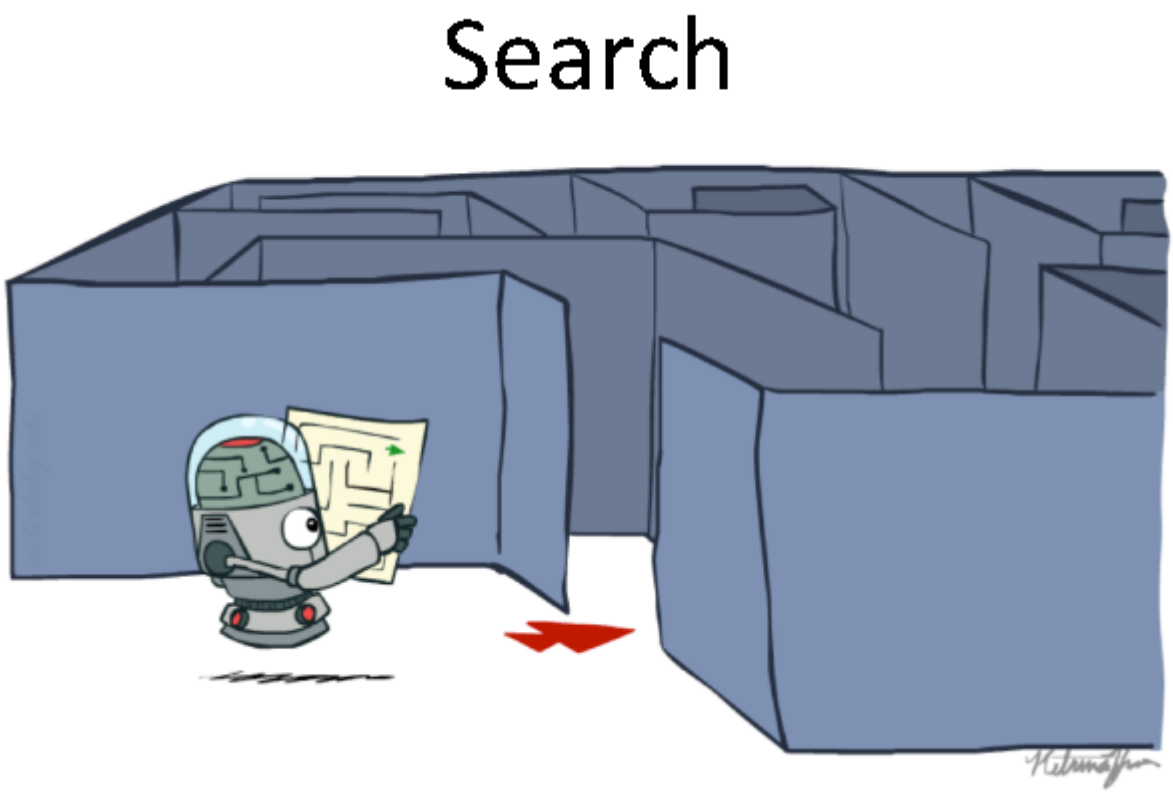
Examen samenvatting

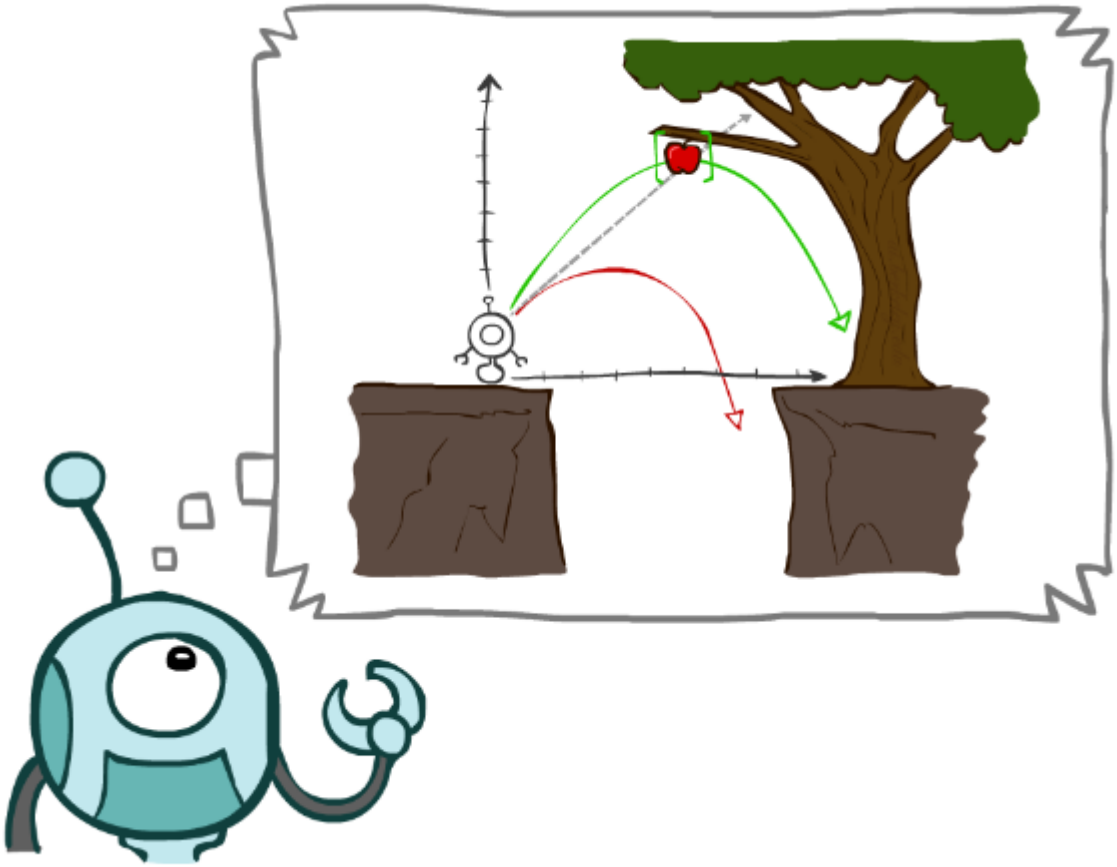
> Samenvatting van AI van lecture 1 - 10
> https://github.com/mebusy/notes/tree/master/dev_notes voor extra info van andere student
> In het begin nederlands vanaf lecture heb ik opgegeven en is het gwn engels.

Lecture 1 Search



Intro

Agent that plan ahead



Reflex agents

- Acties worden bepaald door zijn observaties.
- Kan eventueel een geheugen hebben van de momentele status van de wereld.
- Het bepaald zijn acties zonder de consequences in rekening te houden.
- **Ziet de wereld hoe het is.**

Planning agents

- Vraagt zich af, "Wat als?".
- Bepaalt zijn acties op basis van hypothesis, dus kijkt naar zijn consequences van zijn acties.
- Moet een model bekijken of hoe de wereld kan zijn indien het bepaalde acties neem.
- Het moet een goal hebben. (Testbaar)
- **Ziet hoe de wereld kan zijn.**

Search problems

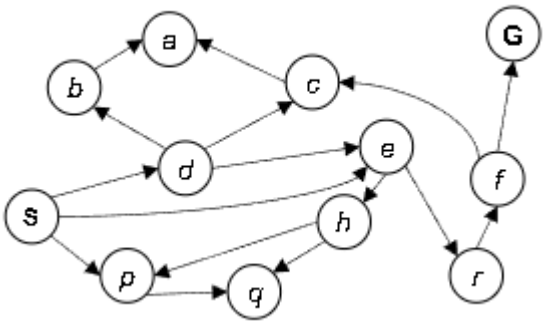
Basis van search problems

- State space
- Successor function met zijn acties en de kost van zijn acties.
- Een start state en goal state.
  Een oplossing van search problem, is een reeks van acties die genomen wordt van start state om de goal state te bereiken.

State space

Een state space is onze model dat we willen voorstellen. Het bevat verschillende componenten: de verschillende states, acties die het kan uitvoeren, successor functies en zijn goal state(s).

State space graphs



Dit is een wiskunde representatie van het zoek probleem.

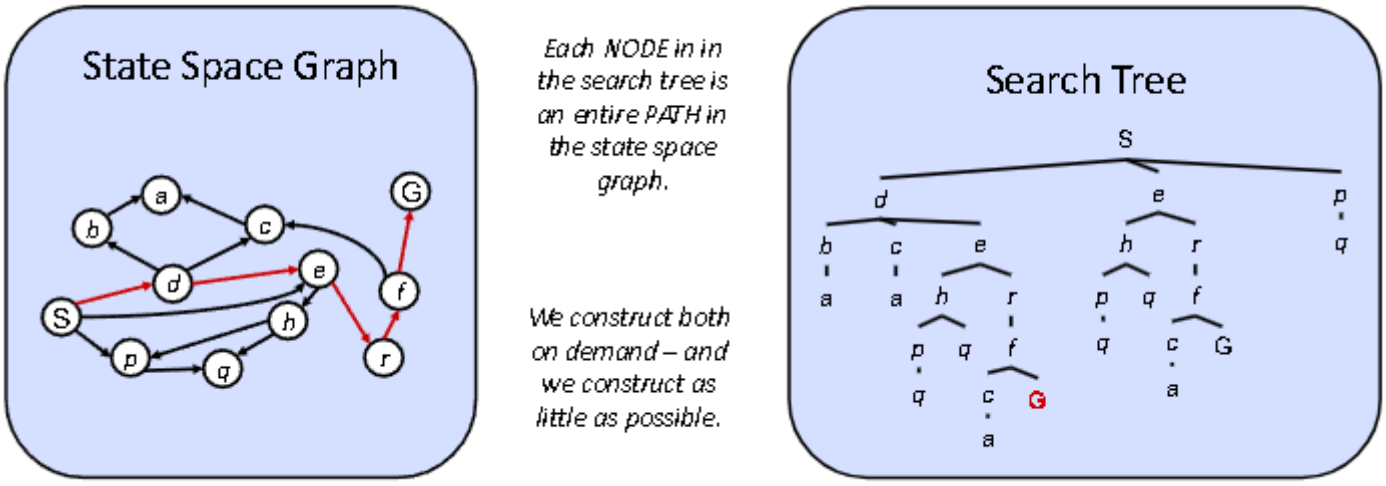
- Nodes zijn abstracte configuraties van de wereld.
- Arcs zijn de verschillende successors (resultaten van verschillende acties).
- Goal test is een set van alle goal nodes.
  Elke mogelijke state in de state space, komt enkel maar 1x voor.
  Meestal is dit niet mogelijk, memory overhead. Maar het is goed concept. (Kan handig zijn voor kleiner problemen)

State space search Tree

Het is een boom representatie van de state space.

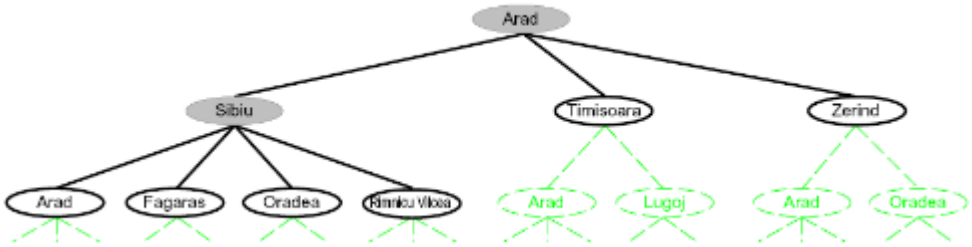
- Beeld meer een "what if?" model uit.
- De root is de start state.
- De kinderen zijn dan de successors die mogelijk zijn.
- Nodes zijn dan eigenlijk de verschillende plans om een bepaalde state te bekomen.
- **Meeste problemen zijn niet representeerbaar in een boommodel.**

Graphs vs Search trees



Een graph met een loop kan resulteren tot een boom dat oneindig door gaat.

Searching with a search trede



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

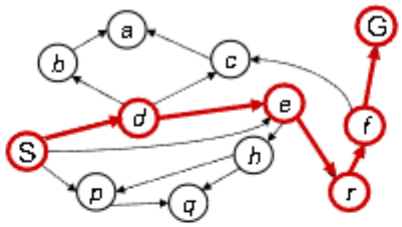
Pseudocode

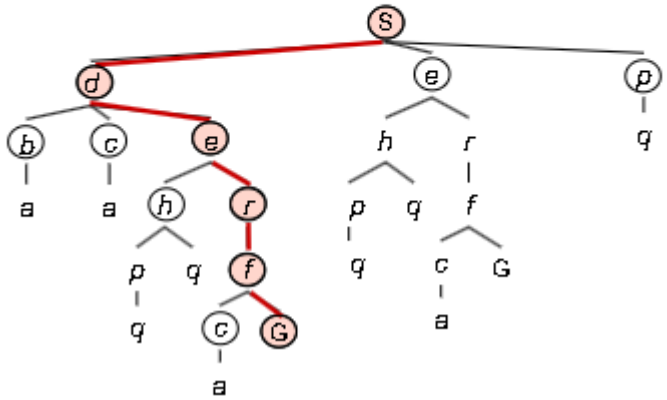
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Enkele belangrijke puntjes

- Fringe
- Expansion

- Exploration strategy





~~s~~
~~s → d~~
s → e
s → p
s → d → b
s → d → c
~~s → d → e~~
s → d → e → h
~~s → d → e → r~~
~~s → d → e → r → f~~
s → d → e → r → f → c
~~s → d → e → r → f → G~~

Uninformed search

Search probleem properties

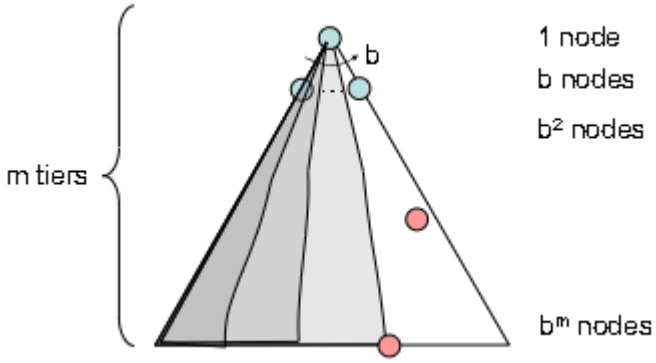
Enkele properties dat het moet nagaan.

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

DFS (Depth-First-Search)

> Bekijk wat de bodem ons eerst resulteert en kijk daarna zijn naasten.
> Onze fringe is een LIFO stack.

Properties



Op welke nodes breidt DFS uit?

- Neemt een linkse noden.
- Kan de hele boom aflopen.
- Indien de hoogte van de boom finite is, dan duurt het $O(b^m)$.

Hoeveel ruimte neemt de fringe op?

- Heeft enkel siblings op zijn pad naar de root, dus $O(bm)$.

Is het compleet?

- Indien m = infinite, dan kan de boom cyclussen hebben. Dit resulteert in geen oplossing omdat het infinite kan blijven runnen.

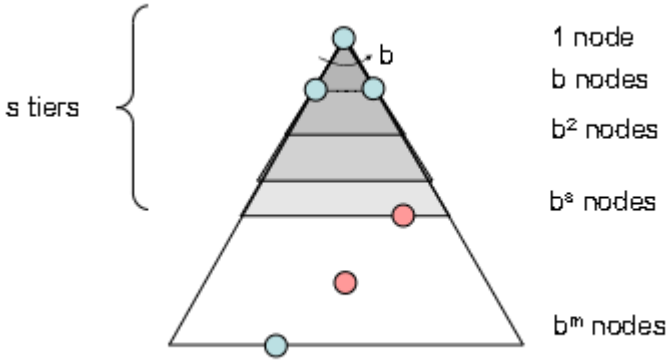
Is het optimaal?

- Nee, want het kiest de "Links mogelijk oplossing", zonder enige gedachten aan de diepte of de kost.

BFS (Breath-First-Search)

> Kijk wat elke laag ons resulteert.
> Onze fringe is een FIFO queue.
>
> 

Properties



Op welke nodes breidt BFS uit?

- Kijk naar alle nodes boven de hoogste oplossing in de boom.
- Laat de diepte van de hoogste oplossing s zijn.
- Dan duurt de search $O(b^s)$.

Hoeveel ruimte neemt de fringe op?

- Indien we heel de laatste laag mee tellen (laag van de hoogste oplossing), dan $O(b^s)$.

Is het compleet?

- s moet finite zijn als een oplossing bestaat, dus ja.

Is het optimaal?

- Enkel als de kost 1 zijn tussen de paden.

DFS vs BFS

DFS zou beter resultaten opleveren indien onze goal state diep links in de boom plaats vindt. BFS resulteert algemeen betere resultaten indien s > m, dus de diepte van goal state is hoger dan diepte van de boom.

Iterative deepening

Idee: Combineer voordelen van beide, dus we incrementen de diepte dat we DFS runnen hele tijd met 1.

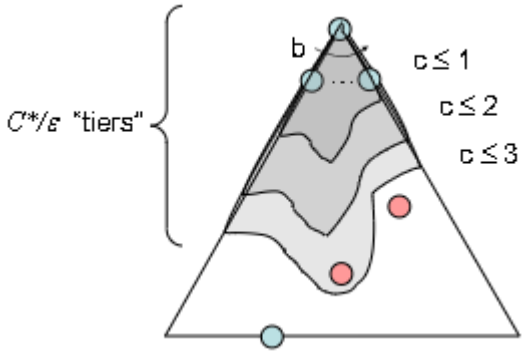
- DFS op diepte 1, check of er oplossingen zijn, indien niet dan ...
- DFS op diepte 2, ...
- ...
  Het is een goed idee maar nog niet perfect.

Uniform Cost Search

> Wat als onze paden geen kost hebben van 1, maar een willekeurige kost. DFS zou misschien ons resulteren in de kortste pad maar niet de meest kost vriendelijke pad.

Onze fringe is een priority queue.

Properties



Op welke nodes breidt UCS uit?

- Kijkt naar alle nodes met een kost minder dan de momentele goedkoopste kost.
- Als de oplossing een kost heeft van C *en de bogen een kost van minstens ϵ, dan is de "effective depth" ongeveer C/ϵ*.
- Heeft een tijdscomplexiteit van $O(b^{C*/\epsilon})$.

Hoeveel ruimte neemt de fringe op?

- $O(b^{C*/\epsilon})$ ongeveer.

Is het compleet?

- Indien de best oplossing een finite set is en de minimum arc cost is positief, dan ja.

Is het optimaal?

- Ja, zie bewijs in A*.

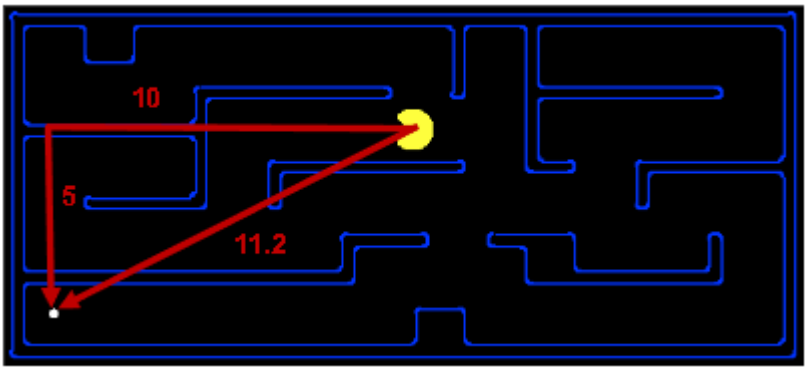
Issues

- Kijkt naar elke mogelijke "richting".
- Heeft geen informatie in verband met de goal state zijn locatie.

Informed search

Heuristics

- Heuristics dienen om een estimatie te maken, hoe dicht de goal state is vergeleken zijn huidige locatie.
- Ze zijn gemaakt voor specifieke zoek problemen.
- Voorbeelden: Manhatten distance, Euclidean distance.



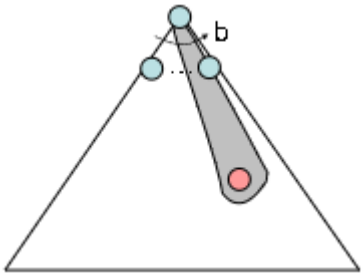
Onze estimatie van heuristic moet kleiner zijn dan onze werkelijke kost. Indien onze heuristic dichter naar onze werkelijke kost komt, hoe beter onze estimatie. Dit komt meestal met nadeel, dat onze heuristic complexer wordt om te berekenen.

Greedy Search

> Kijk naar de node die het dicht bij de goal state is.
>
> 

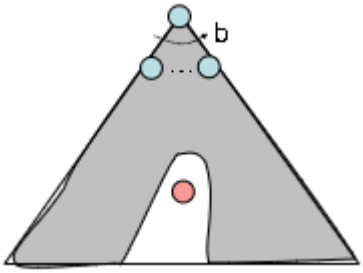
Algemene gevallen:

- Best-first, brengt rechtstreeks naar de goal.



Worst case:

- Dit kan resulteren in een "badly-guided DFS". Ofwel lange paden met grote kost.



A* Search

> Een combinatie van UCS en Greedy, waarbij we de paden sorteren op kost en gebruik maken van heuristics om de kortste pad te nemen.
>
> 

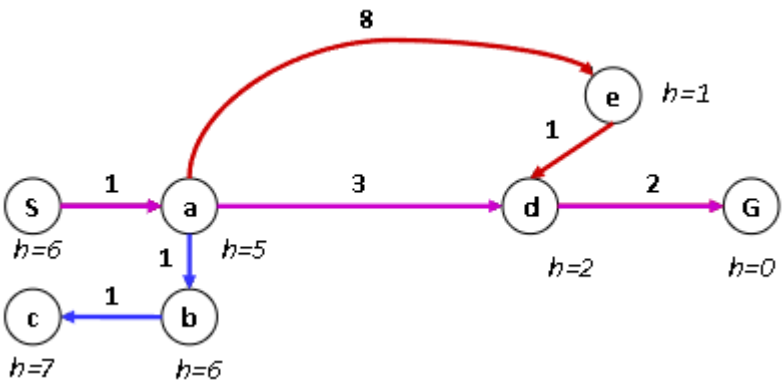
Combining UCS en Greedy

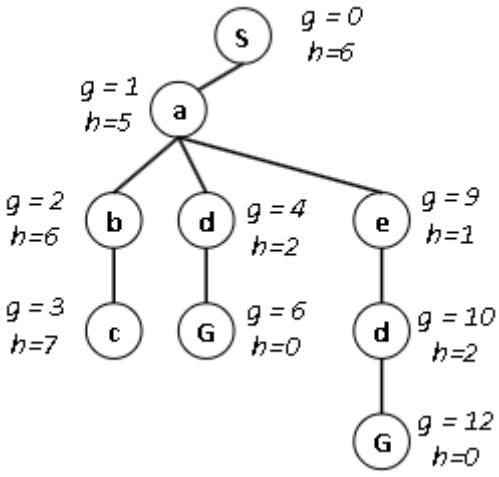
We maken gebruik van twee functies:

- Uniform-cost orders by path cost, ofwel *backward cost g(n)*.
- greedy orders by goal proximity, ofwel *forward cost h(n)*.

A* search orders by the sum:

- f(n) = g(n) + h(n)





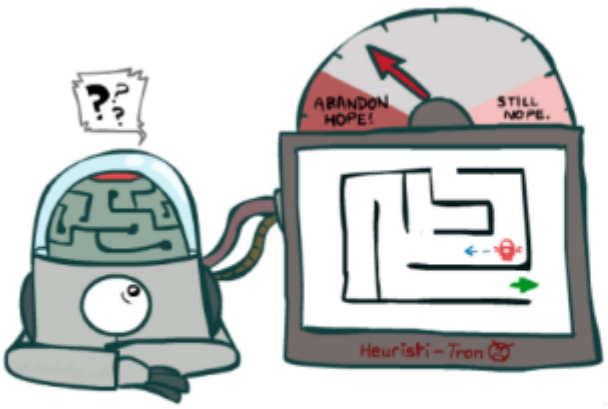
Wanneer moet A* eindigen?

- We eindigen A* search indien we de goal state dequeue van de priority queue. Dit is omdat indien we een goal state enqueue, kunnen er nog andere paden zijn die leiden tot een betere pad.

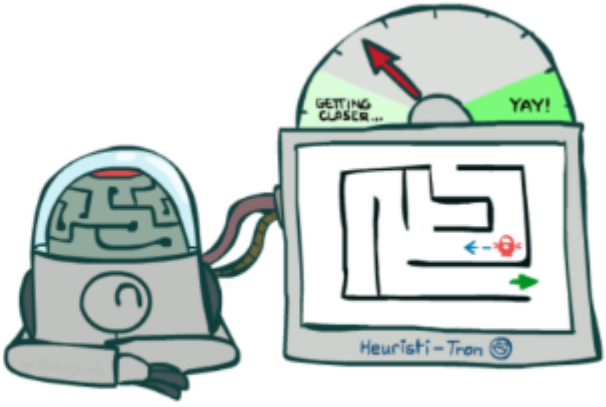
Is A* optimaal?

- Nee, indien we een slechte heuristics hebben.
- Actual bad goal cost < estimated good goal cost.
- We willen dat de estimatie kleiner is dan de werkelijke waardes.

Admissible Heuristics



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe

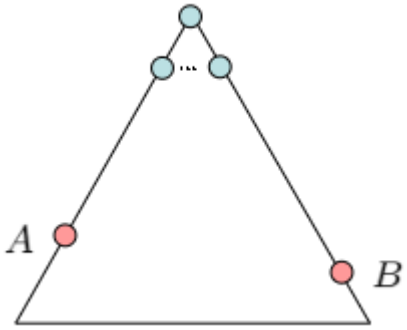


Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Een heuristic h is admissible als:

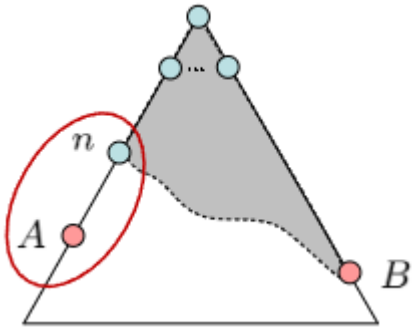
- $0 \leq h(n) \leq h^*(n)$. Waarbij $h^*(n)$ is de true cost van de dicht bij zijnde goal.
  Een admissible heuristic bedenken is de belangrijkste stap om A* te implementeren.

Optimality



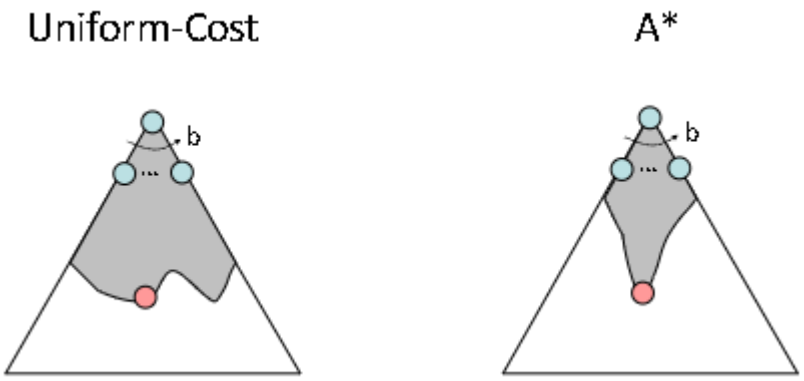
Stel:

- A is een optimale node
- B is een suboptimale node
- h is admissible
  Dan:
- A zou de fringe exiten voor B
  Bewijs:

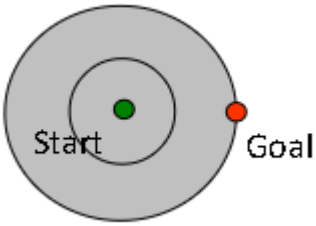


- Stel B is op de fringe.
- Stel de voorganger van A is n, die ook op de fringe zit.
- Dan: n zal eerder uitbreiden dan B
  - f(n) is less or equal to f(A)
    - Definitie van f-cost admissibility van h, h = 0 als we kijken naar een goal state.
      - $f(n) = g(n) + h(n)$
      - $f(n) \leq g(n)$
      - $g(A) = f(A)$
  - f(A) is less than f(B)
    - B is suboptimal, h = 0 at a goal.
      - $g(A) < g(B)$
      - $f(A) < f(B)$
  - n expands before B
    - $f(n) \leq f(A) < f(B)$
- Alle voorgangers van A zullen eerder expanden dan B
- A zou dan uiteindelijk eerder expanden dan B
- A* search is dan optimaal

Properties



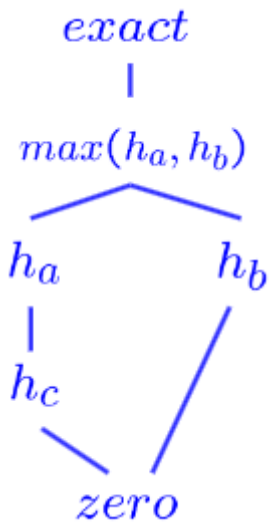
UCS zou uniform expanden, terwijl A* meer zou expanden richting de goal.



Semi-Lattice of heuristics

Trivial heuristics, Dominance

- **Dominance:** $h_a \geq h_c$ **if**
  - $\forall n : h_a(n) \geq h_c(n)$
  - Dus voor alle waardes n, is heuristic a beter dan heuristic c.
- **Heuristic vormen een Semi-Lattice:**
  - Max van de admissible heuristics is admissible
    - $h(n) = max(h_a(n), h_b(n))$



- **Triviale heuristics**
  - Bodem van de lattice is de zero heuristic
  - Top of lattice is de exact heuristic

A* Graph search

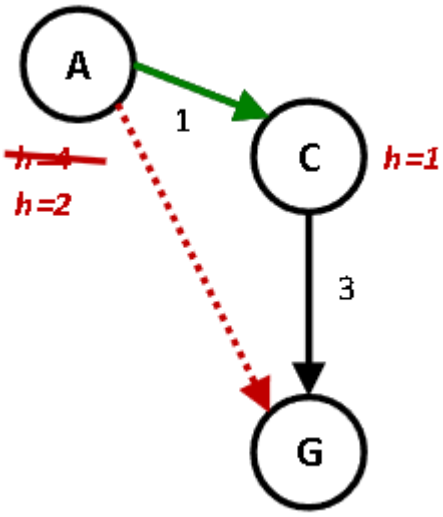
Indien we loops hebben on onze state graph, dit kan leiden in infinite search.

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
    - Before expanding a node, check to make sure its state has never been expanded before
    - If you have, skip it, if you haven't, call the successor function and then add it to closed set.

- Important: **store the closed set as a set**, not a list
- Can graph search wreck completeness? Why/why not?
  - No. It's parts of the Search Tree.
- How about optimality?
  - Unfortunately close list will introduce another problem
  - Admissible heuristic with tree search is optimal but graph search no guarantees.

Er kunnen fouten lopen, indien we in de eerste iteratie een slechte keuzen maken, zullen we de altijd fouten keuzes maken daarna. Dit is omdat we werken met een closed set, en indien een slechte heuristic ons de foute pad leidt, dan kan het zijn dat een node al doorlopen is die leidt tot een betere pad.

Consistency of heuristics



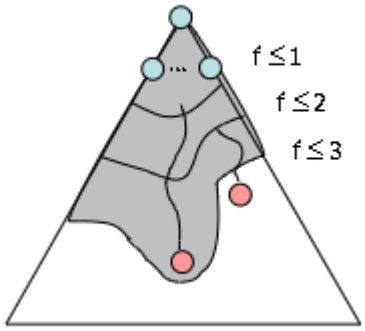
Main idea: estimated heuristic costs ≤ actual costs

- Admissibility: heuristic cost $\leq$ actual cost to goal
  - h(A) ≤ actual cost from A to G
- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  + h(A) – h(C) ≤ cost(A to C)
  Consequences van consistency
- The f value along a path never decreases
  - h(A) ≤ cost(A to C) + h(C)
- A* graph search is optimal

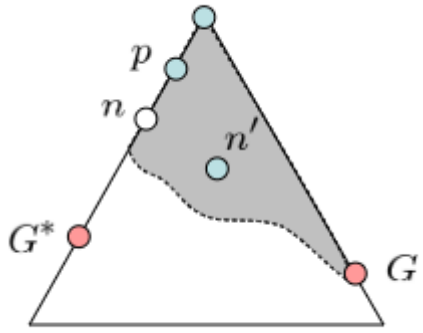
Optimality graph search

- **Sketch: consider what A* does with a consistent heuristic:**
  - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
  - Fact 2: For every state s, nodes that reach ***s*** optimally are expanded before nodes that reach ***s*** suboptimally

- Result: A* graph search is optimal



- **Consider what A* does:**
  - Expands nodes in increasing total f value (f-contours)
  - Reminder: f(n) = g(n) + h(n) = cost to n + heuristic
  - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first
- **Bewijs via contradictie:**
  - New possible problem: some n on path to G* isn't in queue when we need it, because some worse n' for the same state dequeued and expanded first (disaster!)
  - Take the highest such n in tree
  - Let p be the ancestor of n that was on the queue when n' was popped
  - f(p) < f(n) because of consistency
  - f(n) < f(n') because n' is suboptimal
  - p would have been expanded before n'
  - Contradiction!



Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problem.

Lecture 2 CSP

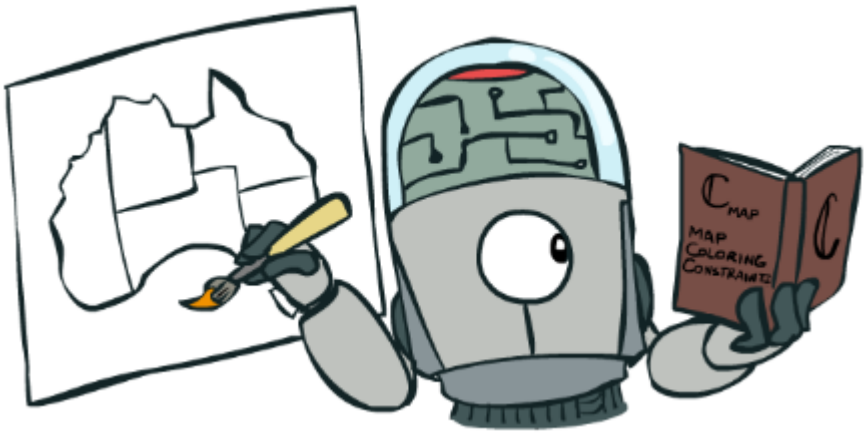
> Start part one.

