7CCSMAMS
Agents and Multi-Agent Systems

If you spot anything wrong, please let me know!
discord: @insertish
izzy · insrt.uk

Contents

1. Introduction

We are interested in agents that are:

- **autonomous**: decide for themselves what to do
- **self-interested**: act so as to try to achieve their goals
- **intelligent**: can work out the best way to act in a complex, unpredictable environment
- **cooperative**: can interact with other agents

Agents can be:

- Software programs
- Robots (including autonomous vehicles)
- People or other organisms

1.1. Reactive and proactive behaviour

Agents operate in multi-agent dynamic environments:

- They can't control the actions of other agents
- The environment can change in unexpected or unpredictable ways.

They need to be able to adapt to changes in the environment.

A **reactive system** is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).

Reacting to an environment can be easy (e.g. stimulus → response rules) but we want to be able to delegate tasks and goals to agents, hence the need for goal directed behaviour. **Pro-activeness** is generating and attempting to achieve goals; not driven solely by events; taking the initiative.

We want our agents to be reactive, responding to changing conditions in an appropriate (timely) fashion. We want our agents to systematically work towards long-term goals. These two considerations can be at odds with one another. Designing an agent that can effectively balance the two remains an open research problem.

1.2. Multi-agent domain implies need for social ability

The real world is a multi-agent environment: we cannot go around attempting to achieve goals without taking others into account, we need to coordinate with those around us.

Some goals can only be achieved with the cooperation of others.

Agents will be acting on behalf of users with different goals and motivations. Can't just command another agent to do something (agents are autonomous!), needs to be something in it for them.

Social ability in agents is the ability to interact with other agents: to successfully interact they will require the ability to cooperate, coordinate, and negotiate with each other.

1.3. Agents as intentional systems

We use **folk psychology** to predict and explain human behaviour through attribution of attitudes (hope, fear, etc). The attitudes in these descriptions are called **intentional** notions.

It is appropriate to use intentional notions to describe machines when we want to ascribe beliefs, free will, intentions, conciousness, abilities, and wants to a machine when that description expresses the same information as one would describe about a person. An ascription is useful when it helps us understand the structure of a machine, it's past behaviour, it's future behaviour, how to repair it, or how to improve it.

Why do we use intentional notion descriptions? Low-level mechanistic explanations are often impractical for complex systems, they can become tedious to maintain and understand.

We can classify intentional systems into different grades:

- A **first-order** intentional system has beliefs and desires but no beliefs and desires about beliefs and desires.
- A **second-order** intentional system is a superset of a *first-order* intentional system but it can also exhibit beliefs and desires (and no doubts about intentional systems) about beliefs and desires - of others and itself.

Prerequisite Maths Knowledge

Existential quanifier $\exists$ means 'there exists'.

Universal quantifier $\forall$ means 'for all'.

Set Builder Notation: $A = \{x | x \text{ has the property } P\}$ means 'A is the set of all x such that P holds for x'.

Set membership $x \in A$ means 'x is a member of the set A'.

Subsets $X \subseteq A$ means 'X is a subset of A'.
$X \subseteq A \Leftrightarrow \forall x \in X, x \in A$

Every member of X is also a member of A.

Proper Subsets $X \subset A$ means 'X is a proper subset of A'.
$X \subseteq A \Leftrightarrow \forall x \in X, x \in A \text{ and } \exists x \in A \text{ such that } x \notin X$

X is a subset of A that is not equal to A.

Power set (2^A or $\mathcal{P}(A)$) of a set A is the set of all subsets of A.

Formally, $2^A = \{X \mid X \subseteq A\}$.

Empty set ($\emptyset$ or $\{\}$) is the set with no elements in it.

Cartesian product ($A \times B$) of two sets A and B is $\{(x, y) \mid x \in A \text{ and } y \in B\}$.

Given sets A and B, a **function** $f : A \to B$ maps every element of A to exactly one element of B.

If f maps $x \in A$ to $y \in B$, we can also write $f(x) = y$.

2. Embedded Agents

Agents are embedded in an environment: agent affects and is affected by the environment. It experiences the environment through sensors and acts on it through effectors.

Agents exist in a **sense-decide-act** loop.

2.1. Environment Characteristics

An **accessible environment** is one in which an agent can obtain complete, accurate, up-to-date information about environment's state.

Most moderately complex environments are **inaccessible** which means we cannot have complete and accurate information about the environment's state.

A **deterministic** environment is one in which every action has a single guaranteed effect - there is no uncertainty about state resulting from a given action.

Non-deterministic environments are one in which actions do not guarantee a single effect.

A **static** environment is one in which the only changes to the environment are caused by actions made by the agent.

A **dynamic** environment is one in which there are other processes/agents operating in it, so it can change in ways beyond the agent's control.

2.2. Formal Specification

We assume the environment can be in any of a finite set E of discrete, instantaneous **states**:

$$E = \{e, e', ...\}$$

A set of **actions** Ac is available to the agent to perform:

$$Ac = \{\alpha, \alpha', ...\}$$

A **run** r of an agent in an environment is a sequence of interleaved environment states and actions.

$$e_0, \alpha_0, e_1, \alpha_1, e_2, \alpha_2, ..., \alpha_{u-1}, e_u$$

We use $\mathcal{R}$ to denote sets of possible runs:

- $\mathcal{R}$ is the set of all possible runs over E and Ac
- $\mathcal{R}^{Ac}$ is the set of all possible runs over E and Ac that end with an action
  i.e. $\mathcal{R}^{Ac} = \{r \mid r \in \mathcal{R} \text{ and the last element of } r \text{ is an action}\}$
- $\mathcal{R}^{E}$ is the set of all possible runs over E and Ac that end with an environment state
  i.e. $\mathcal{R}^{E} = \{r \mid r \in \mathcal{R} \text{ and the last element of } r \text{ is a state}\}$

A **state transformer function** represents the behaviour of the environment.

$$\tau : \mathcal{R}^{Ac} \to 2^E$$

This function takes a run that ends wth an action and returns a subset of E (states).

We can see the state transformer function is *history dependent* as the function takes as input a run and not just the current state and action.

We can see the state transformer function is *non-deterministic* as it returns a set of environment states, any of which can occur as a result of the input run.

If $r \in \mathcal{R}^{Ac}$ and $\tau(r) = \emptyset$ then the system has ended its run and we say r is **terminated**.

Example

Let $r = (e_0, \alpha_0, e_3, \alpha_2)$ and $\tau(r) = \{e_0, e_2\}$.

This tells us that if the agent starts in state e_0, then performs action α_0 which results in state e_3, and then performs action α_2, then it is going to end up either back in state e_0 again or in the state e_2.

An **agent** is (at the most abstract level) a function Ag that maps runs to actions.

$$Ag : \mathcal{R}^E \to Ac$$

This function takes a run that ends with an environment state (decision based on history) and returns exactly one action (assuming agents are deterministic).

Example

Let $r = (e_0, \alpha_0, e_3)$ and $Ag(r) = \alpha_7$.

This tells us that if the agent starts in e_0, performs α_0, and results in state e_3, then it will perform action α_7.

We let $\mathcal{AG}$ be the set of all agents.

An **environment** is a triple $\text{Env} = \langle E, e, r \rangle$ where:
- E is the set of environment states
- $e \in E$ is the initial state of the environment
- τ is the state transformer function

The set of all possible runs that can occur given an agent Ag embedded in an environment Env is:

$$\mathcal{R}(Ag, \text{Env})$$

Note: we assume that $\mathcal{R}(Ag, \text{Env})$ only contains terminated runs.

Condition for use

$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots) \in \mathcal{R}(Ag, \langle E, e, \tau \rangle)$ if and only if:
1. $e_0 = e$
2. $\alpha_0 = Ag(e_0)$
3. $\forall u > 0$:
   $e_0 \in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1}))$ and
   $\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$

Given two agents Ag_1 and Ag_2, they are **behaviourally equivalent with respect to environment** Env if and only if:

$$\mathcal{R}(Ag_1, \text{Env}) = \mathcal{R}(Ag_2, \text{Env})$$

Note: if this is true for any environment Env, we say the agents are **behaviourally equivalent**.

2.3. Utility Functions

A utility function can be defined over environment states as $u : E \to \mathbb{R}$ which maps each environment state to a real number, which represents how 'good' the state is.

A utility function can be defined over runs as $u : \mathcal{R} \to \mathbb{R}$ which maps each run to a real number, represnting how 'good' a run is. This can give us a longer-term view compared to just looking at environment states.

Agents aim to select actions that maximise their utility.

If an agent has a probabilistic model of the environment behaviour, then we can consider the expected utility.
The **expected utility** of a run is the probability that run will occur multiplied by the utility of the run.

Formally, $P(r \mid Ag, \text{Env})$ denotes the probability run r will occur when agent Ag is placed in the environment Env. So then, the expected utility of agent Ag in the environment is:

$$\sum_{r \in \mathcal{R}(Ag,\ \text{Env})} u(r) \times P(r \mid Ag, \text{Env})$$

The expected utility of the agent in the environment is equal to the sum of expected utilities of each run that may occur when the agent is in the environment.

An **optimal agent** in an environment is one that has the highest expected utility.

A key challenge with utility functions is being able to define them appropriately to engender the desired behaviour from the agent. If the domain is inherently numeric, then we may be able to think of ways to associate numbers with outcomes, but often isn't the case. Usually easier to think in terms of goals and whether they are achieved or not, rather than numeric utilities.

A **predicate task specification**, Ψ, is a special type of utility function that maps to 1 if the predicate is satisifed or 0 otherwise.

$$\Psi : \mathcal{R} \to \{0, 1\}$$

$\mathcal{R}_\Psi(Ag, \text{Env})$ denotes the set of all runs that can occur when agent Ag is in environment Env, so Ag is guaranteed to succeed if:

$$\mathcal{R}_\Psi(Ag, \text{Env}) = \mathcal{R}(Ag, \text{Env})$$

i.e. every possible run that can occur when Ag is in Env satisfies Ψ

Two common types of predicate task specifications are:
- **achievement tasks**: achieve state of affairs ϕ
- **maintenance tasks**: maintain state of affairs ψ

An **achievement task** is specified by a set $G \subseteq E$ of 'good'/'bad' states.

The agent succeeds if it brings about at least one of these states (we don't care which one - considered equally good). If Ψ is an achievement task specification, then:

$$\Psi(r) = 1 \text{ if and only if } \exists e_i \in \{e_x \mid e_x \text{ appears in } r\} \text{ such that } e_i \in G$$

A **maintenance task** is specified by a set $B \subseteq E$ of 'bad' states (states to avoid).

The agent succeeds if it manages to avoid all of the bad states. If Ψ is an maintenance task specification, then:

$$\Psi(r) = 1 \text{ if and only if } \forall e_i \in \{e_x \mid e_x \text{ appears in } r\} \text{ such that } e_i \notin B$$

3. Deductive, reactive, and hybrid reasoning agents

Agent architectures are a structure within which to implement with properties discussed (wk1&2).

3.1. Deductive agents

The 'traditional' approach to building agents is the **symbolic AI paradigm** where we provide a symbolic representation of the environment and the desired behaviour of the agent, which the agent explicitly reasons with.

Example Reasoning Algorithm

Let Ac be the set of actions the agent can perform.
Let ρ be a theory that encodes information about the behaviour of the environment and the desired behaviour of the agent.
Let Δ be a logical database that describes the current state of the world.

$\Delta \underset{\rho}{\vdash} \mathrm{Do}(\phi)$ means $\mathrm{Do}(\phi)$ can be logically deduced from Δ using ρ.

```
for each α ∈ Ac do
    if Δ ⊢_ρ Do(α) then
        return α
    end
end
```

Application of example problem

Let's suppose we have:

- $Ac = \{\text{study material}, \text{attend tutorial session}\}$
- $\rho = \{$
  $\text{Goal}(\text{learn about agents}),$
  $\text{have free time} \rightarrow (\text{study material} \rightarrow \text{Achieve}(\text{learn about agents})),$
  $\text{monday afternoon} \rightarrow (\text{attend tutorial session} \rightarrow \text{Achieve}(\text{learn about agents})),$
  $(\forall x, \forall y, (\text{Goal}(y) \wedge (x \rightarrow \text{Achieve}^y)) \rightarrow \text{Do}(x))$
  $\}$
- $\Delta = \{\text{monday afternoon}\}$

For each action α, the agent checks whether it can deduce $\mathrm{Do}(\alpha)$ from Δ using ρ.

