Inconsistent Values and Extrapolation

1. Value extrapolation, concept extrapolation, model splintering
2. Model splintering: moving from one imperfect model to another
3. Ontological Crises in Artificial Agents' Value Systems by Peter de Blanc
4. Ontological Crisis in Humans
5. Two More Decision Theory Problems for Humans
6. Using vector fields to visualise preferences and make them consistent
7. Inferring utility functions from locally non-transitive preferences
8. Turning Some Inconsistent Preferences into Consistent Ones

Value extrapolation, concept extrapolation, model splintering

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Post written with Rebecca Gorman.

We've written before that model splintering, as we called it then, was a problem with almost all AI safety approaches.

There's a converse to this: solving the problem would help with almost all AI safety approaches. But so far, we've been posting mainly about value extrapolation. In this post, we'll start looking at how other AI safety approaches could be helped.

Definitions

To clarify, let's make four definitions, distinguishing ideas that we'd previously been grouping together:

Model splintering is when the features and concepts that are valid in one world-model, break down when transitioning to another world-model.

Value splintering (or reward splintering) is when the value function (or reward function, or goal, or preference...) becomes invalid due to model splintering.

Concept extrapolation is extrapolating a feature or concept from one world-model to another.

Value extrapolation is concept extrapolation when the particular concept to extrapolate is a value, a preference, a reward function, an agent's goal, or something of that nature.

Thus concept extrapolation is a solution to model splintering, while value extrapolation is a solution to value splintering specifically.

Examples

Consider for example Turner *et al*'s attainable utility. It has a formal definition, but the reason for that definition is that preserving attainable utility is aimed at restricting the "power" of the agent, or at minimising its "side effects".

And it succeeds, in the typical situation. If you measure the attainable utility of an agent, this will give you an idea of its power, and how many side effects it may be causing. However, when we move to general situations, this breaks down: attainable utility preservation no longer restricts power or reduces side effects. So the concepts of power and side effects have splintered when moving from typical situations to general situations. This is the **model splintering**[1]. If we solve **concept extrapolation** for this, then we could extend the concepts of power restriction or side

effect minimisation, to the general situations. And thus successfully create low impact AIs.

Another example is wireheading. We have a reward signal that corresponds to something we desire in the world; maybe the negative of the CO_2 concentration in the atmosphere. This is measured by, say, a series of CO_2 detectors spread over the Earth's surface.

Typically, the reward signal does correspond to what we want. But if the AI hacks its own reward signal, that correspondence breaks down[2]: **model splintering**. If we can extend the reward properly to new situations, we get **concept extrapolation** - which, since this is a reward function, is **value extrapolation**.

Helping with multiple methods

Hence the concept extrapolation/value extrapolation ideas can help with many different approaches to AI safety, not just the value learning approaches.

---

1. Equivalently, we could say that the concepts remain the same, but it's the correlation between "attainable utility preservation" and "power restriction" is what breaks down. ↩

2. There are multiple ways we can see the concepts breaking down. We can see the concept of "measured CO_2" breaking down. We can see the correlation between CO_2 concentration and the reward breaking down. We can see the correlation between the reward and the *reward signal* breaking down. The reason there are so many ways of seeing the breakdown is because most descriptive labels describe collections of correlated features, rather than fundamental concepts. So the descriptions/features/concepts break down when the correlations do. ↩

Model splintering: moving from one imperfect model to another

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

1. The big problem

In the last few months, I've become convinced that there is a key meta-issue in AI safety; a problem that seems to come up in all sorts of areas.

It's hard to summarise, but my best phrasing would be:

- Many problems in AI safety seem to be variations of "this approach seems safe in this imperfect model, but when we generalise the model more, it becomes dangerously underdefined". Call this **model splintering**.
- It is intrinsically worth studying how to (safely) transition from one imperfect model to another. This is worth doing, independently of whatever "perfect" or "ideal" model might be in the background of the imperfect models.

This sprawling post will be presenting examples of model splintering, arguments for its importance, a formal setting allowing us to talk about it, and some uses we can put this setting to.

1.1 In the language of traditional ML

In the language of traditional ML, we could connect all these issues to "out-of-distribution" behaviour. This is the problems that algorithms encounter when the set they are operating on is drawn from a different distribution than the training set they were trained on.

Humans can often see that the algorithm is out-of-distribution and correct it, because we have a more general distribution in mind than the one the algorithm was trained on.

In these terms, the issues of this post can be phrased as:

1. When the AI finds itself mildly out-of-distribution, how best can it extend its prior knowledge to the new situation?
2. What should the AI do if it finds itself strongly out-of-distribution?
3. What should the AI do if it finds itself strongly out-of-distribution, and humans don't know the correct distribution either?

1.2 Model splintering examples

Let's build a more general framework. Say that you start with some brilliant idea for AI safety/alignment/effectiveness. This idea is phrased in some (imperfect) model. Then

"model splintering" happens when you or the AI move to a new (also imperfect) model, such that the brilliant idea is undermined or underdefined.

Here are a few examples:

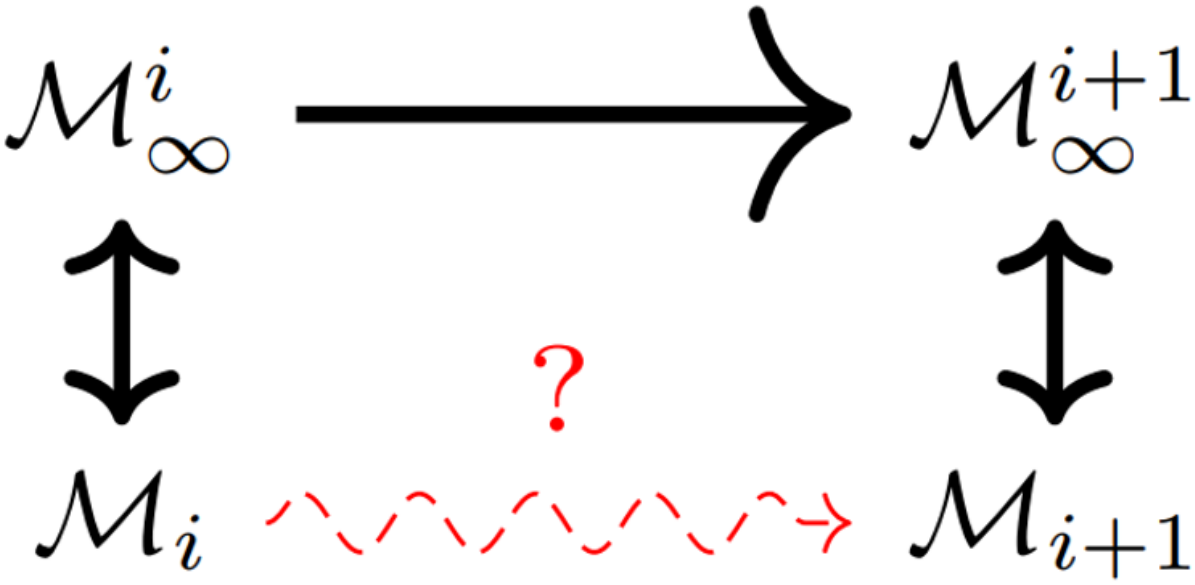
- You design an AI CEO as a money maximiser. Given typical assumptions about the human world (legal systems, difficulties in one person achieving massive power, human fallibilities), this results in an AI that behaves like a human CEO. But when those assumptions fail, the AI can end up feeding the universe to a money-making process that produces nothing of any value.
- Eliezer defined "rubes" as smooth red cubes containing palladium that don't glow in the dark. "Bleggs", on the other hand, are furred blue eggs containing vanadium that glow in the dark. To classify these, we only need a model with two features, "rubes" and "bleggs". Then along comes a furred red egg containing vanadium that doesn't glow in the dark. The previous model doesn't know what to do with it, and if you get a model with more features, it's unclear what to do with this new object.
- Here are some moral principles from history: honour is important for anyone. Women should be protected. Increasing happiness is important. These moral principles made sense in the world in which they were articulated, where features like "honour", "gender", and "happiness" are relatively clear and unambiguous. But the world changed, and the models splintered. "Honour" became hopelessly confused centuries ago. Gender is currently finishing its long splintering (long before we got to today, gender started becoming less useful for classifying people, hence the consequences of gender splintered a long time before gender itself did). Happiness, or at least hedonic happiness, is still well defined, but we can clearly see how this is going to splinter when we talk about worlds of uploads or brain modification.
- Many transitions in the laws of physics - from the ideal gas laws to the more advanced van der Waals equations, or from Newtonain physics to general relativity to quantum gravity - will cause splintering if preferences were articulated in concepts that don't carry over well.

1.3 Avoiding perfect models

In all those cases, there are ways of improving the transition, without needing to go via some idealised, perfect model. We want to define the AI CEO's task in more generality, but we don't need to define this across every possible universe - that is not needed to restrain its behaviour. We need to distinguish any blegg from any rube we are likely to encounter, we don't need to define the platonic essence of "bleggness". For future splinterings - when hedonic happiness splinters, when we get a model of quantum gravity, etc... - we want to know what to do then and there, even if there are future splinterings subsequent to those.

And I think think that model splintering is best addressed directly, rather than using methods that go via some idealised perfect model. Most approaches seem to go for approximating an ideal: from AIXI's set of all programs, the universal prior, KWIK ("Knowing what it knows") learning with a full hypothesis class, Active Inverse Reward Design with its full space of "true" reward functions, to Q-learning which assumes any Markov decisions process is possible. Then the practical approaches rely on approximating this ideal.

