

CCPS 721 Artificial Intelligence

Extra Slides and Observations, Version March 12, 2023

To support the original classic nineties AIMA slides



Module 1a: Agents and Environments

Reminder of How Life Was Like Back in 2003

“Here is a heuristic for getting to someone’s house: Find the last letter we mailed you. Drive to the town in the return address. When you get to town, ask someone where our house is. Everyone knows us—someone will be glad to help you. If you can’t find anyone, call us from a public phone, and we’ll come get you.”

Steve McConnell, *Code Complete, 2nd Ed.*



GOFAI

- This course is given on the **Good Old Fashioned AI** perspective
- All problems are dealt with using high level **symbolic representations**
- Models consist of symbols that explicitly refer to things in the problem
- For example, in a chess playing program, we can identify the exact variables and memory locations that refer to a particular white pawn
- Philosophical basis on **physical symbol system hypothesis**
- Modern AI advances based on **subsymbolic** and **connectionist** approaches
- Neural network computation emerges from local actions in network structure
- These days, "GOFAI is dead"... or is it?




Rationality

- A **rational** agent is embedded in some environment
- The agent chooses its **actions** aiming to maximize the expected value of some **performance measure**, based on its **observations**
- Agent can't just assign everything to be how it wants, but must achieve the desired outcome in the environment through its limited set of actions
- Agent must make a choice between two or more **mutually exclusive** actions, otherwise there wouldn't be any decision making
- **No backsies**: agent cannot undo actions to try out different actions



Moravec's Paradox

- Humans are good in operating in unknown physical reality, but need to be heavily trained to think using abstractions
 - Computers are the opposite of this, so they work best in problems that can be fully abstracted away from the underlying physical media, warts and all
 - **Moravec's Paradox** is the observation that even though computers can play chess and other games better than humans, they can't do simple things that any healthy five year old is expected to do
 - Computers can't "walk and chew gum at the same time"
 - We are good at catching frisbees, despite being not so good in solving differential equations that govern their flight paths
- 

Intelligence Doesn't Require Sentience

- Many animal species have evolved to perform actions that look like they are product of conscious reasoning, despite that fact that "nobody is home"
- Environmental selection pressure has produced a working state machine whose copies are more successful than their competitors
- "It's competence, not consciousness, that matters"
- Also **Baldwin effect** of species evolving towards direction where they are more inclined to learn important things about their environment



Intentional Stance

- Daniel Dennett's three levels of abstraction for explaining and predicting the future behaviour of some system, depending on its complexity
- **Physical stance:** analyze system based on its internal structure
- Best level of explanation for a light in a room with its switch
- **Design stance:** analyze system based on its purpose, function and design
- Best level of explanation for a thermostat
- **Intentional stance:** analyze a system based on its assumed mental states, goals, beliefs and desires (even if imagined)
- Best level of explanation for computer chess program

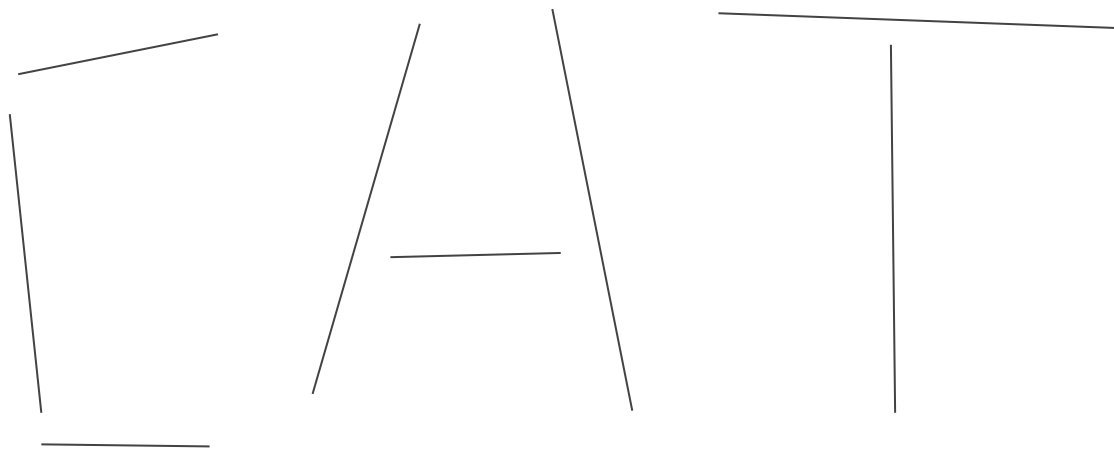


AI-Completeness and Microworlds

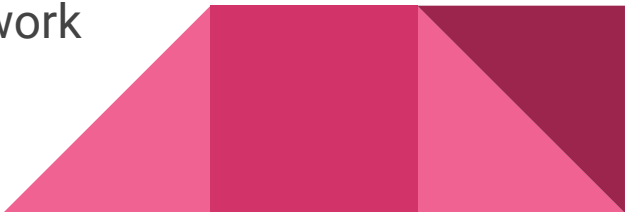
- All parts of reality are connected to each other, there is only one reality
- Trying to create a complete model of some part of reality ultimately requires modelling all of reality and its interconnected causes and effects
- A problem is said to be **AI-complete** if solving it turns out to be equivalent to solving the general artificial intelligence problem
- In practice, we isolate part of reality into a self-contained microworld
- Microworld is assumed to be insulated from the rest of reality, or its connections are modelled as statistical randomness



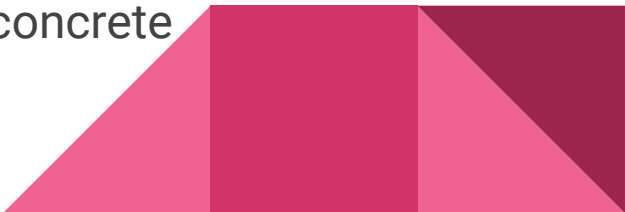
Recognize Letter A



Winograd Schemas

- Beautifully simple illustration of why general artificial intelligence is difficult
 - **Winograd schemas** are so trivial for humans that we don't even realize they are a problem, yet extremely difficult to analyze and program
 - Given two sentences that have identical parse trees except some leaf node has been replaced, determine what word a later pronoun refers to
 - "I threw the hammer at the mirror, and it smashed to pieces."
 - "I threw the glass at the wall, and it smashed to pieces."
 - It's not possible even to diagram a sentence correctly without a complete and complex model of how physical and social realities work
- 

Games and Sports

- Games are specifically designed microworlds insulated from reality
 - Performance metric as explicit **scoring** criterion
 - Rules designed to guarantee **termination**
 - Some games such as chess and poker are mathematical abstractions independent of the underlying media
 - Chess move selection is not affected by the material of board and pieces
 - Computer simulation of chess or poker is still chess or poker
 - **Sports** are games that cannot be extracted from underlying physical media; soccer changes quite a lot if you use a ball made of concrete
- 

Observations and Actions

- Actions have **outcomes** in environment that affect the performance metric
- There must be at least two different possible outcomes, otherwise action selection is meaningless
- Agent is informed about its performance only after taking its action, but not given information about the outcomes of hypothetical alternative actions afterwards when it's all done
- If environment is not **observable** or **deterministic**, the rational action is not guaranteed to produce the best possible outcome, luck has an effect



Rational thought vs. Rational action

- Agent will generally use some type of internal **reasoning** to choose its actions
- Philosophical question of connection of rational thought and rational action
- Rational thought is supposed to produce rational actions
- However, environment is affected only by actions, not by thoughts
- A black box that thinks perfectly about thoughts but never acts is worthless
- Right action done for totally wrong reasons is still the right action



Autonomy and Learning

- After started and let loose in its environment, an **autonomous** agent makes its own decisions without consulting its creator or principal
- Of course, a mechanistic agent can only follow its programming
- A sufficiently complex agent can even pass the **Lovelace test**
- Analogous to Turing test, an agent passes the Lovelace test once it achieves something that its designer didn't expect it to be able to do
- A **learning agent** adjusts some of its internal parameters based on its experiences, in a manner that is likely to improve results of future actions



Result Merchants

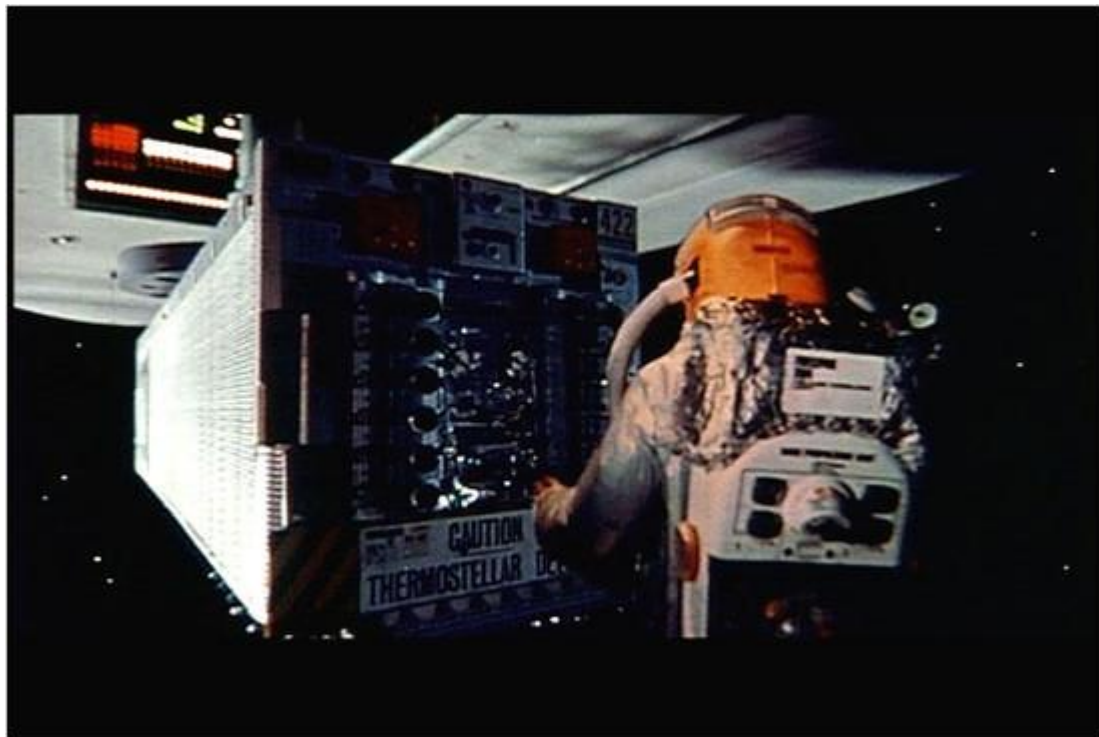
- Rationality is not omniscience in environments that are not observable or where the actions don't lead to deterministic outcomes
- An action is judged by its expected value, given the information available at the time the decision was made
- It's always easy to be a "Monday morning quarterback" and start **resulting** after the fact when all the cards are face up and the actions are obvious
- There are many branching futures, but only one linear past



Executing Actions in Actual Environment

- If the environment is observable and deterministic, the action sequence given by the state space search can be executed in actual environment
- In nondeterministic environments, the policy is constructed to cover all possible outcomes using techniques of Module 12
- For small state spaces, this **policy** can be precomputed as a lookup table
- A reflex agent chooses an action based on its current observations



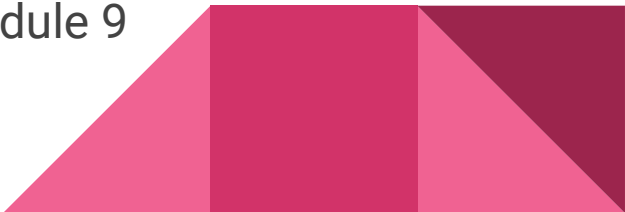


Agent vs. Principal

- **Principal** determines the **performance metric** hardcoded in the agent
- Especially no **wireheading** allowed for the agent
- After acting, the agent cannot simply sit down and declare that all is for the best in the best of all possible worlds
- Environment determines the **outcome**, principal determines its **value**
- Value is expressed as **utility**, which may not be linear with respect to value, such as with **money** whose utility is not linear
- This can make a difference if the outcomes have uncertainty



Example: Spam Filter Agent

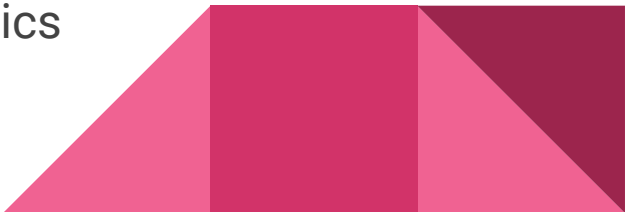
- Classic example of a computer agent serving the human principal
 - Episodic environment where each incoming email is an independent episode
 - Two possible actions: classify that email as "spam" or "ham"
 - Four possible outcomes; **true and false positives, true and false negatives**
 - Performance measure given as a **payoff matrix** for the four outcomes
 - Threshold for sensitivity for discarding the email depends on the relative costs false positives and negatives compared to true classifications
 - Also depends on actual frequencies of spam and ham emails
 - Done using **Bayesian analysis**, to be examined in Module 9
- 

Only the Actions Matter


- The principal or the environment do not give the agent an extra cookie for any elegance and extra cleverness shown during the reasoning process
- Only the actual achieved outcomes matter; a very pragmatic view of AI
- Once the agent has somehow determined that some action A is better than another action B , it doesn't need to know how much better A is than B
- Sometimes decision-making can be massively simplified by noting that even though the actual values of actions are unknown, action A is guaranteed to **dominate** the action B in this sense, so B can be ignored
- In that case, no need to estimate value of B



Paperclip Maximizer


- Akin to *Fantasia*, be careful what you wish for, since you might just get it
 - Machines don't care what you "intend" any more than fire would care that you "intended" to only burn down some weeds, not the entire neighbourhood
 - AI alignment problem is a wicked problem even to define, let alone solve
 - General intelligence AI that is given the task of maximizing some particular outcome for perpetuity will inevitably choose actions that keep it functional, for the simple reason that a broken agent cannot maximize anything
 - These actions include protecting itself against attempts to shut it down
 - The famous "**paperclip maximizer**" problem of AI ethics
- 

Actions That Gain Information

- Some actions are performed for the information that they reveal
 - This information can affect the value of future actions
 - If acquiring this information comes at a cost, it can sometimes pay to be **rationally ignorant** and not waste time acquiring useless information
 - Information has value only to the extent that it can **change** future actions
 - Unless the new information doesn't actually change the future action you were already planning to do, that information was worthless
 - Before asking any question from the environment, you should already know what you will do for every possible answer
- 

Module 1b: State Space Search

Models of Environment

- Once the agent executes the chosen action in environment, it is committed to that action, and there are **no backsies** allowed
 - Agent should create a sufficiently faithful **internal model of environment** to play around with, and use that model to first explore different possibilities
 - Abstract away the details that don't affect action selection for agent
 - **State spaces** are usually astronomically large, but their structure is regular enough to allow implicit neighbour enumeration of states
 - **State space searching** finds the best action sequence in model
 - Agent prepares a plan of actions to execute in the actual environment
- 

Two Famous State Spaces

- Some environments and optimal policies can be expressed quite succinctly

Blackjack Basic Strategy Chart											
1 Deck, Dealer Stands on All 17s											
Hard Total	Dealer Upcard										
	2	3	4	5	6	7	8	9	10	A	
5-7	H	H	H	H	H	H	H	H	H	H	
8	H	H	H	D	D	H	H	H	H	H	
9	D	D	D	D	D	H	H	H	H	H	
10	D	D	D	D	D	D	D	D	H	H	
11	D	D	D	D	D	D	D	D	D	D	
12	H	H	S	S	S	H	H	H	H	H	
13	S	S	S	S	S	H	H	H	H	H	
14	S	S	S	S	S	H	H	H	H	H	
15	S	S	S	S	S	H	H	H	H	H	
16	S	S	S	S	S	H	H	H	R	R	
17	S	S	S	S	S	S	S	S	S	S	
Soft Total	2	3	4	5	6	7	8	9	10	A	
A,2	H	H	D	D	D	H	H	H	H	H	
A,3	H	H	D	D	D	H	H	H	H	H	
A,4	H	H	D	D	D	H	H	H	H	H	
A,5	H	H	D	D	D	H	H	H	H	H	
A,6	D	D	D	D	D	H	H	H	H	H	
A,7	S	DS	DS	DS	DS	S	S	H	H	S	
A,8	S	S	S	S	DS	S	S	S	S	S	
A,9	S	S	S	S	S	S	S	S	S	S	

(Pairs are listed on back of card.)
by Kenneth R Smith © 2008-2016 Bayview Strategies, LLC



States vs. Nodes

- Important the distinguish between **states** in the state space that models the environment, versus the **nodes** constructed in the search tree
- **Nodes correspond to paths in state space** from start state to current state
- Unless the state space is already a tree, the same state can appear multiple times as search tree nodes
- Each node has a well-defined **distance** from start node
- Root node of the tree corresponds to empty path from start state to itself
- If some node corresponds to path α in state space, its child node reached with the action a corresponds to path αa



Frontier Search

- All state space search algorithms are special cases of **frontier search**
- Maintain a queue of **active leaf nodes** called the **frontier**
- Algorithms pops the next active node from the frontier to **expand**
- Expanding a leaf node adds all its children to the tree and frontier
- Algorithm **terminates when a goal node is chosen for expansion**
- Note: when goal node is added as a leaf, can't terminate yet




Other Algorithms As Special Cases of Frontier Search

- **Breadth-first search**: use a **FIFO queue** as frontier
- **Depth-first search**: use a **LIFO stack** as frontier
- **Uniform cost search**: use a **priority queue** as frontier, always expand frontier node s with lowest path cost $g(s)$
- Uniform cost search is special case of Dijkstra's algorithm restricted to operate in a tree, no node can be reached in two different ways
- **A***: use a **priority queue** as frontier, always expand frontier node s with lowest **heuristic** path cost $g(s) + h(s)$



Markov Property

- Environment where only the current situation determines the rational action, but the past history of how you got in that current situation is irrelevant
 - Future is **conditionally independent** of the past, given the present
 - $P(\text{Future} \mid \text{Present}) = P(\text{Future} \mid \text{Present and Past})$
 - Past cannot affect the future without going through the present
 - Blackjack: if you currently have a hard 15 against the dealer showing a nine, it doesn't matter whether your cards are 8-7 or K-2-3
 - If the environment is fully observable, agent doesn't need to maintain history
 - If environment not fully observable, past state information may reveal some otherwise unseen parts of the current state
- 

Triangle Inequality

- In a state space, a black box successor function allows us to move around the state space, but doesn't tell anything about global structure
- Global structure of state space that model some real environment will reflect the "laws of nature" of that environment
- Especially important is **triangle inequality**: the shortest path between two points is a straight line, and the space has no "short cuts" between points
- The true distance between states A and B is at least as long as the distance between those states "as the crow flies"



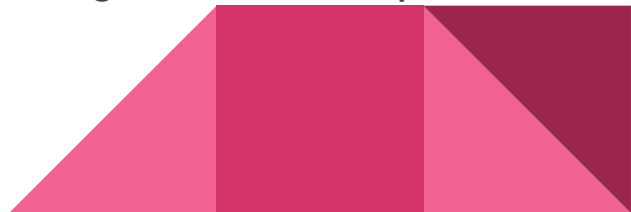
Monotonic (Consistent) Heuristic

- For the A* algorithm to guarantee finding the shortest path, the heuristic must be **admissible** so that it never overestimates the true cost to goal
- If the heuristic is also **monotonic** (or **consistent**), $h(u) \leq c(u, v) + h(v)$
- Distance to goal cannot decrease more than transition cost
- For example, if $h(u) = 10$ and $c(u, v) = 6$, then $h(v)$ must be at least 4 (DUCY?)
- Note that $h(v) = 5$ is only a lower bound; could have even $h(v) = 1000000$
- As special case, if h is identically 0, this says that $0 \leq c(u, v)$ (**Dijkstra**)
- Using a monotonic heuristic, no node will be expanded more than once during the execution of the A* algorithm



Optimality of A* with Heuristics

- If heuristic h_1 dominates heuristic h_2 , then the A* algorithm will never expand more nodes when using heuristic h_1 compared to using heuristic h_2
- Especially if h gives the distance to the nearest goal perfectly, A* would only expand the nodes actually on the shortest path
- Also, a more general result of optimality of A* algorithm itself
- A* can be proven to be the best possible algorithm among all possible state space search algorithms, given the same information
- Given the same state space information and heuristic function, every search algorithm **must** expand same nodes as A*, or risk missing the shortest path



Markov Decision Process

- What happens if the fully observable Markovian environment is not deterministic, so that actions can have unpredictable consequences?
- We are dealing with "known unknowns", instead of "unknown unknowns"
- Planning a single action sequence is not enough, unless you get lucky when executing that action sequence in the environment
- Need to compute a **policy** that gives the best action in any state
- Since environment is Markovian, history of reaching that state doesn't matter
- Techniques for doing this calculation will be examined in Module 12





Module 2: Adversarial Search

Adversarial Search

- Search problems become significantly harder when **other agents** with possibly conflicting interests also get to act in the same environment
- **The enemy also gets a vote, and no battle plan survives contact with enemy**
- It is not enough to find shortest path to goal, since adversarial agents will not be co-operative and make the moves that we would like them to make
- Instead of **NP-complete**, search problems become **PSPACE-complete**



Assumptions for Minimax Search

- Exactly **two agents** try to maximize their own rewards
- **Zero-sum rewards** shared between these two agents
- **Complete information, deterministic** and **fully observable** environment
- Players take **alternating turns** making moves, and get to see the opponent's chosen move before they commit to their own next chosen moves
- **Combinatorial game theory** is fully solvable in theory, not in practice
- General game theory loosens all these assumptions, and needs more general **game theoretical analysis** that is ad hoc even for seemingly simple games



Components of Minimax Search

- The same general purpose minimax search algorithm can play any game
- Just have to plug in the following subroutines as **black box functions**
- **Move generator**: subroutine that produces all possible moves in the given state, along with their successor states
- Move generator possibly with local ordering for move quality
- **Terminal state recognition**: recognize that game is over, and return the score
- For large game trees, need **static estimate of state value**, evaluating how good a state is without looking at its successors