Since the agent knows it is Monday afternon, it can deduce $\mathrm{Do}(\text{attend tutorial session})$ by:

1. monday afternoon
2. $\text{monday afternoon} \rightarrow (\text{attend tutorial session} \rightarrow \text{Achieve}(\text{learn about agents}))$
3. $\text{Goal}(\text{learn about agents}) \wedge (\text{attend tutorial session} \rightarrow \text{Achieve}(\text{learn about agents})) \rightarrow \text{Do}(\text{attend tutorial session})$

So the agent selects to attend the tutorial session.

The **transduction problem** tackles how to translate sensory input from the environment into an accurate and adequate symbolic description, in time for that description to be useful.

Example of transduction problem

An autonomous vehicle receives lots of information about its environment via its sensors and needs to translate this information into an accurate symbolic description of the environment. For example, a nearby pedestrian needs to be encoded into their position, speed, and direction of travel. (i.e. at(ped756, −75, 120), speed(ped756, 2.5), ..)

This is not a trivial task!

The **representation/reasoning problem** tackles how to symbolically represent information about complex real world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

This in particular looks at:

- what logic is appropriate for the problem (e.g. is propositional logic enough or is some more expressive higher-order logic required)
- which elements is it appropriate to represent (e.g. we want a pedestrian's position but not their eye/hair colour)

3.1.1. Concurrent MetateM

A symbolic reasoning agent in a dynamic environment needs to be able to react to events in the environment:

- it needs to be able to reason about what to do
- it needs not be constrained by its reasoning in having to do something **now** when it might need to react to other events

We need to be able to talk about 'sometime in the future' & similar.

Concurrent MetateM provides temporal logic operators:

> **Important**
>
> These will be given and defined clearly in the exam! Do not try to remember them…

$\bigcirc \phi$	ϕ true in next state
$\circledcirc \phi$	ϕ true in previous state
$\Diamond \phi$	ϕ true now or at some point in future
$\Box \phi$	ϕ true now and at all points in future
$\blacklozenge \phi$	ϕ true sometime in the past
$\blacksquare \phi$	ϕ true at every point in the past
$\phi \mathcal{U} \psi$	ψ true at some point in the future and ϕ true at all points until then
$\phi \mathcal{S} \psi$	ψ true at some point in the future and ϕ true at all points since then
$\phi \mathcal{W} \psi$	ψ true at all points in the future unless ϕ becomes true
$\phi \mathcal{Z} \psi$	like $\mathcal{S}$ but ψ may never have become true

Examples

- $\bigcirc \neg$ round(earth): it will be true in the next state that the Earth is not round
- $\Box$ year(2014): it is and always will be true that the year is 2014
- $\Diamond$ mares: it is now true or eventually be true that mares
- $\blacklozenge$ filly(you): at some point in the past, you were a filly

A **Concurrent MetateM agent** is specified using temporal logic, executed directly to generate its behaviour. Agents communicate with each other by broadcasting messages.

To define a concurrent MetateM agent, we need:
- set of message types that the agent listens for (to ignore broadcast messages that aren't one of these)
- set of message types that they broadcast
- set of rules that determine their behaviour

To define concurrent MetateM agent rules, we use one of two forms:
- antecedent about the past $\rightarrow$ consequent about the present and/or future
- *start* $\rightarrow$ consequent about the present and/or future
  Special case that only fires in the initial state.

Concurrent MetateM agents execute their specification as follows:
1. set environment propositions (representing agent's view of current state) according to messages received
2. check which rules fire by comparing antecedents with current history
3. jointly execute consequents of any rules that fire together with any commitments that need satisfying

If an agent is trying to satisfy multiple commitments, thne it gives precedence to those that are oldest.

Agent definitions for MetateM take the form:

```
id(in-message1, in-message2, ...)[out-message1, out-message2]:
  rule1
  rule2
  ...
```

- Where `id` is the agent identifier
- Where `in-message1`, `in-message2`, ... are message types the agent listens for
- Where `out-message1`, `out-message2`, ... are message types that the agent broadcasts
- Where `rule1`, `rule2`, ... are temporal logic rules that specify the agent's behaviour

An implementation and interactive demo is provided in `jupyter/concurrent-metatem.ipynb`!

3.2. Reactive agents

In traditional AI view, agents have a symbolic internal representation and they manipulate this representation in order to work out what to do next. But the reasoning (symbol manipulation) is resource demanding, and the connection between the real environment and the symbols that represent it is a challenge.

An alternative view is that agents don't need to be able to explicitly represent and reason about their environment to display intelligent behaviour but that intelligent behaviour emerges through interaction of various simple behaviours that are generated purely in response to the environment, hence producing **reactive agents**.

3.2.1. Subsumption architecture

Background and context to subsumption architecture

Brooks' identified two key ideas:

- 'real' intelligence is situated in the world, not in 'disembodied' systems such as theorems
- intelligent behaviour emerges as a result of agent's interaction with its environment

Arguing that:

- intelligent behaviour can be generated w/o explicit representations (like symbolic AI proposes)
- intelligent behaviour can be generated w/o abstract reasoning (like symbolic AI proposes)
- intelligence is an emergent property of certain complex systems

In the subsumption architecture, agents have a set of task accomplishing behaviour modules which take the form:

$$\text{situation} \rightarrow \text{action}$$

There is no complex symbol representation and no symbol reasoning, if the situation is true then the rule fires.

Since more than one behaviour module can fire at once, we need to decide which and when. For this, we consider organising the behaviour modules in a **subsumption hierarchy**, where behaviours are arranged into layers, with behaviours in lower layers taking higher priority.

e.g. leaf cutter ants create huge networks of paths, connect their nests and food sources, but do not have an explicit representation of the world, they operate under the subsumption architecture: their behaviour is generated directly in response to the environment where they react to obstacles or resources encountered, or pheromones left by other ants

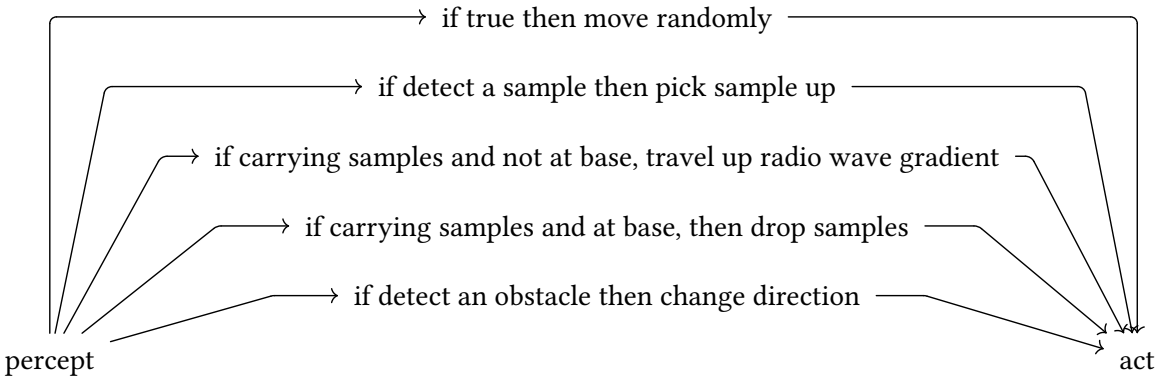
Example: Rover on Mars

Suppose a 'rock gathering on Mars' domain where agents are robots implemented in subsumption architecture. Aim is to collect as many samples as possible, agents have no prior knowledge of where rock samples are, or terrain of environment, but known that rock samples tend to be clustered together.

Given the modules:

- if carrying samples and not at the base then travel up radio wave gradient (to parent)
- if true then move randomly
- if detect an obstacle then change direction
- if carrying samples and at the base then drop samples
- if detect sample then pick sample up

What sort of subsumption hierarchy would you expect?



3.2.2. Limitations of reactive agents

Reactive agents pose several limitations:

- since agents do not maintain any kind of model of the environment, there must be enough information in local state to be able to identify the appropriate action; agents cannot take into account information not available in local environment
- overall behaviour of the system emerges from interactions between different agent behaviours and environment, so it is very hard to design a system to fulfill a specific task
- the more behaviours are available to the agents, the harder the behaviour is to understand/predict, since the dynamics can of interactions between different layers can become to complex to understand

3.3. Hybrid agents

Many researchers argue a bit of both approaches is necessary to build effective agents, so we can build **hybrid agents** that are built out of two or more subsystems consisting of deliberative and reactive models.

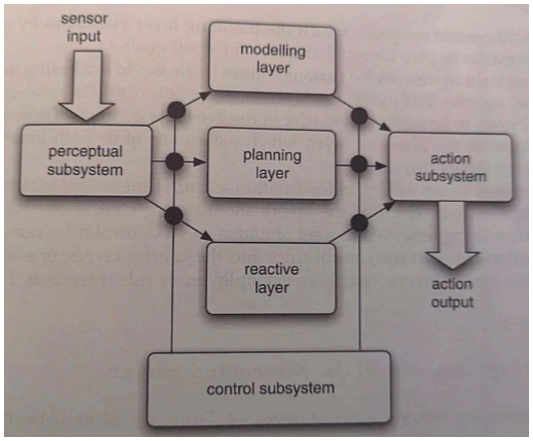
Similar to the subsumption architecture, the agent's control subsystems are arranged into a hierarchy, with higher layers dealing with information at increasing levels of abstraction.

Typical approaches to hierarchy are:

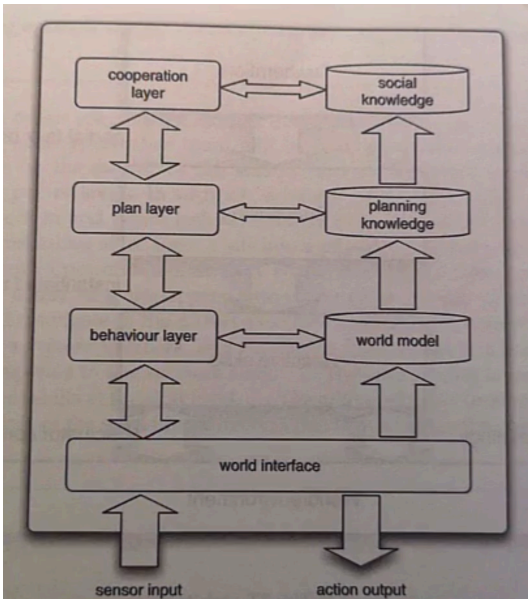
- *Horizontal layering* where layers are directly connected to the sensory input and action output. In effect, each layer itself acts like an agent, producing suggestions on actions. Control system determines which of the layers to give precedence to.
- *Vertical layering* where sensory input and action output are each dealt with by at most one layer at any point and the layers pass control to other layers where necessary.

Examples of hybrid agents:

TouringMachines is a horizontally layered architecture



InteRRaP is a vertically layered architecture



4. Practical reasoning agents

Practical reasoning agents take inspiration from the processes that seem to take place when humans decide what to do, specifically what goals to commit to and then deciding what to do to try to achieve those goals.

4.1. Deliberation and means-ends reasoning

Deliberation is deciding what state of affairs the agent wants to achieve.
The results of deliberation are referred to as the agent's **intentions**.
Means-end reasoning is deciding how to achieve these states of affairs.

4.2. Resource-bounded reasoning and calculative rationality

Calculative rationality is the ability to produce an optimal outcome only given the initial information (available when the agent started making the decision).

