Artificial Intelligence (COMP30024) Summary Notes
Akira Wang

1 Agents

1.1 Agent Types

1.1.1 Simple Reflex Agents

- Very simple and works on a basis of *if X do Y*.
- Good for problems or tasks relating to categorization.

1.1.2 Model-Based Reflex Agents

- Has an *internal* representation of the current state.
- Requires a lot of memory.
- Will decide actions based on a current state (Given current state, what action should I take?

1.1.3 Goal-Based Agents

- Contains or generates *future* states.
- Has a *goal* to reach or achieve.
- Decides actions based on a current state and future predictions (Given current state and future states, what action should I take to get closer to the goal?)

1.1.4 Utility-Based Agents

- Similar to Goal-Based Agents but has utility values.
- Tries to maximize happiness.

1.2 Environment Types

Environments for agents may or may not be:

- **Observable vs Partially-Observable**: Can the agent see all the relevant information? Yes.
- **Deterministic vs Stochastic**: Is there are any probability involved for each action? Yes.
- **Episodic vs Sequential**: Does the current action depend on the past actions? No.
- **Static vs Dynamic**: Does the environment change while the agent is deciding on an action? No.
- **Discrete vs Continuous**: Is there a finite number of actions and state? Yes.

2 Problem Solving and Search Strategies

2.1 Formulating a Problem

- Initial State
- Actions
- Goal
- Path Cost

A solution is a sequence of actions leading from the initial state to the goal state. We measure if a solution is optimal by using the path cost.

2.2 Complexities of Search Strategies

A strategy is defined by picking the *order of node expansion* (usually in the form of a queue).

- **Completeness**: Does it always find a solution if it exists?
- **Time Complexity**: The number of nodes generated / expanded.
- **Space Complexity**: The maximum number of nodes in memory at any time.
- **Optimality**: Given a solution exists, does it always find the least-cost solution.

2.2.1 Terminology

Time and Space Complexity are measured in terms of

- b: The maximum branching factor of the search tree.
- d: The depth of the least-cost solution.
- m: The maximum depth of the state space (could be ∞).

2.3 Uninformed Search Strategies

Uninformed strategies only use the information available in the problem definition.

2.3.1 Breadth-First Search

- Expands the shallowest unexpanded node.
- Implementation: Queue (first in first out).
- **Complete** iff the branching factor b is finite.
- **Optimal** iff the path cost is *uniform*.
- **Time**: $\mathcal{O}(b^d)$
- **Space**: $\mathcal{O}(b^d)$

2.3.2 Breadth-First Search

- Expands the shallowest unexpanded node.
- Implementation: Queue (first in first out).
- **Complete** iff the branching factor b is finite.
- **Optimal** iff the path cost is *uniform*.
- **Time**: $\mathcal{O}(b^d)$
- **Space**: $\mathcal{O}(b^d)$

2.3.3 Uniform Cost Search

- Expands the least-cost unexpanded nodes in sorted order of their cost from the root.
    1. Expand the lowest cost node
    2. Sort all the nodes
    3. Visit the closest node
- Cost is calculated by the total cost required to travel from the root to node.
- Implementation: Queue (first in first out).
- **Complete** if there are weights (path cost).
- **Optimal**.
- **Time**: $\mathcal{O}$(number of nodes with g path cost which is $\leq$ cost of optimal solution)
- **Space**: $\mathcal{O}$(number of nodes with g path cost which is $\leq$ cost of optimal solution)

2.3.4 Depth-First Search

- Expands the deepest unexpanded node.
- Implementation: Stack (last in first out).
- **Not Complete** since it fails in infinite-depth spaces or spaces with loops.
- **Not Optimal**.
- **Time**: $\mathcal{O}(b^m)$
- **Space**: $\mathcal{O}(bm)$

2.3.5 Depth-Limited Search

- Same as DFS but has a limiting depth limit l.
- Implementation: DFS with threshold l.
- **Not Complete**.
- **Not Optimal**.
- **Time**: $\mathcal{O}(b^l)$
- **Space**: $\mathcal{O}(bl)$

2.3.6 Iterative Deepening Search

- Expands the deepest unexpanded node iteratively.
- Implementation: Stack (last in first out).
- **Complete**.
- **Optimal** iff the step size is 1.
- **Time**: $\mathcal{O}(b^d)$
- **Space**: $\mathcal{O}(bd)$
- Essentially combines the best of DFS and BFS

2.3.7 Bidirectional Search

- Searches from both the goal state and initial state.
- **Complete**.
- **Optimal** if done with the correct strategy (such as BFS from both sides).
- **Time**: $\mathcal{O}(b^{\frac{d}{2}})$
- **Space**: $\mathcal{O}(b^{\frac{d}{2}})$

2.4 Informed Search Algorithms

2.4.1 Best-First Search

- Uses an *evaluation function* for each node to estimate its *desirability*.
- The algorithm works by expanding the mos *desirable* unexpanded node.
- Implementation: Priority Queue
- Variants: A* and Greedy Best-First Search.