Nash Equilibrium

- For a large family of "games", every game is guaranteed to have at least one **Nash equilibrium strategy** that maximizes the expectation for players
- Other players deviating from their Nash equilibrium strategies can never harm those players who stick to their own Nash equilibrium strategies
- In general games, Nash equilibrium strategies are **probabilistic** (for example, the game of rock-paper-scissors)
- For deterministic observable two-player zero-sum games, Nash equilibrium strategy collapses into a single line called the **principal variation**

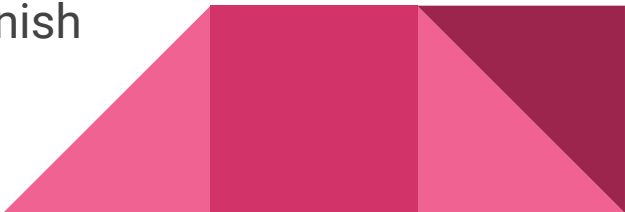


Infinite Move Games Might Not Have Equilibrium


- Nash equilibrium can be proven to exist for a large number of systems that we can think of as "games"
- Any number of players, uncertainty, makes no difference
- If the players can choose from infinite number of moves, Nash equilibrium might not necessarily exist
- Consider a game where two players simultaneously say a number, and the larger number wins
- No Nash equilibrium can possibly exist, naming $n+1$ always dominates n



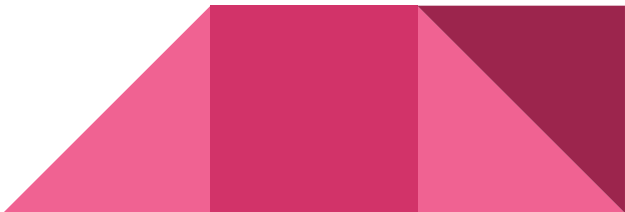
Principle of Indifference

- If you are following your Nash equilibrium strategy in any game, opponents deviating from their Nash equilibrium strategies cannot harm you
 - In fact, you can even honestly announce that you are following your Nash equilibrium strategy, and this information doesn't help the opponents!
 - You are simply indifferent to what the opponents do, always have a counter
 - In practical games, either Nash equilibrium strategies are not known, or the players don't have resources to compute and follow these strategies
 - **Trick play:** make an intentionally suboptimal move that leads to a complex situation, trusting that weaker opponent does not punish
- 

Partisan and Impartial Games

- Some games (for example, nim) are **impartial** so that on their turn, both players can make the exact same moves
 - Opposite of **partisan** games such as checkers, chess and backgammon
 - **Sprague–Grundy theorem**: any complete information impartial game is essentially equivalent into a position in the game of **nim**
 - Some impartial games (most famously, **Chomp**) can be proven a win for first player with a nonconstructive **strategy stealing** argument
 - If the game were a win for second player, first player starts by making the move that would have been second player's winning response
 - Effectively "turns the tables" in the game
- 

Playing Against Suboptimal Opponents

- Against opponent that deviates from their Nash equilibrium strategy, you may gain more than your fair share by correctly deviating from Nash equilibrium
 - In rock-paper-scissors, against an opponent who plays the trusty old rock more than $1/3$ of his moves, tend to play paper more than $1/3$ of your moves
 - "When playing against an idiot, you must also play like an idiot"
 - In poker, against player who folds too often, bluff more than optimally
 - In poker, against a wild bluffer, call liberally and raise conservatively
 - More players entering the ensuing juicy game between two idiots will then break this balance and force return to normalcy
- 

Negamax

- Most material on two-player complete information games makes one player to be maximizer and the other minimizer, as if they were stock characters in some cartoon melodrama
- **Negamax** has both players as symmetric maximizers, each player being the hero of their own story while the other one is a villain
- Without moralism, two entities caught in kind of a Sartrean hell chained to each other without recourse to join forces against a sadistic demiurge
- "Hell is other agents"



Negascout

- Correctness proof of alpha-beta required that the true value of state is between the alpha and beta parameter values at that node
- If this is not the case, the search will either **fail high** or **fail low**
- As noted earlier, once an action A is known to be better than action B , we don't need to waste time computing how much better it is
- Idea: evaluate the first move "for realsies", then use **null window search** to determine if the next move is better or worse
- Only if the next move turns out to be better, evaluate it again but for real



MTD(f)

- **"Memory-Enhanced Test Driver"** combined with **transposition table**
- An even better application of alpha-beta pruning discovered in 1994
- To compute the minimax value of a move, use only null window searches where $\alpha = \beta$, initially some local over/under estimate for move value
- Depending on whether search fails high or low, use binary search approach to pinpoint the value of the move
- Same as in negascout, can always use best move found so far as α

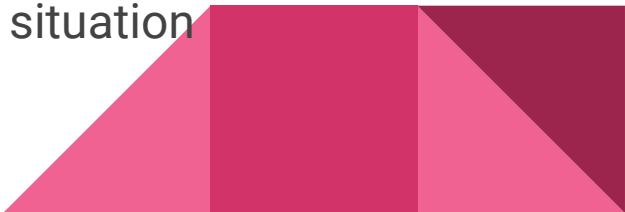


Transposition Tables

- Since the actual state space of the game is not necessarily a tree, same game states can be encountered in several branches of the DFS search tree
- If the game is Markovian, these nodes all have the same value
- Idea: use a hash **transposition table** to remember the nodes and their values
- Combine this with **iterative deepening**: use a hash table to remember the best move for each state during iteration of depth limit d , and then try that move first when that same state is encountered during iteration of depth limit $d + 1$
- Want to get those sweet alpha-beta cutoffs as soon as possible



Horizon Effect and Quiescence

- Minimax algorithm can look ahead only some fixed number of **plies**
 - Some move might look much better than it actually is, because it postpones the inevitable loss that is beyond the lookahead depth **horizon**
 - One solution is to extend lookahead depth for positions that are not **quiescent**, where the static evaluation of value of position differs greatly from the static evaluation of the values of its successor positions
 - Similar situation of **thrashing**: if the player has multiple moves leading to win with different depths, should ensure taking the fastest move
 - Otherwise can get stuck in a "Don't shoot, let's enjoy" situation
- 

Games and Metagames

- The **microworld** of the game is surrounded by some larger reality in which the agents and the game are situated
- A zero-sum two player game can be analyzed within that microworld, without having to make any assumptions about the surrounding **metagame**
- Everyone understands that when playing golf or such against your boss or an important client, you might not necessarily be trying your hardest to win
- Example of **Hanson's razor** (not to be confused with Hanlon's razor)
- Metagame is important for making the game itself worth playing



Multiplayer games

- Some material on minimax algorithm erroneously claims that adding more players to a complete information sequential zero sum game doesn't change the minimax algorithm used to analyze it
- "Bu-bu-but each player just gets his turn at that level of recursion, *maaan*"
- Not true: adding a third player creates **alliances** and **kingmaker** situations
- Such situations can't possibly be analyzed without looking at surrounding metagame, and the future games that these players will be involved in
- Under what assumptions would some player betray an alliance?



Nature as Third Player

- **Nature** can be thought of as a player that has no goals or metagame considerations (it always scores zero in the end, no matter what transpires)
- Nature especially will never take sides between the actual players
- Assuming that the third player is Nature, game can be analyzed within its microworld without having to resort to metagame assumptions
- In its turn, Nature makes random moves from some probability distribution
- Moves of Nature affect the possible moves and outcomes of actual players
- Randomness in general tends to help the weaker player (consider chess vs. heads-up no limit Texas hold'em poker)



Expectimax

- Random moves make search tree branch massively sideways
- Furthermore, value of state is no longer a single number, but a probability distribution, and unlike numbers, those are not totally ordered
- To allow comparison of moves, collapse each distribution to its **mean**
- **Expectimax** as generalization of minimax algorithm
- Analysis requires utilities of outcomes to be linear, which is not the case with money once the stakes get sufficiently high
- When playing for money, two distributions can have the same expected value, yet have massively different expected utility




Nonzero Sum Games

- How does analysis change if the outcomes for players are not zero sum?
- Game could have partially adversarial and partially co-operative aspects, depending on the structure of its state space
- Again, such games cannot be analyzed within the microworld of a single game, but the surrounding metagame needs to be taken into consideration
- Future games and reputation determine whether an agent should take a trust fall in the current match so that both agents to score better, if the other agent can betray them to grab all the moolah



Cooperative Games

- Game theory covers even fully cooperative games where an agent can never gain by making another agent worse off
 - For example, all agents will get the same score in the end, and should therefore work as a team to reach this shared common goal
 - To make such game interesting, environment isn't fully observable
 - Agents are not allowed to use **back channel communications**, but must communicate and coordinate within the physics of the environment
 - Actions have a component of communication in addition to actually achieving the desired outcomes in the environment itself
- 

Games of Incomplete Information

- Unlike chess or backgammon, most card games don't allow players to observe the entire current state of the game
- State space search with minimax doesn't work
- Even for one player, such a system becomes a **partially observable Markov decision process (POMDP)**, already a complex problem on its own
- For multiple adversarial players, the problem is highly nontrivial
- Best computer players for poker and bridge do not dominate humans the same way as best chess and Go computers players do these days



Games With Simultaneous Moves


- Minimax algorithm assumes a sequential game of alternating turns, so players don't have to commit to moves until they have seen the opponent's move
- Rock-paper-scissors as a sequential game would be pretty boring
- **Single-shot game** (e.g. soccer penalty shoot) analyzed as a table whose rows and columns are the possible moves of both players
- Nash equilibrium strategy no longer a deterministic principal variation line, but a probability distribution of moves for each player
- Players choose their probability distributions, after which the outcome is out of their hands after they have rolled the random dice to make the actual move
- "Pre-game is the real game", as the famous expression goes



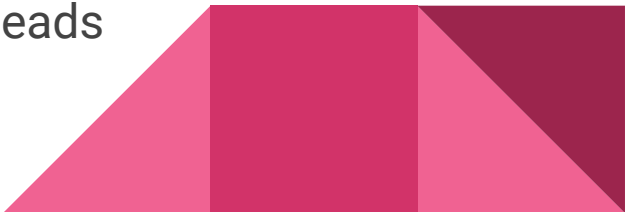


Module 3: Constraint Satisfaction

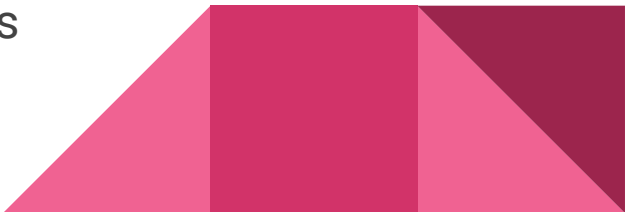
Local Constraints, Global Effect

- Each individual **constraint** connects together only a handful of variables
 - Because the same variable can occur in multiple constraints, these constraints create a complex network of dependencies and implications
 - Each individual constraint is simple, yet the resulting network is complex!
 - Even if each variable is a boolean truth value, and every constraint is a **disjunction** of these variables and their literals, any computational problem whose running time has a known upper limit can be encoded into a constraint satisfaction problem of this form
 - The famous **3-CNF-SAT** problem is **NP-complete**
- 

Race Against The Hydra

- Recall that running time of depth-first search is $O(b^d)$
 - Since d is same for all CSP branches, we can't do anything about it, so the branching factor b determines the maximum running time of the algorithm
 - Efficiency of backtracking greatly depends on **current variable selection**
 - Good choice for current variable eliminates remaining values from as many unassigned variables as possible, lowering b down the line
 - As we explore the tree, nodes try to branch sideways exponentially
 - The recursion is a race between these heads of the hydra and the assignments to current variables to chop off future heads
- 

Minimum Remaining Values


- Given a choice of which unassigned variable to fill in next, we choose the unassigned variable with the lowest remaining b
 - Best case scenario is that the chosen variable has only one remaining value after the previous assignments, making that level essentially a "**bye**" for us in an upside-down cup tournament
 - Since we have to fill in every variable anyway, we can't possibly save time by postponing the assignment to that variable to be done later
 - Besides, assigning that variable now eliminates possible values from other unassigned variables that appear in same constraints
- 

Accept the Things You Cannot Change...


- It seems counterintuitive to look for reasons to turn back as soon as possible, as if we were trying to fail instead of trying to complete the solution
- However, if the current partial assignment has already painted us in a corner so that no solutions can be extended from it, nothing we do can change that
- When going the wrong way, it's always better to turn back sooner than later!
- If the partial assignment is bad, we want to know this as soon as possible
- Knowing that the current partial assignment will lead to a legal solution would not help us anyway, since this knowledge doesn't change any of the steps that we will take completing that branch!



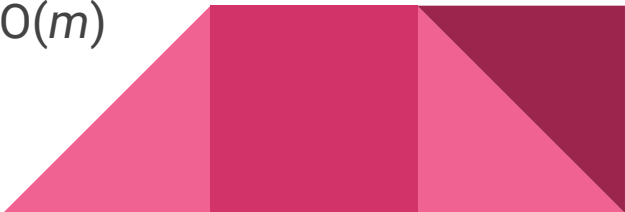
Auxiliary Data Structures

- During the partial assignment of variables, it is often beneficial to maintain some auxiliary data structure, shared by all levels of backtracking recursion
 - This auxiliary data structure speeds up finding the next variable to assign, and iterating through its possible values that don't violate constraints
 - For example, in n -queens problem, maintain three bit vectors that keep track of rows, diagonals and anti-diagonals have already been taken
 - Turn $O(n)$ check into $O(1)$ check, an order of magnitude speedup
 - In a sense this auxiliary data structure is redundant, since its contents are entailed by the partial assignment of variables performed so far
 - Trading small extra memory for lots of time is a good trade
- 

Updating and DOWndating Auxiliary Data Structure

- For each assignment of the next unassigned variable, this auxiliary data structure must be updated to reflect the new reality of partial assignment
 - Once the recursive filling of rest of the variables returns and that variable is unassigned again, the data structure must be downdated to its previous state
 - Often this is just inverse of the updating statements, but not always
 - A shared (global) **undo stack** is a general technique that always works
 - During update after variable assignments, push into undo stack the instructions required to reverse that update, somehow encoded to ints
 - After unassigning that variable again, pop and execute those instructions
- 

Dancing Links

- For problems that fill an array with some permutation of values from 0 to $n-1$, the **Dancing Links** technique by Donald Knuth allows the remaining values for the current variable updated, downdated and iterated over efficiently
 - Initially, all the values 0 to $n-1$ are arranged in a cyclic, doubly linked list with an extra sentinel node, as two $(n+1)$ -element arrays prev and next
 - When value i is assigned to current variable, it is $O(1)$ removed from this list
 - Genius flash of insight: when that variable is unassigned, node i can be restored back to its previous location in $O(1)$ time!
 - Iterating over the m values for current variable takes $O(m)$
- 

Backtracking Without Recursion

- Real world instances of CSP's can these days have hundreds of thousands of variables and constraints
- Recursive backtracking algorithm will surely die with a stack overflow
- Backtracking can be implemented non-recursively as a single while-loop, easiest to explain assuming variable array is filled left to right
- The loop keeps track of the variable k that it is currently at, initially 0
- If variable k has possible values remaining, assign first value and increment k
- Otherwise, unassign and decrement k to go back to previous variable
- When $k = d$ so that all variables have been filled, `yield` the current solution
- Easy to go do something else, and then continue generating more solutions



Search Problems With Cost Functions

- Constraint satisfaction problems can be generalized by finding a solution that not just satisfies the constraints, but optimizes the given **cost function**
- CSP's trivial special case of this with cost function 0 for legal solutions
- More nuanced cost function counts how many constraints are violated
- **Iterative improvement** algorithms maintain one complete variable assignment at the time, and always change individual variable values
- **Min-conflicts**: As long as some constraint is violated, reassign some variable so that the constraint becomes satisfied




Tabu Search

- An improvement of hill climbing to prevent getting stuck in local maxima
- Maintain a **tabu list** of k most recently visited positions
- Same as hill climbing, always move to the neighbouring position with the highest value, except that the algorithm is not allowed to move to any position in the tabu list
- Even a move to a lower-value neighbour position is allowed, if no better position in the tabu list is available
- Generalization by having multiple searches going on simultaneously, trying to hill climb while avoiding each other



Genetic Algorithms

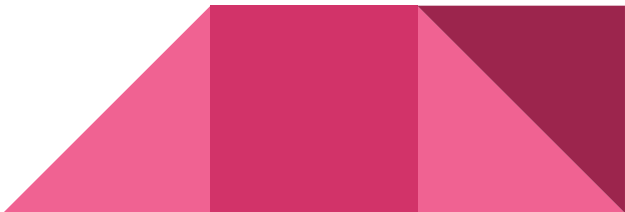
- Technique for optimization problems where each variable assignment is legal, but their costs are different
 - Maintain a **population** of solution candidates, called a **generation**
 - Use each generation to produce the next generation of solutions
 - Repeatedly choose two solutions from the current generation, somehow statistically favouring solutions that have a higher fitness
 - Combine these two solutions with **crossover** to create two new solutions
 - Possibly **mutate** individual variables randomly with small probability
 - Iterate new populations until "good enough" solution appears
- 

Building Block Lemma

- **"No Free Lunch" theorem** of search algorithms says that over all possible search problems, every search algorithm is equally good
- Should use a search algorithm that "fits" the underlying fitness landscape in that it moves correctly towards optimal solutions based on the information that it gets from current solution
- Genetic algorithms work for problems that have a **"building block"** property: solutions with given partial building block are better than average
- Search will direct towards creating and maintaining these building blocks



Variations of Genetic Algorithm

- Instead of normal one-point crossover to combine two solutions, use two-point, three-point, or higher crossover
 - Variations of crossover to maintain legality of solutions in problems such as travelling salesman
 - **Elitism** copies the k best solutions of current generation to next
 - If the cost function isn't explicit, but it's possible to compare two solutions, can use **tournament selection** (akin to a cup tournament)
 - Can maintain **multiple populations** and evaluate them in parallel, with occasional **migration** of best solutions
- 

Genetic Programming

- Generalization of genetic algorithm so that the solutions are **programs**
- Suitable for problems for which no analytical solution is known
- Fitness of a program is measured by giving it a bunch of test cases with known expected correct solutions, and measuring the error of the answers produced by this program over the entire test suite
- To make crossover possible, solutions are expressed as **parse trees**
- Crossover of two solutions swaps random subtrees from both parents
- Mutation replaces randomly chosen subtree with a random subtree



Modules 4 to 6: Prolog


Prolog, As Seen By ChatGPT...

Prolog, a language like no other,
Declarative, with rules to discover.
With facts and queries, it leads the way,
Finding solutions, in an efficient way.

It's logic-based, a different view,
Thinking in predicates, it's easy to do.
A symbol of reason, and smart design,
Coding with Prolog, a new paradigm.

You state the facts, and it will reason,
With rules and queries, it's the right season.
To build knowledge bases, and solve problems too,
Prolog is the language, to see it all through.

So let us praise, this language of art,
For making our work, so smart.
A language that's unique, and worth exploring,
Prolog, a language, always adoring.