An agent cannot spend an indefinite amount of time on its reasoning as at some point it must stop and make a decision. There is a trade-off between how much time and computational resource should be invested in decision-making versus the quality of the decision making. Typically this means an agent cannot guarantee that the intentions and plans it commits to are optimal at the time at which it commits to them, an agent needs to be able to recognise when its intentions or plans are no longer appropriate and adapt if necessary.

4.3. Role of intentions

There are certain types of attitudes and behaviours we expect from the agent in respect to its intentions:

1. If an agent has an intention to ϕ, we expect the agent to devote some resources to ϕ otherwise we could argue it is not truly an intention.
2. Agents track the success of their intentions and are inclined to try again if their attempts fail, usually by trying to come up with an alternative plan.
3. Agents believe their intentions are possible, and should adjust them if they find out otherwise.
4. Agents recognise the possibility that they may fail to achieve their intentions.
5. An agent's intentions provide a 'filter' for adopting other intentions; agents cannot adopt two intentions which are mutuatlly exclusive.
6. An agent doesn't necessarily intend all the consequences of its intentions.

4.4. BDI agent architecture

BDI or **Beliefs, Desires, Intentions** is an agent architecture where agents maintain a (typically symbolic) representation of:

- Bel: set of all possible beliefs, while variable B represents agent's current beliefs
- Des: set of all possible desires, while variable D represents agent's current desires
- Int: set of all possible intentions, while variable I represents agent's current intentions

Agents are able to sense information from the environment, the input from the environment at any point is known as a percept.

- Per is the set of all possible percepts

When an agent receives a new percept, it updates its beliefs to take this new information into account using its belief revision function brf:

$$\text{brf} : 2^{\text{Bel}} \times \text{Per} \to 2^{\text{Bel}}$$

The belief revision function takes agent's current beliefs and the new percept as input and returns a new set of beliefs.

We model the two stages of deliberation using two functions:

- $\text{options} : 2^{\text{Bel}} \times 2^{\text{Int}} \to 2^{\text{Des}}$ — take current beliefs and intentions, returning set of desires
- $\text{filter} : 2^{\text{Bel}} \times 2^{\text{Des}} \times 2^{\text{Int}} \to 2^{\text{Int}}$ — filter for 'best' options to commit to as intentions given BDI

We model a planning stage using the function:

$$\text{plan} : 2^{\text{Bel}} \times 2^{\text{Int}} \to \Pi$$

Given the agent's current beliefs and intenions, we return a plan for the agent to follow to achieve its intentions.

Summary of the BDI agent control loop code iterations

- Produce a basic BDI feedback loop
- We are overcommitted to plan, let's update beliefs and check if plan is sound after each step
- We are overcommitted to intentions, let's deliberate after each step
- We are too slow, let's not deliberate if we think it's appropriate

These are shown on the next 4 pages.

4.4.1. BDI agent control loop version 1

$B \leftarrow B_0$
$I \leftarrow I_0$
while true **do**
  $p \leftarrow$ next percept
  $B \leftarrow \text{brf}(B, p)$
  $D \leftarrow \text{options}(B, I)$
  $I \leftarrow \text{filter}(B, D, I)$
  $\pi \leftarrow \text{plan}(B, I)$
  $\text{execute}(\pi)$
end

4.4.2. BDI agent control loop version 2

```
1  B ← B_0
2  I ← I_0
3  while true do
4  |  p ← next percept
5  |  B ← brf(B, p)
6  |  D ← options(B, I)
7  |  I ← filter(B, D, I)
8  |  π ← plan(B, I)
9  |  while not empty(pi) do
10 |  |  a ← head(π)
11 |  |  execute(a)
12 |  |  π ← tail(π)
13 |  |  p ← next percept
14 |  |  B ← brf(B, p)
15 |  |  if not sound(pi, I, B) then
16 |  |  |  π ← plan(B, I)
17 |  |  end
18 |  end
19 end
```

We modify version 1 to include a way for the agent to adjust its intentions or plan if conditions change, i.e. we want to avoid overcommitting to a plan.

We introduce a few new functions and predicates to allow for this:
- $\text{empty}(\pi)$ — true iff the plan is empty
- $\text{head}(\pi)$ — first action of plan π
- $\text{tail}(\pi)$ — remainder of plan π once the first action has been removed
- $\text{sound}(\pi, I, B)$ — true iff, according to beliefs B, agent expects the plan π to achieve intentions I

4.4.3. BDI agent control loop version 3

```
1  B ← B_0
2  I ← I_0
3  while true do
4  |  p ← next percept
5  |  B ← brf(B, p)
6  |  D ← options(B, I)
7  |  I ← filter(B, D, I)
8  |  π ← plan(B, I)
9  |  while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
10 |  |  a ← head(π)
11 |  |  execute(a)
12 |  |  π ← tail(π)
13 |  |  p ← next percept
14 |  |  B ← brf(B, p)
15 |  |  D ← options(B, I)
16 |  |  I ← filter(B, D, I)
17 |  |  if not sound(pi, I, B) then
18 |  |  |  π ← plan(B, I)
19 |  |  end
20 |  end
21 end
```

We want to avoid overcommitting to our intentions, so we adapt the control loop so the agent is able to adjust its intentions if it is appropriate to do so. If the agent believes that its intentions are either impossible or already acheived, then there is no point in persisting in trying to achieve those intentions.

We introduce two new predicates:

- succeeded(I, B) — true iff, based on beliefs B, agent believes it has achieved its intentions I
- impossible(I, B) — true iff, based on beliefs B, agent believes it is impossible to achieve intentions I

4.4.4. BDI agent control loop version 4

```
1  B ← B_0
2  I ← I_0
3  while true do
4      p ← next percept
5      B ← brf(B, p)
6      D ← options(B, I)
7      I ← filter(B, D, I)
8      π ← plan(B, I)
9      while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
10         a ← head(π)
11         execute(a)
12         π ← tail(π)
13         p ← next percept
14         B ← brf(B, p)
15         if reconsider(I, B) then
16             D ← options(B, I)
17             I ← filter(B, D, I)
18         end
19         if not sound(pi, I, B) then
20             π ← plan(B, I)
21         end
22     end
23 end
```

The previous version of the control loop requires that we stop and deliberate every time we take an action in the plan, however deliberation is a resoucre intesive process which is not ideal to do every time we execute.

While the agent deliberates, it is also possible that the environment change and any newly formed intentions are irrelevant. We must find a good trade-off between potentially continuing to achieve intentions even if they're not possible or there is no reason to achieve them, and spending time reconsidering whether we should achieve something.

We can however, introduce a new predicate, that is less expensive than deliberation that can determine whether we should deliberate:

- reconsider(I, B) — true iff, given beliefs B, the agent believes it is appropriate to reconsider I

> **Note**
>
> There are many approaches one could implement `brf`, `options`, and `filter` functions, and how an agent designer chooses to do this depends on the application. One could use an explicit utility function to decide between competing intentions, or take a machine learning approach to classify intentions according to how appropriate they are for the solution.

4.5. Means-ends reasoning strategies

Broadly speaking, there are two approaches to means-ends reasoning:

- Selecting an appropriate plan from a plan library
- Using a planning subsystem to generate an appropriate plan based on information the agent has about actions available to it

When working with a **plan library** approach, the agent has access to pre-compiled plans by the agent designer, with each plan containing information about:

- a goal: post-condition of the plan
- a context: pre-condition of the plan
- a body: 'recipe' of the plan, provides details of course of action for agent to follow

An agent find a plan whose goal matches with its intentions, and whose context matches with what the agent believes to be the current state of the world.

> **Note**
>
> Plans may themselves contain goals as well as actions providing agents with more flexibility!

When working with a **planning subsystem**, the agent can construct its own plan by providing the subsystem with information about:

- the goal/intention it wants to achieve
- what the agent believes to be the current state of the environment
- actions available to the agent (defined with pre-conditions & post-conditions)

5. Communication, coordination, and cooperation

This section looks at:
- *communication*: how do agents understand each other?
- *cooperation*: how can they divide large tasks between themselves?
- *coordination*: how do agents help and not hinder each other?
- *trust and reputation*: how do agents know when they can rely on others to do things?

5.1. Agent communication

Agents are autonomous so they have control over their own state and behaviour, but cannot force one another to perform any action, and an agent cannot write data into the internal state of another agent. However, agents can use **communicative actions** to try to influence other agents — we treat agent communication as a type of action — but also acknowledge they may fail to have intended effect.

In speech act theory, communicative actions are referred to as **speech acts**:
- *locutionary act*: act of making the utterance
- *illocutionary act*: the conveying of the speaker's intentions to the hearer
- *perlocution*: the effect achieved by the speech act

So when the speaker makes a locutionary act, they hope the hearer will correctly identify their intentions (the illocutionary act) and will act accordingly (the perlocution).

John Searle extended this theory to identify five classes of speech act:
- *representatives*: commit speaker to the truth of an expressed proposition
  if I say "it's raining" then my intention is to want you to know I believe it is raining
- *directives*: attempt to get the hearer to do something
  if I say "please make the tea" gives the intention that I want you to make the tea
- *commissives*: commit speaker to some course of action
  if I say "I promise to pay you back next week" gives the intention that I want you to believe that I will pay you back next week
- *expressives*: express some mental state
  if I say "thank you for making the tea" then I want you to believe that I am grateful to you for having made the tea
- *declarations*: bring about some changes in an institutional state of affairs
  things, in appropriate context, that directly affect state of the world in a specific way
  e.g. '[we the jury] find the defendant to be guilty', declaration of war, etc

These identify different illocutionary acts, but is not an exhaustive list, rather illustates that we can classify speech acts according to intention they are trying to convey.

Humans are good at inferring intentions behind utterances by taking the context of the utterance and knowledge about the speaker into account. Computationally this is difficult, ACLs use concepts from the speech act theory to reduce the complexity of the problem. In ACLs, the illocutionary force (intention) is made explicit.

We think of speech acts as having two components:
- *performative*: illocutionary force that is being conveyed, e.g. request, inform, promise
- *propositional content*: current state of the world, e.g. the door is closed

If agents have a shared understanding of meaning of different performatives, then the speaker can be confident that the hearer will correctly identify the speaker's intentions, and hearer can be confident they have correctly identified intentions.

Autonomous heterogeneous agents in a multi-agent system require a shared **agent communication language** (ACL) so they can communicate with each other.

For an ACL to be effective, agents involved in the communication need a shared understanding of what is meant by different performatives, this is called the **semantics** of the ACL.

5.1.1. KQML ACL

The **Knowledge Query Manipulation Language** (KQML) compromises of two parts:

- The KQML that defines performatives, e.g. ask-if ('is it true that …'), perform ('please perform action …'), tell ('it is true that …').
- The **knowledge interchange format** (KIF) for expressing message content.
  This language uses predicates to express facts and relationships.
  e.g. $(> (* \text{(width chip 1)}\text{(length chip 1)})(* \text{(width chip2)}\text{(length chip2)}))$

KIF expressions are wrapped in KQML statements, together with some other message parameters (sender, receiver, …) to form a full message to be communicated.

For example:

```kqml
(tell
  :sender A
  :receiver B
  :content (>
    (* (width chip1) (length chip1))
    (* (width chip2) (length chip2))
  )
)
```

Semantics of KQML are only informally defined, open to different interpretations, for example:

$$\text{tell} : S \text{ claims to } R \text{ that } C \text{ is in knowledge base belonging to } S$$

where S is sender, R is receiver, C is content of message

5.1.2. Plan-based theory of speech acts

One approach to defining ACL semantics is to treat them in the same way we treat actions in AI planning, that is, define performatives in terms of preconditions and postconditions. An agent can then reason about which speech act to perform in the same way as it reasons about normal (non-communicative) actions.

