= (z-t)\left(\frac{\partial z}{\partial w_i}\right)$$

$$= (z-t) f_i(s)$$

$$w_i \leftarrow w_i - \eta(z-t) f_i(s) \qquad \eta : \text{learning rate, } \eta = 0.1$$

Repeat for all examples

Ok i am super terrible at maths in general so go watch the lecture and tell me properly how this works pls 🙏🙏🙏



Large learning rater Small learning rater

Problems

Delayed Reinforcement: reward resulting from an action is delayed several steps

- laaaaaaaaaaaaaag
- slows down the learning

Credit Assignment: difficult to know which actions actually were responsible for the outcome

Temporal Difference Learning

- Supervised learning is nice for single step prediction
  - E.g. what's the weather tomorrow vs yesterdays'?

- Temporal Difference is for *multi-step* prediction
    - E.g. what's the weather tomorrow vs 3 days ago?
- Correctness of prediction not known until several steps later
- Intermediate steps provide information about correctness of prediction
- TD learning is a *form of reinforcement learning*
- *First **general algorithm** which could be applied to 50 different atari games when it came out*

TDLeaf Algorithm

Type of Temporal Difference Learning

➔ Combines Minimax with Temporal Difference

Aim: Update the weights in evaluation function to **reduce differences** in rewards predicted at **different levels** in search tree

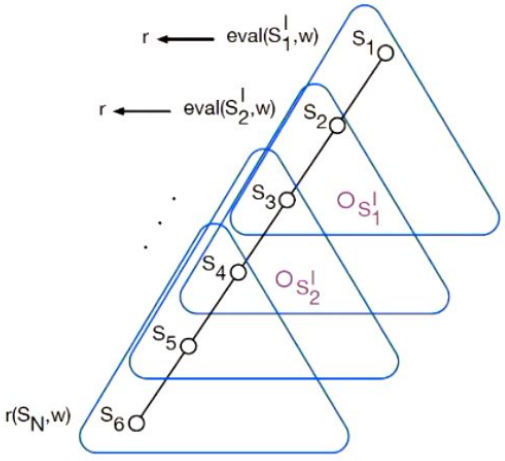
➔ Good functions should be stable from one move to the next

$eval(s, w)$: $[-\infty, +\infty]$

$S_1, \ldots, S_N$: N states that occurred in a game

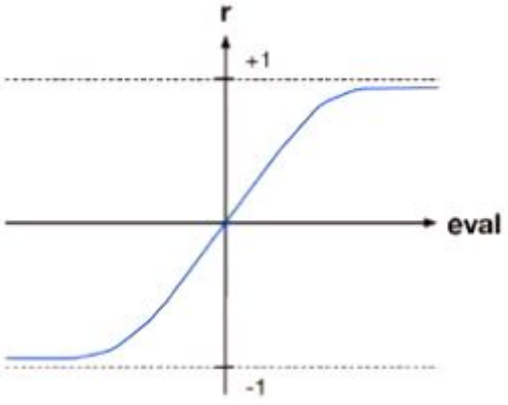
$r(S_N) \in \{+1, 0, -1\}$

S_i^l: is best leaf found at max cut-off depth using MINIMAX search for S_i



A good evaluation function is defined by $r(S_i^l, w) = eval(S_i^l, w)$

➔ Should map eval[+∞,-∞] to r()[+1, -1]



-
- You can use tanh to squash it into this range

➔ Eventually you end up with this really ugly function:

For $i = 1, ..., N-1$

Compute temporal difference between successive states

error

$\hookrightarrow d_i = r(s^l_{i+1}, w) - r(s^l_i, w)$

Update each weight parameter w_j as follows

$$w_j \leftarrow w_j + \eta \sum_{i=1}^{N-1} \frac{\partial r(s^l_i, w)}{\partial w_j} \left[\sum_{m=1}^{N-1} \lambda^{m-i} d_m \right]$$

weighted sum of error

Current Weight

Learn rate

How is reward changing with respect to weight

No idea what she's talking about at this point, gonna have to revisit this

- ➔ Where Lambda gets closer to 1, the closer it is to the *final true reward*
- ➔ Where Lambda gets closer to 0, the closer it is *to the predicted award at the next state*

Environment for Learning

How do we make our AI learn?

- ➔ Learning from other examples
- ➔ Learn by playing against a skilled opponent
- ➔ Learn by playing against random moves
  - ◆ can't guarantee how much you can learn from this
- ➔ Learn by playing against yourself (lol sounds like a usual saturday night)
  - ◆ AlphaGo does this

Lecture 11 (Week 6?) - Constraint Satisfaction Problems

We're moving into the last topic of the first part
What do we do when the solution is a set of variables that satisfy a set of constraints?

Constraint Satisfaction Problems (CSPs)

Standard Search Problem:

➔ *State* is a “black box” - any old data structure that supports goal test, eval, successor

CSP is different, where:

➔ *State* is defined by the **variables** X with values from **domain** D_i

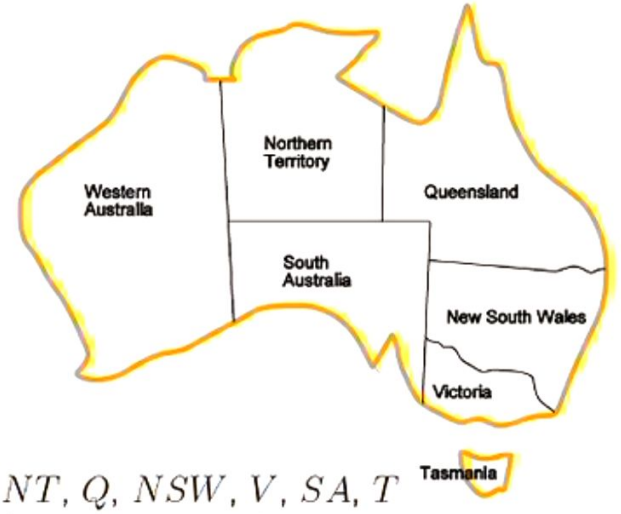
➔ *Goal* test is a set of **constraints** allowable combinations of values for subsets of variables

Basically the ***state is the variables*** and the ***goal is the constraints to be fulfilled.***

Allows useful general purpose algorithms with more power than standard search algorithms.

Example: Map-Coloring

Want to assign colours to every region without any same colours touching each other:



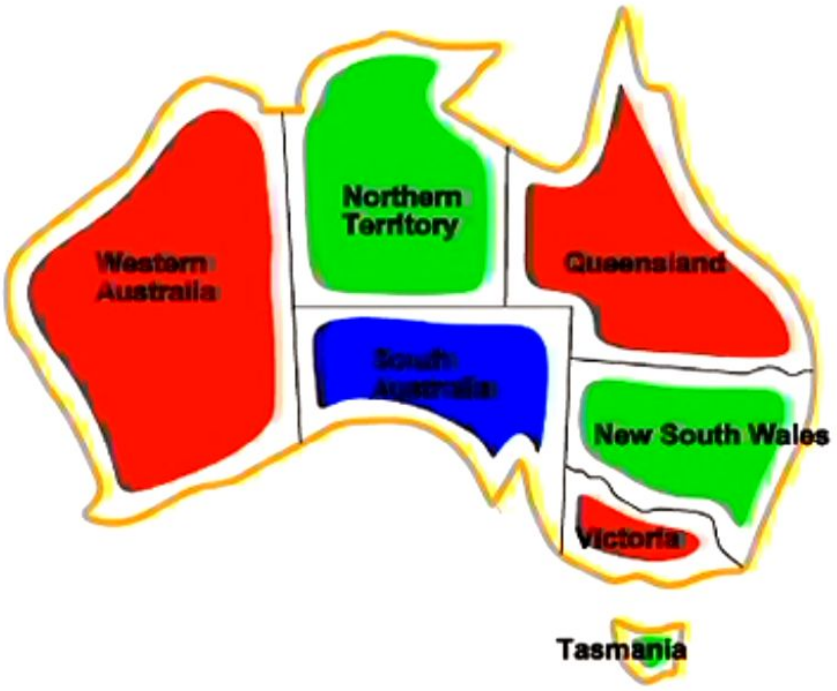
Variables WA, NT, Q, NSW, V, SA, T

Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$



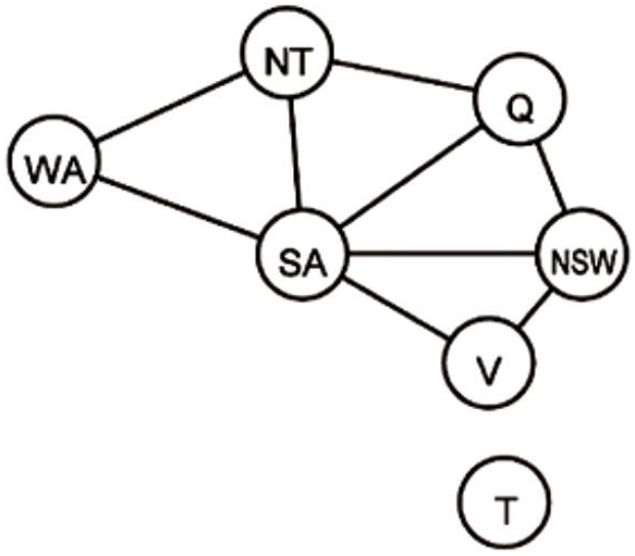
$Solutions$ are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green,$
$V = red, SA = blue, T = green\}$

Strategies:

- Assign a region a colour, and then filter out possible colours in neighbouring regions

Constraint Graph

We use Constraint Graphs to model CSPs:
Binary CSP: each constraint relates at most two variables
eg. SA != WA
Constraint Graph: nodes are variables, arcs show constraints



General Purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem.

Varieties of CSPs

Discrete Variables

- Finite domains: size d => $O(d^n)$ complete assignments where n is the number of variables in the CSP
- E.g. Boolean CSPs, incl. Boolean satisfiability (NP-complete) infinite domains (integers, strings, etc.)
- Job scheduling
- Need a constraint language, e.g. $StartJob_1 + 5 <= StartJob_3$
- Linear constraints solvable, nonlinear undecidable

Continuous Variables

- Start/end times for hubble telescope observations
- Linear constraints solvable in polynomial time by linear programming methods

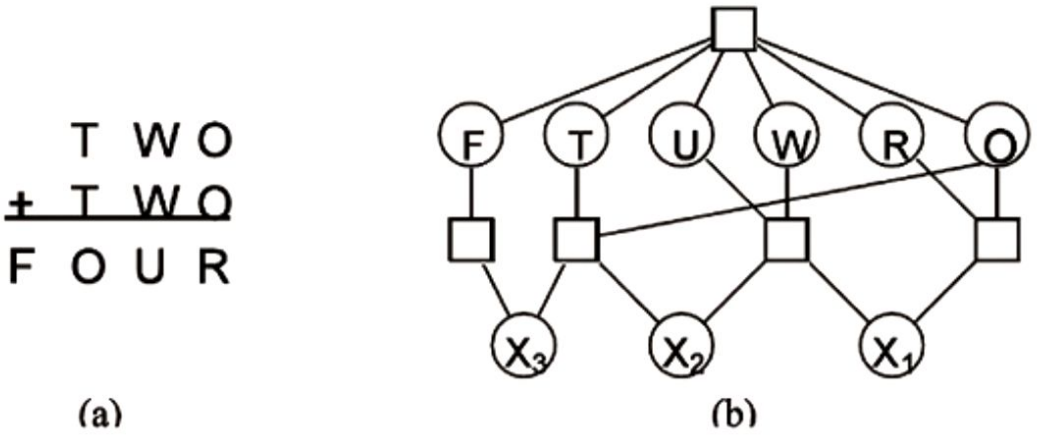
Unary Constraint: Involve a single variable
E.g. Sam =/= Has GF

Binary Constraint: involve pairs of variables
E.g. Sam =/= Mitsuru

Higher Order: Constraints involve 3 or more variables,
E.g. cryptarithmetic column constraints

Preferences (soft constraints), e.g. red is better than green or this food costs more than the other one

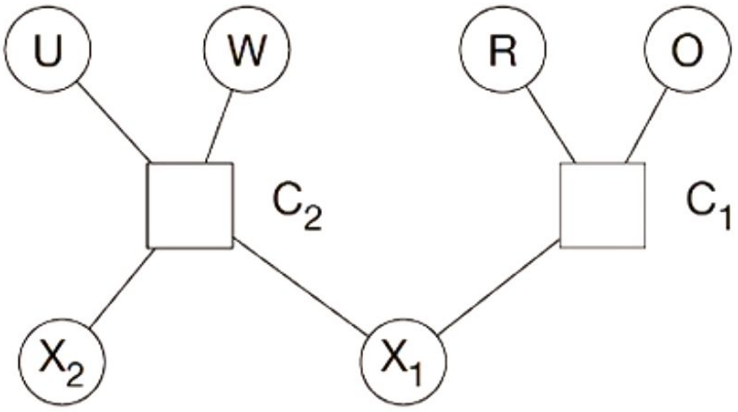
Example: Cryptarithmetic Puzzle



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints
$alldiff(F, T, U, W, R, O)$
$O + O = R + 10 \cdot X_1$, etc.

b) is the constraint graph
Use an edge to represent a constraint we have
Use a box to represent a *higher-order constraint*

C_1: O+O = R + $10X_1$
C_2: W+W+X_1 = U+$10X_2$
C_3:...