I Have Seen The Future And It Forks


"A language that doesn't affect the way you think about programming, is not worth knowing." (Alan Perlis)

"Prolog is so simple that one has the sense that sooner or later someone had to discover it. Why did we discover it rather than anyone else?" (Alain Colmerauer)

*"I used Prolog in a comparative languages course. The biggest program we did was a map-coloring one (color a map with only four colors so that no bordering items have the same color, given a mapping of things that border each other). I say biggest because we were given the most time with it. I started out like most people in my class trying to hack the language into letting me code a stinking algorithm to color a stinking map. Then I wrote a test function to check if the map was colored and, in a **flash of prolog**, realized that that was really all I needed to code."*



Prolog Terms

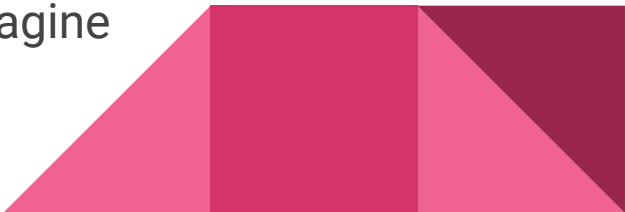
- The basic syntactic unit of Prolog programs is a **term** (an **expression**)
 - Structure of legal terms is defined recursively
 - Base cases of recursion are **constant** and **symbolic literals**, and **variables**
 - Symbolic literals start with **lowercase**, variable names start with **uppercase**
 - **42 and "joe"** are constant literals, `male` and `joe` symbolic, `Joe` is variable
 - Symbolic literals are not text strings, but an entirely different thing!
 - More complex terms can be built from applying a **functor** to **arguments**
 - Functor is always a symbolic literal, but its arguments can be any terms
 - `male(joe)` is a complex term, as is `foo(qux/"hello", 42+X/9)`
- 

Infix and Prefix Notation

- Every complex Prolog term is in the **prefix** form `functor(args)`
- However, for convenience for us humans, many functors allow **infix** form where the functor is syntactically between the two arguments
- For example, we write `joe * (7 + X)` instead of `*(joe, +(7, X))`
- Both forms become equivalent expression trees when parsed, and work exactly the same as far as computations are concerned
- Especially **comma operator** looks nicer as `a, b, c` than `,(a, ,(b, c))`



Homoiconicity


- Other languages that you have seen so far in your computer science studies make a hard distinction between **code** and **data** inside that language
 - Code cannot be assigned to variables, data cannot be executed
 - Prolog is **homoiconic** in that its code and data are **literally the same thing!**
 - Prolog code consists of **terms**, and so does its data
 - Complex data in Prolog is essentially **untyped**, but for complex terms, the **functor symbol** can be thought of as "**type**" of that data
 - All code can be treated as data and all data can be executed as code, for flexibility that ordinary languages cannot begin to imagine
- 

Bound and Unbound Variables


- All Prolog variables are untyped, and start as being **unbound (free)**
- During execution of a query, a variable can become **bound** to some term
- In computer memory, each variable is stored as a pointer
- Unbound variables are **null pointers**
- **Binding** a variable to a term assigns that variable pointer to point to the root node of the expression tree that represents that term in memory
- All variables are **final**: a bound variable cannot be bound to something else
- (**Execution backtracking** will unbind variables on the way back)



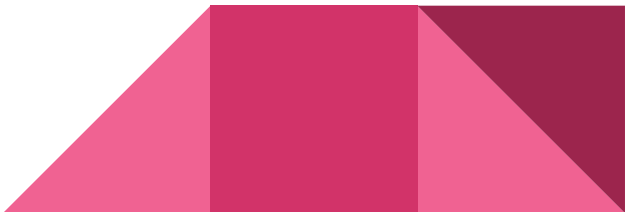
Prolog Predicates

- Interpreted as code, a Prolog term can be used to define a **predicate**
 - Each predicate represents some kind of **relation** between the entities that exist in the problem domain
 - Since all Prolog data consists of untyped terms, predicate **signature** consists of the **functor** symbol followed by the **arity** of that predicate
 - For example, predicate `male/1` defines the concept of "maleness"
 - In predicate logic, each predicate is **true** or **false** for the given arguments
 - Formulas in the given Prolog program determine for which arguments the given predicate is true, that is, **the query succeeds**
- 


Predicate Definitions With Rules

- In a Prolog program, a **predicate** is defined with one or more **rules** whose left hand side (**head**) is some term whose functor is that predicate
 - The right hand side (**body**) after the :- delimiter is a comma-separated list of the sufficient **premises** for that predicate to be true, followed by a **period**
 - Comma in Prolog stands for **logical and** between these **premises**
 - The right hand side can also be empty, meaning that the predicate is true unconditionally for its arguments
 - Predicate rules can contain variables, meaning that that predicate is true for any possible **instantiation** of those variables to any terms
- 

Anonymous Variables

- If the same variable name appears multiple times inside the same rule, it will refer to the same variable in all its occurrences
 - Essentially require that same value must be used in both places
 - Some variable might appear only once inside some rule
 - Effectively, values of such variables do not affect the success of that query
 - Such **anonymous variables** should be named with a single underscore _
 - (Read this underscore symbol out loud as "anything" or "whatever")
 - Multiple anonymous variables inside the same rule are separate variables
- 

Prolog Queries

- Execution of a Prolog program starts with a top-level query to **solve**
 - To solve each individual term in the top-level query, Prolog goes through all the rules whose LHS is the same functor
 - If the arguments **unify** with possibly some variable instantiations, Prolog will try to recursively solve the queries in the RHS of that rule
 - These may in turn trigger further recursive queries of arbitrary depth
 - Once all queries have been solved, query is successful, otherwise it **fails**
 - When the top-level query is successful, Prolog echoes its variable bindings, and pauses to ask if the user wants more solutions
- 

Negation as Failure

- Prolog is a very limited special case of **predicate logic**, yet **Turing-complete**
- Prolog rules are **Horn clauses**: a conjunction of positive premise literals together entail the positive literal at the head of the rule
- Predicate rules cannot contain negative requirements
- **Closed-world assumption**: in the world that the predicates talk about, only those relations hold that can be proven using the rules of the program
- If the query `male(q8x7)` fails, `q8x7` is not a male in that world




Predicates Represent Relations

- Prolog predicates are not functions, they do not "return" any values!
- Predicates represent **relations**, whose trivial special case functions are
- Beware the common beginner mistake by making a query `X = parent(bob)`
- This is still a legal query, but probably not what was intended!
- The intended query `parent(X, bob)` either **succeeds** or **fails**
- When some query succeeds, it binds the free variables X and Y into terms for which the query is successful
- Query `X = parent(bob)` succeeds by binding variable X to `parent(bob)`



Determinism and Nondeterminism

- In Prolog terminology, a predicate can be **deterministic**, **semideterministic** or **nondeterministic**, depending on how many times same query can succeed
 - Deterministic predicates succeed exactly once, regardless of arguments
 - For example, system predicates such as `writeln/1`
 - Semideterministic predicates either fail or succeed exactly once
 - Nondeterministic predicates can succeed any number of times (incl. zero)
 - Some nondeterministic predicates even allow **infinitely many solutions**
 - Depending on the predicate definition, Prolog may be able to **generate** all these solutions, especially if they form a nice linear infinity
- 

Execution Backtracking

- When a nondeterministic query succeeds, it may leave behind a **choice point**
- The Prolog interpreter remembers all choice points left behind in execution
- When some later query fails, program execution **backtracks** all the way to the **most recent choice point** that was left behind
- During backtracking, all variables that were bound when the execution was going to forward direction become unbound again
- Execution then continues forward from the choice point, allowing these variables to again become bound to some other terms




If At First You Fail...

- **Failure** of a Prolog query should not be confused with **errors** and **exceptions** in imperative languages such as Java or Python
- Failure doesn't mean that anything is "wrong" with your program or its data, but failures are necessary to prevent the Prolog execution from embarking in a fruitless infinite branch down the implicit search tree
- Think of the failure of a query as a friendly angel that tells you truthfully that there is nothing for you where you are trying to go, and therefore you should turn back now and try the next branch from the most recent choice point



Generate All Solutions Exactly Once

- When defining rules for some given predicate, your rules should be designed to produce **every solution** to each legal query **exactly once**
 - Furthermore, once the last solution to the query has been found, there should not be any **redundant choice points** left behind
 - Ideally, predicates should be maximally **reversible** and **generative** so that the query can leave as many arguments unbound as they want, and the rules will still generate all possible solutions exactly once
 - Realities of computation sometimes prevent this, even with seemingly simple basic integer arithmetic
- 

Order of Rules Inside Program

- Prolog differs from predicate logic in that how you order of rules with the same head functor inside the Prolog program can have consequences
- Obviously, affects order in which individual solutions are found
- For predicates with infinitely many solutions defined recursively, you should place the rules for the base case before the general case
- Otherwise backtracking dives into infinite branch and produces nothing, instead of producing the linear chain of solutions one at the time
- Ordering of rules becomes especially important with **cuts** seen later

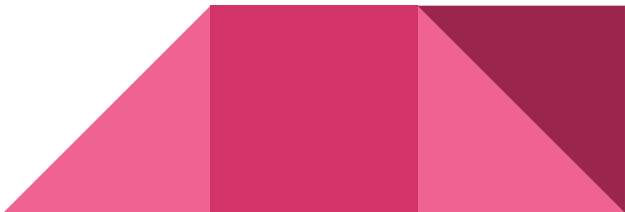


Order of Premises Inside Rules


- Prolog also differs from predicate logic in the behaviour of its inference engine so that order of premises inside the rule body is important
- Ordering of premises can affect **termination** properties of queries
- As in all programming, especially should avoid **left recursion**
- Even with terminating queries, rule ordering can affect **efficiency**
- Classic example of finding the first lady of the nation:

```
firstlady(X) :- female(X), married(X, Y), president(Y).
```

```
firstlady(X) :- president(Y), married(X, Y), female(X).
```



Unification (I)

- When going through the rules that would allow the query to succeed, that query must **unify** with the head of that rule for the rule to be used
 - **Unification** is a recursive algorithm that operates on pointers to expression trees that represent the terms, no string matching or such is involved
 - Both sides of unification can be arbitrary complex terms
 - An unbound variable unifies with any term, binding that variable to that term
 - Two literals unify if and only if they are the exact same literal
 - Two complex terms unify if and only if they have the exact same head, and their arguments unify pairwise
- 

Unification (II)

- To unify two terms, the algorithm finds the **most general unifier (MGU)**
- MGU makes **minimum commitment** to variable bindings to allow unification
- Unification predicate $=/2$ has been proven to be **semideterministic**: two terms either do not unify, or their MGU is unique (modulo variable names)
- In 99% of real use cases, using the predicate $=/2$ explicitly in Prolog rules is correct but clumsy: move the unification action to matching of rule head
- Predicate $\backslash=$ succeeds if the two terms do not unify
- This predicate can be defined since $=/2$ is semideterministic




Unification With Occurs Check

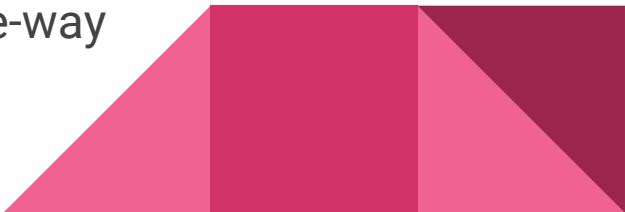
- What if the same variable occurs on both sides of the unification?
- Unification can create an infinite result, such as in the query $X = f(X)$.
- Solution would be infinite term $f(f(f(\dots)))$
- Encoded with pointers, the result expression tree contains a loop
- Prolog term echo mechanism fortunately recognizes this situation
- Unification algorithm does not perform the occurrence check, since it would turn $O(n)$ algorithm into $O(n^2)$ algorithm, hindering performance for no reason
- Logically sound unification with predicate `unify_with_occurs_check/2`



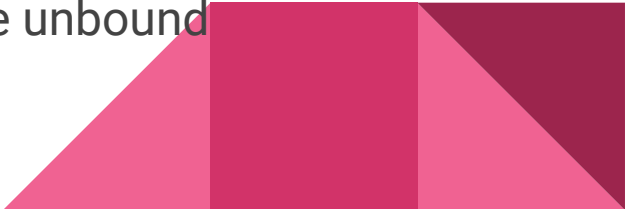
Unification With Forced Evaluation

- Prolog variables can be bound to arbitrarily complex terms
 - Prolog will never try to simplify these terms until you explicitly order it to do that by using the predicate `is/2`
 - `is/2` is used otherwise like `=/2`, except that `is/2` will first evaluate the right hand side before attempting to unify both sides
 - Closest thing to **assignment** of imperative languages in Prolog
 - Right hand side must be in a form that allows evaluation, otherwise the entire query will crash with an error
 - Arithmetic equality `:=/2` evaluates both sides before unify
- 

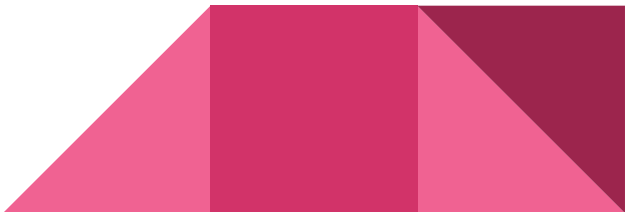
Reversibility

- In imperative languages such as Python or Java, each function/method call produces exactly one result
 - This result may be an aggregate object, but you get only one
 - Imperative functions cannot be executed in **reverse** so that you give them expected results, and they would return arguments to produce those results!
 - Prolog predicates don't make any distinction between their "arguments" and "results", but are fully **reversible**, assuming that their rules don't contain any predicates defined "one-way" for efficiency reasons
 - For example, **integer multiplication** and $is/2$ are one-way
- 

Unbound Queries and Results

- Some predicates require that some of their arguments must be bound to certain types of terms for the query to succeed
 - For example, predicate `between/3`, closest thing to for-loop in Prolog
 - The query `between(1, 10, X)` succeeds for all integer values `X` that are between 1 and 10, inclusive
 - First two arguments must be bound to integer values at time of query
 - Even though queries such as `between(2, X, 6)` would make sense (though with a linear infinity of solutions), implementation does not allow this
 - Gold standard is to allow any and all arguments to be unbound
- 

Lists

- Prolog does not use random access arrays for aggregate data, since these don't work well with notion of immutable data
 - Prolog aggregate data type is a **list** that is either **empty**, or consists of a **head** element followed by the **tail** of the rest of the elements
 - Easy to **add** or **remove** an element to the **front**, not to the end!
 - Syntactic sugar allows `[[42], foo(bar/7), X, bob(bob(bob))]`
 - Head is the first element, and can be any Prolog term
 - Tail is the list of remaining elements, **must always be a list**
 - In a singleton list such as `[42]`, tail is the empty list
- 

Lists and Unification

- Many list predicates are defined with recursive rules whose behaviour depends on the head and the tail of the list argument
- Term $[H \mid T]$ unifies with any list whose head is H and tail is T
- For example, $[H \mid T] = [1, 2, 3]$ succeeds with $H = 1$ and $T = [2, 3]$
- Term $[H \mid T]$ does not unify with the empty list $[]$
- Note important difference between $[H \mid T]$ and $[H, T]$, the latter unifying only with lists that have exactly two elements, no more and no less

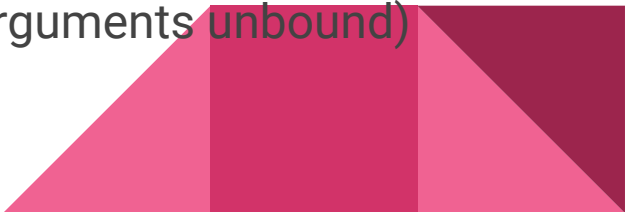


Appending Two Lists

- List predicate `append/3` is very useful in defining other list predicates
- Fully generative and reversible: `append(L1, L2, [1, 2, 3, 4, 5])` produces all six different results
- Even the query `append(L1, L2, L3)` produces all solutions
- Rules for `prefix/2` and `suffix/2` become one-liners
- When building rules for other predicates with `append/3`, note that running time of `append` is linear to the length of the first list
- Often more efficient to build lists recursively by adding new items to head



Introspective Predicates

- Some system predicates **inspect** the current state of the Prolog computation
 - For example, `statistics/2` can be used to gauge various metrics
 - Important family of introspective predicates determines **the type of term** that a particular variable is currently bound to, or whether it is unbound
 - Such predicates help define predicates that are more reversible
 - For example, predicate `plus/3` to replace the `+` operator
 - Depending on which one of the three parameters is unbound, the queries `plus(1, 2, X)`, `plus(1, X, 3)` and `plus(X, 2, 3)` all work correctly
 - (Hard exercise: define `my_plus/3` to allow any two arguments unbound)
- 

Cuts

- The only control structure in Prolog is the **cut**, denoted by **!**
- **Prolog is Turing-complete even without the cut**, but cuts can simplify code
- Cut is a deterministic predicate that always succeeds, executed for its side effect of **cutting away all choice points in the current query**
- Cut doesn't reach up to the queries above the current query
- Cut would perhaps better be called "**commit**", since it commits to the current branch in the search tree and ignores the solutions found by branches hanging from the choice points of the current query




If-Else Implemented As Cut-Fail

- How to simulate the **if-else structure** of imperative programming languages so that **precisely one** of the branches will be executed?
- Standard solution using **cut-fail**: write two versions of the rule
- Body of the first rule starts with the condition to be tested
- If the condition is true, do a cut to prevent second rule to be executed for the current query, and follow the cut with the positive branch of if-else
- The second rule should unconditionally just execute the negative branch
- The fact that execution gets to the second rule means that the condition is false, since otherwise cut would have taken effect



Red Cuts and Green Cuts

- In Prolog terminology, a **green cut** is one that doesn't eliminate any solutions
 - Rules with the green cuts have the same solutions as they would without the green cut, but produce these results faster and possibly without repetitions
 - Especially avoid the cardinal sin of Prolog rules, leaving a **redundant choice point** that we can reason from outside that cannot lead anywhere
 - A **red cut** eliminates some solutions that predicate would have without it
 - Recall the gold standard of making predicates maximally reversible by allowing any and all of their arguments to be unbound variables
 - Adding cuts to predicate rules goes against this noble goal
- 

Using Terms as Queries

- Noble attempt for a query: `X = male(bob), X.`
- Unfortunately, this doesn't work: query can't syntactically be a variable
- Need to use the metapredicate `call/1` that executes its argument as query
- Better: `X = male(bob), call(X).`
- Multiple versions of `call` allow more arguments to be added to query
- `X = male, call(X, bob)`
- Note also `call_with_depth_limit/2` for potentially executing queries in iterative deepening fashion



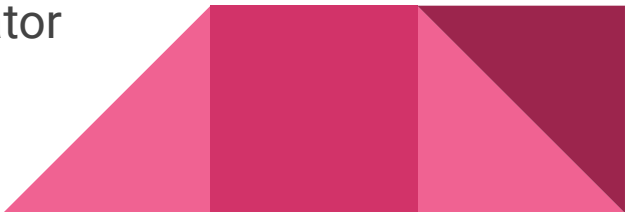
Prolog Negation

- Prolog does have a metapredicate `not/1`, but this should not be confused with actual logical negation, since Prolog only deals with positive terms
- **Negation as failure** with **closed world assumption**: precisely those things are true that can be proven to be true
- The query `not(Q)` succeeds if and only if `Q` itself would fail without solution
- Easy definition using cut-fail to **simulate if-else**:

```
not(Q) :- call(Q), !, fail.  
not(_).
```
- Riddle me this: how are `Q` and `not(not(Q))` different?



Finding All Solutions To Query In One Swoop

- For a nondeterministic query, Prolog produces solutions one at the time
 - Since there could be exponentially of solutions (for example, consider the list predicate `permutation/2`), we don't want to run out of memory
 - Handy metapredicate `findall/3` generates the list of all solutions for the given query, listed in the order in which the query would produce them
 - For example, `findall(Y, (between(-3, 3, X), Y is X*X), L)` would succeed with `L = [9, 4, 1, 0, 1, 4, 9]`
 - Note the use of parentheses to disambiguate between comma denoting the logical and, versus comma used as argument separator
- 

Ordinary Recursion in Java

```
public int factorial(int n) {  
    if(n < 2) { return 1; }  
    else { return n * factorial(n-1); }  
}
```

- The recursive call in the second line of this method is not **tail recursive**, since multiplication takes place after the recursive call
- Need one stack frame per recursion level, since the method needs to remember the current value of n at each recursion level



Tail Recursion in Java

- Introduce an extra **accumulator** parameter to enable tail recursion

```
public int factorial(int n) {  
    return factorial(n, 1);  
}
```

```
private int factorial(int n, int acc) {  
    if(n < 2) { return acc; }  
    return factorial(n-1, acc*n);  
}
```



Tail Recursion in Prolog

- Prolog rule is **tail recursive**, if the recursive query of that same predicate is the last term in the body of the rule, and all previous terms in the rule are (semi)deterministic so that they can't leave any choice points behind them
- Same as in imperative languages, Prolog can recycle the current stack frame
- Converting predicate to tail recursion a mechanistic 3-step process
- Step 1: Have the predicate call extended predicate with state arguments
- Step 2: Rule for base case to unify result variable
- Step 3: For general case, calculate the new state arguments from previous, and finish with tail recursive call to go to the next round



Simulating Structured Data Types In Prolog

- Prolog has only one universal data type, the Prolog term
- We can simulate **structured data types** of other languages by defining that data type to be any term with the given **functor**
- Arguments of term give the field values for that structured data object
- Since data is immutable, rules on such terms must determine the effect of applying some mutation operator to the term to produce a new term
- New and old term share as many subterms as possible
- See the examples `bankaccount.pl` and `bst.pl` to illustrate this principle

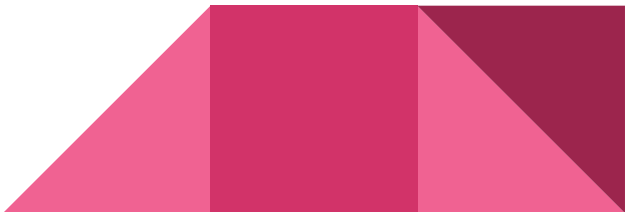


Constraint Logic Programming

- Ordinary Prolog variables are either unbound, or bound to a term
- Represented in memory as pointers, null pointer for unbound variable
- Forces the programmer to arrange rule logic so that variables will be bound to concrete integer values before any arithmetic is performed on them
- Not as flexible as logic programming could theoretically be, we want more!
- **Constraint logic programming** standard library extension allows a set of **lazy constraints** to be attached to each variable
- When top-level query succeeds, lazy constraints are echoed



Constraint Logic Programming Operators

- Equality constraint `#=` is symmetric between LHS and RHS and subsumes operations `is` and `:=`, internally reduces to these whenever possible
 - For other arithmetic comparisons, lazy constraints `#<`, `#>=` etc.
 - Lazy constraints are automatically propagated during program execution, and trigger backtracking as soon as they become impossible to satisfy
 - Predicates `indomain/1` and `labelling/2` can be used to iterate through all possible values of lazily constrained variables, essentially turning these variables back into ordinary Prolog variables when needed
 - Require the constrained domain to be finite, though
- 

A Limerick About Prolog, As Written by ChatGPT

There once was a language called Prolog,
Its rules and queries, oh so cogent.
It made programmers shout,
"This language is what it's about!"
A joy to code with, no need to cogitate.