What is search for?

- Assumptions about the world:
  - a single agent
  - deterministic actions
  - fully observed state
    - you KNOW the configuration that you start in
    - and then you plan about exactly how the world will evolve
  - discrete state space
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance
- Identification: assignments to variables
  - The goal itself is important, **not the path**
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

CSP (Constraint Satisfaction Problems)

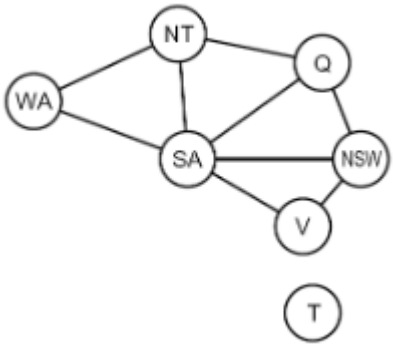
Constraint Graphs



- Standard search problems
  - State is een "Black box": arbitraire data structuur.
  - Goal test kan ener welke functie zijn over states.
  - Successor functie kan ook ener welke functie zijn.

- CSP:
  - Het is een speciale subset van zoekproblemen
  - Een state is gedefinieerd door **variabelen X**, met waardes van **een domein D**.
  - Goal test is een **set van constraints**, die specificeren welke combinaties toegelaten zijn voor een subset van de variabelen.
- Allows useful general-purpose algorithms with more power than standard search algorithms.

Constraint graph



- Binary CSP:
  - Each constraint relates (at most) two variables
- Binary constraint graph:
  - Nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search.
  - E.g., Tasmania is an independent subproblem!

Varieties of CSPs and Constraints

CSPs

- Discrete variables
  - Finite domains
    - Indien we size d mogelijkheden hebben in het domein, dan hebben we $O(d^n)$ mogelijke complete assignments.
    - E.g., Boolean CSPs, including Boolean satisfiability (NP- complete)

- Infinite domains (integers, strings, etc.)
  - E.g., job scheduling, variables are start/end times for each job
  - Linear constraints solvable, nonlinear undecidable

- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

Constraints

- Varieties of Constraints
  - Unary constraints
    - Involve a single variable (equivalent met reducing domains)
    - SA ≠ green
  - Binary constraint
    - Involves pairs of variables
    - SA ≠ WA
  - Higher-order constraints
    - Involves 3 or more variables
      - e.g., cryptarithmetic column constraints
- Preferences
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets

Standard Search Formulation

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - Initial state:
    - The empty assignment, {}
  - Successor function:
    - Assign a value to an ***unassigned*** variable
  - Goal test:
    - The current assignment is complete and satisfies all constraints

- We'll start with the straightforward, naïve approach, then improve it
- What would BFS do ?
  - Will expand all levels
- What would DFS do ?
  - Going to look everywhere where they aren't first.
- What problems does naïve search have?
  - It is inefficient.

Backtracking Search

- Backtracking search is the basic ***uninformed*** algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called backtracking search (not the best name)

Pseudocode

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

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?
  - Het kiest zijn variables op basis wat hij nog niet geprobeerd heeft.

Improving backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering:
  - Can we detect inevitable failure early?
- Structure:
  - Can we exploit the problem structure?

Filtering

> Keep track of domains for unassigned variables and cross off bad options
>
> 

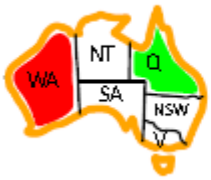
Forward checking

- Cross off values that violate a constraint when added to the existing assignment.

- If we assigned red to WA , we should remove red choice from NT , SA.
- So that basic idea when I assigned something I look at its ***neighbors*** in the graph and cross things off, that's called forward checking. So the neighbors of WA would lose red.
- Forward checking doesn't check interactions between unassigned variables just checks interactions between assigned variables and their neighbors. Anything further is thinking too hard for forward checking.

Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

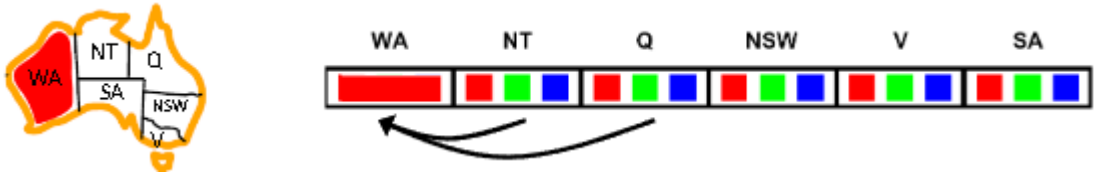


  - Hier zien we duidelijk dat SA en NT alleen als opties blauw hebben, dit zal leiden tot een fout oplossing.
- Constraint propagation: reason from constraint to constraint,
  - Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable.

Consistency of A single arc

- So far we talked about checking an assignment against its neighbors, here we were talking out checking between two unassigned variables.

- An arc X → Y is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint
- Example:



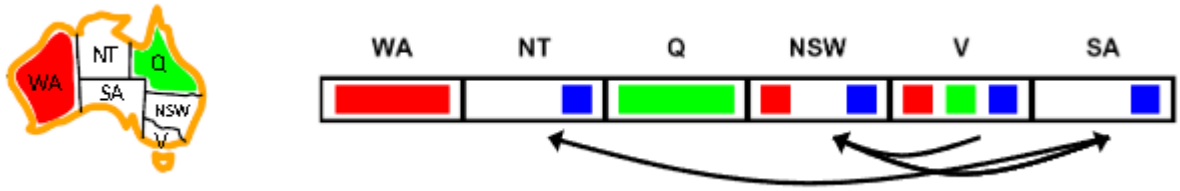
  - Here WA is assigned. The NT is not assigned, but I can still look at the 2 of them and check if this arc is consistent. So let's do it, we check everything in the tail. So we look at the NT and we say, is there anything in you remaining domain which would have no continuation into the head?
  - So we say, well, if I assigned you blue, it would be ok. Green ? it would be ok. Red ? not ok. So red is something in the tail for which there is no assignments in the head which doesn't cause a constraint violation. So this arc is not constraint.
  - We can however, make it consistent. We can remove things from the tail, the red in NT.
  - So now we can check other consistencies, let's try Q->WA. These two are not actually connected by a constraint, so it should be easy to check. This is arc is already consistent.
- How can you remember this?
  - Remember, the constraints are like rules, and these algorithms are like police. They're going to go and enforce the rules. And you can imagine this arc is going to get pulled over by your algorithm, which is the CSP police. And what do they do when they pull the arc over? Right they pop open the trunk and they look for anything that's illegal. They are going to take anything bad out.
  - All these algorithms have the same shape. You pull over an arc, you fish around in its trunk and cross the bad thing off. That's enforcing the consistency of a single arc.

Delete from the tail!

- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Arc Consistency of an Entire CSP

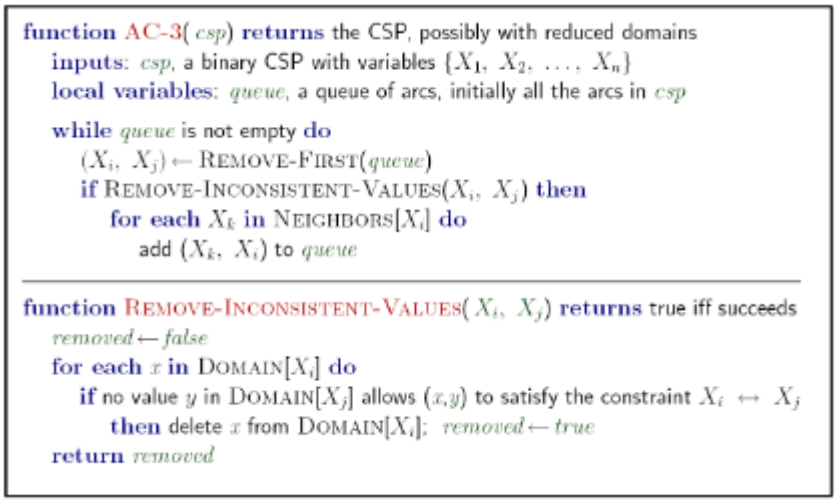
- A simple form of propagation makes sure **all** arcs are consistent:
- Example:



  - WA and Q have been assigned , red and green. NT, NSW, SA had their domain reduced by some previous pruning.
  - I can go visit arcs. First we check V->NSW. We notice they are neighbors. All right, this is the first time we're checking the consistency of an arc that doesn't point to an assignment. So I go through and I check the tail V, blue is find, red is fine, green is fine too.
  - Let's look SA->NSW. SA and NSW are adjacent, so I'm going to look at SA. What is in the tail? Blue, it is fine.
  - BUT, let's check the arc in the other direction. So now I look at NSW, is red ok? Yes. Is blue ok? NO. So we erase blue from NSW. Not it's consistent.

    - There's a tricky case. We just check V->NSW. We just declared it consistent, but that was on the basis of having blue and red available in the head at NSW. And one of those is gone, so the consistency may no longer hold.
    - So I have to go back to V, and I have to check V->NSW again. Red now is no longer ok. Erasing red from V.
    - So any time you do a delete of a value from a domain, every arc pointing into it needs to be rechecked.
    - So I keep doing this. The whole reason to do this is actually a completely different arc, SA->NT, neither of which is assigned. You noticed that you have to delete the blue from SA, which results in an empty domain, and an empty domain means a detected failure, which means backtracking.
- ***Remember: Delete from the tail!***
- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency **detects failure earlier** than forward checking
- Can be run as a preprocessor, or more commonly after each assignment
- What's the downside of enforceing arc consistency ?
    - Runtime is bad.
    - So there's a trade-off between doing more filtering and just making the core search run faster.
    - In general, this is a very powerful method.

Pseudocode

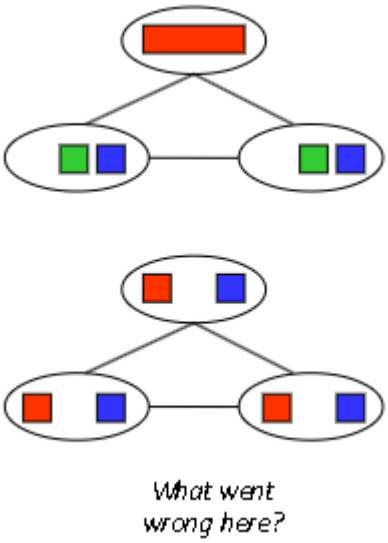


- Runtime: O(n 2d 3), can be reduced to O(n 2d 2)\
- … but detecting all possible future problems is NP-hard – why?
    - The reason is that it essentially requires checking all possible assignments of values to variables to ensure they do not violate any constraints. This is computationally expensive as the number of possible assignments is exponential in the number of variables and the size of their domains.

Limitations of Arc Consistency

- After enforcing arc consistency:
    - Can have one solution left
    - Can have multiple solutions left
    - Can have no solutions left
- Arc consistency still runs inside a backtracking search!
- The reason why our consistency in this bottom case wasn't sufficient to discover the inevitable failures because it's only checked in pairs. While it could be consistent in pairs, it's not consistent if we look in groups

of 3.



Ordering

Minimum remaining values (MRV)

- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain
- Why min rather than max?
  - "Fail-fast" ordering
  - Also called "most constrained variable"
- **Hardest variable**

Least constraining value (LCV)

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the least constraining value
  - I.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)
- To choose which value is the least-constraining value, enforce arc consistency for each value (on a scratch piece of paper).
  - For each value, count the total number of values remaining over all variables.
- **Easiest value**

    - We want the one that has the least impact on the rest of the graph
- Why least rather than most?
    - Because it's a CSP, and in CSP, you have to do every variable. Sooner or later, you have to do it. You don't have to do every value.
    - So you might as well do the hard variables first, but if you're picking values, you want to pick the ones that are likely to work out, and maybe you don't even have to try the hard ones.
- Combining these ordering ideas makes 1000 queens feasible

> Start part 2

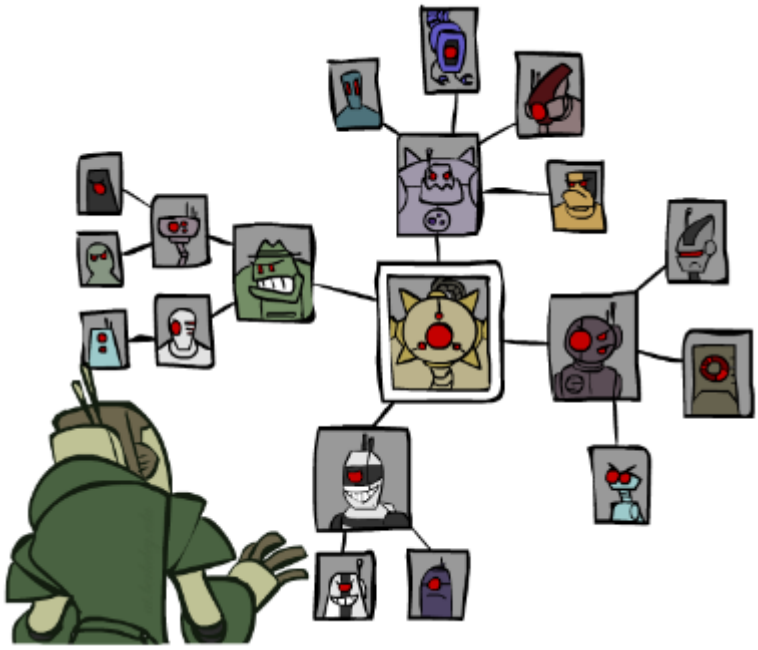
K-Consistency

- Increasing degrees of consistency
    - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
        - Every node's domain has at least one value that meets that node's constraints
        - basically just means you enforce unary constraints
    - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
    - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.
        - Arc-consistency says, if you can get one assigned, you can get 2 assigned
        - K-consistency says , if you can get k-1 assigned, then you can get k assigned. It's sort of mathematically a little weird, because it assumes that you can get to k-1, but who says you actually can? There is a stronger notion called strong k-consistency.
- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)

Strong K-Consistency

- Strong k-consistency: also k-1, k-2, … 1 consistent
- Claim: **strong n-consistency means we can solve without backtracking**!
- Why?
    - Choose any assignment to any variable
    - Choose a new variable
    - By 2-consistency, there is a choice consistent with the first
    - Choose a new variable
    - By 3-consistency, there is a choice consistent with the first 2
    - …
- Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

Structure



Sometimes you look at a CSP that you're trying to solve and you see it has some special graph structure and based on that graph structure there will be some technique available to you that allows you to solve it in a particularly efficient way and we're going to see a couple examples.

So for example if your CSP involved this giant criminal robot network you might think you should go after that guy in the center. That would be an example of exploiting a structure.

Problem

```
![[Pasted image 20240121195903.png]]
```

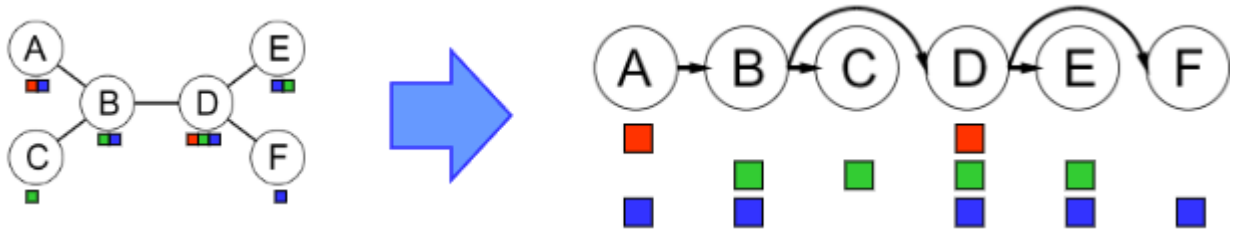
- Extreme case: independent subproblems
  - Example: Tasmania (T) and mainland do not interact (see figure)
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs

```
![[Pasted image 20240121200237.png]]
```

- It is a theorem that if the constraint graph has no loops then the CSP can be solved in time that is linear in the size of graph and quadratic in the size of domains. That's so much better than general CSPs worst exponential.
- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n \cdot d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later):
  - An example of the relation between syntactic restrictions and the complexity of reasoning
- Algorithm for solving a tree-structured CSPs:

- Order: Choose a root variable, order variables so that parents precede children.



  - Remove backward: For i := n to 2, apply RemoveInconsistent(Parent(X_i),X_i)
    - Once you've ordered it, we do a backwards path.
    - We start at F, and we go leftward. And for each node in this pass, we are going to make the arc which pointing to that node consistent.
  - Assign forward: For i = 1 : n, assign X i consistently with Parent(X_i)
- Runtime $O(n \cdot d^2)$
  - The “Remove backward” step takes $O(d^2)$ time for each of the n nodes in the tree (since each arc can be made consistent in $O(d^2)$ time), and the “Assign forward” step takes O(d) time for each node.
  - Since we have n nodes, this $n(d^2 + d)$, so $O(n \cdot d^2)$
- **Claim 1:** After backward pass, all root-to-leaf arcs are consistent
- **Proof:** Each X→Y was made consistent at one point and Y’s domain could not have been reduced thereafter (because Y’s children were processed before Y)

- **Claim 2:** If root-to-leaf arcs are consistent, forward assignment will not backtrack
- **Proof:** Induction on position
- Why doesn’t this algorithm work with cycles in the constraint graph?
  - The tree-structured CSP algorithm assumes a tree-like structure with no cycles. It orders variables so parents precede children and enforces arc consistency in one pass. If there are cycles, these assumptions fail, making the algorithm ineffective. For graphs with cycles, other algorithms like backtracking or local search are used.
- Note: we’ll see this basic idea again with Bayes’ nets

Improving Structure

```
![[Pasted image 20240121201845.png]]
```

So we can use this great algorithm on tree-structured CSP, but CSP is probably not tree-structured either. So we need some way of taking graphs which are not in these wonderful configurations, but or maybe closer.

Nearly Tree-Structured CSPs

```
![[Pasted image 20240121202153.png]]
```

- We are going to assign a value to it . Once we've assigned a value to it and we imagine that that value we assigned to SA is fixed. The rest of the arcs connected to SA can now be "forgotten". (Conditioning)
- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree.
- Cutset size c gives runtime O((dc) (n-c) d2), very fast for small c

Cutset Conditioning

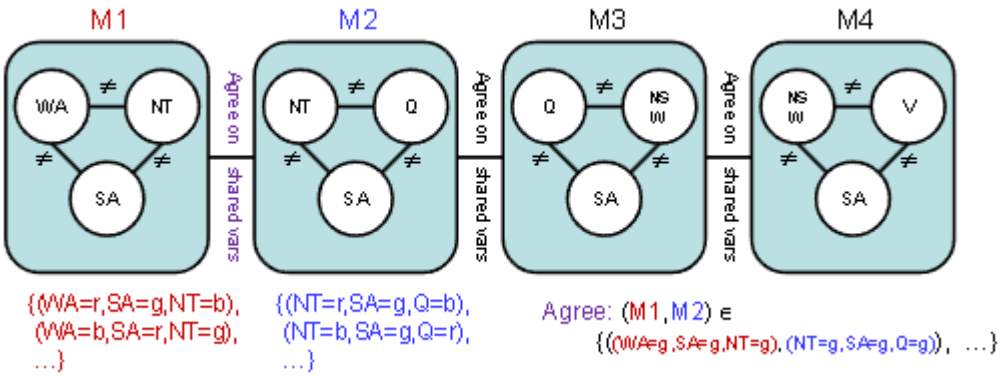
```
![[Pasted image 20240121202506.png]]
```

- Finding smallest cut-set is np-hard !

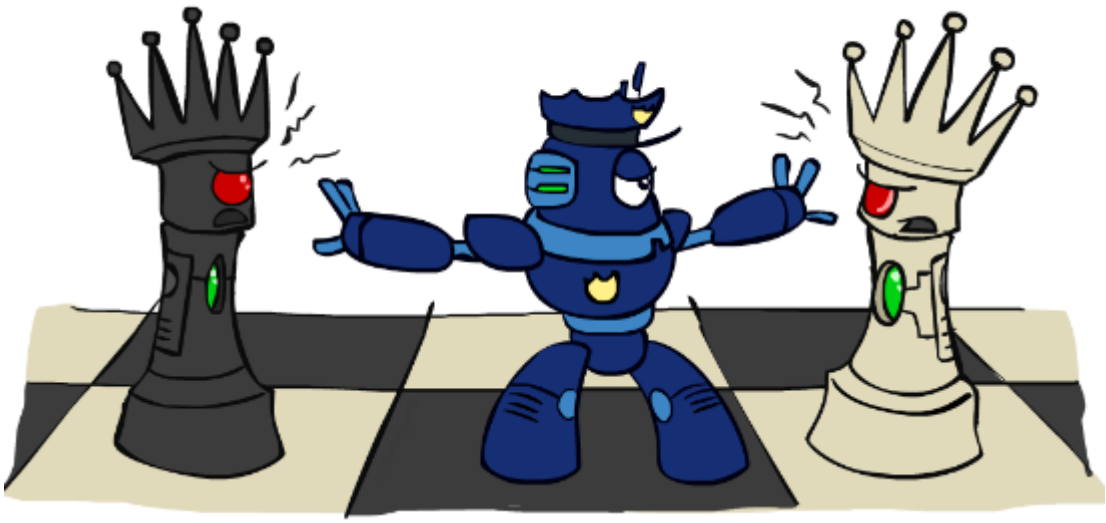
Tree Decomposition*

Tree Decomposition is another approach :

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Subproblems overlap to ensure consistent solutions



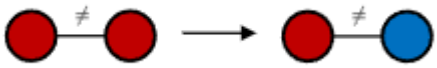
Iterative Improvement



For CSPs

- Iterative Algorithms for CSPs is our first example of a local search.
- Local search methods typically work with **“complete”** states, i.e., all variables assigned
- To apply to CSPs:
  1. Algorithm starts by assigning some value to each of the variables
     - Ignoring the constraints when doing so
  2. Take an assignment with unsatisfied constraints

3. Operators reassign variable values

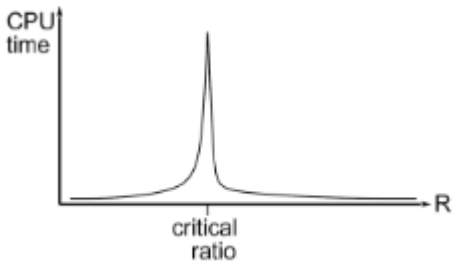


- No fringe! Live on the edge.
  - There is no backup plan, like the queue in search problem.
- Algorithm: While not solved, while at least one constraint is violated, repeat:
  - Variable selection:
    - randomly select any conflicted variable
  - Value selection: min-conflicts heuristic:
    - Choose a value that violates the fewest constraints (among all possible selections of values in its domain)
    - I.e., hill climb with h(n) = total number of violated constraints
- Questions
  - Do you risk introducing a new conflict? Yes
  - Can this thing run forever? Yes
  - Will it give you an optimal solution if there are waits? No.
  - Do you have any guarantee whatever? No
  - **BUT** it's very fast very often.

Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!
- The same appears to be true for any randomly-generated CSP except in a narrow range of the ratio.

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



- Looking at the graph, having allot of constraints and minimum constraints are both great. This is because having allot of constraints will make it easier to find the solution, while having as little as possible gives us more freedom.
- As u can see, there is a critical ratio, where the amount of constraints / amount of variables result in a high CPU time.

Summary: CSPs

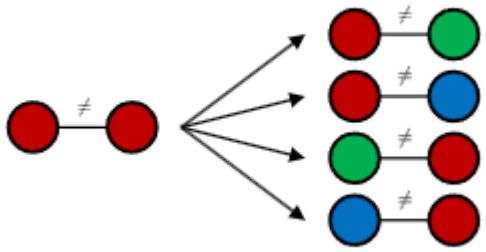
- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints
- Basic solution: backtracking search
- Speed-ups:
  - Ordering
  - Filtering

    - Structure
- ***Iterative min-conflicts is often effective in practice***
    - though, you have almost no guarantees.

Local search

```
![[Pasted image 20240121205239.png]]
```

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can’t make it better (no fringe!)
    - In local search you don't have the safety net. You got one position that you are currently at and you're trying to hill climb in some way.
- New successor function: local changes
    - You have a new idea of a successor function. The successor function now does not take a plan and extend the plan, instead it takes a complete assignment of some kind and modifies it. Your successor function is more about modification than about extension.

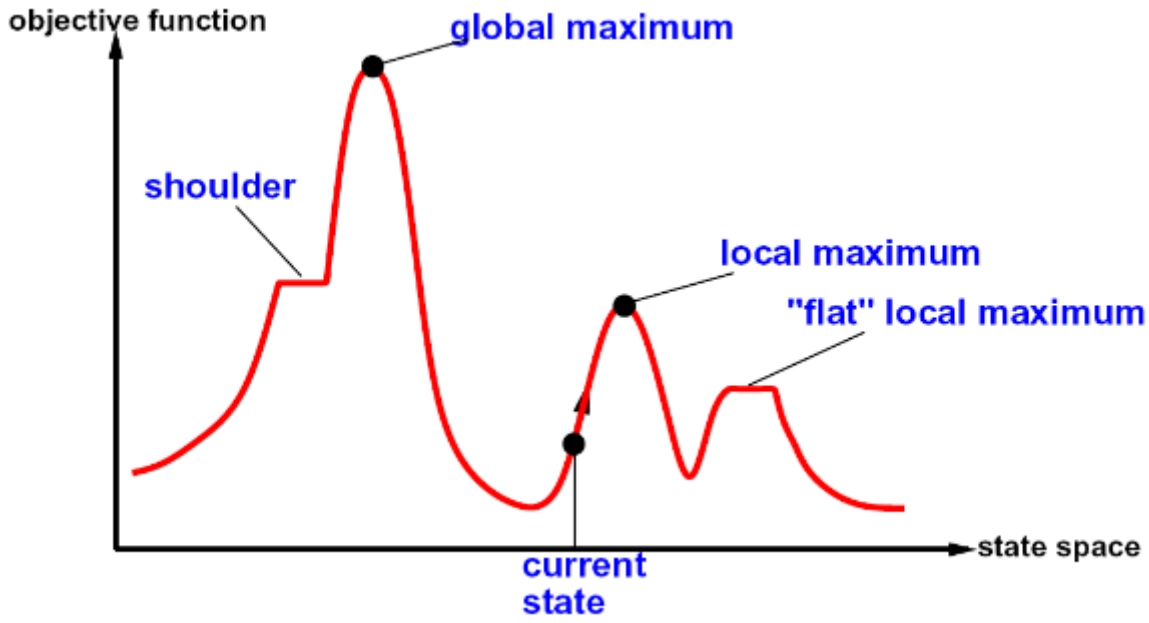


- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill climbing

- Simple, general idea:
    - Start wherever
    - Repeat: move to the best neighboring state
    - If no neighbors better than current, quit
- What’s bad about this approach?
    - Complete ? No.
    - Optimal ? No.
    - may reach local maximum.
- What’s good about it?
    - you can start anywhere you can do the best you can and there are a wide range of problems in the real world where kind of any solution will work, and you'd like to make it as good as possible and you know you

can't get to the optimal solution.



Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves □ But make them rarer as time goes on

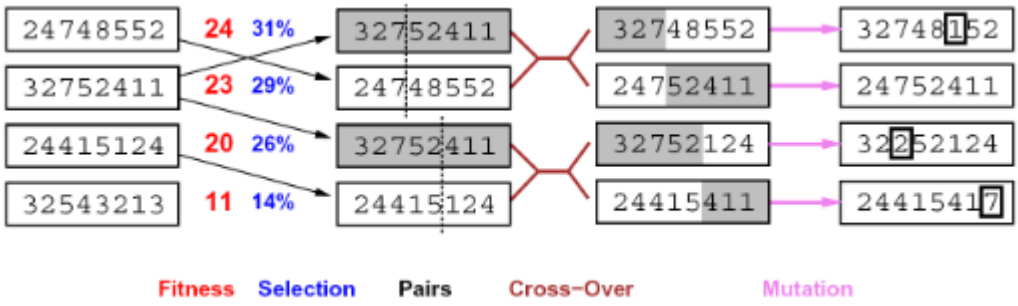
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

- Theoretical guarantee:
  - Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
  - So, while SA increases the likelihood of finding the global maximum compared to hill climbing, it does not provide an absolute guarantee. It does, however, offer a good balance between exploration and exploitation, making it a valuable tool in optimization problems.
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row.
  - People think hard about ridge operators which let you jump around the space in better way.

Genetic Algorithms

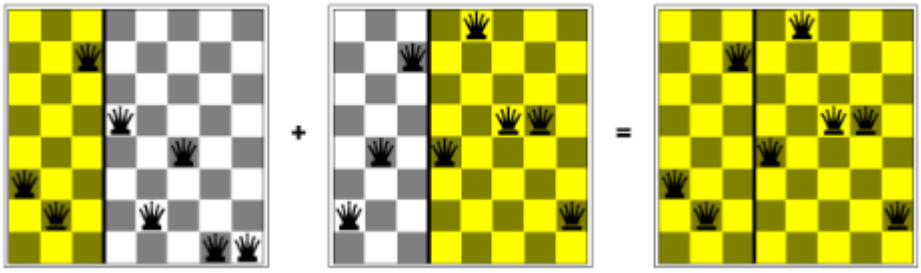
Genetic algorithms are kind of local search in this case not one hypothesis but a bunch of hypothesis and rather than just locally improving all of them, it is just mutation.

You keep the best hypotheses at each step. In addition to just keeping the best one, you find pairs and you do cross-over, you made them (2 robots -> big robot)



- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Example



I have 2 pretty good n-queen solutions. Neither is actually a solution. They both have a small number of conflicts.

So I'm going to slice my board down the middle, and I'm going to take part of a and part of b, and slam them together.

Why does this make sense ?

This is an example of not just nudging your thing locally, but just taking entirely different ways of traversing the space.

Lecture 3 Adversarial Search

Types of games

- Many different kinds of games
- Axes
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

We will talk about zero sum and deterministic games. They are games of perfect information.

Think about how this is different from search.

In search I gave you the search problem, and what you gave me back is a plan or path it is a sequences of actions that executed and it was guaranteed to succeed.

That's not going to work here because we don't control our opponent. So we can't just give a plan that guarantes to succeed. What we need to do is we need a function which tells us in any given state what to do. That is the **policy** in the game case it's often called strategy.

Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
    - Defines which player has the move in a state.
  - Actions: A (may depend on player / state)
    - Returns the set of legal moves in a state.
  - Transition Function: SxA → S
    - The result of a move depending on the given state.
  - Terminal Test: S →{t,f}
    - Which is true when the game is over and false otherwise.
    - States where the game has ended are called ***terminal states*** .
  - Terminal Utilities: SxP → R
    - Every outcome of the game will be scored.
    - Defines the final numeric value for a game that ends in terminal state ***s*** for a player ***p***
    - This tell us for an end-state how much it is worth to each of the players.
    - In chess, the outcome is a win, draw, or loss, with values +1, 0, or -1
- Solution for a player is a **policy**: S → A
  - the solution to a game like this is a policy which map states to actions.

Zero-Sum Games

Zero-Sum Games	**General Games**
Agents have opposite utilities (values on outcomes)	Agents have independent utilities (values on outcomes)
Lets us think of a single value that one maximizes and the other minimizes	Cooperation, indifference, competition, and more are all possible
Adversarial, pure competition	More later on non-zero-sum games

Adversarial Search

```
![[Pasted image 20240121230436.png]]
```

So instead of just taking actions with no consideration to the opponent, we will now make decisions based on what our opponent might or might not do.

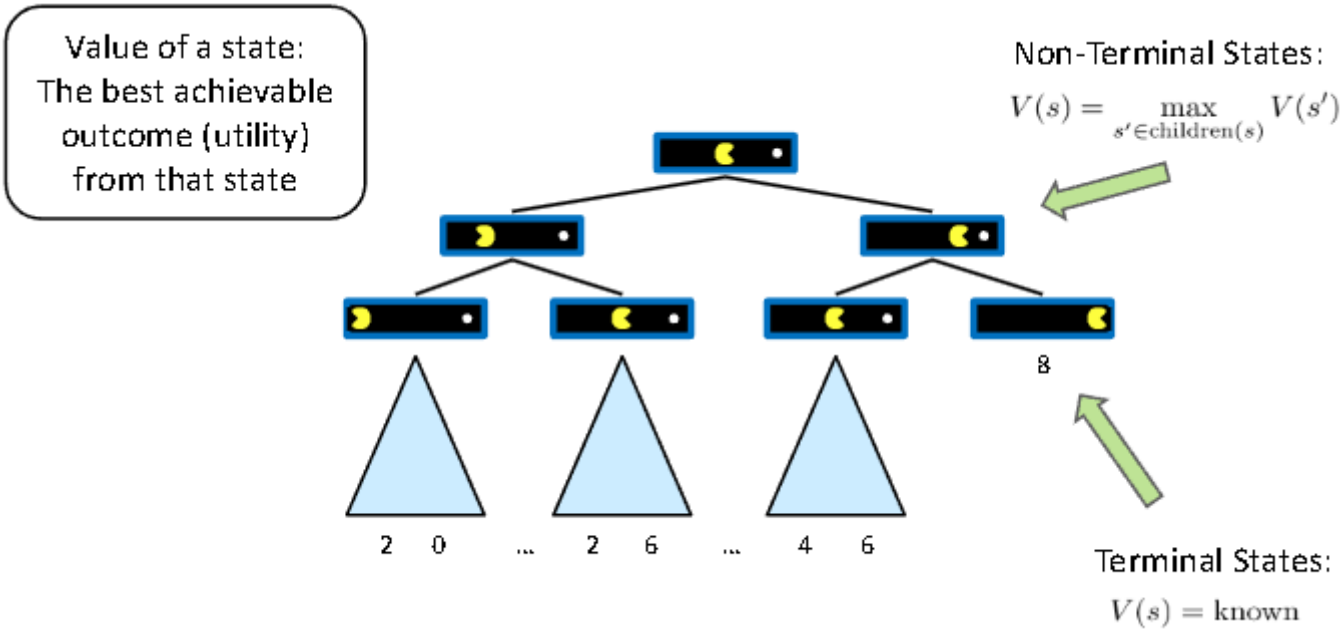
So for instance, if we look at the example of Pacman, we might need to consider what the ghost will do in case we take this direction of movement.