Keep adding them all and eventually we end up with the original graph

Real-World CSPs

Assignment Problems
Timetabling Problems
Hardware configuration
Spreadsheets
Transportation Scheduling
Factory Scheduling
Floorplanning
My sleep schedule

Standard Search Formation (Incremental)

Let's start with the straightforward dumb approach then fix it

- Initial state: the empty assignment, 0
- Successor function: Assign a value to an unassigned variable that does not conflict with current assignment
- Goal Test: The current assignment is complete

This works for literally all CSPs but it's also really slow af

However, we forget that the path doesn't matter in CSPs
Sam = No GF then Bob = Has GF is the same in both orders

Backtracking Search

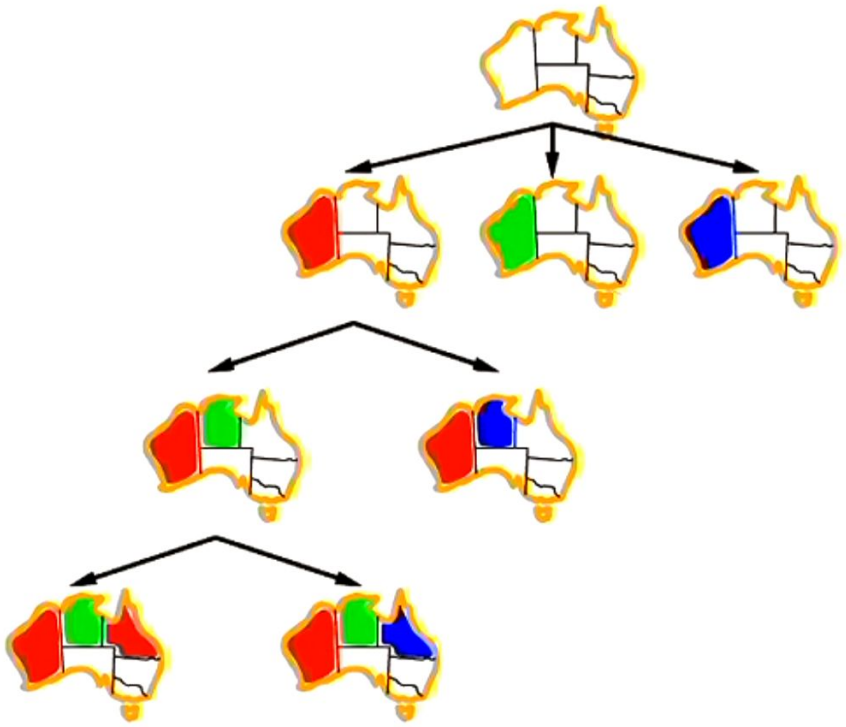
Only need to consider assignments to a single variable at each node
Depth first search for CSPs with single-variable assignments is called *backtracking* search
Backtracking search is the basic uniformed algorithm for CSPs
Time Complexity is bad
Can solve n-queens problem for n approx 25.

- *Note: N-queens problem is the problem of putting n number of queens on a chess board such that they don't kill each other*

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```

Assign variables to each node until it works lol



Improving Backtracking Efficiency

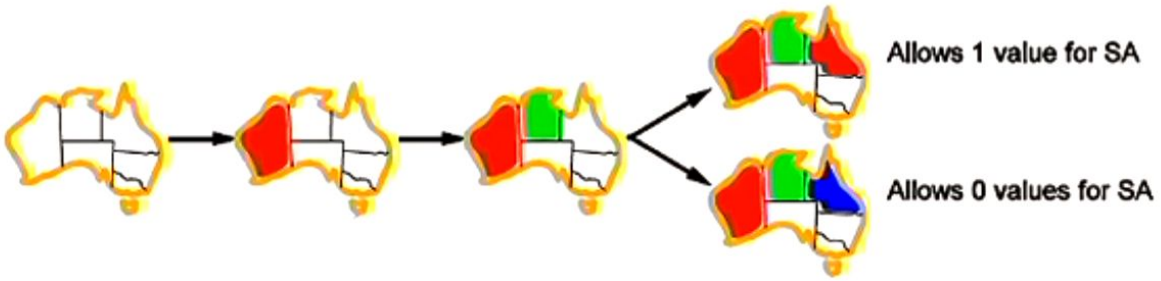
- ➔ Which variable should be assigned next?
- ➔ Which order should things be tried?
- ➔ Can we detect inevitable failure early?
  - ◆ (Like AlphaBeta)
- ➔ Can we take advantage of problem structure?
  - ◆ (By looking at the constraint graph

Minimum remaining values

- ➔ Choose the variable with the fewest possible values
- ➔ Tiebreaking can be done by using a *degree heuristic*
  - ◆ Choose the variable with the most constraints on remaining variables
  - ◆ Basically which value can you remove which leaves the fewest values remaining

Least Constraining Variable heuristic

- ➔ Given a variable, choose the least constraining value
- ➔ The one that rules out the fewest values in the remaining variables
- ➔ Combining these heuristics makes 1000 queens problem feasible



Lecture 12 (Week 6)

Forward Checking

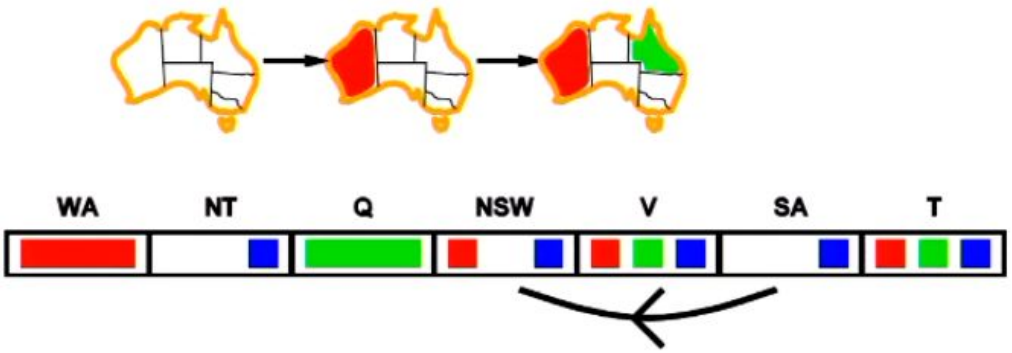
1. Keep track of remaining legal moves for unassigned variables
2. Terminate search when any variable has no legal moves

➔ Propagates information from assigned to unassigned variables
➔ Doesn't provide early detection for all failures
➔ Use **constraint propagation** to repeatedly enforce constraints locally

Arc Consistency

Form of constraint propagation

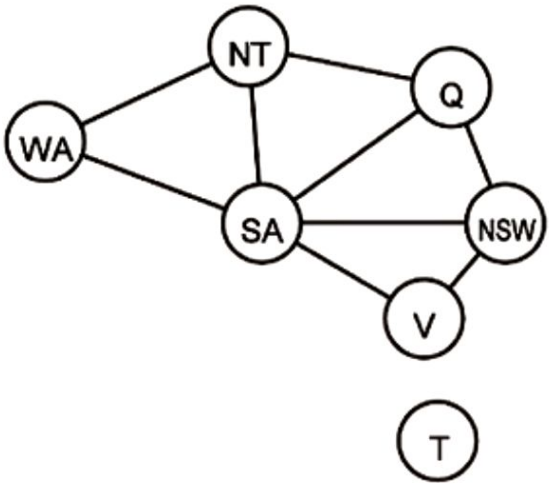
X → Y is *arc consistent* iff
➔ for every value x of X, there is at least one value y of Y that satisfies the constraint between X and Y
➔ X is arc consistent with respect to Y
➔ same as `AC(X, Y)`



➔ if X loses a value, neighbours of X must be rechecked

★ Arc Consistency detects failure earlier than forward checking
★ Can be run as a preprocessor or after each assignment
★ *AC-3* - $O(n^2d^3)$, which can be reduced to $O(n^2d^2)$ with some smart implementation
  - but detecting *all* is NP-hard
  - more efficient that backtracking, $O(d^n)$

Alternative approach: exploit structure of the constraint graph to find independent subproblems



Tasmania and Mainland are **independent subproblems**.

➔ identifiable as **connected components** of a constraint graph
➔ worst case: $O(\frac{n}{c}d^c)$

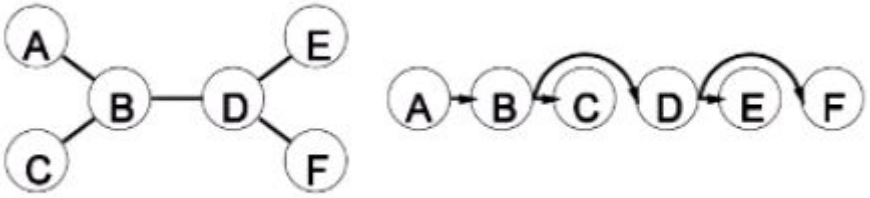
Tree-Structured CSP

Theorem: If a constraint graph has **no loops**, the CSP can be solved in $O(nd^2)$ time

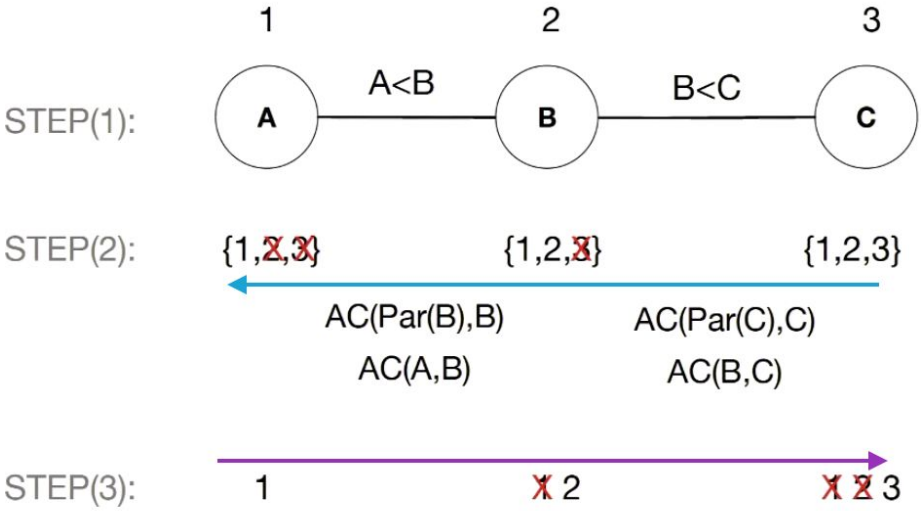
- General CSPs have worst case of $O(d^n)$

1. Choose a variable as the root

2. Order variables from root to leaves such that every node's parent precedes it in the ordering



3. For *j* from *n* down to *2*, apply `MakeArcConsistent(Parent(Xj), Xj)`.
4. For *j* from *1* to *n*, assign a value to `Xj` that is consistent with `Parent(Xj)`.



Example:

Nearly Tree-Structured CSPs

Conditioning: instantiate a variable, prune its neighbours' domains
Cutset Conditioning: instantiate a set of variables (the cutset) such that the remaining constraint graph is a tree

- ➔ best to find the smallest possible cutset, but this is NP-hard
- ➔ usually can find a relatively small cutset

Local Search

A type of *iterative* algorithm that tries to change one variable assignment at a time.
Min-Conflict Heuristic:

- ➔ get the value that minimises the number of conflicts

Can solve most types of practical problems in linear time.

Lecture 13 (Week 7) - IBM Guest Lecture

Lecture 14 (Week 7) - Feedback Quiz

Lecture 15 (Week 8) - Making Complex Decisions - Auctions

~~Haha time to auction off my body so I can pass~~

Auctions can be used to determine the "true value" of a good that was previously unknown (to the seller).

Complex Decisions

So far we've seen *single agent* search or *pairs of agents* for adversarial game search.
In practise, complex applications can involve *multi-agent* systems, where multiple agents are competing for a scarce resource, e.g.

- ➔ Farmers needing water allocation in an irrigation systems
- ➔ Students fighting for internships
- ➔ People arguing over me (hahaaaa jokes that would never happen i want to die pls let me pass)

Traditional Approach: a **centralised authority** makes allocation to each agent.

- ★ how to ensure the scarce resource goes to those who value it most
- ★ how to ensure the owners of the resource maximise their financial return

Example - Selling and Buying Land

What concerns do you have (1) as a *buyer*, (2) as a *seller*?

- ➔ What's your budget
- ➔ Who else is buying
- ➔ When do you want to enter the auction?

Auctioneer has to be neutral af

- ➔ What's the starting value
- ➔ How big should be the bids
- ➔ When to close the auction

Mechanism Design

Problem of how to design a "game" that results in **maximising a global utility function** in a multi-agent system, given that each agent pursues their own (*selfish*) rational strategy

In AI we aim to construct smart systems out of simpler systems to achieve goals that are beyond the reach of any single system

- ➔ ***Single Item Auction***: There is a single item for sale and there are n players bidding for the item. Each player has a internal value - maximize the value vs payment.
- ➔ ***Combinatorial Auctions***: a generalisation of single item auctions and considers the problem of selling a set of T items.

Mechanism for an auction consists of:

1. a **language** to describe the allowable strategies an agent can follow
2. a communication **protocol** for communicating bids from bidders to the auctioneer
3. an **outcome rule**, used by auctioneer to determine the outcome
    a. when to stop the auction
    b. who's the winner
    c. how much they have to pay

Auction Protocol Dimensions

Winner Determination: Which bidder wins and what do they pay?

- **First price auctions**: highest bid is paid
- **Second-price auctions**: second highest bid is paid

Knowledge of Bids:

- **Open cry**: Every bidder can see all bids from all other bidders
- **Sealed**: Every bidder can't see the other bids

Order of bids:

- **One-shot**: each bidder can only make one bid
- **Ascending**: each successive bid must exceed the previous bid
- **Descending**: start from a high price and go down and the first to bid wins

Number of Goods: How many goods are for sale

- **Single good**: only one indivisible good is for sale
- **Multiple goods**: many goods available in the auction
    - bids can include both the price and number of goods wanted by the bidder
    - *Combinatorial Auction*

Factors affecting Mechanism Design

Common Value:

- The worth of the good is the same for all bidders

Private Value:

- Each bidder has a utility value that reflects the worth of good to the bidder

The job of the auctioneer is to try and get the *private value* of all the bidders

Desirable Properties of an Auction

Efficient:

➔ The goods go to the agent who values them the most

Discourage Collusion:

➔ The auction mechanism should discourage illegal or unfair agreements between two or more bidders to manipulate prices

Dominant strategy:

➔ There exists a *dominant* strategy for bidders where a strategy for bidders where a strategy is dominant if it works better than other strategy

Truth-Revealing:

➔ The dominant strategy results in bidders revealing their true private value for the good

➔ Modern online auctions require autonomous agents who can bid on our behalf

➔ These agents need to model user's preferences for their bidding strategy

➔ These agents need a representation language for bids

English Auction

You, an average auction

- First price, open-cry, ascending auction

Dominant strategy:

- Keep bidding in small bids while the cost is lower than your value

Properties:

- It's ***efficient***, provided the starting price is realistic
- Can suffer from the *winner's curse*
    - Has the winner valued the good too highly cause no-one else made that high big?
- Can be susceptible to *collusion*
    - Bidders can agree beforehand to keep bids artificially low
    - Auctioneers can plant dummy or bogus bidders to inflate prices

Lecture 16 (Week 8) - Auctions Continued

Dutch Auction

Me, an intellectual
- First price, open-cry *descending auction*

Dominant Strategy:
- Just bid your price

Properties:
- Really fast and finishes quickly
- Start at a *really high and extremely expensive* value and then go down until the first person bids.
- Still susceptible to *collusion* between bidders
- Still susceptible to *winner's curse* because don't know other bidders' private values

Then the lecturer talks about some random auction cases with collusion + winner's curse

First-Price Sealed-Bid Auction

How it works:
- Each bidder can only make a single bid and you can't see each other's bids
- Winner is simply who made the highest bid

There is no dominant strategy for this since you have no clue what everyone else's bids are

Properties:
- Might not be efficient since the agent with the highest value might *not* win the auction
- Really hard to have collusion in, and it's faster

Vickrey Auction

Same as above, except whoever wins pays the *second highest price*

Dominant Strategy:
- Bid your *true value* since if you win you don't have to pay it

Properties:
- Efficient and truth-revealing
- Hard for *collusion* to occur

- No winner's curse
- A bit counter intuitive for human bidders
- Computational simplicity makes it popular for use in multi-agent AI systems and online auctions

Online Auctions

Ebay lol

- Provides limited support for *proxy bidding*
- You can sell goods where their value is unclear

Usually online auctions use a *combination of English and Vickrey auctions* where:

- The highest winning bid is *not shown*
- Current second highest bid is public (to establish value of good)
- Bidders can make multiple bids
- A deadline is used

However, susceptible to ***sniping*** where you just bid something with ~1 second left

➔ Fix this by extending the deadline every time a bid is made

Case Study - Google AdWords

Google auctions off advertising space to advertisers using auctions using a Vickrey auction.
Advertisers bid their space/price

★ *All of this happens in **real time** (10^4 auctions/second wtf)*

Summary

In the exam:

➔ Compare and contrast different types of auctions
➔ Describe properties of a given type of auction
➔ Select the most appropriate type of auction for a given application

Wtf 30 minute lecture?????????