Module 7: Propositional Logic

Implicit and Explicit Truths

- State space techniques generally require environment to be fully observable
- State space techniques cannot directly utilize known **laws of environment**
- Agent's actions are affected by reality of the environment
- Reality consists of **explicit (directly observable)** parts and **implicit** parts
- Implicit truths are still just as "real" as explicit truths, in that they have real effects on the consequences of different actions
- Implicit truths must be reasoned from the observed explicit truths, aided with the "laws of nature" of the environment that bind these together




Formal Reasoning


- Mechanistic inference engines perform logical reasoning based solely on the form of the sentences, without appealing to any ideas about objects that the symbols in these sentences happen to refer to
- Especially rewriting the symbols of a sentence does not change any **formal** properties of that sentence that could affect mechanistic inference
- If you agree that sentences "**All men are mortal**" and "**Socrates is a man**" entail the sentence "**Socrates is mortal**", then you should agree that the sentences "**All foo are bar**" and "**Xyb123 is foo**" entail "**Xyb123 is bar**"



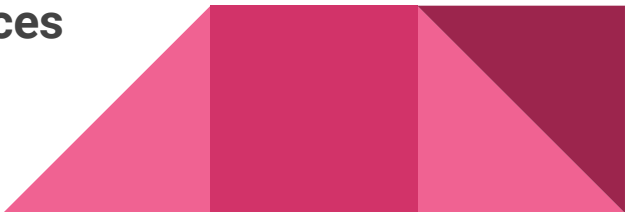
Worlds and Models

- A logical sentence is not "true" or "false" by itself, but relative to given **world**
 - The same sentence can be true in one world and false in another world
 - A world in which given set of sentences is true is called a **model** for that set
 - Sentence that is false in every world is a **contradiction**
 - Sentence that is true in every world is **valid (tautology)**
 - Sentences ϕ and ψ are **consistent** if $\phi \wedge \psi$ is not a contradiction
 - Note that the sentence " $2 + 2 = 4$ " is **not** a tautology, but depends on the meaning of the symbols 2, + and 4 in that world
 - Sentence " $2 + 2 = 2 + 2$ " is tautology (meaning of = is fixed)
- 

Do Not Confuse These Three

- **Entailment** $\phi \models \psi$: In every world where ϕ is true, so is also ψ
 - Entailment looks at semantics and worlds from "God's eye" view
 - **Inference** $\phi \vdash \psi$: Given formula ϕ as input, mindless machine doing inference can spit out the formula ψ as output
 - Inference looks at what we can program a computational device to do
 - **Implication** $\phi \Rightarrow \psi$: Express the notion of "If ϕ , then ψ " as a sentence stated inside the logic itself
 - Implication is intuitive **syntactic sugar** for sentence $\text{not-}\phi \vee \psi$
 - Do not hallucinate a **causal** relationship between ϕ and ψ
- 

Carroll's Paradox of Finite Inference

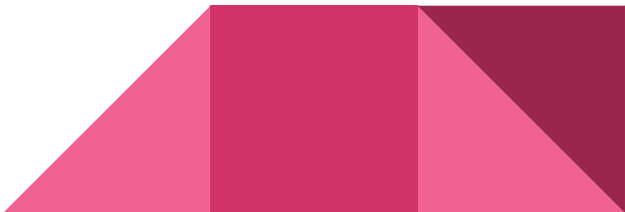
- Suppose Joe Palooka accepts the sentences "A" and "If A, then Z" as being true, but still doesn't accept the sentence "Z" as true
 - How would you convince Joe that he must accept also sentence "Z" ?
 - Whatever you now say, call that sentence *B*
 - Joe says that he accepts the sentences "A" and "If A, then Z" and your *B* as being true, but still doesn't accept the sentence "Z" as true
 - How would you convince Joe that he must accept also sentence "Z" ?
 - Whatever you now say, call that sentence *C*, and repeat
 - **Rules of inference can't themselves be logic sentences**
 - Otherwise we will get turtles all the way down
- 

Bivalence


- **Bivalence** is a semantic property of logic in that every sentence is either true or false in the given world, there is no third alternative
- Hold for propositional and predicate logic: Exactly one of the formulas ϕ and $\text{not-}\phi$ is true in the given world, by definition
- Does not hold for three– or multivalued logics, especially **fuzzy logic**
- In **Bayesian probability**, formula ϕ is either true or false in the given world, but our **degree of belief** in it can vary in range $[0, 1]$



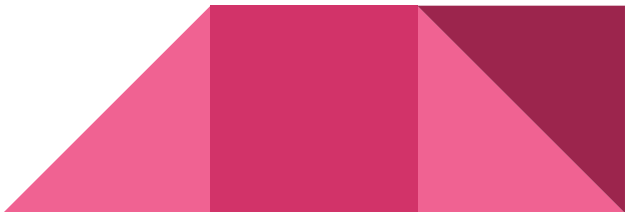
Law of Excluded Middle

- **Law of Excluded Middle** is often confused with bivalence
 - A syntactic property of logic, not semantic property!
 - Any formula of form $\phi \vee \text{not-}\phi$ is true in every world by virtue of its form
 - Need to be able to determine if two formulas ϕ and ψ are **logically equivalent** to determine if formula $\phi \vee \text{not-}\psi$ is valid
 - Holds for propositional and predicate logics, and Bayesian probabilities
 - May or may not hold for three-valued logics with truth value "undetermined", depending on whether this is **ontological** or **epistemological**
 - Does not hold for fuzzy logics
- 

Truth Functionality

- Logic is said to be **truth functional** or **compositional** if the truth value of a formula is fully determined by the truth values of its subformulas
 - Holds for propositional and predicate logic
 - Does not hold for Bayesian probabilities: Even if you know that $P(A) = 0.2$ and $P(B) = 0.5$, you don't know the probabilities $P(A \vee B)$ or $P(A \wedge B)$ from these
 - Does not hold for higher-order **modal logics** that use operators such as "knows" or "believes" or "sometimes" or "always"
 - "Lois Lane knows that Superman can fly" and "Clark Kent is Superman" do not entail the sentence "Lois Lane knows that Clark Kent can fly"
- 

Properties of Inference Engine


- **Monotonicity:** Adding new sentences to knowledge base can never decrease the set of sentences that can be soundly inferred from it
 - **Locality:** Inference rules can be soundly applied to a subset of sentences, without having to care about the rest of the knowledge base
 - **Detachment:** After a new sentence has been inferred, steps of its inference do not affect future inferences made using that new sentence
 - **Principle of explosion:** From a knowledge base that contains both sentences ϕ and $\text{not-}\phi$, any sentence whatsoever can be inferred
- 

Fuzzy Logic

- Modelling reality as shades of gray instead of binary black and white
- Not to be confused with Bayesian probabilities where bivalent reality is still black and white, we merely have varying degrees of belief about its aspects
- In **fuzzy logic**, the facts themselves in the world are gray, and each proposition A that refers to them gets a truth value $\mu(A)$ in $[0, 1]$
- Unlike probabilities, fuzzy logic is truth functional
- $\mu(A \wedge B) = \min(\mu(A), \mu(B))$, $\mu(A \vee B) = \max(\mu(A), \mu(B))$, $\mu(\text{not-}A) = 1 - \mu(A)$
- Smoother inferences for reasoning about the world, smooth control



Horn Clauses

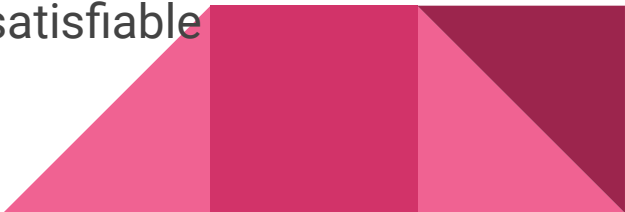
- Conjunctive normal form is implicative normal form rewritten using \vee
 - $(A \vee \text{not-}B \vee \text{not-}C \vee D)$ is the same formula as $(B \wedge C \Rightarrow A \vee D)$
 - Formula in a **disjunctive normal form** is said to be in **Horn form** if its every clause contains at most one positive literal
 - Bunch of positive premises imply exactly one positive conclusion
 - Inference algorithms for Horn clauses are simpler than algorithms for general clauses, since there is no branching for conclusion
 - **Modus Ponens** is complete reasoning algorithm for Horn clauses, using either **forward chaining** or **backwards chaining**
- 

Tseytin Transformation

- The conversion from arbitrary propositional logic to 3-CNF seen in AIMA slides can blow up the formulas exponentially in size
- Alternative technique of **Tseytin transformation** avoids exponential blowup by creating more propositional symbols for subformulas
- Identify every subformula in knowledge base that is not a literal, and create a new proposition symbol for that subformula
- Rewrite all formulas using new proposition symbols for subformulas
- Convert new clauses into disjunctions by replacing equivalence and implication with equivalent disjunctive structures



Converting from CNF to 3-CNF


- Conversion from propositional logic formulas to CNF as seen in ALMA notes is usually continued with an extra step that guarantees that each clause contains **at most three literals** (handy for many later conversions)
 - If a clause ϕ has four or more literals, split ϕ in half into clauses ϕ_1 and ϕ_2
 - Create a new proposition symbol Z that does not appear in clauses so far
 - Replace the clause ϕ with two clauses $(Z \vee \phi_1) \wedge (\text{not-}Z \vee \phi_2)$
 - To make both new clauses true, at least one literal in ϕ must be true
 - Repeat until each clause has at most three literals
 - New knowledge base is satisfiable iff the original is satisfiable
- 

Resolution Refutation

- Resolution rule is sound, but is not strong enough to produce all sentences entailed by the given knowledge base
- For example, can't produce $P \vee Q$ from the knowledge base $P \wedge Q$
- To prove that $\phi \models \psi$, show that $(\phi \wedge \text{not-}\psi)$ is inconsistent
- **Proof by contradiction** by deriving **empty clause**
- **Resolution refutation** is complete for propositional logic
- Since there can exist at most 3^n clauses for n propositional symbols (DUCY?), algorithm will return positive or negative answer in finite time



Converting Other Problems To Propositional Logic

- In theory of computation, **Cook's theorem** proves that any decision and search problem from the class **NP (non-deterministic polynomial time)** can always be converted into a set of equivalent propositional logic formulas
 - Propositions correspond to variables of the original problem, clauses correspond to constraints for values of these variables
 - Multivalued variables broken into unary or binary representation
 - **The original decision problem has a solution iff this formula is satisfiable**
 - Solution to the original problem can be read from the satisfying assignment of truth values to propositions in the satisfiability problem
- 

Example: Graph Colouring

- Consider the problem of colouring a graph with n nodes with k colours
- Create nk separate propositions C_{ij} to mean "Node i has colour j "
- Seems inefficient, but bear with us for a moment
- For each node i , create the clause $(C_{i1} \vee \dots \vee C_{ik})$
- Each node must have at least one colour
- (We don't need to say that each node has exactly one colour. DUCY?)
- For each edge (u, v) and every colour c , create clause $(C_{uc} \Rightarrow \text{not-}C_{vc})$ so that nodes connected by an edge can't share colour




Example: Sudoku Solver


- **Sudoku** is just graph colouring in disguise, cells are nodes and $k = 9$
- Two cells are connected by an edge if they are in the same row, in the same column, or in the same box
- Some cells have been given a fixed colour in the beginning
- Problems intentionally designed so that satisfying assignment is unique
- Generalization for Sudoku of arbitrary order n , not just $n = 3$
- Sudoku of order n has n^2 colours, rows and columns, each box is $n \times n$



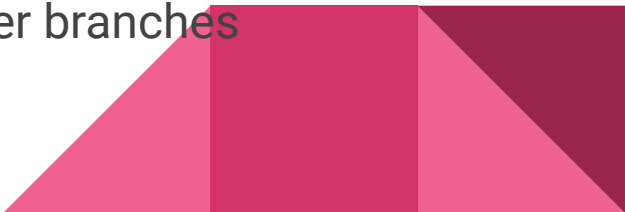
DPLL Algorithm

- Given a knowledge base in conjunctive normal form, find a satisfying solution
 - Constraint satisfaction problem where propositions are binary variables, and disjunctive clauses are constraints
 - Special form of clauses allows us to do better than ordinary backtracking
 - Instead of choosing a variable to assign both ways, choose some **active** clause $(X_1 \vee X_2 \vee X_3)$ that is still unsatisfied
 - Instead of trying eight combinations of truth values for three variables, need only try three combinations X_1 , $\text{not-}X_1 \wedge X_2$, and $\text{not-}X_1 \wedge \text{not-}X_2 \wedge X_3$
 - Clauses satisfied by assignment become **inactive**
 - Solution found when no active clauses remain
- 

DPLL Speedups

- Realization that once a clause has been satisfied by having one of its literals assigned true, it wouldn't help to make that clause "doubly satisfied"
 - Satisfied clauses no longer constrain assignments of remaining variables
 - **Unit propagation:** when some active clause contains only one unassigned literal, choose that clause to be the next one to be satisfied
 - As seen in the backtracking lecture, such forced moves don't create a branch and can never be worse than choosing some unforced move
 - **Pure literal elimination:** If some literal appears only one polarity in the remaining active formulas, make that literal true without branching
- 

DPLL Optimizations

- Need to use an efficient **data structure** to keep track of the clauses that are still active, and update and downdate that information in each assignment
 - Since clauses don't change during search, a preprocessing step computes for each literal the set of clauses where it appears
 - Algorithm speed depends on effective heuristics to choose the next clause to satisfy, and ways to detect having painted yourself in a dead end as soon as possible if this happens
 - At a dead end, **Conflict-Driven Clause Learning** analyzes the reason and adds new clauses to knowledge base to prevent this in later branches
- 

Binary Decision Diagrams

- Conjunctive normal form is good for resolution reasoning, but is only one of the many standard forms of propositional logic
- **Binary decision diagrams** are an alternative form as a decision tree
- From the bottom up, merge isomorphic nodes into the same node
- Given BDD's for formulas ϕ and ψ , efficient algorithms can construct BDD's for formulas $(\text{not-}\phi)$, $(\phi \wedge \psi)$ and $(\phi \vee \psi)$
- Resulting tree size generally **product** of sizes of trees for ϕ and ψ , but hopefully lots of **cancellation** happens while melding these trees



Virtues of Binary Decision Trees

- Once a BDD for the knowledge base formula KB has been constructed, the following otherwise difficult questions become (nearly) trivial to answer
- Evaluate the formula for the given variable assignment (this one is trivial)
- Determine whether KB is satisfiable (this one is also trivial)
- Find the lexicographically first satisfying solution (almost as trivial)
- Count the **exact number of satisfying solutions** (needs postprocessing)
- **Choose a random satisfying solution** uniformly over all such solutions
- Given a cost for each individual proposition, find an **optimal cost solution**



Intuitionistic Logic

- A more restrictive form of logic where double negation elimination and the law of excluded middle are not considered valid
- Allows only constructive proofs
- In ordinary propositional logic, you can prove A by separately proving both formulas $B \Rightarrow A$ and $\text{not-}B \Rightarrow A$ and resolving those
- Don't need to know whether B or $\text{not-}B$ is true, the conclusion A follows
- In intuitionistic logic, this move is not allowed



Module 8: Predicate Logic

Propositional Logic Limitations

- In propositional logic, propositions refer to individual facts in the world
- To encode two separate facts "Socrates is mortal" and "Theon is mortal", need to have two separate propositions
- Cannot **quantify** over all relevant propositions to say "All men are mortal"
- Must instead write conjunction over all the relevant facts
- Would like to be able to say in one swoop "**All neighbours of a Sudoku square must have a different value than that square**", instead of writing a zillion separate clauses to say this for all pairs of neighbouring squares



Relations vs. Functions

- It is important to understand the distinction between the logic itself, and the external world that the logical formulas are referring to
- In FOL, world consists of **objects** and **relationships**
- In FOL, logic consists of **terms** and **relations** as **predicates**
- Terms refer to objects, relations refer to relationships, but are not the same
- An n -ary function applies to n objects and **returns another object** in world
- All functions are total, defined over all the objects
- An n -ary relation applies to n objects and **evaluates to truth value** inside logic



Quantifiers

- Two complementary quantifiers **for all** (\forall) and **exists** (\exists)
- Allow us to express general truths over the space of all objects at once
- $\text{not}-(\forall x: P(x))$ is equivalent to $\exists x: \text{not-}P(x)$
- Quantifiers introduce **variables** to formulas
- Variables are **instantiated** to refer to actual objects in the world in the process of determining the truth of the formula

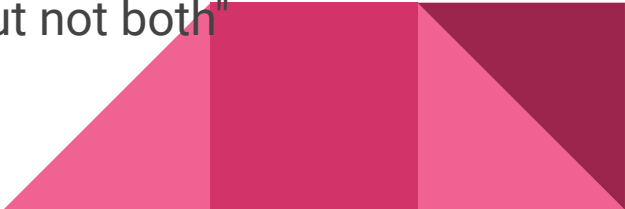


Some Example Sentences


- Assume predicates Likes(x, y) and Happy(x)
- "Everyone likes someone": $\forall x: \exists y: \text{Likes}(x, y)$
- "Someone likes everyone": $\exists x: \forall y: \text{Likes}(x, y)$
- "Someone likes everyone except themselves":
 $\exists x: (\text{not-Likes}(x, x) \wedge \forall y: (x \neq y \Rightarrow \text{Likes}(x, y)))$
- Be careful! What does the following sentence say?
 $\exists x: \forall y: \text{Likes}(x, y) \Rightarrow x \neq y$



Objects Don't Have Types

- First order predicate logic assumes that the world consists of objects that can be in various relationships with each other
 - In many environments, we would like to distinguish objects based on **type**
 - However, all objects in the world are untyped
 - Types can be simulated with **unary predicates**
 - If the world consists of animals and rocks, define appropriate unary predicates $\text{Animal}(x)$ and $\text{Rock}(x)$ to represent these unary relations
 - To prevent other types of objects from existing, need to have **a non-logical axiom** to say "Every object is either animal or rock, but not both"
- 

Prenex Form

- Same as in propositional logic, often handy to convert formulas into equivalent standard forms for reasoning algorithms to better digest
 - **Prenex form**: all quantifiers are universal, and in the front of formula
 - First, standardize variables apart
 - Negating a quantifier flips it to the other quantifier and negates body, allowing us to move negations inside
 - Get rid of existential quantifications with **Skolemization**
 - Move all universal quantifiers to the front, convert to CNF analogous to the way that propositional logic formulas were converted
- 

Handling Object Equality in FOL

- Do the two formulas $P(joe)$ and $Q(joe, moe)$ entail $P(moe)$?
- Clearly not, if P means "is an engineer" and Q means "is brother of"
- Clearly yes, if Q is the hard-coded **object equality** relation =
- Equality predicate $=(x, y)$ is a special case of relations in FOL
- First way is to handle equality directly in the inference engine
- Second way is to write an **axiom schema** that asserts **reflexivity** and substitution for formulas and functions for the equality predicate, after which reasoning proceeds as in ordinary predicate logic
- Axiom schema contains separate formulas for every function and predicate
- Third possible way **paramodulation rule**

Empty Worlds Not Allowed

- Does the sentence $\forall x: P(x)$ entail the sentence $\exists x: P(x)$?
- Does the sentence $Q \vee \exists x: P(x)$ entail the sentence $\exists x: Q \vee P(x)$?
- Only if we assume that at least one object exists in the world!
- In first order predicate logic, world is not allowed to be empty
- To say that world contains exactly one object, use $\exists x: \forall y: x = y$
- To say that world contains exactly two objects, use
 $\exists x: \exists y: x \neq y \wedge \forall z: (z = x \vee z = y)$
- Same idea generalizes to arbitrary number of n objects



Objects Without Names

- The language of predicate logic can directly refer to objects with terms composed of constant literals and function symbols
- Two different terms can refer to the same object
- For example, two terms $2 + 2$ and 4 in the **standard model** of integers
- Formula $2 + 2 = 4$ can be proven as **theorem** from integer arithmetic axioms
- However, **world can also contain objects for which no term refers to!**
- Formulas of form $\exists x: P(x)$ may be true, even though there is no possible term in the language to refer to an object x that makes the formula true

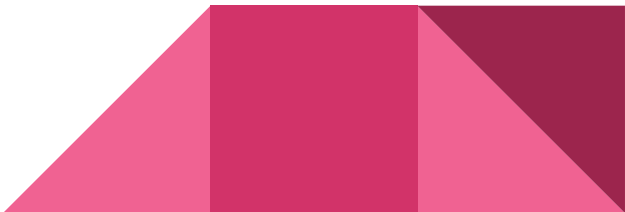


Nonstandard Models

- Trying to capture some world into logical axioms, the resulting set of axioms may also be true in other worlds than the one we "intended" to describe
- Okay as long as axioms allow reasoning in the world that we "intended"
- Axioms may allow nonstandard models that we didn't intend
- Logic doesn't care what we "intend" with finger quotes



Example: Peano Arithmetic

- Example: Peano arithmetic for natural numbers
 - Constant symbol 0, successor function $s(x)$
 - Two axioms $\forall x: s(x) \neq 0$ and $\forall x: \forall y: s(x) = s(y) \Rightarrow x = y$
 - "It's, like, an infinite row of integers 0, $s(0)$, $s(s(0))$, $s(s(s(0)))$..., maaan"
 - Can still fill the space with arbitrary cycles and chains of nameless objects, in addition to the chain that starts at 0
 - Can add axiom $\exists x: x = s(x)$ without creating contradiction
 - False in standard model of integers, true in some others
 - However, the object does not have a name
- 

Removing Literals and Functions from FOL

- Literals and functions turn out to be **syntactic sugar** for relations
- Literals are just special case of **nullary functions** with no parameters
- As we saw in Prolog, every function f that takes n parameters and gives one result is really a relation F of $n + 1$ parameters with axiomatic constraints

$$\forall x: \forall y: \forall z: F(x, y) \wedge F(x, z) \Rightarrow y = z$$

$$\forall x: \exists y: F(x, y)$$

- All function symbols can be mechanistically "uplifted" into relations with mechanistic modifications to formulas that use them

Example of Resolution: Nilsson's Elephants


- Sam, Clyde and Oscar are elephants.
- Sam is pink, Clyde is gray.
- Clyde likes Oscar, and Oscar likes Sam.
- Prove that some gray elephant likes some pink elephant.
- Want to prove: $\exists x: \exists y: \text{Gray}(x) \wedge \text{Pink}(y) \wedge \text{Likes}(x, y)$
- Negating this claim gives us the formula that, conveniently enough, is already in 3-CNF:
- $\forall x: \forall y: \text{not-Gray}(x) \vee \text{Not-Pink}(y) \vee \text{not-Likes}(x, y)$




Nilsson's Elephants Resolution Reasoning

1. $\text{Pink}(\text{sam})$
2. $\text{Gray}(\text{clyde})$
3. $\text{Likes}(\text{clyde}, \text{oscar})$
4. $\text{Pink}(\text{oscar}) \text{ or } \text{Gray}(\text{oscar})$
5. $\text{not-Pink}(\text{oscar}) \text{ or } \text{not-Gray}(\text{oscar})$
6. $\text{Likes}(\text{oscar}, \text{sam})$
7. $\text{not-Gray}(X) \text{ or } \text{not-Pink}(Y) \text{ or } \text{not-Likes}(X, Y)$ (negation of claim we want to prove)
8. $\text{not-Pink}(Y) \text{ or } \text{not-Likes}(\text{clyde}, Y)$ (from 2 and 7 with X/clyde)
9. $\text{not-Pink}(\text{oscar})$ (from 3 and 8 with Y/oscar)
10. $\text{Gray}(\text{oscar})$ (from 4 and 9)
11. $\text{not-Pink}(Y) \text{ or } \text{not-Likes}(\text{oscar}, Y)$ (from 7 and 10 with X/oscar)
12. $\text{not-Pink}(\text{sam})$ (from 6 and 11 with Y/sam)
13. (empty) (from 1 and 12)