Single-Agent Trees

```
![[Pasted image 20240121230814.png]]
```

So when we look at a single agent, we can see that different movement leads to different outcomes of values.

What we want is the value 8, so for our agent we want the best possible outcome, and in this case it is the maximum.



Adversarial Game trees

```
![[Pasted image 20240121231123.png]]
```

But what if we have more than one agent, this we will be a tad bit complexer, because now we can't just move to our pellet just recklessly.

And how do we decide on how the opponent should move. So if our values are based on our movement, than the best possible move for the opponent is when the value of the state for us is the worst. So in other words the minimum of the states.

Minimax values

```
![[Pasted image 20240121231454.png]]
```

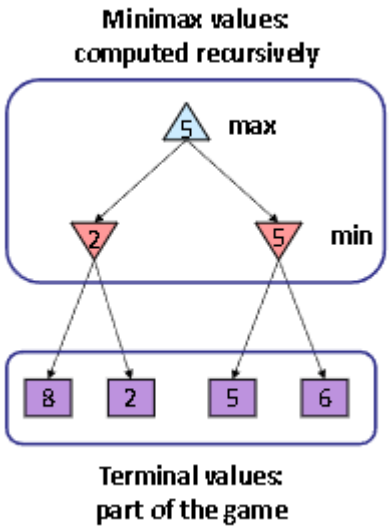
Here we can see a combination of us making a move and then the opponent making a move.

Example Tic-Tac-Toe Game

```
![[Pasted image 20240121231603.png]]
```

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary



Pseudocode

```
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
![[Pasted image 20240121231739.png]]
```

Properties

```
![[Pasted image 20240121232214.png]]
```

If you play against a perfect player you want to use minimax but if you are not playing against a player move random then minimax is going to be overly pessimistic. And so, if just periodically they were to make a mistake, it's worth going the right way (9-100). So now we're doing some probability calculation really. It's like what's the chances that they might make a mistake.

Efficiency

- How efficient is minimax?

    - Just like (exhaustive) DFS
    - Time: $O(b^m)$
    - Space: O(bm)
- Example: For chess, b≈35, m≈100
    - Exact solution is completely infeasible
    - But, do we need to explore the whole tree?

Game Tree Pruning

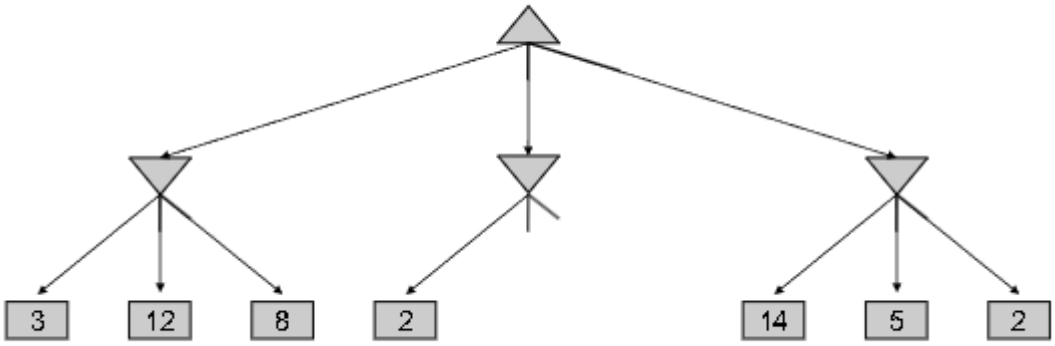
```
![[Pasted image 20240121232650.png]]
```

We will do some pruning because not all values are relevant in case we already found the minimum or maximum of the layer. So calculating different branches is not needed anymore.

Example

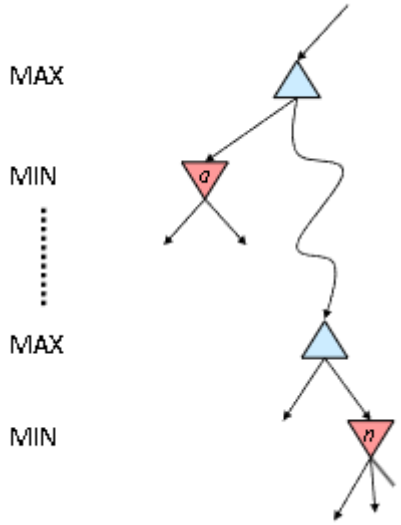
```
![[Pasted image 20240121232840.png]]
```

For the second branch (2-4-6), we need to look at the minimum of these values. We can already see that 2 is the minimum and branch 4 and 6, don't need to be explored further.



Alpha-Beta Pruning

- General configuration (MIN version)
    - We're computing the MIN-VALUE at some node n
    - We're looping over n's children
    - n's estimate of the children's min is dropping
    - Who cares about n's value? MAX
    - Let 'a' be the best value that MAX can get at any choice point along the current path from the root
    - If n becomes worse than a, MAX will avoid it, so we can stop considering n's other children (it's already bad enough that it won't be played)



- MAX version is symmetric

Pseudocode

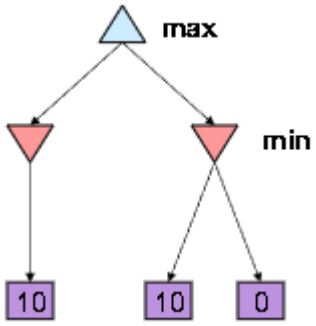
- α: MAX's best option on path to root
- β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        # top min-value not care what remains, if v > β
        if v > β return v  # must not prune on equality
        # update global max value
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        # top max-value not care what remains, if v < α
        if v < α return v # must not prune on equality
        # update global min value
        β = min(β, v)
    return v
```

Properties

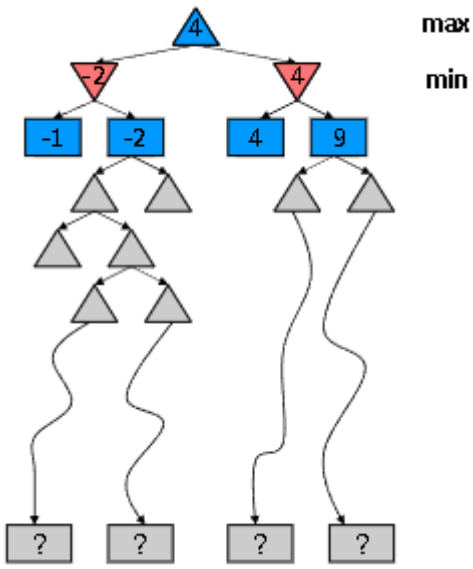
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to O(bm/2)
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of **metareasoning** (computing about what to compute)



Resource Limits

```
![[Pasted image 20240121232515.png]]
```

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is ***gone***
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



> For a chess game, we can't possibly search the whole game tree. Essentially we've got resource limits in this case time. That tell us we can only look forward so far into the tree before the exponential growth of the tree gets this.

So we can only search just some limited depth from the tree. Now the problem is we get to the end of our search we don't have terminal utilities because we are not actually at the end of the game.

So we need to replace the terminal utilities in the minimax algorithm with what's called evaluation function, which takes a non-terminal position and gives us some estimate of what the terminal utility under that tree would be under minimax plan.

Why Pacman Starves

```
![[Pasted image 20240121234443.png]]
```

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

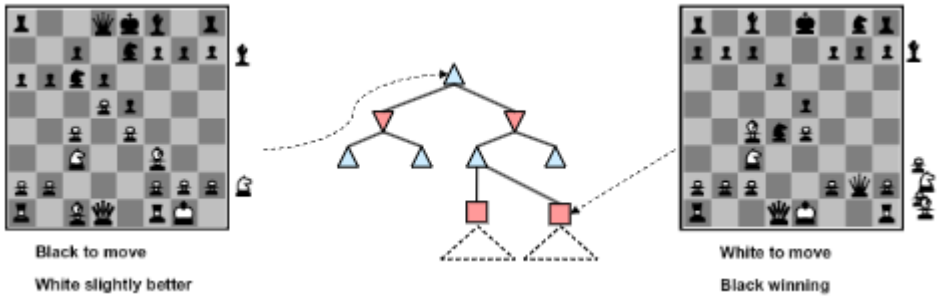
Evaluation Functions

```
![[Pasted image 20240121234613.png]]
```

A function takes a non-terminal state and return some number, just like the heuristic value in A* search.

In this case we want that number to return the actual minimax value of that position. That is not going to happen. In practice what people do is, they try to come up with some function which on average is positive when the minimax value is positive, is negative when the minimax value is negative.

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:
  - $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$
  - eg. $f_1(s)$=(number white queens – number black queens), etc.

Depth Matters

```
![[Pasted image 20240121235240.png]]
```

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the trade-off between complexity of features and complexity of computation

Synergies between Evaluation Function and Alpha-Beta?

- Alpha-Beta: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - somewhat similar to role of A* heuristic , CSPs filtering
- Alpha-Beta: (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune

- Hence: If evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root THEN can prune.

Uncertain Outcomes

```
![[Pasted image 20240121235641.png]]
```

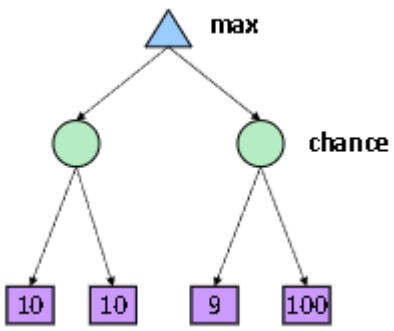
Worst Case vs Average Case

```
![[Pasted image 20240121235725.png]]
```

- Idea for today:
  - Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children
- Later, we'll learn how to formalize the underlying uncertain-result problems as **Markov Decision Processes**



Pseudocode

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX:
        return max-value(state)
    if the next agent is EXP:
        return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
```

```
        v = max(v, value(successor))
    return v
```

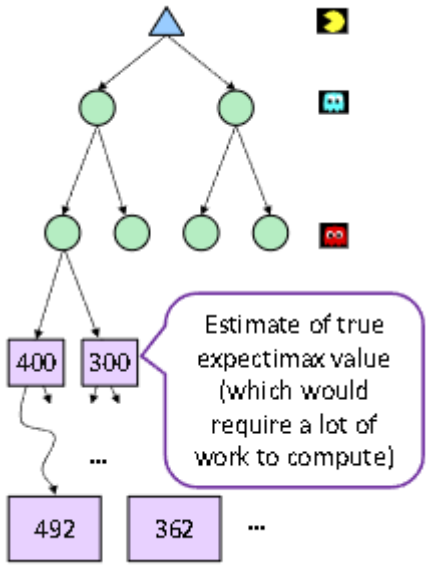
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

```
![[Pasted image 20240122000017.png]]
v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10
```

Expectimax Pruning?

Expectimax can not apply pruning. This is because we make use of the weighted average of every node, so every node is involved.

Depth-Limited Expectimax



Computing the full expectimax is computationally heavy, so we can limit it on the amount of layers we compute it on.

Probabilities

Reminder: Probabilities

```
![[Pasted image 20240122000418.png]]
```

- A **random variable** represents an event whose outcome is unknown
- A **probability distribution** is an assignment of weights to outcomes
- Example: Traffic on freeway
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}

- Distribution: P(T=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25


0.25


0.50

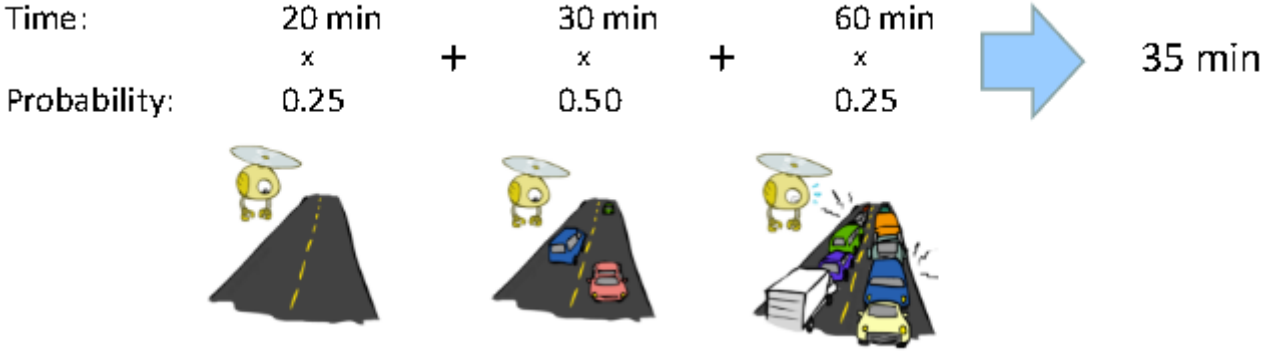

0.25

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one
- As we get more evidence, probabilities may change:
  - P(T=heavy) = 0.25, P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

Reminder: Expectations

- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes.

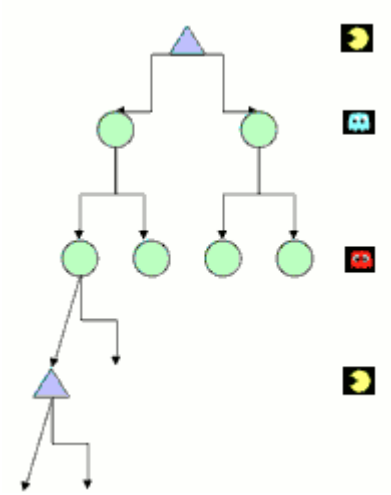
- Example: How long to get to an airport?



What probabilities to use?

- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment

    - The model might say that adversarial actions are likely!
- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes



One important thing to remember is that just because we assign probabilities that reflect our believes to the outcome, that does not mean that the thing on the other side of flipping a coin.

If I think there is a 20% chance that the ghost go to left , it doesn't mean that the ghost has a random number generator. It just means that given my model which may be a simplification that's the best I can say given my evidence.

Quiz: Informed Probabilities

- Let's say you know that your opponent is actually running a depth 2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search should you use?
- Answer: Expectimax!
    - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
    - This kind of thing gets very slow very quickly
    - Even worse if you have to simulate your opponent simulating you…
    - … except for minimax, which has the nice property that it all collapses into one game tree

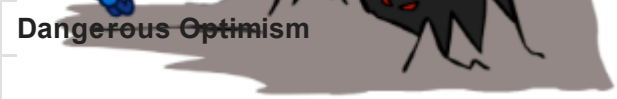
In general expectimax is the more general search procedures. You should always in principle use expectimax

Modelling Assumptions

```
![[Pasted image 20240122001500.png]]
```

The Dangers of Optimism and Pessimism

Dangerous Optimism	Dangerous Pessimism
Assuming chance when the world is adversarial.	Assuming the worst case when it's not likely.
	



Assumptions vs. Reality

```
![[Pasted image 20240122001718.png]]
```

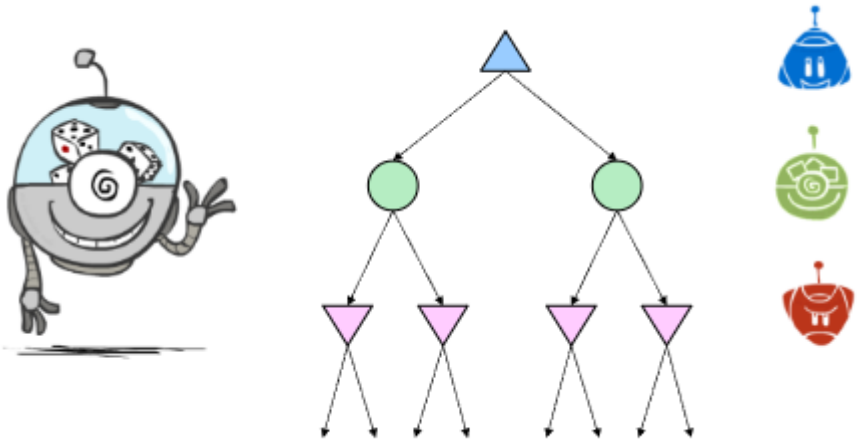
- Pacman used depth 4 search with an eval function that avoids trouble
- Ghost used depth 2 search with an eval function that seeks Pacman

Other Game Types

```
![[Pasted image 20240122001826.png]]
```

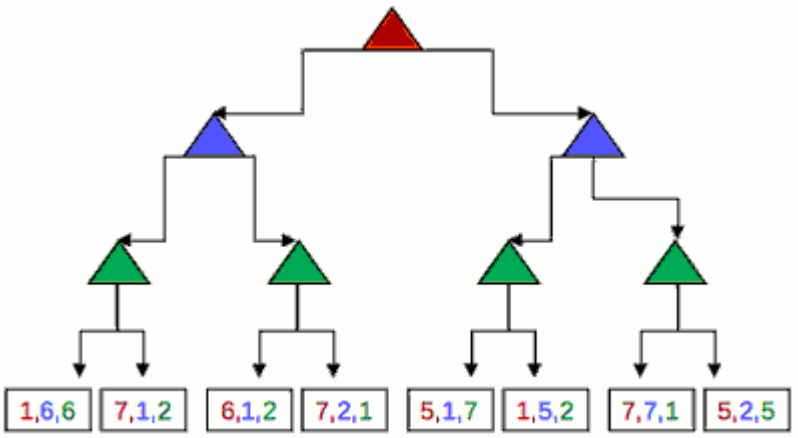
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…



> Each player has its own value of terminal node, and will optimize for their own outcome. Minimax , we kind of have that too. The maximizer had the number that we were showing, and the minimizer had the negative of that number. So there were actually 2 numbers sitting there, but it was the negative of each other, so we only showed one. Minimax is a special case of this where we just collapse those 2 opposite numbers into one number that we display.

The leaf utilities are now written as pairs (UA , UB, UC). In this generalized setting, A seeks to maximize UA, the first component, while B seeks to maximize UB , the second component.

In above example, the leftmost green node should be: ***(1,6,6)***

Different things can emerge here, though these numbers are not just complementary to each other. In the left sub tree, blue prefer to 6, can they make that happen? What will green do ? Green will choose the (1,6,6), which will happen that blue also gets 6 and gets what they want. So what we have here is actually a collaboration between blue and green.

Lecture 4 Bayes' Nets

> Bayes' Nets, also known as graphical models, which are a technique for building probabilistic models over large numbers of random variables, in a way that is efficient to specify and efficient to reason over.
>
> The development of AI stop in 80s because the difficulty of uncertainty. You can not compute such a large joint probability table with thousands variable.
>
> Bayes' Nets wil give us a way to deal with distributions of our large sets of random variable in a meaningful way

```
![[Pasted image 20240122002334.png]]
```

Probabilistic Models

- Models describe how (a portion of) the world works
- **Models are always simplifications**
  - May not account for every variable
  - May not account for all interactions between variables
  - “All models are wrong; but some are useful.” – George E. P. Box
- What do we do with probabilistic models?
  - We (or our agents) need to reason about unknown variables, given evidence
  - Example: explanation (diagnostic reasoning)
  - Example: prediction (causal reasoning)

- Example: value of information

Independence

```
![[Pasted image 20240122123030.png]]
```

- Two variables are independent if:
  - $\forall x, y : P(x, y) = P(x)P(y)$.
  - This says that their joint distribution factors into a product two simpler distributions.
  - Another form:
    - $\forall x, y : P(x|y) = P(x)$
    - This implies: the probability of x knowing y, is the probability of x. So knowing y doesn't change what the probability is over x.
  - We write this as:
    - $X \perp\!\!\!\perp Y$
- Independence is a simplifying modelling assumption
  - Empirical joint distributions: at best “close” to independent

Conditional Independence

```
![[Pasted image 20240122123814.png]]
```

- P(Toothache, Cavity, Catch)
- If I have a cavity, the probability that the probe catches in it doesn't depend on whether I have a toothache:
  - P(+catch | +toothache, +cavity) = P(+catch | +cavity)
- The same independence holds if I don’t have a cavity:
  - P(+catch | +toothache, -cavity) = P(+catch| -cavity)
- Catch is conditionally independent of Toothache given Cavity:
  - P(Catch | Toothache, Cavity) = P(Catch | Cavity)
- Equivalent statements:
  - P(Toothache | Catch , Cavity) = P(Toothache | Cavity)
  - P(Toothache, Catch | Cavity) = P(Toothache | Cavity) P(Catch | Cavity)
  - One can be derived from the other easily
- Unconditional (absolute) independence very rare (why?)
  - Unconditional independence is rare because most systems are complex with interrelated events. Conditional independence is more common because it allows for the possibility of relationships among events under certain conditions. It’s important to note that independence does not imply conditional independence, and vice versa.
- Conditional independence is our most basic and robust form of knowledge about uncertain environments.
- X is conditionally independent of Y given Z: $X \perp\!\!\!\perp Y|Z$.
  - if and only if:
    - $\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)$
  - or, equivalent, if and only if:
    - $\forall x, y, z : P(x|z, y) = P(x|z)$

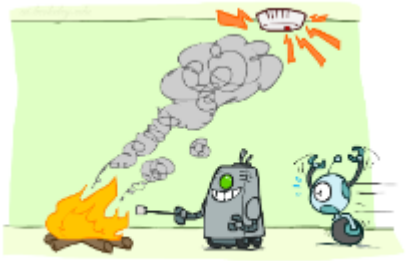
Examples

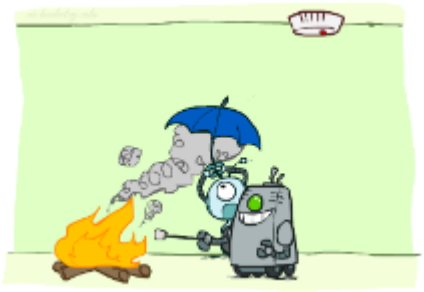
- What about this domain:
  - Traffic

- Umbrella
- Raining

We can see clearly from the graphic that rain causes both traffic and umbrellas. Therefore, traffic must be conditionally independent from umbrellas given that it's raining. More intuitively speaking, once we know it's raining the probability that there is traffic does not depend on whether people are walking around with umbrellas.

- What about this domain:
  - Fire
  - Smoke
  - Alarm





In this, we see that fire causes smoke which triggers the alarm. We can say that the alarm going off is conditionally independent of the fire given that there is smoke. In other words, if we know that there is smoke, whether or not that the smoke came from the fire does not affect the probability that the alarm will go off.

Chain rule

- Chain rule: $P(X_1, X_2, \cdots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)\cdots$
- Trivial decomposition:
  - $P(Traffic, Rain, Umbrella) = P(Rain)P(Traffic|Rain)P(Umbrella|Rain, Traffic)$
- With assumption of conditional independence:
  - $P(Traffic, Rain, Umbrella) = P(Rain)P(Traffic|Rain)P(Umbrella|Rain)$
- Bayes’nets / graphical models help us express conditional independence assumptions

Example

```
![[Pasted image 20240122130223.png]]![[Pasted image 20240122130304.png]]
```

In this case ,there are just 2 locations for the ghost , top or bottom.
So you could have a measurement on each of these squares so each of them could give you a red measurement or not a red measurement, so those are 2 random variables. The ghost's location is another random variable , whether it is being in the top .

- Each sensor depends only on where the ghost is
- That means, the two sensors are conditionally independent, given the ghost position
  - T: Top square is red:
    - B: Bottom square is red:
    - G: Ghost is in the top
- Givens:
  - P(+g) = 0.5
  - P(-g) = 0.5
  - P(+t | +g) = 0.8
  - P(+t | -g) = 0.4
  - P(+b | +g) = 0.4
  - P(+b | -g) = 0.8
- In general it is not enough to specify a full joint distribution
- But if we're willing to assume the 2 sensor are independent given where the ghost is , then this is enough.
  - P(G,B,T) = P(G)P(T|G)P(B|G)

T	B	G	P(T,B,G)
+t	+b	+g	0.16
+t	+b	-g	0.16
+t	-b	+g	0.24
+t	-b	-g	0.04
-t	+b	+g	0.04
-t	+b	-g	0.24
-t	-b	+g	0.06
-t	-b	-g	0.06

Bayes' Nets: Big Picture

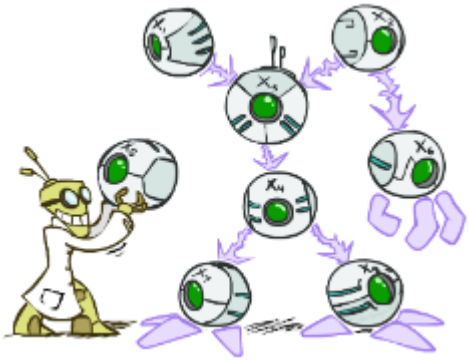
```
![[Pasted image 20240122130653.png]]
```

It's a new way of representing joint distributions. It's very closely related to the chain rule.

We've always talked about discrete random variables, and then a distribution can be a table. If you have continuous random variables then the way to represent the distribution is commonly done using a probability density function, which effectively encodes how much probability mass is associated with each interval if it's a 1D distribution, or each volume in space was probability of landing in a certain volume of space. It turns out if you had continuous random variables you can still use Bayes Nets, and the conditional distributions that you'd be working with would then be conditional distributions that are densities rather than the discrete distributions.

- Two problems with using full joint distribution tables as our probabilistic models:
  - Unless there are only a few variables, the joint is WAY too big to represent explicitly
  - Hard to learn (estimate) anything empirically about more than a few variables at a time
- **Bayes' nets:** a technique for describing complex joint distributions (models) using simple, local distributions (conditional probabilities)

- More properly called **graphical models**
- We describe how variables locally interact
- Local interactions chain together to give global, indirect interactions
- For about 10 min, we'll be vague about how these interactions are specified



Examples

Insurance

```
![[Pasted image 20240122131026.png]]
```

- 27 variables
  - assuming they are binary
- $2^{27} \approx 134M$ entries
- Difficult to estimate the relationships between all variables

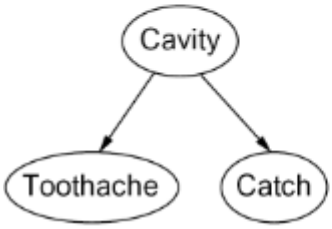
Car

```
![[Pasted image 20240122131035.png]]
```

Graphical Model Notation

- Nodes: variables (with domains)
  - Can be assigned (observed) or unassigned (unobserved)
    - Often the ones that are assigned we shade in with gray
- Arcs: interactions
  - Similar to CSP constraints
  - Indicate "direct influence" between variables
  - Formally: encode conditional independence (more later)
- For now: imagine that arrows mean direct causation (in general, they don't!)

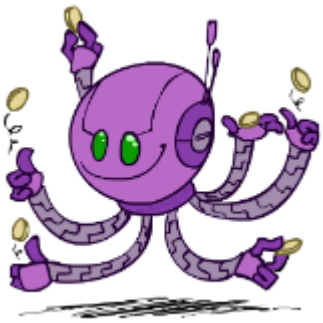
- it's often true, but it doesn't have to be true.



Examples

Coin flips

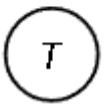
- N independent coin flips



- No interactions between variables: **absolute independence**

Traffic

- Variables:
  - R: It rains
  - T: There is traffic
- Model 1: independence

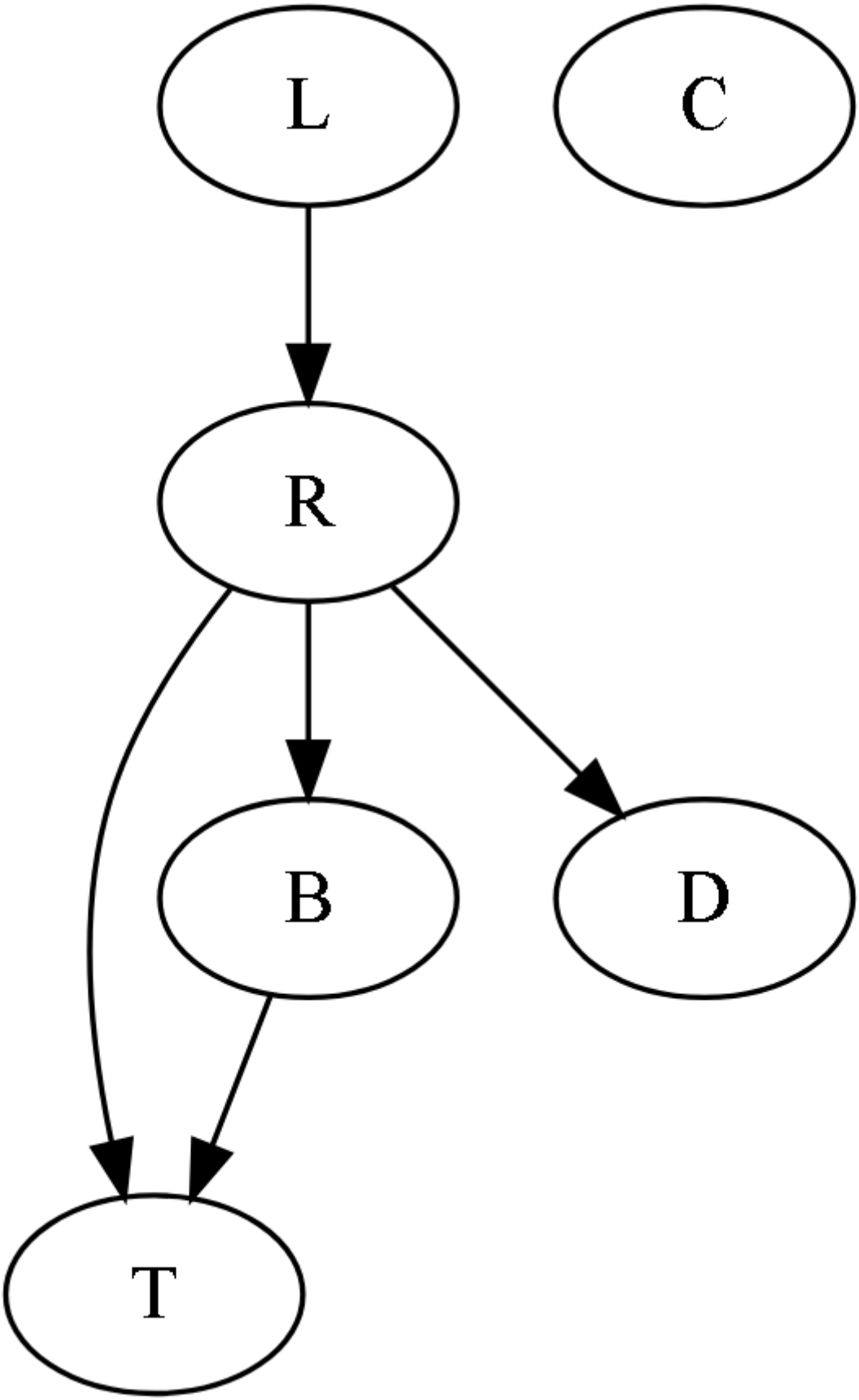


- Model 2: rain causes traffic

- In general model 2 is better, because we have an relationship. Relationships help with knowing what impact a certain variable has on an other variable.

Traffic 2

- Variables
  - T: Traffic
  - R: It rains
  - L: Low pressure
  - D: Roof drips
  - B: Ballgame
  - C: Cavity



A model is a representation of some thought a person has depending on the variables, so a different model could be valid depending on the reasoning.

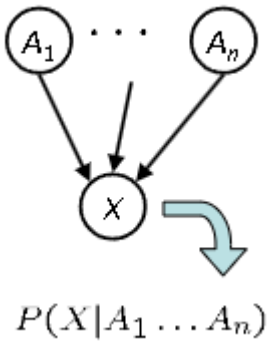
Bayes' Net Semantics

```
![[Pasted image 20240122133134.png]]
```

For a given Bayes' Net, what does it mean? What joint probability distribution does it encode? How do we know that?

- A set of nodes, one per variable X

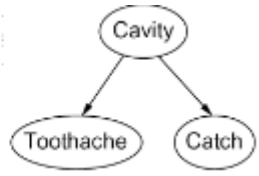
- A directed, acyclic graph
- A conditional distribution for each node



  - A collection of distributions over X, one for each combination of parents' values
    - $P(X|a_1, \cdots, a_n)$
  - CPT: conditional probability table
  - Description of a noisy "causal" process
    - E.g. if it rains, then there's 90% probability of traffic, if it doesn't rain, there's a 30% probability of traffic.
    - This is not part of the definition
- **A Bayes net = Topology (graph) + Local Conditional Probabilities**.
  - A graph, plus all the little local conditional probabilities that live inside the node

Probabilities in BNs

- Bayes' nets ***implicitly*** encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:
    - $P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$
  - Example



    - P(+cavity, +catch, -toothache) = P(+cavity)·P(-toothache|+cavity)·P(+catch|+cavity)
- Why are we guaranteed that setting $P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$ results in a proper joint distribution?
  - Chain rule (valid for all distributions): $P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|x_1, \cdots, x_{i-1})$
  - Assume conditional independences: $P(x_i, \cdots, x_{i-1}) = P(x_i|parents(X_i))$
    - Then consequence: $P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$
- Not every BN can represent every joint distribution
  - The topology enforces certain conditional independencies

Examples

Alarm Network

```
![[Pasted image 20240122134759.png]]
```

Traffic

```
![[Pasted image 20240122134822.png]]
```

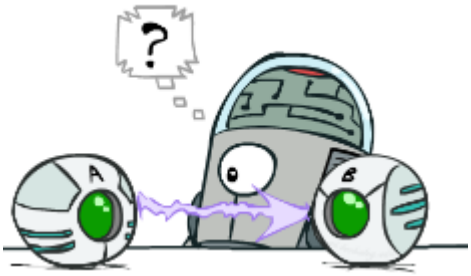
Reverse Traffic

```
![[Pasted image 20240122134843.png]]
```

This network here, which does not match the causal process, encodes the exact same joint distribution over those variables as the previous one. Now you might like the previous one better and there's a lot of advantages to drawing these things causally, but mathematically, it is just an expansion of the chain rule.