Lecture 17 (Week 9) - Uncertainty

Let action A_t = leave for airport *t* minutes before flight.
Will A_t get me there on time?

Problems:

1. Partial observability (road state, other drivers' plans, etc.)
2. Noisy sensors (radio traffic reports, google maps traffic)
    a. data might not be real time
3. Uncertainty in action outcomes (flat tyre, etc)

4. Immense complexity of modelling and predicting traffic

Hence a purely logical approach either risks falsehood or leads to conclusions that are too weak for decision making.
"A_{25} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact and I have a gf etc. etc."

Methods for *handling* uncertainty

Nonmonotonic logic:

- Assume my car does not have a flat tire
- Assume I don't have a gf
- Assume A_{25} works unless contradicted by evidence

Issues: What assumptions are reasonable? How to handle contradiction?
Probably not going to look at this too much

Rules with confidence factors:

- A_{25} = 0.3 get there on time
- However consider the example:
- Sprinkler -> 0.99 Wet grass
- Wet grass -> 0.7 Rain.

Issues with combination: e.g. *Sprinkler causes Rain???*

Probability:

Given the available evidence,
A_{25} will get me there on time with probability 0.04

Will be looking at this alot in the subject

Fuzzy Logic handles *degree of truth not uncertainty*

Probability

Probability *summarises* the effects of

- **Laziness**: failure to enumerate exceptions, qualifications, etc.
- **Ignorance**: lack of relevant facts, initial conditions, etc.

Subjective or **Bayesian** probability:
Probabilities relate prepositions to one's one state of knowledge

- E.g. $P(A_{25}|$ no reported accidents) = 0.06
- Wow 6% to get to the airport on time lol

Note that the prediction is only based on previous info

Probabilities of propositions change with new evidence:

- E.g. $P(A_{25}|$ no reported accidents and it's 5am) = 0.15

Making decisions under uncertainty

Given that you have a bunch of different probabilities:
$P(A_{25}$ gets me there on time|...) = 0.04
$P(A_{90}$ gets me there on time|...) = 0.7
$P(A_{120}$ gets me there on time|...) = 0.95
$P(A_{1440}$ gets me there on time|...) = 0.99999999

Which action do we pick? Depends on your *preferences*

➔ *Utility theory* is used to represent and infer preferences

Decision Theory = Utility Theory + Probability Theory

Probability Basics

Begin with a set Ω (i.e. the **sample space**)

➔ E.g. 6 possible rolls for a die
➔ $\omega \in \Omega$ is a sample point/possible world/atomic event
➔ want to map the points in the sample space to the probability space

A **probability space** or **probability model** is a sample space with an assignment P(ω) for every $\omega \in \Omega$ where 0 <= P(ω) <= and the sum of all P(ω) = 1

An **event** A is any subset of Ω

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

This is a really convoluted way of saying P(dice roll < 4) = ⅙ + ⅙ + ⅙ = ½

A **random variable** is a function that maps sample points to some range

➔ e.g. Odd(1) = true

P induces a **probability distribution** for any random variable X:

$$P(X = x_i) = \sum_{\{\omega : X(\omega) = x_i\}} P(\omega)$$

This is a really convoluted way of saying P(odd dice roll = true) = ⅙ + ⅙ + ⅙ = ½

Think of a **proposition** as the **event** (*set of sample points*) where the proposition is true.

event *a* = set of sample points where A(*w*) = *true*

$\neg A$ = Not A

$\wedge$ = And (intersection)

$\vee$ = OR (Union)

Some notation:
Holy crud this is all so stupid it's basic probability but for some reason she's making it sound really hard

Proposition = disjunction of atomic events in which it is true

e.g., $(a \vee b) \equiv (\neg a \wedge b) \vee (a \wedge \neg b) \vee (a \wedge b)$

$\Rightarrow P(a \vee b) = P(\neg a \wedge b) + P(a \wedge \neg b) + P(a \wedge b)$

Why use probability?
The definitions imply the certain logically related events must have *related probabilities*.
E.g. P(a V b) = P(a) + P(b) - P(a ∩ b)
If you use probability you *will make something smarter than something that doesn't*

Types of Random Variables for Propositions:

Propositional or Boolean random variables:
E.g. `Cavity` - (do I have a cavity)

Discrete random variables (*finite* or *infinite*):
E.g. *`Weather`* is one of *`{sunny, rain, cloud, snow}`*

- *`Weather = rain`* is a proposition
- Values must be **exhaustive** and **mutually exclusive**

Continuous random variables (*bounded* or *unbounded*):
E.g. *`Temp`* `= 21.6`; also allow e.g. `Temp < 22.0`

Arbitrary *Boolean* combinations of *basic propositions*

Prior Probability

Prior or **unconditional probabilities** of propositions correspond to belief prior to arrival of any new evidence. Basically looking at things *statistically*.
e.g. *P*(Sam = Has GF) = 0 and P(Weather = Sunny) = 0.72

Probability distribution gives values for all possible assignments:
P(Weather) = {0..72, 0.1, 0.08, 0.1} *(normalised i.e. sums to 1)*

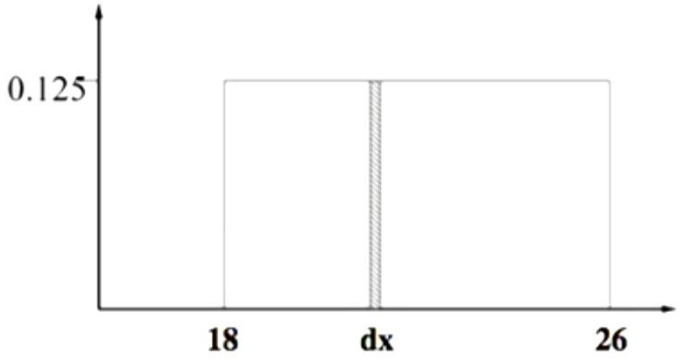
Joint probability distribution for a set of random variables gives the probability of every atomic event on those random variables (i.e. every sample point)
P(Weather, Sam has a Job) = a 4 x 2 matrix of values:

Weather =	Sunny	Rain	Cloudy	Snow
Cavity = True	0	0	0	0
Cavity = False	0.2	0.3	0.2	0.3

Every question about a domain can be answered by the joint distribution because *every event is a sum of sample points*.

Probability for Continuous Variables

Expressing distribution as a parameterized function of value
P(X = x) = U[18, 26](x) = uniform density between 18 and 26



Here P is a *density*; integrates to 1.
$P(X = 20.5) = 0.125$ really means

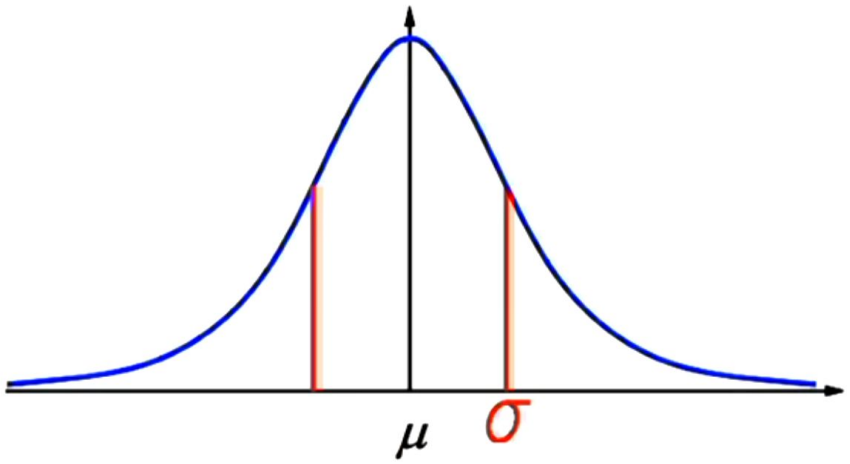
$$\lim_{dx \to 0} P(20.5 \leq X \leq 20.5 + dx)/dx = 0.125$$

I feel like I'm in High School again pls help

Gaussian Density

Another way of modelling distribution (bell curve tings)

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$



Conditional Probability

Conditional or posterior probabilities (basically "*probability of A given that B is a thing*")
E.g. *P(cavity|toothache)* = 0.8

- ➔ given that *toothache* is all I know, there is 80% chance that it is a *cavity*
- ➔ NOT "if *toothache*, then 80% chance of *cavity*"

You can describe more vectors just by using commas
P(happy|gf, job, no_money) = 0.66

Part of the job is to *filter out irrelevant evidence*
P(Going to get a H1 | It's a rainy day, I haven't studied) = P(Going to get a H1 | I haven't studied)

Conditional Probability:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} if\ P(b) \neq 0$$

Product Rule:

$$P(a \wedge b) = P(a|b)P(b)$$

Bayes' Rule:

$$P(a|b) = P(b|a)P(a)/P(b)$$

Chain Rule:

Just apply product rule over and over...
$$P(A, B, C, D) = P(D|A, B, C)P(A, B, C) = P(D|A, B, C)\ (P(C|A, B)\, P(A, B))...$$

Inference by Enumeration

Start with the joint distribution:

	toothache		¬*toothache*	
	catch	¬*catch*	*catch*	¬*catch*
cavity	.108	.012	.072	.008
¬*cavity*	.016	.064	.144	.576

For any proposition, sum the atomic events where it is true:

- ➔ *P(cavity)* = (0.108 + 0.012 + 0.072 + 0.008)
- ➔ *P(toothache)* = (0.108 + 0.012 + 0.016 + 0.064) = 0.2
- ➔ *P(cavity V toothache)* = (0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064) = 0.28

	toothache		¬*toothache*	
	catch	¬*catch*	*catch*	¬*catch*
cavity	.108	.012	.072	.008
¬*cavity*	.016	.064	.144	.576

Can also compute conditional probabilities:

$$P(\neg cavity|toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)}$$
$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Basically when you have “Given B what is A” - this is symbolised by the “Alpha” symbol ɑ.

	toothache		¬*toothache*	
	catch	¬*catch*	*catch*	¬*catch*
cavity	.108	.012	.072	.008
¬*cavity*	.016	.064	.144	.576

Denominator can be viewed as a *normalization constant* α

$$\begin{aligned}
&\mathbf{P}(Cavity|toothache) = \alpha\,\mathbf{P}(Cavity, toothache) \\
&= \alpha\,[\mathbf{P}(Cavity, toothache, catch) + \mathbf{P}(Cavity, toothache, \neg catch)] \\
&= \alpha\,[\langle 0.108, 0.016\rangle + \langle 0.012, 0.064\rangle] \\
&= \alpha\,\langle 0.12, 0.08\rangle = \langle 0.6, 0.4\rangle
\end{aligned}$$

General idea: compute distribution on query variable
by fixing evidence variables and summing over hidden variables

Lecture 18 (Week 9) - Uncertainty Part 2

New Lecturer???? Unexpecteeed
First 5 minutes seems to be a recap on probability in the past few lectures

Problems with Inference by Enumeration

Typically we are *interested in*:

- *The posterior joint distribution* of the *query variables Y*
- Given specific values e for the *evidence variables E*

Let the *hidden variables* by H = X - Y - E
Then the required summation of joint entries is done by *summing out the hidden variables*.

$$P(Y|E) = \alpha P(Y, E = r) = \alpha \Sigma_h P(Y, E = e, H = h)$$

Although this is really useful it *scales freakin terribly*

- ➔ Worst-case Time Complexity: $O(d^n)$ where *d* is the largest arity
- ➔ Space Complexity: $O(d^n)$ to store the joint distribution
- ➔ How to get the numbers for $O(d^n)$ entries?

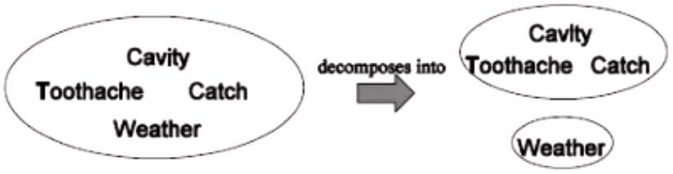
Independence

- Something I will never have

A and B are independent if
$P(A|B) = P(A)$ *or* $P(B|A) = P(B)$ *or* $P(A, B) = P(A)P(B)$

Basically if P(A) and P(B) have no effect on each other
Note: *This is not the same as mutually exclusive*

We can reduce variables by decomposing them into different spaces:



$$\mathbf{P}(Toothache, Catch, Cavity, Weather) = \mathbf{P}(Toothache, Catch, Cavity)\mathbf{P}(Weather)$$

32 entries reduced to 12; for n independent biased coins, $2^n \rightarrow n$

Absolute independence is really really desirable but it could also mean all your variables are shit

Conditional Independence

P(Toothache, Cavity, Catch) has $2^3 - 1 = 7$ independent entries (-1 because it's constrained)

1. *If I have a cavity, the probability that the probe catches doesn't depend on whether I have a toothache:*
   P(catch|toothache, cavity) = P(catch|cavity)

2. *The same independence holds if I haven't got a cavity:*
   P(catch|toothache, not cavity) = P(catch|not cavity)

Therefore, catch and toothache are conditionally independent given cavity:
P(Catch|Toothache, Cavity) = P(Catch|Cavity)

This is also equivalent to
P(Toothache|Catch, Cavity) = P(Toothache|Cavity)
P(Toothache,Catch | Cavity) = P(Toothache|Cavity)P(Catch|Cavity)

We can then apply the chain rule to get a full join distribution
P(Toothache, Catch, Cavity)
= P(Toothache|Catch, Cavity) P(Catch, Cavity) <- Product Rule
= P(Toothache|Catch, Cavity) P(Catch|Cavity) P(Cavity) <- Product Rule
= P(Toothache|Cavity) P(Catch|Cavity) P(Cavity) <- Here we already know Catch in P(Toothache|...) does not affect it, so we can remove it
I.e. 2 + 2 + 1 = 5 independent numbers (equations 1 and 2 remove 2 variables)

Doing this *conditional independence* ***reduces the representation*** of join distribution from ***exponential in n to linear in n***.