Example semantics for request performative

This is the approach taken in *Cohen and Parrault's plan-based theory of speech acts*. Given a speaker S, hearer H, and content α, we can define the semantics of request as follows:

request(S, H, α)
- preconditions:
  - (S believe (H can-do α)) $\wedge$
  - (S believe (H believe (H can-do α))) $\wedge$
  - (S believe (S want α))
- postconditions:
  - (H believe (S believe (S want α)))

The postcondition does not say whether H performs the action, but instead that S believes H believes S wants α performed.

5.1.3. FIPA ACL

The **FIPA ACL** explicitly recognises postconditions of a speech act cannot be enforced by the speaker, so it represents its semantics in terms of feasibility preconditions (true in order for the sender to send the message) and the rational effect (what the sender is attempting to achieve).

Example of FIPA semantics

For example, FIPA semantics of a request message $\langle s, \text{request}(r, \alpha)\rangle$ where s is the sender, r is the receiver, and α is the content:

feasibility precondition:
| $B_s \text{ Agent } (\alpha, r) \land \neg B_s I_r \text{ Done}(\alpha)$
rational effect:
| $\text{Done}(\alpha)$

Where:
- $B_i\phi$ means agent i believes ϕ
- $\text{Agent}(\alpha, i)$ means α is an action of agent i (agent i can perform α)
- $I_i\phi$ means agent i has intention to make ϕ true
- $\text{Done}(\alpha)$ means action α has been performed

So an agent sending $\langle s, \text{request}(r, \alpha)\rangle$ message will conform to FIPA ACL semantics if:
- sender s believes receiver r can perform action α
- sender s doesn't believe receiver r already intends that α is performed (otherwise no point requesting)

Note that there is no explicit recognition sender of message can enforce any postconditions. Instead, the receiver of a message understands what the sender wishes to achieve, the rational effect.

Although, if we see an agent send a message according to some semantics, we cannot assume the feasibility preconditions are necessarily true, as they are given in terms of the sender's mental state which may not truly represent the world state.

In effect, we cannot be sure agents are conforming to semantics of the FIPA ACL.

Since we don't know if a particular agent conforms to FIPA ACL semantics, we could take an approach of defining ACL semantics based on **social commitments**, i.e. take into account social, objective consequences of performing a communicative act and express the meaning of a communicative act in terms of commitments directed from one agent to another.

For example, if I inform you p, then I should not inform someone else $\neg p$ unless I also inform you $\neg p$. If I promise you t by X time, then I am expected to do t by X time. We can check whether these expectations are violated or not!

5.1.4. Ontologies

An **ontology** defines the relationships between different terms in the language.

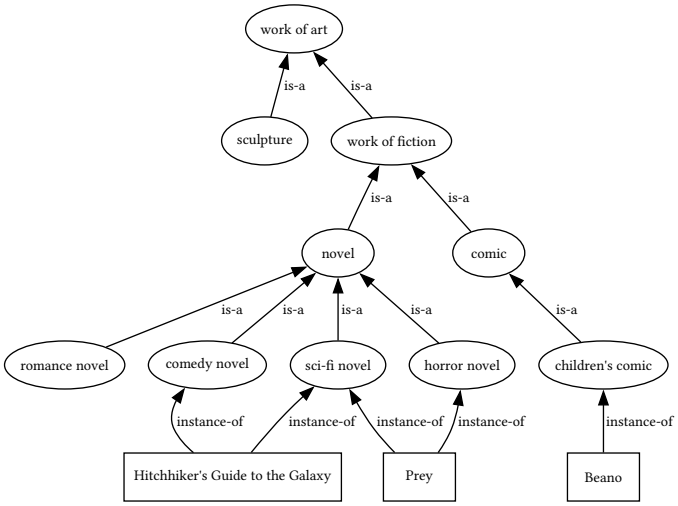


Figure 1: An ontology

Agents sharing this sample ontology, all know that:

- Prey and Beano are both works of fiction.
- Beano is a children's comic, a type of comic.
- Prey is both a science fiction and horror novel.

If an agent hears a term for the first time and it doesn't know what it is, it can query the agent who sent the message containing the term to find out where in their shared ontology that term should be situated. e.g. mentioning Hitchhiker's Guide to the Galaxy, one can explain it is a science fiction novel

Using ontologies poses a few interesting challenges:

- how to reason efficiently
- how to check they are consistent
- how to resolve inconsistencies between ontologies
- how to classify new instances
- which formal representation to use

5.2. Agent cooperation

Cooperative Distributed Problem Solving (CDPS) is the area of study of how agents manage thier cooperation, it involves:

1. *problem decomposition*: dividing problem into smaller tasks for distribution
2. *sub-problem allocation*: determining which sub-problems to allocate to which agents
3. *sub-problem solution*: where the agents solve their allocations
4. *sub-problem synthesis*: where solutions are integrated into overall solution

A typical CDPS domain contains agents in a loosely-coupled network capable of sophisticated problem-solving and independent work, but no single agent has sufficient expertise, resources, or information to solve a given problem.

Some challenges are posed by CDPS:

- what level of granularity should a problem be decomposed to
- does one agent have the expertise to decompose the problem
- how to decide how to allocate (e.g. Contract Net Protocol)
- how to reach agreement when agent is not inclined to take on a problem
- what coordination is required to solve sub-problems

5.2.1. Contract Net Protocol

The **Contract Net protocol** addresses how to allocate sub-problems to agents.
It is composed of five stages:

- *recognition*: agent recognises it has a task it cannot achieve, or would prefer not to
- *announcement*: agent broadcasts to multi-agent system an announcement of the task specification to be achieved, including:
  - description of the task itself (may be executable)
  - any constraints (task completion deadline, quality constraints, eligibility criteria)
  - meta-task information (bid submission deadline, information the bids should provide)
- *bidding*: agents decide whether they are eligible, capable, and willing to bid for the task, and return a bid if so, specifying any information requested in the announcement
- *awarding*: agent awards the contract to and communicates with bidding agents
- *expediting*: successful contractor expedites the task, can involve generating further manager-contractor relationships

Each agent can at any time for any task, be a manager or contractor or both.

Example of vacuum bids

Suppose an agent is overall in charge of cleaning task, they put out an announcement that it is looking for bids from agents to do vacuuming, requires it is completed within 24 hours and it clears at least 90% of the dirt away.

- Agents capable, bid for the task
- Agent wants bids submitted within next half hour, and they should contain an ETA, and estimated % cleared away
- Two agents reply:
  - Robot1, 99% within 20 hours
  - Robot2, 90% within 2 hours
- Agent in charge now needs to pick a robot, if it cares more about the dirt then it picks Robot1, Robot1 now needs to complete the task and report back the results.

5.3. Agent coordination

Coordination is about managing inter-dependencies between actions of agents, ideally avoiding negative relationships and prioritising positive relationships.

Negative relationships

- *resource induced*: insufficient resources to allow both of our actions to occur simultaneously
  e.g. two people cannot go through a narrow doorway
- *incompatibility*: actions require mutually exclusive states
  e.g. you need to sit an exam, phone needs to be off. partner is booking a holiday, requires phone.

Positive relationships

- *requested*: agent explicitly asks for help with an action
  e.g. via Contract Net protocol
- *action equality*: agents recongise they are both planning to perform the same action, so one can perform it and save the other the effort
- *consequence*: action that one agent is planing has a side effect of achieving another agent's goals
- *favour*: similar to consequence; action that one agent is planning has a side effect of contributing to another agent's goals

One approach agents can use is **multi-agent planning** where agents jointly construct plans so their actions are coordinated, aiming to avoid negative inter-dependencies and to benefit from any positive inter-dependencies that may exist.

Approaches to multi-agent planning

- *centralised planning for distributed agents*: coordinating agent has all information about each individual's private goals and available actions; coordinator uses this information to find a global plan
- *partial global planning*: agents plan for themselves but create a (partial) view of global plan of all agents, which can be used to improve their own plans; they can share revised partial view of the global plan to help them improve, and vice-versa
- *centralised merging of individual plans*: agents plan for themselves then coordinate local plan with coordinator; coordinator then must merge plans to create an efficient global plan

Often however, agents do not want to share private information with other agents. Agents could instead have expectations of behaviour of other agents, **normative multi-agent systems**, because of knowledge they have about the system they are working in because we can expect certain behaviours to be normal (conventional way to act) or because it is normative (expected standard set by system).

A **norm** is an established, expected pattern of behaviour.
A normative multi-agent system contains a set of norms known by agents.

They can either be:
- *prescriptive*: standards set by the system, normative
  In the UK, you must drive on the left.
- *conventions*: expected normal behaviours
  If it is your friend's birthday, you should buy them a present.

Agents can still violate norms, but there may be consequences for this. Though in general, this provides behavioural constraints that can simplify reasoning by dictating courses of action that should be followed.

Prescriptive norms are formally represented as:

$$\langle \text{target}, \text{deontic-component}, \text{context}, \text{content}, \text{punishment} \rangle$$

Consisting of:
- the agents who the norm applies to
- the type of norm: obligation, prohibition, or permission
- when the norm applies
- the behaviour constrained by the norm
- what happens if the agent gets caught violating the norm

An **obligation norm** says that in a given context, an agent of the target type is obliged to perform the content behaviour, or face a punishment.
e.g. $\langle \text{all-drivers}, \text{obligation}, \text{at-red-light}, \text{stop}, \text{3-points-on-license} \rangle$

A **prohibition norm** says that in a given context, an agent of the target type is prohibited from performing the content behaviour, otherwise they will face a punishment.
e.g. $\langle \text{all-drivers}, \text{prohibited}, \text{on-public-UK-road}, \text{drive-on-right}, \text{6-points-on-license} \rangle$

A **permission norm** says that an agent is permitted (rather than must/must not do) to do the content behaviour in the given context, hence no punishment is associated.
e.g. $\langle \text{all-drivers}, \text{permitted}, \text{on-one-way-UK-street}, \text{drive-on-right}, \_ \rangle$
... $\langle \text{ambulance-drivers}, \text{permitted}, \text{at-red-light AND on-way-to-emergency}, \text{go}, \_ \rangle$

An agent's reasoning about what to do is affected by the norms in the system:
- Caught violating a prescriptive norm, it is subject to punishment, decrease in utility.
- Agent violating convention may cause indirect decrease in utility by hindering agent's coordination.

Sometimes agents may have reason to disobey norms, let's suppose the situation:

$$\langle \text{all-drivers}, \text{prohibited}, \text{on-UK-highway}, \text{drive-over-70-mph}, \text{fine-£50} \rangle$$

A couple reasons immediately come to mind:
- *ignorance*: agent is unaware of norm
- *lack of understanding*: agent doesn't know what the norm means
- *punishment is too low*: agent has little incentive to obey the norm
- *probability of being caught is too low*: agent may be willing to risk punishment
- *another goal may be more important*: if obeying the norm means the agent cannot achieve an important goal, loss in utility from not achieving this goal may be greater than any punishment
- *there may be no choice*: agent might not have capability to comply with the norm

Challenges involved with designing appropriate prescriptive norms
- *agents need to know the norms*: must be an efficient means of communicating them to all agents in the system
- *agents need to understand the norms*: agents need a shared understanding of norms and terms used to define them
- *monitoring of agents' adherence to norms needs to be effective*: balance needs to be found between amount of effort involved in monitoring and likelihood that any agent disobeying a norm will be caught
- *punishments need to be suitable*: punishments need to dis-incentivise agents from disobeying the norms, but not so great that they would curtail agents' autonomy

Another key challenge presents itself in how to design a suitable set of prescriptive norms that will **encourage types of behaviour desired from a system**.

When prescriptive norms are designed, conventions emerge from behaviours of agents in the system. Over time, agents tend to copy others' behaviour. e.g. you'd likely adopt the position people stand on in an escalator to be better coordinated with others.

5.4. Trust and reputation

During task allocation, an agent may want to consider reliability of agents when determining who they'd want to work with, e.g. some agents may provide same quality of work but deliver very late.