Causality?

- When Bayes' nets reflect the true causal patterns:
  - Often simpler (nodes have fewer parents)
    - OFTEN we choose them to be causal, and the reason we choose to be causal is that you'll end up with fewer parents.
  - Often easier to think about
  - Often easier to elicit from experts
    - $P(Traffic|Rain)$ is easy to get rather than $P(Rain|Traffic)$

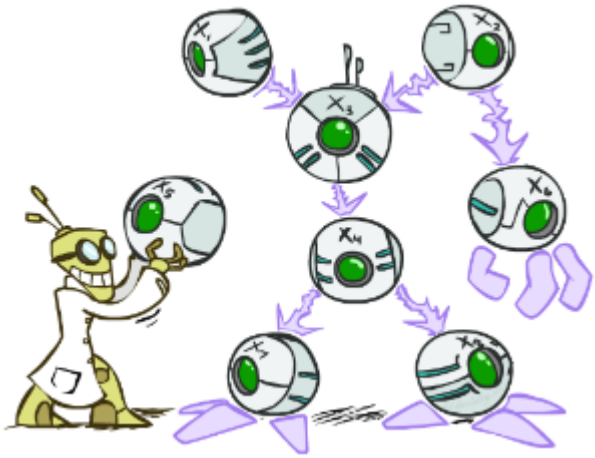


- BNs need not actually be causal
  - Sometimes no causal net exists over the domain (especially if variables are missing)
  - E.g. consider the variables Traffic and Drips
    - you don't care about Raining any more
    - T→D may be a reasonable choice, also D→R may be a reasonable choice. but both of them are not causal.
  - End up with arrows that reflect correlation, not causation
- What do the arrows really mean?
  - Topology may happen to encode causal structure
  - **Topology really encodes conditional independence**
    - $P(x_i, \cdots, x_{i-1}) = P(x_i|parents(X_i))$

Bayes' Nets

- So far: how a Bayes' net encodes a joint distribution
- Next: how to answer queries about that distribution
  - Today:
    - First assembled BNs using an intuitive notion of conditional independence as causality
    - Then saw that key property is conditional independence
  - Main goal: answer queries about conditional independence and influence

- After that: how to answer numerical queries (inference)



> Start of part two Bayes' Nets: Independence

Size of a Bayes' Net

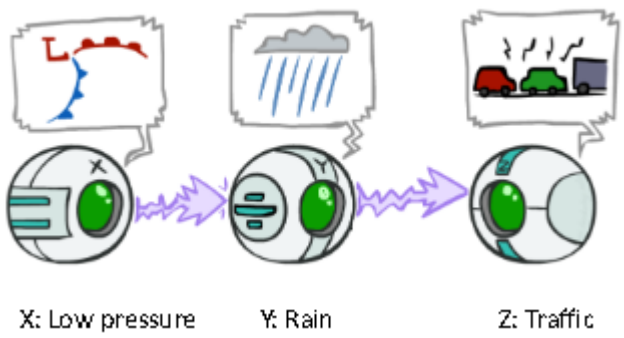
- How big is a joint distribution over N Boolean variables?
  - 2^N
- How big is an N-node net if nodes have up to k parents?
  - $O(N * 2^{k+1})$
- Both give you the power to calculate
  - $P(X_1, X_2, \cdots, X_n)$
- BNs: Huge space savings!
- Also easier to elicit local CPTs
- Also faster to answer queries (coming)

D-separation: Outline

- Study independence properties for triples
- Analyse complex cases in terms of member triples
- D-separation: a condition / algorithm for answering such queries

Causal Chains

- This configuration is a "causal chain"

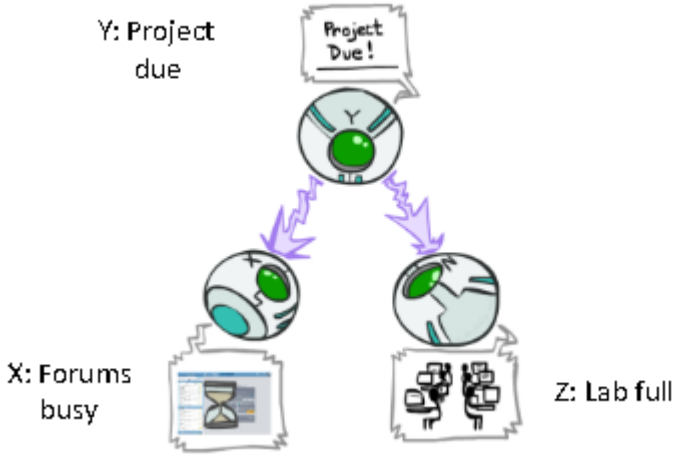


- $P(x, y, z) = P(x)P(y|x)P(z|y)$
- Guaranteed X independent of Z ? **NO**
  - One example set of CPTs for which X is not independent of Z is sufficient to show this independence is not guaranteed.
  - Example:
    - Low pressure causes rain causes traffic, high pressure causes no rain causes no traffic.

        - In numbers:
            - $P(+y|+x) = 1, P(-y|-x) = 1$
            - $P(+z|+y) = 1, P(-z|-y) = 1$
- Guaranteed X independent of Z given Y?
    - $P(z|x,y) = \frac{P(x,y,z)}{P(x,y)}$
    - $= \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)}$
    - $= P(z|y)$
    - So **YES**
- **Evidence along the chain "blocks" the influence**

Common Cause

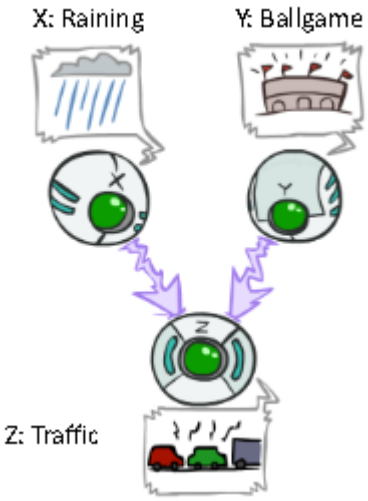
- This configuration is a "common cause"



- $P(x,y,z) = P(y)P(x|y)P(z|y)$
- Guaranteed X independent of Z ? **NO**
    - One example set of CPTs for which X is not independent of Z is sufficient to show this independence is not guaranteed.
    - Example:
        - Project due causes both forums busy and lab full
        - In numbers:
            - $P(+x|+y) = 1, P(-x|-y) = 1,$
            - $P(+z|+y) = 1, P(-z|-y) = 1$
- Guaranteed X and Z independent given Y?
    - $P(z|x,y) = \frac{P(x,y,z)}{P(x,y)}$
    - $= \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)}$
    - $= P(z|y)$
    - So **YES**
- **Observing the cause blocks influence between effects.**

Common Effect

- Last configuration: two causes of one effect (v-structures)

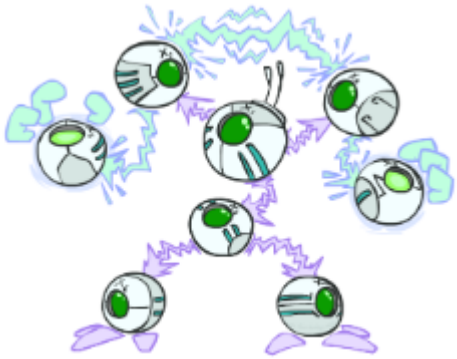


- Are X and Y independent?
  - **YES**: the ballgame and the rain cause traffic, but they are not correlated
    - $P(X, Y, Z) = P(X)P(Y)P(Z \mid X, Y)$
    - and $P(X, Y|Z) = \frac{P(X, Y, Z)}{P(Z)}$
    - Then if we combine them, $P(X, Y|Z) = \frac{P(X)P(Y)P(Z \mid X, Y)}{P(Z)}$
    - Z is conditionally independent of X and Y so $P(Z \mid X, Y) = P(Z)$
    - So $P(X, Y|Z) = \frac{P(X)P(Y)P(Z)}{P(Z)} = P(X)P(Z)$
- Are X and Y independent given Z?
  - **NO**: seeing traffic puts the rain and the ballgame in competition as explanation.
- **This is backwards from the other cases**
  - Observing an effect **activates** influence between possible causes.

The General Case

```
![[Pasted image 20240122161443.png]]
```

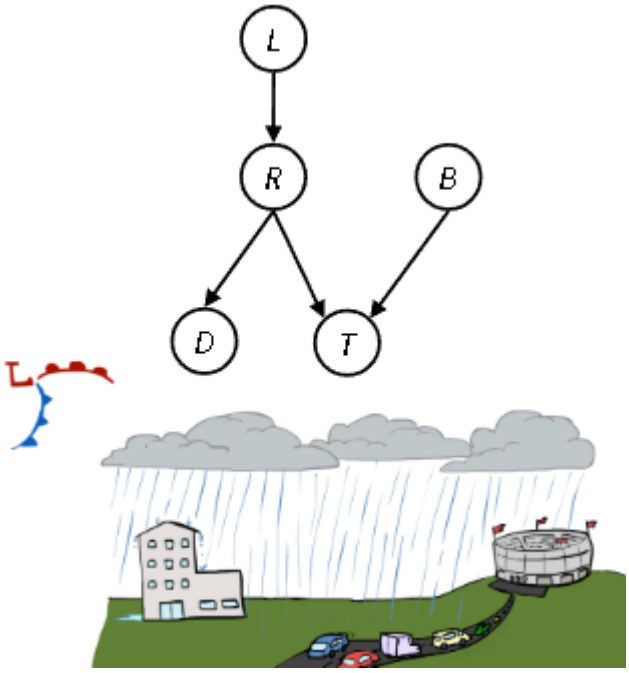
- General question: in a given BN, are two variables independent (given evidence)?
- Solution: analyse the graph
- Any complex example can be broken into repetitions of the three canonical cases



Reachability

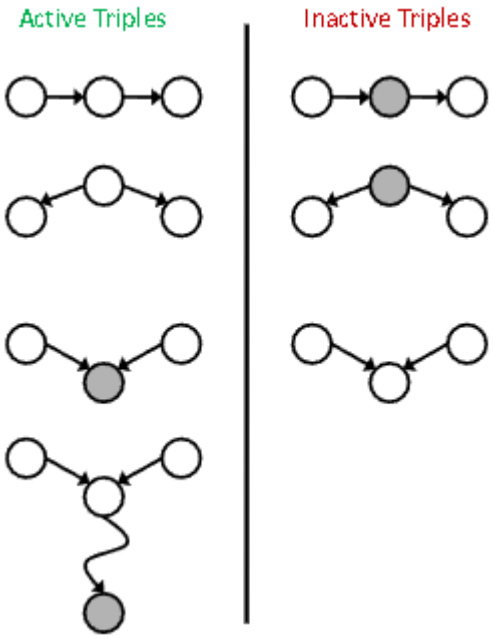
- Recipe: shade evidence nodes, look for paths in the resulting graph
- Attempt 1: if two nodes are connected by an undirected path not blocked by a shaded node, they are conditionally independent

- Almost works, but not quite
  - Where does it break?
    - Answer: the v-structure at T doesn’t count as a link in a path unless “active



Active / Inactive Paths

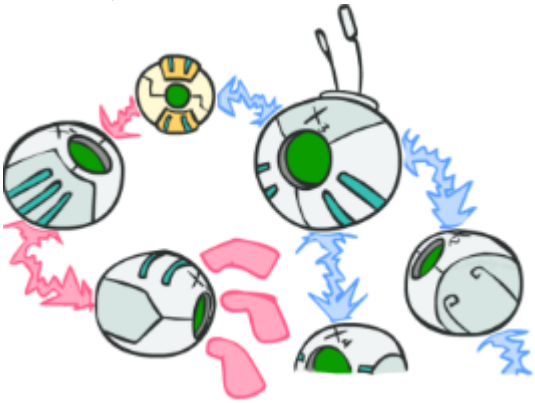
- Question: Are X and Y conditionally independent given evidence variables {Z}?
  - Yes, if X and Y "d-separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain A -> B -> C where B is unobserved (either direction)
  - Common cause A <- B -> C where B is unobserved
  - Common effect (aka v-structure) A -> B <- C where B or one of its *descendants* is observed
- All it takes to block a path is a single inactive segment



D-Separation

- Query: $X_i \perp\!\!\!\perp X_j | \{X_{k1}, \cdots, X_{kn}\}$?
- Check all (undirected!) paths between X_i and X_j

- If one or more active, then independence not guaranteed
  - $X_i \not\perp\!\!\!\perp X_j | \{X_{k1}, \cdots, X_{kn}\}$
- Otherwise (i.e. if all paths are inactive), then independence is guaranteed
  - $X_i \perp\!\!\!\perp X_j | \{X_{k1}, \cdots, X_{kn}\}$



Example

```
![[Pasted image 20240122162915.png]]
```

Structure Implications

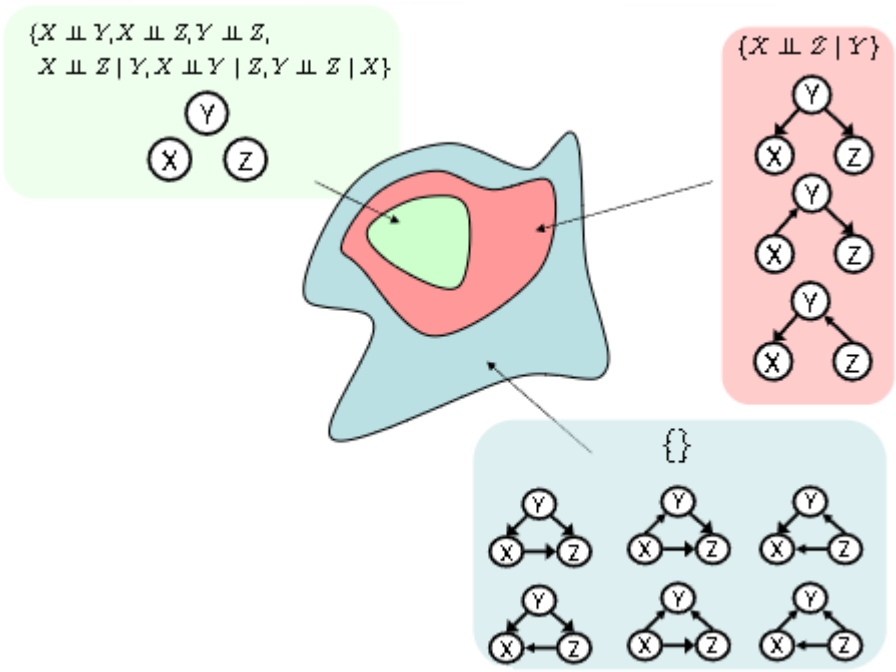
```
![[Pasted image 20240122163040.png]]
```

- Given a Bayes net structure, can run d- separation algorithm to build a complete list of conditional independences that are necessarily true of the form
  - $X_i \perp\!\!\!\perp X_j | \{X_{k1}, \cdots, X_{kn}\}$
- This list determines the set of probability distributions that can be represented

Topology Limits Distributions

- Given some graph topology G, only certain joint distributions can be encoded
- The graph structure guarantees certain (conditional) independences
- (There might be more independence)
- Adding arcs increases the set of distributions, but has several costs

- Full conditioning can encode any distribution



Summary

- Bayes nets compactly encode joint distributions
- Guaranteed independencies of distributions can be deduced from BN graph structure
- D-separation gives precise conditional independence guarantees from graph alone
- A Bayes' net's joint distribution may have further (conditional) independence that is not detectable until you inspect its specific distribution

---

Lecture 5 Bayes' Nets: Inference

```
![[Pasted image 20240122163724.png]]
```

Representation

- A directed, acyclic graph, one node per random variable
- A conditional probability table (CPT) for each node
  - A collection of distributions over X, one for each combination of parents' values
    - $P(X|a_1, \cdots, a_n)$
- Bayes' nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:
    - $P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$

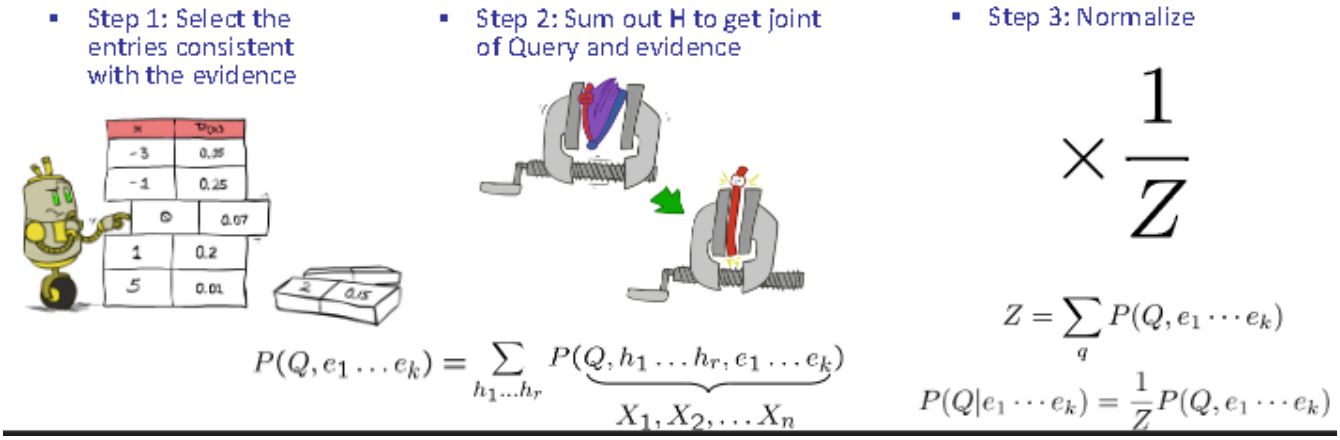
Inference

- Inference: calculating some useful quantity from a joint probability distribution
  - So you image the given is some collection of probabilities.
  - Maybe it's the whole joint probability table in all its exponential size glory. More often, it's a bunch of conditional probabilities that define a Bayes' Net.

    - Some examples of things you might calculate from those givens are the canonical thing is a posterior probability.
    - Another classic canonical query you might do is a most likely explanation query, where you say, I have some evidence, I would like to know the most likely value of a one or more variables given that evidence.
- Examples:
    - Posterior probability
        - P(Q| E_1=e_1,\cdots,E_k=e_k)
            - I care about variable Q. There's a bunch of evidence variables whose values I know and unfortunately there's going to be a bunch of other variables called hidden variables that I don't care about, and I also don't observe. And we're going to have to sum them out, and that creates a lot of time complexity.
    - Most likely explanation
        - argmax_q P(Q=q | E_1=e_1, \cdots)
            - Given some evidence has bee observed, what's the most likely association of some query variables.

Inference By Enumeration

- General case:
    - All variables: $X_1, X_2, \cdots, X_n$
        - Evidence variables: E_1,\cdots,E_k = e_1,\cdots,e_k
        - Query* variable: Q
        - Hidden variables: $H_1, \cdots, H_r$
- We want:
    - $P(Q|e_1, \cdots, e_k)$



Inference by Enumeration in Bayes' Net

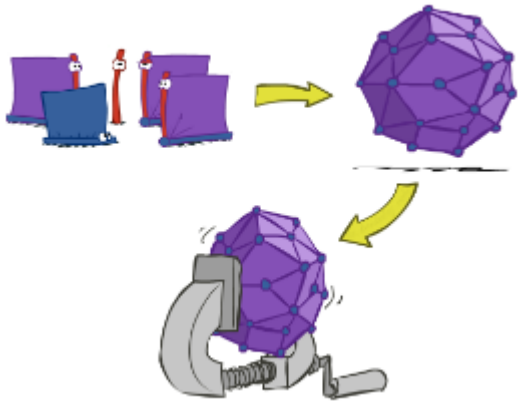
```
![[Pasted image 20240122170122.png]]
```

- Given unlimited time, inference in BNs is easy
- Reminder of inference by enumeration by example
    - $P(B|+j,+m)$
        - ∝_B P(B,+j,+m)
            - proportional to
            - this says these 2 things are almost equal, not actually equal.
            - but they'd be equal if we summed up the values over b, and multiply by the inverse.
            - And so what this says if you want a conditional probability, you compute the equivalent joint probability, and then normalize.

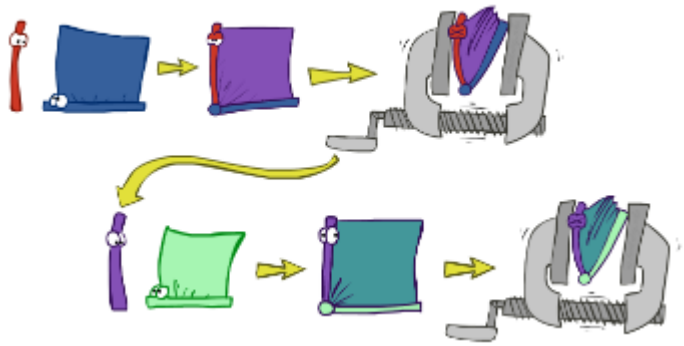
    - = ∑_{e,a} P(B,e,a,+j,+m)
        - now we can use the definition if the joint distribution as its specified through the Bayes net :
    - = ∑_{e,a} P(B)P(e)P(a|B,e)P(+j|a)P(+m|a)
    - = $P(B)P(+e)P(+a|B,+e)P(+j|+a)P(+m|+a) + P(B)P(+e)P(-a|B,+e)P(+j|-a)P(+m|-a) + P(B)P(-e)P(+a|B,-e)P(+j|+a)P(+m|+a) + P(B)P(-e)P(-a|B,-e)P(+j|-a)P(+m|-a)$
- f we have 100 variables in the BNs , 50 of them are evidence variables, means 50 of them not evidence variables that will look at 2^{50} entries
    - it gets very expensive, exponential in a number of non-evidence variables , unless almost everything is evidence
- It's not ideal to do it this way

Inference by Enumeration vs. Variable Elimination

- Why is inference by enumeration so slow?
    - You join up the whole joint distribution before you sum out the hidden variables.



- Idea: interleave joining and marginalizing!
    - Called “Variable Elimination”
    - Still NP-hard, but usually much faster than inference by enumeration



    - First we’ll need some new notation: factors

Factor Zoo

```
![[Pasted image 20240122170854.png]]
```

Factor Zoo 1

- Joint distribution: P(X,Y)
    - Entries P(x,y) for all x, y
    - Sums to 1
    - Example:

    - P (T,W)

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

- Selected joint: P(x,Y)
    - A slice of the joint distribution
    - Entries P(x,y) for fixed x, all y
    - Sums to P(x)
        - factors don't have to sum to 1
    - Example:
        - P(cold , W)

T	W	P
cold	sun	0.2
cold	rain	0.3

- **Number of capitals = dimensionality of the table**
    - P(T,W) is a 2D array
    - P(cold,W) is a 1D array
    - p(cold, sun), it's still a joint probability, but now it's a 1D object as a scalar.
    - That's important. We can reduce the size of data structure without changing the semantics of what's in it.
- So as we work in this variable elimination process, the game will be one of trying to keep the number of capitalized variables small in our factor.

Factor Zoo 2

- Single conditional: P(Y | x)
    - Entries P(y | x) for fixed x, all y
    - Sums to 1
    - Example:
        - P(W|cold)

T	W	P
cold	sun	0.4
cold	rain	0.6

- Family of conditionals: P(X |Y)
    - Multiple conditionals
    - Entries P(x | y) for all x, y
    - Sums to |Y|
    - A little weird. This is distributions over Y, but it's one for every value of x , not a fixed x. It's got a value for every value of x and every value for y, If you add them up, each little distribution over Y adds up to 1, and there are many of x. So this whole thing together sums to more than 1.
    - Example:
        - P(W|T)

T	W	P
hot	sun	0.8
hot	rain	0.2
cold	sun	0.4
cold	rain	0.6
First two rows result in P(W	hot)	
Last two rows result in P(W	cold)	

Factor Zoo 3

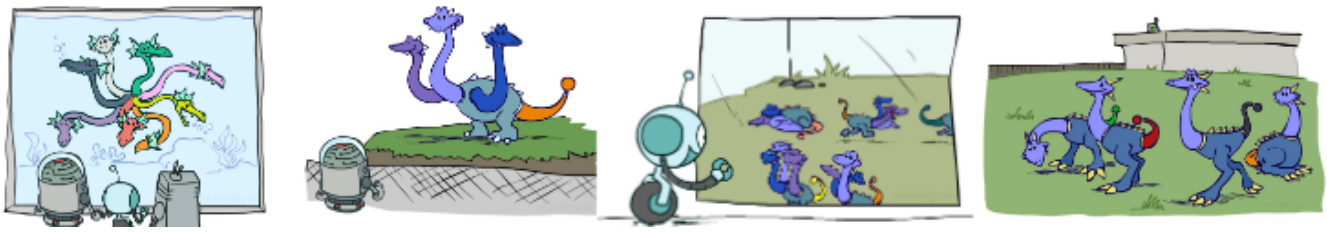
- Specified family: P(y | X)
  - Entries P(y | x) for fixed y, but for all x
  - Sums to … who knows!
  - Example:
    - P(rain | T)

T	W	P
hot	rain	0.2
cold	rain	0.6

- It's not a distribution over rain and sun, it's a bunch of probability of rains.
  - This is no longer a distribution. It's no longer a family of distributions. It's a 1D array, each of those entries is the conditional probability of some particular value for Y, which is usually an evidence value.
- How do we get these things. We get these things because somebody hands us little pieces of Bayes' Net. We're going to selecting the value of the evidence, and then we're going to start multiplying things together. So we're going to get all kinds of stuff.

Summary

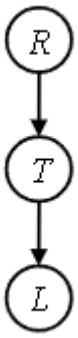
- In general, when we write P(Y_1, \cdots, Y_N | X_1, \cdots, X_M)
  - It is a “factor,” a multi-dimensional array
  - Its values are P(y_1 … yN | x_1 … xM)
  - Any assigned (=lower-case) X or Y is a dimension missing (selected) from the array



Example: Traffic Domain

- Random Variables
  - R: Raining
  - T: Traffic

- L: Late for class!



$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- query: P(L) = ?
  - for inference enumeration:
  - P(L) = ∑_{r,t} P(r,t,L)
  - = ∑_{r,t} P(r)P(t|r)P(L|t)

Inference by Enumeration: Procedural Outline

- Track objects called factors
- Initial factors are local CPTs (one per node)

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- Any known values are selected
  - E.g. if we know L = +l , the initial factors are

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(+\ell|T)$

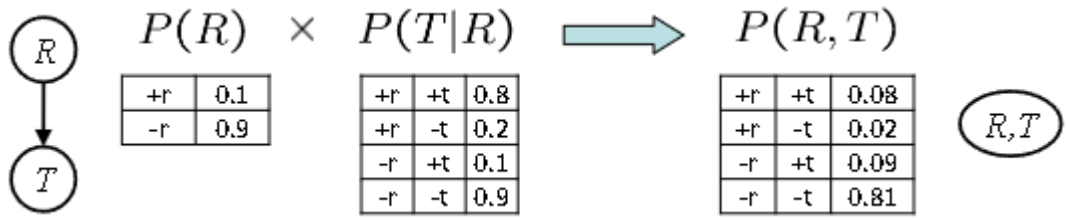
+t	+l	0.3
-t	+l	0.1

- Procedure: Join all factors, then eliminate all hidden variables

Operation 1: Join Factors

```
![[Pasted image 20240122172935.png]]
```

- First basic operation: ***joining factors***
- Combining factors:
  - Just like a ***database join***
  - Get all factors over the joining variable
  - Build a new factor over the union of the variables involved
- Example: Join on R



- Computation for each entry: pointwise products
  - ∀r,t : P(r,t) = P(r)·P(t|r)

Example: Multiple Joins

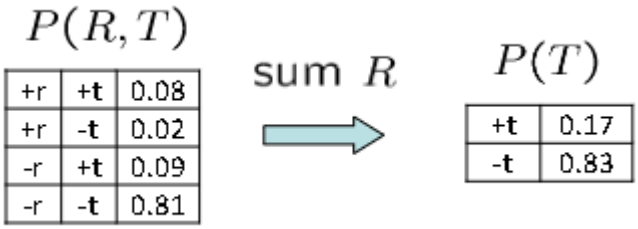
```
![[Pasted image 20240122173201.png]]
![[Pasted image 20240122173212.png]]
```

- We call "Join on R" means that you grab all tables that have `R` in them.

Operation 2: Eliminate

```
![[Pasted image 20240122173650.png]]
```

- Second basic operation: ***marginalization***
- Take a factor and sum out a variable
  - Shrinks a factor to a smaller one
  - A ***projection*** operation
    - Get rid of the variables that don't matter -- the hidden variables
    - Why do we even have hidden variables ?
      - The reason we have it because we started with a joint distribution that was over more than the variables that appear in our query.
- Example:



- We sum all columns with the same T, so P(+r, +t) + P(-r, +t) = P(+t)

Example: Multiple Elimination

```
![[Pasted image 20240122173916.png]]
```

Thus Far: Multiple Join, Multiple Eliminate (= Inference by Enumeration)

```
![[Pasted image 20240122173959.png]]
```

Marginalizing Early (= Variable Elimination)

```
![[Pasted image 20240122174047.png]]
```

- Switch the some order of join/eliminate
- Intuition
  - if you want to eliminate a variable, you can not do this until you have joined on that variable.

Traffic Domain

```
![[Pasted image 20240122174204.png]]
```

- P(L) = ?
- Inference by Enumeration

$$= \sum_t \sum_r \underbrace{P(L|t)P(r)P(t|r)}_{\text{Join on r}}$$

$$\underbrace{\qquad\qquad}_{\text{Join on t}}$$

$$\underbrace{\qquad\qquad}_{\text{Eliminate r}}$$

$$\underbrace{\qquad\qquad}_{\text{Eliminate t}}$$

- Variable Elimination

$$= \sum_t P(L|t) \sum_r \underbrace{P(r)P(t|r)}_{\text{Join on r}}$$

$$\underbrace{\qquad\qquad}_{\text{Eliminate r}}$$

$$\underbrace{\qquad\qquad}_{\text{Join on t}}$$

$$\underbrace{\qquad\qquad}_{\text{Eliminate t}}$$

Marginalizing Early! (aka VE)

```
![[Pasted image 20240122174439.png]]
```

Evidence

- If evidence, start with factors that select that evidence

- No evidence looks like this: uses these initial factors

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- with evidence looks like this: computing P(L|+r) the initial factors become:

$P(+r)$

+r	0.1

$P(T|+r)$

+r	+t	0.8
+r	-t	0.2

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- We eliminate all vars other than query + evidence
- Result will be a selected joint of query and evidence
  - E.g. for P(L | +r), we would end up with:

$P(+r, L)$

+r	+l	0.026
+r	-l	0.074

Normalize →

$P(L|+r)$

+l	0.26
-l	0.74

    - So P(L | +r) = P(+r, L) / P(+r)
- To get our answer, just normalize this!
- That's it!

General Variable Elimination

- Query: P(Q| E_1=e_1,\cdots,E_k=e_k)
- Start with initial factors:
  - Local CPTs (but instantiated by evidence)
- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H
    - any ordering of hidden variables is valid
    - but some orderings will lead to very big factors being generated along the way
    - and some orderings might be able to keep the factors generated along the way very small
  - Join all factors mentioning H
  - Eliminate (sum out) H
- Join all remaining factors and normalize

Example

```
![[Pasted image 20240122182030.png]]
![[Pasted image 20240122182428.png]]
![[Pasted image 20240122182435.png]]
```

Example in Equations

```
![[Pasted image 20240122182142.png]]
```

- Do not process the slide! Don't do it. The slide is simply saying that instead of working with these factors, I could have worked all that out using products, and sums, and laws of distribution, and all of that.

- equations from top to bottom
  1. marginal can be obtained from joint by summing out
  2. use Bayes' net joint distribution expression
  3. use `x*(y+z) = xy + xz`
  4. joining on *a* , and then summing out give f_1
  5. use `x*(y+z) = xy + xz`
  6. joining on *e* , and then summing out give f_2
- **All we are doing is exploiting uwy + uwz + uxy + uxz + vwy + vwz + vxy +vxz = (u+v)(w+x)(y+z) to improve computational efficiency** !
- how do you decide which variables to pick first ?
  - Suggestion here was **a variable with very few connections** . Connections means that it is participating in a factor.