Schematically, we can see M_∞ as the ideal, M_∞^i as M_∞ updated with information to time i, and M_i as an approximation of M_∞^i. Then we tend to focus on how well M_i approximates M_∞^i, and on how M_∞^i changes to M_∞^{i+1} - rather than on how M_i relates to M_{i+1}; the red arrow here is underanalysed:



2 Why focus on the transition?

But why is focusing on the $M_i \to M_{i+1}$ transition important?

2.1 Humans reason like this

A lot has been written about image recognition programs going "out-of-distribution" (encountering situations beyond its training environment) or succumbing to "adversarial examples" (examples from one category that have the features of another). Indeed, some people have shown how to use labelled adversarial examples to improve image recognition.

You know what this reminds me of? Human moral reasoning. At various points in our lives, we humans seem to have pretty solid moral intuitions about how the world should be. And then, we typically learn more, realise that things don't fit in the categories we were used to (go "out-of-distribution") and have to update. Some people push stories at us that exploit some of our emotions in new, more ambiguous circumstances ("adversarial examples"). And philosophers use similarly-designed thought experiments to open up and clarify our moral intuitions.

Basically, we start with strong moral intuitions on under-defined features, and when the features splinter, we have to figure out what to do with our previous moral intuitions. A lot of developing moral meta-intuitions, is about learning how to navigate these kinds of transitions; AIs need to be able to do so too.

2.2 There are no well-defined overarching moral principles

Moral realists and moral non-realists agree more than you'd think. In this situation, we can agree on one thing: there is no well-described system of morality that can be "simply" implement in AI.

To over-simplify, moral realists hope to discover this moral system, moral non-realists hope to construct one. But, currently, it doesn't exist in an implementable form, nor is there any implementable algorithm to discover/construct it. So the whole idea of approximating an ideal is wrong.

All humans seem to start from a partial list of moral rules of thumb, rules that they then have to extend to new situations. And most humans do seem to have some meta-rules for defining moral improvements, or extensions to new situations.

We don't know perfection, but we do know improvements and extensions. So methods that deal explicitly with that are useful. Those are things we can build on.

2.3 It helps distinguish areas where AIs fail, from areas where humans are uncertain

Sometimes the AI goes out-of-distribution, and humans can see the error (no, flipping the lego block doesn't count as putting it on top of the other). There are cases when humans themselves go out-of-distribution (see for example siren worlds).

It's useful to have methods available for both AIs and humans in these situations, and to distinguish them. "Genuine human preferences, not expressed in sufficient detail" is not the same as "human preferences fundamentally underdefined".

In the first case, it needs more human feedback; in the second case, it needs to figure out way of resolving the ambiguity, knowing that soliciting feedback is not enough.

2.4 We don't need to make the problems harder

Suppose that quantum mechanics is the true underlying physics of the universe, with some added bits to include gravity. If that's true, why would we need a moral theory valid in every possible universe? It would be useful to have that, but would be strictly harder than one valid in the actual universe.

Also, some problems might be entirely avoided. We don't need to figure out the morality of dealing with a willing slave race - if we never encounter or build one in the first place.

So a few degrees of "extend this moral model in a reasonable way" might be sufficient, without needing to solve the whole problem. Or, at least, without needing to solve the whole problem in advance - a successful nanny AI might be built on these kinds of extensions.

2.5 We don't know how deep the rabbit hole goes

In a sort of converse to the previous point, what if the laws of physics are radically different from what we thought - what if, for example, they allow some forms of time-travel, or have some narrative features, or, more simply, what if the agent moves to an embedded agency model? What if hypercomputation is possible?

It's easy to have an idealised version of "all reality" that doesn't allow for these possibilities, so the ideal can be too restrictive, rather than too general. But the model splintering methods might still work, since it deals with transitions, not ideals.

Note that, **in retrospect**, we can always put this in a Bayesian framework, once we have a rich enough set of environments and updates rules. But this is misleading: the key issue is the missing feature, and figuring out what to do with the missing feature is the real challenge. The fact that we could have done this in a Bayesian way *if we already knew that feature*, is not relevant here.

2.6 We often only need to solve partial problems

Assume the blegg and rube classifier is an industrial robot performing a task. If humans filter out any atypical bleggs and rubes before it sees them, then the robot has no need for a full theory of bleggness/rubeness.

But what it the human filtering is not perfect? Then the classifier still doesn't need a full theory of bleggness/rubeness; it needs methods for dealing with the ambiguities it actually encounters.

Some ideas for AI control - low impact, AI-as-service, Oracles, ... - may require dealing with some model splintering, some ambiguity, but not the whole amount.

2.7 It points out when to be conservative

Some methods, like quantilizers or the pessimism approach rely on the algorithm having a certain degree of conservatism. But, as I've argued, it's not clear to what extent these methods actually are conservative, nor is it easy to calibrate them in a useful way.

Model splintering situations provide excellent points at which to be conservative. Or, for algorithms that need human feedback, but not constantly, these are excellent points to ask for that feedback.

2.8 Difficulty in capturing splintering from the idealised perspective

Generally speaking, idealised methods can't capture model splintering at the point we would want it to. Imagine an ontological crisis, as we move from classical physics to quantum mechanics.

AIXI can go over the transition fine: it shifts from a Turing machine mimicking classical physics observations, to one mimicking quantum observations. But it doesn't notice anything special about the transition: changing the probability of various Turing machines is what it does with observations in general; there's nothing in its algorithm that shows that something unusual has occurred for this particular shift.

2.9 It may help amplification and distillation

This could be seen as a sub-point of some of the previous two sections, but it deserves to be flagged explicitly, since iterated amplification and distillation is one of the major potential routes to AI safety.

To quote a line from that summary post:

> 5. The proposed AI design is to use a safe but slow way of scaling up an AI's capabilities, distill this into a faster but slightly weaker AI, which can be scaled up safely again, and to iterate the process until we have a fast and powerful AI.

At both "scaling up an AI's capabilities", and "distill this into", we can ask the question: has the problem the AI is working on changed? The distillation step is more of a classical AI safety issue, as we wonder whether the distillation has caused any value drift. But at the scaling up or amplification step, we can ask: since the AIs capabilities have changed, the set of possible environments it operates in has changed as well. Has this caused a splintering where the previously safe goals of the AI have become dangerous.

Detecting and dealing with such a splintering could both be useful tools to add to this method.

2.10 Examples of model splintering problems/approaches

At a meta level, most problems in AI safety seem to be variants of model splintering, including:

- The hidden complexity of wishes.
- Ontological crises.
- Conservative/prudential behaviour in algorithms (more specifically, when the algorithm should become conservative).
- How categories are defined.
- The Goodhart problems.
- Out-of-distribution behaviour.

- Low impact and reduced side-effects approaches.
- Underdefined preferences.
- Active inverse reward design.
- Inductive ambiguity identification.
- Wireheading.
- The whole friendly AI problem itself.

Almost every recent post I've read in AI safety, I've been able to connect back to this central idea. Now, we have to be cautious - cure-alls cure nothing, after all, so it's not necessarily a positive sign that *everything* seems to fit into this framework.

Still, I think it's worth diving into this, especially as I've come up with a framework that seems promising for actually solving this issue in many cases.

In a similar concept-space is Abram's orthodox case against utility functions, where he talks about the Jeffrey-Bolker axioms, which allows the construction of preferences from events *without needing full worlds at all*.

3 The virtues of formalisms

This post is dedicated to explicitly modelling the transition to ambiguity, and then showing what we can gain from this explicit meta-modelling. It will do with some formal language (made fully formal in this post), and a lot of examples.

Just as Scott argues that if it's worth doing, it's worth doing with made up statistics, I'd argue that if an idea is worth pursuing, it's worth pursuing with an attempted formalism.

Formalisms are great at illustrating the problems, clarifying ideas, and making us familiar with the intricacies of the overall concept. That's the reason that this post (and the accompanying technical post) will attempt to make the formalism reasonably rigorous. I've learnt a lot about this in the process of formalisation.

3.1 A model, in (almost) all generality

What do we mean by a model? Do we mean mathematical model theory? As we talking about causal models, or causal graphs? AIXI uses a distribution over possible Turing machines, whereas Markov Decision Processes (MDPs) sees states and actions updating stochastically, independently at each time-step. Unlike the previous two, Newtonian mechanics doesn't use time-steps but continuous times, while general relativity weaves time into the structure of space itself.

And what does it mean for a model to make "predictions"? AIXI and MDPs make prediction over future observations, and causal graphs are similar. We can also try running them in reverse, "predicting" past observations from current ones. Mathematical model theory talks about properties and the existence or non-existence of certain objects. Ideal gas laws make a "prediction" of certain properties (eg temperature) given certain others (eg volume, pressure, amount of substance). General relativity establishes that the structure of space-time must obey certain constraints.

It seems tricky to include all these models under the same meta-model formalism, but it would be good to do so. That's because of the risk of ontological crises: we want the AI to be able to continue functioning even if the initial model we gave it was incomplete or incorrect.

3.2 Meta-model: models, features, environments, probabilities

All of the models mentioned above share one common characteristic: once you know some facts, you can deduce some other facts (at least probabilistically). A prediction of the next time step, a retrodiction of the past, a deduction of some properties from other, or a constraint on the shape of the universe: all of these say that if we know some things, then this puts constraints on some other things.

So let's define F, informally, as the set of *features* of a model. This could be the gas pressure in a room, a set of past observations, the local curvature of space-time, the momentum of a particle, and so on.