2.4.2 Greedy Best-First Search

- Expands the edge with the lowest evaluated cost.
- Heuristic function $h(n)$ is the estimated remaining distance to goal node.
- Not Complete.
- Not Optimal (since it chooses locally optimal choices).
- **Time**: $\mathcal{O}(b^m)$
- **Space**: $\mathcal{O}(b^m)$

2.4.3 A* Search

- Avoids expanding paths that are already expensive.
- Keeps an ordered list of evaluated path costs and expands the lowest cost.
- Evaluates cost with $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of reaching node n and $h(n)$ is the estimated cost to goal from node n.
- Complete.
- Optimal.
- **Time**: Exponential growth dependent on the error rate of the heuristic.
- **Space**: Stores all nodes in memory.
- Variants: IDA*, D*, H*

2.4.4 Heuristics

A heuristic function is given as $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of reaching node n and $h(n)$ is the estimated cost to goal from node n.

- If $h(n) > g(N)$, then A* becomes a Greedy Best-First Search.
- If $h(n) = 0$, then A* becomes Uniform Cost Search.
- If $h(n)$ is too small, then it will take too long to reach the goal, whilst if $h(n)$ is too large, it overestimates the goal and becomes non-optimal.

- If $h(n)$ is equal to the cost of moving from node n to the goal, then A* will only follow the **best path** and **never** expand other nodes, making it very fast.

Essentially, you want a heuristic such that $h(n) \rightarrow$ true path cost to goal.

Examples of well-known heuristic functions include: *Manhattan Distance, Diagonal Distance, and Euclidean Distance.*

2.4.5 Admissible Heuristics

An admissible heuristic is when the heuristic value is **always** lower than the cost of getting to the goal. If a heuristic is admissible, then it will **always** find an optimal path to the goal. If a heuristic is **not** admissible, then it will **not** find an optimal path to the goal.

Admissible heuristics can be derived from the exact solution cost by using a *relaxed* version of the problem.

- If the rules of 15-puzzle are relaxed such that a tile can move anywhere, then the heuristic is admissible.
- If the rules of 15-puzzle are relaxed such that a tile can move to any adjacent square, then the heuristic is admissible.
- For Chexers, if the game is relaxed such that a piece can jump at any time, then the heuristic is admissible.

2.4.6 Iterative Improvement Algorithms

In many optimization problems, the path is irrelevant, rather the goal state itself is the solution.

Then, our state space becomes a set of **complete** configurations.

2.4.7 Hill-Climbing / Gradient Descent

Intuition: depending on the initial state, an algorithm may converge to a local minimum rather than a global minimum.

- Only knows what the current state is (no history).

- May get stuck in a local maximum unless it is convex.

3 Game Playing and Adversarial Search

3.1 Types of Games

	Deterministic	Stochastic
Perfect Information	chess, checkers, go, othello	backgammon, monopoly
Imperfect Information		bridge, poker, scrabble, warfare

3.2 Representing a Game as a Search Problem

We can formally define a strategic two-player game by:

- Initial State
- Actions
- Terminal Test (win / loss / draw)
- Utility Function - The numeric reward for a certain outcome

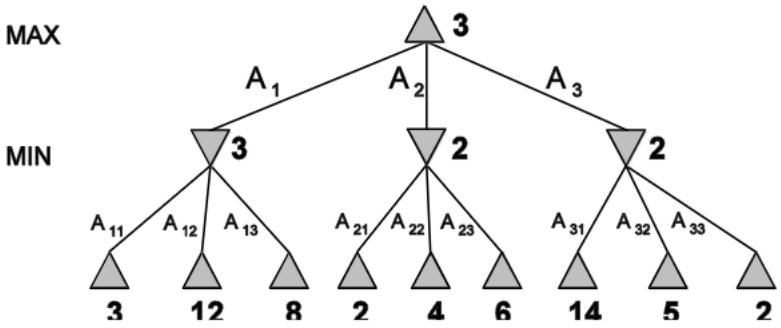
In a zero-sum game with 2 players, each player's utility for a state are equal and opposite.

3.3 Minimax

Intuition: Perfect play for **deterministic**and **perfect information**games. The idea is to choose to move to a position with the **highest minimax value** which is the best possible playoff against best play.

- Checks all possible moves for their value up to a certain level
- Calculate the utility value with an evaluation function (equivalent of A* and its heuristic)
- We wish to **maximize** our *utility value* when it is our turn, and **minimize** the *enemy utility value*.

- Absolute **perfect play** with an *infinite* look ahead (given it can generate all board states)



example.PNG

- Implementation: Recursive
- **Complete**.
- **Optimal** against optimal opponents (will still perform well against non-optimal opponents).
- **Time Complexity**: $\mathcal{O}(b^m)$
- **Space Complexity**: $\mathcal{O}(bm)$

To increase the speed of Minimax in problems with time constraints:

- **Cutoff test** (a depth limit) is used instead of **Terminal test**.
- **Evaluation function** (the estimated desirability of the position) is used instead of the **Utility function**

3.4 Alpha-Beta Pruning

Intuition: Prunes parts of a minimax tree that have no impact, and therefore will not examine them.