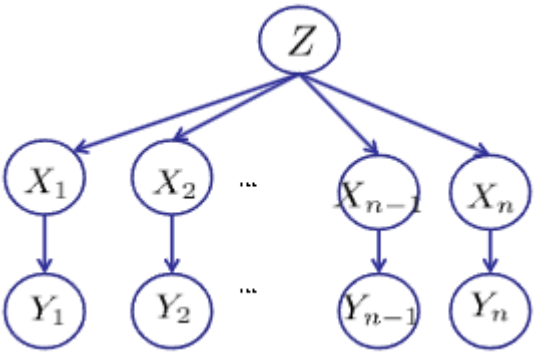
Another Variable Elimination Example

```
![[Pasted image 20240122182917.png]]
```

- Query: $P(X_3|Y_1=y_1,Y_2=y_2,Y_3=y_3)$
- Start by inserting evidence , which gives the following initial factors:
  - $p(Z),p(X_1|Z),p(X_2|Z),p(X_3|Z),p(y_1|X_1),p(y_2|X_2),p(y_3|X_3)$
- Eliminate X_1, this introduce the factor $f_1(Z, y_1) = \sum_{x1} p(x_1|Z)p(y_1|x_1)$, and we are left with
  - $p(Z),f_1(Z,y_1),p(X_2|Z),p(X_3|Z),p(y_2|X_2),p(y_3|X_3)$
- Eliminate X_2, this introduce the factor $f_2(Z,y_2) = \sum_{x2} p(x_2|Z)p(y_2|x_2)$, and we are left with
  - $p(Z),f_1(Z,y_1),f_2(Z,y_2),p(X_3|Z),p(y_3|X_3)$
- Eliminate Z, this introduces the factor $f_3(y_1,y_2,X_3) = \sum_z p(z)f_1(Z,y_1)f_2(Z,y_2)p(X_3|z)$, and we are left:
  - $p(y_3|X_3),f_3(y_1,y_2,X_3)$
  - quiz: why there is a factor $p(X_3|z)$? why not the other ones , like $p(X_3,z)$ or $P(Z,y)$?
    - X_3 is the query variable, the query variable doesn't go through the elimination process, the same way as a hidden variable
- No hidden variables left. Join the remaining factors to get
  - $f_4(y_1,y_2,y_3, X_3) = p(y_3|X_3)\cdot f_3(y_1,y_2,X_3)$
- Normalizing over X_3 gives $P(X_3|y_1,y_2,y_3)$

Variable Elimination Ordering

- For the query $P(Xn|y_1,\ldots,yn)$ work through the following two different orderings as done in previous slide: Z, X_1, …, Xn-1 and X_1, …, Xn-1, Z. What is the size of the maximum factor generated for each of the orderings?



- Answer: 2^{n+1} versus 2^2 (assuming binary)
- In general: the ordering can greatly affect efficiency.

VE: Computational and Space Complexity

- The computational and space complexity of variable elimination is determined by the largest factor
- The elimination ordering can greatly affect the size of the largest factor.
  - E.g., previous slide's example 2^n vs. 2
- Does there always exist an ordering that only results in small factors?
  - **No!**
  - There are BNs and we'll see one in next slides where no matter which ordering you pick it's going to generate large factors along the way.

Worst Case Complexity?

```
![[Pasted image 20240122183616.png]]
```

- CPS
  - A 3-Sat problem, a special kind of CSP

    $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_2 \vee x_4) \wedge (\neg x_3 \vee \neg x_4 \vee \neg x_5) \wedge (x_2 \vee x_5 \vee x_7) \wedge (x_4 \vee x_5 \vee x_6) \wedge (\neg x_5 \vee x_6 \vee \neg x_7) \wedge (\neg x_5 \vee \neg x_6 \vee x_7)$

  - There are 7 variables , X_1 ... X_7, I want to find an assignment to these 7 variables , such that this clause is true
    - The clause is saying : x_1 or x_2 or not x_3 has to be true , and not x_1 or x_3 or not x_4 has to be true , and ... so forth
  - $P(X_i=0) = P(X_i=1) = 0.5$
  - $Y_1 = X_1 \vee X_2 \vee \neg V_3$, ... , $Y_8 = \neg X_5 \vee X_6 \vee V_7$
  - $Y_{1,2} = Y_1 \wedge Y_2$, ... , $Y_{7,8} = Y_7 \wedge Y_8$
  - $Y_{1,2,3,4} = Y_{1,2} \wedge Y_{3,4}$, $Y_{5,6,7,8} = Y_{5,6} \wedge Y_{7,8}$
  - $Z = Y_{1,2,3,4} \wedge Y_{5,6,7,8}$
- Why we use so many Y variables here ?
  - BNs where variable has many parents is a large BNs , it's not very compact
  - Those Y variables gives us a BNs where every variables has at most 3 parents. So it's a very small BNs.
- If we can answer P(z) equal to zero or not, we answered whether the 3-SAT problem has a solution.
- Hence inference in Bayes' nets is NP-hard. No known efficient probabilistic inference in general.

Polytrees

There are special graph structures of BNs, where inference can be done efficiently. One example is Polytree.

- A polytree is a directed graph with no undirected cycles
  - so directed acyclic graph can have undirected cycles.
  - but if you don't allow those undirected cycles, you have a polytree.
- For polytrees you can always find an ordering that is efficient
  - Try it!!
- Cut-set conditioning for Bayes' net inference
  - Choose set of variables such that if removed only a polytree remains
  - Exercise: Think about how the specifics would work out!

> Start of part 2 Bayes' Nets: Sampling

Approximate Inference: Sampling

```
![[Pasted image 20240122184207.png]]
```

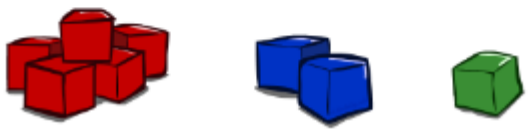
Sampling

- Sampling is a lot like repeated simulation
  - Predicting the weather, basketball games, …
- Why sample?
  - Learning: get samples from a distribution you don’t know
  - Inference: getting a sample is faster than computing the right answer (e.g. with variable elimination)
    - We're going to go to our Bayes net, in which we know all of the probabilities. But we're going to sample anyway.
    - When you sample in a network that you already know, it looks like simulation. You walk along the network, and you say, hmm if this happened, let's see what would happen at this variable. And you slip some coins. And you get a sample out that is a sort of probabilistic simulation.
    - And in this case, the reason you're getting a sample is not because you don't actually know or are not able to compute the underlying probabilities. It's because sampling turns out to be faster than computing the right answer through brute force.
- Basic idea
  - Draw N samples from a sampling distribution S
    - we get to define the sampling distribution.
  - Compute an approximate posterior probability
    - Inside those samples, which are all events that look like outcomes of S, and you're going to compute an approximate posterior probability. Whatever query you're trying to answer, you'll compute it over your samples.
  - Show this converges to the true probability P
- Sampling from given distribution
  - Step 1: Get sample *u* from uniform distribution over [0, 1)
    - E.g. random() in python
  - Step 2: Convert this sample *u* that lies in [0,1) interval into an outcome from the distribution that we want a sample from
- Example:

C	P(C)
red	0.6
green	0.1
blue	0.3

$$0 \leq u < 0.6, \rightarrow C = red$$
$$0.6 \leq u < 0.7, \rightarrow C = green$$
$$0.7 \leq u < 1, \rightarrow C = blue$$

  - If random() returns $u = 0.83$, then our sample is C = blue
  - E.g, after sampling 8 times:

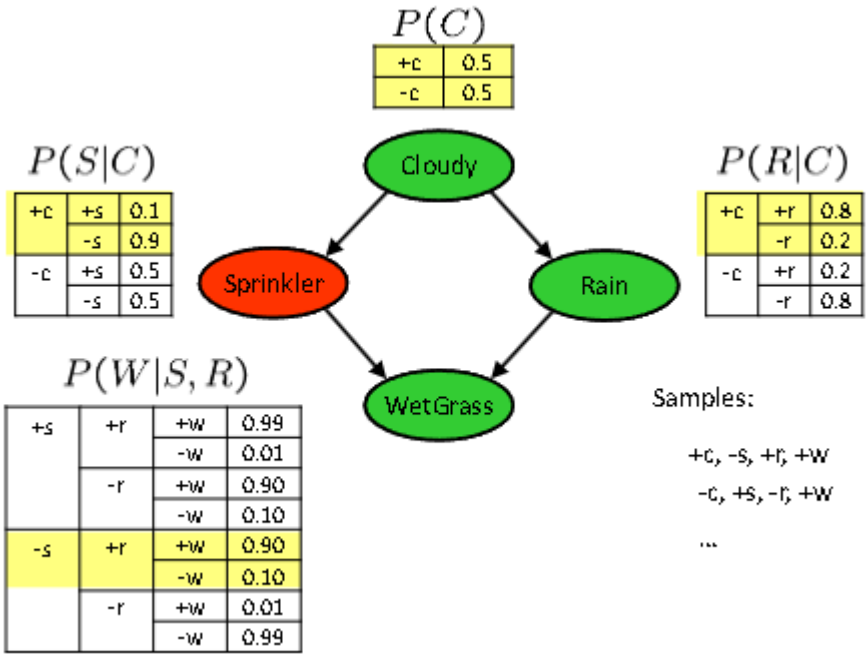


Sampling in Bayes’ Nets

Prior Sampling

```
![[Pasted image 20240122191714.png]]
```

We have a BNs, and we want to generate samples of the full joint distribution, but without building full joint distribution.



Given time, I know how to create the whole joint distribution. So I could do that right now. I could ask you, hey, what's the probability that it's cloudy, there's a sprinkler, it's rainy, but the grass is dry? We can compute that right now by multiplying a bunch of conditional probabilities together.

We're going to do something similar to that. Instead of computing an entry of the joint distribution, we are going to create an event which is an assignment to these 4 variables. But we're going to create it by walking along the Bayes net.

- Ordering: C → S → R → W , or C → R → S → W
- We start with the 1st variable in the ordering **C**
  - we generate the number in `[0,1)` , then map it to +c , or -c.
  - this case , we got +c
- Then we pick up next variable , this case *S*
  - C is already sampled as +c, so we just consider the highlighted part of this table.
  - again we generate a number, and map it into either +s , or -s
  - this case , we got -s
- Next we have to proceed with R
  - we can not proceed W yet, because W depends on both S and R
  - this case , we got +r
- Now we can sample W
  - this case , we got +w
  - this generated our first sample from this BNs. +c,-s,+r,+w
- Repeat this process , and build up a set of samples from this BNs distribution here.
- What's the probability that I'll get the sample +c,-s,+r,+w
  - 0.5 *0.9* 0.8 * 0.9
  - If I multiply those together, I can determine the probability of the sample, which is just the probability of this event in the distribution described by the Bayes net.

Pseudocode

```
// single sample
For i=1,2,...,n
```

```
    Sample xi from P(Xi| Parents(Xi))
return (x1,x2,...,xn)
```

- This process generates samples with probability:

$$S_{PS}(x_1 \dots x_n) = \prod_{i=1}^{n} P(x_i|\text{Parents}(X_i)) = P(x_1 \dots x_n)$$

  - S_{PS}: sampling distribution *S* using prior sampling
  - i.e. the BN's joint probability
- Let the number of samples of an event be
  - N_{PS} (x1,x2,...,xn)
- If the number of samples goes to infinity then the number of samples you get for a particular event divided by N will converge to the actual probability for that event.

$$\begin{aligned} \lim_{N\to\infty} \hat{P}(x_1, \dots, x_n) &= \lim_{N\to\infty} N_{PS}(x_1, \dots, x_n)/N \\ &= S_{PS}(x_1, \dots, x_n) \\ &= P(x_1 \dots x_n) \end{aligned}$$

- I.e., the sampling procedure is ***consistent***.

Example

```
![[Pasted image 20240122192641.png]]
```

- We'll get a bunch of samples from the BN: (let's say we ended up with following samples)
  - +c, -s, +r, +w
  - +c, +s, +r, +w
  - -c, +s, +r, -w
  - +c, -s, +r, +w
  - -c, -s, -r, +w
- If we want to know P(W)
  - We have counts : <+w:4 , -w:1>
  - Normalize to get P(W) = <+w:0.8, -w:0.2>
  - This will get closer to the true distribution with more samples
  - Can estimate anything else, too
  - What about P(C| +w)? P(C| +r, +w)? P(C| -r, -w)?
    - P(+c|+w) = 3/4 , P(-c|+w) = 1/4
    - P(+c|+r,+w) = 1
    - P(-c|-r,-w) : not computable from the samples we have.
  - Fast: can use fewer samples if less time
    - what's the drawback? the accuracy of course.

Rejection Sampling

```
![[Pasted image 20240122192851.png]]
```

As the samples come off our conveyor belt, we take a look at them. And if they're the samples we can use, meaning they match our evidence, we keep them. And if they're samples we can't use, we reject them.

- Let's say we want P(C)

    - No point keeping all samples around
    - Just tally counts of C as we go
        - we know we sampled top down through BNs, so we know once we sampled C and if later all interesting is counting how often we have +c/-c
        - there's no need to still sample S, R and W.
        - we just stop sampling after we got C.
- Let’s say we want P(C| +s)
    - Same thing: tally C outcomes, but ignore (reject) samples which don’t have S=+s
        - when you sampled -s , there is no point in continuing.
        - that sample is going to be going unused when you answer your query
    - This is called rejection sampling
    - It is also consistent for conditional probabilities (i.e., correct in the limit)

Pseudocode

```
// single sample
IN: evidence instantiation
For i=1, 2, …, n
    Sample x_i from P(X_i| Parents(X_i))
    If x_i not consistent with evidence
        Reject: Return, and no sample is generated in this cycle
Return (x1, x2, …, xn)
```

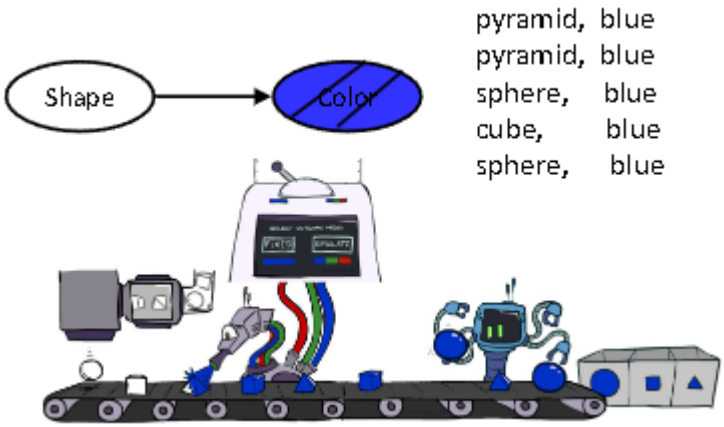
Likelihood Weighting

```
![[Pasted image 20240122193211.png]]
```

Again, we know the query ahead of time, and see if we can further improve the procedure beyond what we did for a rejection sampling.

- Problem with rejection sampling:
    - If evidence is unlikely, rejects lots of samples
    - Evidence not exploited as you sample
        - if the evidence variable are very deep in your BNs, you might have done all that work
        - reach all the way to the bottom of you BNs, you sample your evidence variable , you sample it the wrong way, now you reject -- that sample can not be used.
    - Consider P(Shape | blue)
        - Shape → Colour
            - ~~pyramid, green~~
            - ~~pyramid, red~~
            - sphere , blue
            - ~~cube, red~~
            - ~~sphere, green~~

- Idea: fix evidence variables and sample the rest



  - Now we don't have the problem of throwing out samples, because we make them all blue. That is when we're drawing samples from the network, we don't risk the sample not matching the evidence. We fix it to equal the evidence.
  - Problem: sample distribution not consistent!
  - Solution: weight by probability of evidence given parents
    - You instantiated some shape to be blue, and the probability for blue was 0.2 for that shape
    - you now weight that particular sample by a factor 0.2.
    - **No dropping samples, but adding a weight to each sample**. Instead of being rejected, you get a small weight.

Example

```
![[Pasted image 20240122193556.png]]
```

- Evidence is +s , +w
- We start with Cloudy , we got +c
  - Weight is 1.0
- Next we go to Sprinkler, it is instantiated to be +s, we have weight P(+s|+c) = 0.1
  - Now weight is 1.0 * 0.1
  - *Here, instead of flipping a coin and risking a 90% chance of getting minus sprinkler, I'm going to take the universes where I actually get +s and I pretend I'm going into those universes, which means this sample only really represents 10% of the sample that make it to this point.*
- Next we look at rain, we sample, we got +r
  - Weight is still 1.0 * 0.1
- Last is WetGrass , it is instantiated to be +w , we weight it P(+w|+s,+r) = 0.99
  - now the weight is 1.0 *0.1* 0.99 = 0.099
- So this sample +c,+s,+r,+w has weight 0.099

Pseudocode

```
// single sample
IN: evidence instantiation
w = 1.0
for i=1, 2, …, n
    if Xᵢ is an evidence variable
        Xᵢ = observation xᵢ for X
        Set w = w * P(xᵢ | Parents(Xᵢ))
    else
```

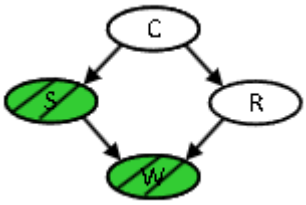
```
        Sample xi from P(Xi | Parents(Xi))
Return (x1, x2, …, xn) , w
```

- Sampling distribution if z sampled and e fixed evidence

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | \text{Parents}(Z_i))$$

- Now, samples have weights

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | \text{Parents}(E_i))$$



- Together, weighted sampling distribution is consistent

$$S_{\text{WS}}(z, e) \cdot w(z, e) = \prod_{i=1}^{l} P(z_i | \text{Parents}(z_i)) \prod_{i=1}^{m} P(e_i | \text{Parents}(e_i))$$
$$= P(\mathbf{z}, \mathbf{e})$$

- Likelihood weighting is good
  - We have taken evidence into account as we generate the sample
  - E.g. here, W's value will get picked based on the evidence values of S, R
  - More of our samples will reflect the state of the world suggested by the evidence
- *BUT*, Likelihood weighting doesn't solve all our problems
  - Evidence influences the choice of downstream variables, but not upstream ones (C isn't more likely to get a value matching the evidence)
    - *Likelihood works really well when your evidence is at the top because then everything else that falls out of the network after that, conditions on that evidence naturally.*
    - but things that came before an evidence variable are not influenced by the evidence variable.
    - so things have to come before the evidence variable are still samples from a prior that doesn't involve the evidence
    - **so it's possible that you are sampling things at the top of your BNs that end up being very inconsistent with what happens at the bottom in terms of evidence.**
    - so if you always encounter this evidence variable at the very end and it's very unlikely, then you will get samples that are not informative yet. e.g. no burglary no earthquake no alarm, but mary called, john called, this sample comes with a very low weight.
    - You'll have to wait a really long time till you finally sample this really unlikely cause of this really unlikely observation. When you finally sample that really unlikely cause, finally we'll have a sample with a high weight and that'll give you a good estimate of the distribution.
- We would like to consider evidence when we sample every variable
  - leads to Gibbs sampling, which has its own issues, but fixes this problem of wanting your samples to take into account the evidence, regardless of where the evidence is placed in the network.

Gibbs Sampling

```
![[Pasted image 20240122194406.png]]
```

In Gibbs Sampling, we don't walk through the network from top to bottom, get a sample, and reset. Instead, we're going to do sort of like iterative improvement for CSP. We're going to start with a complete assignment, and we're

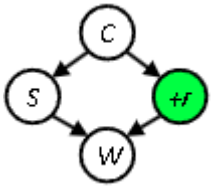
going to tweak it a little bit. The end result is going to be what we take into account evidence upstream and downstream, but there's going to be a price.

- *Procedure*:
  - keep track of a full instantiation $x_1, x_2, \cdots, x_n$.
  - Start with an arbitrary instantiation consistent with the evidence.
    - *we already account for the evidence, the other variables are arbitrarily instantiated.*
  - Sample one variable at a time, conditioned on all the rest, but keep evidence fixed.
    - *You're then going to walk to the variables one at a time, round robin, and leave the evidence fixed. But for every non-evidence variable, you will resample just that variable, conditioned on all the reset staying fixed.*
    - *we have a distribution, a conditional distribution , compute that distribution and then sample from that conditional distribution.*
    - *we haven't yet looked at what it takes to compute that conditional distribution but in principle you could compute a conditional distribution for one variable given all other variables being instantiated , and then sample from that.*
  - Keep repeating this for a long time.
- *Property*: in the limit of repeating this infinitely many times the resulting sample is coming from the correct distribution
  - P(unobserved variables | evidence variables)
  - so we're not going to have to weight samples anymore. We're going to directly get samples from the conditional distribution.
- *Rationale*: both upstream and downstream variables condition on evidence.
- In contrast: likelihood weighting only conditions on upstream evidence, and hence weights obtained in likelihood weighting can sometimes be very small. Sum of weights over all samples is indicative of how many “effective” samples were obtained, so want high weight.

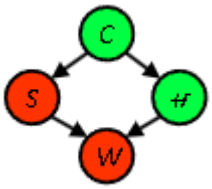
Example

We're going to generate one sample from our BNs, Cloudy, Sprinkler, Rain, WetGrass. Our evidence is +r.

- Step 1: Fix evidence
  - R = +r

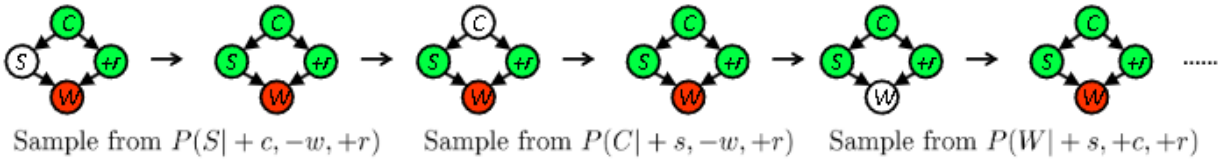


- Step 2: Initialize other variables
  - Randomly
    - say we pick +c, -s, -w



- Steps 3: Repeat
  - Choose a non-evidence variable X
    - this choice here is also supposed to be random
    - so you randomly pick one of your non-evidence variables as your current variable, that you're going to update

- Resample X from P(X | all other variables)
  - Note: **It is not your Bayes Net**. Your Bayes Net has P(X | it's parents)



  - we picked , we uninstantiated it
    - so we compute the conditional distribution for S : P(S|+c,+r,-w)
      - this distribution can answer some query very efficiently
    - sample from that distribution, and we happen to sample +s
  - again, we randomly pick C
    - we compute the conditional distribution for C : P(C|+s,+r,-w)
    - sample from that distribution, this case we got +c
  - now we uninstantiate W
    - we compute the conditional distribution for W : P(W|+s,+c,+r)
    - sample from that distribution, this case we got -w
  - keep going
    - we can look at properties of your BNs, you can infer how long -- how many steps -- you might need to sample
    - Unlike Prior Sampling, here every sample you get looks exactly the same as the previous one. The samples are now highly correlated.
    - If we grab a sample, and we do this thing for a long time, and we grab another sample, and then we rotate around robin for a while, and grap another sample ... If we wait long enough between samples, then the correlations reduce, and we're grabbing samples from the joint distribution over all non-evidence variables conditioned on the evidence.

This is just giving you the very basic idea of how Gibbs Sampling works and you can make it work this way, but if you used it in practice, you'd want to use a lot of methods to make it more efficient.

Efficient Resampling of One Variable

```
![[Pasted image 20240122195304.png]]
```

- Sample from P(S | +c, +r, -w)

$$
\begin{aligned}
P(S|+c,+r,-w) &= \frac{P(S,+c,+r,-w)}{P(+c,+r,-w)} \\
&= \frac{P(S,+c,+r,-w)}{\sum_s P(s,+c,+r,-w)} \\
&= \frac{P(+c)P(S|+c)P(+r|+c)P(-w|S,+r)}{\sum_s P(+c)P(s|+c)P(+r|+c)P(-w|s,+r)} \\
&= \frac{P(+c)P(S|+c)P(+r|+c)P(-w|S,+r)}{P(+c)P(+r|+c)\sum_s P(s|+c)P(-w|s,+r)} \\
&= \frac{P(S|+c)P(-w|S,+r)}{\sum_s P(s|+c)P(-w|s,+r)}
\end{aligned}
$$

  - a) The numerator came from the Bayes net formula. And the denominator is the exact same thing, but summed over S.
- Many things cancel out – only CPTs with S remain!
- More generally: only CPTs that have resampled variable need to be considered, and joined together

    - b) You take all the terms that mention S, you assign them, and then you normalize over all the different values of S. The whole reset of the network doesn't matter.
- It's not so bad.

Further Reading on Gibbs Sampling*

- Gibbs sampling produces sample from the query distribution P(Q | e) in limit of re-sampling infinitely often
- Gibbs sampling is a special case of more general methods called Markov chain Monte Carlo (MCMC) methods
    - Metropolis-Hastings is one of the more famous MCMC methods (in fact, Gibbs sampling is a special case of Metropolis-Hastings)
- You may read about Monte Carlo methods – they're just sampling

---

Lecture 6 Decision Networks and Value of Information

```
![[Pasted image 20240122195523.png]]
```

Decision Networks

```
![[Pasted image 20240122205131.png]]
```

- New node types:
    -  Chance nodes (just like BNs)
    -  Actions (rectangles, cannot have parents, act as observed evidence)
        - You have a choice here
        - You can either take your umbrella with you, or you can leave it at home
        - So this is something you get to set
    -  Utility node (diamond, depends on **action** and **chance** nodes)
        - Unlike the utility we met before, this node is not a number, but a function, a table
        - It tells you for every possible combination of its parent values, what is the utility for experiencing that combination of parent values.
        - Over there, there are 2 parents: umbrella and weather. It could be that
            - It's sunny, you left your umbrella at home, now you get to play with the beach ball, probably have high utility
            - It's sunny, but you brought your umbrella, and now you don't get to play the beach ball, we get to carry umbrella around, you are not so happy
            - It's rainy, you didn't bring an umbrella, that's the worst case
            - It's rainy, but you brought your umbrella, at least you have a way to protect yourself from the rain.
        - All 4 of these will have a number associated with them, the utility for that particular outcome.
        - **If there is only 1 agent, where will only be one utility node, if there's more than 1 agent there could be more than 1 utility node.**

What are we going to be doing? We are still going to be maximizing expected utility.

- **MEU: choose the action which maximizes the expected utility given the evidence**
- Can directly operationalize this with decision networks

    - Bayes nets with nodes for utility and actions
    - Lets us calculate the expected utility for each action
- What will we do with a network like above ?
    - We'll look at every possible action we might take, compute the expected utility if we were to take that action and then pick the action that maximizes the expected utility.

Selection of Action

- Action selection
    - Instantiate all evidence
    - Set action node(s) each possible way
        - loop over all possible choices for the actions
    - Calculate posterior for all **parents of utility node**, given the evidence
        - this case you would compute the conditional distribution of Weather, given the evidence.
        - if weather itself is evidence it's very easy, if there's no evidence then it's just a prior for Weather
        - if there is some forecast you will essentially apply Bayes rule to find out P(weather | forecast)
    - Calculate expected utility for each action
    - Choose maximizing action

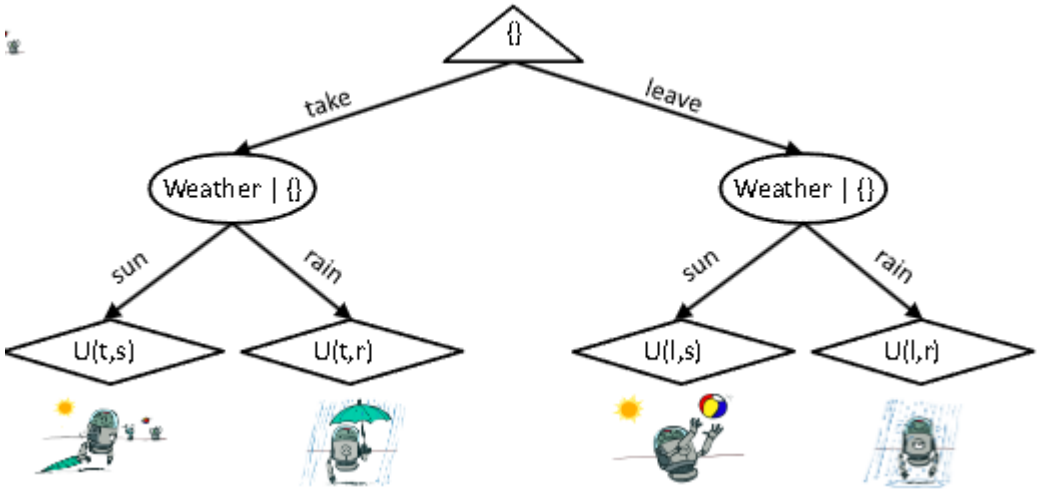
Example

```
![[Pasted image 20240122210452.png]]
```

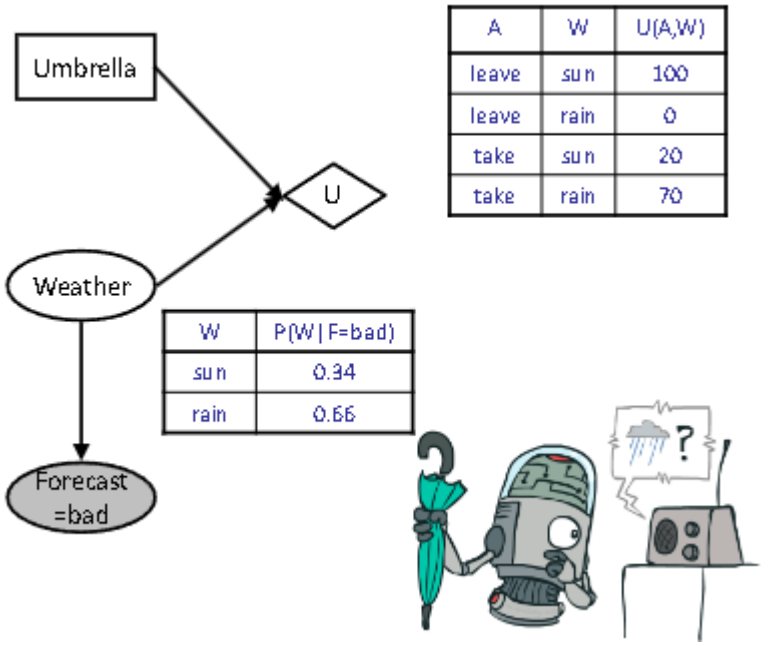
There are part of the problem specification of course. If you are designing a robot to be deployed somewhere, you would decide for that robot what the utilities are such that when the robot maximizes expected utility it does what you want it to try to achieve.

- We need to loop over all possible actions, the Umbrella
    - Umbrella = leave
        - what is the expected utility? Sum over all possible outcomes for Weather
        - EU(leave) = $\sum_w$ P(w)U(leave, w) = 0.7 *100* + *0.3* 0 = 70
    - Umbrella = take
        - EU(take) =$\sum_w$ P(w)U(take, w) = 0.7 *20* + *0.3* 70 = 35
- Optimal decision = leave

    - MEU(∅) = max_a EU(a) = 70
        - ∅ means no evidence.

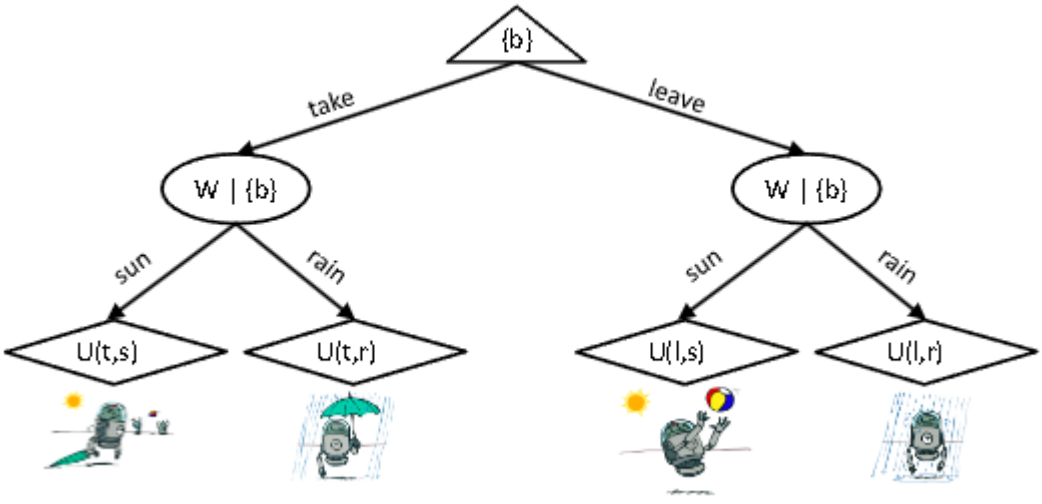


- Almost exactly like expectimax / MDPs
- What's changed?
  - when we did expectimax, we have the probability
  - when in DNs, here, when we say, what's the probability of weather given the forecast, we actually have to do computation to figure out the probabilities from that expectation node. We shall do that by running Bayes Net inference.



We listened to the forecast and the forecast is bad.