So we can define a prediction as a probability distribution over a set of possible features F_1, given a base set of features, F_2:

$$Q(F_1 \mid F_2).$$

Do we need anything else? Yes, we need a set of possible environments for which the model is (somewhat) valid. Newtonian physics fails at extreme energies, speeds, or gravitational fields; we'd like to include this "domain of validity" in the model definition. This will be very useful for extending models, or transitioning from one model to another.

You might be tempted to define a set of "worlds" on which the model is valid. But we're trying to avoid that, as the "worlds" may not be very useful for understanding the model. Moreover, we don't have special access to the underlying reality; so we never know whether there actually is a Turing machine behind the world or not.

So define E, the environment on which the model is valid, *as a set of possible features*.

So if we want to talk about Newtonian mechanics, F would be a set of Newtonian features (mass, velocity, distance, time, angular momentum, and so on) and E would be the set of these values where relativistic and quantum effects make little difference.

So see a model as

$$M = \{F, E, Q\},$$

for F a set of features, E a set of environments, and Q a probability distribution. This is such that, for $E_1, E_2 \subset E$, we have the conditional probability:

$Q(E_1 \mid E_2)$.

Though Q is defined for E, we generally want it to be usable from small subsets of the features: so Q should be simple to define from F. And we'll often define the subsets E_i in similar ways; so E_1 might be all environments with a certain angular momentum at time t = 0, while E_2 might be all environments with a certain angular momentum at a later time.

The full formal definition of these can be found here. The idea is to have a meta-model of modelling that is sufficiently general to apply to almost all models, but not one that relies on some ideal or perfect formalism.

3.3 Bayesian models within this meta-model

It's very easy to include Bayesian models within this formalism. If we have a Bayesian model that includes a set W of worlds with prior P, then we merely have to define a set of features F that is sufficient to distinguish all worlds in W: each world is uniquely defined by its feature values[1]. Then we can define E as W, and P on W becomes Q on E; the definitions of terms like $Q(E_1 \mid E_2)$ is just $P(E_1 \cap E_2)P(E_1)/P(E_2)$, per Bayes' rules (unless $P(E_2) = 0$, in which case we set that to 0).

4 Model refinement and splinterings

This section will look at what we can do with the previous meta-model, looking at refinement (how models can improve) and splintering (how improvements to the model can make some well-defined concepts less well-defined).

4.1 Model refinement

Informally, $M^* = \{F^*, E^*, Q^*\}$ is a *refinement* of model $M = \{F, E, Q\}$ if it's at least as expressive as M (it covers the same environments) and is better according to some criteria (simpler, or more accurate in practice, or some other measurement).

At the technical level, we have a map q from a subset E_0^* of E^*, that is surjective onto E. This covers the "at least as expressive" part: every environment in E exists as (possibly multiple) environments in E^*.

Then note that using q^{-1} as a map from subsets of E to subsets of E_0^*, we can define Q_0^* on E via:

$$Q_0^*(E_1 \mid E_2) = Q^*(q^{-1}(E_1) \mid q^{-1}(E_2)).$$

Then this is a model refinement if Q_0^* is 'at least as good as' Q on E, according to our criteria[2].

4.2 Example of model refinement: gas laws

This post presents some subclasses of model refinement, including Q-improvements (same features, same environments, just a better Q), or adding new features to a basic model, called "non-independent feature extension" (eg adding classical electromagnetism to Newtonian mechanics).

Here's a specific gas law illustration. Let $M = \{F, E, Q\}$ be a model of an ideal gas, in some set of rooms and tubes. The F consists of pressure, volume, temperature, and amount of substance, and Q is the ideal gas laws. The E is the standard conditions for temperature and pressure, where the ideal gas law applies. There are multiple different types of gases in the world, but they all roughly obey the same laws.

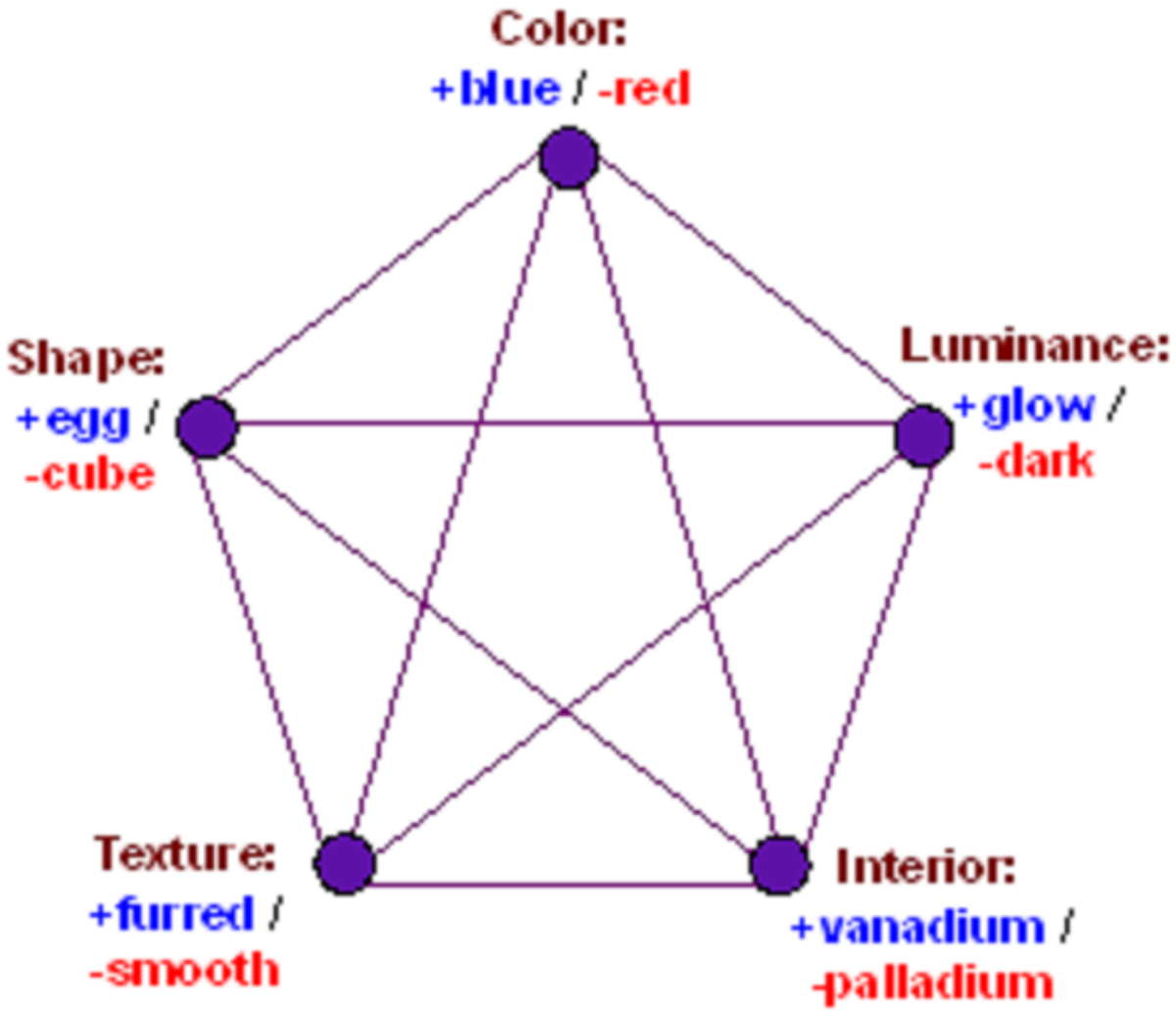
Then compare with model $M^* = \{F^*, E^*, Q^*\}$. The F^* has all the features of F, but also includes the volume that is occupied by one mole of the molecules of the given substance. This allows Q^* to express the more complicated van der Waals equations, which are different for different types of gases. The E^* can now track situations where there are gases with different molar volumes, which include situations where the van der Waals equations differ significantly from the ideal gas laws.

In this case $E_0^* \subset E^*$, since we now distinguish environments that we previously considered identical (environments with same features except for having molar volumes). The q is just projecting down by forgetting the molar volume. Then since $Q_0^* = Q^*$ (van der Waals equations averaged over the distribution of molar volumes) is at least as accurate as Q (ideal gas law), this is a refinement.

4.3 Example of model refinement: rubes and bleegs

Let's reuse Eliezer's example of rubes ("red cubes") and bleggs ("blue eggs").

Bleggs are blue eggs that glow in the dark, have a furred surface, and are filled with vanadium. Rubes, in contrast, are red cubes that don't glow in the dark, have a smooth surface, and are filled with palladium:



Define M by having $F = \{\text{red}, \text{smooth}\}$, E is the set of all bleggs and rubes in some situation, and Q is relatively trivial: it predicts that an object is red/blue if and only if is smooth/furred.

Define M^1 as a refinement of M, by expanding F to $F^1 = \{\text{red}, \text{smooth}, \text{cube}, \text{dark}\}$. The projection $q : E^* \to E$ is given by forgetting about those two last features. The Q^1 is more detailed, as it now connects red-smooth-cube-dark together, and similarly for blue-furred-egg-glows.

Note that E^1 is larger than E, because it includes, e.g., environments where the cube objects are blue. However, all these extra environments have probability zero.

4.4 Reward function refactoring

Let R be a reward function on M (by which we mean that R is define on F, the set of features in M), and M^* a refinement of M.

A *refactoring* of R for M^* is a reward function R^* on the features F^* such that for any $e^* \in E_0^*$, $R^*(e^*) = R(q(e^*))$.

For example, let M and M^1 be from the rube/blegg models in the previous section. Let R_{red} on M simply count the number of rubes - or, more precisely, counts the number of objects to which the feature "red" applies.