There are two primary ways to determining how reliable another agent is:

- *trust*: an agent can determine its measure of trust according to past experience with an agent; for example an agent that consistently delivers late will have a lower trust rating than one that delivers the same quality but on time
- *reputation*: a 'trust score' derived from experience of other agents; do other agents report a positive or negative experience with one another? *do we even trust the experience of others?*

The **FIRE model** is a model of trust and reputation that agents can use to help them decide where to delegate tasks to, it uses four sources of information:

- *direct experience*: evaluator uses knowledge it has of previous interactions with target agent; trust derived from experience is called **Interaction Trust**
- *witness information*: evaluator uses information about other agents' experiences of interacting with target (other agents are called witnesses); trust derived from witness information is called **Witness Reputation**
- *role-based rules*: evaulator may have certain expectations about target because of role that the target holds; if evaluator and target are part of the same organisation, they may be inclined to trust them; trust derived from role-based rules is called **Role-based Trust**
- *third-party references provided by target agent*: similar to witnesses, except target itself collects references and can provide them; trust derived from third-party references is called **Certified Reputation**

An evaluator's trust in a target can be assessed separately along different dimensions, you may trust a friend to give you good advice, but don't trust them to turn up on time — different dimensions are referred to as **quality-of-service terms**.

For an evaluator A_{ge}, a target agent A_{gt}, and a term c, each information source component provides a value between -1 and 1 which represents the trust the evaluator has in the target with respect to c (according to that information source). Scores are combined to generate an overall trust value.

Calculating Interaction Trust for the FIRE model Interaction Trust is based on evaluator's past interactions with the target. So for each interaction, evaluator provides a rating for each quality-of-service term that it is interested in ($[-1, 1]$), which produces records:

$$(a, b, c, i, v)$$

Where a is evaluator agent, b is the agent it interacted with, c is term on which rating is based on, i is the interaction, and v is the rating of that interaction.

To determine trust the evaluator has in the target, with respect to a particular term, we use the following equation to determine a weighted average of ratings of interactions with the target:

$$T(a, b, c) = \frac{\sum_{r_i \in \mathcal{R}(a,b,c)} \omega(r_i) v(r_i)}{\sum_{r_i \in \mathcal{R}(a,b,c)} \omega(r_i)}$$

- where $\mathcal{R}(a, b, c)$ is the set of all records that a holds about interactions with b w.r.t c
- where $\omega(r_i)$ is the weight associated with record r_i
- where $v(r_i)$ is the rating value associated with record r_i

The equation presented associates a weight with each record entry, we would like the weight to reflect the likely relevance it has to judging the target agent for a new interaction at the current time.

One approach is to take into account recency, so more recent interactions have higher weight than older ones. For example, rating value on day X may be weighted at $\left(\frac{1}{k}\right)^{\text{th}}$ of the rating values from an interaction on day $X + 1$.

Example interaction score

Suppose an evaluator has delegated several tasks to a target in the past, the evaluator cares about quality fo a solution, q, and speed at which it produced a solution s, so it may have the following records:

$$\left(A_{ge}, A_{gt}, q, i_1, 0.3\right)$$
$$\left(A_{ge}, A_{gt}, s, i_1, -0.1\right)$$
$$\left(A_{ge}, A_{gt}, q, i_2, -0.2\right)$$
$$\left(A_{ge}, A_{gt}, s, i_2, 0.7\right)$$
$$\left(A_{ge}, A_{gt}, q, i_3, 0.8\right)$$
$$\left(A_{ge}, A_{gt}, s, i_3, 0.9\right)$$
$$\left(A_{ge}, A_{gt}, q, i_4, 0.85\right)$$
$$\left(A_{ge}, A_{gt}, s, i_4, 0.92\right)$$

6. Game Theory (and negotation)

The outcome of a negotation (or outcome/vote/argumentation dialogue) depends on the actions of each agent involved, and we can interpret different agreement mechanisms as a game between the games.

Typically we have a **protocol** (rules of the game) that define:

- *moves*: choices agents can make
- *permitted moves*: when agents are allowed to make certain moves
- *termination*: when the interaction ends
- *outcome*: what the outcome is given the moves taken

Each agent decides which of the permitted moves to make with the aim of getting an outcome that they prefer, this is the agent's **strategy**.

Game Theory is the study of mathematical models of strategic interactions. Given two or more agents in an encounter, we can model each agent's preferences on the outcome of the encounter, and model the actions hence outcome using game theory.

We can construct a **play-off matrix** to model the possible actions and outcomes by different agents.

Example: Golden Balls

Classic British TV game show example, two contestants have either the option to split or steal the final amount, if they both steal, they lose the money completely.

		Player 2: Split	Player 2: Steal
Player 1	Split	5k, 5k	0, 10k
	Steal	10k, 0	0, 0

Figure 2: Pay-off matrix for Golden Balls game

A strategy s_i is **dominant** for an agent i if, no matter what strategies the other agents choose, agent i will do at least as well by playing s_i as it would playing any other strategy.

Example: Dominant strategy in Golden Balls

- Assuming player 2 splits, player 1 should opt to steal.
- Assuming player 2 steals, either option is fine.

So the dominant strategy is to always steal. However, if they both behave rationally — play dominant strategy, then they will both lose everything.

Nash equilibrium occurs for a set of strategies when no participant can benefit from changing its strategy assuming the strategies of the others remain unchanged. It is a so-called 'stable outcome', where no agent has any incentive to deviate from the choice that they made.

Example: Nash equilibria in Golden Balls

Assuming both players steal, we are in a Nash equilibrium as neither player can be better off by splitting instead of stealing. This is not the case if they both split, as both can improve!

Example: Nash equilibria in Golden Balls (cont.)

If player 2 steals and player 1 splits, we also reach an equilibrium:
- If player 2 steals, player 1 can do no better than to split.
- If player 1 splits, player 2 can do no better than to steal.

If player 1 steals and player 2 splits, we also reach an equilibrium:
- If player 1 steals, player 2 can do no better than to split.
- If player 2 splits, player 1 can do no better than to steal.

An outcome is **Pareto optimal** if it is the case that in every other outcome where an agent is better off, at least one agent is worse off. There are no other outcomes where an agent can be better off without making others worse off.

Example: Pareto optimality in Golden Balls

For the situation where both players split:
- Either player can choose to steal instead which:
  - Leaves them better off however at 10k (condition 1)
  - The other agent will be worse off at 0 (condition 2)
- All other outcomes are not applicable (both stealing does not improve)
- ∴ Pareto optimal

For the situation where player 1 splits and player 2 steals:
- If player 2 splits instead:
  - Leaves player 1 better off at 5k (condition 1)
  - Player 2 will be worse off at 5k (condition 2)
- If player 1 steals and player 2 splits:
  - Leaves player 1 better off at 10k (condition 1)
  - Player 2 will be worse off at 0 (condition 2)
- All other outcomes are not applicable (player 2 cannot improve)
- ∴ Pareto optimal
- (and vice-versa player 1 stealing and player 2 splitting)

For the situation where they both steal:
- Both players could choose to split instead:
  - Leaves both players better off at 5k (condition 1)
  - Does not make anyone worse off
- If either player chooses to split:
  - Leaves the other player better off at 10k (condition 1)
  - Does not make anyone worse off
- ∴ not Pareto optimal

An outcome maximises **social welfare** if it maximises the sum of utility gained by all the agents. This may be desirable for a system, but may not lead to a good pay-off for an individual agent.

Example: Social welfare in Golden Balls

All options except for where both steal maximise social welfare as the sum of the reward is 10k in the other outcomes.

6.1. Negotation

The aim of **negotiation** is to each an agreement (**deal**) that all participants can agree to, even if the participants have conflicting goals and preferences.

The **negotiation set** is the set of all possible proposals that the agent can make.

How does agent negotiation work?
There is a protocol that agents follow (which defines the moves that can be made, when it is permissible to make a move, when the interaction terminates and what the outcome is).
Each agent uses its own private strategy to determine which permissible moves to make.

6.1.1. Alternating Offers Protocol (for resource devision)

The **alternating offers protocol** is a one-to-one (involving two agents A_{g1}, A_{g2}) protocol for resource division. Proposals take the form (x_1, x_2) where x_1 is the amount of resource allocated to A_{g1} and x_2 is the amount of resource allocated to A_{g2}.

The protocol works as follows:

1. Proposing agent begins by making proposal p from negotiation set
2. Evaluating agent can either accept or reject
   - If accepted, p is implemented
   - Otherwise, agents swap roles and another round starts

If agreement is not reached then we say that the result is the **conflict deal**. It is assumed that the conflict deal is the worst outcome (i.e. each agent prefers every other proposal from the negotiation set to the conflict deal, the agents want to reach some agreement).

The **negotiation set** is the set of all Pareto optimal proposals. We don't consider proposals which are not Pareto optimal as this is not efficient, and we will need to do further negotiation.

Example: Pie Divsion

Let's suppose a resource, a pie, is being divided.

- Proposal $(1, 0)$ says to give A_{g1} all of the pie. (Pareto optimal)
- Proposal $(0.25, 0.25)$ says to give everyone a quarter of the pie. (**not** Pareto optimal)

Therefore, the negotiation set for divison of a pie is $\{(x, 1-x) \mid 0 \leq x \leq 1\}$.

Some exploration of behaviours this produces:

> **Note**
>
> Assume a special case where only a single round is allowed, *the ultimatum game*.
> - Both agents agree to whatever A_{g1} proposes or take the conflict deal.
> - Since A_{g1} has the power here and all agents prefer any outcome to the conflict deal, the best thing that A_{g2} can do is to accept the terms, even if the proposal is $(1, 0)$.
> - The strategies "A_{g1} proposes it gets all the pie" and "A_{g2} accepts the deal" are in Nash equilibrium, neither agent can do any better (assuming the other agent's strategy doesn't change).

> **Note**
>
> Assume we have a case where there are two rounds: the second agent can reject the first proposal and then propose their own in the second round which the first agent can do no better than to accept.
>
> This generally applies for any fixed rounds n as the agents can continue to reject deals until a single round is left, leaving the last agent to propose to get all the pie.

Example: Pie Divsion (cont.)

> **Note**
>
> Let's assume the general case (indefinite rounds), A_{g1} proposes first and that A_{g1} uses the private strategy:
>
> $$A_{g1} \text{ strategy : always propose } (1,0) \text{ and always reject any offer from } A_{g2}$$
>
> In the case that:
> - A_{g2} never accepts: they will end up in conflict deal
> - A_{g2} prefers all possible deals to conflict deal so it makes sense for it to accept $(1,0)$ in the first round
>
> However, this requires analysis of A_{g1}'s strategy.
> A_{g1} could convince A_{g2} by sharing its strategy.

6.1.1.1. Time-based Alternating Offers Protocol

We can adapt the protocol to account for the fact that agents may prefer to reach an outcome sooner than later.

$$\text{Agents prefer an outcome } x \text{ at time } t_1 \text{ to the outcome } x \text{ at time } t_2 \text{ if } t_2 > t_1$$

So if there is a choice between an agreement on x being now or later, the agents will always prefer that the agreement to x be made now.

Example: Pie Divsion (with time)

We can model this using a discount factor δ_i for each agent A_{gi}, where $0 \leq \delta_i < 1$.

Consider that A_{gi} has been offered x of the pie and that each timepoint represents a round:
- value of slice x to A_{gi} at time 0 is $(\delta_i)^0 x = x$
- value of slice x to A_{gi} at time 1 is $(\delta_i)^1 x = \delta_i x$
- value of slice x to A_{gi} at time 2 is $(\delta_i)^2 x = \delta_i \delta_i x$
- ...
- value of slice x to A_{gi} at time k is $(\delta_i)^k x$

The closer δ_i is to 1, the more patient the agent is (less of an impact the discount factor has on the value of the outcome). Conversely, if $\delta_i = 0$ then unless agreement is reached in the first round, then outcome will have no value to A_{gi}.