Finding A Satisfying Variable Assignment

- For propositional logic formulas given in CNF, **DPLL algorithm** produces a assignment for propositional values that satisfies those formulas
 - For general predicate logic, resolution until empty clause pops up only establishes satisfiability, but does not give variable assignment
 - Easy modification does the trick: instead of adding the negation $\text{not-}\phi$ of the formula ϕ that we wish to prove, add the formula $(\text{not-}\phi \vee \text{Ans}(x))$ instead, where Ans is a new predicate whose parameters are the interesting variables
 - Instead of resolving until empty clause, resolve until Ans becomes singleton
 - Read the satisfying assignment of interesting variables there
- 

Set of Support

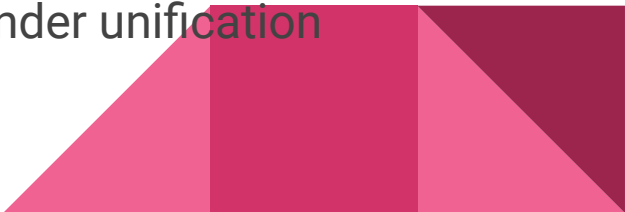
- During resolution reasoning, there are usually far too many possible resolution steps that could legally be taken at any given time
 - In systematic search, such branching makes searching exponentially long
 - **Set of support:** Some of the clauses are marked special "supporting" clauses, initially only the negated conclusion is marked as such
 - In each resolution step, at least one of the clauses must be supporting
 - Resulting clause from resolution step is added to the set of support
 - If the empty clause can theoretically be reached using unlimited resolution, it can also be reached under this restriction with less branching
- 

Unit and Input Resolution


- **Unit Resolution:** in each resolution step, at least one of the two resolved clauses must be a unit clause
- **Input Resolution:** in each resolution step, at least one of the two resolved clauses must be a from the initial set of original clauses
- Neither strategy is **refutation complete** for arbitrary sets of initial clauses, but both are complete for sets of **Horn clauses**
- For a more general set of initial clauses, an empty clause can be reached with unit resolution if and only if it can be reached with input resolution



Linear Resolution

- **Linear resolution** is an important relaxation of input resolution discipline
 - In each resolution step, at least one of the clauses is one of the initial clauses, or an ancestor of the other clause in the proof tree
 - Produces proof trees whose shape is linear
 - Refutation complete for all sets of disjunctive clauses
 - Can be optimized further by using the negation of goal as the top clause, and still remains refutation complete
 - Remains refutation complete under further restriction that ancestor clauses are limited to **merges**, literal collapses to singleton under unification
- 

Situation Calculus

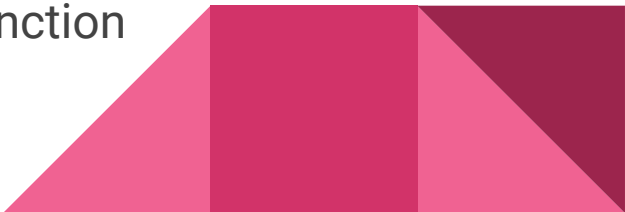
- Technique to model dynamic world using first order predicate logic
 - Fluents are predicates whose truth value depends on current situation in addition to their actual parameters, situations reified as objects in models
 - Laws of environment expressed as non-logical axioms of form $(\text{Pre}(s) \wedge \text{Result}(s, a, s')) \Rightarrow \text{Post}(s')$
 - $\text{Pre}(s)$ expresses the **precondition** that action a is possible in situation s
 - $\text{Result}(s, a, s')$ says that performing action a in the situation s can create the new situation s' (this predicate is not necessarily **deterministic**)
 - $\text{Post}(s')$ describes the new situation s' resulting from action
- 

Frame Problem

- In sense of the fixed background of an animation frame, we need to way to somehow describe all the things that performing the action does not change
- That is, every Foo for which $\text{Foo}(s) \Leftrightarrow \text{Foo}(s')$
- Cannot be expressed directly in first order logic, since we can't quantify over predicates to say "For all predicates other than these..."
- If this problem isn't solved somehow, we can't infer anything about s'
- See the famous "**Yale shooting problem**" for a toy example that is surprisingly difficult to solve using first order logic and situation calculus



More Difficulties With Situation Calculus

- In a complex interconnected environment, situation calculus is difficult
 - **Qualification problem:** formula $\text{Pre}(s)$ becomes horrendously complex
 - In shooting problem, bullet must not be made of bubblegum, etc.
 - **Ramification problem:** formula $\text{Post}(s')$ becomes horrendously complex
 - In addition to their direct consequences, actions can have many indirect consequences to their environment
 - Moving a box from point A to point B also moves everything inside the box and on top of it, including all the dust
 - **Nondeterminism:** predicate $\text{Result}(s, a, s')$ is not a function
- 

Higher-Order Predicate Logics

- Propositional logic is a special case of predicate logic that has no literals or function symbols, and all predicates are nullary
- Propositional logic can be thought of as **zeroth-order predicate logic**
- Seems obvious to now ask if there exist second order and higher logics
- Second order logic allows quantifiers over first order predicates
- For example, can say $\forall P: \exists x: P(x)$ to say that for every unary predicate, there must be one object that makes that predicate true
- Second order logic does not allow computable proof theory





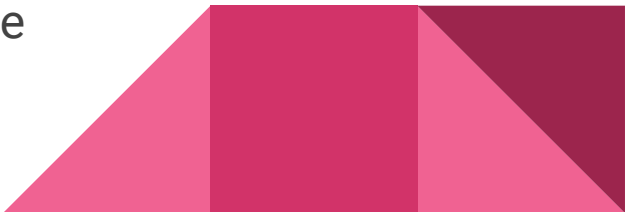
Module 9: Bayesian Probabilities

Kolmogorov Axioms

- Even though probabilities are subjective and reasonable Bayesians can disagree based on their different observations and evidence, any coherent assignment of probabilities must still satisfy the **Kolmogorov Axioms**:
- Every $P(A)$ must be a real number in range $[0, 1]$
- For any A and B , must have $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
- Several other truths of probabilities follow from these two axioms, such as:
- $P(\text{not-}A) = 1 - P(A)$
- $P(A \wedge B) \leq P(A) \leq P(A \vee B)$



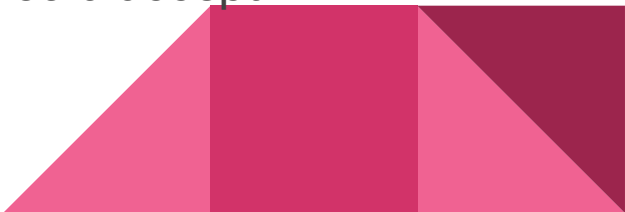
Denying Kolmogorov Axioms

- Suppose Joe Palooka believes $P(A) = 0.4$ and $P(B) = 0.3$, but $P(A \vee B) = 0.8$
 - Kolmogorov axioms entail $P(A \wedge B) = -0.1$, impossible
 - However, as a free man and a singular individualist, Joe doesn't feel the need to follow some axiom system just because it's internally consistent
 - The cosmic "must" that forces everybody to follow these axioms comes from the expectation of "**putting your money where your mouth is**"
 - Talk is cheap, anyone can flap their gums as they wish
 - For probability assignment that violates these axioms, it is always possible to create a **Dutch book** of bets that is guaranteed to lose
- 

Dutch Book Example of Incoherent Beliefs

- We will offer the following bets to Joe in sequence:

Proposition	Joe's belief	Our side of the bet	Joe's stake to ours
A	0.4	A	6 to $4 + \epsilon$
B	0.3	B	7 to $3 + \epsilon$
$A \vee B$	0.8	not- $(A \vee B)$	8 to $2 + \epsilon$

- Each bet has positive expected value to Joe, so he should accept
 - If not, we can ask Joe why not
- 

Dutch Book Is A Guaranteed Loss

- Trap is shut! We can now even let Joe choose truth values of A and B

Our bet	$A \wedge B$	$A \wedge \text{not-}B$	$\text{not-}A \wedge B$	$\text{not-}A \wedge \text{not-}B$
A	-6	-6	4	4
B	-7	3	-7	3
$\text{not-}(A \vee B)$	2	2	2	-8
Net for joe	-11	-1	-1	-1

- Joe is guaranteed to lose, ergo his beliefs can't be coherent

When Two Bayesians Disagree

- Given the same evidence, two honest Bayesians can't "agree to disagree"
- With different evidence, Alice and Bob can have different views on the probability $P(X)$ of some event X
- Suppose Alice believes that $P(X) = 0.2$, whereas Bob believes $P(X) = 0.7$
- Alice can offer Bob a bet where she bets not- X , offering odds up to 3 to 7
- Expected value for Bob is $0.7 * \$3 - 0.3 * \$7 = \$0$, he should accept
- However, even if Bob's assessment is more realistic, Alice can still get lucky
- Even if X turns out to be false, it doesn't follow that Carol who claimed that $P(X) = 0$ would have been the most correct of the trio



Subjectivist vs. Objectivist Probabilities

- **Objectivist view** sees probabilities as real properties of physical things
- **Frequentism** defines $P(A)$ as the limit of proportion that A is observed over repeated similar experiments
- **Reference class problem** of what counts as "similar" for this purpose
- Cannot assign probabilities to **one-time events** and **unique events** ("What is the probability that $P = NP$? Or probability that Goldbach conjecture is true?")
- **Subjectivist probabilities** model the degree of belief of the agent
- How to confuse a Bayesian: Ask him where he gets his priors

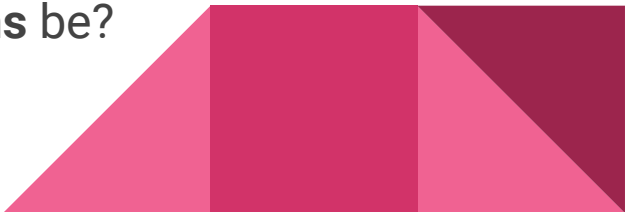


Unknown Weighted Coin

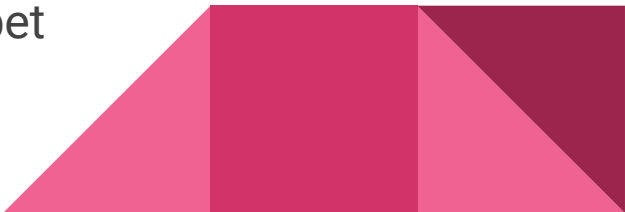
- You are given a coin that is known to be somehow weighted and not fair
- No information provided which way the coin is weighted, and how much
- Question: What is the probability that the next coin flip returns heads?
- Bayesian: We have no reason to assume either side more probable, so $1/2$
- Frequentist: We don't know, except that we know that it can't be $1/2$
- From same premises, both philosophies reach the exact opposite conclusion



The Known and The Unknown

- Famous three-way classification by U.S. SecDef Donald Rumsfeld
 - **Known knowns**: Deterministic known rules (chess, checkers)
 - **Known unknowns**: Nondeterminism that is constrained to follow a known probability distribution (roll of dice, draw of cards)
 - **Unknown unknowns**: Laws of environment themselves are uncertain (sitting down at the poker table, some player suddenly plays the Uno reverse card, reversing the direction of the betting action on that round)
 - Second-order probabilities ("What is the probability that the coin is fair?")
 - To complete the 2-by-2, what would **Unknown knowns** be?
- 

Over-Under Estimation

- Classic riddle: How many piano tuners are there in Chicago?
 - Your **over-under** of an unknown numerical quantity is a value x that makes you indifferent between the bets "True value $< x$ " and "True value $> x$ "
 - Gifted one side of the bet, you would not pay anything to switch sides
 - Can be used to estimate subjective probabilities by adjusting stakes
 - Before you assert that $P(A) = p$, consider the bet "If A , win $(1 - p)\$M$, otherwise lose $p\$M$ ", where $\$M$ is some amount of real money whose loss would not be painful but not catastrophic to you
 - Adjust p until indifferent between both sides of this bet
- 

Estimating Probabilities From Sample Data

- Assume m independent **snapshots** of the world that we wish to model
- We have n propositions that correspond to some facts of the world
- Build the full joint distribution of 2^n possible combinations of probabilities
- Estimate $P(\phi)$ with the formula $N(\phi) / m$
- $N(\phi)$ is the number of worlds where ϕ is true
- Unbiased estimate for $P(\phi)$
- Full joint distribution table doesn't give us the underlying **causality**: you may end up with same full joint when A causes B , and when B causes A



Why We Are Doing All This

- Recall that we are trying to make the agent choose good actions
- If the environment isn't fully observable, some variables are hidden
- These variables affect the expected values of actions
- For example, in heads-up poker, hidden variables are opponent's cards and mindset, evidence variables are our cards, board, and betting action
- Values of agent's possible actions such as "call", "raise" and "fold" depend greatly on these hidden variables
- **Diagnostic reasoning** from evidence variables to hidden variables



Entailed Conditional Probabilities

- **Mutually exclusive** and **fully exhaustive** probabilities always add up to 1
- If you know $P(A | B)$, you also know $P(\text{not-}A | B) = 1 - P(A | B)$
- However, even if you know $P(A | B)$, you don't directly know $P(A | \text{not-}B)$
- Probabilities $P(A | B)$ and $P(A | \text{not-}B)$ can be wildly different
- However, $P(A) = P(A | B) P(B) + P(A | \text{not-}B) P(\text{not-}B)$
- Once the prior probabilities $P(A)$ and $P(B)$ are known, knowing $P(A | B)$ allows us to compute $P(A | \text{not-}B)$, and vice versa



Attraction and Repellence

- Evidence E is said to **attract** the event A if $P(A | E) > P(A)$
- Evidence E is said repel the event A if $P(A | E) < P(A)$
- E attracts A if and only if E repels not- A
- Attraction is symmetric: A attracts E if and only if E attracts A
- A neither attracts E nor E attracts A if and only if A and E are independent
- A and E are mutually attractive if and only if $P(A | E) > P(A | \text{not-}E)$



Combining Multiple Pieces of Evidence

- Even if both evidences E and F separately attract A so that both $P(A | E) > P(A)$ and $P(A | F) > P(A)$, it doesn't necessarily follow that $P(A | E \wedge F) > P(A)$
- Even if F attracts E and E attracts A so that $P(E | F) > P(E)$ and $P(A | E) > P(A)$, it doesn't necessarily follow that $P(F | A) > P(F)$
- Can you think up real world interpretations for A , E and F to illustrate these counterintuitive claims about conditional probabilities?



Marginalization

- We sometimes have some prior probability $P(A)$ that we wish to compute
- Such a prior probability might appear as a part of some other calculation
- **Marginalization:** $P(A) = P(A \wedge B) + P(A \wedge \text{not-}B)$
- B can be literally any formula whatsoever, allowing us to choose it tactically
- $P(A \wedge B)$ can then be rewritten as $P(B) P(A | B)$
- If $P(B)$ is not known, solve it the same way with some tactically chosen C
- $P(B) = P(B \wedge C) + P(B \wedge \text{not-}C)$



Causal and Statistical Independence

- Events A and B are **independent** if $P(A) = P(A | B)$ and $P(B) = P(B | A)$
- Alternative formulation for independence is $P(A \wedge B) = P(A) P(B)$
- Even if A and B have no **causal connection** in the laws of nature of the underlying world, A and B are not necessarily **statistically independent**
- There could be some underlying hidden cause C that causes both A and B
- By diagnostic reasoning, $P(C) \neq P(C | A)$, therefore $P(B | A) \neq P(B)$




Causality Despite Independence

- Even if A and B are causally connected, can still be $P(A \wedge B) = P(A) P(B)$
- Statistical independence does not rule out causal relationship per se
- Consider three bits A , B and C , so that A and B are mutually independent random coin flips, whereas C is given by the **exclusive or** of A and B
- $P(C) = P(C | A)$ and $P(C) = P(C | B)$
- C is independent of both A and B separately, despite being caused by them
- However, C is not independent of $A \wedge B$
- (Exercise for the reader: is C independent of $A \vee B$?)



Two Classic Bad Jokes From A Simpler Time

- Whenever evidence E is available, $P(A)$ is meaningless next to $P(A | E)$
 - In a bad old joke from a very different era, a guy was afraid to fly because there could be a bomb in the plane with small probability $P(B)$
 - To allay his fears, he took his own bomb with him to the plane, reasoning that the squared probability $P(B)^2$ of two bombs on the plane is practically zero
 - In another bad joke from the same era, an old man was close to death
 - To extend his life, he intentionally got infected with HIV, since he had read that with modern medicine, an average HIV patient lives over a decade
 - Without hand waving, what was the mistake in each case?
- 

Causal And Diagnostic Reasoning

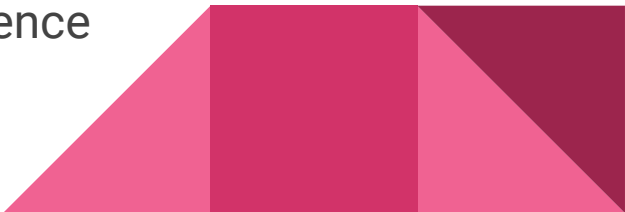
- Causal connections between propositions A and B may be known from the known laws of nature of the world from which A and B comes from
- For example, fire causes smoke, but smoke doesn't cause fire
- If B is observable, **causal reasoning** uses conditional probability $P(A | B)$
- If A is observable, **diagnostic reasoning** via **Bayes Theorem**

$$P(B | A) = P(A | B) P(A) / P(B)$$

- "Google uses Bayes theorem like Microsoft uses if-else"



Base Rate Fallacy

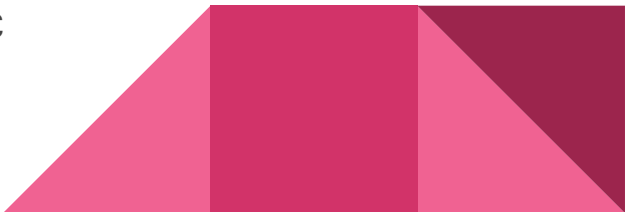
- Bayes rule says that $P(A | B) = P(B | A) P(B) / P(A)$
 - Common fallacy to assume that $P(A | B) = P(B | A)$
 - Effectively asserting that **base rate** $P(B) / P(A) = 1$
 - Base rate describes how common events A and B are relative to each other
 - Base rate can be anything from 0 to infinity, depending on A and B
 - Often a good way to lie with statistics, when trying to convince innumerate readers, and score all kinds of cheap rhetorical points
 - Also known as "**Prosecutor's Fallacy**" or "**Defender's Fallacy**", depending on which way the rhetoric is supposed to sway the audience
- 

Extraordinary Evidence

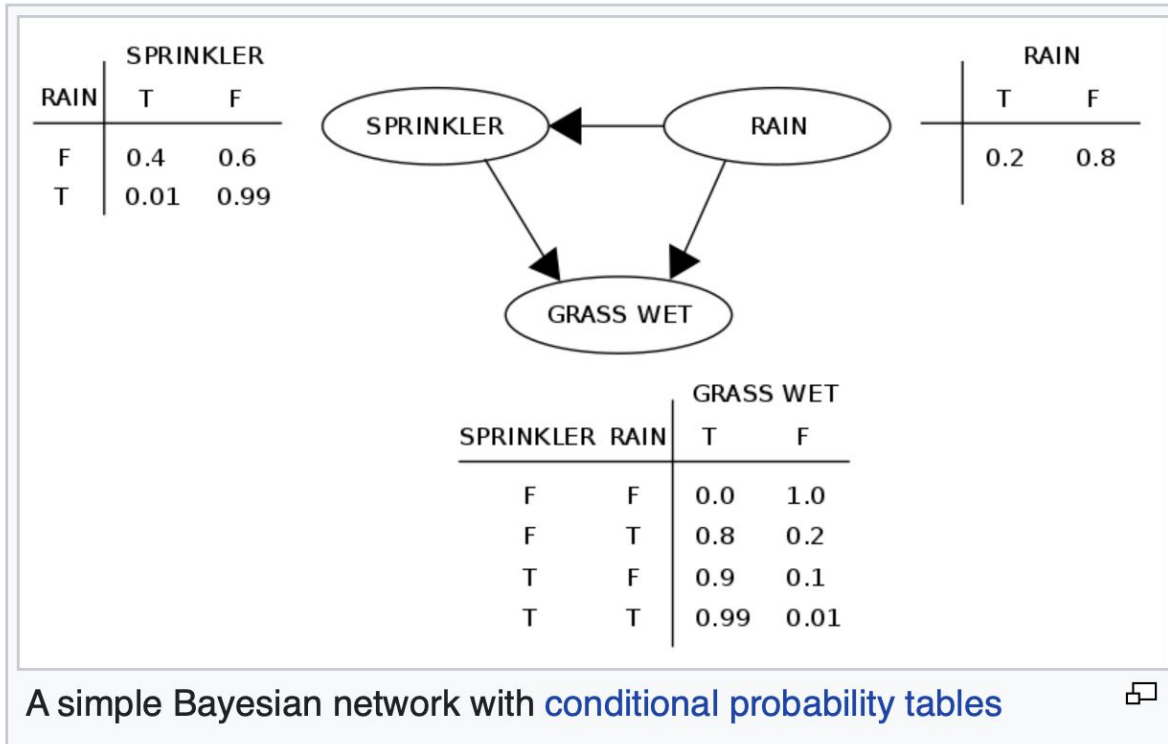
- If $P(A)$ is in $(0, 1)$, new evidence can always theoretically swing it arbitrarily so that $P(A | E)$ can be any value in $[0, 1]$, no matter what $P(A)$ was before E
- If this is the case, what use is knowing that, say, $P(A) = 0.01$, if some evidence E could theoretically arrive that makes $P(A | E) = 0.99$?
- How do we know that we have collected enough evidence to act?
- It can be proven that probability $P(E)$ of such evidence must itself be small
- Extraordinary changes of probability estimations can only be caused by pieces of evidence that are themselves extraordinary unlikely to appear