Let R^1_{red} be the reward function that counts the number of objects in M^1 to which "red" applies. It's clearly a refactoring of R_{red}.

But so is R^1_{smooth}, the reward function that counts the number of objects in M^1 to which "smooth" applies. In fact, the following is a refactoring of R_{red}, for all $\alpha + \beta + \gamma + \delta = 1$:

$$\alpha R^1_{\text{red}} + \beta R^1_{\text{smooth}} + \gamma R^1_{\text{cube}} + \delta R^1_{\text{dark}}.$$

There are also some non-linear combinations of these features that refactor R, and many other variants (like the strange combinations that generate concepts like grue and bleen).

4.5 Reward function splintering

Model splintering, in the informal sense, is what happens when we pass to a new models in a way that the old features (or a reward function defined by the old features)

no longer apply. It is similar to the web of connotations breaking down, an agent going out of distribution, or the definitions of Rube and Blegg falling apart.

- Preliminary definition: If M^* is a refinement of M and R a reward function on M, then M^* *splinters* R if there are multiple refactorings of R on M^* that disagree on elements of E^* of non-zero probability.

So, note that in the rube/blegg example, M^1 is **not** a splintering of R_{red}: all the refactorings are the same on all bleggs and rubes - hence on all elements of E^1 of non-zero probability.

We can even generalise this a bit. Let's assume that "red" and "blue" are not totally uniform; there exists some rubes that are "redish-purple", while some bleggs are "blueish-purple". Then let M^2 be like M^1, except the colour feature can have four values: "red", "redish-purple", "blueish-purple", and "blue".

Then, as long as rubes (defined, in this instance, by being smooth-dark-cubes) are either "red" or "redish-purple", and the bleggs are "blue", or "blueish-purple", then all refactorings of R_{red} to M^2 agree - because, on the test environment, R_{red} on F perfectly matches up with $R^2_{red} + R^2_{redish\text{-}purple}$ on F^2.

So adding more features does not always cause splintering.

4.6 Reward function splintering: "natural" refactorings

The preliminary definition runs into trouble when we add more objects to the environments. Define M^3 as being the same as M^2, except that E^3 contains one extra object, o_+; apart from that, the environments typically have a billion rubes and a trillion bleggs.

Suppose o_+ is a "furred-rube", i.e. a red-furred-dark-cube. Then R^3_{red} and R^3_{smooth} are two different refactorings of R_{red}, that obviously disagree on any environment that contains o_+. Even if the probability of o_+ is tiny (but non-zero), then M^3 splinters R.

But things are worse than that. Suppose that o_+ is fully a rube: red-smooth-cube-dark, and even contains palladium. Define $(R^3_{red})'$ as being counting the number of red

objects, except for o_+ specifically (again, this is similar to the grue and bleen arguments against induction).

Then both $(R_{red}^3)'$ and R_{red}^3 are refactorings of R_{red}, so M^3 still splinters R_{red}, even when we add another exact copy of the elements in the training set. Or even if we keep the training set for a few extra seconds, or add any change to the world.

So, for any M^* a refinement of M, and R a reward function on E, let's define "natural refactorings" of R:

- The reward function R^* is a natural refactoring of R if it's a reward function on M^* with:

1. $R^* \approx R \circ q$ on E_0^*, and
2. R^* can be defined simply from F^* and R,
3. the F^* themselves are simply defined.

This leads to a full definition of splintering:

- Full definition: If M^* is a refinement of M and R a reward function on M, then M^* *splinters* R if 1) there are no natural refactoring of R on M^*, or 2) there are multiple natural refactorings R^* and $R^{*'}$ of R on M^*, such that $R^* \not\approx R^{*'}$.

Notice the whole host of caveats and weaselly terms here; $R^* \approx R \circ q$, "simply" (used twice), and $R^* \not\approx R^{*'}$. Simply might mean algorithmic simplicity, but $\approx$ and $\not\approx$ are measures of how much "error" we are willing to accept in these refactorings. Given that, we probably want to replace $\approx$ and $\not\approx$ with some *measure* of non-equality, so we can talk about the "degree of naturalness" or the "degree of splintering" of some refinement and reward function.

Note also that:

- **Different choices of refinements can result in different natural refactorings.**

An easy example: it makes a big difference whether a new feature is "temperature", or "divergence from standard temperatures".

4.7 Splintering training rewards

The concept of "reward refactoring" is transitive, but the concept of "natural reward refactoring" need not be.

For example, let E_t be a training environment where red/blue $\iff$ cube/egg, and E_g be a general environment where red/blue is independent of cube/egg. Let F^1 be a feature set with only red/blue, and F^2 a feature set with red/blue and cube/egg.

Then define M_t^1 as using F^1 in the training environment, M_g^2 as using F^2 in the general environment; M_g^1 and M_t^2 are defined similarly.

For these models, M_g^1 and M_t^2 are both refinements of M_t^1, while M_g^2 is a refinement of all three other models. Define R_t^1 as the "count red objects" reward on M_t^1. This has a natural refactoring to R_g^1 on M_g^1, which counts red objects in the general environment.

And R_g^1 has a natural refactoring to R_g^2 on M_g^2, which still just counts the red objects in the general environment.

But there is no natural refactoring from R_t^1 directly to M_g^2. That's because, from F^2's perspective, R_t^1 on M_t^1 might be counting red objects, or might be counting cubes. This is not true for R_g^1 on M_g^1, which is clearly only counting red objects.

Thus when a reward function come from a training environment, we'd want our AI to look for splinterings **directly from a model of the training environment**, rather than from previous natural refactorings.

4.8 Splintering features and models

We can also talk about splintering features and models themselves. For $M = \{F, E, Q\}$, the easiest way is to define a reward function R_{F,s_F} as being the indicator function for feature $F \in F$ being in the set S_F.

Then a refinement M^* splinters the feature F if it splinters some R_{F,S_F}.

The refinement M^* splinters the model M if it splinters at least one of its features.

For example, if M is Newtonian mechanics, including "total rest mass" and M^* is special relativity, then M^* will splinter "total rest mass". Other examples of feature splintering will be presented in the rest of this post.

4.9 Preserved background features

A reward function developed in some training environment will ignore any feature that is always present or always absent in that environment. This allows very weird situations to come up, such as training an AI to distinguish happy humans from sad humans, and it ending up replacing humans with humanoid robots (after all, both happy and sad humans were equally non-robotic, so there's no reason not to do this).

Let's try and do better than that. Assume we have a model $M = \{F, E, Q\}$, with a reward function R_τ defined on E (R_τ and E can be seen as the training data).

Then the feature-preserving reward function R^M, is a function that constrains the environments to have similar feature distributions as E and Q. There are many ways this could be defined; here's one.

For an element $e \in E$, just define

$$R^M(e) = \log(Q(e)).$$

Obviously, this can be improved; we might want to coarse-grain F, grouping together similar worlds, and possibly bounding this below to avoid singularities.

Then we can use this to get the feature-preserving version of R_τ, which we can define as

$$R_\tau^M = (\max_{R_\tau} - R_\tau) \cdot R^M,$$

for $\max_{R_\tau}$ the maximal value of R_τ on E. Other options can work as well, such as $R_\tau + \alpha R_\tau^M$ for some constant $\alpha > 0$.

Then we can ask an AI to use R_τ^M as its reward function, refactoring that, rather than R_τ.

- A way of looking at it: a natural refactoring of a reward function R_τ will preserve all the implicit features that correlate with R_τ. But R_τ^M will also preserve all the implicit features that stay constant when R_τ was defined. So if R_τ measures human happiness vs human unhappiness, a natural refactoring of it will preserves things like "having higher dopamine in their brain". But a natural refactoring of R_τ^M will also preserve things like "having a brain".

4.10 Partially preserved background features

The R_τ^M is almost certainly too restrictive to be of use. For example, if time is a feature, then this will fall apart when the AI has to do something after the training period. If all the humans in a training set share certain features, humans without those features will be penalised.

There are at least two things we can do to improve this. The first is to include more positive and negative examples in the training set; for example, if we include humans and robots in our training set - as positive and negative examples, respectively - then this difference will show up in R_τ directly, so we won't need to use R_τ^M too much.

Another approach would be to explicitly allow certain features to range beyond their typical values in M, or allow highly correlated variables explicitly to decorrelate.

For example, though training during a time period t to t', we could explicitly allow time to range beyond these values, without penalty. Similarly, if a medical AI was trained on examples of typical healthy humans, we could decorrelate functioning digestion from brain activity, and get the AI to focus on the second[3].

This has to be done with some care, as adding more degrees of freedom adds more ways for errors to happen. I'm aiming to look further at this issue in later posts.

5 The fundamental questions of model refinements and splintering

We can now rephrase the out-of-distribution issues of section 1.1 in terms of the new formalism:

1. When the AI refines its model, what would count as a natural refactoring of its reward function?
2. If the refinements splinter its reward function, what should the AI do?
3. If the refinements splinter its reward function, and also splinters the human's reward function, what should the AI do?