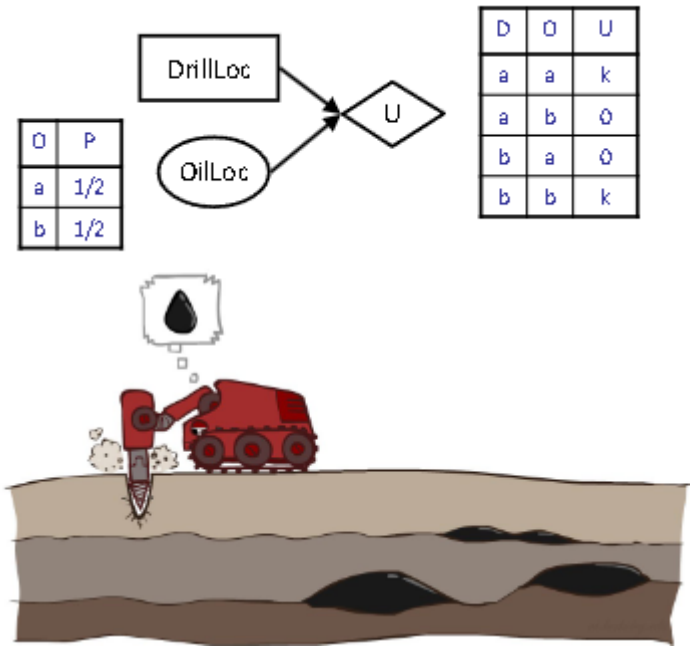
- So in this kind of computations we compute the conditional distribution of the parents given evidence.
  - P(W|F=bad)
  - Bayes net provides the CPT P(F|W), we need to compute P(W|F=bad).
- loop
  - Umbrella = leave
    - EU(leave|bad) = $\sum_w$ P(w|bad)U(leave, w) = 0.34 *100* + *0.66* 0 = 34
  - Umbrella = take
    - EU(take|bad) = $\sum_w$ P(w|bad)U(take, w) = 0.34 *20* + *0.66* 70 = 53
- Optimal decision = take
  - MEU(F=bad) = $\max_a$ EU(a|bad) = 53



Value of Information

```
![[Pasted image 20240122212518.png]]
```

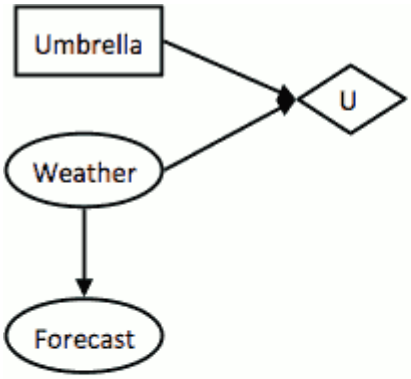
- Idea: compute value of acquiring evidence
  - Can be done directly from decision network
  - What does that mean?
    - *Well remember, if I put DNs in front of you and say, all right, decision time. Are you taking the umbrella or not? You can say, ok, hold on a second, calculate, calculate, calculate. I'm going to take the umbrella, and my MEU is going to be 53. We can then talk hypothetically about what would happen if I showed you a variable. You'd make better decisions, and you get higher utilities.*
- Example: buying oil drilling rights
  - Two blocks A and B, exactly one has oil, worth k
  - You can drill in one location
  - Prior probabilities 0.5 each, & mutually exclusive
  - Drilling in either A or B has EU = k/2, MEU = k/2



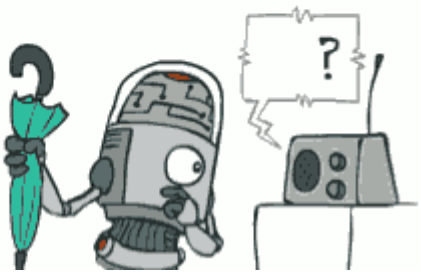
- Question: what's the **value of information** of O?
  - Value of knowing which of A or B has oil.
    - if you know the oil location (somebody tells you), then your MEU is k.
    - before, your MEU is k/2
    - the difference is k/2, so rationally you're willing to pay k/2 to get to know where the oil is.
  - Value is expected gain in MEU from new info
  - Survey may say "oil in a" or "oil in b," prob 0.5 each
  - If we know OilLoc, MEU is k (either way)
  - Gain in MEU from knowing OilLoc?
  - VPI(OilLoc) = k/2
    - value of **perfect** information
    - That is the difference between the MEU whether or not taking the action with the variable.
  - Fair price of information: k/2

VPI Example: Weather

We can observe only Forecast. Question is how valuable is it to observe the forecast.



A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

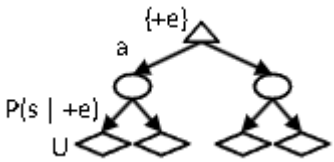


- MEU with no evidence
  - MEU(∅) = 70
- MEU if forecast is bad
  - MEU(F=bad) = 53
- MEU if forecast is good
  - MEU(F=good) = 95
- Forecast distribution
  - P(F=good) = 0.59, P(F=bad) = 0.41
    - P(F) is not in BNs. We can compute by running inference in BNs
  - 0.59 · 95 + 0.41 · 53 - 70 = 7.8
  - In this case 7.8 means that you would be willing to pay 7.8 to get to listen to the forecast.

$$\text{VPI}(E'|e) = \left(\sum_{e'} P(e'|e)\text{MEU}(e,e')\right) - \text{MEU}(e)$$

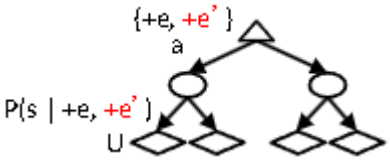
  - We have the VPI of a particular variable or set of variables E' given that you already observed another set of variables *e* which could be the empty set.
  - In our example *e* was the empty set , E' was equal to forecast.
- Assume we have evidence E=e. Value if we act now:

$$\text{MEU}(e) = \max_a \sum_s P(s|e)\, U(s,a)$$



  - You have initial evidence +e, so choose an action, that point the chance node kick in, which will instantiate the parent variables of the utility node and then you have your utility nodes.
- Assume we see that E' = e'. Value if we act then:

$$\text{MEU}(e,e') = \max_a \sum_s P(s|e,e')\, U(s,a)$$

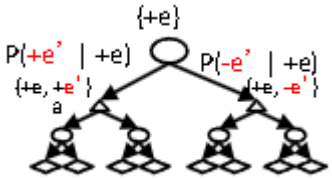


- After you observe both {+e, +e'}, you take an action, and then the chance nodes kick in and then the utility nodes.

- BUT **E' is a random variable whose value is unknown**, so we don't know what e' will be.
  - So we need to have to prediction about what e' will be in order to compute how valuable that information is to us.
- Expected value if E' is revealed and then we act:

$$\text{MEU}(e, E') = \sum_{e'} P(e'|e)\text{MEU}(e, e')$$

  - *the missing image part is* e'



  - now we have an extra chance node
  - you were to get to observe evidence but you don't know yet what the evidence is going to be. You start with chance node.
  - first thing that happens is the evidence e' will be observed. you don't know yet what it's going to be, could be +e' or -e'.
  - after that get instantiated you get to choose your action, after that more chance nodes will kick in for the parent variables of the utility node after which utility nodes kick in.
- Value of information: how much MEU goes up by revealing E' first then acting, over acting now:

$$\text{VPI}(E'|e) = \text{MEU}(e, E') - \text{MEU}(e)$$

Properties

- Nonnegative
  - ∀E',e : VPI(E'|e) ≥ 0
  - More information will always help you. **on the average**, when you don't know yet what the information is going to be, it is good to get information.
- Nonadditive (think of observing E_j twice)
  - VPI(E_j,E_k |e) ≠ $VPI(E_j|e)$ + VPI(E_k |e)
  - that VPI(E_j,E_k |e) is not the same as the sum of the individual VPI.
- Order-independent
  - VPI(E_j,E_k |e) = $VPI(E_j|e)$ + VPI(E_k |e , E_j)
  - = VPI(E_k |e) + VPI(E_j |e , E_k)
  - a lot like the chain rule
- The following statements are true:
  - VPI is guaranteed to be nonnegative (≥0).
  - The MEU after observing a node could potentially be less than the MEU before observing that node.
  - VPI is guaranteed to be exactly zero for any node that is conditionally independent (given the evidence so far) of all parents of the utility node.

Quick VPI Questions

- The soup of the day is either clam chowder or split pea, but you wouldn't order either one. What's the value of knowing which it is?
  - 0
- There are two kinds of plastic forks at a picnic. One kind is slightly sturdier. What's the value of knowing which?
  - somewhat positive. can not really put a number on it this.
- You're playing the lottery. The prize will be $0 or $100. You can play any number between 1 and 100 (chance of winning is 1%). What is the value of knowing the winning number?
  - 99$

Value of Imperfect Information?

```
![[Pasted image 20240122214440.png]]
```

- No such thing (as we formulate it)
  - *in our formulation, there is no imperfect information.*
- Information corresponds to the observation of a node in the decision network
  - Information is revealing the value of a random variable in your network.
  - observing means you know what the value is
- If data is "noisy" that just means we don't observe the original variable, but another variable which is a noisy version of the original one
  - e.g. we observe forecast, not weather.

VPI Question

```
![[Pasted image 20240122214605.png]]
```

- VPI(OilLoc)?
  - saw that before, that is still k/2
- VPI(ScoutingReport)?
  - [0,k/2]
  - can not put the number on this.
- VPI(Scout) ?
  - for knowing what scout is doing the scouting report
  - = 0
  - if we observe evidence that does not change the distribution for the parent variables , it will not change our decision, it will not change our expected utility.
  - Scout is independent of OilLoc with no other evidence. So knowing the Scout is not affecting the distribution of parent variables , hence not affecting our MEU.
- VPI(Scout | ScoutingReport)?
  - > 0
- Generally:
  - If Parents(U) $\perp$ Z | CurrentEvidence
    - Then VPI(Z | CurrentEvidence) = 0
  - that is, VPI is guaranteed to be exactly zero for any node that is conditionally independent (given the evidence so far) of all parents of the utility node.

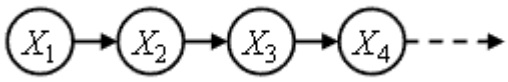
Lecture 7 Hidden Markov Models

Reasoning over Time or Space

- Often, we want to reason about a sequence of observations
  - Speech recognition
  - Robot localization
  - User attention
  - Medical monitoring
- Need to introduce time (or space) into our models

Markov Models

- A **Markov Model** is a chain-structured Bayes' net (BN)
  - Each node is identically distributed (stationarity)
  - Value of X at a given time is called the **state**



    - **So you have some variable X that replicates at every time step**
  - As a Bayes' net
    - $P(X_1)$
    - P(X_t | X_{t-1})
  - Parameters: called **transition probability** or dynamics, specify how the state evolves over time (also, initial state probabilities)
    - This is the model how the world changes
  - Stationarity assumption: transition probabilities the same at all times
    - This means transition probabilities P(X_t | X_{t-1}) don't depend on time, they are always the same.
  - Same as MDP transition model, but no choice of action
    - Here is no action, you are just interested in watching how its value evolves probabilistically over time, for example, what's the probability of rain 5 days after it was sunny?

Conditional Independence

- Basic conditional independence:
  - Past and future independent given the present
    - Bayes Net D-separation
  - Each time step only depends on the previous
    - To predict what happens at the next time, just knowing the current time is the best thing. Knowing more things about the past is not going to help you.
  - This is called the (first order) Markov property
    - You might say, well, what if it doesn't apply in my situation? What if my situation, the state of the next time, depends on the state of the current time and the state of the previous time?
    - Well, to still be able to fit in this format and to really fit the notion of state, you should then combine the state of the current time and the state of the previous time in one bigger state variable that you now call your state.
- Note that the chain is just a (growable) BN
  - We can always use generic BN reasoning on it if we truncate the chain at a fixed length.

Example

```
![[Pasted image 20240122224843.png]]
```

- States: X = {rain, sun}
- Initial distribution: 1.0 sun
- CPT $P(X_t|X_{t-1})$:

X_{t-1}	X_t	$P(X_t \mid X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

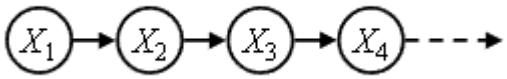
- What is the probability distribution after one step?

$$P(X_2 = \text{sun}) = P(X_2 = \text{sun}|X_1 = \text{sun})P(X_1 = \text{sun}) + P(X_2 = \text{sun}|X_1 = \text{rain})P(X_1 = \text{rain})$$

$$0.9 \cdot 1.0 + 0.3 \cdot 0.0 = 0.9$$

Mini-Forward Algorithm

- Question: What's P(X) on some day t?



$$P(x_1) = \text{known}$$

$$P(x_t) = \sum_{x_{t-1}} P(x_{t-1}, x_t) = \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1})$$

Forward simulation

- What is actually going on here, think about the Markov model.
  - You have X_1, X_2. It's a Bayes net with two variables. You have all the distributions, all the tables. You have instantiated a distribution for the initial time X_1 and now you're running variable elimination to get the distribution for X_2. So you're trying to compute $P(X_2)$ in this Bayes net.

Example

```
![[Pasted image 20240122225402.png]]
Running it long enough will make it converge to some value, this is what we call the stationary distribution.
```

Stationary Distributions

- For most chains:
  - Influence of the initial distribution gets less and less over time.
  - The distribution we end up in is independent of the initial distribution
- Stationary distribution:

- The distribution we end up with is called the **stationary distribution P∞** of the chain

$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

- It satisfies

- As the property of Markov matrix , it will converge to 0.94868/0.31623 = 3:1, that means:
  - P∞(sun) = 3/4
  - P∞(rain) = 1/4

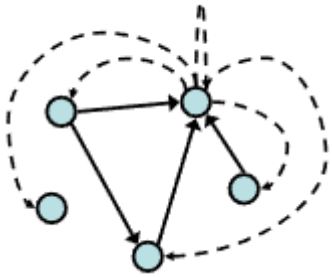
Example

```
![[Pasted image 20240122225906.png]]
```

Applications of Stationary Distributions

Web Link Analysis

- PageRank over a web graph
  - Each web page is a state
  - Initial distribution: uniform over pages
  - Transitions:
    - With prob. c, uniform jump to a random page (dotted lines, not all shown)
    - With prob. 1-c, follow a random outlink (solid lines)



- Stationary distribution
  - Will spend more time on highly reachable pages
  - E.g. many ways to get to the Acrobat Reader download page
  - Somewhat robust to link spam
  - Google 1.0 returned the set of pages containing all your keywords in decreasing rank, now all search engines use link analysis along with many other factors (rank actually getting less important over time

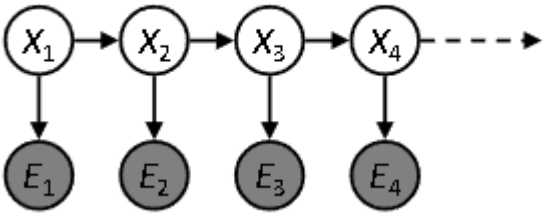
Gibbs Sampling*

- Each joint instantiation over all hidden and query variables is a state: $X_1, \cdots, X_n$ = H U Q
- Transitions:
  - With probability 1/n resample variable X_j according to $P(X_j|x_1, x_2, \cdots, x_{j-1}, x_{j+1}, \cdots, x_n, e_1, \cdots, e_m)$
- Stationary distribution:
  - Conditional distribution $P(X_1, X_2, \cdots, X_n|e_1, \cdots, e_m)$
  - Means that when running Gibbs sampling long enough we get a sample from the desired distribution
  - Requires some proof to show this is true!

Hidden Markov Models

- Markov chains not so useful for most agents
  - Need observations to update your beliefs

- Hidden Markov models (HMMs)
  - Underlying Markov chain over states X
  - You observe outputs (effects) at each time step



  - In HMM, we don't get to observe the hidden state -- X, but the hope is that by observing the evidence variables, we can somehow infer a posterior distribution over the hidden states that allows us to do something interesting.
  - What we're going to have is that every time there's going to be a hidden variable. That structure like a Markov chain and each time the evidence depends only on the state unobserved but only on the state at that time (E | X) (Bayes net D-separation)

Example

Weather

```
![[Pasted image 20240122234130.png]]
```

The sad grad student HMM.

There's a grad student. He is just in the basement at all times. They don't come out of that basement. But luck has it, every now and then Professor stops by and says hi to them and sometimes, professor has an umbrella, sometimes not. And that, for the grad student, is a way to extract information about whether today might be a sunny day or a rainy day.

- Hidden variable: where or not it's raining -- True or False
- Observed variable: an umbrella
- Now, there's multiple distributions involved.
  - the initial state distribution
    - initially sunny or rainy
  - a distribution of for next day
    - like we saw in the Markov Model, the transition model.
  - a distribution for evidence given current state
    - the probability of various evidence values given the underlying stat , which we then used to predict the opposite -- something about X.
- So an HMM is defined by:
  - **Initial distribution: $P(X_1)$**
  - **Transitions: $P(X_t|X_{t-1})$**
  - **Emissions: $P(E_t|X_t)$**

So what do we need to define the HMM ?

- we need 1 function which says how rain on one day depends on the previous day.

R_{t-1}	$P(R_t)$
t	0.7
f	0.3

  - This is the rain to sun transition probability
- we also need a function says given rain and separately given sun, what's probability of seeing an umbrella

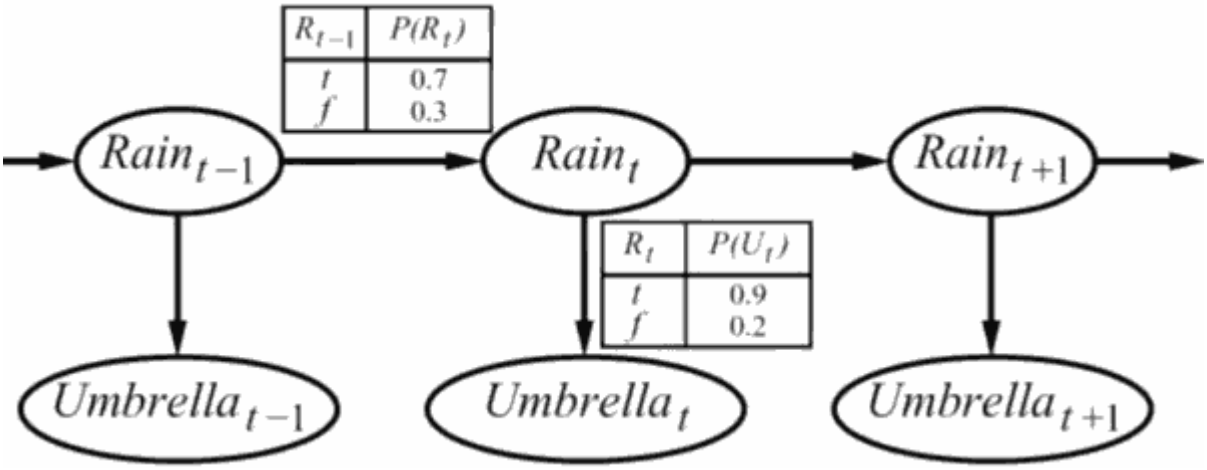
R_t	$P(U_t)$
t	0.9
f	0.2

  - That is called emission model. This tells you what probability of seeing various evidences values is for each underlying state.
  - In this case it says that when it's raining you see the umbrella 90% of time, but when it's not raining you still see it 20% of time.
- equivalent to

R_{t-1}	R_t	$P(R_t \mid R_{t-1})$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

R_t	U_t	$P(U_t \mid R_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

- We have HMM:



So from a single observation of an umbrella you don't know very much, but if day after day you're seeing the umbrella you start to kind of gain some confidence.

Ghostbusters

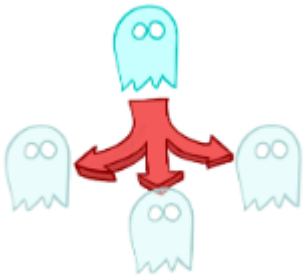
- $P(X_1)$ = uniform

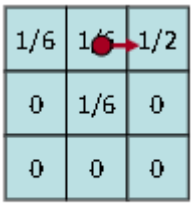
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$P(X_1)$

  - in the course demo, it is 0.02 that you see everywhere on map.

- P(X|X') = usually move clockwise, but sometimes move in a random direction or stay in place

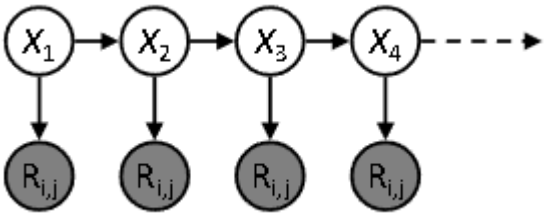




P(X|X' =<1,2>)

  - For that red square, it's 50% precent probability of moving right , 1/6 probability that you'll stand, 1/6 probability to go in other direction.
    - Where do these conditional probabilities come from?
      - This is your assumptions about the world, you might learn them from data, for now that's just an input.
    - That's what happens from that one state. **But you generally don't know what state you're in**, and you need to sum over all the options, that's the forward algorithm was about.
- $P(R_{ij}|X)$ = same sensor model as before: red means close, green means far away.
  - somewhere there has to be specified precisely the probability of reading at a certain position given the underlying state.

  - so it might say if you read at (3,3) and the ghost is there , your probability of getting red is 0.9. Those facts live in the emission model, they say how the evidence directly relates to the state at that time.



- The dynamics of HMM: alternation between *time passing* and *measurement*, *time passing* and *measurement*, ...
  - Time passes
    - Which tends to diffuse where the ghost probability mass is
  - Measurement
    - Which tends to help us concentrate where the ghosts might be.
  - This is the HMM process in action.

Conditional Independence

Let's think about the independence assumptions we make in this model.

- HMMs have 2 import independence properties

    - Markov hidden process: future depends on past via the present
        - That is, knowing a state at a given time separates past from future.
        - Same as MM.
    - Current observation independent of all else given current state
        - Given X_3 , E_3 is independent of all anything else we could find out once we know X_3. It's just X_3 directly influences our measurement, and nothing else has any influence anymore.
- Quiz: does this mean that evidence variables are guaranteed to be independent ?
    - If I don't observe anything I could say: Is the evidence I see at $time_1$ independent of the evidence I see at $time_2$?
    - It is like the umbrella on thuesday is independent of the umbrella on Wednesday.
    - So It seems like it shouldn't be. The evidence variables are absolutely not independent. They're only conditionally independent.
    - for example, from E_1 to E_4, there is an active path, it consist of a common cause, and a causal chains. And nothing is observed in-between

Real HMM Examples

- Speech recognition HMMs:
    - Observations are acoustic signals (continuous valued)
    - States are specific positions in specific words (so, tens of thousands)
- Machine translation HMMs:
    - Observations are words (tens of thousands)
    - States are translation options
- Robot tracking:
    - Observations are range readings (continuous)
    - States are positions on a map (continuous)

Filtering / Monitoring

Now we are going to talk about how to keep track of what you believe about a variable X -- the state variable -- as evidence comes it and time passes, and from this we'll build up the full-forward algorithm.

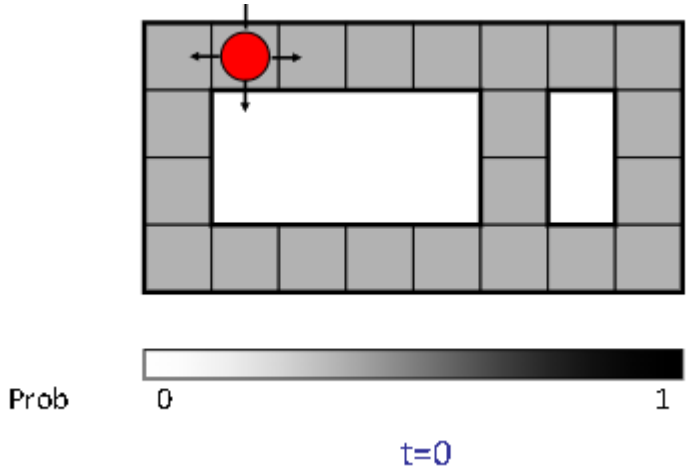
- Filtering, or monitoring, is the task of tracking the distribution B_t(X) = P_t(X_t | e_1, \cdots, e_t) (the belief state) over time
    - $B_t(X)$: belief state over state, which is a conditional of X_t given all evidence up to time t.
- We start with $B_1(X)$ in an initial setting, usually uniform
- As **time passes**, or we **get observations**, we update B(X)
- The Kalman filter was invented in the 60's and first implemented as a method of trajectory estimation for the Apollo program

Example Robot Localization

The robot doesn't know where it is . It knows the map -- somebody gave blueprints .

It is going down the corridor and all do is shoot out lasers in each direction and see how close they bounced off a

wall.



- Sensor model: can read in which directions there is a wall or not, it makes at most 1 error.
- Motion model: may not execute action with small prob.

So it knows there's a wall right here, but no wall in front of me. And if it gets a reading that says there's wall on my left and right but not in front or behind, then suddenly it shouldn't think it's anywhere in this building. Where should I think it is? It is kind of thinks it's in the corridors, corridors look like that. It's the sensor model, from is conditioned on my current position, I need to say a distribution over readings. Let's imagine that instead of the continuous readings, the readings are wall or not in each direction.

We first do a sensor reading.

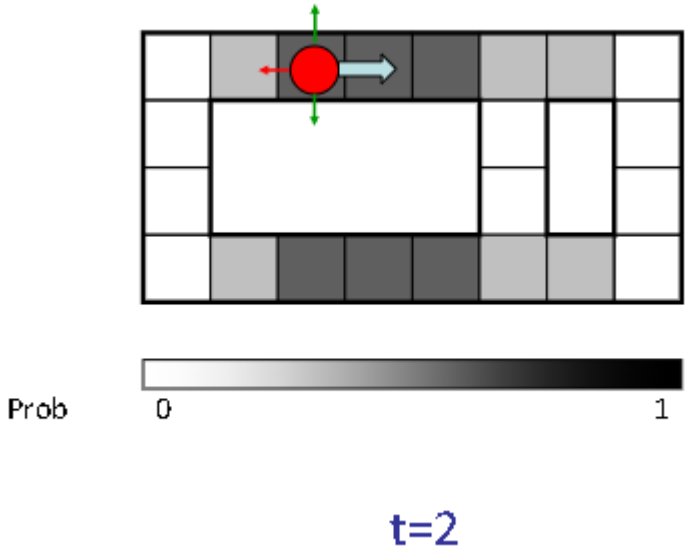
So if I sense that is a wall above and below I should have pretty high probability of my belief distribution of being in the dark gray squares. I should have some smaller probability of being in the lighter grey square.

```
![[Pasted image 20240123000142.png]]
```

- Lighter grey: was possible to get the reading, but less likely b/c required 1 mistake

Then the robot will move to the right. We'll have a transition model for that. It'll not guarantee to move one to the right, but with high probability, it will.

Then we get another sensor reading.



Then moves again, also does a sensory update,

Time passes.

As I continue reading north and south walls , what will happen is there will be fewer and fewer places which are consistent with my history of readings.

```
![[Pasted image 20240123000315.png]]

![[Pasted image 20240123000327.png]]

![[Pasted image 20240123000336.png]]
```

Inference:

Base cases

So, what are the base cases?
Inference in Markov model is actually the approximate inference.

Let's do the base cases. There's really 2 things that happen in HMMs.

1. One is time passes.
    - You go from X_t to X_{t+1} to.
2. The other thing happens you see evidence.

But if all you had was a single time slice, you had X_1 and you saw evidence at time$_1$ E_1, and you want to compute what's probability over my hidden state X_1 given my evidence E_1. So you just want to compute this conditional probability : $P(X_1|e_1)$.

- I know $P(X_1)$, this is what before I see the evidence.
- I also know a distribution that tells me how the evidence relates to X_1 : $P(E_1|X_1)$.

That's not quite what I want. What I want is $P(X_1|e_1)$.

$$
\begin{aligned}
P(X_1|e_1) &= P(X_1,e_1)/P(e_1) \\
&= P(e_1|X_1)\cdot P(X_1) \ /P(e_1)
\end{aligned}
$$

We're interested over X_1. e_1 is not a variable. Anything that does not involve X_1 we can just remove if it's just multiplied in, because this constant e_7 is present for each value of x_1. It say it's proportional to this.

And I can do the computation being off by that constant and then renormalize at the end.

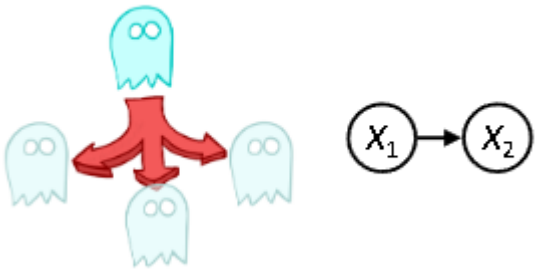
$$
\begin{aligned}
P(x_1|e_1) &= P(x_1,e_1)/P(e_1) \\
&\propto P(x_1,e_1) \\
&= P(x_1)\cdot P(e_1|x_1)
\end{aligned}
$$

That says for each value X, they get weighted by the probability of the evidence given that underlying state.

What would be the result of this calculation ?

x_1	$\propto P(x_1,e_1)$	$P(x_1)$
0	0.2	0.4
1	0.3	0.6

I'm instead going to compute $P(X_1,e_1)$ and normalize in the end. I know how to compute it -- $P(X_1,e_1) = P(e_1|X_1)\cdot P(X_1)$. So I compute all of these products: current probability times evidence probability. And then once I have that whole vector of those I renormalize and now I have the conditional distribution of $P(X_1|e_1)$. That is what happens when you incorporate evidence: you take your current vector of probabilities $P(X_1)$, you multiply each one by the appropriate evidence factor and then you **renormalize** it.



The other base case is transition over time. This is the Markov model update.
You have a distribution over X_1 and rather than seeing evidence, time passes by one step. Well in this case I know $P(X_1)$, and I know $P(X_2|X_1)$. But I want is $P(X_2)$.

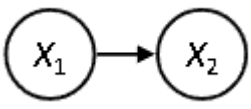
$$P(x_2) = \sum_{x_1} P(x_1, x_2)$$
$$= \sum_{x_1} P(x_1)P(x_2|x_1)$$

That's exactly what we've been doing in the first half of lecture
These things are interleaved.

Base Case 2: Passage of Time

- Assume we have current belief P(X | evidence to date)

$$B(X_t) = P(X_t|e_{1:t})$$



- Then, after one time step passes:

$$P(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}, x_t|e_{1:t})$$
$$= \sum_{x_t} P(X_{t+1}|x_t, e_{1:t})P(x_t|e_{1:t})$$
$$= \sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t})$$

  - Or compactly

$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t)B(x_t)$$

- Basic idea: beliefs get "pushed" through the transitions
  - With the "B" notation, we have to be careful about what time step t the belief is about, and what evidence it includes

Example

```
![[Pasted image 20240123001541.png]]
```

Base Case 1: Observation

```
![[Pasted image 20240123001705.png]]
```

- Assume we have current belief before the evidence P(X | previous evidence):
  - $B'(X_{t+1}) = P(X_{t+1}|e_{1:t})$
  - I have a belief vector that says here's my probability distribution over what's going on at a certain time BEFORE I see my evidence.
- Then, after the evidence tomorrow comes in:

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t})/P(e_{t+1}|e_{1:t})$$
$$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|e_{1:t}, X_{t+1})P(X_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$$

  - step1: conditional probability rule
  - step3: chain rule
  - step4: conditional independent
  - the evidence e_{t+1} does not depend on any past evidence if we know X_{t+1}. That's an assumption.
  - so the blue thing is your probability before you saw your evidence, your current belief
    - the black thing is our measurement model

    - you weighted by the evidence, and then this vector doesn't add to 1 anymore. So you renormalized it now, it adds up to 1 again, now the red thing is including the evidence you just saw.
- Or compactly:

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$

    - if the evidence is very compatible with the next state, then the probability mass will go up for that next state, otherwise it goes down.
    - you take your vector (blue) , you do point product with the evidence vector, and you normalize them. Now you have your new beliefs.
- Basic idea: beliefs “reweighted” by likelihood of evidence
- **Unlike passage of time, we have to renormalize**

Example

```
![[Pasted image 20240123002152.png]]
```

Example: Weather HMM

```
![[Pasted image 20240123002300.png]]
```

The Forward Algorithm

Rather than doing a time update and doing an evidence update, I'm going to just do one update.

- We are given evidence at each time and want to know
    - B_t (X) = P(X_t|e_{1:t})
- We can derive the following updates:

$$P(x_t|e_{1:t}) \propto_X P(x_t, e_{1:t})$$

We can normalize as we go if we want to have P(x|e) at each time step, or just once at the end...

$$= \sum_{x_{t-1}} P(x_{t-1}, x_t, e_{1:t})$$

$$= \sum_{x_{t-1}} P(x_{t-1}, e_{1:t-1})P(x_t|x_{t-1})P(e_t|x_t)$$

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1})P(x_{t-1}, e_{1:t-1})$$

    - step1: get rid of / *P(*$e{1:t}$)_
    - step2: bring in x_{t-1} and sum out. because we want a recursive update equation as a function of what we had at the previous time.
    - step3: decompose into transition model and measurement model.
        - P(x_{t-1}, x_t, $e_{1:t}$)
        - = P(x_{t-1}, $e_{1:t-1}$)P(x_t, e_t | x_{t-1}, $e_{1:t-1}$) = P(x_{t-1}, $e_{1:t-1}$)P(x_t, e_t | x_{t-1})
        - = P(x_{t-1}, $e_{1:t-1}$)P(e_t | x_{t-1}, x_t)P(x_t | x_{t-1})
        - = P(x_{t-1}, $e_{1:t-1}$)P(e_t | x_t)P(x_t | x_{t-1})
    - step4: reorganize a little bit
- This is exactly variable elimination with order X_1,X_2,...

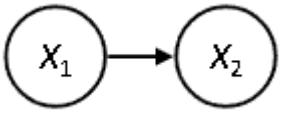
The forward algorithm is a dynamic program for computing at each time slice, the distribution over the state at that time given all the evidence to date.

Online Belief Updates

Online belief update, which is most common, is where you would essentially just do these thing.