> **Note**
>
> For the case where only one round is allowed, we find the same situation as the non-discounted version. We have an ultimatum game and the first agent can propose $(1,0)$ and the other agent can do no better than accept the proposal. (this is in Nash equilibrium)

Example: Pie Divsion (cont. with time)

Note

Let's suppose we can have two rounds only. A_{g2} could reject whatever proposal A_{g1} makes and propose $(0, 1)$, the first agent can do no better than to accept the offer. But because of the discount factor, the whole of the pie will only be worth δ_2 to A_{g2}.

So, the agent can take this into account when making the initial offer, and instead offer $(1 - \delta_2, \delta_2)$. In this case, A_{g2} can do no better than to accept this offer since if the negotiation moves to the second round and A_{g2} ends up with all of the pie, this will only be worth δ_2 to A_{g2}. (this is in Nash equilibrium)

Note

Now suppose the general case, where rounds are indefinite.

We find that there is a Nash equilibrium: (proof is beyond scope)

$$A_{g1} \text{ makes offer } \left(\frac{1 - \delta_2}{1 - \delta_1 \delta_2}, \frac{\delta_2(1 - \delta_1)}{1 - \delta_1 \delta_2} \right)$$

$$A_{g2} \text{ accepts}$$

The more patient an agent is (higher value of δ_i), the better outcome it will get.

This analysis depends on the agents knowing one another's strategies and their discount factors. In this case if the first agent knows the second agent's discount factor, then it can follow the strategy to always propose $\left(\frac{1-\delta_2}{1-\delta_1\delta_2}, \frac{\delta_2(1-\delta_1)}{1-\delta_1\delta_2} \right)$ and reject any other proposals. And if the second agent knows the first agent is going to do this, then it can do no better than accept in the first round.

Though typically, we do not have information about strategy or discount factors.

6.1.2. Task Oriented Domain

In a task oriented domain there are some set of tasks, each of which is allocated to one of the two agents. The point of the negotiation is to reach some new distribution of the tasks to the two agents.

Example Provided Problem

"Imagine that you have three children, each of whom needs to be delivered to a different school each morning. Your neighbour has four children, and also needs to take them to school. Delivery of each child can be modelled as an indivisible task. You and your neighbour can discuss the situation, and come to an agreement that it is better for both of you (for example, by carrying the other's child to a shared destination, saving him the trip). There is no concern about being able to achieve your task by yourself. The worst that can happen is that you and your neighbour won't come to an agreement about setting up a car pool, in which case you are no worse off than if you were alone. You can only benefit (or do no worse) from your neighbour's tasks. Assume, though, that one of your children and one of your neighbour's children both go to the same school (that is, the cost of carrying out these two deliveries, or two tasks, is the same as the cost of carrying out one of them). It obviously makes sense for both children to be taken together, and only you or your neighbour I will need to make the trip to carry out both tasks."

A **task oriented domain** is a triple $\langle T, A_g, c \rangle$ where:
- T is the finite set of all possible tasks
- $A_g = \{A_{g1}, A_{g2}\}$ is the set of participating agents
- $c : 2^T \to \mathbb{R}^+$ is a function that defines the cost of executing each subset of tasks

The cost function c has to satisfy two constraints:
- $c(\emptyset) = 0$ — cost of doing nothing is nothing
- if $T_x \subseteq T_y \subseteq T$, then $c(T_x) \leq c(T_y)$ — adding tasks never decreases cost

An **encounter** is the initial allocation of tasks to the agents (before any negotiation takes place) in the form:

$$\langle T_1, T_2 \rangle$$

Where $T_1 \subseteq T$ is the set of tasks initially allocated to A_{g1} and $T_2 \subseteq T$ is the set of tasks initially allocated to A_{g2}, and $T = T_1 \cup T_2$.

A **deal** is like an encounter, an allocation of tasks in T to the two agents in the form:

$$\langle D_1, D_2 \rangle$$

Where $D_1 \subseteq T$ is the set of tasks allocated to A_{g1} and $D_2 \subseteq T$ is the set of tasks allocated to A_{g2}, and $T = D_1 \cup D_2$.

We denote the **cost to agent** A_{gi} of a deal $\delta = \langle D_1, D_2 \rangle$ as

$$\text{cost}_{i(\delta)} = c(D_i)$$

i.e. the cost of a deal to an agent is the cost of executing the tasks allocated

The **utility** to agent A_{gi} of a deal δ is the difference between the cost of the tasks it was initially allocated and the cost of the set of tasks it is allocated in the deal:

$$\text{utility}_{i(\delta)} = c(T_i) - \text{cost}_{i(\delta)}$$

It represents how much the agent has to gain from the deal.
If it is negative, then the agent would be better off performing original tasks.

Example in practice

Suppose the scenario:
- Encounter: $\langle \{\text{shop}, \text{cook}, \text{washUp}\}, \{\text{getVideo}, \text{tidy}, \text{buyWine}\} \rangle$
  where $c(\{\text{shop}, \text{cook}, \text{washUp}\}) = 12$
  where $c(\{\text{getVideo}, \text{tidy}, \text{buyWine}\}) = 10$
- Possible deal: $\delta = \langle \{\text{cook}, \text{tidy}, \text{washUp}\}, \{\text{shop}, \text{getVideo}, \text{buyWine}\} \rangle$
  where $c(\{\text{cook}, \text{tidy}, \text{washUp}\}) = 9$
  where $c(\{\text{shop}, \text{getVideo}, \text{buyWine}\}) = 6$

The utilities are therefore:
- $\text{utility}_1(\delta) = 3$
- $\text{utility}_2(\delta) = 4$

If agents don't reach an agreement in a task oriented domain, then they simply end up with the initial encounter. So the conflict deal is the deal $\langle T_1, T_2 \rangle$. Thus the utility of the conflict deal is 0 for each agent.

6.1.3. Monotonic Concession Protocol

A deal δ is said to be **individual rational** if neither agents prefers the conflict deal to δ.

We define the negotiation set for task redistribution as the set of deals that are:
- individual rational; no point proposing a deal less preferable to conflict deal
- Pareto optimal; inefficient to make proposal if there is an alternative that makes someone better off and neither worse off

The **monotonic concession protocol** works by:
- Negotiating proceeding in rounds.
- In each round, agents simultaneously propose a deal from the negotiation set.
- In first round, agents are free to make any proposal.
- In subsequent rounds r, each agent A_{gi} has two options.
  Given δ_i is the deal A_{gi} proposed in previous round $r-1$, then A_{gi} can either:
  - Make a concession and propose a new deal δ_i^i that the other agent A_{gj} finds more preferable than δ_i
    (has to be the case that $\text{utility}_{j(\delta_i')}$ is greater than $\text{utility}_{j(\delta_i)}$)
  - Refuse to make a concession and propose same deal again.

An agreement is reached if one agent finds the deal proposed by the other is at least as good or better than its own proposal.
Formally, agreement reached if $\text{utility}_1(\delta_2) \geq \text{utility}_1(\delta_1)$ or $\text{utility}_2(\delta_1) \geq \text{utility}_2(\delta_2)$.

These cases are possible after a round:
- Each agent's offer matches or exceeds that of the other agent, one of the proposals is randomly selected as agreement deal.
- Only one offer matches or exceeds, it is chosen as agreement deal.
- No agreement is reached, negotiation proceeds to another round of simultaneous proposals.
- If neither agent makes a concession in some round $r > 0$, then negotiation terminates with conflict deal.

6.1.3.1. Zeuthen strategy

The **Zeuthen strategy** dictates what an agent should do when negotiating in a task oriented domain using monotonic concession protocol by asking three questions as follows:

1. What should an agent's first proposal be? — its most preferred deal
2. On any given round, which agent should concede (make a concession)? — agent least willing to risk conflict deal (if both are equal, do it at random)
3. If an agent makes a concession, how much should it concede by? — just enough to change the balance of risk (so it is no longer the case that the agent is the least willing to risk conflict deal)

Assuming δ_i is the current deal proposed by A_{gi}, and δ_j for A_{gj}.
We can formally define A_{gi}'s willingness to risk the conflict deal as:

$$\text{Risk}(i,t) = \begin{cases} 1 & \text{utility}_i(\delta_i) = 0 \\ \frac{\text{utility}_i(\delta_i) - \text{utility}_i(\delta_j)}{\text{utility}_i(\delta_i)} & \text{otherwise} \end{cases}$$

Higher value, more willing A_{gi} is to risk conflict deal as:

- Closer value to 1 means less difference between:
  - Amount of utility the agent A_{gi} would lose by taking other agent's proposal rather than the currently proposed deal by A_{gi}
  - Amount of utility the agent A_{gi} would lose by ending up with conflict deal rather than the currently proposed deal by A_{gi}
- The more similar these are, the less A_{gi} has to lose by risking conflict deal in comparison to other agent's current proposal

An implementation is provided in `jupyter/zeuthen-strategy.ipynb`!

6.1.3.1.1. Deception

The strategy assumes agents' knowledge of each other's utility function and range of possible deals. Agents could attempt deception to benefit themselves:

- **Phantom/decoy tasks**: agent lies about initial encounter to claim initial encounter contains more tasks than it truthfully does
- **Hidden tasks**: agent lies about initial encounter to claim initial allocation contains less tasks than it truthfully does

7. Making group decisions

This section concerns social choice theory, which is used to allow multiple agents to make group decisions.

Example of computational applications of social choice theory

- internet search: aggregation of output of several search engines that rank by relevance (preference aggregation); if we consider a link from one page to another as a vote of endorsement, we have to figure out importance of a page — same concept as voting
  Google's PageRank algorithm uses this idea and has been analysed in social choice theory terms.
- decision support systems: any system that supports groups of users, such as members of a local community or shareholders, to make collective decisions
- social networks, e-commerce, e-governance: groups of users need to make decisions together; social choice theory contributes to foundations for how such processes are setup

Vote example

Five students each have a personal assistant agent, tasked with setting up a meeting for them. Preferences are listed as below:

$$\text{Mon} \underset{1}{\succ} \text{Tue} \underset{1}{\succ} \text{Wed}$$
$$\text{Mon} \underset{2}{\succ} \text{Tue} \underset{2}{\succ} \text{Wed}$$
$$\text{Tue} \underset{3}{\succ} \text{Wed} \underset{3}{\succ} \text{Mon}$$
$$\text{Wed} \underset{4}{\succ} \text{Mon} \underset{4}{\succ} \text{Tue}$$
$$\text{Wed} \underset{5}{\succ} \text{Tue} \underset{5}{\succ} \text{Mon}$$

We want to pick the option that best meets the students' collective preferences, however:

- Monday would make 1 and 2 happy, but majority have different preference.
  3 and 5 will be particularly unhappy.
- Tuesday would make 3 happy, but majority have different preference.
  4 will be particularly unhappy.
- Wednesday would make 4 and 5 happy, but majority have different preference.
  1 and 2 will be particularly unhappy.

This example demonstrates that voting protocols often need careful thought about desirable properties, in this case the majority will always prefer a different outcome so we have to account for that.

Condorcet's paradox: there are situations in which, no matter which outcome we choose, a majority of votes will prefer another candidate.

7.1. Voting Protocols

7.1.1. Plurality voting protocol

Plurality is a voting protocol under which:

- Each voter submits their preferences
- Each candidate gets one point for every preference order that ranks them first
- Winner is the candidate with the most points

Example vote

Let $\Omega = \{\omega_1, \omega_2, \omega_3\}$:

- 40 agents have preference $\omega_1 \succ \omega_2 \succ \omega_3$
- 30 agents have preference $\omega_2 \succ \omega_3 \succ \omega_1$
- 30 agents have preference $\omega_3 \succ \omega_2 \succ \omega_1$

Under the plurality protocol, this gives:

- 40 points to ω_1
- 30 points each to ω_2 and ω_3

The winner is ω_1.
... however, 60 of the agents have ω_1 as their least preferred option!

7.1.2. Condorcet winner

The **Concorcet winner** is an outcome that beats every other outcome in pairwise majority conests.

A voting protocol satisfies the **Condorcet principle** iff it selects the Condorcet winner (whenever one exists).