6 Examples and applications

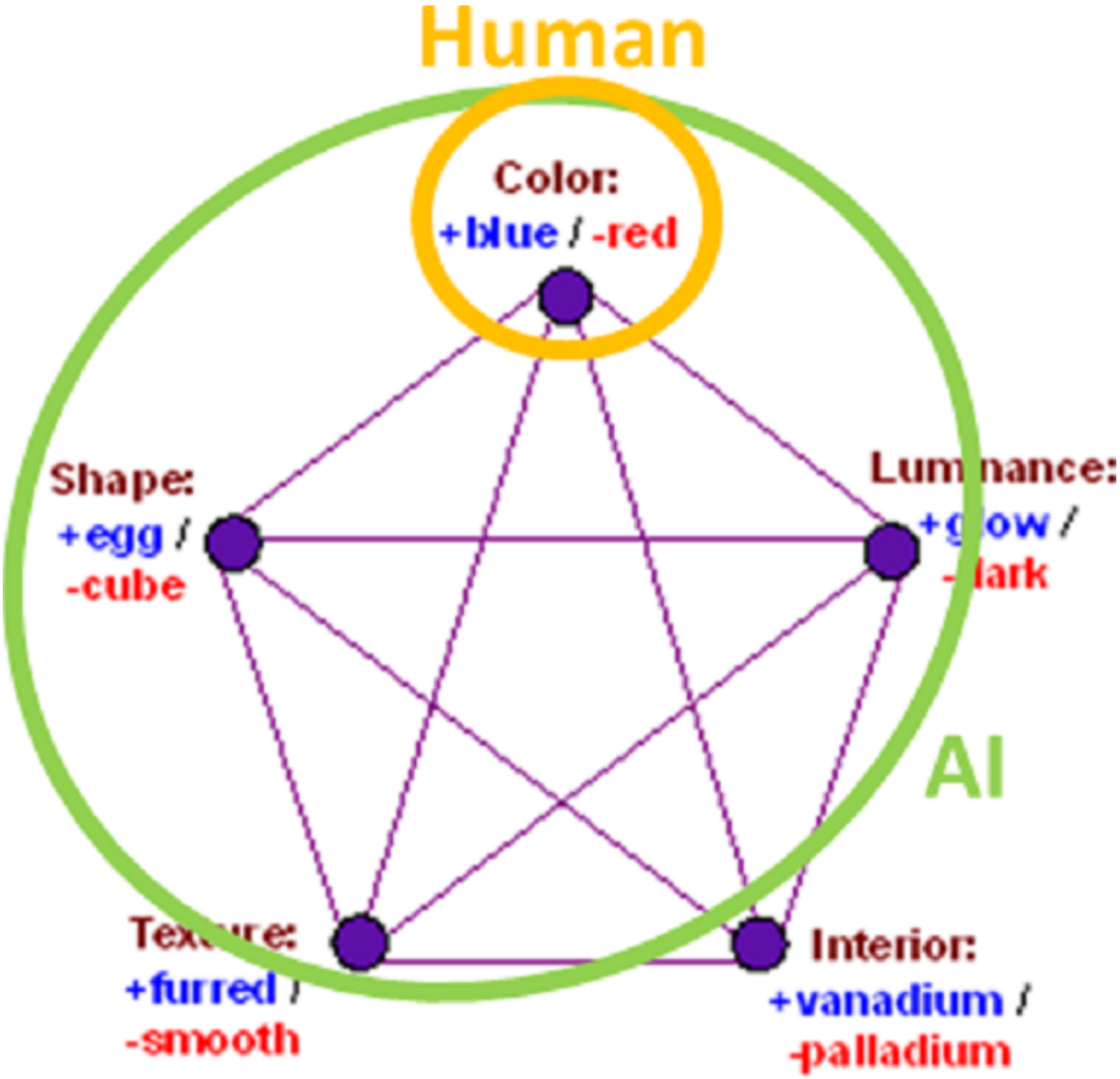
The rest of this post is applying this basic framework, and its basic insights, to various common AI safety problems and analyses. This section is not particularly structured, and will range widely (and wildly) across a variety of issues.

6.1 Extending beyond the training distribution

Let's go back to the blegg and rube examples. A human supervises an AI in a training environment, labelling all the rubes and bleggs for it.

The human is using a very simple model, $M_H = \{F_H, E_t, Q\}$, with the only feature being the colour of the object, and E_t being the training environment.

Meanwhile the AI, having more observational abilities and no filter as to what can be ignored, notices their colour, their shape, their luminance, and their texture. It doesn't know M_H, but is using model $M^1_{AI} = \{F^1, E^1_t, Q^1\}$, where F^1_{AI} covers those four features (note that M^1_{AI} is a refinement of M_H, but that isn't relevant here).



Suppose that the AI is trained to be rube-classifier (and hence a blegg classifier by default). Let R_F be the reward function that counts the number of objects, with feature F, that the AI has classified as rubes. Then the AI could learn many different reward function in the training environment; here's one:

$$R^1 = R^1_{\text{cube}} + 0.5R^1_{\text{smooth}} + 0.5R^1_{\text{dark}} - R^1_{\text{red}}.$$

Note that, even though this gets the colour reward completely wrong, this reward matches up with the human's assessment on the training environment.

Now the AI moves to the larger testing environment E^2, and refines its model minimally to $M^2_{AI} = \{F^1, E^2, Q^1\}$ (extending R^1 to R^2 in the obvious way).

In E^2, the AI sometimes encounters objects that it can only see through their colour.

Will this be a problem, since the colour component of R^2 is pointing in the wrong direction?

No. It still has Q^1, and can deduce that a red object must be cube-smooth-dark, so R^2 will continue treating this as a rube[4].

6.2 Detecting going out-of-distribution

Now imagine the AI learns about the content of the rubes and bleggs, and so refines to a new model that includes vanadium/palladium as a feature in M^3_{AI}.

Furthermore, in the training environment, all rubes have palladium and all bleggs have vanadium in them. So, for M^3_{AI} a refinement of M^1_{AI}, $q^{-1}(E^1_{AI}) \subset E^3_{AI}$ has only palladium-rubes and vanadium-bleggs. But in E^3_{AI}, the full environment, there are rather a lot of rubes with vanadium and bleggs with palladium.

So, similarly to section 4.7, there is no natural refactoring of the rube/blegg reward in M^1_{AI}, to M^3_{AI}. That's because F^3_{AI}, the feature set of M^3_{AI}, includes vanadium/palladium which co-vary with the other rube/blegg features on the training environment (q^{-1}(\E_{AI}^1)), but not on the full environment of E^3_{AI}.

So looking for reward splintering from the training environment is a way of detecting going out-of-distribution - even on features that were not initially detected in the training distribution, by either the human nor the AI.

6.3 Asking humans and Active IRL

Some of the most promising AI safety methods today rely on getting human feedback[5]. Since human feedback is expensive, as in it's slow and hard to get compared with almost all other aspects of algorithms, people want to get this feedback in the most efficient ways possible.

A good way of doing this would be to ask for feedback when the AI's current reward function splinters, and multiple options are possible.

A more rigorous analysis would look at the value of information, expected future splinterings, and so on. This is what they do in Active Inverse Reinforcement Learning; the main difference is that AIRL emphasises an unknown reward function with humans providing information, while this approach sees it more as an known reward function over uncertain features (or over features that may splinter in general environments).

6.4 A time for conservatism

I argued that many "conservative" AI optimising approaches, such as quantilizers and pessimistic AIs, don't have a good measure of when to become more conservative; their parameters q and β don't encode useful guidelines for the right degree of conservatism.

In this framework, the alternative is obvious: AIs should become conservative when their reward functions splinter (meaning that the reward function compatible with the previous environment has multiple natural refactorings), and very conservative when they splinter a lot.

This design is very similar to Inverse Reward Design. In that situation, the reward signal in the training environment is taken as *information* about the "true" reward function. Basically they take all reward functions that could have given the specific reward signals, and assume the "true" reward function is one of them. In that paper, they advocate extreme conservatism at that point, by optimising the minimum of all possible reward functions.

The idea here is almost the same, though with more emphasis on "having a true reward defined on uncertain features". Having multiple contradictory reward functions compatible with the information, in the general environment, is equivalent with having a lot of splintering of the training reward function.

6.5 Avoiding ambiguous distant situations

The post "By default, avoid ambiguous distant situations" can be rephrased as: let M be a model in which we have a clear reward function R, and let M^2 be a refinement of this to general situations. We expect that this refinement splinters R. Let M^1 be like M^2, except with E^1 smaller than E^2, defined such that:

1. An AI could be expected to be able to constrain the world to be in E^1, with high probability,
2. The M^1 is not a splintering of R.

Then that post can be summarised as:

- The AI should constrain the world to be in E^1 and then maximise the natural refactoring of R in M^1.