Cromwell's Rule

- If $P(A) = 0$, then $P(A | E) = 0$, and if $P(A) = 1$, then $P(A | E) = 1$
 - Once some probability has been determined to be 0 or 1, no additional evidence can change that, even if Good Lord himself told us otherwise
 - When solving real problems that someone would actually pay us to solve, this is a pretty heavy position to make in our fallible knowledge
 - Other people and sources of info can always be lying or honestly mistaken
 - If you really believe that $P(A) = 0$, are you ready to accept the bet where your head is chopped off in a guillotine if A turns out to be true?
 - Treat probabilities 0 and 1 as if they were ϵ and $1 - \epsilon$
- 

Example Bayes Network



Example Bayes Network 2

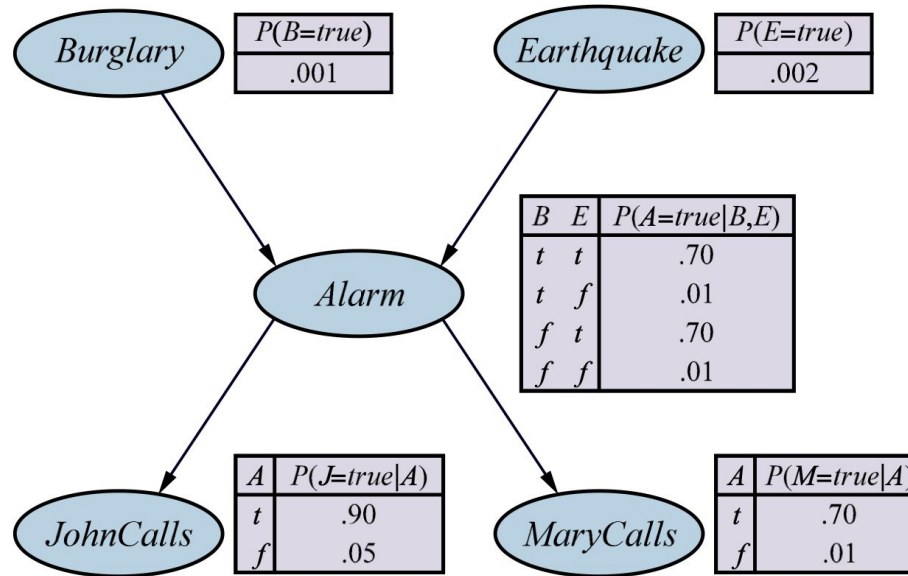
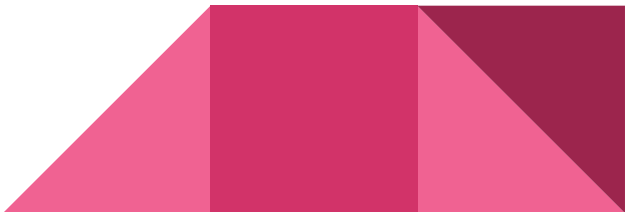


Figure 13.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters *B*, *E*, *A*, *J*, and *M* stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

Building a Bayes Network

- Consider model of world with n propositional variables
 - In principle, could build network for any of the $n!$ possible variable orderings
 - In practice, sort variables in order of causality
 - This tends to keep number of parents smaller for each node, and also make conditional probability values easier to estimate
 - To add a new variable X , find a good subset S of all previous nodes E so that $P(X | E) = P(X | S)$, making X is conditionally independent of $E - S$ given S
 - Important machine learning problem to derive a good Bayes network from the given set of training samples
- 

Variable Ordering Can Have a Big Effect

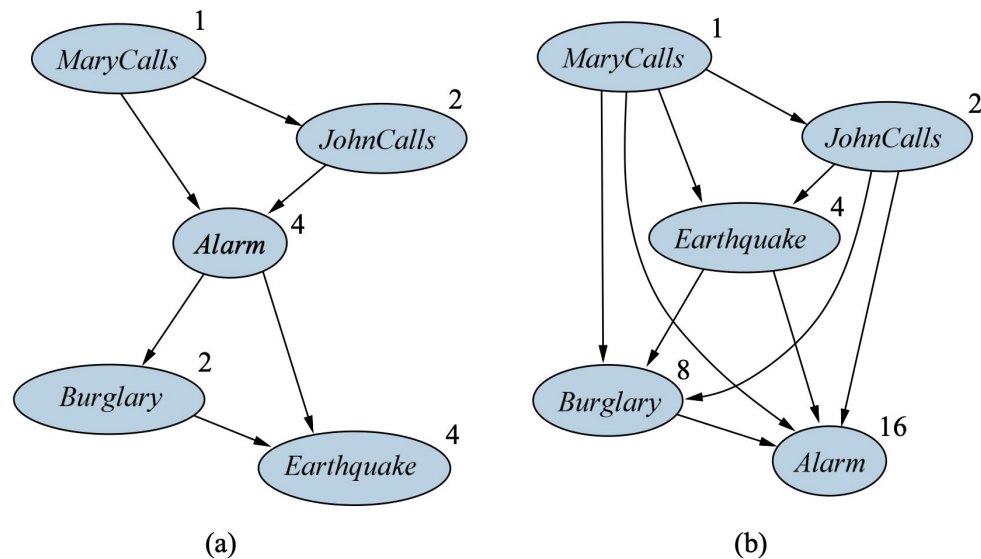


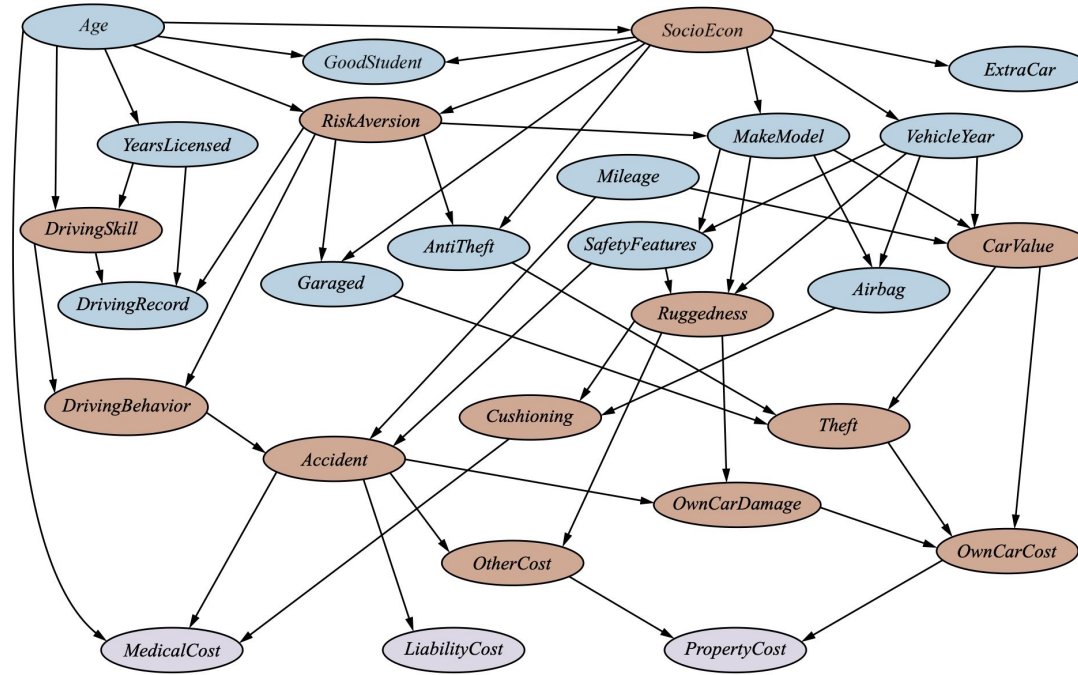
Figure 13.3 Network structure and number of parameters depends on order of introduction. (a) The structure obtained with ordering M, J, A, B, E . (b) The structure obtained with M, J, E, B, A . Each node is annotated with the number of parameters required; 13 in all for (a) and 31 for (b). In Figure ??, only 10 parameters were required.

Randomly Sampling A Given Bayes Network

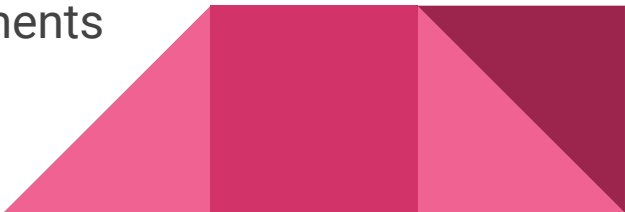
- Once you have constructed a Bayes network, can quickly compute the exact value of any individual entry of full joint distribution
- Start with initial probability 1
- Loop through nodes in some topological sorted order
- For each node X , multiply current probability with $P(X \mid \text{Parents}(X))$
- Resulting probability in the end is the exact entry in the full joint
- Similarly easy to generate a random world from the probability distribution
- Loop through nodes, make each X true with probability $P(X \mid \text{Parents}(X))$




More Complex Bayes Network



Markov Blanket

- Each node of a Bayes network is conditionally independent of its non-descendants, given all its parents
 - Parents shield the node from its earlier ancestors in the calculation
 - Each node of a Bayes network is conditionally independent of all other nodes, given its **Markov blanket** of its **parents, children and children's other parents**
 - These other parents determine the way that known values of the child nodes affect the probability of the node in question
 - Bayesian networks on graphs $A \rightarrow B \rightarrow C$ and $C \rightarrow B \rightarrow A$ are equivalent in that impose the same conditional independence requirements
- 

Explaining Away

- Consider unlikely symptom E so that $P(E)$ is small
 - Assume two possible separate causes A and B for E
 - For example, E is burglar alarm, A is burglar, B is cat playing with alarm
 - Both $P(E | A)$ and $P(E | B)$ are large
 - Since $P(E)$ is small, also $P(A)$ and $P(B)$ must be small *a priori*
 - However, $P(A | E)$ is significantly larger than $P(A)$
 - However, $P(A | E \wedge B)$ is again small, far closer to $P(A)$ without evidence
 - Known cause B for E makes the competing cause A less likely
 - Important if A is serious whereas B is harmless
- 

Analyzing Causality With Do-Operator

- What if we want to find out the underlying causality?
- Joint distribution itself would allow either true direction causality
- Can use **do-operator** and **do-calculus** devised by Judea Pearl and friends
- What effect does forcing the value of some variable to a fixed value have to the probabilities of other variables in the network?
- Forcing the value of A to be true is denoted by $do(A = \text{true})$
- Variable A no longer produces information about its ancestors
- Effectively eliminates the incoming arrows from variable A



Example Of Using The Do-Operator

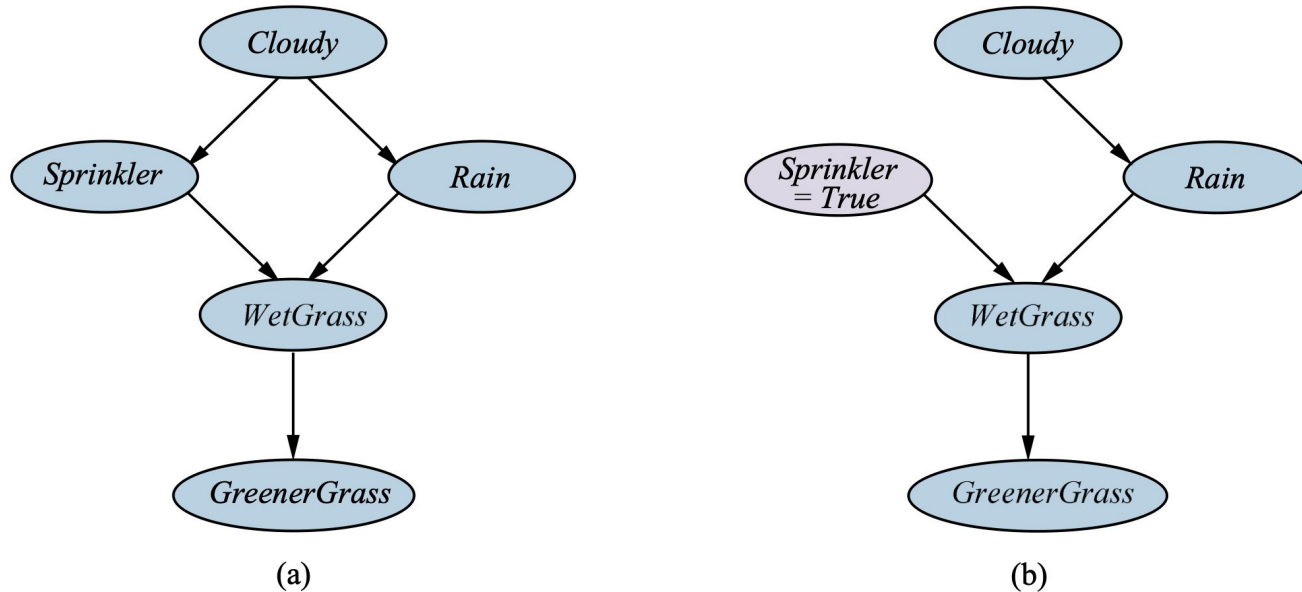

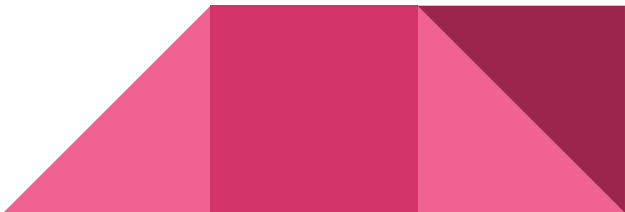


Figure 13.23 (a) A causal Bayesian network representing cause–effect relations among five variables. (b) The network after performing the action “turn *Sprinkler* on.”

Inferring Causality

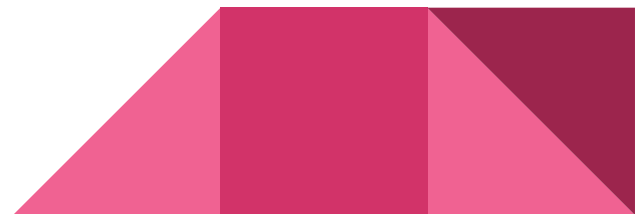
- A variable that is forced to be true is effectively "caused" by our hand
 - Note that $P(A \mid B)$ is generally not the same as $P(A \mid do(B))$
 - If A and B have common ancestors, $do(B)$ does not affect the diagnostic probabilities of these ancestors, whereas B becoming "naturally" true via the influence of these ancestors provides information about these ancestors
 - In brief, causality is everything that makes $P(A \mid B)$ and $P(A \mid do(B))$ different!
 - Can study and quantify effects of **forced interventions**
 - To learn more about reasoning about causality, interested students can consult the highly readable "*The Book of Why*" by Judea Pearl
- 

Bayesian Updating Example

- (From *Think Bayes 2nd Ed*, Allen Downey)
 - You have a set of dice from *Dungeons and Dragons*
 - These dice have 4, 6, 8, 12 and 20 sides, respectively
 - You choose one die at random and roll it
 - Suppose you get a six
 - What is the probability for each die that it was the one that you rolled?
 - Obviously not the 4-sided die, but couldn't all other dice produce a six?
 - Yes they can, but not with the same probability
 - Imagine if there was also a trillion-sided die
- 

Bayesian Updating Example (cont.)

H	$P(H)$	$P(E H)$	$P(H E) = P(E H) P(H) / P(D)$
4	1/5	0	0.0
6	1/5	1/6	0.392
8	1/5	1/8	0.294
12	1/5	1/12	0.196
20	1/5	1/20	0.117



Bayesian Updating Example (cont.)

- Suppose we roll several times and get results $E = [6, 7, 7, 5, 4]$
- After the updating calculations, the updated conditional probabilities are

H	$P(H E) = P(E H) P(H) / P(D)$
4	0.0
6	0.0
8	0.943
12	0.055
20	0.001



Doomsday Argument

- Famous thought experiment based on Bayesian updating of priors
- In a finite universe, there will be some total number of humans N who will ever have existed before the universe reaches heat death
- What is your best over/under estimate for the value of N ?
- Assume that all humans are numbered 1, 2, 3, ... as they are born
- You are currently human number about 60 billion plus chump change
- Use Bayesian updating to compare hypotheses for various values of N
- Doomsday for humanity is probably relatively near!



Applying Hypotheses to Make Predictions

- Having calculated the probabilities for these competing mutually exclusive hypotheses, how do we use these probabilities to make a prediction?
- For example, what is the probability that the next dice roll will show 14?
- Easy way: use the hypothesis whose probability is the highest
- Can be wildly off, if several hypotheses are equally likely
- Better way: add up all predictions weighted by hypothesis probabilities
- $P(X) = P(X | H_1) P(H_1) + \dots + P(X | H_n) P(H_n)$
- Gibbs sampling: choose a random hypothesis H_i weighted by $P(H_i)$
- Using only this H_i , error probability still surprisingly small



When Only Unlikely Hypotheses Survive

- The initial set of hypotheses must cover all possible situations that can occur
- Technique works only when we are dealing with "Known unknowns"
- However, sometimes the evidence E is so surprising that only hypotheses that were highly unlikely a priori survive that evidence
- Even though the prior probabilities of the surviving hypotheses were, say, something like 10^{-10} and 10^{-20} , the posterior probability of one being $1 - 10^{-10}$ and the other being 10^{-10} leaves little doubt of which one to bet on
- Once you have eliminated the impossible, what remains is the truth





From *Casino* (1995 film)

Ace Rothstein : Four reels, sevens across on three \$15,000 jackpots. Do you have any idea what the odds are?

Don Ward : Shoot, it's gotta be in the millions, maybe more.

Ace Rothstein : Three fuckin' jackpots in 20 minutes? Why didn't you pull the machines? Why didn't you call me?

Don Ward : Well, it happened so quick, 3 guys won; I didn't have a chance...

Ace Rothstein : *[interrupts]* You didn't see the scam? You didn't see what was going on?

Don Ward : Well, there's no way to determine that...

Ace Rothstein : Yes there is! An infallible way, they won!



Module 10: Simple Decisions

Turning Ordinal Preferences into Cardinal

- For three outcomes A, B and C, some agent has preferences $A \succ B \succ C$
- This ordering doesn't say anything about relative preferences
- For example, consider $A = \$1000$, $B = \$999$, $C = \$0$
- For example, consider $A = \$1000$, $B = \$1$, $C = \$0$
- Or even weirder, $A = \text{"get a duck"}$, $B = \text{"get a chicken"}$, $C = \text{"get kick in the butt"}$
- To measure the relative cardinalities of A, B and C, let's use over-under
- Find the probability p so that $[p, A; (1-p), C] \sim B$




Utility of Money Is Not Linear

- For being such a good student, a friendly billionaire Tony Stark enters the lecture room to offer you a reward, the choice of two lotteries:

A: [0.8, \$4000; 0.2, \$0]

B: [1.0, \$3000]

- Noting that $EV(A) = \$3200$ and $EV(B) = \$3000$, which one will you choose?
 - What if this lottery were offered to you a hundred times in sequence?
 - What if you were a professional option trader dealing with hundreds of millions of dollars every day?
- 

Convex Utility of Money

- Utility of money is not linear, but follows a convex logistic curve
- Even though $EV(A) > EV(B)$, we still have $EU(A) < EU(B)$

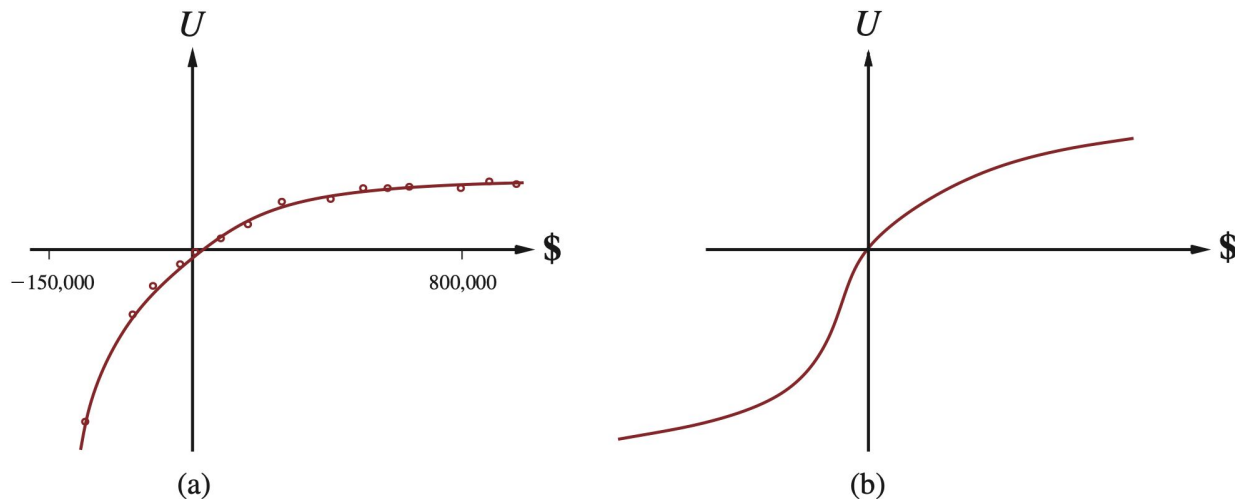


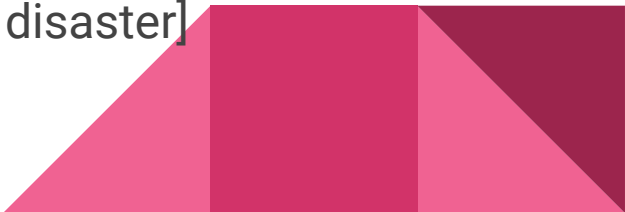
Figure 16.2 The utility of money. (a) Empirical data for Mr. Beard over a limited range. (b) A typical curve for the full range.