Works the same as minimax, but every new minimum/maximum is updated. This means any minimum/maximum lower can be discarded and essentially pruned from the tree.

- Since it is not feasible to search the entire game tree, a depth limit usually needs to be set.

- Alpha-Beta is designed to select all good moves, but still has to calculate the values of **all** legal moves.
- **Not Complete!!!**.
- **Optimal** since it does not affect the final result.
- **Time Complexity**: $\mathcal{O}(b^{\frac{m}{2}})$

3.4.1 Expectiminimax

A variant of minimax that supports stochastic games (i.e backgammon) and allows for alpha-beta pruning.

Intuition: Adds a *chance node* which has a probability outcome. Since all probabilities must be accounted for, the branching factor is very large and the probability of reaching increased depth nodes becomes diminished.

Strategy: At most look 3-ply ahead to be realistic!!!

4 Machine Learning in Game Search

4.1 Book Learning

Aim: To learn the sequence of moves for important positions (such as opening moves).

- Consider chess, there are several opening move strategies such as *4-move checkmate*.
- Learn from previous games and mistakes - identify moves that will always lead to a good outcome.
- Find strategic positioning - such as corner tiles in 15-puzzle.

The question is, how can recognise *which moves* are important?

4.1.1 Search Control Learning

Idea to order actions so that alpha-beta can be fully utilized to prune as soon as possible.

Given that the moves are sorted, if we find the *first* utility value is smaller than the current maximum value, we can *prune* the rest of the branch.

4.1.2 Learning Evaluation Function Weights

Adjust the weights of the utility function based on experience. Eventually, the goal is to reach the **true utility** value through trial and error.

4.2 Gradient Descent Learning

We define the Error function as:

$$\text{Error} = \frac{1}{2}\sum(t-z)^2, \tag{1}$$

where t is the desired output, and z is the actual output.

We aim to find weight values such that Error(θ) is minimised.

Given our weight function $w_i = \frac{-\partial E}{\partial w_i}$, we can see that we take the negative of the partial derivative. This means that if the Error function points uphill, we go downhill and vice versa.

For the learning rate α,

- If large, we may overestimate the global minimum
- If small, it will take too long and become an infeasible method.

4.3 TDLeaf(λ) Algorithm

A variant of Temporal Difference learning combined with minimax.

Updates weights in the evaluation function to reduce difference in rewards predicted at different levels of the tree.

- Evaluation function $eval(s, w)$
- $S_1, \ldots, S_N$ where S_1 is the root (initial state) node and S_N is terminal (state) node.

- Reward function $r(S_N) \in [+1, 0, -1]$ (transformed using $tanh$)
- The best leaf found S_i^ℓ at the cut-off depth using minimax between $1, \ldots, N$.
- The difference d_i which is the reward between state S_i and S_{i+1}. **If the evaluation function is good, then d_i should be small.**
- The learning rate η.
- If $\lambda = 0$: Encourages the algorithm to make the current reward similar to the next state (we don't want to modify weights w_j much). We should use $\lambda = 0$ when we have a stable algorithm with good results (later stage of training).
- If $\lambda = 1$: Encourages the algorithm to adjust the weights significantly so that the current reward approaches the true reward. We should use large λ at the beginning of training when we have really bad weight estimates.

Possible environments for learning include:

- Learning from labelled examples / past games
- Learning by playing a skilled opponent
- Learning by playing against random moves
- Learning by playing against itself

5 Constraint Satisfaction Problems

A CSP is made up of three main parts:

- A set of variables X
- A set of domains for each variable D
- A set of constraints that must be fulfilled C
  - Unery constraint (itself - $SA \neq green$)
  - Binary constraint (2 variables - $SA \neq WA$)
  - Higher Order / Tertiary constraints (multiple variables - crypt-arithmetic column constraints)
  - A goal test which is defined of the constraints

Example:

- X can define the lectures of a given subject
- D refers to the times available for each lecture
- C is the constraint that all lecture times must be different to avoid clashes

5.1 Discrete Variables

Discrete variables have finite domain with roughly $\mathcal{O}(d^n)$ complete assignments, where n is the number of variables in the CSP. *Example: Timetable scheduling.*

5.2 Continuous Variables

Continuous variables have an infinite domain with linear constraints being solvable in polynomial time. *Example: Start / End times for Hubble Telescope observations.*

5.3 Standard Search Formulation (Incremental)

A naive approach to a CSP is to start with an empty assignment $\varnothing$, and assign value to an unassigned variable *given* that it does not conflict with the current assignment.

However, this will fail if there are **no legal** assignments and is **not** fixable.

5.4 Backtracking Search

Backtracking search is a Depth-First approach for CSPs with single-variable assignments. *If constraint fails, lets go back and try a different branch*

- Ends up making $n!d^n$ leaves - **unfeasible**.

General purpose methods can help speed up Backtracking:

- Which variables should be assigned next?