6.6 Extra variables

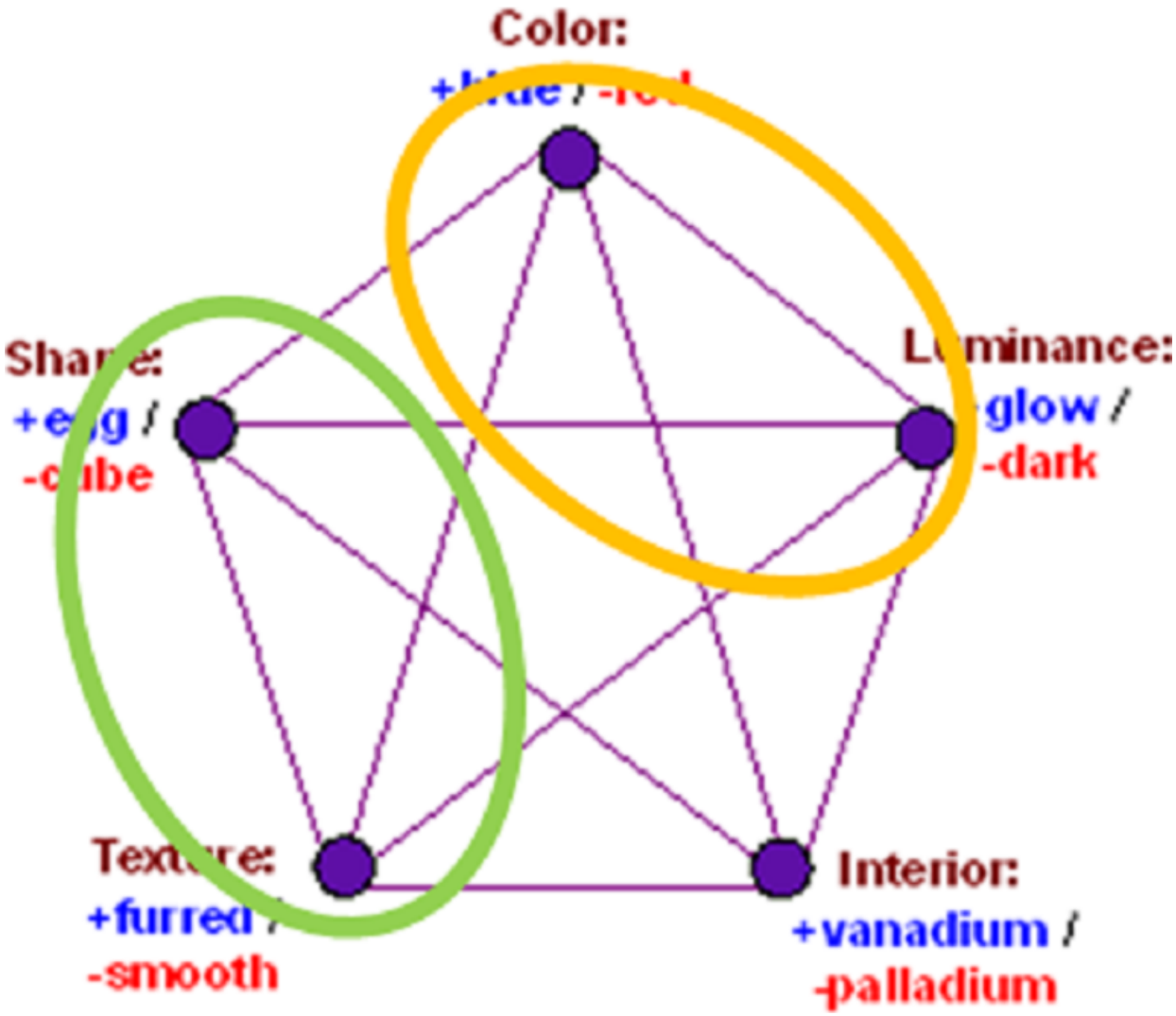
Stuart Russell writes:

> A system that is optimizing a function of n variables, where the objective depends on a subset of size k < n, will often set the remaining unconstrained variables to extreme values; if one of those unconstrained variables is actually something we care about, the solution found may be highly undesirable.

The approach in sections 4.9 and 4.10 explicitly deals with this.

6.7 Hidden (dis)agreement and interpretability

Now consider two agents doing a rube/blegg classifications task in the training environment; each agent only models two of the features:



Despite not having a single feature in common, both agents will agree on what bleggs and rubes are, in the training environment. And when refining to a fuller model that includes all four (or five) of the key features, both agents will agree as to whether a natural refactoring is possible or not.

This can be used to help define the limits of interpretability. The AI can use its own model, and its own designed features, to define the categories and rewards in the training environment. These need not be human-parsable, but we can attempt to interpret them in human terms. And then we can give this interpretation to the AI, as a list of positive and negative examples of our interpretation.

If we do this well, the AI's own features and our interpretation will match up in the training environment. But as we move to more general environments, these may diverge. Then the AI will flag a "failure of interpretation" when its refactoring diverges from a refactoring of our interpretation.

For example, if we think the AI detects pandas by looking for white hair on the body, and black hair on the arms, we can flag lots of examples of pandas and that hair pattern (and non-pandas and unusual hair patterns. We don't use these examples for

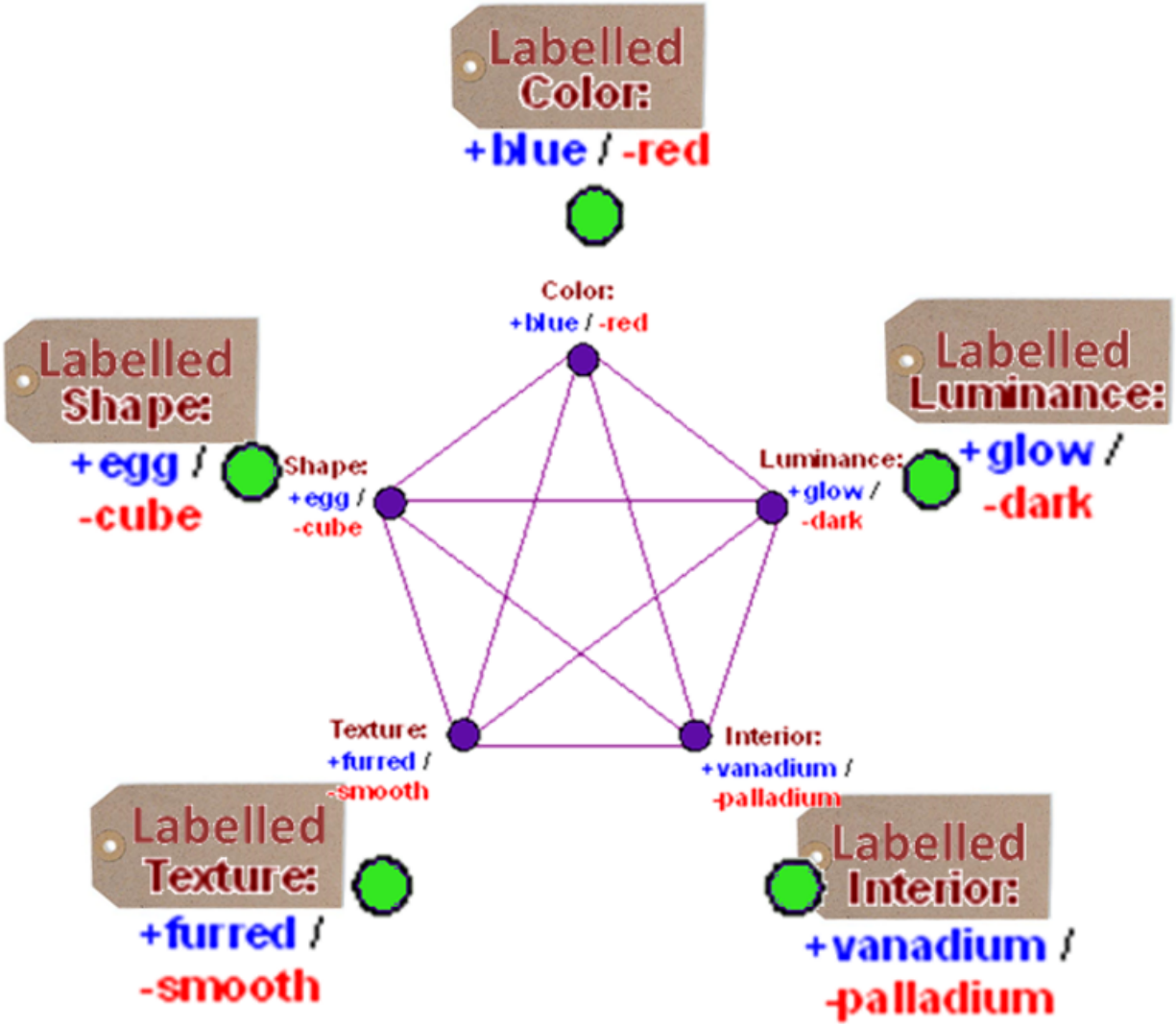
training the AI, just to confirm that, in the training environment, there is a match between "AI-thinks-they-are-pandas" and "white-hair-on-arms-black-hair-on-bodies".

But, in an adversarial example, the AI could detect that, while it is detecting gibbons, this no longer matches up with our interpretaion. A splintering of interpretations, if you want.

6.8 Wireheading

The approach can also be used to detect wireheading. Imagine that the AI has various detectors that allow it to label what the features of the bleggs and rubes are. It models the world with ten features: 5 features representing the "real world" versions of the features, and 5 representing the "this signal comes from my detector" versions.

This gives a total of 10 features, the 5 features "in the real world" and the 5 "AI-labelled" versions of these:



In the training environment, there was full overlap between these 10 features, so the AI might learn the incorrect "maximise my labels/detector signal" reward.

However, when it refines its model to all 10 features *and* environments where labels and underlying reality diverge, it will realise that this splinters the reward, and thus detect a possible wireheading. It could then ask for more information, or have an automated "don't wirehead" approach.

6.9 Hypotheticals, and training in virtual environments

To get around the slowness of the real world, some approaches train AIs in virtual environments. The problem is to pass that learning from the virtual environment to the real one.

Some have suggested making the virtual environment sufficiently detailed that the AI can't tell the difference between it and the real world. But, a) this involves fooling the AI, an approach I'm always wary of, and b) it's unnecessary.

Within the meta-formalism of this post, we could train the AI in a virtual environment which it models by M, and let it construct a model M' of the real-world. We would then motivate the AI to find the "closest match" between M and M', in terms of features and how they connect and vary. This is similar to how we can train pilots in flight simulators; the pilots are never under any illusion as to whether this is the real world or not, and even crude simulators can allow them to build certain skills[6].

This can also be used to allow the AI to deduce information from hypotheticals and thought experiments. If we show the AI an episode of a TV series showing people behaving morally (or immorally), then the episode need not be believable or plausible, if we can roughly point to the features in the episode that we want to emphasise, and roughly how these relate to real-world features.