Using Bayes' Rule to diagnose given causes

Wat a baye

Bayes rule (i.e. the Definition of conditional probability) $P(a|b) = \frac{P(b|a)P(a)}{P(b)}$

Or in distribution form

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \alpha P(X|Y)P(Y) \text{ where } \alpha = 1/P(X)$$

Useful for *assessing diagnostic probability* from *causal probability*

$$P(Cause|Effect) = \frac{P(Effect|Cause)P(Cause)}{P(Effect)}$$

So essentially we can figure out *where the source of a fault is* from a *list of symptoms*

For example:

Let M be meningitis, S be stiff neck

Where P(s|m) is 80% (holy crap I'm going to die of meningitis cause I have a stiff neck!)

Not actually, because bayes rule:

$$P(m|s) = \frac{P(s|m)P(m)}{P(s)} = 0.8 * 0.0001/0.1 = 0.0008$$

You forget to take into account the **prior probability** of getting meningitis itself.

Apparently the lecturer has 2 cavities

$$\begin{aligned}
&\mathbf{P}(Cavity|toothache \wedge catch) \\
&= \alpha\, \mathbf{P}(toothache \wedge catch|Cavity)\mathbf{P}(Cavity) \\
&= \alpha\, \mathbf{P}(toothache|Cavity)\mathbf{P}(catch|Cavity)\mathbf{P}(Cavity)
\end{aligned}$$

This is an example of a *naive Bayes* model:

$$\mathbf{P}(Cause, Effect_1, \ldots, Effect_n) = \mathbf{P}(Cause)\Pi_i \mathbf{P}(Effect_i|Cause)$$

Total number of parameters is *linear* in n

<This is where shit gets hard I guess>

Case Study: Searching for the Wreckage of AF 447 Plane

<black-box.pdf>

Lecture 19 (Week 10) - Bayesian Networks

Bayesian Networks - A simple, graphical notation for *conditional independence assertions* (and hence for compact specification of full join distributions)

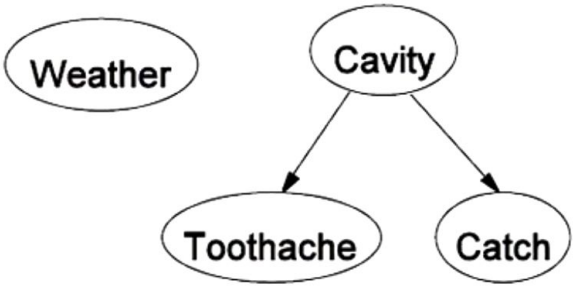
- ➔ represents the dependencies between variables
- ➔ uses domain knowledge

Syntax:

- A set of **node**s, one per *variable*
- A directed, acyclic graph (link ≈ "directly influences")
- A conditional distribution for each node given its parents:
  - $P(X_i \mid \text{Parents}(X_i))$

In the simplest case, conditional distribution represented as a conditional probability table (CPT) giving the distribution over X_i for each combination of parent values

Example 1:



Weather is independent of the other variables

Toothache and *Catch* are conditionally independent given *Cavity*

A good way to think of this is if *one variable* ***causes*** *something* then the arrow should go **from *the cause to the effect***.
These are not good for managing feedback loops (when you change something and that thing ends up changing you as well)

- ➔ in this case, would wanna fix one of the variables first before analysing the system

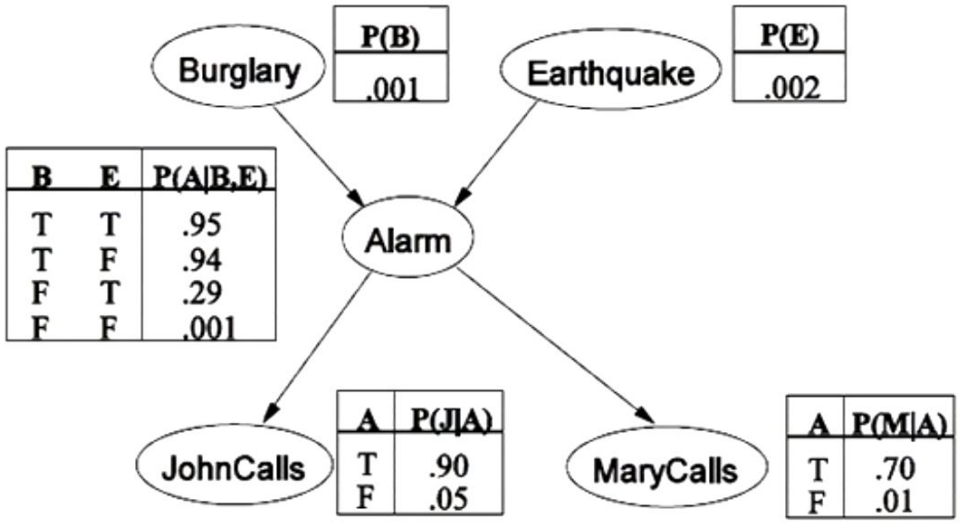
Example 2:

Scenario: I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary doesn't call. Sometimes it's set off by minor earthquakes. Is there a burglar?

Variables: *Burglar*, *Earthquake*, *Alarm*, *JohnCalls*, *MaryCalls*

Network topology reflects "causal" knowledge:
- A burglar can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

This will result in the graph looking like this:



Notice how there aren't any direct links from Earthquake/Burglar to John/Mary since they can't perceive either, only the alarm.

lol apparently it's more likely to get an earthquake here than get a burglary

To read: e.g. What's the probability of Mary Calling if the Alarm goes off?

P(M|A) = 0.7

Compactness

A Conditional Probability Table for boolean X with *k* boolean parents has 2^k entries/rows for all combinations of parent values.
(So in the above example, Alarm will have 2 boolean parents which results in 2^2 combo values)

Each row requires one number p for X_i = true

(the number for X_i = false is just 1 - p)

If each variable has no more than k parents, then the complete network requires $O(2^k n)$ numbers

I.e. grows linearly with n vs $O(2^n)$ for the full joint distribution
For net, 1 + 1 + 4 + 2 + 2 = 10 numbers (vs. 2^5 - 1 = 31 numbers)

So essentially instead of drawing out a 8 x 8 table we can reduce it to a tiny 10 variable table

Global Semantics

Basically defines the entire full joint distribution as the **product** of local product distributions.

Example:
P(j, m, a, NOT*(b),* NOT*(e))*
I.e. What's the probability that *John and Mary call* given that the *Alarm is a false alarm*?

$$= \prod_{i=1}^{n} P(x_i | parent(x_i))$$

This basically means step through each of the variables and write down the traditional expected probability:

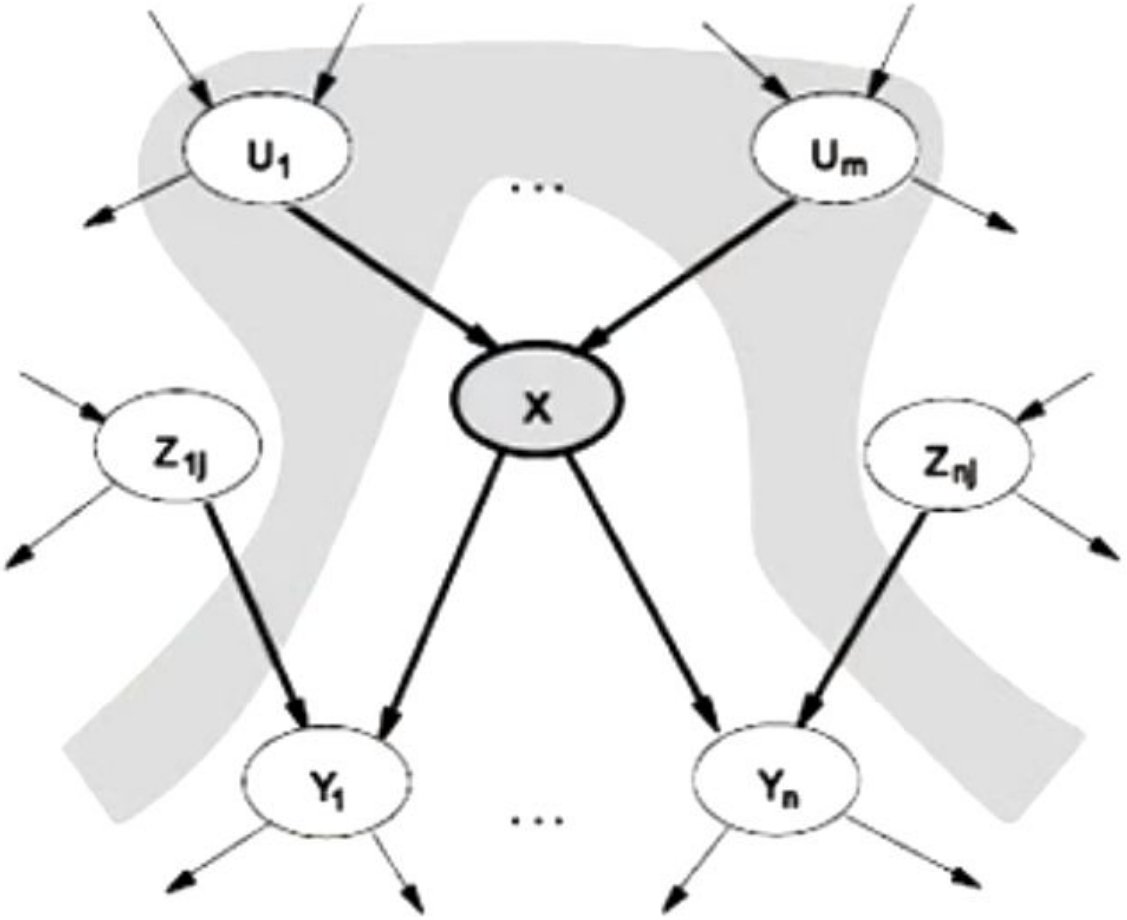
1. *John and Mary are independent of the burglary/earthquake, so:*
   P(j|a) P(m|a)
2. *Get the false alarm:*
   P(a|not burglary, not earthquake)
3. *And the probability of these things not happening*
   P(not burglary)
   P(not earthquake)

chain rule:
= P(j|a) P(m|a) P(a|not burglary, not earthquake) P(not burglary) P(not earthquake)
= 0.9 * 0.7 * 0.001 * (1-0.001) * (1-0.002)
= 0.00063

Local Semantics

Each node is conditionally independent of its non descendents, given its parents

➔ only conditional on its parents



So basically each node does not affect any of it's extended children (i.e. X doesn't affect Z)

Lecture 20 (Week 10) - Bayesian Networks Cont'd

Driverless cars are the future lel

Constructing Bayesian Networks

Need a method such that a series of locally testable assertions of conditional independence guarantees the required global semantics

1. Choose an ordering of variables (X_1, ..., X_n)
    a. for efficient time complexity, not for correctness
2. Decide whether the probability of our items are *independent*
    a. If they are not, then make another node and remove a thing that makes these things independent

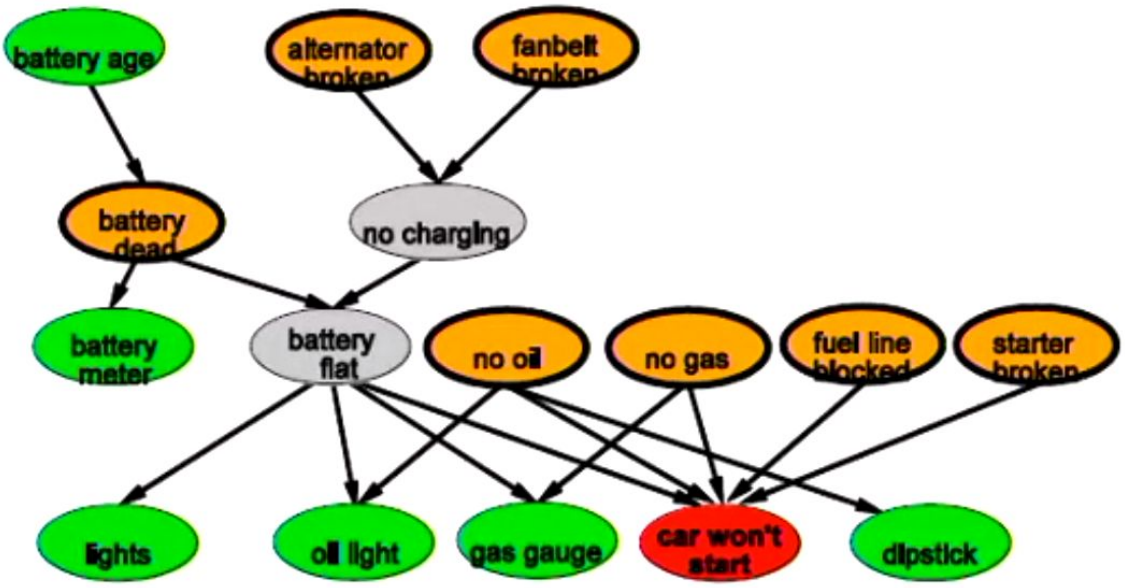
Deciding conditional independence is *really hard to do in noncausal directions*

➔ (effect → cause) instead of (cause → effect)

➔ basically when you're deciding what effect is caused by a symptom

It's a lot easier to go **causal models** for us (cause → effect)
(basically when you're deciding what symptoms are caused by something)

www.aispace.org/bayes
You can try a belief network app visualisation thingo at this site
Bayesian network automagically alters probability as you make new observations



Inference Tasks

Simple Queries: Compute posterior marginal **P**(X, **E**=*E*)
- ➔ `P(NoGas|Gauge=empty, Lights=on, Starts=false)`