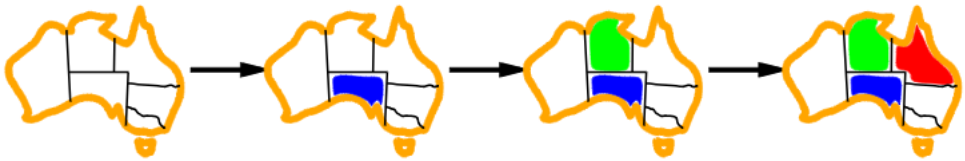
- What order of values should be tried?
- Can we detect certain failures early?
- Can we take advantage of the problem structure?

5.4.1 Minimum Remaining Values

Minimum remaining values will choose the variable with the fewest legal values.

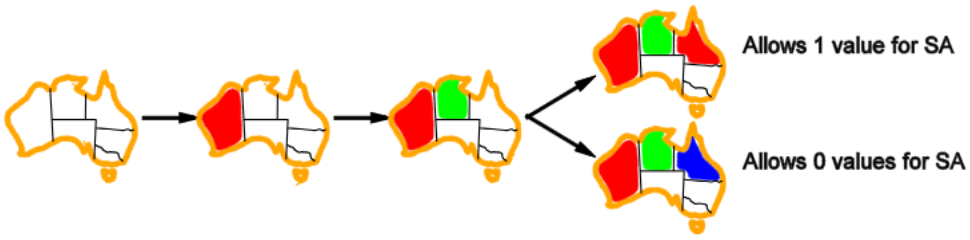
5.4.2 Degree Heuristic

Uses a Tie-Breaker among Minimum Remaining Value (MRV) variables. Chooses variables with the most constraints on remaining variables.



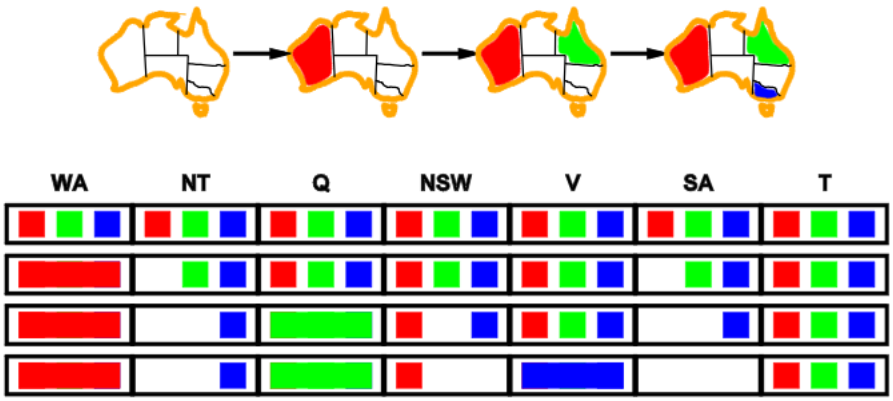
5.4.3 Least Constraining Value

Given a variable, choose the least constraining value which rules out the fewest values in the remaining variables.



5.5 Forward Checking

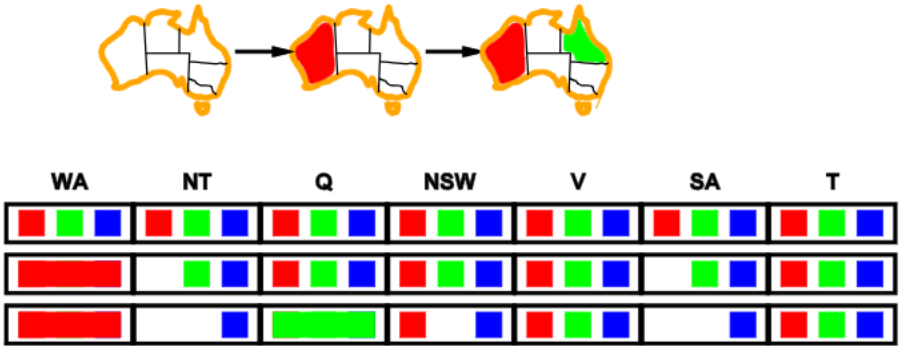
Intuition: Keep track of remaining legal values for unassigned variables. Then, terminate the search when any variable has **no** legal values remaining.



5.5.1 Constraint Propagation

Although Forward Checking propagates information from assigned to unassigned variables, it **does not** provide early detection for all failures.

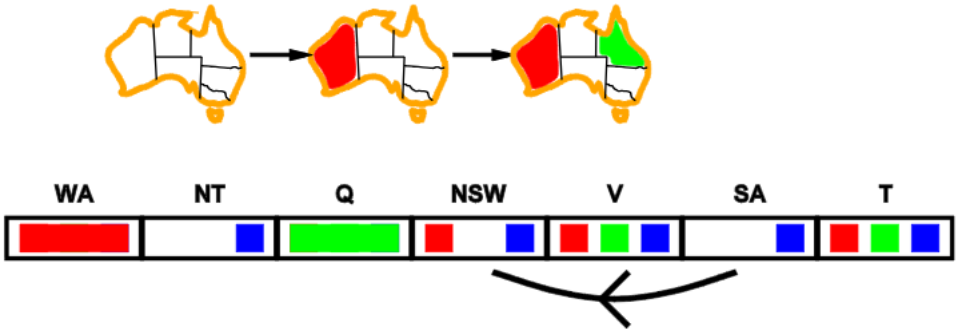
Consider the fact that NT and SA both cannot be blue - Constraint Propagation will repeatedly enforce constraints locally.