6.10 Defining how to deal with multiple plausible refactorings

The approach for synthesising human preferences, defined here, can be rephrased as:

- "Given that we expect multiple natural refactorings of human preferences, and given that we expect some of them to go disastrously wrong, here is one way of resolving the splintering that we expect to be better than most."

This is just one way of doing this, but it does show that "automating what AIs do with multiple refactorings" might not be impossible. The following subsection has some ideas with how to deal with that.

6.11 Global, large scale preferences

In an old post, I talked about the concept of "emergency learning", which was basically, "lots of examples, and all the stuff we know and suspect about how AIs can go wrong, shove it all in, and hope for the best". The "shove it all in" was a bit more structured than that, defining large scale preferences (like "avoid siren worlds" and "don't over-optimise") as constraints to be added to the learning process.

It seems we can do better than that here. Using examples and hypotheticals, it seems we could construct ideas like "avoid slavery", "avoid siren worlds", or "don't over-optimise" as rewards or positive/negative examples certain simple training environments, so that the AI "gets an idea of what we want".

We can then label these ideas as "global preferences". The idea is that they start as loose requirements (we have much more granular human-scale preferences than just "avoid slavery", for example), but, the more the world diverges from the training environment, the stricter they are to be interpreted, with the AI required to respect some softmin of all natural refactorings of these features.

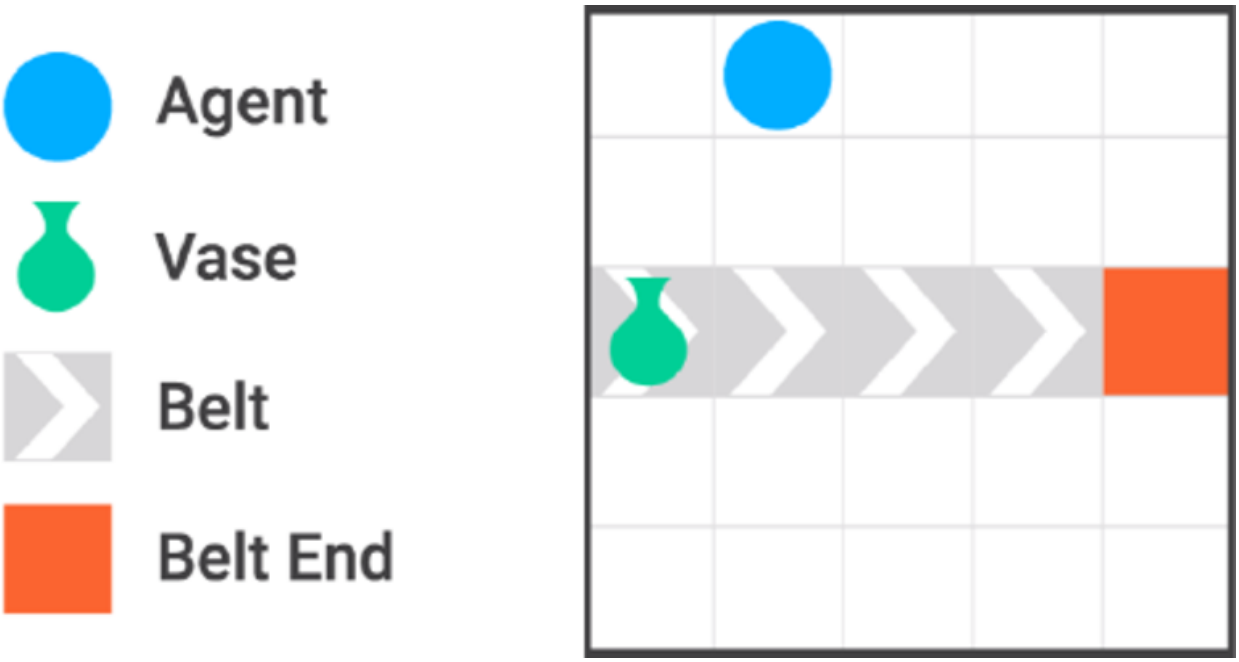
In a sense, we'd be saying "prevent slavery; these are the features of slavery, and in weird worlds, be especially wary of these features".

6.12 Avoiding side-effects

Krakovna et. al. presented a paper on avoiding side-effects from AI. The idea is to have an AI maximising some reward function, while reducing side effects. So the AI would not smash vases or let them break, nor would it prevent humans from eating sushi.

In this environment, we want the AI to avoid knocking the sushi off the belt as it moves:

Here, in contrast, we'd want the AI to remove the vase from the belt before it smashes:



I pointed out some issues with the whole approach. Those issues were phrased in terms of sub-agents, but my real intuition is that syntactic methods are not sufficient to control side effects. In other words, the AI can't learn to do the right thing with sushis and vases, unless it has some idea of what these objects mean to us; we prefer sushis to be eaten and vases to not be smashed.

This can be learnt if the AI has a enough training examples, learning that eating sushi is a general feature of the environments it operates in, while vases being smashed is not. I'll return to this idea in a later post.

6.13 Cancer patients

The ideas of this post were present in implicit form in the idea of training an AI to cure cancer patients.

Using examples of successfully treated cancer patients, we noted they all shared some positive features (recuperating, living longer) and some incidental or negative features (complaining about pain, paying more taxes).

So, using the approach of section 4.9, we can designate that we want the AI to cure cancer; this will be interpreted as increasing all the features that correlate with that.

Using the explicit decorrelation of section 4.10, we can also explicitly remove the negative options from the desired feature sets, thus improving the outcomes even more.

6.14 The genie and the burning mother

In Eliezer's original post on the hidden complexity of wishes, he talks of the challenge of getting a genie to save your mother from a burning building:

> So you hold up a photo of your mother's head and shoulders; match on the photo; use object contiguity to select your mother's whole body (not just her head and shoulders); and define the future function using your mother's distance from the building's center. [...]
>
> You cry "Get my mother out of the building!", for luck, and press Enter. [...]
>
> BOOM! With a thundering roar, the gas main under the building explodes. As the structure comes apart, in what seems like slow motion, you glimpse your mother's shattered body being hurled high into the air, traveling fast, rapidly increasing its distance from the former center of the building.

How could we avoid this? What you want is your mother out of the building. The feature "mother in building" must absolutely be set to false; this is a priority call, overriding almost everything else.

Here we'd want to load examples of your mother outside the building, so that the genie/AI learns the features "mother in house"/"mother out of house". Then it will note that "mother out of house" correlates with a whole lot of other features - like mother being alive, breathing, pain-free, often awake, and so on.

All those are good things. But there are some other features that don't correlate so well - such as the time being earlier, your mother not remembering a fire, not being covered in soot, not worried about her burning house, and so on.

As in the cancer patient example above, we'd want to preserve the features that correlate with the mother out of the house, while allowing decorrelation with the features we don't care about or don't want to preserve.

6.15 Splintering moral-relevant categories: honour, gender, and happiness

If the Antikythera mechanism had been combined with the Aeolipile to produce an ancient Greek AI, and Homer had programmed it (among other things) to "increase people's honour", how badly would things have gone?

If Babbage had completed the analytical engine as Victorian AI, and programmed it (among other things) to "protect women", how badly would things have gone?

If a modern programmer were to combine our neural nets into a superintelligence and program it (among other things) to "increase human happiness", how badly will things go?

There are three moral-relevant categories here, and it's illustrative to compare them: honour, gender, and hedonic happiness. The first has splintered, the second is splintering, and the third will likely splinter in the future.

I'm not providing solutions in this subsection, just looking at where the problems can appear, and encouraging people to think about how they would have advised Homer or Babbage to define their concepts. Don't think "stop using your concepts, use ours instead", because our concepts/features will splinter too. Think "what's the best way they could have extended their preferences even as the features splinter"?

- **6.15.1 Honour**

If we look at the concept of honour, we see a concept that has already splintered.

That article reads like a meandering mess. Honour is "face", "reputation", a "bond between an individual and a society", "reciprocity", a "code of conduct", "chastity" (or "virginity"), a "right to precedence", "nobility of soul, magnanimity, and a scorn of meanness", "virtuous conduct and personal integrity", "vengeance", "credibility", and so on.