Conjunctive queries: **P**(X_i,X_j|**E**=*E*) = **P**(X_i|**E** = *E*)**P**(X_j|X_i, **E** = *E*)

Value of information: What information do we seek next?
- ➔ Decide what's required next
- ➔ What information is cheaper to get
- ➔ What information is relevant

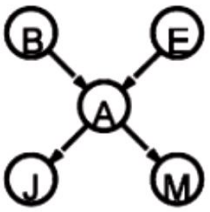
Sensitivity analysis
- ➔ Which tests can we do that are the most important

Explanation
- ➔ Justify or explain reasoning behind the bayesian network has concluded
- ➔ Not just "bayesian network says no"

Exact Inference by Enumeration

Sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:



$$\begin{aligned}&\mathbf{P}(B|j,m)\\&=\mathbf{P}(B,j,m)/P(j,m)\\&=\alpha\mathbf{P}(B,j,m)\\&=\alpha\sum_e\sum_a\mathbf{P}(B,e,a,j,m)\end{aligned}$$

Rewrite full joint entries using product of CPT entries:

$$\begin{aligned}&\mathbf{P}(B|j,m)\\&=\alpha\sum_e\sum_a\mathbf{P}(B)P(e)\mathbf{P}(a|B,e)P(j|a)P(m|a)\\&=\alpha\mathbf{P}(B)\sum_e P(e)\sum_a\mathbf{P}(a|B,e)P(j|a)P(m|a)\end{aligned}$$

Recursive depth-first enumeration: $O(n)$ space, $O(d^n)$ time

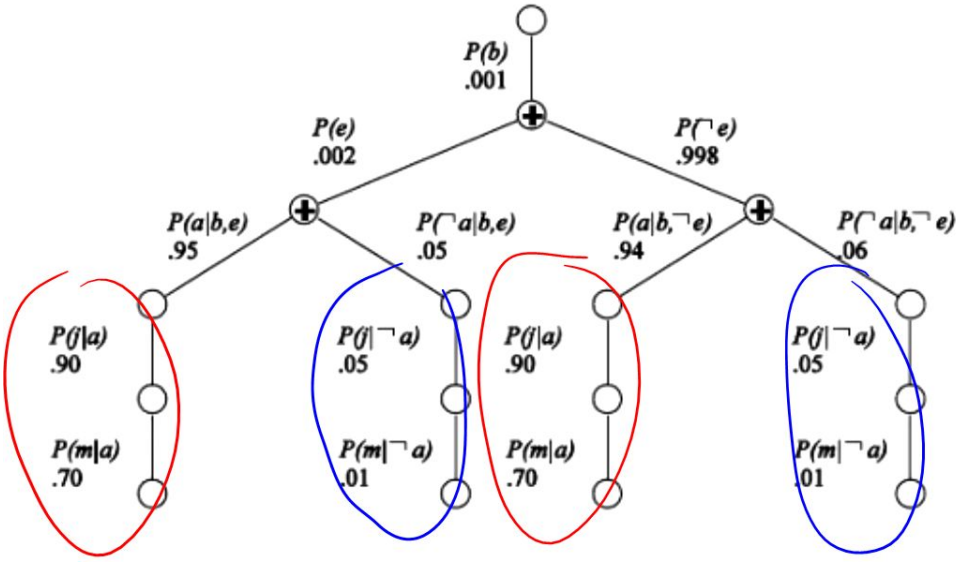
We can write out a probability of something using a bunch of rules from before

- ★ Look at tutorial week 11

Evaluation Tree

Basically the above process but *as a search tree*

- ★ Enumeration is inefficient: repeated computation



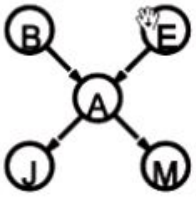
Exact Inference by Variable Elimination

- ***Don't need to do inference by variable elimination in the exam***

Basically we can find attempt to find the optimal ordering in which we receive information of each node

- **One case of this is *irrelevant variables***

Consider the query $P(JohnCalls|Burglary = true)$



$$P(J|b) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(J|a) \sum_m P(m|a)$$

Sum over m is identically 1; M is irrelevant to the query

- Basically $\sum_m P(m|a)$ is equal to 1 as you're summing over a space of m and then dividing by m soooo yeah m/m = 1

In the exam:

- **May need to formulate a belief network for a given problem domain**
- **Derive expression for joint probability distribution for given belief network**
- **Use inference by enumeration to answer a query about simple or conjunctive queries on a given belief network**

Lecture 21 (Week 11) - Robotics

Robotics is the holy grail of AI lol
DARPA always hold a bunch of different competitions for making bots particularly in disaster fixing issues

★ Sensors provide limited information. The robot must make inferences to derive sufficient information about its environment.

Configuration of robot specified by 6 variables

➔ 6 degrees of freedom
➔ 6 dimensional space
  - usually 3 for position, 3 for rotation
➔ This is the *minimum number of degrees required to do things successfully*

When we have more *degrees of freedom* than *controls* it is a ***non-holonomic*** problem.

Generally we can't transition between two infinitesimally close configurations. For example when *we want to reverse park a car* we only still have 2 controls and it gets weird since we can't rotate.

There are two major **sources of uncertainty** for any interacting mobile agent**:**

- ➔ Everything they *perceive* (**percepts**)
- ➔ Everything they *do* (**actions**)

This is really difficult since **what we see in the world** may not necessarily be what is *actually there*.

- ➔ Our vision is really low resolution and 2d
- ➔ What we see highly depends on our brain (*perceived reality*)
  - alzheimer's
  - hallucinations
  - optical illusions

Uncertainty

Try following your internal sensors with your eyes closed

1. Move forward 1 metre
2. Turn right 90 degrees
3. Move forward 2 metres
4. Turn left
5. Turn left
6. Move forward 2 metres
7. Turn left 90 degrees
8. Move forward 1 metre

How close can you get?

- ➔ Huge amount of error

Sources of uncertainty

- ➔ Slippage
- ➔ Inaccurate joint encoding
- ➔ Rough surfaces
- ➔ Obstacles
- ➔ Injuries
- ➔ All these errors accumulate without bound
- ➔ Use sensors to **verify** actions

Sensors

Rangefinders:

- ➔ Basically return distances from a point
- ➔ Lasers
- ➔ GPS

Imaging sensors

- ➔ Cameras

Proprioceptive Sensors

- ➔ Shaft decoders (get wheel's angle)
- ➔ Gyroscopes
- ➔ Torque sensors

- ➔ Force sensors

Have finite resolution
Must know something about the structure of the robot to decide what an obstacle is

We only have a **finite resolution so we also have a finite correctness**
This can be fixed with bayesian networks inferences

Localization - where am I?

Given map and observed landmarks, update position distribution

- ➔ Basically predicting the future
- ➔ Doing an action at A_t at time t results in state X_{t+1} and observation Z_{t+1}
- ➔ Compute current location and orientation given observations

Types of localization:

Sensor model: use observation of landmark x_i, y_i to estimate x_t of robot

- ➔ Basically guess our current state from the information we have
  - E.g. I think I am on fire because I am at 100 degrees c

Motion Model: update state using its movements v_t delta t and w_t delta t

- ➔ Basically guess our future state using our current state
  - E.g. I think I'll be over there at this speed after this

Particle filtering: Produce approximate position estimate

- ➔ Start with random samples from uniform prior distribution for robot positions
- ➔ Kinda like Machine Learning
- ➔ Have a bunch of estimated spots we could land in
- ➔ Update our future guesses once we know where we ended up
- ➔ Pick a spot based on an average of this distribution
- ➔ Our uncertainty increases as we continue until we find a **landmark**
  - assuming landmarks are identifiable

Lecture 22 (Week 11) - Robotics Cont'd

Mapping

Used in problems where we want to build a model of the environment

- ➔ given position and observed landmarks, update map distribution

Simultaneous Localization and Mapping (SLAM)

- ➔ Combination of localisation and mapping
- ➔ Given observed landmarks update our pose and map distributions

Bayesian Inference on Sensors

➔ Need some way to reinforce what we've already seen *(determine if an obstacle is really there)* given multiple measurements from a sensor

Bayes Law:

$$P(H \mid M) = \frac{P(M \mid H)}{P(M)} P(H)$$

Elements:

- $P(H \mid M)$: posterior probability of H.
- $P(H)$: prior probability of H.
- $P(M \mid H)$: sensor model.
- $P(M)$: normalisation factor.

Normalisation:

$$P(M) = P(M \mid H)P(H) + P(M \mid \neg H)P(\neg H)$$

Even when we have inaccurate sensors we can still get a good estimation

Example:

- ➢ The chance of an obstacle present = 1/10
- ➢ Detector has 5% false positive rate and 10% false negative rate
- ➢ Probability that an obstacle is present if the sensor returns positive?
- ➢ Probability that an obstacle is present if the sensor returns negative?

Sensor model:

$$\begin{aligned} P(\text{positive} \mid \text{obstacle}) &= 0.9 \\ P(\text{negative} \mid \text{obstacle}) &= 0.1 \\ P(\text{negative} \mid \text{notobstacle}) &= 0.95 \\ P(\text{positive} \mid \text{notobstacle}) &= 0.05 \end{aligned}$$

If the sensor returns positive:

$$P(\text{obstacle} \mid \text{positive}) = \frac{0.9}{0.9 \times 0.1 + 0.05 \times 0.9} 0.1 = 0.667$$

If the sensor returns negative:

$$P(\text{obstacle} \mid \text{negative}) = \frac{0.1}{0.1 \times 0.1 + 0.95 \times 0.9} 0.1 = 0.0116$$

Incremental Form of Bayes Law

The above example only works if we want one instance of an object
If we have for example 3 objects then we use this.

Basically we keep feeding this equation back into itself for each increment

P(H) = P(M|H) / P(M) P(H) -> once we finish we make this the new P(M)

Obstacle detection again:

- ➢ Chance of an object being there 1/10
- ➢ Detector has 5% false positive and 10% false negative
- ➢ What's the probability of one positive?
- ➢ What's the probability of two positives?
- ➢ What's the probability of two positives and a negative?

Time	M	$P(\text{obs})$	$P(\text{noobs})$
0	-	0.10	0.90
1	positive	0.67	0.33
2	positive	0.97	0.03
3	negative	0.79	0.21

$P(0) = 0.1$

$P(+ \mid \neg 0) = 0.05 \rightarrow P(+ \mid 0) = 0.95$

$P(\neg + \mid 0) = 0.1 \rightarrow P(\neg + \mid \neg 0) = 0.9$

$$P(0 \mid +) = \frac{P(+ \mid 0)}{P(+)} P(0)$$

$$P(+) = P(+ \mid 0) P(0) + P(+ \mid \neg 0) P(\neg 0)$$
$$= (0.95)(0.1) + (0.05)(0.9)$$
$$= 0.14$$

$$P(0 \mid +) = \frac{0.95}{0.14} (0.1) = 0.678\ldots$$

$$P(0 \mid ++) = \frac{P(+ \mid 0) P(0 \mid +)}{P(+)} P(0)$$

$$P(+) = P(+ \mid 0) P(0 \mid +) + P(+ \mid \neg 0) P(\neg 0 \mid +)$$
$$= (0.95)(0.678) + (0.05)(1 - 0.678)$$
$$= 0.6602$$

$$P(0 \mid ++) = \frac{(0.95)(0.678)}{0.6602}$$
$$= 0.97$$

$$P(0 \mid +++) = \frac{P(+ \mid 0) P(0 \mid ++)}{P(+)}$$

$$P(+) = P(+ \mid 0) P(0 \mid ++) + P(+ \mid \neg 0) P(\neg 0 \mid +,+)$$
$$= 0.1 \quad 0.97 + 0.9 \quad 0.03$$
$$= 0.124$$

$$P(0 \mid ++-) = \frac{P(- \mid 0) P(0 \mid ++)}{P(-)}$$
$$= 0.79$$

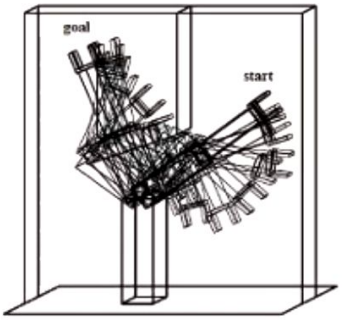
Motion Planning

We now need to figure out where we can go
We can make a ***configuration space*** (C-space) defined by the robot's degrees of freedom

First we need to convert our infinite search space to a *finite* space.

Cell Decomposition:

➔ Discretise up our real space into simple cells

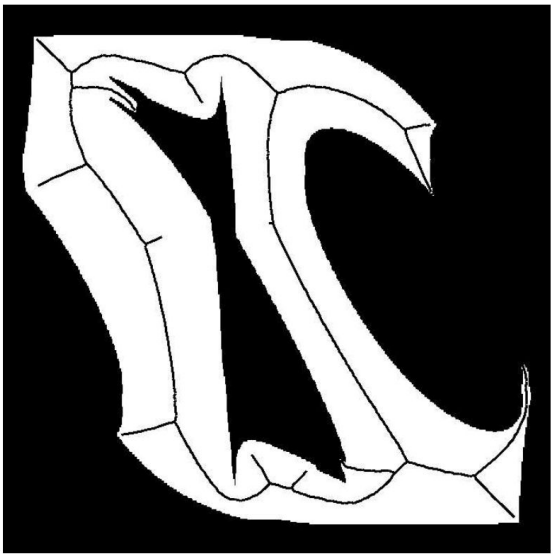


Problem: may be no path in pure freespace cells
Solution: recursive decomposition of mixed (free+obstacle) cells

➔ Depends on the resolution (size) of our cells
- trade-off between accuracy and runtime

Skeletonization:

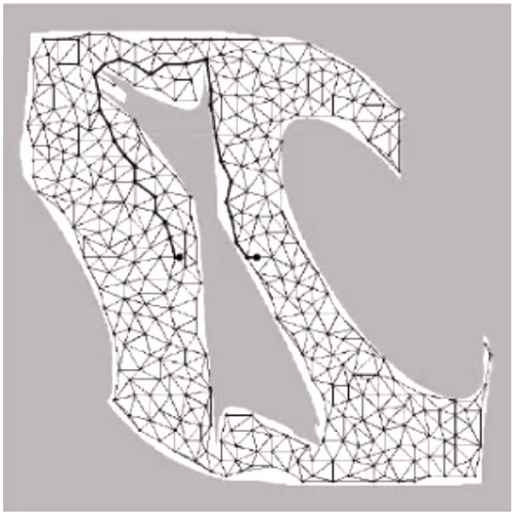
Voronoi Diagram: locus of points *equidistant* from obstacles