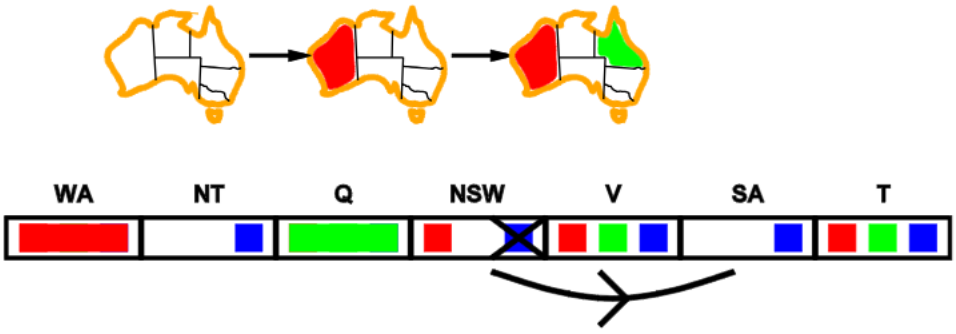
5.5.2 Arc Consistency / AC3

The simplest form of propagation makes *each* arc consistent.

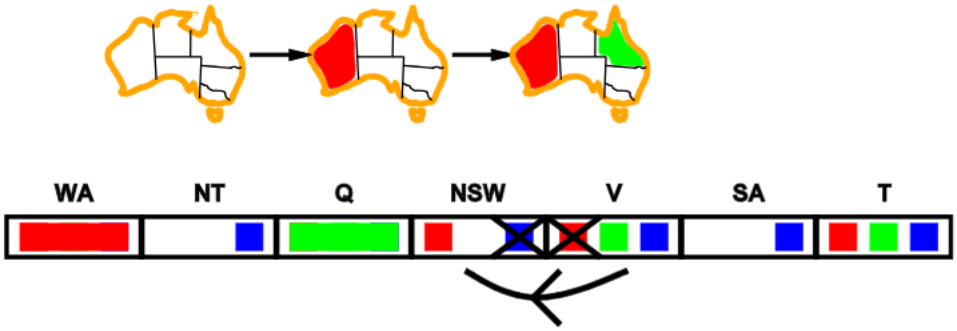
$X \to Y$ is arc consistent iff for *every* value of $x \in X$, there is *at least one value* of $y \in Y$ such that the constraint between X and Y is satisfied.



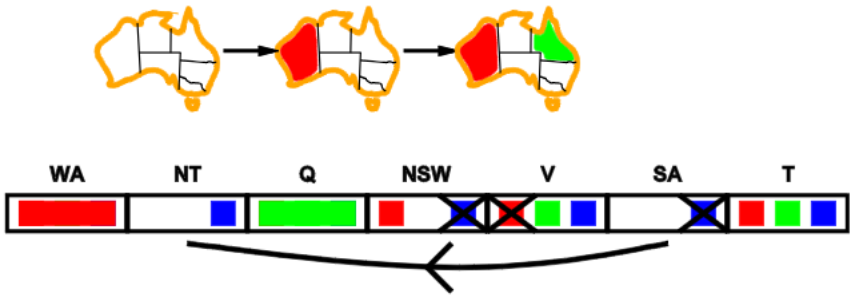
Step 1



Step 2



Step 3



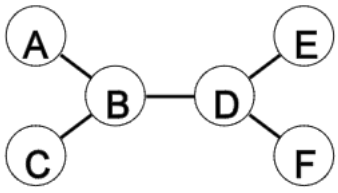
Step 4

If X loses a value then the neighbors of X needs to be rechecked. Arc consistency detects failure *earlier* than Forward Checking and can be run as a preprocessor after each assignment.

5.6 Problem Structures and Tree Structured CSPs

If a node is not part of the main graph, then they are **independent sub-problems**. Otherwise, they are identifiable as the *connected components* of the constraint graph.

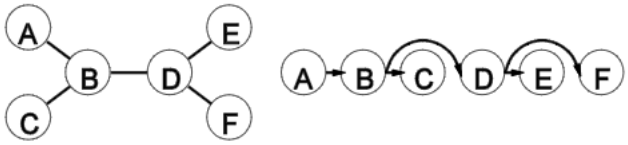
Consider an acyclic graph:



The CSP can then be solved in $\mathcal{O}(nd^2)$ compared to general CSPs with time complexity $\mathcal{O}(d^n)$.

5.6.1 Algorithm for Tree Structured CSPs

- Choose a variable to be the root, and order variables from root to leaves such that every node's parent precedes it in the ordering (topo-sort)



- For $j \in \text{range}(n, 2, -1)$, apply MakeArcConsistent($Parent(X_j), X_j$)
- For $j \in \text{range}(1, n)$, assign X_J consistently with Parent(X_j)

5.6.2 Nearly Tree Structured CSPs

Cutset Conditioning: Represent all possible ways, a set of variables such that the remaining constraint graph is a tree.