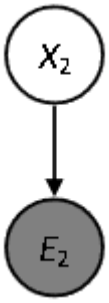
- Every time step, we start with current P(X | evidence)
- We update for time:

$$P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})$$



- We update for evidence:

$$P(x_t|e_{1:t}) \propto_X P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)$$



- The forward algorithm does both at once (and doesn't normalize)

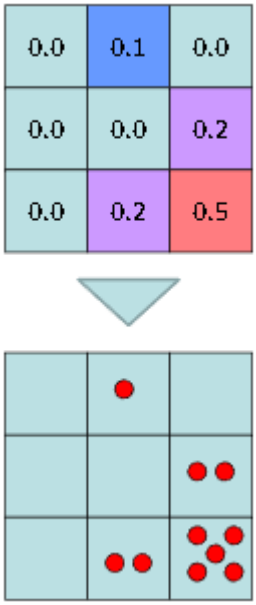
> Start of part 2 HMMs, Particle Filters, and Applications

Particle Filtering

```
![[Pasted image 20240123003401.png]]
```

- Filtering: approximate solution
  - *Remember filtering is one name for the problem where I ask given all of my evidence, what is the current hidden variable X_t? Where is the robot now?*
- Sometimes |X| is too big to use exact inference
  - |X| may be too big to even store B(X)
  - E.g. X is continuous
- Solution: approximate inference
  - Track samples of X, not all values
    - I can't enumerate all the pairs or triples of real numbers. Instead of keeping track of a map from X to real numbers , I'm going to keep track of a list of samples.
    - Here are 10 samples each red dot is a sample
    - And in this case my samples live on some but not all of the location on the grid.
  - Samples are called particles
  - Time per step is linear in the number of samples
    - I'm going to do operations on these samples to give me new samples, and I'm going to do it basically by scanning over the old samples.
  - But: number needed may be large
  - In memory: list of particles, not states

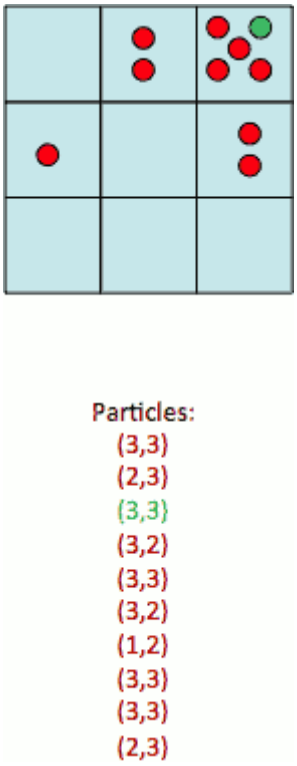
- This is how robot localization works in practice



- Particle is just new name for sample

Representation: Particles

- Our representation of P(X) is now a list of N particles (samples)
  - Generally, `N << |X|`
    - N is a lot smaller than |X|
  - Storing map from X to counts would defeat the point



    - Here instead of writing 9 numbers which is 1 probability for each square I'm gonna have maybe 10 particles.
      - **Each particle has a specific value of X** , eg. green particle (3,3), and it's not the only particle that represents that hypothesis, we actually have 5 completely different particle that all predicting the same state.
      - so what's the probability of (3,3)? 50% ! It's probably wrong but that's what the particles say.
- P(x) approximated by number of particles with value x
  - So, many x may have P(x) = 0!
  - More particles, more accuracy

- For now, all particles have a weight of 1

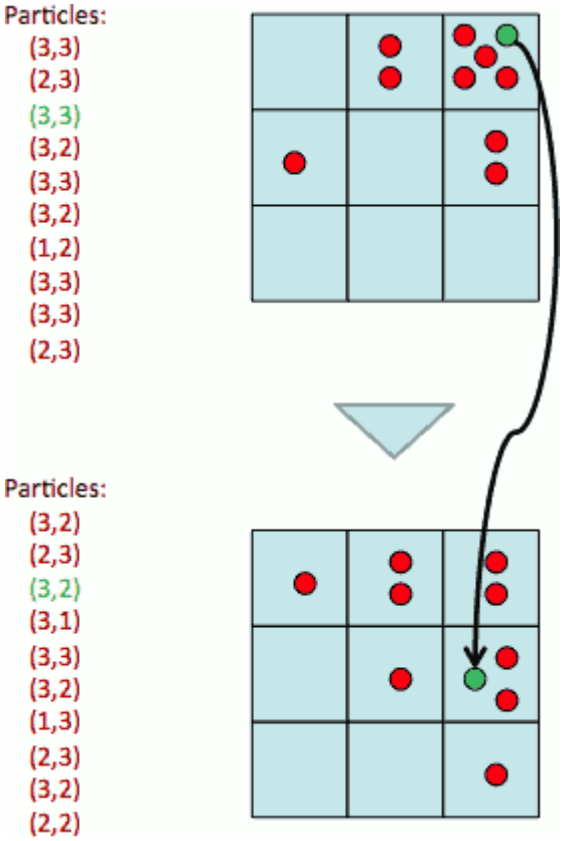
Particle Filtering: Elapse Time

Now what do I do? I might start with my particles uniform or I have some particular belief and when time passes I need to move these particles around to reflect that.

So I pick up each particle -- let's pick up the green one, a hypothesis of (3,3) -- where will it be next time?

Well I grab my transition model -- which might say counter-clockwise motion with high probability -- so I grap this particle and I say you're no longer a distribution, you're a single value of X, you maybe wrong but you're a single value of X and for that particular value of X.

- Each particle is moved by sampling its next position from the transition model
  - `x' = sample( P(X'|x) )`
  - This is like prior sampling -- samples' frequencies reflect the transition probabilities
  - Here, most samples move clockwise, but some move in another direction or stay in place



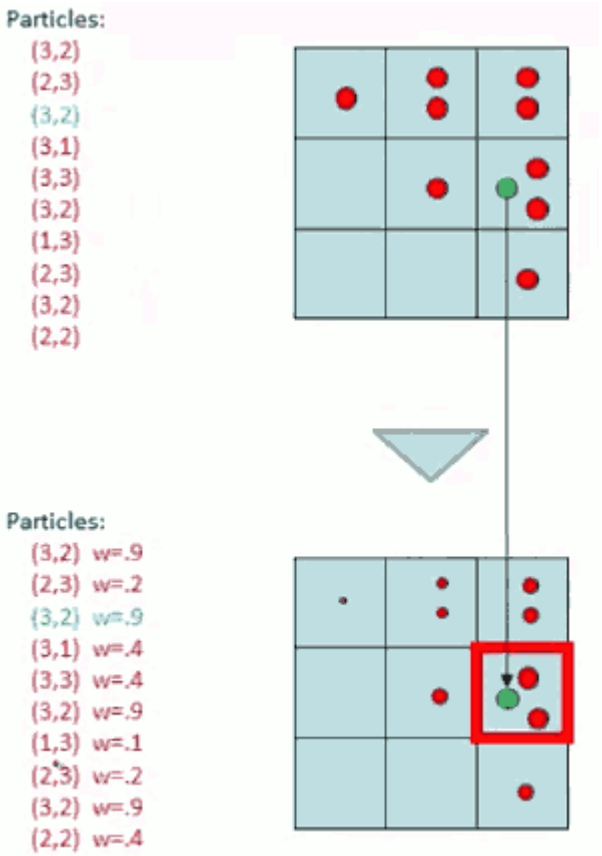
    - *I don't create particles, don't destroy particles, I picked them up 1 by 1 and I simulate what might happen to that particle in the next time step.*
    - *So there might be 5 particles on (3,3) but they might not all get the same future because I flip a coin for each one. They might spread out in this case they do.*
- This captures the passage of time
  - If enough samples, close to exact values before and after (consistent)
  - *So someone gives me a HMM, that means they've given me the transition probabilities. I take my particles and each particle get simulated. That is like letting time pass in my model. That's how in particle filtering time passes.*

Particle Filtering: Observe

What happens when I get evidence? It's a little tricky.

- Slightly trickier:
  - Don't sample observation, fix it

    - Similar to likelihood weighting, down weight samples based on the evidence
        - w(x) = P(e|x)
        - $B(X) \propto P(e|X)B'(x)$
- As before, the probabilities don't sum to one, since all have been downweighted (in fact they now sum to (N times) an approximation of P(e))



Let's say here are my 10 particles. What happens when I get evidence that there's a reading of red meaning the ghost is close right here in this square (3,2). We'll remember how evidence works in that case: I take each of my probabilities and I down weighted by the probability of the evidence. In that analoge of that here is to take each of your particles and give it a weight that reflects how likely the evidence is from that location. So this green particle at (3,2) maybe the probability of seeing red if you actually are at (3,2) is 0.9. So this is a reasonable hypothesis. But this guy at top-left square has a very low probability of seeing this reading.

So we get new samples weighted from the old samples.

Particle Filtering: Resample

- Rather than tracking weighted samples, we resample
    - We're not going to track these weighted samples, they are no good to me, because their weights are starting to shrink. And if I do this for too long, their weights will all go to zero.
- N times, we choose from our weighted sample distribution (i.e. draw with replacement)
    - So what I'll do is to create new particles. The new particles, I sample with replacement from the old weighted samples, are now equally weighted.
    - (3,2) had a pretty high weight. So even though we get rid of all the old particles there going to be a lot of new ones which choose (3,2).
- This is equivalent to renormalizing the distribution
    - procedurally the idea is when you see evidence in your particle filter , you line up your particles , you weight them by the evidence and then you clone new particles through your old particles , and now the weights are all gone.
- Now the update is complete for this time step, continue with the next one

Summary

```
![[Pasted image 20240123012712.png]]
```

Robot Localization

```
![[Pasted image 20240123012949.png]]
```

- In robot localization:
  - We know the map, but not the robot's position
  - Observations may be vectors of range finder readings
  - State space and readings are typically continuous (works basically like a very fine grid) and so we cannot store B(X)
  - Particle filtering is a main technique

Robot Mapping

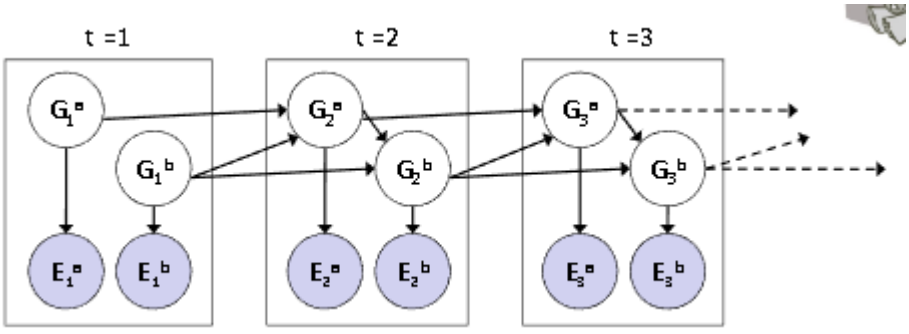
```
![[Pasted image 20240123012937.png]]
```

- SLAM: Simultaneous Localization And Mapping
  - We do not know the map or our location
  - State consists of position AND map!
  - Main techniques: Kalman filtering (Gaussian HMMs) and particle methods

Dynamic Bayes Nets

```
![[Pasted image 20240123013011.png]]
```

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables from time t can condition on those from t-1

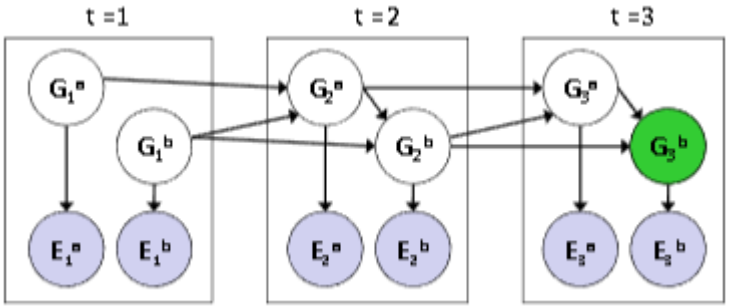


- Dynamic Bayes nets are a generalization of HMMs

Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets

- Procedure: “unroll” the network for T time steps, then eliminate variables until P(XT|e1:T) is computed



- Online belief updates: Eliminate all variables from the previous time step; store factors for current time only

DBN Particle Filters

- A particle is a complete sample for a time step
- **Initialize:** Generate prior samples for the t=1 Bayes net
  - Example particle: G_1^a = (3,3) G_1^b = (5,3)
- **Elapse time:** Sample a successor for each particle
  - Example successor: G_2^a = (2,3) G_2^b = (6,3)
- **Observe:** Weight each entire sample by the likelihood of the evidence conditioned on the sample
  - Likelihood: P(E_1^a |G_1^a) * P(E_1^b |G_1^b)
- **Resample:** Select prior samples (tuples of values) in proportion to their likelihood

Most Likely Explanation

```
![[Pasted image 20240123015115.png]]
```

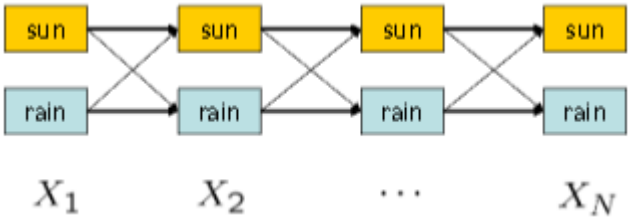
HMMs: MLE Queries

```
![[Pasted image 20240123015501.png]]
```

- HMMs defined by
  - States X
  - Observations E
  - Initial distribution: $P(X_1)$
  - Transitions: $P(X|X_{-1})$
  - Emissions: $P(E|X)$
- New query: most likely explanation: $argmax_{x_{1:t}} P(x_{1:t}|e_{1:t})$
- New method: the Viterbi algorithm

State Trellis

- State trellis: graph of states and transitions over time



- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t|x_{t-1})P(e_t|x_t)$

- Each path is a sequence of states
- The product of weights on a path is that sequence's probability along with the evidence
- Forward algorithm computes sums of paths, Viterbi computes best paths

Forward / Viterbi Algorithms

```
![[Pasted image 20240123015935.png]]
```

- Forward Algorithm (Sum)

$$f_t[x_t] = P(x_t, e_{1:t})$$

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) f_{t-1}[x_{t-1}]$$

- Viterbi Algorithm (Max)

$$m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$$

$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

Lecture 8 Naïve Bayes

Machine Learning

- Up until now: how use a model to make optimal decisions
- Machine learning: how to acquire a model from data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. BN graphs)
  - Learning hidden concepts (e.g. clustering)
- Today: model-based classification with Naïve Bayes

Classification

```
![[Pasted image 20240123020304.png]]
```

Examples

Spam Filter

```
![[Pasted image 20240123130110.png]]
```

Digit Recognition

```
![[Pasted image 20240123130121.png]]
```

Other Classification Tasks

- Classification: given input x, predict labels (classes) y
- Classification is an important commercial technique.

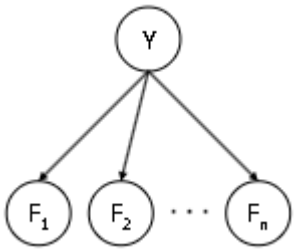
Model-Based Classification

```
![[Pasted image 20240123140112.png]]
```

- Model-based approach
  - Build a model (e.g. Bayes' net) where both the output label and input features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features
- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?

Naïve Bayes for Digits

- Naïve Bayes: Assume all features are independent effects of the label



  - What we will do is, we will assume that there is one special node, it's going to be the label. Here, that's written out as Y and that label his domain will be various digits 0-9.
- Simple digit recognition version:
  - One feature (variable) F_{ij} for each grid position `<i,j>`
  - Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.
    - 1 $\rightarrow$ < F_{0,0}=0, F_{0,1}=1, \cdots , F_{15,15} =0 >
  - Here: lots of features, each is binary valued
- Naïve Bayes model:
  - P(Y|F_{0,0} \cdots F_{15,15}) ∝ P(Y)·∏_{i,j} P(F_{i,j}|Y)
- What do we need to learn?

General Naïve Bayes

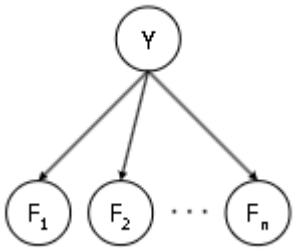
- A general Naïve Bayes model:

$$P(Y, F_1 \dots F_n) = P(Y) \prod_i P(F_i|Y)$$

|Y| parameters

|Y| x |F|^n values

n x |F| x |Y| parameters



- We only have to specify how each feature depends on the class

- Total number of parameters is **linear** in n
- Model is very simplistic, but often works anyway

Inference for Naïve Bayes

- Goal: compute posterior distribution over label variable Y
  - Step 1: get joint probability of label and evidence for each label
  - Step 2: sum to get probability of evidence
  - Step 3: normalize by dividing Step 1 by Step 2

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \Rightarrow \begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix} \Rightarrow \frac{\quad}{P(f_1 \dots f_n)} \Rightarrow P(Y|f_1 \dots f_n)$$

General Naïve Bayes Continued

- What do we need in order to use Naïve Bayes?
  - Inference method (we just saw this part)
    - Start with a bunch of probabilities: P(Y) and the $P(F_i|Y)$ tables
    - Use standard inference to compute $P(Y|F_1 \dots Fn)$
    - Nothing new here
  - Estimates of local conditional probability tables
    - P(Y), the prior over labels
    - $P(F_i|Y)$ for each feature (evidence variable)
    - These probabilities are collectively called **the parameter**s of the model and denoted by **θ**
    - Up until now, we assumed these appeared by magic, but…
    - …they typically come from training data counts: we'll look at this soon

Example: Conditional Probabilities

```
![[Pasted image 20240123141452.png]]
Where every grid is mapped with a probability of how likely this square is marked when we have
digit 3.
```

Naïve Bayes for Text

- Bag-of-words Naïve Bayes:
  - Features: W_i is the word at position i
  - As before: predict label conditioned on feature variables (spam vs. ham)
  - As before: assume features are conditionally independent given label
  - New: each W_i is identically distributed
- Generative model:
  - $P(Y, W_1 \cdots W_n) = P(Y) \prod_i P(W_i|Y)$

    - $P(W_i|Y)$ = Word at position i, not i^{th} word in dictionary!
- “Tied” distributions and bag-of-words
    - Usually, each variable gets its own conditional probability distribution P(F|Y)
    - In a bag-of-words model
        - Each position is identically distributed
        - All positions share the same conditional probs P(W|Y)
        - Why make this assumption?
            - e.g. for the purposes of detecting spam versus ham, it doesn't really matter whether a word occurs at position 23 or 24.
    - Called “bag-of-words” because model is insensitive to word order or reordering
        - And that means for a naïve bayes model for text, the only thing you actually have to learn is for each class, what is the histogram of words. What is the probability distribution of words in that class?

Example: Spam Filtering

```
![[Pasted image 20240123142437.png]]
```

- example below
- Tot field calculate the log value of total probability
    - $log(P(Y, W_1, \cdots, W_n)) = logP(Y) + \sum log(P(W_i|Y))$
- (prior) is P(Y)
- the first word is `Gray`

Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4
Gray	0.00002	0.00021	-11.8	-8.9
world	0.00069	0.00084	-19.1	-16.0
...	...	...	...	...
sleep	0.00006	0.00001	-76.0	-80.5

- P(spam | w) = 98.9
    - e^{-76.0} / (e^{-76.0} + e^{-80.5}) = 98.9

Training and Testing

```
![[Pasted image 20240123142841.png]]
```

Empirical Risk Minimization

- Empirical risk minimization
    - Basic principle of machine learning
    - We want the model (classifier, etc) that does best on the true test distribution
    - Don’t know the true distribution so pick the best model on our actual training set
    - Finding “the best” model on the training set is phrased as an optimization problem
- Main worry: overfitting to the training set
    - Better with more training data (less sampling variance, training more like test)
    - Better if we limit the complexity of our hypotheses (regularization and/or small hypothesis spaces)

    - In case we train our model too much, it will be too specific and model will be useless. So overfitting is a big issue in training.
    - Training more does not result in better model, train based on the patterns we want to find.

Important concepts

- Data: labelled instances, e.g. emails marked spam/ham
    - Training set
    - Held out set
    - Test set (data not in training data, to check if the model is accurate)
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
    - Learn parameters (e.g. model probabilities) on training set
    - (Tune hyperparameters on held-out set)
    - Compute accuracy of test set
    - Very important: never “peek” at the test set!
- Evaluation
    - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
    - Want a classifier which does well on test data
    - Overfitting: fitting the training data very closely, but not generalizing well
    - We’ll investigate overfitting and generalization formally in a few lectures

Generalization and Overfitting

```
![[Pasted image 20240123143438.png]]
```

Overfitting

```
![[Pasted image 20240123143501.png]]
```

Example

Digits

```
![[Pasted image 20240123143518.png]]
```

Spam and ham

```
![[Pasted image 20240123143545.png]]
```

South-west maybe occurs only once in ham but zero times in spam, this doesn't imply that it never occurs but it just didn't occur in our training data. Dangerous to give probabilities zero.

Generalization and Overfitting

- Relative frequency parameters will **overfit** the training data!

    - Just because we never saw a 3 with pixel (15,15) on during training doesn't mean we won't see it at test time
    - Unlikely that every occurrence of "minute" is 100% spam
    - Unlikely that every occurrence of "seriously" is 100% ham
    - What about all the words that don't occur in the training set at all?
    - In general, we can't go around giving unseen events zero probability
- As an extreme case, imagine using the entire email as the only feature (e.g. document ID)
    - Would get the training data perfect (if deterministic labeling)
    - Wouldn't generalize at all
    - Just making the bag-of-words assumption gives us some generalization, but isn't enough
- To generalize better: we need to **smooth** or **regularize** the estimates

Parameter Estimation

```
![[Pasted image 20240123144415.png]]
```

- Estimating the distribution of a random variable
- Elicitation: ask a human (why is this hard?)
    - In person
    - Sample size might be inaccurate
- Empirically: use training data (learning!)
    - E.g.: for each outcome x, look at the *empirical rate* of that value:

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

$$P_{\text{ML}}(\text{r}) = 2/3$$

    - This is the estimate that maximizes the *likelihood of the data*

$$L(x, \theta) = \prod_i P_\theta(x_i)$$

Smoothing

```
![[Pasted image 20240123144715.png]]
```

Maximum Likelihood?

- Relative frequencies are the maximum likelihood estimates

$$\theta_{ML} = \arg\max_\theta P(\mathbf{X}|\theta) = \arg\max_\theta \prod_i P_\theta(X_i) \Rightarrow P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

- Another option is to consider the most likely parameter value given the data

$$\theta_{MAP} = \arg\max_\theta P(\theta|\mathbf{X}) = \arg\max_\theta P(\mathbf{X}|\theta)P(\theta)/P(\mathbf{X}) = \arg\max_\theta P(\mathbf{X}|\theta)P(\theta)$$

????

Unseen Events

```
![[Pasted image 20240123145339.png]]
How do I estimate if the sun is going to rise, for now it always has rissen so probability one, but we all know that is not right. This is because we all know that for in the future the sun might die or it will fade away, so the actual probability will not be one.
```

Laplace Smoothing

So what Laplace said, is for every measurement we make, hold out an extra one for unseen events.

- Laplace's estimate:
  - Pretend you saw **every** outcome once more than you actually did

$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

- Can derive this estimate with Dirichlet priors (see cs281a)
- for some purpose like zero is not allowed.
- example
  - samples: red, red, blue
  - $P_{ML}(X) = (2/3, 1/3)$
  - $P_{LAP}(X) = (3/5, 2/5)$
    - adding 1 red, 1 blue
- Laplace's estimate (extended):
  - Pretend you saw **every** outcome **k extra** time

$$P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$$

  - What's Laplace with k = 0?
  - k is the **strength** of the prior
  - $P_{LAP,0}(X) = (2/3, 1/3)$
  - $P_{LAP,1}(X) = (3/5, 2/5)$
  - $P_{LAP,100}(X) = (102/203, 101/203)$ almost 50/50
    - So increasing k, we fit less
    - Decreasing k, we fit more
- Laplace for conditionals:
  - Smooth each condition independently:

$$P_{LAP,k}(x|y) = \frac{c(x,y)+k}{c(y)+k|X|}$$

\

Estimation: Linear Interpolation*

- In practice, Laplace often performs poorly for P(X|Y):
  - When |X| is very large
  - When |Y| is very large

- Another option: linear interpolation
  - Also get the empirical P(X) from the data
  - Make sure the estimate of P(X|Y) isn't too different from the empirical P(X)

$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha)\hat{P}(x)$$

  - What if α is 0? 1?
    - 0? Here our data will match more our empirical data then our estimate data.
    - 1? Here our our data will match more our estimations than what we observe.

Real NB: Smoothing

- For real classification problems, smoothing is critical
- New odds ratios:

$$\frac{P(W|\text{ham})}{P(W|\text{spam})} \qquad \frac{P(W|\text{spam})}{P(W|\text{ham})}$$

helvetica	:	11.4	
seems	:	10.8	
group	:	10.2	
ago	:	8.4	
areas	:	8.3	
...			

verdana	:	28.8
Credit	:	28.4
ORDER	:	27.2
<FONT>	:	26.9
money	:	26.5
...		

Do these make more sense?

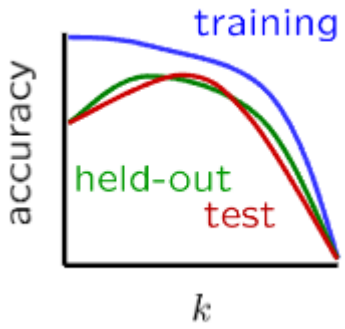
Data that didn't occur much will not be seen at the top, and will not overwhelm data that occurred more. So yes this makes more sense.

Tuning

```
![[Pasted image 20240123151200.png]]
```

Tuning on Held-Out Data

Because in case we train based on our training data, our tuning will most likely result in best case scenario in case we do no tuning or almost none.



- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g. the amount / type of smoothing to do, k, α
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data

        - Why?
    - For each value of the hyperparameters, train and test on the held-out data
    - Choose the best value and do a final test on the test data

Features

Errors, and What to Do

```
![[Pasted image 20240123154824.png]]
```

What to Do About Errors?

- Need more features– words aren't enough!
    - Have you emailed the sender before?
    - Have 1K other people just gotten the same email?
    - Is the sending information consistent?
    - Is the email in ALL CAPS?
    - Do inline URLs point where they say they point?
    - Does the email address you by (your) name?
- Can add these information sources as new variables in the NB model
- Next class we'll talk about classifiers which let you easily add arbitrary features more easily

Baselines

- First step: get a baseline
    - Baselines are very simple "straw man" procedures
    - Help determine how hard the task is
    - Help know what a "good" accuracy is
- Weak baseline: most frequent label classifier
    - Gives all test instances whatever label was most common in the training set
    - E.g. for spam filtering, might label everything as ham
    - Accuracy might be very high if the problem is skewed
    - E.g. calling everything "ham" gets 66%, so a classifier that gets 70% isn't very good…
- For real research, usually use previous work as a (strong) baseline

Confidences from a Classifier

- The **confidence** of a probabilistic classifier
    - Posterior probability of the top label

      $$\text{confidence}(x) = \max_y P(y|x)$$

    - Represents how sure the classifier is of the classification
    - Any probabilistic model will have confidences
    - No guarantee confidence is correct
- Calibration
    - Weak calibration: higher confidences mean higher accuracy
    - Strong calibration: confidence predicts accuracy rate
    - What's the value of calibration?

Lecture 9 Perceptrons and Logistic Regression

```
![[Pasted image 20240123155258.png]]
```

Linear Classifiers

```
![[Pasted image 20240123155356.png]]
```

Feature Vectors

```
![[Pasted image 20240123155416.png]]
input -> feature vector -> decision
```

Some (Simplified) Biology

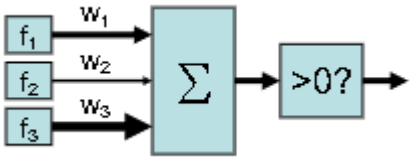
```
![[Pasted image 20240123163318.png]]
Different parts of the "brain" will help decide what decisions to make. A neuron has different
inputs and with these signals we make a decision.
```

Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

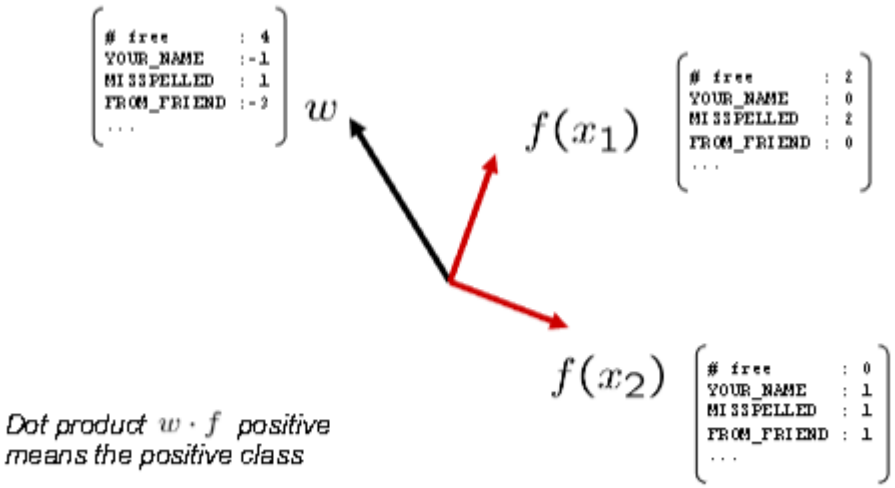
- If the activation is:
  - Positive, output +1
  - Negative, output -1



Weights

- Binary case: compare features to a weight vector

- Learning: figure out the weight vector from examples



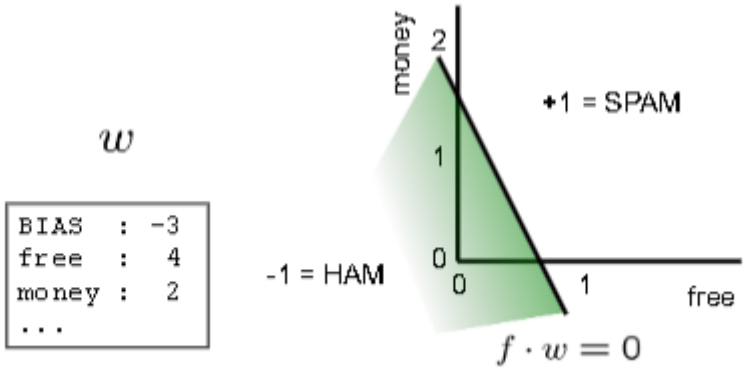
Positive when it's spam, and negative when it's not.
More features means, higher dimensions.
$f(x_1)$ will be in the positive class, because it's more aligned with the feature vector. While $f(x_2)$ is in the negative class, because it's more pointed in the opposite direction compared to the feature vector.

Decision Rules

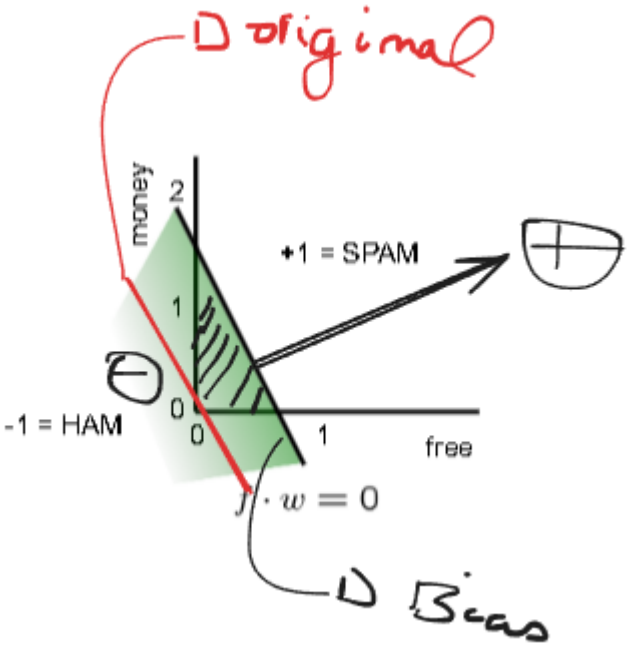
```
![[Pasted image 20240123163701.png]]
```

Binary Decision Rule

- In the space of feature vectors
- Any weight vector is a hyperplane
- One side corresponds to Y=+1
- Other corresponds to Y=-1



If we look at the weight vector, we can see that the vector is more pointed to free and a little to money.



Weight Updates

```
![[Pasted image 20240123164938.png]]
```

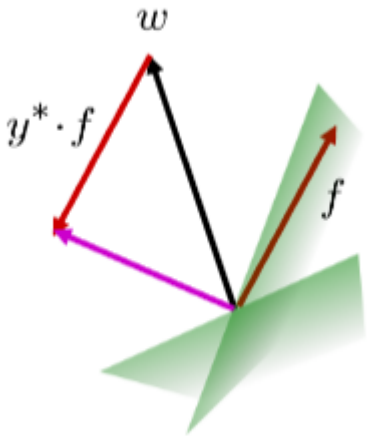
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
    - So in case, our true label is positive. So we predicted true and it is true.
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y *is -1.*
    - So our true label is negative.

$$w = w + y^* \cdot f$$



This new w, will be less aligned with f.
Let's say $w^{'} = w + y^{} \cdot f, then $w' \cdot f = (w + y^* \cdot f) \cdot f$

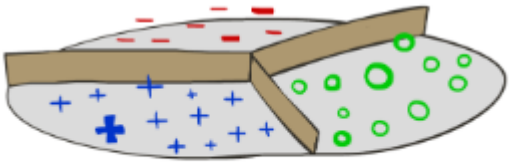
$= w \cdot f + y^* \cdot f^2$, as you can see we have the original and the correction and $f^2 > 0$.
So if y^* is negative, it moves away from f and if it's positive it will move closer to f.