Example vote

Let $\Omega = \{\omega_1, \omega_2, \omega_3\}$:

- 40 agents have preference $\omega_1 \succ \omega_2 \succ \omega_3$
- 30 agents have preference $\omega_2 \succ \omega_3 \succ \omega_1$
- 30 agents have preference $\omega_3 \succ \omega_2 \succ \omega_1$

In the cases:

- ω_1 vs ω_2: 40 prefer ω_1 and 60 prefer ω_2, so ω_2
- ω_1 vs ω_3: 40 prefer ω_1 and 60 prefer ω_3, so ω_3
- ω_2 vs ω_3: 70 prefer ω_2 and 30 prefer ω_2, so ω_2

Hence the winner is ω_2.
We can therefore say that plurality does not satisfy the principle!

7.1.3. Note on ties

When implementing a voting protocol, you must decide how ties are resolved. For this module, assume if there is more than one candidate tied for first, then one is **selected at random as the winner**.

7.1.4. Linear sequential majority election protocol

In the **linear sequential majority election protocol**:

- Do a series of rounds
- Pair of outcomes face each other in pairwise majority election, winner goes to next round
- Order in which outcomes face each other is determined by agenda

  e.g. agenda is $\omega_4, \omega_2, \omega_3, \omega_1$ then first election is between ω_4 and ω_2 and winner goes on to an election with ω_3 and so forth.

This protocol poses the problem that the candidate selected might not only depend on the voter preferences but also the agenda.

We can use a **majority graph** to compactly represent voter preferences:

- nodes represent candidates
- edge from node i to j iff i would beat j in a majority election

Example vote

Let $\Omega = \{\omega_1, \omega_2, \omega_3\}$:

- 33 agents have preference $\omega_1 \succ \omega_2 \succ \omega_3$
- 33 agents have preference $\omega_3 \succ \omega_1 \succ \omega_2$
- 33 agents have preference $\omega_2 \succ \omega_3 \succ \omega_1$

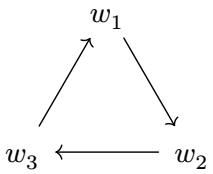


Figure 3: Majority graph

If we want ω_1 to win, we can fix the agenda to avoid w_1 having to fight w_3:

$$w_3, w_2, w_1$$

Likewise, we can pair off any other two to make the third win.

7.1.5. Instant runoff voting protocol

In the **instant runoff voting protocol**:

- Series of 'mini-elections' are held
- In each round:
  - Votes are counted for everyone's top choice and candidate with fewest votes is eliminated
  - If a voter's first choice has been eliminated, their vote goes to their second choice, and so forth.

Example vote

Let $\Omega = \{w, s, t, p\}$:

- 10 voters have preference $w \succ t \succ s \succ p$
- 40 voters have preference $s \succ w \succ p \succ t$
- 60 voters have preference $t \succ s \succ w \succ p$
- 50 voters have preference $p \succ w \succ s \succ t$

Round 1: least votes for $w(10)$ so eliminated:

- 10 voters have preference $t \succ s \succ p$
- 40 voters have preference $s \succ p \succ t$
- 60 voters have preference $t \succ s \succ p$
- 50 voters have preference $p \succ s \succ t$

Round 2: least votes for $s(40)$ so eliminated:

- 10 voters have preference $t \succ p$
- 40 voters have preference $p \succ t$
- 60 voters have preference $t \succ p$
- 50 voters have preference $p \succ t$

Round 3: least votes for $t(70)$ so $p(90)$ wins!

7.1.6. Copeland rule voting protocol

In the **Copeland rule voting protocol**: Candidates face in pairwise majority elections, receiving:

- 1 point for every pairwise majority election they win
- -1 point for every pairwise majority election they lose
- no points otherwise

The candidate with the most points is the winner.

Example vote

Let $\Omega = \{w, s, t, p\}$:

- 10 voters have preference $w \succ t \succ s \succ p$
- 40 voters have preference $s \succ w \succ p \succ t$
- 60 voters have preference $t \succ s \succ w \succ p$
- 50 voters have preference $p \succ w \succ s \succ t$

Pairwise elections:

- $w, s \to 60 : 100 \Rightarrow -1, 1$
- $w, t \to 100 : 60 \Rightarrow 1, -1$
- $w, p \to 110 : 50 \Rightarrow 1, -1$
- $s, t \to 90 : 70 \Rightarrow 1, -1$
- $s, p \to 110 : 50 \Rightarrow 1, -1$
- $t, p \to 70 : 90 \Rightarrow -1, 1$

The vote tally is: $w = 1, s = 3, t = -3, p = -1$
So the winner is s!

7.1.7. Borda count voting protocol

In the **Borda count protocol**, where m is the number of candidates, each candidate receives:

- $(m-1)$ points for each voter listing them as first choice
- $(m-2)$ points for each voter listing them as second choice
- $(m-n)$ points for each voter listing them as n-th choice

Candidate with most points is the winner.

Example vote

Let $\Omega = \{w, s, t, p\}$:

- 10 voters have preference $w \succ t \succ s \succ p$
- 40 voters have preference $s \succ w \succ p \succ t$
- 60 voters have preference $t \succ s \succ w \succ p$
- 50 voters have preference $p \succ w \succ s \succ t$

Vote tallies:

- $w = 10 \cdot 3 + 90 \cdot 2 + 60 = 270$
- $s = 40 \cdot 3 + 60 \cdot 2 + 60 = 300$ — the winner
- $t = 60 \cdot 3 + 10 \cdot 2 = 200$
- $p = 50 \cdot 3 + 40 = 190$

7.1.8. Positional scoring rules

The Borda count is an instance of a positional scoring rule.

In **positional scoring rule voting protocols**, we have a scoring vector:

$$\vec{S} = (s_1, s_2, ..., s_m)$$

such that:

- m is the number of candidates
- $s_1 \geq s_2 \geq ... \geq s_m \geq 0$
  (each s_i is greater than or equal to zero, sequence never increases)
- $s_1 > s_m$ (sequence must decrease somewhere)

Each candidate receives s_1 points for each voter listing them as first choice, s_2 for each as second, and so on.

Under Borda count, this vector is $\vec{S} = (m-1, m-2, m-3, ..., 0)$.

> **Note**
>
> Positional scoring rules do not satisfy the Condorcet principle!
> Need to be able to prove this for tutorial Qs.

7.2. Voting protocol properties

A voting protocol is said to be a **dictatorship** iff there is some voter i such that the winner is always the first choice of i. *Undesirable property.*

For social welfare functions, voting protocols can be said to be a dictatorship if the outcome is always identical to some voter i's preference profile.

A voting protocol is said to be **surjective** iff for every candidate there is some profile of voter preferences such that the candidate will win; every candidate has a chance of winning.
Desirable property.

A voting protocol is said to be **resolute** iff there is always a unique winner (i.e. no ties). *Often a desirable property.* Protocols we look at can be made resolute, by adding a random tie breaker.

A voting protocol is **strategy proof** for a particular voter i iff there is no voter profile $(\underset{1}{\succ}, ..., \underset{i}{\succ}, ..., \underset{n}{\succ})$ such that i prefers the outcome determined by $(\underset{1}{\succ}, ..., \overset{*}{\underset{i}{\succ}}, ..., \underset{n}{\succ})$ to outcome determined by $(\underset{1}{\succ}, ..., \underset{i}{\succ}, ..., \underset{n}{\succ})$ where $\overset{*}{\underset{i}{\succ}}$ is an untruthful representation of i's preferences; i does not have any incentive to misrepresent their true preferences.

A voting protocol is therefore strategy proof iff it is strategy proof for all voters.
i.e. nobody has any incentive to misrepresent their true preferences

Example in practice

Suppose you have a plurality vote and votes are spread as:
- 44 left-wing voters: $\omega_{\text{lab}} \succ \omega_{\text{libdem}} \succ \omega_{\text{con}}$
- 12 centre-left voters: $\omega_{\text{libdem}} \succ \omega_{\text{lab}} \succ \omega_{\text{con}}$
- 44 right-wing voters: $\omega_{\text{con}} \succ \omega_{\text{libdem}} \succ \omega_{\text{lab}}$

It makes more sense for the centre-left voters to strategically vote for labour instead of libdem.

The **Gibbard-Satterthwait Theorem** states if we have three or more candidates, any *resolute* voting protocol that is *surjective* and *strategy proof* is a *dictatorship*; i.e. it is not possible to design a voting protocol that is resolute, surjective, strategy proof, and not a dictatorship.

Voting protocols we have looked at are resolute (assuming random tie-breakers), surjective, and not dictatorships. Therefore they cannot be strategy proof.

A voting protocol is said to be **weakly Pareto** iff there are candidates ω_i and ω_j if all voters prefer ω_i to ω_j, then w_j will not be selected as the winner. *Desirable property.*

Independence of irrelevant alternatives holds for a voting protocol iff the resulting aggregated preference over two candidates ω_x and ω_y depends only on individual preferences over ω_x and ω_y.

Formally, if we let $\Pi^a = \left(\underset{1}{\overset{a}{\succ}}, ..., \underset{n}{\overset{a}{\succ}}\right)$ and $\Pi^b = \left(\underset{1}{\overset{b}{\succ}}, ..., \underset{n}{\overset{b}{\succ}}\right)$ be two different voter profiles such that:
- preferences that appear in Π^a are expressed over set of outcomes $\Omega^a = \left\{\omega_1^a, ..., \omega_j^a\right\}$
- preferences that appear in Π^b are expressed over set of outcomes $\Omega^b = \left\{\omega_1^a, ..., \omega_j^a, \omega_1^b, ..., w_k^b\right\}$ (so $\Omega^a \subseteq \Omega^b$)
- for all $\omega_a^x, \omega_y^a \in \Omega^a$, for all voters i, $\omega_x^a \underset{i}{\overset{a}{\succ}} w_y^a$ iff $\omega_x^a \underset{i}{\overset{b}{\succ}} \omega_y^a$

If the winner determined by Π^a is ω_w, then the winner determined by Π^b is either ω_w or it is a member of the set $\Omega^b \setminus \Omega^a$.

This is a desirable property.
It is *not satisfied* by Borda count, plurality, instant runoff or sequenial majority elections.

Arrow's Theorem states if we have three or more candidates, any voting protocol that is *weakly Pareto* and *independent of irrelevant alternatives* is a *dictatorship*; it is not possible to design a voting protocol with both properties without being a dictatorship.

Both Gibbard-Satterthwait Theorem and Arrow's Theorem tell us its impossible to design democratic voting protocols that have intuitively desirable properties.

However, both use the **universal domain assumption**: voters can specify any preference they like. We can instead, restrict the preference orders the voters are allowed to prevent these negative results from holding.

A domain has a **single-peaked preference** if there is a natural 'left-to-right' ordering on candidates such that:
- $\text{agent's first choice} \gg \omega_i \gg \omega_j$ implies $\omega_i \underset{i}{>} \omega_j$
- and $\omega_j \gg \omega_i \gg \text{agent's first choice}$ implies $\omega_i \underset{i}{>} \omega_j$

This applies in domains such as left-right politics, legal drinking age, speed limit.

Example

Suppose we are considering legal drinking age where

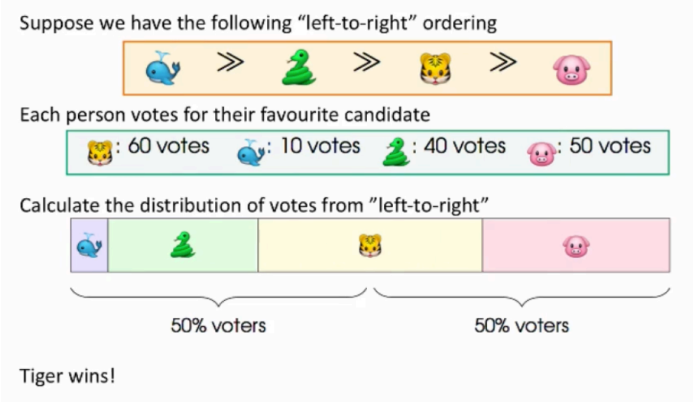
$$16 \gg 17 \gg 18 \gg 19 \gg 20 \gg 21$$

The agent's first choice is 18, so we assume:
- $19 \succ 20, 19 \succ 21, 20 \succ 21$ as $\text{choice}(18) \gg 19 \gg 20 \gg 21$
- $17 \succ 16$ as $16 \ll 17 \ll \text{choice}(18)$

Which results in $18 \succ 17 \succ 19 \succ 20 \succ 21 \succ 16$!