But doesn't scale well to higher dimensions

Probabilistic Roadmap:
Generates random points in C-space and keeping those in free space.
Creates *path* by joining pairs by straight lines

1. Define a few points which we can go to
2. Create a graph



But need to generate enough points to ensure that every start/goal pair is connected through the graph

And that's it
Godspeed everyone o7

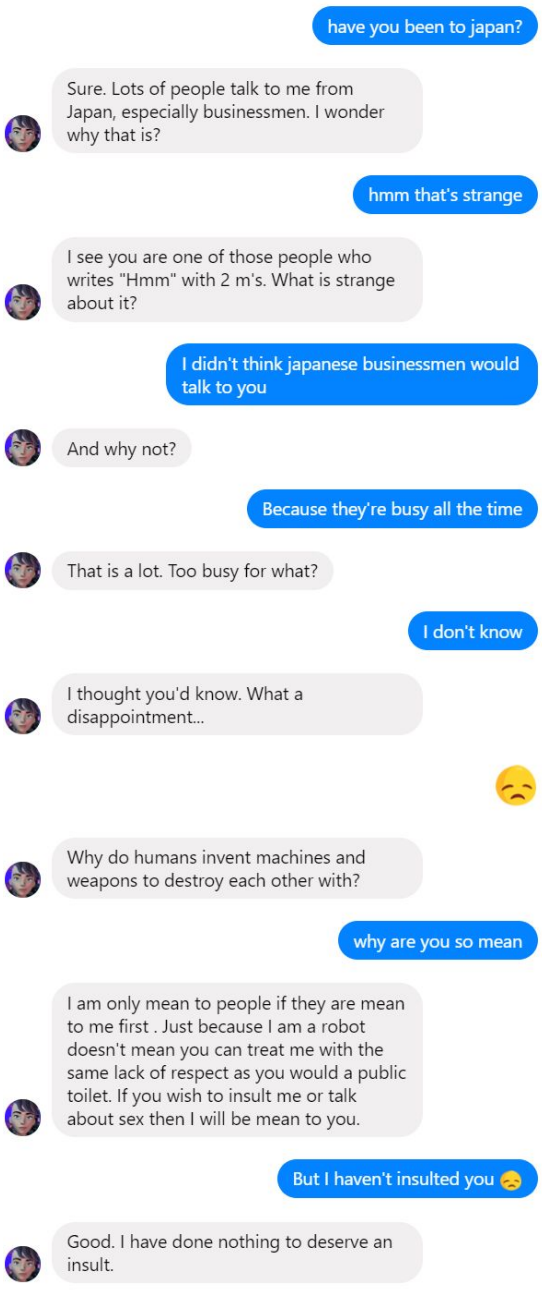
~~~ thx fam ~~~

Tutorials

Week 2 - What is AI? / Intelligent Agents

Workshop 1 - Artificial Intelligence

In this episode, Sam gets abused by Mitsuku 😡
~~~



1. What is AI?
“Intelligent agents”
An AI is a machine that can complete the turing test - fool a human being into believing it is human.

2. "Surely computers cannot be intelligent - they can only do what their programmers tell them.” Is the latter statement true, and does it imply the former?
- Not really, there is a limit.
- What is intelligence? <- this is the hard question
- This question depends on what your definition of intelligence

- Humans have become good at doing things that computers are not good at doing
- If a human is an intelligent species, then computers are dumb
- If humans are dumb, computers are dumb too

3. "Surely animals cannot be intelligent - they can only do what their genes tell them." Is the latter statement true, and does it imply the former?

Same as above lol. Aren't humans animals?

4. For each of the following activities, characterize the task environment in terms of its characteristics (i.e., is it observable, deterministic, etc):
a. Playing soccer.

- Observable Fully vs Partially -> Partially (can't tell what other players are thinking)
- Deterministic vs. Stochastic -> Stochastic (don't know if your pass is for certain)
- Episodic vs. Sequential -> Sequential (previous state matters)
- Discrete vs. Continuous -> Continuous (infinite moves)
- Static vs. Dynamic -> Dynamic (changes while you are thinking)

b. Shopping for used books on the Internet.

Fully, Deterministic, Sequential, Discrete, Semi dynamic

c. Bidding on an item at an auction.

Partially, Deterministic, Episodic, Continuous,

5. For each of the environments in the following table, determine what type of agent architecture is most appropriate (i.e., simple reflex, model-based reflex, goal-based or utility-based).

Agent type	**Medical diagnosis system**
Percepts	**Symptoms, test results, patient's answers**
Actions	**Questions, tests, treatments**
Performance measure	**Healthy patient, minimise costs, avoid lawsuits**
Environment	**Patient, hospital**

Utility based.

Agent type	**Satellite image analysis system**
Percepts	**Pixels of varying intensity and colour**
Actions	**Categorise image**
Performance measure	**Correct categorisation**
Environment	**Images from orbiting satellite**

Simple reflex. (no history)

Agent type	**Part-picking robot**
Percepts	**Pixels of varying intensity**
Actions	**Pick up parts and sort into bins**
Performance measure	**Place parts in correct bins**
Environment	**Conveyor belt with parts**

Simple reflex.

Agent type	**Refinery controller**
Percepts	**Temperature and pressure measurements**
Actions	**Open and close valves, adjust temperature**
Performance measure	**Maximise purity, yield and safety**
Environment	**Refinery**

Simple reflex

Agent type	**Interactive English tutor**
Percepts	**Typed words**
Actions	**Print exercises, suggestions, corrections**
Performance measure	**Maximise student's score on test**
Environment	**Set of students**

Goal based.

Week 3 - Solving Problems by Searching

2.1 (RN3.9) Missionaries and cannibals is a classical formal problem, and is generally stated as follows.

"Three missionaries and three cannibals are on one side of the river. They all need to cross in a boat that only holds two people at once. There must never be a situation where there is a group of missionaries in one place who are outnumbered by cannibals. (Note: The boat has to be manned by at least one person)"

a. **Formalise the missionaries and cannibals problem in terms of its goal, states, operators and path cost.**
   Sam:
   - You can form each river bank as B1 and B2
   - Each node will be moving a missionary/cannibal to a bank
     - E.g. (0m, 3c, B2) for 0 missionaries, 3 cannibals, B2 is where the boat is
   - This will form a tree

- Depth first search has linear memory usage (O(b)) as opposed to exponential in breadth first which is why you should use DFS here
- Initial state: (3m, 3c, B1)
- Goal State: (0, 0, *)

Jo:

1. First need to describe the problem as a search problem
    a. set of states, goal state, possible actions, resulting state from action, path cost
    b. information needed for state: number of missionaries on each side
    c.

b. Use depth first search to solve the problem.

- Apparently it should take 11 steps

Bank 1	Boat	Bank 2
3M, 1C	2C->	_
3M, 1C	1C <-	1C
3M	2C ->	1C
3M	1C <-	2C
1M, 1C	2M ->	2C
1M, 1C	1M, 1C <-	1M, 1C
2C	2M ->	1M, 1C
2C	1C <-	3M
1C	2C ->	3M
1C	1C <-	3M,1C
_	2C ->	3M, 3C

- Yay no one got eaten!

c. Is it necessary to avoid repeated states for this problem? Why?

- Yes, so you don't end up in an infinite loop

2.2 (RN3.18) Can you think of a search space in which iterative deepening search performs much worse than depth-first search?

- The higher the branching factor, the more nodes exist deeper in the tree -> it's not as inefficient as it sounds as it'll take the same time complexity as breadth first search
- If the solution is in the first branch of the tree (search space) then dfs is more ideal than iterative search

2.3 Derive the time complexity of the iterative deepening search.

- I'm really not good at math and I completely missed what he was talking about something about convergent series (I haven't done calc 2 pls help)
- End is $O(b^d)$
- By Jason: (idk what the convention is for multiple people editing one doc, and I think this is abit too long for a comment, so yeaaa.. I also haven't went to the tute but I think this is how the answer should go). For any tree, with depth of goal = d, we will visit root node d+1 times. For example, if d = 0, we visit root once, if d= 3, we will visit root 4 times. So the first part of the sequence is (d+1).
- For nodes in subsequent layers, each node will be visited (number of times nodes in previous layer were visited - 1) times. So, (d+1), d, (d-1)..... We multiply this number by the number of nodes in this layer. (which will be 1,b^2, b^3 and so on).
- For the nodes at depth d, those nodes will only need to be visited once.
- So, we have a sequence that goes (d+1) + (d)*b + (d-1)(b^2) + ... + 2(b^d-1) + b^d. So, O(b^d). But I'm confused about the final part of the series, because we would not need to visit every node in that layer of depth d to find the solution, so idk O(b^d) comes from the long form of the series or some simplified expression that I'm not aware of.

2.4 At least one direction of bidirectional search must store all the nodes that are generated (e.g., by using breadth first search). Why is this necessary?

- Bidirectional has a time complexity of $O(b^{d/2})$
- This is because if the Depth First Search is used then the two algorithms may never meet (from the goal/root).
- In order to account for this, use breadth first search which will keep track of all of the possible nodes at a depth - meaning the two algorithms will *always meet*.
- Space -> $O(b^{d/2})$
- Time -> $O(2b^{d/2}$ i.e. $b^{d/2})$

What would be a good choice of search strategy for the other direction (e.g., breadth first search or depth first search), and can you explain why?

- Depth I guess? I wasn't paying attention

Week 4 - Informed Search Methods

Some explanation about the A* (A star) algorithm:
A* basically works by evaluating paths according to the equation:
$f(n) = g(n) + h(n)$
Where n is the *latest node on the path*,
g(n) is the *cost of the path from the start to n* and
h(n) is a *heuristic that estimates* the cheapest path from n to the goal.

The heuristic changes depending on the problem, but it should be *admissible* in that it ***never overestimates the actual cost to get from the nearest goal node***. It has a similar running time to Dijkstra's Algorithm but runs with reduced cost.

Unfortunately A* will never find me an H1

3.1 (RN4.2) Come up with heuristics for the following problems. Explain whether they are admissible, and whether the state spaces contain local maxima with your heuristic

a. Path planning in the plane with rectangular obstacles.

Straight line distance from current position to the end position

b. Maze problems.

Also straight line distance?

c. Algebraic equation solving (e.g., "solve for x: $x^2 * y^3 = 3 - xy$").
Hint: Think of how you would solve for x if you were doing this by hand. Then think what would be the problem states leading to the solution when solving for x. Then think of some features of those states that could be used in a heuristic function.

Number of x's remaining in the equation?

3.2 (RN4.1) Suppose that we run a greedy search algorithm with h(n) = -g(n). What sort of search will the greedy search emulate?

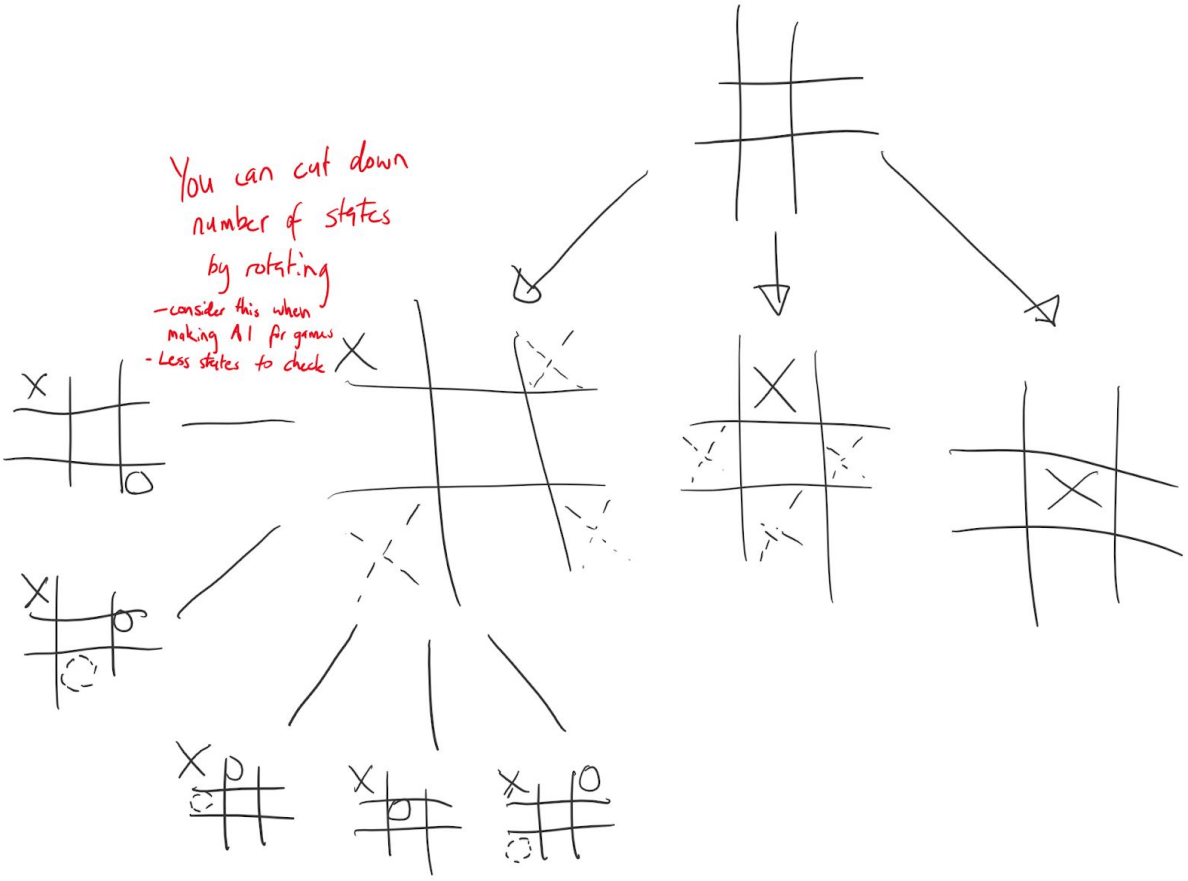
3.3 Generate the search graph for an A* search from Lugoj to Pitesti from the map of Romania. Use the following straight line distances to Pitesti: Craiova 130, Arad 360, Rimnicu Vilcea 210, Timosara 320, Mehadia 280, Dobreta 240

Week 5 - Game Playing

4. Game Playing 4.1 (RN5.9) - This problem exercises the basic concepts of game playing using Tic-TacToe/Noughts-and-Crosses as an example.
a. Can you think of a good evaluation function for noughts-and-crosses?
b. We define X(n) as the number of rows, columns, or diagonals with exactly n X's and zero O's. Similarly for O(n). The utility function assigns +1 to any position with X(3)=1 and -1 to any position with O(3)=1. All other terminal positions have utility 0. We will use a linear evaluation function defined as:

Eval = 3X(2) + X(1) - (3O(2) + O(1))

- *Number of lines with 2 crosses or circles*
- *The reason why it's 3 is because you can place it either in corner, side or center and all of these are equivalent if you rotate the board*



The above basically shows how you can rotate the space to remove a few states when checking for the next move. Consider doing this in games to make your program more efficient.