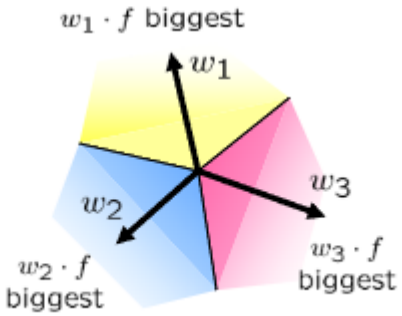
Example

```
![[Pasted image 20240123170535.png]]
In the beginning there might be many cases where we have adjustment, but eventually our model
stagnates and most cases are as predicted.
```

Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:
    - w_y
  - Score (activation) of a class y:
    - $w_y \cdot f(x)$
  - Prediction highest score wins
    - $y = argmax_y w_y \cdot f(x)$





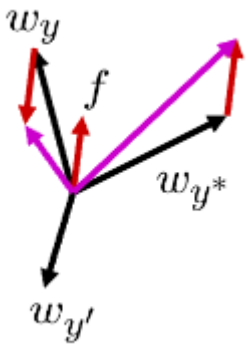
- Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights
  - $y = argmax_y w_y \cdot f(x)$
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

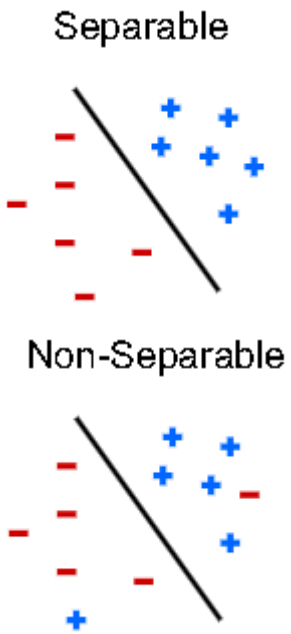


Example

```
![[Pasted image 20240123171226.png]]
```

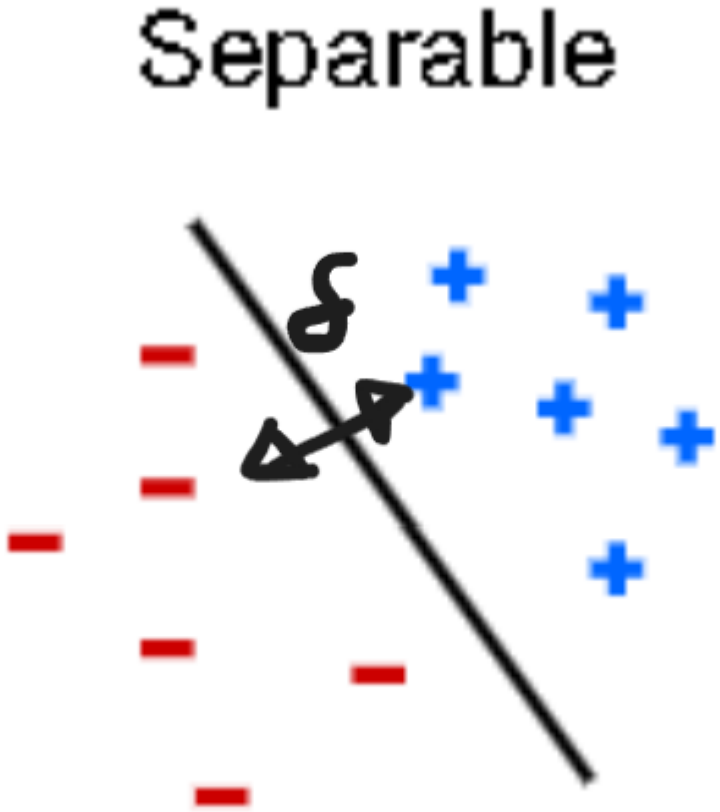
Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct



- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability
  - mistakes $< \frac{k}{\delta^2}$
    - Where k is number of features

- δ separability measure



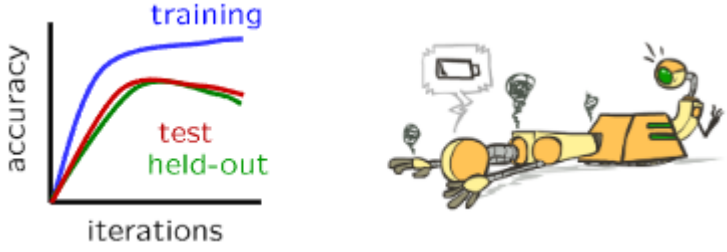
- So in other words, how further the classes are separated from each other the lower the chance is that we make a mistake.

Problems with the Perceptron

- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)

- Mediocre generalization: finds a "barely" separating solution

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



Improving the Perceptron

Non-Separable Case:

Deterministic Decision

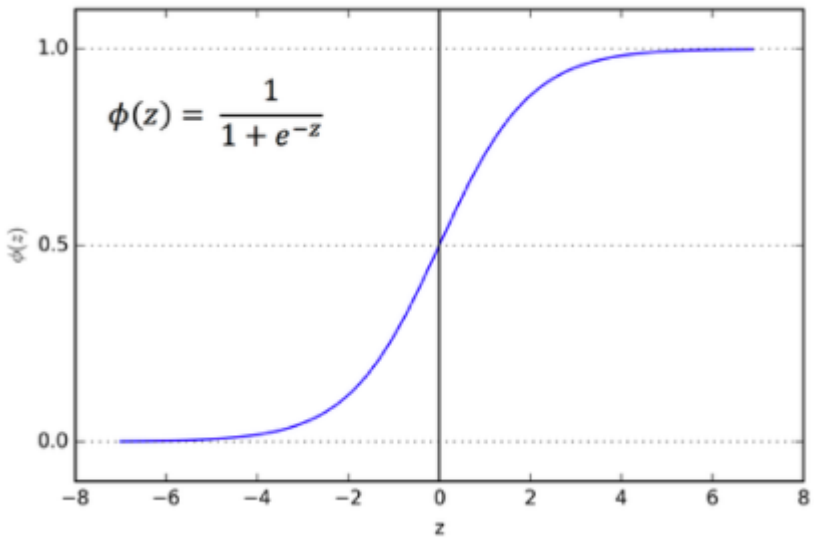
```
![[Pasted image 20240123174930.png]]
In case data is not fully separable, there will always be adjustments that have to be made. This
cause the function vector to move a lot and is not something we want.
```

Probabilistic Decision

```
![[Pasted image 20240123175103.png]]
To avoid this we can separate our data in probabilities, saying if it's between these two
probabilities, it's likely to be blue instead of red.
```

How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive -> want probability going to 1
- If $z = w \cdot f(x)$ very negative -> want probability going to 0
- Sigmoid function (could be any other function that behave in the same manner)
  - $\phi(z) = \frac{1}{1 + e^{-z}}$



In case z is positive, it will slowly go to one and if it's negative, it will slowly go to zero.

Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- with:

$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

  = Logistic Regression

Separable Cases

Deterministic Decision – Many Options

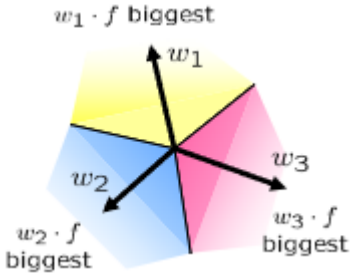
```
![[Pasted image 20240123182812.png]]
```

Probabilistic Decision – Clear Preference

```
![[Pasted image 20240123182838.png]]
```

Multiclass Logistic Regression

- Recall Perceptron
  - A weight vector for each class:
    - w_y
  - Score (activation) of a class y:
    - $w_y \cdot f(x)$
  - Prediction highest score wins
  - $y = argmax_y w_y \cdot f(x)$



- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

  z_1, z_2 and z_3 are numbers element of $[-\inf, \inf]$. While if we look at e^{z_i}, we can see that this will always be a positive number.

Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)};w)$$

- with:

$$P(y^{(i)}|x^{(i)};w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

> part two Optimization and Neural Nets

In this lecture we look at optimization

- i.e., how do we solve:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)};w)$$

Hill climbing

- Recall from CSPs lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighbouring state
  - If no neighbours better than current, quit
- What's particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
  - Infinitely many neighbours!
  - How to do this efficiently?

Optimization

1-D Optimization

```
![[Pasted image 20240123191459.png]]
```

- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$
  - Tells which direction to step into
  - This computes the linear approximations to our functions locally
    - Not hard to compute

2-D optimization

```
![[Pasted image 20240123192257.png]]
At the bottom of the graph, shows height contours. So the closer we go to the middle, the higher
```

the value.

Gradient Ascent (Descent, to find the bottom)

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate (this is just a choice).
- E.g., consider: $g(w_1, w_2)$
  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

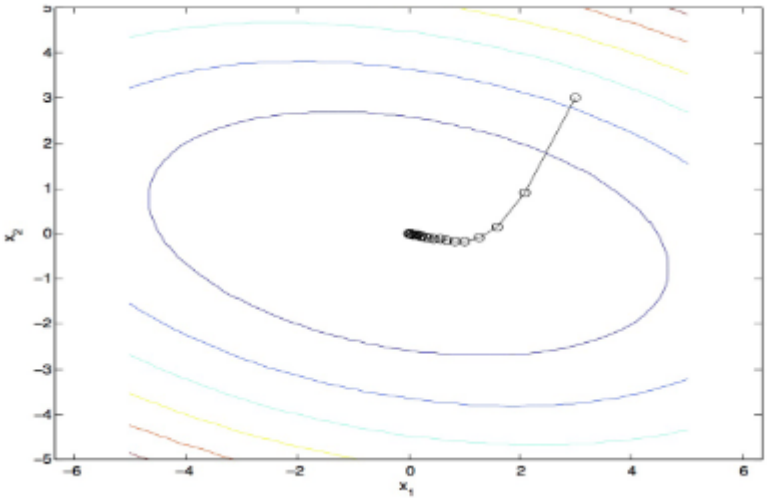
$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ = gradient

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction



What is the Steepest Direction?

```
![[Pasted image 20240123193053.png]]
```

Gradient in n dimensions

```
![[Pasted image 20240123193131.png]]
```

Optimization Procedure: Gradient Ascent

```
![[Pasted image 20240123193417.png]]
```

- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes about 0.1 – 1%

Batch Gradient Ascent on the Log Likelihood Objective

```
![[Pasted image 20240123194040.png]]
We do a bunch of updates and then sum it up and apply it on w.
```

Stochastic Gradient Ascent on the Log Likelihood Objective

```
![[Pasted image 20240123194502.png]]
Instead of doing a bunch of updates and sum it up. What if we only do a single update, can't it be that we then already have enough information to make changes on w. So here we will apply every update immediately on w.
Computing a gradient on a better point, is what the concept we go for here.
This is a good idea, but no parallelism.
```

Mini-Batch Gradient Ascent on the Log Likelihood Objective

```
![[Pasted image 20240123194747.png]]
This is a combination of both, where we take a batch of entries and make use of only that to train on w. So we train a batch and train it on a new point. This makes it that we can run it in parallel and update it on more of a recent w.
```

Lecture 10

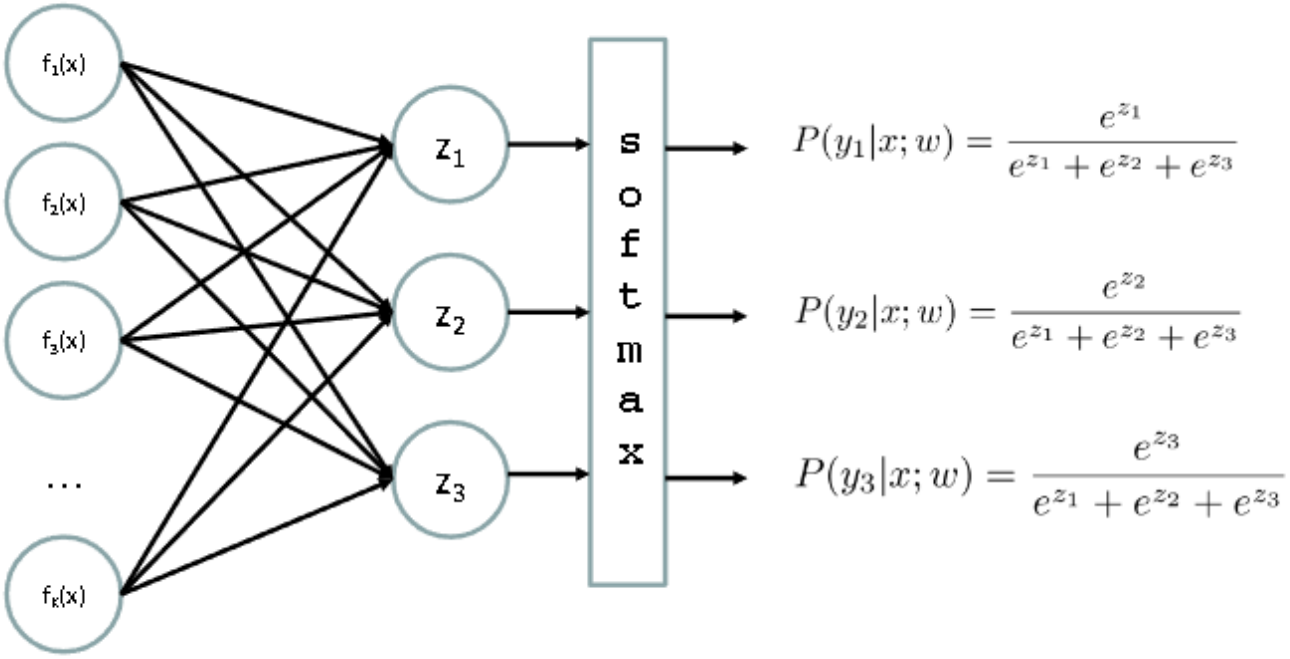
> Start of neural networks

Neural Networks

```
![[Pasted image 20240123195321.png]]
```

Multi-class Logistic Regression

- = Special case of neural network

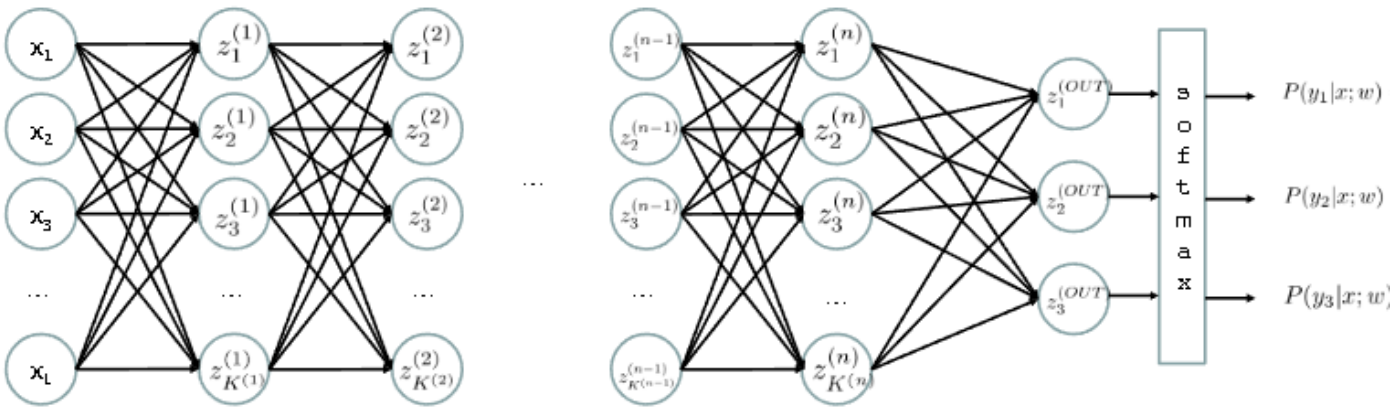


This is a good representation but knowing what our features are, is also really important. Because essentially that's what we will use to make our decisions.
But what if we want to learn the features to.

Deep Neural Network = Also learn the features!

```
![[Pasted image 20240123204120.png]]
We make use of mass amount computations to let it decide what features we want. They will be
flexible, so we can get more flexible features.
```

- Is this good?
  - Just taking weighted sum wouldn't work
  - Adding the g will make it nonlinear.
    - This is because we don't get anything new if we work with linear values and adding this g will do.



$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

g = nonlinear activation function

Common Activation Functions

```
![[Pasted image 20240123204539.png]]
These are commonly used Activation functions for
The yellow curve is the derivative, we need to use this too. So in case we use a different one, they also need to have a derivative.
Some problems have different needs, so using different Activation functions is might result in better results.
Not only that Hyperbolic Tangent centers more to zero, and this is also helpful for calculations.
```

Deep Neural Network: Also Learn the Features! Continued

- Training the deep neural network is just like logistic regression:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

  Just w tends to be a much, much larger vector :)
- Just run Gradient ascent
- - Stop when log likelihood of hold-out data starts to decrease

Neural Networks Properties

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.
  - If the network is big enough, we can learn the pattern.
- Practical considerations
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)

Universal Function Approximation Theorem*

```
![[Pasted image 20240123205638.png]]
```

- **In words:** Given any continuous function f(x), if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate f(x).

How about computing all the derivatives?

- Derivatives tables:

$$\frac{d}{dx}(a) = 0$$
$$\frac{d}{dx}(x) = 1$$
$$\frac{d}{dx}(au) = a\frac{du}{dx}$$
$$\frac{d}{dx}(u+v-w) = \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx}$$
$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx} - \frac{u}{v^2}\frac{dv}{dx}$$
$$\frac{d}{dx}(u^n) = nu^{n-1}\frac{du}{dx}$$
$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}}\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2}\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}}\frac{du}{dx}$$
$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u}\frac{du}{dx}$$
$$\frac{d}{dx}[\log_a u] = \log_a e\frac{1}{u}\frac{du}{dx}$$
$$\frac{d}{dx}e^u = e^u\frac{du}{dx}$$
$$\frac{d}{dx}a^u = a^u \ln a\frac{du}{dx}$$
$$\frac{d}{dx}(u^v) = vu^{v-1}\frac{du}{dx} + \ln u\ u^v\frac{dv}{dx}$$
$$\frac{d}{dx}\sin u = \cos u\frac{du}{dx}$$
$$\frac{d}{dx}\cos u = -\sin u\frac{du}{dx}$$
$$\frac{d}{dx}\tan u = \sec^2 u\frac{du}{dx}$$
$$\frac{d}{dx}\cot u = -\csc^2 u\frac{du}{dx}$$
$$\frac{d}{dx}\sec u = \sec u\tan u\frac{du}{dx}$$
$$\frac{d}{dx}\csc u = -\csc u\cot u\frac{du}{dx}$$

- But neural net f is never one of those?
  - No problem: CHAIN RULE:
    - If $f(x) = g(h(x))$
    - Then $f'(x) = g'(h(x))h'(x)$

Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function g(x,y,w)
  - Can automatically compute all derivatives w.r.t. all entries in w
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists

Summary of Key Ideas

- Optimize probability of label given input

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = “early stopping”)
- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - à the features are learned rather than hand-designed
  - Universal function approximation theorem
    - If neural net is large enough
    - Then neural net can represent any continuous mapping from input to output with arbitrary accuracy

    - But remember: need to avoid overfitting / memorizing the training data à early stopping!
- Automatic differentiation gives the derivatives efficiently

> Part two Neural Nets (wrap-up) and Decision Trees

How well does it work?

Computer vision

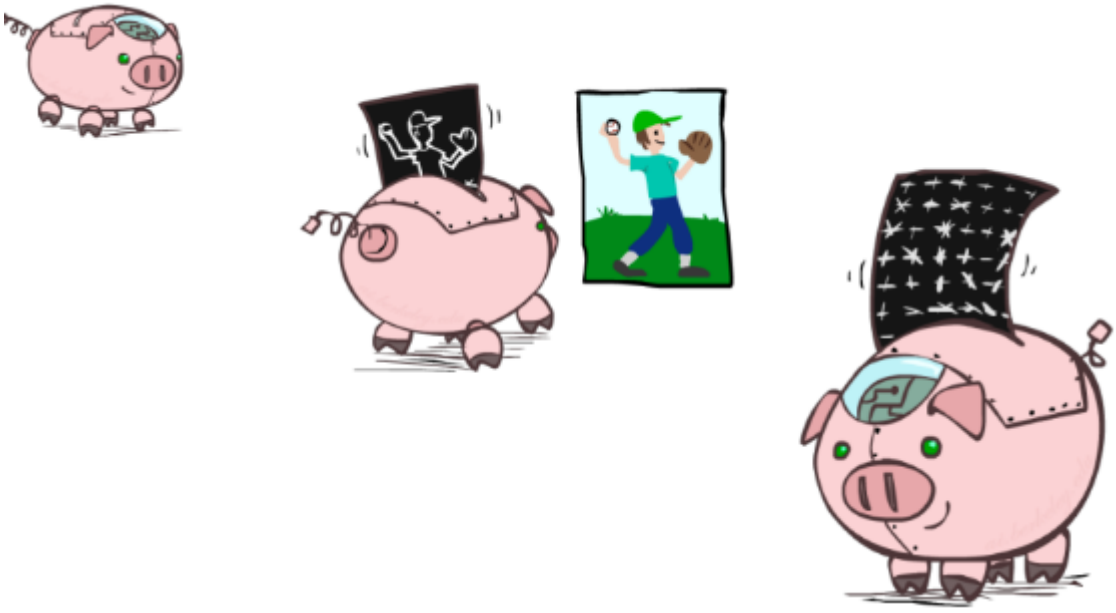
```
![[Pasted image 20240123215220.png]]
```

Object Detection

```
![[Pasted image 20240123215228.png]]
```

Manual Feature Design

This was how it used to be, to design these applications.



Features and Generalization

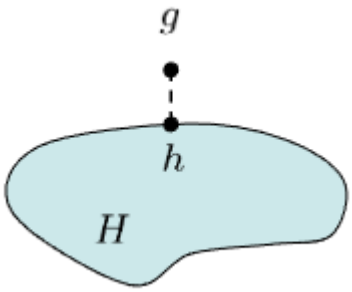
```
![[Pasted image 20240123215500.png]]
```

Inductive Learning

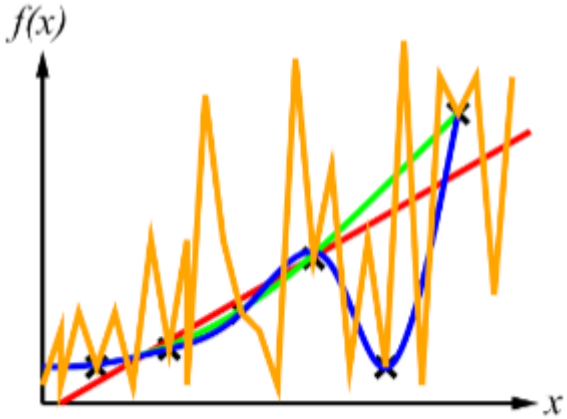
```
![[Pasted image 20240123215537.png]]
```

- Simplest form: learn a function from examples
    - A target function: g
    - Examples: input-output pairs (x, g(x))
    - E.g. x is an email and g(x) is spam / ham

- E.g. x is a house and g(x) is its selling price



- Problem:
  - Given a hypothesis space H
    - These are the function we might want to consider. All the functions that can represent is our hypothesis space and we want to find a h that closely represents g.
  - Given a training set of examples x_i
  - Find a hypothesis h(x) such that h ~ g
- Includes:
  - Classification (outputs = class labels)
  - Regression (outputs = real numbers)
- How do perceptron and naïve Bayes fit in? (H, h, g, etc.)
- Curve fitting (regression, function approximation):



  So choosing from a wider hypothesis space might not be always the solution. It might better but like previously mentioned, it might be overfitted.
- **Consistency** vs. **simplicity**
  - Consistency will give more consistent result
  - Simple function will generalize to new data better
  - Usually not achievable to gain both so we need to have a trade-off
- Ockham's razor

Consistency vs. Simplicity

- Fundamental trade-off: bias vs. variance
  - bias -> to small hypothesis space -> we can't match the training data -> the results will be more consistent so small variance
  - variance -> but if we have a complex system, we probably have a low bias or even zero but we choose a to specific one and could have high variance.
- Usually algorithms prefer consistency by default (why?)
  - We usually drive errors to zero in algorithms. So we are basically asking for it.
- Several ways to operationalize "simplicity"
  - Reduce the **hypothesis space**
    - Assume more: e.g. independence assumptions, as in naïve Bayes

    - Have fewer, better features / attributes: feature selection
    - Other structural limitations (decision lists vs trees)
  - **Regularization**
    - Smoothing: cautious use of small counts
    - Many other generalization parameters (pruning cut-offs today)
    - Hypothesis space stays big, but harder to get to the outskirts
      - Stopping early, or limiting amount of updates

Decision Trees

```
![[Pasted image 20240123220414.png]]
```

Reminder: Features

- Features, aka attributes
  - Sometimes: TYPE=French
  - Sometimes: $f_{TYPE=French}(x) = 1$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

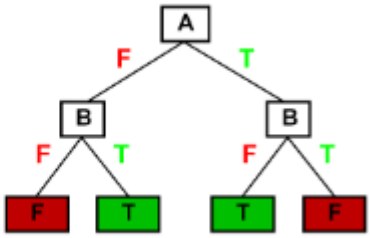
Decision Trees

- Compact representation of a function:
  - Truth table
  - Conditional probability table
  - Regression values
- True function
  - Realizable: in H

Expressiveness of DTs

- Can express any function of the features

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



P(C | A, B)

- However, we hope for compact trees

Comparison: Perceptrons

- What is the expressiveness of a perceptron over these features?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F

- For a perceptron, a feature's contribution is either positive or negative
  - If you want one feature's effect to depend on another, you have to add a new conjunction feature
  - E.g. adding "PATRONS=full ∧ WAIT = 60" allows a perceptron to model the interaction between the two atomic features
    - Everything contributes independently
- DTs automatically conjoin features / attributes
  - Features can have different effects in different branches of the tree!
  - So we invent features as we go on.
- Difference between modelling relative evidence weighting (NB) and complex evidence interaction (DTs)
  - Though if the interactions are too complex, may not find the DT greedily

Hypothesis Spaces

- How many distinct decision trees with n Boolean attributes?
  - = number of Boolean functions over n attributes
  - = number of distinct truth tables with 2n rows
  - = 2^(2n)
  - E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- How many trees of depth 1 (decision stumps)?
  - = number of Boolean functions over 1 attribute
  - = number of truth tables with 2 rows, times n
  - = 4n
  - E.g. with 6 Boolean attributes, there are 24 decision stumps
- More expressive hypothesis space:
  - Increases chance that target function can be expressed (good)
  - Increases number of hypotheses consistent with training set (bad, why?)
    - We might overfit and not generalize for new data.
  - Means we can get better predictions (lower bias)
  - But we may get worse predictions (higher variance)

Decision Tree Learning

- Aim: find a small tree consistent with the training examples

- Idea: (recursively) choose “most significant” attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an Attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

- So: we need a measure of how “good” a split is, even if the results aren’t perfectly separated out

Entropy and Information

- Information answers questions
  - The more uncertain about the answer initially, the more information in the answer
  - Scale: bits
    - Answer to Boolean question with prior <1/2, 1/2>?
    - Answer to 4-way question with prior <1/4, 1/4, 1/4, 1/4>?
    - Answer to 4-way question with prior <0, 0, 0, 1>?
    - Answer to 3-way question with prior <1/2, 1/4, 1/4>?
      - 1/2 1 bit + 1/4 2 bits + 1/4 2 bits = 3/2 bits we need
  - So depending on our distribution, the amount of information we need is different.
- A probability p is typical of:
  - A uniform distribution of size 1/p
  - A code of length log 1/p

Entropy

```
![[Pasted image 20240123223932.png]]
```

- General answer: if prior is <$p_1, \cdots, p_n$>:

- Information is the expected code length

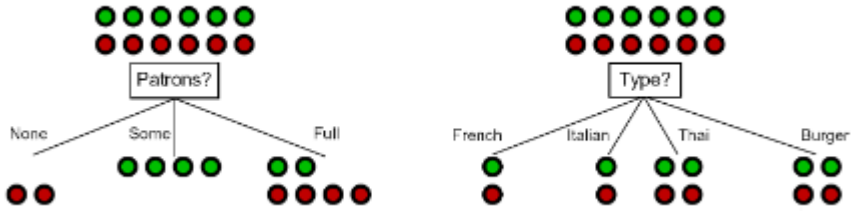
$$H(\langle p_1, \ldots, p_n \rangle) = E_p \log_2 1/p_i$$

$$= \sum_{i=1}^{n} -p_i \log_2 p_i$$

- Also called the entropy of the distribution
  - More uniform = higher entropy
  - More values = higher entropy
  - More peaked = lower entropy
  - Rare values almost “don’t count”

Information Gain

- Back to decision trees!
- For each split, compare entropy before and after
  - Difference is the **information gain**
  - Problem: there’s more than one distribution after split!



- Solution: use expected entropy, weighted by the number of examples

Next Step: Recurse

```
![[Pasted image 20240123224917.png]]
```

- Now we need to keep growing the tree!
- Two branches are done (why?)
- What to do under “full”?
  - See what examples are there…

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Han*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Examples

Learned Tree

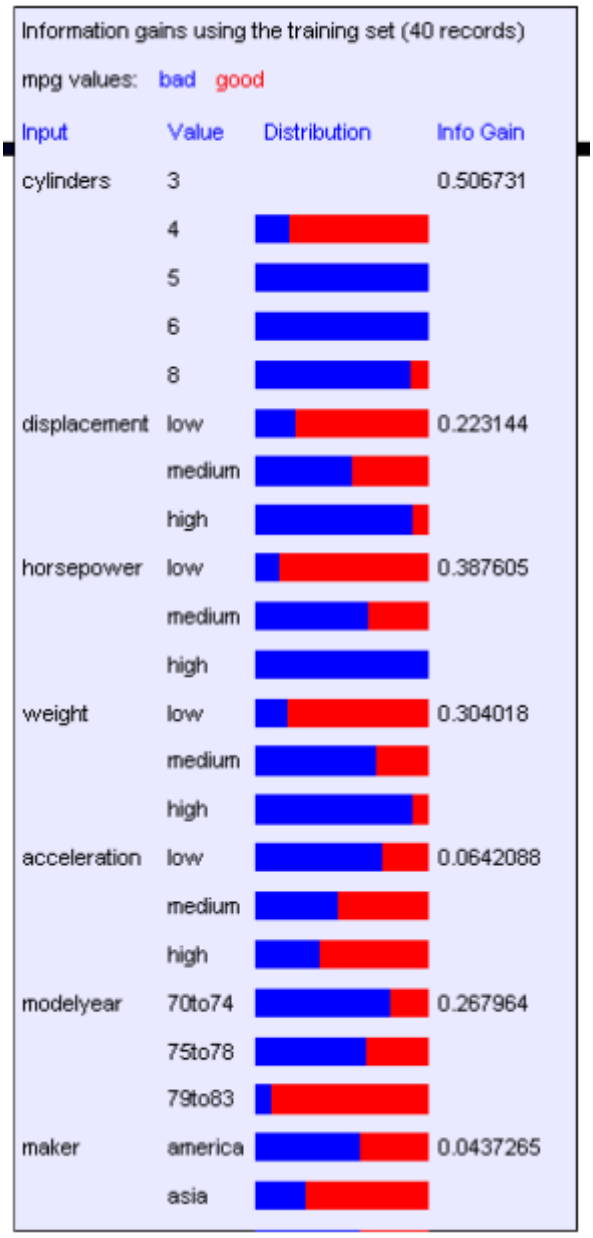
```
![[Pasted image 20240123224958.png]]
```

Miles Per Gallon

```
![[Pasted image 20240123225035.png]]
```

Find the First Split

- Look at information gain for each attribute
- Note that each attribute is correlated with the target!
- What do we split on?



Result: Decision Stump

```
![[Pasted image 20240123225152.png]]
```

Second Level

```
![[Pasted image 20240123225201.png]]
```

Final Tree

```
![[Pasted image 20240123225209.png]]
```

Reminder: Overfitting

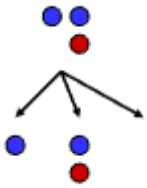
- **Overfitting:**
  - When you stop modelling the patterns in the training data (which generalize)
  - And start modelling the noise (which doesn't)
- We had this before:
  - Naïve Bayes: needed to smooth
  - Perceptron: early stopping

MPG Training Error

```
![[Pasted image 20240123225436.png]]
We overfitted, we fit every pattern in the training data.
![[Pasted image 20240123225445.png]]
```

Significance of a Split

- Starting with:
  - Three cars with 4 cylinders, from Asia, with medium HP
  - 2 bad MPG
  - 1 good MPG



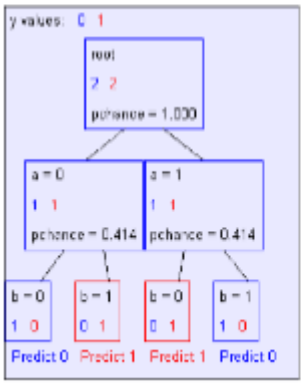
- What do we expect from a three-way split?
  - Maybe each example in its own subset?
  - Maybe just what we saw in the last slide?
- Probably shouldn't split if the counts are so small they could be due to chance
- A chi-squared test can tell us how likely it is that deviations from a perfect split are due to chance*
- Each split will have a significance value, p_{CHANCE}
  - If this probability is too high, we don't do the split

Keeping it General

- Pruning:
  - Build the full decision tree
  - Begin at the bottom of the tree
  - Delete splits in which $p_{CHANCE} > MaxP_{CHANCE}$
  - Continue working upward until there are no more prunable nodes
  - Note: some chance nodes may not get pruned because they were "redeemed" later

y = a XOR b

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

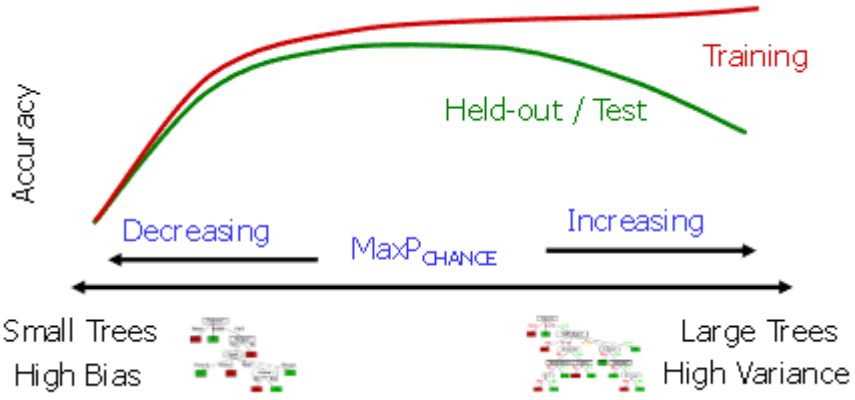


Example: Pruning

```
![[Pasted image 20240123225839.png]]
```

Regularization

- $MaxP_{CHANCE}$ is a regularization parameter
- Generally, set it using held-out data (as usual)



Two Ways of Controlling Overfitting

- Limit the hypothesis space
  - E.g. limit the max depth of trees
  - Easier to analyze
- Regularize the hypothesis selection
  - E.g. chance cut-off
  - Disprefer most of the hypotheses unless data is clear
  - Usually done in practice