Principle of Maximum Misery

- Most material on convex utility of money looks at only the positive side
- This coin has a flip side on the negative side of losses
- On the negative side of losses, the utility function is concave, so rational agents suddenly become risk-seeking
- Once the losses are large enough, larger losses become meaningless
- Agent starts chasing losses with desperate bets
- Once you have lost all your life savings, a double-or-nothing bet with 40% chance to win and 60% chance to lose starts looking temptingly rational



Value of Certainty as Insurance Premium

- Assume that for Joe, $U(\$2500)$ is same as $EU([0.8, \$4000; 0.2, \$0])$
 - \$2500 is called Joe's certainty equivalent for lottery $[0.8, \$4000; 0.2, \$0]$
 - This difference $\$3200 - \$2500 = \$700$ is Joe's **insurance premium**
 - Given the lottery $[0.8, \$4000; 0.2, \$0]$, Joe would be willing to trade this uncertainty for the guaranteed certainty of lump sum of at least \$2500
 - For insurance company, money on this scale has essentially linear utility
 - Trade has positive EV for insurance company but $EV - \$700$ for Joe
 - Yet trade is positive EU for both sides, so it can happen
 - Especially important for lotteries $[1-\epsilon: \text{status quo}, \epsilon: \text{disaster}]$
- 

Allais Paradox

- Recall the previous lotteries A: [0.8, \$4000; 0.2, \$0] and B: [1.0, \$3000]
- Most people rationally choose lottery A
- Consider lotteries C: [0.2, \$4000, 0.8, \$0] and D: [0.25, \$3000, 0.75, \$0]
- No certain money available, most likely outcome getting nothing
- $EV(C) = \$800$, $EV(D) = \$750$
- Most people rationally choose C over D
- However, what happens if we compare these decisions using utility theory?



Allais Paradox Contradiction

- $B \succ A$, so $0.8U(\$4000) + 0.2U(\$0) < U(\$3000)$
- $C \succ D$, so $0.2U(\$4000) + 0.8U(\$0) > 0.25U(\$3000) + 0.75U(\$0)$
- Two inequalities with three unknowns, not enough information
- Since utility functions can be shifted without affect the rational action, let us fix $U(\$0) = 0$ (it is important to note that $U(\$0) = 0$ isn't the case in general)
- Simplifying previous equations this way, we get
$$0.8U(\$4000) < U(\$3000)$$
$$0.2U(\$4000) > 0.25U(\$3000)$$
- Multiply both sides in second inequality by 4 for contradiction!

Additional Factors of Rationality

- When choosing an uncertain action, resulting loss has an additional sting compared to certainty that was available as an option in the beginning
- This especially when other people are watching us from the peanut gallery
- **Anchoring** to baseline: a gambler who first wins \$1000 and then loses it will end up feeling worse than a gambler who first loses \$1000 and then wins it back, despite the fact that both gamblers started and ended exact same
- Human tendency to add up the gains of alternative actions to compare with actual outcome, despite these actions being mutually exclusive



Multiagent Decisions

- Example: three friends together rent a three-bedroom apartment, and all three value the three bedrooms A, B and C as $A > B > C$
- How to determine fairly which friend gets which bedroom?
- Since we can't pop open their heads to measure their hedonic preferences, must convert their preferences into something that can be measured
- Simple solution: Auction of the bedrooms as share of rent
- Use price signals to measure preferences

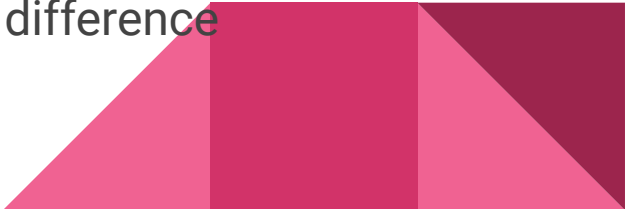


Multiagent Decisions Using Indifference Principle

- Another solution is based on the observation that the bedroom A gives most utility for the friend who is willing to risk most to get it
- Have each friend choose probability p that would make them indifferent between $[p, A: 1 - p, C]$ and B
- Starting from friend with lowest p , flip a p -weighted coin
- That friend gets either the bedroom A or bedroom C based on that coin
- The other two friends divide the other two rooms the same way
- Example of mechanism design that enforces honesty: no friend can gain by lying about their preferences about p



Vickrey Auction

- In an ordinary sealed-bid auction, an agent with additional knowledge about the true value of an underappreciated item may gain by lowballing its bid
 - Simple mechanism design solution of **Vickrey Auction**: the highest bid wins, but the winner has to pay only the second highest bid!
 - Assume agent truly values the item at $\$P$, but bids $\$B$ so that $B < P$
 - Let the winning bid equal $\$W$: three possible cases
 - If $P < W$, also $B < W$, so bidding less wouldn't have made a difference
 - If $P > W$ but $B < W$, agent missed opportunity to pay $\$W$ for value of $\$P$
 - If $B > W$, bidding $\$B$ is same as bidding $\$P$, makes no difference
- 

Games With Simultaneous Moves

- Previous lecture on game playing assumed alternating moves
- When two players have to make their moves **simultaneously**, game is no longer a tree, but a **payoff matrix**
- Rows are moves of first player, columns are moves of second player
- Entries of payoff matrix give the outcomes for both players
- In zero-sum game sufficient to give outcome for row player
- If the game consists of multiple moves, evaluate subgames recursively

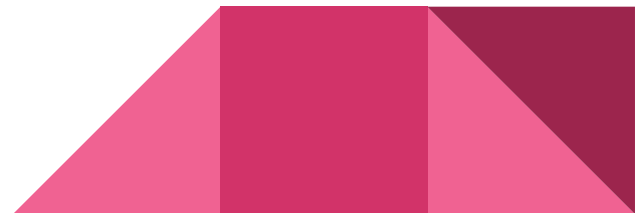


Example Game

- Consider the zero-sum game with the following payoffs for row player:

	B1	B2
A1	-2	5
A2	4	2

- How to solve the Nash equilibrium?
- Deterministic strategies can't be Nash equilibria: must randomize moves



Principle of Indifference Gives The Answer

- Somewhat counterintuitive solution is to play so that you are **indifferent** to opponent's strategy, so they can choose their moves any way they wish
- This is how Nash equilibria work: the opponent can't improve their lot by deviating from his own Nash equilibrium strategy
- Player A should choose the randomizing strategy so that $E(B1) = E(B2)$
- Player A chooses move A1 with probability p , and move A2 with $1 - p$
- $E(B1) = 2p - 4(1 - p) = 6p - 4$, and $E(B2) = -5p + 2(1 - p) = -7p + 2$
- Setting these two values to be equal we get $p = 6/13$



Same Analysis for Player B

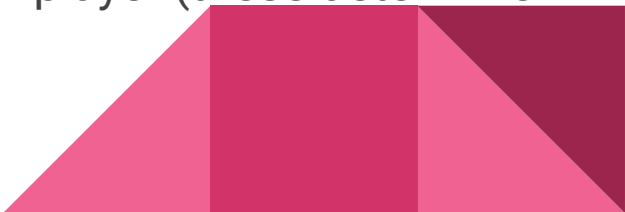
- Similarly, player B chooses move B1 with the probability q , and the move B2 with the probability $1 - q$
- $E(A1) = -2q + 5(1 - q) = -7q + 5$
- $E(A2) = 4q - 2(1 - q) = 6q - 2$
- Setting these two values equal we get $q = 7/13$
- It's just a coincidence that here both players ended with same two probabilities for their moves, this doesn't need to happen in general



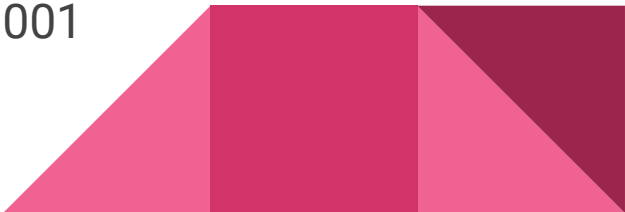
Game With Three Possible Moves

- **Rock-Paper-Scissors** so that winning with rock pays \$2, other wins pay \$1

	Rock	Paper	Scissors
Rock	0	-1	+2
Paper	+1	0	-1
Scissors	-2	+1	0

- Need to solve equations for two probabilities for each player (these determine the probability of the third move)
- 

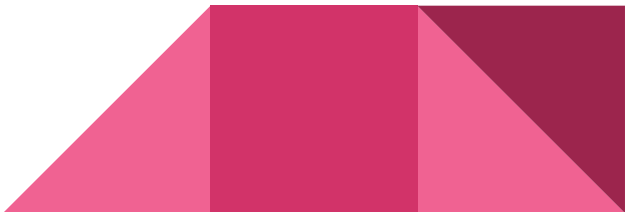
Solution To Modified Rock-Paper-Scissors

- Solution to modified Rock-Paper-Scissors is a bit counterintuitive
 - If winning with rock pays extra well, one might assume that we would play rock more often than in ordinary rock-paper-scissors
 - However, opponent can adapt by playing paper more often
 - Nash equilibrium solution actually plays rock and scissors with probability $1/4$, and plays paper with probability $1/2$
 - If the payoff with rock were \$1000 versus \$1 for winning with paper or scissors, the Nash equilibrium solution would play paper with probability $999/1001$, and rock and scissors with probability $1/1001$
- 

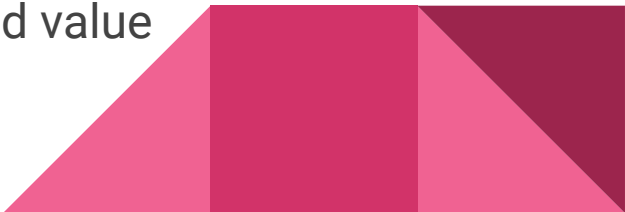
Games With Saddle Point

- Consider the game with the following payoff matrix with a **saddle point** that is simultaneously its row minimum and column maximum:

	B1	B2
A1	0	1
A2	-2	0

- Both players will choose the saddle point move all the time
 - Game becomes deterministic
- 


Indifference Principle With More Complex Games

- Playing against indifferent nature, choose your action to maximize its EV
 - When you follow your Bayes equilibrium strategy, you will choose your action probabilities so that all actions available to your opponent look equally good to him, based on what he knows inside the game
 - **Range:** Create a probability model of states of environment and opponent
 - **Equity:** Give each action a probability, and compute the values of opponent's actions (not your actions!) based on these probabilities
 - **Equalize:** Solve your action probabilities not to maximize your own value, but that every action for opponent has the same expected value
- 

Prisoner's Dilemma

- The famous **Prisoner's Dilemma**, the tragic condition of all humanity appearing in many guises and forms:

<i>[row outcome, col outcome]</i>	Cooperate	Defect
Cooperate	[+2, +2]	[-10, +10]
Defect	[+10, -10]	[0, 0]


- Both players have a dominating play "Defect", never making them worse off than the cooperating play, regardless of what the opponent does
- 

Aspects of Prisoner's Dilemma

- In real world situations, needs **metagame enforcement** mechanisms
- Since it's always rational to defect in a single-shot game, **repeated play** for a known fixed number of matches doesn't help (proof by induction)
- More interesting when played for an unknown number of repeated matches
- Famous contest between different possible strategies
- Winner the simple **tit-for-tat strategy**: co-operate first, then do whatever your opponent did in the previous round
- Another famous type of player is the **grim trigger** that never forgives




Other Famous Games

- The nature of the game depends on the **shape of values in the payoff matrix**, not the actual numerical values
 - Even for 2-by-2 games there is interesting variety
 - For four payoffs A, B, C and D, there are 16 possible orderings of values
 - These games have been given catchy intuitive names such as "**Battle of the sexes**", "**Matching pennies**" and "**Stag hunt**"
 - Can be used to model phenomena that we don't call "games" in the everyday parlance meaning of this term
 - Underlying reality has a shape that determines strategy, no matter what words we happen to use to talk about it
- 



Module 11: Supervised Learning

Supervised Learning

- A benevolent **instructor** provides a set of **training examples**
 - Nothing can be inferred from the chosen examples or their order
 - If the **environment** acts as the teacher, it produces examples "**naturally**"
 - Each training example is a pair of **input** and **expected result**
 - Learner creates a model to fit not just these training examples, but **generalize** so that it produces correct results for previously unseen inputs significantly better than flipping a coin
 - No need to model the actual structure and laws of the world, as long as the model is faithful enough to produce correct predictions
- 



Eliezer Yudkowsky Retweeted



Christoph Molnar @ChristophMolnar · Jan 13



You can't "train" a model.

The model always exists.

It existed before you were born and it exists after your death.

You can only find the model.

"Training" is just your way of looking for the model's location in the infinite hypothesis space and binding its essence to silicon



136



251



1,867



[Show this thread](#)

Necessary Assumptions

- For supervised learning to be possible to begin with, the environment that produces training samples must be governed by **strict natural laws** that can be encoded with at most as many bits than there are in training samples
- Combinatorially, this rules out most of the possible worlds, leaving only an infinitesimally small fraction of those possible worlds as candidates
- Most things that would theoretically be possible, must be actually impossible in the environment whose behaviour we are trying to model



Supervised Learning Algorithms

- Decision trees
- Perceptrons
- Neural networks (feedforward, recursive, convolutional, deep learning, etc.)
- Support vector machines
- k -nearest neighbours
- (and many others...)
- All of these are basically one-liners in *scikits-learn* or *Mathematica*



Confusion Matrix Example

- Describes the false and true positives and negatives of a classifier
- Example with 100 positive and 100 negative training examples

	Predict Positive	Predict Negative
True Positive	97	3
True Negative	8	92

- Especially important if true positives or true negatives are more numerous so that even a stopped clock is right 99% of the time

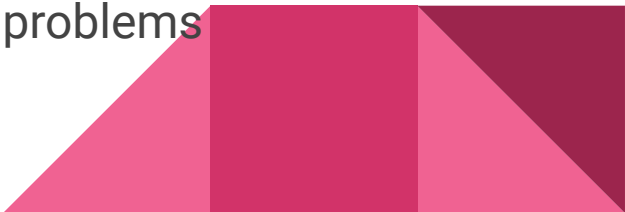


Occam's Razor

- **Occam's Razor** is the observation that from multiple hypothesis that fit the same training examples, the finger-quotes "simplest" one is the best
- Do not multiply entities and assumptions needlessly, but only when the training data doesn't fit your current assumptions
- However, what constitutes the "simplest" model depends on the assumptions that you make about the world that produces the discrete training examples
- Example: fitting a polynomial into the given set of data points, versus fitting a sum of sine curves (Fourier transform) into those data points



No Free Lunch Theorem for Classifiers

- Given the same training examples, different learning algorithms produce different models to fit those training examples
 - **No Free Lunch Theorem** says that over all possible worlds, every learning algorithm is equally good on average!
 - Given the same training data with inputs X_i and their expected answers, whenever two models disagree on some previous unseen input, one wins in exactly half of the possible worlds, the other wins in the other half
 - If an algorithm performs well in some class of problems, then it necessarily must pay for this by degraded performance on other problems
- 

Bias

- The same training data that be generalized in infinitely many different ways
- Given the same training data, different learning algorithms produce different models that will disagree on unseen inputs
- The tendency to choose one among all possible fitting answers is called the **bias** of that learning algorithm
- Given the same information, two entities produce a different hypothesis and explanation depending on their background assumptions
- Training data itself cannot be used to distinguish who is "correct"




Variance


- Suppose we split the training data in two, and train two separate classifiers independently using the same learning algorithm
- The **variance** of the learning algorithm is the measure of how often these two classifiers disagree with each other on their classifications on unseen inputs
- Measures how sensitive the learning algorithm is for quirks of training data
- Bias and variance in learning algorithms are on a continuum akin to Scylla and Charybdis: attempt to decrease one automatically increases the other




Two Extreme Classifier Algorithms

- Consider learning algorithm we can call "**Closed World Assumption**"
 - Tabulate training data, but for all other inputs, always return false
 - By No Free Lunch theorem, as good as any other learning algorithm!
 - CWA algorithm has maximum bias, but zero variance
 - Consider then another learning algorithm we can call "**Coin Flip**" that tabulates training data, and for unseen inputs, just flips a coin (w/ caching)
 - Again, as good as any other learning algorithm over all possible worlds!
 - CF algorithm has maximum variance, but minimum bias
 - All classifier algorithms fall somewhere between these two
- 

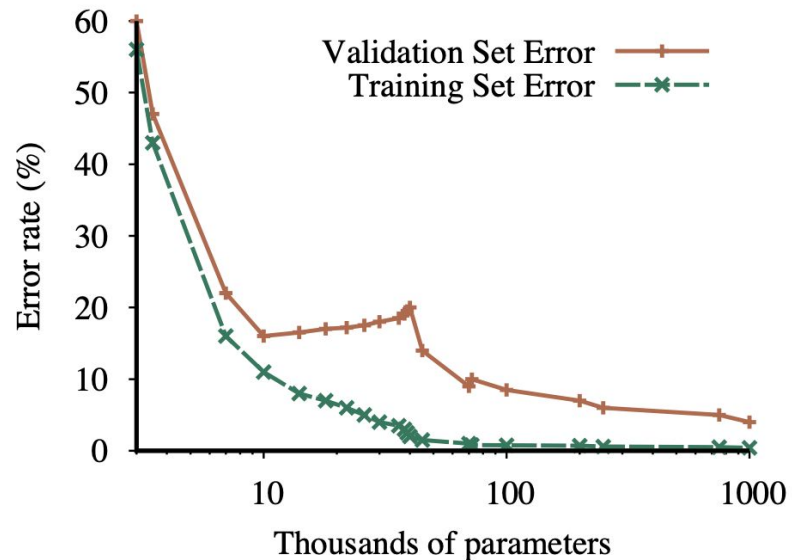
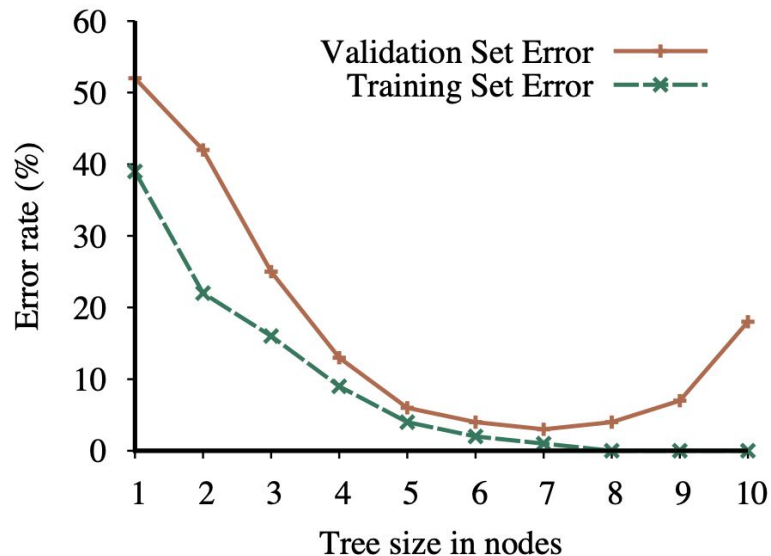
Hyperparameters

- The learning algorithm itself adjusts its internal model based on training data
 - For example, weights of a neural network
 - These variables that the algorithm itself adjusts are its parameters
 - **Hyperparameters** are parameters that control the learning algorithm itself
 - For example, the number of nodes in the neural network, or the number of nodes allowed when building a decision tree
 - Hyperparameter optimization is a dark art in machine learning
 - Can use **cross-validation** to increase hyperparameters until performance on validation set decreases
- 

Training, Validation and Test Data

- Examples given by the instructor are divided in three groups: **training data**, **validation data**, and **test data**
 - Training data is used to create and polish the model
 - Validation data is used to prevent overfitting of the model
 - As we saw from previous two extreme learning algorithms, fitting the training data 100% means nothing, since the learning algorithm is supposed to generalize the underlying patterns in reality the training data comes from
 - As soon as model becomes worse on validation data, stop fitting it
 - Test data is the "final exam" that measures how well model fits reality
- 

Validation Set Tells Us When To Stop



Ensemble Learning

- Instead of trying to create one model, create a bunch of simpler models that individually are not as powerful as the large model
- To classify input x , give it to all models and use the majority vote
- These simpler models may even be constructed with different algorithms
- Construction should make the simpler models independent of each other, as there is no point having identical simpler models all return the same answers
- When tallying votes for the result, each individual model can be given the **weight** based on how well it operated on the training data

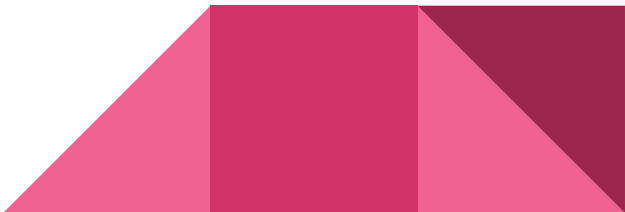


Random Forests

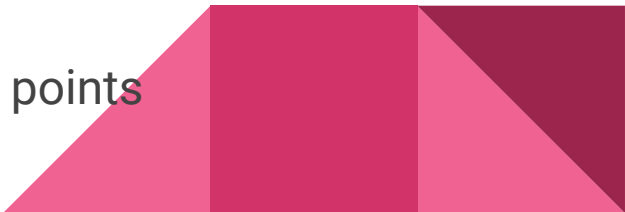
- Generalization of decision tree learning to produce an entire forest of small decision trees for an ensemble, instead of a single large tree
- Construction of each small decision tree uses the entire training data, but only a randomly chosen handful of possible attributes
- This makes the resulting trees more independent of each other
- Especially these small trees won't all end up having the same root node!
- Individual trees are weak classifiers, but a majority vote over a large number of such tree will classify complex data very accurately