Cutset: Set of variables that can be deleted so the constraint graph forms a tree.

Let c be the Cutset size,

$$\text{Runtime} = \mathcal{O}(d^c \times (n-c)d^2). \tag{2}$$

5.6.3 Iterative Algorithms for CSPs (Local Search)

Recall the Gradient Descent algorithm. These typically work with *complete* states (i.e. all variables are assigned a value).

Local Search then tries to change one variable assignment at a time. To apply this to CSPs:

- Allow states with unsatisfied constraints (variable selection).
- Operators then *reassign* variable values (Value selection).

The variable is chose at random, and the value is chosen such that it violates the fewest constraints.

6 Complex Decisions - Auctions

6.1 Mechanism Design

Mechanism Design is the problem of how to design a game which results in maximising a global utility function in a multi-agent system, assuming each agent maximises their own rational strategy.

- **Single Item Auction**: There is a single item for sale with n bidders competing for the item. The outcome of the game is the choice of a winning player, and payment from each player.
- **Combinatorial Auctions**: A generalization of single item auctions and considers the problem of selling a set T of m items.
- **Public Projects**: There are n players where each player has a *private value* for building certain public projects (i.e a bridge). The outcome of the problem is the choice of whether or not to build the project, and a payment from each player covering the cost of the project if built. The utility of a player is determined by the value of the project if built, minus the payment.
- **Voting**: There are n players and m candidates running for office. Each player has a total preference order on the m candidates where the outcome of the problem is the choice of the winning candidate. The goal is usually to select the candidate that makes the most people *"happy"*.

6.1.1 Dimensions of Auction Protocols

- **Winner Determination**:
  - First-Price Auction - bidder with the highest bid is allocated the good(s).
  - Second-Price Auction - bidder with the highest bid wins, but pays the auctioneer the price of the second highest bidder.
- **Knowledge of Bids**:
  - Open-Cry - every bidder can see all bids from all other bidders.
  - Sealed-Bid - Bidder can only see its own bid(s), and not those of other bidder(s).
- **Order of Bids**:
  - One-Shot - each bidder makes one bid only.

  - Ascending - Each successive bid must exceed the previous bid

- **Order of Bids**:
  - One-Shot - Each bidder can only make a single bid.
  - Ascending - Each successive bid must exceed the previous bid.
  - Descending - Auctioneer starts from a high price, and each bid must be lower than the previous bid.

- **Number of Goods**:
  - Single Good - Only one indivisible good is for sale.
  - Many Goods (Combinatorial Auction) - Many goods are available in the auction, so bids can include both the price and number of goods wanted by the bidder.

6.2 Factors and Properties of an Auction

- A **common value** which means the good is the same worth to all bidders.
- A **private value** where the good reflects the worth of the good to the bidder.
- **Efficient**: The goods go to the agent who values them the most.
- **Discourage Collusion**: The auction mechanism should discourage illegal or unfair agreements between two or more bidders to manipulate prices.
- **Dominant Strategy**: There exists a *dominant* strategy for bidders, where a strategy is dominant if it gives the bidder a better pay-off than any other strategy.
- **Truth-Revealing**: The dominant strategy results in bidders revealing their true value for the good(s).

6.3 English Auction

First-Price, Open-Cry, Ascending.

Protocol and outcome rule:

- Auctioneer starts by asking for a minimum (reserve) price
- Auctioneer invites bids from bidders, where each bid must be higher than the current highest bid (maybe also have a minimum bid increment)

- Auction ends when no further bids are received and the good is sold if the final bid exceeds the reserve price
- Price paid by winner is their final (highest) bid

The **dominant strategy** is to keep bidding in small bids while the current cost is below your utility value for the good.

6.3.1 Properties of English Auctions

- Is **efficient** provided the reserve price is realistic.
- Can suffer from the **winner's curse**, where the winner may over-bid
- Susceptible to **collusion** since bidders can agree beforehand to keep bids artificially low.

6.4 Dutch Auction

Open-Cry, Descending

Protocol and outcome rule:

- Auctioneer starts by asking for an extremely high initial price
- Auctioneer repeatedly lowers the price of the good in small steps
- Auction ends when someone makes a bid at the current offer price
- Price paid by winner is the price where their bid was made

Suffers similar problems to the *English Auction*.

6.5 First-Price Sealed-Bid Auction

Protocol and outcome rule:

- Each bidder makes a single bid
- Bid sent to auctioneer so that the bidders cannot see each other's bid

- WInner is the bidder who made the highest bid
- Price paid by winner is the highest bid

The **dominant strategy** is unclear since you usually bid less than your true utility value. This however, depends on other agents bids which are unknown.

6.5.1 Properties of FP SB Auctions

- Is **not efficient** since agents with the highest utility value may not win the auction.
- Much simpler communication than English or Dutch auctions
- Sealed bids make it harder for collusion to occur

6.6 Vickrey Auction

Second-Price, Sealed-Bid

Protocol and outcome rule:

- Essentially the same as FP SB Auctions
- However, the price paid by the winner is the price of the **second highest bid**

The **dominant strategy** is to simply bid your max utility value for the good.