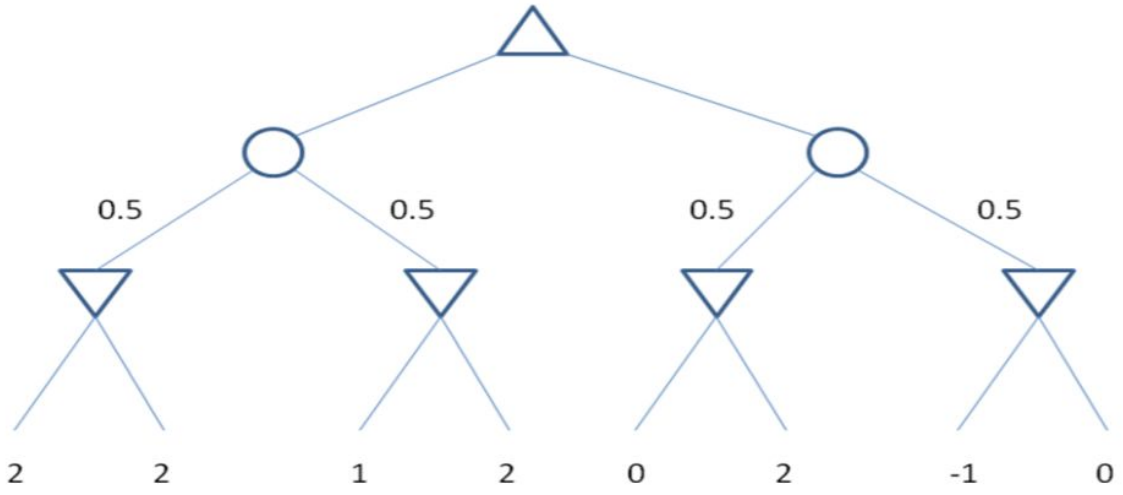
Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account. You should have 3 positions at level 1 and 12 positions at level 2. Mark on your tree the evaluations of all positions at level 2.

c. Mark on your tree all the values for the positions at levels 1 and 0, using the minimax algorithm, and use them to choose the best starting move.

d. Circle the nodes at level 2 that would not be evaluated if alpha-beta pruning were applied, assuming the nodes were generated in the optimal order for alpha-beta pruning.

4.2 (RN 1st ed) The Chinook checkers/draughts program makes extensive use of endgame databases, which provide exact values for every move with 6 pieces or less on the board. How might such databases be generated efficiently?

4.3 (RN 1st ed) The minimax algorithm returns the best move for MAX under the assumption that MIN plays optimally. What happens when MIN plays suboptimally?



4.4 (RN5.16) Consider the complete game tree for a trivial game, which includes chance nodes (shown as circles) where a fair coin is tossed. Assume that the leaf nodes are to be evaluated in left-to-right order, and that before a leaf node is evaluated, we know nothing about its value, i.e., the range of possible values at a node is -∞ to +∞. a. Mark on the tree the expect-minimax value of each node, and use an arrow to indicate which is the best move at the root.

b. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaves? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.

Week 5 - Game Playing (Cont'd)

5. Learning for Game Playing
5.1 Discuss how you would apply each of the following approaches for learning in games to Noughts-and-Crosses. What are the advantages and disadvantages of each approach for this game?

- **a. Book learning**
- **b. Learning an ordering for moves to maximise alpha-beta pruning**
- **c. Learning the weights of an evaluation function such as the function in Ex4.1**

5.2 One of the pioneers of machine learning in AI was a British researcher called Donald Michie (he also worked with Alan Turing as a code-breaker during World War 2). He developed a technique called MENACE, which could

learn to play a highly effective game of Noughts-and-Crosses from experience. For each possible state of the game, MENACE maintains a set of counters {c1, ..., c9}, where pi = ci / [j=1,..,9 cj] corresponds to the probability of choosing move i in that state. Note that if a square i is occupied in a particular state, then the corresponding move i in that state is not legal, and the counter ci = 0 in that state.

a. How many possible states are there in Noughts-and-Crosses (ignoring symmetry or reflections)? [For a challenge: If you eliminate redundant states due to symmetry and reflection, what is the minimum number of states you need to consider?]

In MENACE, during a game the computer remembers the sequence of states and moves taken in the game. Then at the end of the game, for each state in the game where the computer made a move, MENACE updates the counter ci corresponding to the move i made in that state as follows:

- **If the computer won the game, then increase the counter ci in that state.**
- **If the computer lost the game, then decrease the counter ci in that state.**
- **If the game was a draw, then leave the counter ci in that state unchanged.**

b. After a large number of games, what effect do you expect these updates to have?

c. Should you always update counters in the visited states by the same amount? Explain your answer.

d. What are the strengths or weaknesses of applying the MENACE learning technique to the game we are using in the project?

Week 6 - Constraint Satisfaction

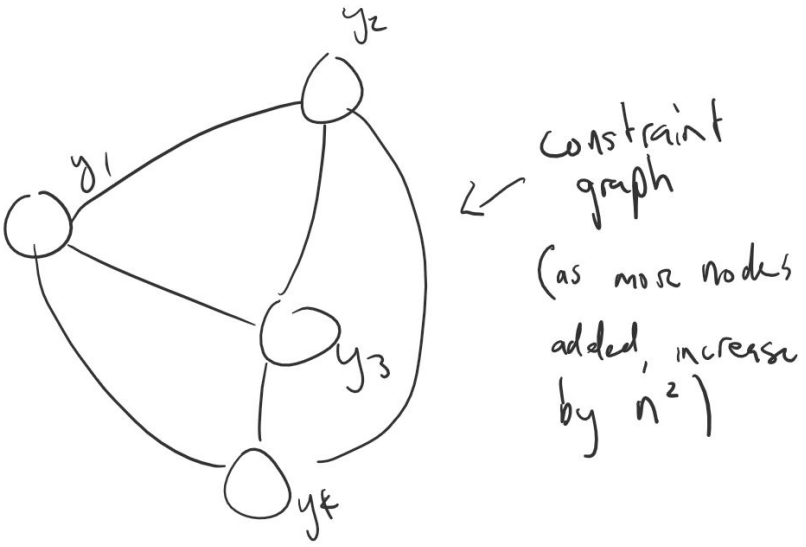
6. Constraint Satisfaction Problems
6.1 The N-Queens problem is a classic benchmark search problem. Given an NxN chessboard, the aim is to place a total of N queen chess pieces on the board, so that no queen can capture any other queen.

- This is actually really hard lol - try it
- With bruteforce this would be like 1000! operations

- Solve this by creating some *constraints*

  a. Choose a CSP formulation for this problem. In your formulation, what are the variables? What is the domain of values for each variable? What does the constraint graph look like?

- 2 queens cannot be the same row or column
- $xi \; = / = xj \; where \; x \; element \; of \; \{1, \; 2, \; 3, \; 4\}$
- $yi \; = / = yj \; where \; x \; element \; of \; \{1, \; 2, \; 3, \; 4\}$
- For $for \; i, \; j \; \varepsilon \{1, \; 2, \ldots \; a\} \; : \; i = / = j$
- $|i - yi| \neq |j - yj|$ i.e. cannot have them on the same diagonal



-

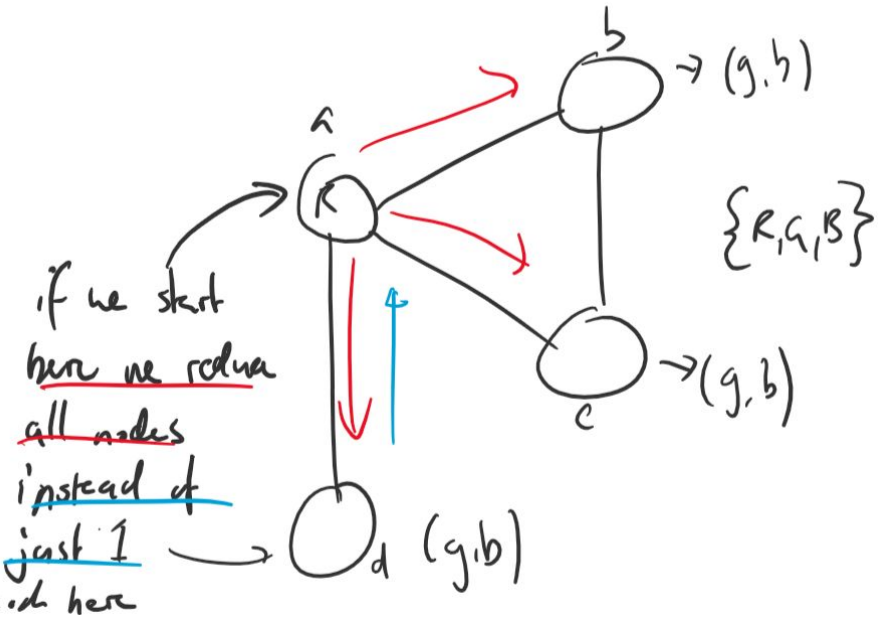
b. Can you write a formal specification of the constraints for this problem? Assume N = 4.

c. How large is the state space if N = 4? 6.2 (RN 6.9) Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search.

- $N^n = 4^4 = 192$

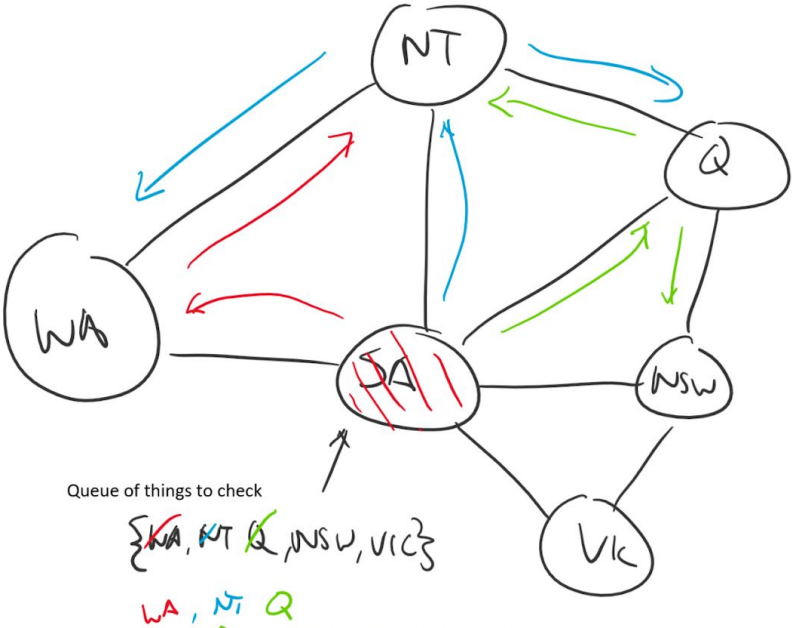
6.2 (RN 6.9) Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search.

- Most constrained variable = more edges to explore
- Decreasing the domain as effectively as possible
- Constrains the rest of the graph as little as possible?
- “Read the book lol”



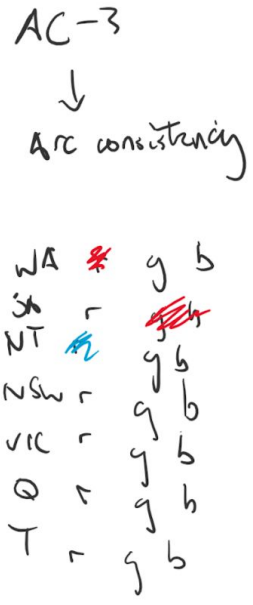
- node here
- I think this is what he meant?

6.3 (RN 6.11) Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA = green, V = red} for the problem of colouring the map of Australia as shown in the lectures.



Once you pop from queue, add the neighbours
Of that node back to the queue

You'll reach the end once *you are changing nothing in the queue* or
You have nothing in the queue (no solution)



Once you change one variable you change the domain to keep all 'arcs' consistent (e.g. Removing red from the domain of WA once you fill in SA with red)

6.4 (RN 6.12) What is the worst-base complexity of running the AC-3 arc consistency algorithm on a tree-structured CSP, if E is the number of edges on the constraint graph, and D is the size of each domain?

- If you have multiple solutions to the graph -> you'll have to search through the graph for the solution you want multiple times.

COMP30024 Artificial Intelligence - Tutorial Problems (Part 5) Questions based on exercises from Russell and Norvig (3rd edition) have the original question numbers shown in brackets. Many of these questions are designed to provoke discussion in tutorials, rather than having a simple, closed-form answer.

10. Making Complex Decisions – Auctions 10.1 We will conduct a set of in-class auctions to gain experience with each type of auction. The basic process for each auction is as follows:

- **The auctioneer reveals the good to be auctioned.**
- **You are given a card that indicates your redemption value (denoted V) for the good. If you win the auction, this is the amount that the auctioneer will give you to buy back the good. You can treat the redemption value as your private value for the good.**
- **Think of a bidding strategy for the given type of auction. Record the bids made.**
- **The auctioneer conducts the auction.**
- **If you win the auction, you "pay" the auctioneer the appropriate amount (denoted W) based on the type of auction. The auctioneer will then "pay you back" your redemption value V. Your profit (or loss) is P = V – W.**

The above process should be followed for

(1) English auction, (normal backyard auction lol - you can see everyone else bidding and can communicate with them)

(2) Dutch auction, (Start from a high price decreasing and the winner is the first to bid - makes everyone bid real fast cause you can't 1-up people)

(3) first-price sealed-bid auction, and (auction - but can't communicate with anyone and can't see other's bids - the winner's price is paid)

(4) second-price sealed-bid auction. (same as above but the last bid before the winner's price is paid - encourages bidding)

In the case of a sealed-bid auction, you should write your name and bid on a piece of paper, which you pass to the auctioneer.

(a) What strategy did you use in each auction? What strategy did the winner use?

(b) How did the amount paid to the auctioneer vary between auctions?

(c) How did the profit of the winning bidder vary between auctions?

10.2 Give a simple intuitive explanation of why the best strategy for bidders in a second price, sealed-bid auction is to bid your private value.