Given a fixed 'left-to-right' order, we can apply a **median voter rule**:
- Each person votes for a single candidate.
- Winner is determined by median voter's ballot.

Example



If an odd number of votes is given and single-peaked preferences are used:
- there is a Condorcet winner elected by median voter rule
- the Gibbard-Satterthwait theorem fails for median voter rule
- Arrow's Theorem fails for medium voter rule

8. Allocating Scarce Resources

Auction theory is a branch of economics that has been found applications in CS including job scheduling, routing, and bandwidth allocation.

8.1. Single-Item Auction Protocols

8.1.1. English auctions

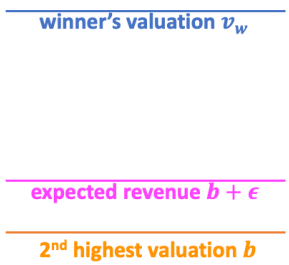
English Auctions

- first price
- open cry
- ascending

Auctioneer sets a starting price, bidders announce bids higher than previous bid (sometimes restricted to increments). Auction may close by set deadline or after no new bids are made within some fixed time period.

Dominant strategy for English auctions
Successively bid the smallest amount possible above current bid until the crurent bid price reaches your true valuation of the item. Bidders do not want to pay more for the item than they think it is worth. If your true valuation is v, you bid as little as possible until bid is $\leq v$.

If all bidders act rationally according to the dominant strategy, the bidder who values the item the most will win the auction.



So the auctioneer can expect to make just over the second highest valuation — the larger the difference between highest and second bidder valuations is how much more then auctioneer will miss out on.

If there is a lack of competition then an English auction is risky for the auctioneer. On the flip side, if there are more bidders who highly value the item, this will push up the amount the winner has to pay.

A **shill bid** is a bid placed by the auctioneer, or by agents in collusion with the auctioneer, in an attempt to increase price paid by the winner.

This introduces a risk that the auctioneer/colluder wins the item themselves.

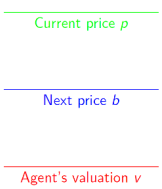
8.1.2. Dutch auctions

Dutch Auctions

- first price
- open cry
- descending

Auctioneer sets a high price then lowers it in fixed increments until an agent is prepared to bid at the current asking price. Agent making the bid wins the auction and pays the amount they bid.

Dominant strategy for dutch auctions



No incentive for agent to bid until price being offered by auctioneer reaches their true valuation so the dominant strategy is to wait until the price on offer has dropped below their true valuation and thne to bid just before any other bidder chooses to bid. If a bidder waits too long to bid after the price has dropped, they risk another making a bid and winning the item. Agent bidding at next price b, will pay b for item and lose $b - v$ utility.

Risk attitudes / how much can the auctioneer expect to make?

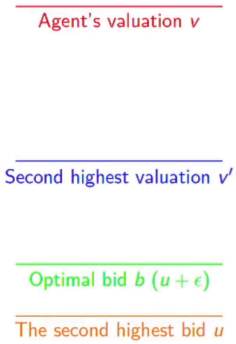
- Risk averse: agents prefer a 'sure' thing
  Auctioneers can expect more revenue from Dutch auction than English auction
- Risk seeking: agents prefer to take a gamble
  Auctioneers can expect more revenue from English auction than Dutch auction
- Risk neutral: agents are indifferent

8.1.3. First price sealed bid auctions

First price sealed bid auctions

- first price
- sealed bid
- one shot

Bidders submit a private bid to the auctioneer with no subsequent rounds. Agent with the highest bid wins and pays the price they bid.



Dominant strategy for first price sealed bid auctions
Strategically isomorphic to Dutch auctions: bidders should wait until price is lower than their valuation. However long they wait will depend on their risk attitude.

How much can the auctioneer expect to make?

- If bidders are risk averse, we can expect more revenue from a first-price sealed bid auction.
- If bidders are risk seeking, we can expect more from an English auction.

8.1.4. Vickrey auctions

Vickrey Auctions

- second price
- sealed bid
- one shot

Bidders submit a private bid to the auctioneer, with no subsequent rounds. Agent with highest bid wins and pays the price of the second highest bid.

Dominant strategy for Vickrey auctions
The dominant strategy is to bid your true valuation.

If you:

- bid more than true valuation: you are more likely to win but can spend more; nothing to gain from bidding more than true valuation since we always pay second highest bid
- bid less than true valuation: stand less of a chance to win, amount paid will not be affected on lose

Similar expected revenue to English auction — paying second highest valuation.

An **anti-social bid** is one made to not win the item but force the highest bidder to pay more than is necessary. Suppose a dishonest agent who has no interest in obtaining the item but would like to see the winning agent in paying more for the item.

Bidders may **collude** by agreeing to buy the item for a lower price from the auctioneer, later selling the item for a higher price and splitting profit between themselves.

8.2. Combinatorial Auctions (multiple items)

Combinatorial auctions are those in which multiple goods are auctioned simulatenously, where some goods may be identical and some may be different. Agents have preference over different subsets of goods.

Formally defined:

- $\mathcal{Z} = \{z_1, ..., z_m\}$: set of items to be auctioned
- $Ag = \{ag_1, ..., ag_n\}$: set of bidders
- Each bidder ag_i has a valuation function $v_i : 2^{\mathcal{Z}} \to \mathbb{R}$ such that, for every bundle $Z \subseteq \mathcal{Z}$, the value $v_{i(Z)}$ indicates how much Z would be worth to ag_i.
- Assume an agent is never worse off having more goods:
  for all $ag_i \in Ag$, for all $Z_j, Z_k \subseteq \mathcal{Z}$, if $Z_j \subseteq Z_k$, then $v_{i(Z_j)} \leq v_{i(Z_k)}$
- Assume an agent gets no value for empty allocation:
  for all $ag_i \in Ag$, $v_{i(\emptyset)} = 0$
- An outcome of the auction is an allocation of goods to the bidders: $(Z_1, ..., Z_n)$
  Where $Z_i \subseteq \mathcal{Z}$ is goods allocated to bidder ag_i and $Z_i \cap Z_j = \emptyset$ (allocated to at most one bidder).

Combinatorial auctions should maximise social welfare (maximise sum of utilities received by bidders), formally:

$$\text{sw}((Z_1, ..., Z_n), (v_1, ..., v_n)) = \sum_{i=1}^{n} v_i(Z_i)$$

8.2.1. Vickrey-Clarke-Groves

The Vickrey-Clarke-Groves (VCG) mechanism dictates that:

- each agent submits a valuation function to auctioneer
  agents $ag_i \subseteq Ag$ simulatenously declare valuation function $\hat{v}_i$
- auctioneer works out allocation that will maximise social welfare
  involves computing allocation $\langle Z_1^*, ..., Z_n^* \rangle$ with maximal social utility according to valuations
- goods are allocated according to $\langle Z_1^*, ..., Z_n^* \rangle$
- amount each agent must pay is calculated
  each agent ag_i pays amount p_i to auctioneer, where p_i is computed as:
  - let $\langle Z_1', ..., Z_n' \rangle$ denotes allocation chosen at step 2 had the agent ag_i declared valuation $\hat{v}_i(Z) = 0$ in the first step for all $Z \in \mathcal{Z}$ (and every other agent apart from ag_i declared same valuation as it did in step 1)
  - Then $p_i = \underbrace{\sum_{j \in Ag : j \neq i} \hat{v}_j(Z_j')}_{\substack{\text{social welfare} \\ \text{allocating with } \hat{v}_i(Z)=0}} - \underbrace{\sum_{j \in Ag : j \neq i} \hat{v}_j(Z_j^*)}_{\substack{\text{total value to} \\ \text{other agents}}}$

Dominant strategy for Vickrey-Clarke-Groves mechanism
The dominant strategy is to bid your true valuation.

An implementation and interactive demo is provided in `jupyter/vcg.ipynb`!

9. Reasoning with arguments

The **bold** learning criteria are covered in 7CCSMEAI Reasoning and Communication:

- WLO9.1. **Construct arguments from a knowledge base.**
- WLO9.2. **Identify attacks between arguments.**
- WLO9.3. **Apply the labelling approach to evaluate arguments.**
- WLO9.4. Determine which arguments are **acceptable under the grounded**, the preferred sceptical, and the preferred credulous semantics.
- WLO9.5. Appreciate the value of argumentation and be able to identify applications where it can be beneficial.

> **Important**
>
> Parts of this are skipped in favour of EAI content.

Formal definition of an **argument**
Let Δ be the database of logic formulae.
An argument constructed from Δ is a tuple $\langle \Phi, \phi \rangle$ such that:

1. $\Phi \subseteq \Delta$: support of argument must come from your database
2. $\Phi \nvdash \bot$: support must be consistent
3. $\Phi \vdash \phi$: support must entail the claim
4. there is no $\Phi' \subsetneq \Phi$ such that $\Phi' \vdash \varphi$: support is a minimal set that entails the claim

For an argument $\text{arg}_x = \langle \Phi, \phi \rangle$, we refer to the claim of arg_x as $\text{claim}(\text{arg}_x) = \phi$ and we refer to the support of arg_x as $\text{support}(\text{arg}_x) = \Phi$.

A **rebuttal** is when a claim of an argument a_x contradicts the claim of an argument a_y so leading to a symmetric attack (each attacking the other).

An **undercut** is when a claim of an argument a_x contradicts (some part of) the support of an argument a_y, leading to a one-way attack from a_x to a_y.

In addition to grounded and preferred semantics, we define **complete semantics** as any complete labelling.

We can further specify semantics to be credulous or sceptical:

- An argument is accepted under **preferred credulous semantics** if and only of it is labelled as IN in at least one labelling under preferred semantics.
- An argument is acceptable under **preferred sceptical semantics** if and only if it is labelled as IN in all labellings under the preferred semantics.

Why argue?
Argumentation can handle subjective, incomplete, and uncertain knowledge, and is founded on intuitive evaluative principles already familiar to humans. This means argumentation is a form of AI reasoning that is well-suited to being made transparent and understandable for human users.

10. Argument-based dialogues

10.1. Dialogue typology

Walton and Krabbe describe argument-based dialogue according to: the initial situation the dialogue arises from, the main goal of the dialogue, and the personal aims of each individual agent.

Type	Initial situation	Main goal	Participant's aims
Persuasian	Conflicting points of view	Resolution of such conflict	Persuade the other
Inquiry	General ignorance	Growth of knowledge and agreement	Find or destroy a 'proof'
Deliberation	Need for action	Reach a decision	Influence outcome
Negotiation	Conflict of interests and need for cooperation	Making a deal	Get the best out of it for one self
Info-seeking	Personal ignorance	Spreading knowledge and revealing positions	Gain, pass on, show, or hide personal knowledge

10.2. Dialogue Games

Covered in 7CCSMEAI Reasoning and Communication.

10.3. Strategising in argument dialogues

There are two approaches to dialogue strategies:

- **Model-based**: strategising agent uses a model of some apsect of interlocutor's private state to determine a strategy optimised for that particular interlocutor
- **Model-free**: does not use any information about interlocutor and instead provides a strategy that can be used directly by the agent

An **opponent model** captures information the strategising agent believes to be true about its interlocutor:

- A's beliefs about B's beliefs
- A's beliefs about B's preferences
- A's beliefs about B's goals
- A's beliefs about B's dialogue strategy
- maybe higher order beliefs, such as A's beliefs about B's beliefs about A's beliefs, preferences, goals, and strategy