6.6.1 Properties of Vickrey Auctions

- Is **not efficient** and **truth revealing**, since the dominant strategy is to bid your max utility value
- Harder for collusion to occur and can be counter-intuitive for human bidders
- Computation is simple which makes it popular in multi-agent AI systems and online auctions

7 Probability (Bayesian Networks)

Yeh look... the notation's all over the place hahaha

7.1 Terminology and Notation

- $\neg$ is NOT
- $\wedge$ is AND
- $\vee$ is OR
- $P(a)$ is the *prior probability* of event $A = a$
- $\mathbf{P}(A)$ is the *probability distribution* of A
- A r.v. is titled (i.e. $Weather$)
- An event is lowercase (i.e. $Weather = sunny$)

7.2 Review of Probability Theorem

Conditional Probability:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} \tag{3}$$

Bayes':

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)} \tag{4}$$

7.3 Joint Density

Joint probability distribution for a set of r.v.s gives the probability of every atomic event on those r.v.s (i.e., every sample point)

$\mathbf{P}(Weather, Cavity)$ = a 4×2 matrix of values:

$Weather =$	$sunny$	$rain$	$cloudy$	$snow$
$Cavity = true$	0.144	0.02	0.016	0.02
$Cavity = false$	0.576	0.08	0.064	0.08

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

$$\begin{aligned}\mathbf{P}(A,B,C) &= \mathbf{P}(A|B,C)\mathbf{P}(B,C)\\ &= \mathbf{P}(A|B,C)\mathbf{P}(B|C)\mathbf{P}(C)\\ &= \mathbf{P}(A|C)\mathbf{P}(B|C)\mathbf{P}(C)\end{aligned}$$

$$\begin{aligned}\mathbf{P}(C|A\wedge B) &= \alpha\mathbf{P}(A\wedge B|C)\mathbf{P}(C)\\ &= \alpha\mathbf{P}(A|C)\mathbf{P}(B|C)\mathbf{P}(C)\end{aligned}$$

7.4 Inference by Enumeration

Start with the joint distribution:

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

For any proposition ϕ, sum the atomic events where it is true:
$P(\phi) = \sum_{\omega:\omega\models\phi} P(\omega)$

$P(toothache) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$

Start with the joint distribution:

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

For any proposition ϕ, sum the atomic events where it is true:
$P(\phi) = \sum_{\omega:\omega \models \phi} P(\omega)$

$P(cavity \vee toothache) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$

Start with the joint distribution:

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

Can also compute conditional probabilities:

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

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

Denominator can be viewed as a *normalization constant* α

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

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

7.5 Bayes' Rule and Conditional Independence

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

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

Total number of parameters is *linear* in n

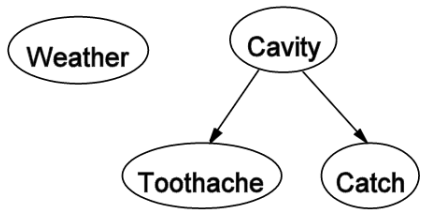
8 Bayesian Network

Bayesian networks are a simple graphical notation for conditional independence assertions. Hence, they are a more compact specification of full joint distributions.

- A set of nodes per variable
- A directed acylcic graph
- A conditional distribution for each node give its parents

Consider a topology of network encodes conditional independence assertions:

- $Weather$ is independent of the other variables
- $Tootache$ and $Catch$ are conditionally independent **given** $Cavity$



8.0.1 Example

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

Variables: $Burglar, Earthquake, Alarm, JohnCalls, MaryCalls$.

The network topology reflects "casual" knowledge:

- A burglar can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

8.1 Compactness

A CPT for Boolean X_i with k Boolean parents has 2^k rows for the 3 combinations of parent values.

Each row requires one number p for X_i =True (the number of X_i=False is $a - p$).

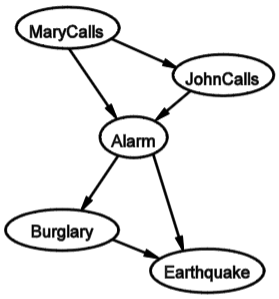
If each variable has no more than k parents, the complete network requires $\mathcal{O}(n \cdot 2^k)$ numbers.

8.2 Constructing Bayesian Networks

- Choose an ordering of variables $X_1, \ldots, X_n$
- For i in range(n):
  add X_i to the network
  select parents from $X_1, \ldots, X_{n-1}$ such that
  $\mathbf{P}(X_i|\text{Parents}(X_i)) = \mathbf{P}(X_i|X_1, \ldots, X_{i-1})$

Continuing from the previous example:

Suppose we choose the ordering M, J, A, B, E

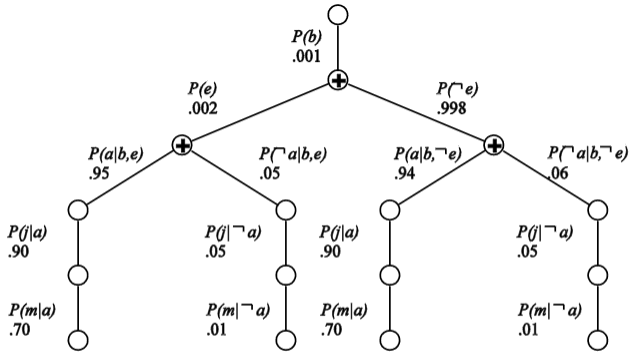


$P(J|M) = P(J)$? No
$P(A|J, M) = P(A|J)$? $P(A|J, M) = P(A)$? No
$P(B|A, J, M) = P(B|A)$? Yes
$P(B|A, J, M) = P(B)$? No
$P(E|B, A, J, M) = P(E|A)$? No
$P(E|B, A, J, M) = P(E|A, B)$? Yes

Deciding conditional independence is hard in non-casual directions.

Casual models and conditional independence is a much easier method (for us humans at least).

8.3 Evaluation Tree



Enumeration is inefficient: repeated computation
e.g., computes $P(j|a)P(m|a)$ for each value of e

9 Robotics

9.0.1 Manipulators

Degrees of Freedom correspond to the minimum number of positions to end-effector arbitrarily.

9.0.2 Non-Holonomic Robots

Robots with more DOF than controls are *non-holonomic*, and cannot generally transition between two infinitesimally close configurations.

Example: Cars.

9.1 Sensors

- **Range Finders:** Sonar (land, underwater), laser range finder, radar (aircraft), tactile sensors, GPS.
- **Imaging Sensors:** Cameras (visual, infrared).
- **Proprioceptive Sensors:** Shaft decoders (joints, wheels), inertial sensors (gyroscopes), force sensors, torque sensors.

We must also make assumptions about the way the world behaves in order to interpret the readings at all. For example:

- Some finite resolution sampling is sufficient to detect obstacles (consider an obstacle that consists solely of a hundreds long pins, sparsely distributed, pointing towards the sensor).
- Must know something about the structure of the robot to decide what an obstacle is.
- Given some sensor reading, only have a finite probability that it is correct.

9.2 Particle Filtering

One such approach to **localization** is *particle filtering*, which produces an approximate position estimate.

- Start with random samples from a uniform prior distribution for the robots position
- Update the likelihood of each sample using sensor measurements
- Re-sample according to update likelihoods
- Repeat until convergence to true location

Uncertainty of a robots state grows larger as it moves away until we find a landmark. However, this assumes that landmarks are *identifiable*, otherwise the posterior becomes *multimodal*.

9.3 Mapping

- **Localization:** Given a map and observed landmarks, update the pose distribution (robot configuration)
- **Mapping:** Given a pose distribution and observed landmark, update map distribution
- **Simultaneous Localization and Mapping (SLAM):** Given observed landmarks, update both the pose and map distribution.

9.4 Bayesian Inference on Sensors

Example:
Obstacle Detection:

- The odds of there being an obstacle present are 1 in 10
- The detector has a 5% FP rate and a 10% FN rate
- FP is "an object was there but NOT detected"
- FN is "an object was NOT detected, but was there

Our priors are therefore:

- $P(obstacle) = 0.1$
- $P(\neg obstacle) = 0.9$.

Then, our Sensor Model is:

- $P(FP|obstacle) = 0.9$
- $P(FN|obstacle) = 0.1$
- $P(FN|\neg obstacle) = 0.95$
- $P(FP|\neg obstacle) = 0.05$

If the sensor returns positive, we have:

$$\begin{aligned} P(obstacle|FP) &= \frac{P(FP|obstacle)P(obstacle)}{P(FP|obstacle)P(obstacle) + P(FP|\neg obstacle)P(\neg obstacle)} \\ &= \frac{0.9 \times 0.1}{0.9 \times 0.1 + 0.05 \times 0.9} \\ &= 0.667. \end{aligned}$$

On the contrary:

$$\begin{aligned} P(obstacle|FN) &= \frac{P(FN|obstacle)P(obstacle)}{P(FN|obstacle)P(obstacle) + P(FN|\neg obstacle)P(\neg obstacle)} \\ &= \frac{0.1 \times 0.1}{0.1 \times 0.1 + 0.95 \times 0.9} \\ &= 0.0116. \end{aligned}$$

9.4.1 Incremental Form of Bayes Law

Bayes Law can be extended to handle multiple measurements.

Given a set of independent measurements, what is the probability of the hypothesis being true?

If the measures are **independent** (NB assumption), we can use the incremental form.

- Given the *current* probability distribution $P(H)$
- Add a *new* measurement M
- What is the updated probability distribution $P(H)$?

To answer the final question,

$$P(H) \xleftarrow{M} \frac{P(M|H)}{P(M)} P(H). \tag{5}$$

9.5 Configuration Space Planning

The basic problem has infinite states, so we need to convert it to a **finite** space.

Cell Decomposition:

- Divide up space into simple cells
- Condition: Each cell needs to be traversed "easily" (convex)

Skeletonization:

- Identify finite number of easily connected points/lines that form a graph such that any two points are connected by a path on the graph