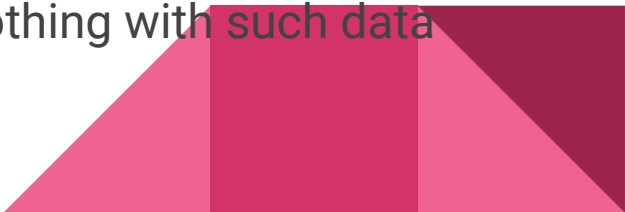
Boosting

- When creating the ensemble of simpler models, it would be nice to have these simpler models have complementing strengths
 - Attach a weight to each training sample, initially all weights are 1
 - After training each simpler model, adjust the weight of the training samples based on whether that model classified that training sample correctly
 - Training samples that were classified incorrectly see their weights increased, whereas correctly classified samples see their weights decreased
 - The simpler model constructed next will have strengths complementing the current model, instead of being yet another yes-man
- 


k -Nearest Neighbours

- Learning algorithm that works well assuming that the correct classifications change smoothly in the space of possible inputs
 - Preprocess the the training data and store it in some geometric data structure
 - To classify the given input x , find its k nearest neighbours in training data, then use majority vote among those neighbours
 - Simplest when $k = 1$, classify as its **nearest neighbour**
 - You should **normalize** multidimensional data among each dimension
 - Replace each numerical component with how many standard deviations away the point is from the mean value of that dimension
 - Preprocessing can eliminate **redundant** training data points
- 

Naive Bayes Classifier

- A simplified Bayes network where the true classification C is the root node, and evidence variables from X_1 to X_n are its children
 - Problem is to compute $P(C \mid X_1 \wedge \dots \wedge X_n)$
 - Simplify the formula with the naive assumption that all X_i are mutually independent, even though they are not
 - In practice, this approach works really well (kinda ultimate random forest)
 - Given a set of training data with many of the individual X_i values are missing, a **Naive Bayes Classifier** can still be constructed from that data
 - Most other supervised learning algorithms can do nothing with such data
- 

Machine Learning Theory

- A learning algorithm constructs a hypothesis h based on its training data
 - Assume the set of possible hypotheses H from which the algorithm returns one hypothesis based on its training data
 - Any hypothesis that is seriously wrong will be revealed with high probability after a small number of training data
 - A classifier h is **approximately correct** if its true error rate is less than ϵ , for some suitably small value of ϵ
 - How many training samples N do we need to ensure that the generated classifier h is approximately correct in this sense?
- 

PAC Learning

- Valiant: Assuming independent training data, for the probability of classifier to be approximately correct to be at least δ , the number of training samples N needs to be at least $N \geq (-\ln \delta + \ln |H|) / \epsilon$
- Depends on desired error rate ϵ , success probability $1 - \delta$, and number of possible hypotheses $|H|$ that the algorithm can generate
- Once the hypothesis classifies perfectly this many independent training samples, the smart money bets that this wasn't just a fluke, but similar performance within ϵ will continue with probability $1 - \delta$





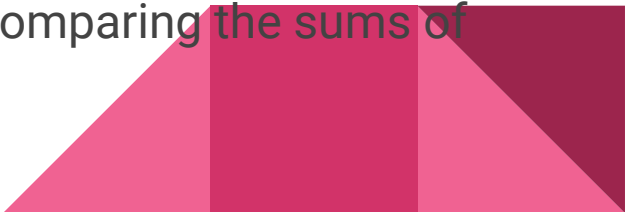
Module 12: Reinforcement Learning

Markov Decision Problems


- Fully observable state space with nondeterministic actions
- **Transition model** $T(s, a, s')$ gives the probability for the system ending up in state s' when performing the action a in state s
- No goal states, instead each transition has a **reward** $r(s, a, s')$
- Negative rewards are **penalties**
- State space may be **episodic** so that some states are **terminal**
- Alternatively state space may contain loops and allow infinite traversal



Example: (Simplified) Blackjack

- A one-player game that can be modelled as a Markov Decision Process
 - State consists of sum of player's cards (all face cards are tens, aces are either 1 or 11), along with one bit of information of whether the sum is soft or hard, along with dealer's visible upcard
 - Same as in all games, state space is a directed acyclic graph without loops
 - Try to get as close to 21 as you can without going bust
 - Two actions "hit" and "stay" leading to the next state
 - Both actions are nondeterministic (nature makes moves for dealer)
 - Terminal state determines the reward or penalty by comparing the sums of player and dealer cards
- 

Policies

- Because of nondeterminism, we can't just pre-plan path through the state space to maximize sum of rewards along the way
 - Instead, design a **policy** π that associates to each state s the action $\pi(s)$ that the agent will perform when finding itself in that state
 - In Markovian environments where history of reaching the state doesn't matter, policy can be **deterministic** and depend on current state alone
 - In fully observable environments, presence of other agents doesn't affect this
 - In partially observable environments, against other agents, and during the training stage policies should be randomized
- 

Example: Optimal Blackjack Policy

- Actual blackjack
- Also moves "double down" and "surrender"
- Policy gives optimal move in each situation
- Don't need to give action values or probabilities
- Those don't affect the actual move selection

Blackjack Basic Strategy Chart											
1 Deck, Dealer Stands on All 17s											
	Dealer Upcard										
Hard Total	2	3	4	5	6	7	8	9	10	A	
5-7	H	H	H	H	H	H	H	H	H	H	
8	H	H	H	D	D	H	H	H	H	H	
9	D	D	D	D	D	H	H	H	H	H	
10	D	D	D	D	D	D	D	D	H	H	
11	D	D	D	D	D	D	D	D	D	D	
12	H	H	S	S	S	H	H	H	H	H	
13	S	S	S	S	S	H	H	H	H	H	
14	S	S	S	S	S	H	H	H	H	H	
15	S	S	S	S	S	H	H	H	H	H	
16	S	S	S	S	S	H	H	H	R	R	
17	S	S	S	S	S	S	S	S	S	S	
Soft Total	2	3	4	5	6	7	8	9	10	A	
A,2	H	H	D	D	D	H	H	H	H	H	
A,3	H	H	D	D	D	H	H	H	H	H	
A,4	H	H	D	D	D	H	H	H	H	H	
A,5	H	H	D	D	D	H	H	H	H	H	
A,6	D	D	D	D	D	H	H	H	H	H	
A,7	S	DS	DS	DS	DS	S	S	H	H	S	
A,8	S	S	S	S	DS	S	S	S	S	S	
A,9	S	S	S	S	S	S	S	S	S	S	

(Pairs are listed on back of card.)
by Kenneth R Smith © 2008-2016 Bayview Strategies, LLC

Discounting Rewards

- Episodic state spaces such as Blackjack are simple enough, but what about state spaces with loops that allow a potentially infinite sum of rewards?
- Must be able to distinguish between policy that gains one dollar per minute, versus policy that gains one dollar per year
- Can compare average reward of policy per finite time unit
- Better solution to introduce **discounting factor** γ (gamma) in range $[0, 1]$
- Reward r gained k steps to the future is worth only $\gamma^k r$
- This will soon work out nicely with **Bellman equations**



State Utilities

- Once policy π is fixed, each state has **utility** $U_{\pi}(s)$, or just $U(s)$
- The utility $U_{\pi}(s)$ of the given state s is the expected sum of rewards by following the policy π starting from that state
- State utilities are connected to each other with **Bellman equation**

$$U(s) = \max_a \sum_{s'} T(s, a, s') (R(s, a, s') + \gamma U(s'))$$

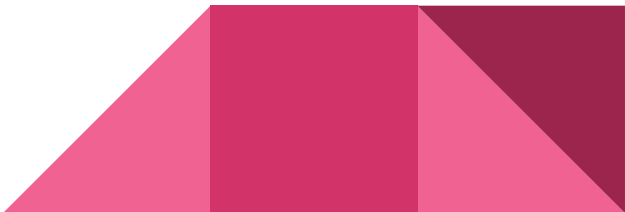
- Says that the utility of the state equals its local reward, plus the discounted rewards starting from the successor state after best action



Q-values

- Alternative, sometimes more convenient way to express utilities
- Instead of utility of state s , associate utility to **state-action pair** (s, a)
- $Q(s, a)$ is the expected utility starting from state s with action a
- Rewriting the previous Bellman equation, we get

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

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


Connecting U, Q, and Optimal Policy π^*

- Once Q-values are known, U-values are trivial to extract:

$$U(s) = \max_a Q(s, a)$$

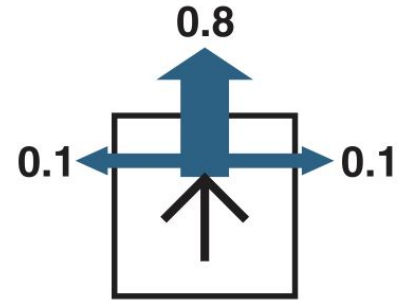
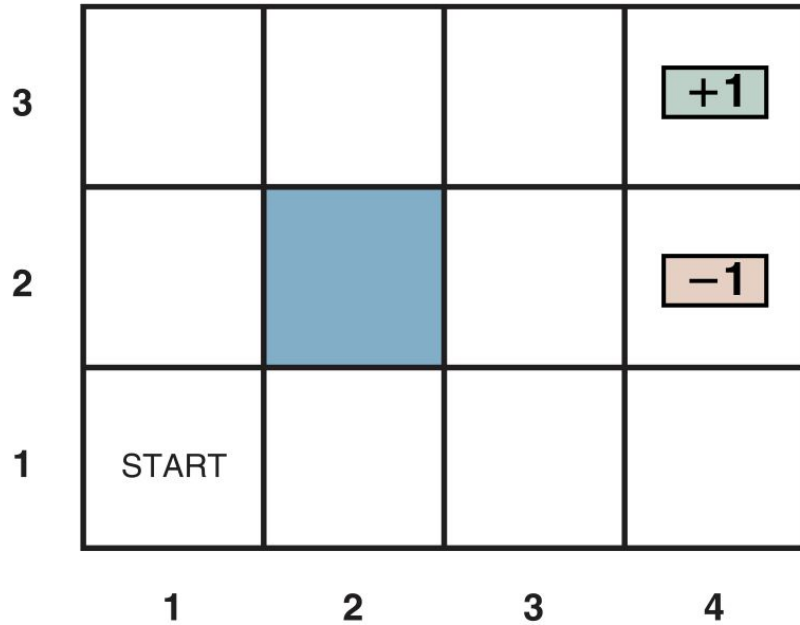
- Q-values can be used to determine the optimal policy π^* :

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

In state s , optimal policy uses action a that maximizes $Q(s, a)$



Example: GridWorld



Okay, So How To Solve U, Q, and π^* ?

- The optimal policy π^* is sufficient for optimal action
- We don't need to know how much better some action is compared to other actions to be able to execute that action
- Need to solve the set of equations either for U or Q
- For n states and b actions, U has n unknowns, Q has nb
- Equations look like linear equations, but unfortunately are not, due to the presence of the max operator
- Can't solve this directly as a system of linear equations



Dynamic Programming

- If the state space is known to be a directed acyclic graph (no loops), utility values $U(s)$ can be tabulated in a "bottom up" **dynamic programming** fashion
- Sort the states in some topological order so that if there is a possible transition from state s to state s' , the state s' is processed before state s
- Loop through all states in this order
- When the loop arrives to look at state s , the utility $U(s')$ for all its possible successor states has already been computed
- Can now compute $U(s)$ on the spot from Bellman equation



Value Iteration

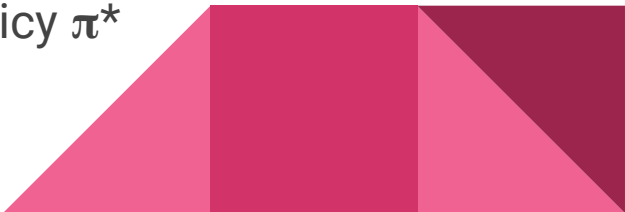
- Solve the n unknowns $U(s)$ together in an iterative fashion
- Initialize each utility estimate $U_0(s)$ to zero
- Use the current set of $U_i(s)$ values to compute new values $U_{i+1}(s)$ using the modified Bellman update equation

$$U_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') (R(s, a, s') + \gamma U_i(s'))$$

- Repeat until convergence



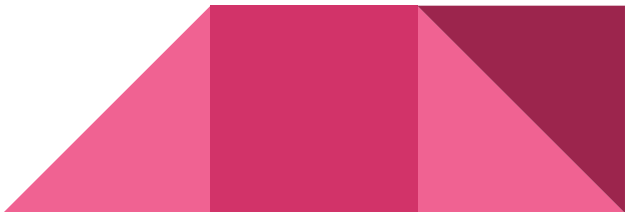
Value Iteration Convergence

- It can be proven that previous equations will converge to actual state utilities that satisfy the Bellman equations
 - However, for tricky state spaces, value iteration may need an exponential number of steps to converge
 - The essential problem is that this method implicitly computes the exact action values $Q(s, a)$ for all states s and actions a
 - However, recall that when an action a is known to be better than action b in some state, we don't need to know how much better it is
 - This redundant knowledge doesn't affect optimal policy π^*
- 

Evaluating Utilities With A Fixed Policy

- Suppose the policy π is fixed to something
- Calculate the utilities $U_{\pi}(s)$ for each state assuming that fixed policy, not necessarily optimal
- Under assumption of fixed policy, Bellman equations become linear!

$$U(s) = \sum_{s'} T(s, a, s') (R(s, \pi(s), s') + \gamma U(s'))$$

- Recall that T and R are known constants, not functions
 - Use your favourite linear algebra solver to solve $U(s)$
- 

Bellman Updates On Fixed Policies

- If we don't happen to have a powerful linear algebra solver at hand, we can still use modified Bellman updates to calculate $U_{\pi}(s)$

$$U_{i+1}(s) = \sum_{s'} T(s, a, s') (R(s, \pi(s), s') + \gamma U_i(s'))$$

- As before, these $U_{i+1}(s)$ converge to true utilities $U_{\pi}(s)$



Policy Iteration

- Faster way to solve for optimal policy directly than value iteration
- Algorithm alternates between two stages of **policy evaluation** and **policy improvement**, until policy improvement step does not change current policy
- Start with any random policy π_0 that can be anything (usually greedy)
- Compute all state utilities $U_0(s)$ assuming the initial policy π_0
- Given utilities $U_0(s)$, compute new policy π_1 from π_0 with greedy local updates
- Repeat until new policy π_{i+1} is the same as its previous policy π_i
- Can be proven to converge to optimal policy π^*



Solutions to GridWorld

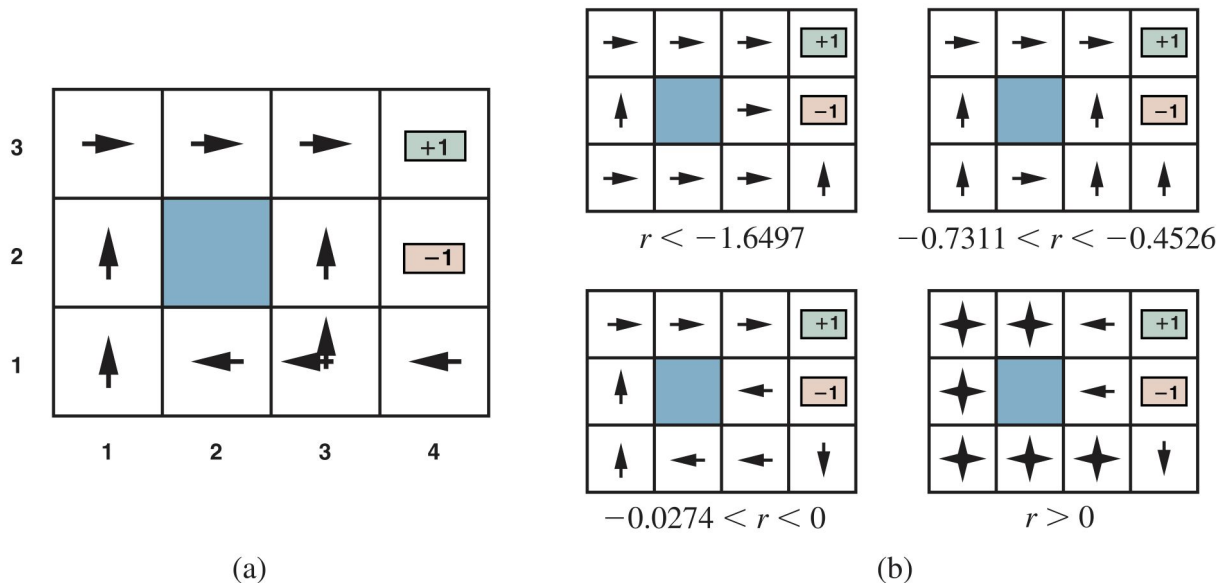


Figure 17.2 (a) The optimal policies for the stochastic environment with $r = -0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r .

Unknown Transitions And Rewards

- Previous algorithms assumed that transition probabilities and reward functions $T(s, a, s')$ and $R(s, a, s')$ are fully known
- When exploring an unknown environment, this is not necessarily the case
- Environment is a black box and only tells us the current state and reward
- How can we find a good policy in such environments?
- Need repeated exploration to see what happens if we do various things
- Instead of trying out an exponential number of possible action sequences, we wish to try out as few sequences as possible and generalize from there




Model-Free Learning

- In an observable state space, we could just perform many action sequences and tabulate how many times action a led from state s to state s'
- As in frequentist probabilities, these counts would converge to $T(s, a, s')$
- However, building such probability model turns out to be a completely redundant middleman who can be eliminated without losing anything!
- **Model-free reinforcement learning** techniques use this information only implicitly, converging to optimal policies without creating any explicit T or R tables along the way



Monte Carlo Policy Evaluation

- Given a fixed policy π , how well does it perform in an unknown environment?
 - Assume an episodic environment where we get to start at any state
 - Generate a large number of training samples following the policy π
 - After generating training examples, loop through each state s and find all training samples where that state s was encountered
 - Average the rewards following the state s seen in these training samples, to use as estimate of the actual value $U_{\pi}(s)$
 - When using a large number of training samples, state transitions behave according to the underlying transition probabilities $T(s, a, s')$
- 

Temporal Difference Learning

- Observation: for fixed policy π , the state utility $U_{\pi}(s)$ is the weighted average of the utilities its successor states s' , plus the transition reward
- For each transition from state s to state s' in each training sample, **temporal difference learning** uses the update formula

$$U_{\pi}(s) \leftarrow U_{\pi}(s) + \alpha(R(s, \pi(s), s') + \gamma U_{\pi}(s') - U_{\pi}(s))$$

- The parameter α is the (somewhat misnamed) **learning rate** of TD-algorithm



Active Reinforcement Learning

- Previous policy evaluation algorithms are passive in that they don't change the policy during the policy evaluation
- Policy changes in the policy improvement stage, to be evaluated again
- **Active reinforcement learning** locally updates the policy as soon as some action value $Q(s, a)$ looks better than $Q(s, b)$ for the previous best action b
- Further training samples generated using this updated policy



Exploration vs. Exploitation


- Dilemma: when performing a training run, following the current policy does not provide samples for values for other possible actions
- A good action that was unlucky in the early training runs never gets a chance to prove its worth, unless the current optimal action value $Q(s, a)$ deteriorates
- Must occasionally try out other actions than the current optimal action
- However, this can't be done all the time, since otherwise the measured action values will get all out of whack
- In each state, ϵ -greedy algorithms uses the current best action with probability ϵ , and a random action with probability $1 - \epsilon$



Softmax Action Selection

- **Softmax** is an easy way to choose randomly from different actions so that actions that currently seem better have a higher chance of being chosen
- Technically, should be called "soft argmax", instead of "softmax"
- Action a whose current value estimate is $Q(s, a)$ is chosen with probability

$$e^{\beta Q(s, a)} / \sum_{a'} e^{\beta Q(s, a')}$$

- Starting with β close to zero makes distribution more uniform
 - Larger values of β over time make distribution approach argmax
- 

Temporal Difference Q-Learning


- When observing the state transition from state s to state s' when taking the action a , apply the **temporal difference update** to state-action function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

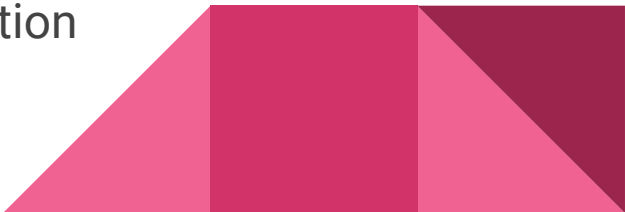
- For each step, rule looks at all possible actions in successor state
- Alternative **SARSA rule** looks at the action a' actually taken at successor:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a, s') + \gamma Q(s', a') - Q(s, a))$$


Multistep Temporal Difference

- In state spaces where all rewards and penalties come in the end, adjusting Q-values using just one step lookahead makes these values converge slowly
 - Generalize the temporal difference formula to look k steps ahead
 - Normal temporal difference special case of this with $k = 1$
 - Rewrite the Bellman update formula to adjust $Q(s, a)$ towards the sum of next k rewards plus the value of action taken k steps from the current state
 - Generalization $TD(\lambda)$ uses **exponentially decaying** weights for summing up the rewards ahead in the training sequence
 - Parameter λ is in range $[0, 1]$, easy update rule
- 

Reward Shaping

- Technique analogous to using a heuristic function in A* state space search
 - Uses background knowledge to speed up searching for the optimal policy
 - Define a potential function $\Phi(s)$ for states to guesstimate how good they are
 - This potential function should reflect the features of the current state
 - Instead of transition reward function $R(s, a, s')$, use $R(s, a, s') + \gamma\Phi(s') - \Phi(s)$
 - It can be proven that the optimal policy found using such modified rewards will still be optimal under the original reward function
 - However, the search for optimal policy will now converge faster, assuming that the potential function Φ contains useful information
- 

Generalization in Reinforcement Learning

- In astronomically large state spaces (for example, Backgammon), it is not possible to explore but an infinitesimal portion of all possible states
- Must generalize the knowledge acquired in the visited states to unseen states that are in some sense "similar" to those visited states
- Project a complex state into a smaller **feature vector**
- Use a neural network or similar **function approximator** to estimate the utility of the state based on this feature vector
- Temporal difference updates performed as neural network feedback



TD-Gammon

- Back in 1992, backgammon player using feedforward neural networks that estimate the state utility as cutoff evaluation in expectimax search
- Start with random neural network that surely played pretty comically
- Have two copies of such network repeatedly play against each other
- After each match, observe the outcome, and use temporal difference updates to adjust the neural network weights closer to the observed outcome
- Converged to fixed weights after 1.5 million matches using expert-chosen features, resulting player revolutionized the theory of backgammon