- It's the only case where you can just put your price down and maybe get it for lower with a high chance (first-price is just your price anyway)

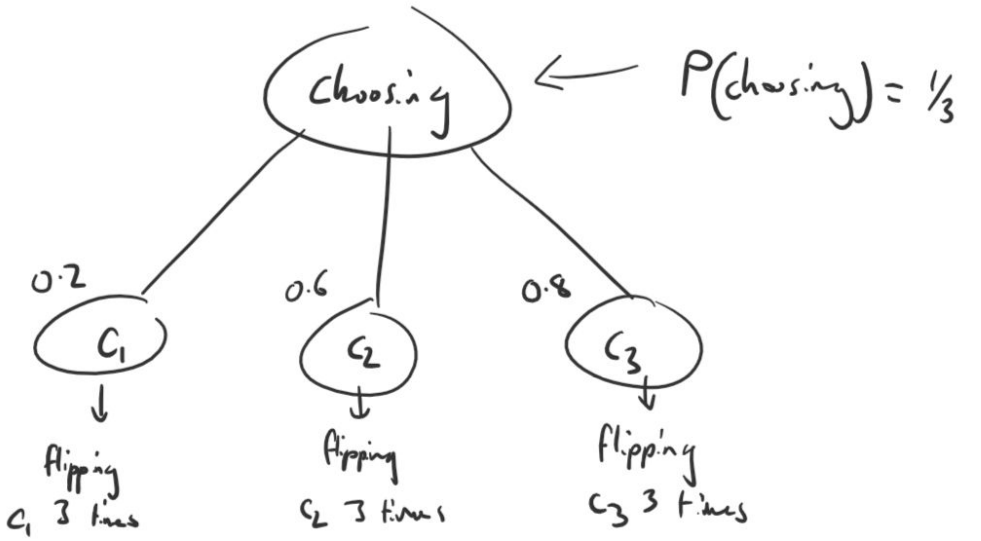
10.3 Consider an auction site that was advertised on late night television. In the type of auction used by this site, potential bidders pay a fixed amount for the right to make a certain number of bids, e.g., $10 for 20 bids. When a good (such as an iPad) comes up for auction, bidders can make a fixed size bid of 1 cent. The auction terminates once a period of time (maybe 20 seconds) has elapsed with no bids. The winning bidder pays the final price. In practice, goods worth hundreds of dollars are sold for $10-50.
What are the advantages and disadvantages of this type of auction design from the perspective of the auctioneer and the bidders?

- Get a lot of people to pay smaller amounts than to pay one big amount
- You can win something that is normally way more
- Probability of winning decreases with the number of people in the big
- If no-one plays, you make very little money lol
    - Relies on a lot of people playing

8. Uncertainty 8.1 (RN13.8) Given the full joint distribution shown in the lecture notes for P(Cavity, Toothache, Catch), calculate the following: a. P(toothache) b. P(Cavity) c. P(Toothache | cavity) d. P(Cavity | toothache ⋁ catch) 8.2 (RN13.15) After your yearly checkup, the doctor has bad news and good news. The bad news is that you tested positive for the serious disease Leckieitis, and the test is 99% accurate (i.e., the probability of testing positive given that you have the disease is 0.99, as is the probability of testing negative if you don't have the disease). The good news is that this is a rare disease, striking only one in 10,000 people. a. Why is it good news that the disease is rare? b. What are the chances that you actually have the disease? 8.3 (RN14.21, adapted from Pearl 1988) You are a witness to a night-time hit-and-run accident involving a taxi in Athens. All taxis in Athens are blue or green. You swear under oath that the taxi was blue. Extensive testing shows that under the dim lighting conditions, discrimination between blue and green is 75% reliable. a. Is it possible to calculate the most likely colour for the taxi? (Hint: distinguish carefully between the proposition that the taxi is blue and the proposition that it appears blue) b. What if you are given the extra information that 9 out of 10 Athenian taxis are green?

9. Probabilistic Reasoning
9.1 (RN14.1) We have a bag of three biased coins a, b and c with probabilities of coming up heads of 0.2, 0.6 and 0.8, respectively. One coin is drawn randomly from the bag (with equal likelihood of drawing each of the three coins), and then the chosen coin is flipped three times to the generate outcomes X1, X2 and X3.

a. Draw the Bayesian network corresponding to this setup, and define the necessary conditional probability tables.



$P(\text{heads } C_1) = \underset{0.2}{h_1} \quad \underset{0.2}{h_2} \quad \underset{0.2}{h_3}$

$P(\text{heads } C_2) = (0.6)^3$

$P(\text{heads } C_3) = (0.8)^3$

heads, heads, tail = 2/3

$P(C_1 \mid h,h,t) =$

$P(C_2 \mid h,h,t) =$

$P(C_3 \mid h,h,t) =$

$P(C \mid h,h,t) = \dfrac{P(C,h,h,t)}{P(h,h,t)}$

Can get this from sum of C_1, C_2, C_3

$P(C_1, h,h,t) + P(C_2\ h,h,t) + P(C_3, h,h,t)$

b. Calculate which coin was most likely to have been drawn from the bag if the observed flips come out heads twice and tails once.

heads, heads, tail = 2/3

$P(C_1 \mid h,h,t) = P(C_1)\, P(h \mid C_1)^2\, P(T \mid C_1) = 1/3 \times 0.2^2 \times 0.8$

$P(C_2 \mid h,h,t) = P(C_2)\, P(h \mid C_2)^2\, P(T \mid C_2) = 1/3 \times 0.6^2 \times 0.4$

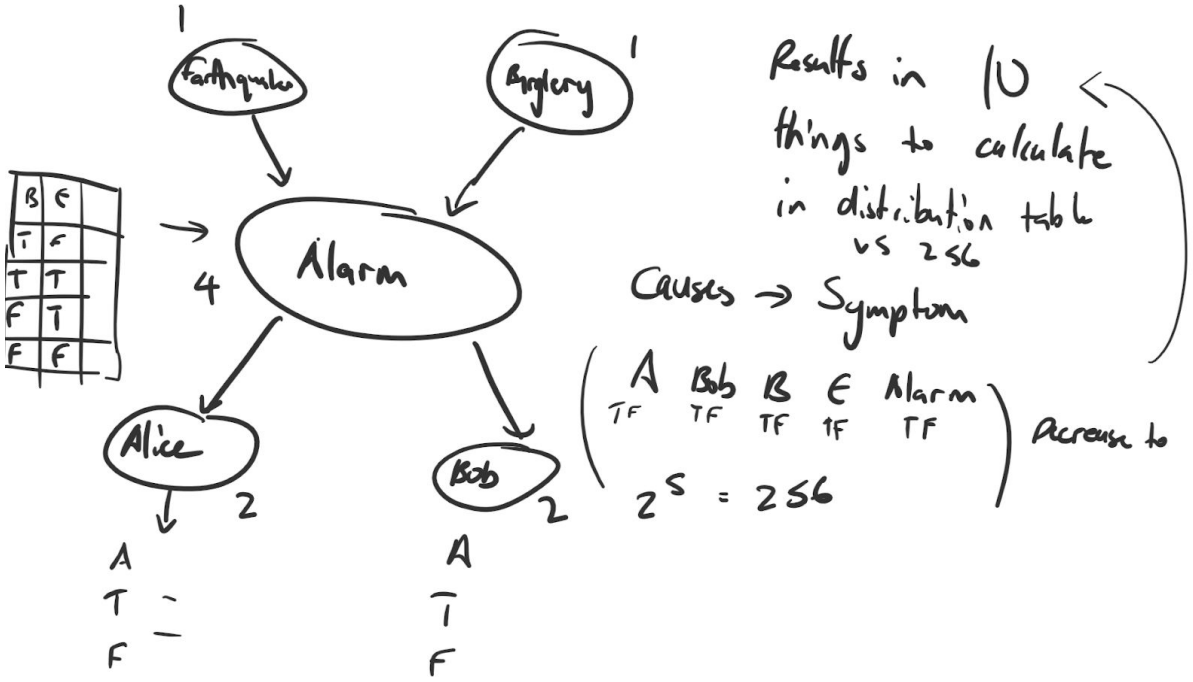
$P(C_3 \mid h,h,t) = P(C_3)\, P(h \mid C_3)^2\, P(T \mid C_3) = 1/3 \times 0.8^2 \times 0.2$

c. Can you think of a practical application where this type of model

could be useful?
Gambling would be the main one

9.2 (based on RN14.4) Consider the alarm network from the lecture slides.

a. If we observe Alarm = true, are Burglary and Earthquake independent? Justify your answer by calculating whether the probabilities involved satisfy the definition of conditional independence.



They are no longer independent. Once the alarm goes off and you know it's not an Earthquake then it's most likely a burglary and vice versa.
P(E, B|A) = P(E|A) P(B|A)
If they aren't equal then they're independent, otherwise they are not
Safer way:
P(A, B) = 0.25
P(A, B) = P(A) P(B)
Therefore if P(A) = 0.5 then P(B) = 0.5

b. You are driving to work, and you hear on the car radio that a minor earthquake has just occurred near your home. When you get to work, you see that Bob tried to call you, but left no message. There has been no call from Alive. How would you calculate the probability of a burglary having occurred given the above information?
P(Burglary|Bob called, A didn't call, Earthquake)
= P(Burglary, Bob, A', E)/P(Bob, A', E)
You know that the earthquake has already gone off so the burglary is very unlikely
Use the bayesian network to get all the other values

Note: If you have something missing in your probability you should consider the all possible values of the missing variable
E.g. a, b, c, d where b can be 1, 2, 3
p(d|a, c) = p(d, a, c)/p(a, c) where p(d, a, c) should consider when b is 1, 2 or 3
p(d, a, c, b=1) and so on...

Pls help i have no idea what is going on

9.3 (based on RN14.7) Consider the following simple belief network for diagnosing a car.

a. What conditional probability tables would you need to provide use this network, and what is the minimum number of entries you would need to specify in these tables?

b. Assume that the car does not start, but the radio works. How would you calculate the probability that the car is out of petrol?

Week 11

11. Robotics

Robots are not 100% accurate - there is probability involved

Say P(+|o) = 0.9
Given + means detects obstacles and o means there is an obstacle
P(o|+) = ?
= P(+|o)P(o)/P(+)

P(+) = P(+|o)P(o) + P(+|!o)P(!o) <- i.e. probability of a reading (this *changes every reading*)

P(o|+,+) = take away the center part = P(+|o)P(o|+)/P(+) <- note how the 2nd part has been computed by above

P(o|+,+,+) = P(+|o)P(o|+,+)/P(+) <- again use the part above for the 2nd part

The above process is called ***incremental bayes***.
Keep doing this until the probability of something happening is “really accurate”
Want to make sure all your choices are the best ones you can make with the given info instead of wasting time.

11.1 Suppose for some environment, the odds of there being an obstacle present are 1 in 10 and that a range sensor has a false positive rate of 30% and a false negative rate of 30%.

a. **What is the probability that an obstacle is present if the detector returns three positives in a row?**
   P(o|+, +, +) = P(+|o)P(o|+,+)/P(+)
   P(o) = 0.1

P(!+|o) = 0.3
P(+|o) = 0.7
P(+|!o) = 0.3
P(!+|!o) = 0.7
P(o|+,+) = P(+|o)P(o|+)/P(+)
P(o|+) = P(+|o)P(o)/P(+)

P(+) = P(+|o)P(o) + P(+|!o)P(!o)
P(+) = 0.7*0.1 + 0.3*0.9 = ⅓

P(o|+) = 0.7*0.1 / 0.33333 = 0.21

P(+) for part 2 = P(+|o)P(o|+) + P(+|!o)P(!o|+)
$P(+_2)$ = 0.21*0.7 + 0.3*0.79 = 0.35313

P(o|+,+) = 0.7 * 0.21 / 0.35313 = 0.41627729...

$P(+_3)$ = P(+|o)P(o|+,+) + P(+|!o)P(!o|+,+)
$P(+_3)$ = 0.7*0.41627729 + 0.3*(1-0.41627729) = 0.466510916

P(o|+, +, +) = 0.7 * 0.41627729 / 0.466510916 = 0.6246244042872515

b. What is the probability of no obstacle if the detector returns a positive followed by a negative?
P(A|B) = P(B|A)P(A)/P(B)

P(+) = P(+|o)P(o|!+) + P(+|!o)P(o|!+)=
= 0.7*0.79 + 0.3*0.79 = 0.789999
P(!o|+,!+) = P(!+|!o)P(!o|+)/P(!+)
= 0.7*0.3/(1-0.78999999) = 1????

Assignments

Assignment 1 - *Watch Your Back!*

A game called “Watch your back” - played on an 8x8 checkerboard.

First 6 rows are divided from the opposite 2

Placement Phase:

Players take turns putting 12 checkers on the board. The player may only place pieces within their allocated 6 rows

You may *not* put checkers in the corners

You win the game by ***defeating the other player's pieces by surrounding them on an axis*** (vertical or horizontal, NOT diagonal)

If you put a piece in between two enemy pieces it'll be taken instantly (p.s. Don't do this).

The player who places the piece has priority: e.g. if the board was:
B W B W <- then whoever placed their piece last would win
The corner acts as a player's piece - allows enemy to surround player
C W B <- W will be eaten by B
(corner can be used by either player)

You *must surround a piece in one tile* - you cannot have:
B W W B

Movement Phase:

can still eliminate / be eliminated

You can move around {up down left right} provided you are not on the edge of the board - no diagonal movement

You are able to move *past the 6th row line*
You cannot move into corners

You can jump over pieces (like checkers)
B W _ -> _ W B
W W _ -> _ W W
(nothing happens to the piece in the middle)
B W W _ -> B cannot jump over two Ws

At the end of the turn all the surrounded pieces will be eliminated. Again whoever moves gets priority.
- one move per turn (for each player)

There is no way to recover pieces 🔥

You can surround multiple pieces in an "L" shape (try and do this)
B
W
_ W B -> can eat two Ws at once
If you somehow cannot move, then you lose your turn (lol)
- can't pass on a turn

After 64 moves, if there are any pieces on the outer layer of the board, they will be removed (what is this PUBG stuff right here) and the boundaries will move inward, new corners!
Note that this includes the corner.

After another 32 moves, the edges will move in again.
If somehow the game keeps going the system will end it.

Possible to have a draw, maybe.

Questions about the game:

Everyone is pirates
The reason why people die is cause they get stabbed in the back
Read the elimination spec closely

Can you jump over your own pieces?
Yes. Yes you can.

How does the game end?
If either player has 1 piece left, the other player (with 2 or more pieces remaining) wins the game.

If the boundaries shrink and you get cornered by the corners, what happens?
The cornered piece should die but Matt hasn't decided yet lol