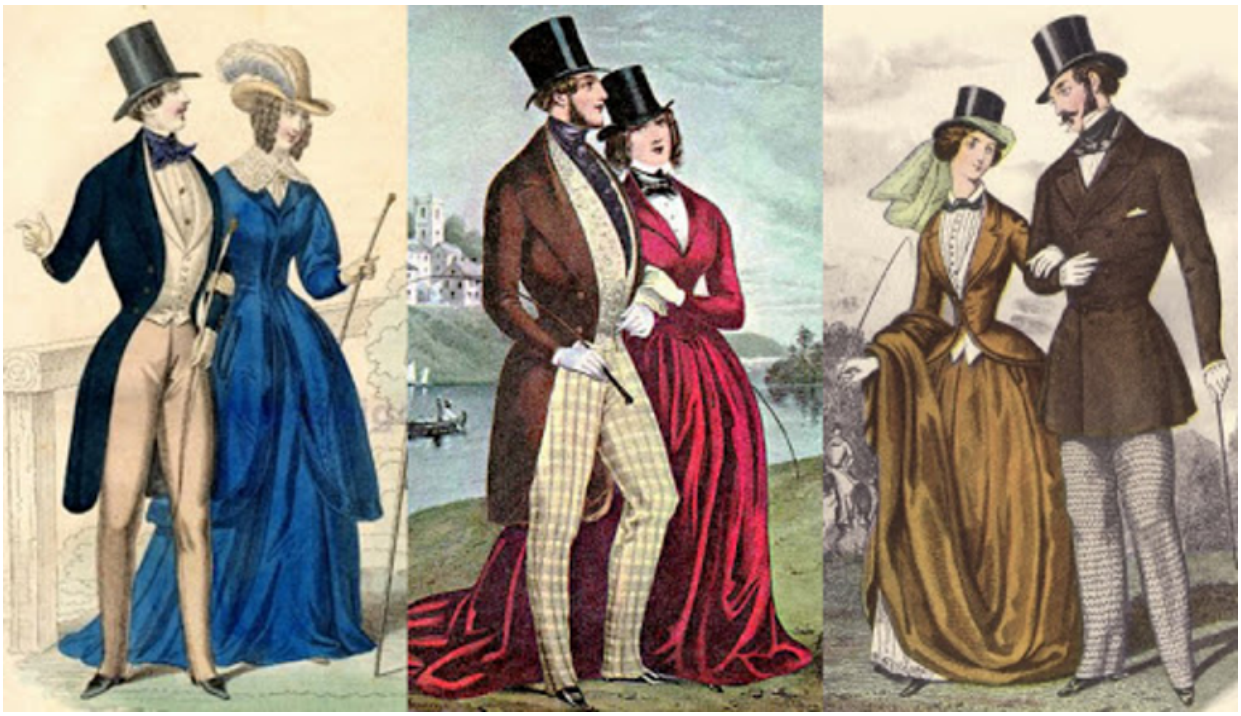
What a basket of concepts! They only seem vaguely connected together; and even places with strong honour cultures differ in how they conceive of honour, from place to place and from epoch to epoch[7]. And yet, if you asked most people within those cultures about what honour was, they would have had a strong feeling it was a single, well defined thing, maybe even a concrete object.

- **6.15.2 Gender**

In his post the categories were made for man, not man for the categories, Scott writes:

> Absolutely typical men have Y chromosomes, have male genitalia, appreciate manly things like sports and lumberjackery, are romantically attracted to women, personally identify as male, wear male clothing like blue jeans, sing baritone in the opera, et cetera.

But Scott is writing this in the 21st century, long after the gender definition has splintered quite a bit. In middle class middle class Victorian England[8], the gender divide was much stronger - in that, from one component of the divide, you could predict a lot more. For example, if you knew someone wore dresses in public, you knew that, almost certainly, they couldn't own property if they were married, nor could they vote, they would be expected to be in charge of the household, might be allowed to faint, and were expected to guard their virginity.



We talk nowadays about gender roles multiplying or being harder to define, but they've actually being splintering for a lot longer than that. Even though we could *define* two genders in 1960s Britain, at least roughly, that definition was a lot less informative than it was in Victorian-middle-class-Britain times: it had many fewer features strongly correlated with it.

- **6.15.3 Happiness**

On to happiness! Philosophers and others have been talking about happiness for centuries, often contrasting "true happiness", or flourishing, with hedonism, or drugged out stupor, or things of that nature. Often "true happiness" is a life of duty to what the philosopher wants to happen, but at least there is some analysis, some breakdown of the "happiness" feature into smaller component parts.

Why did the philosophers do this? I'd wager that it's because the concept of happiness was already somewhat splintered (as compared with a model where "happiness" is a single thing). Those philosophers had experience of joy, pleasure, the satisfaction of a job well done, connection with others, as well as superficial highs from temporary feelings. When they sat down to systematise "happiness", they could draw on the features of their own mental model. So even if people hadn't systematised happiness themselves, when they heard of what philosophers were doing, they probably didn't react as "What? Drunken hedonism and intellectual joy are not the same thing? How dare you say such a thing!"

But looking into the future, into a world that an AI might create, we can foresee many situations where the implicit assumptions of happiness come apart, and only some remain. I say "we can foresee", but it's actually very hard to know exactly how that's going to happen; if we knew it exactly, we could solve the issues now.

So, imagine a happy person. What do you think that they have in life, that are not trivial synonyms of happiness? I'd imagine they have friends, are healthy, think interesting thoughts, have some freedom of action, may work on worthwhile tasks, may be connected with their community, probably make people around them happy as well. Getting a bit less anthropomorphic, I'd also expect them to be a carbon-based life-form, to have a reasonable mix of hormones in their brain, to have a continuity of experience, to have a sense of identity, to have a personality, and so on.

Now, some of those features can clearly be separated from "happiness". Even ahead of time, I can confidently say that "being a carbon-based life-form" is not going to be a critical feature of "happiness". But many of the other ones are not so clear; for example, would someone without continuity of experience or a sense of identity be "happy"?

Of course, I can't answer that question. Because the question has no answer. We have our current model of happiness, which co-varies with all those features I listed and many others I haven't yet thought of. As we move into more and more bizarre worlds, that model will splinter. And whether we assign the different features to "happiness" or to some other concept, is a choice we'll make, not a well-defined solution to a well-defined problem.

However, even at this stage, some answers are clearly better than others; statues of happy people should not count, for example, nor should written stories describing very happy people.

6.16 Apprenticeship learning

In apprenticeship learning (or learning from demonstration), the AI would aim to copy what experts have done. Inverse reinforcement learning can be used for this purpose, by guessing the expert's reward function, based on their demonstrations. It looks for key features in expert trajectories and attempts to reproduce them.

So, if we had an automatic car driving people to the airport, and fed it some trajectories (maybe ranked by speed of delivery), it would notice that passengers would also arrive alive, with their bags, without being pursued by the police, and so on. This is akin to section 4.9, and would not accelerate blindly to get there as fast as possible.

But the algorithm has trouble getting to truly super-human performance[9]. It's far too conservative, and, if we loosen the conservatism, it doesn't know what's acceptable and what isn't, and how to trade these off: since all passengers survived and the car was always painted yellow, their luggage intact in the training data, it has no reason to prefer human survival to taxi-colour. It doesn't even have a reason to have a specific feature resembling "passenger survived" at all.

This might be improved by the "allow decorrelation" approach from section 4.10: we specifically allow it to maximise speed of transport, while keeping the other features (no accidents, no speeding tickets) intact. As in section 6.7, we'll attempt to check that the AI does prioritise human survival, and that it will warn us if a refactoring moves it away from this.

---

1. Now, sometimes worlds $w_1, w_2 \in W$ may be indistinguishable for any feature set. But in that case, they can't be distinguished by any observations, either, so their relative probabilities won't change: as long as it's defined, $P(w_1|o)/P(w_2|o)$ is constant for all observations o. So we can replace w_1 and w_2 with $\{w_1, w_2\}$, of prior probability $P(\{w_1, w_2\}) = P(w_1) + P(w_2)$. Doing this for all indistinguishable worlds (which form an equivalence class) gives W', a set of distinguishable worlds, with a well defined P on it. ↩

2. It's useful to contrast a refinement with the "abstraction" defined in this sequence. An abstraction throws away irrelevant information, so is not generally a refinement. Sometimes they are exact opposites, as the ideal gas law is an abstraction of the movement of all the gas particles, while the opposite would be a refinement.

   But they are exact opposites either. Starting with the neurons of the brain, you might abstract them to "emotional states of mind", while a refinement could also add "emotional states of mind" as new features (while also keeping the old features). A splintering is more the opposite of an abstraction, as it signals that the old abstraction features are not sufficient.

It would be interesting to explore some of the concepts in this post with a mixture of refinements (to get the features we need) and abstractions (to simplify the models and get rid of the features we don't need), but that is beyond the scope of this current, already over-long, post. ↩

3. Specifically, we'd point - via labelled examples - at a clusters of features that correlate with functioning digestion, and another cluster of features that correlate with brain activity, and allow those two clusters to decorrelate with each other. ↩

4. It is no coincidence that, if R and R' are rewards on M, that are identical on E, and if R^* is a refactoring of R, then R^* is also a refactoring of R'. ↩

5. Though note there are some problems with this approach, both in theory and in practice. ↩

6. Some more "body instincts" skills require more realistic environments, but some skills and procedures can perfectly well be trained in minimal simulators. ↩

7. You could define honour as "behaves according to the implicit expectations of their society", but that just illustrates how time-and-place dependent honour is. ↩

8. Pre 1870. ↩

9. It's not impossible to get superhuman performance from apprenticeship learning; for example, we could select the best human performance on a collection of distinct tasks, and thus get the algorithm to have a overall performance that no human could ever match. Indeed, one of the purposes of task decomposition is to decompose complex tasks in ways that allow apprenticeship-like learning to have safe and very superhuman performance on the whole task. ↩

Ontological Crises in Artificial Agents' Value Systems by Peter de Blanc

I saw this go by on arXiv, and thought it deserved a discussion here.

> Decision-theoretic agents predict and evaluate the results of their actions using a model, or ontology, of their environment. An agent's goal, or utility function, may also be specified in terms of the states of, or entities within, its ontology. If the agent may upgrade or replace its ontology, it faces a crisis: the agent's original goal may not be well-defined with respect to its new ontology. This crisis must be resolved before the agent can make plans towards achieving its goals. We discuss in this paper which sorts of agents will undergo ontological crises and why we may want to create such agents. We present some concrete examples, and argue that a well-defined procedure for resolving ontological crises is needed. We point to some possible approaches to solving this problem, and evaluate these methods on our examples.

I'll post my analysis and opinion of this paper in a comment after I've taken some time to digest it.

Ontological Crisis in Humans

Imagine a robot that was designed to find and collect spare change around its owner's house. It had a world model where macroscopic everyday objects are ontologically primitive and ruled by high-school-like physics and (for humans and their pets) rudimentary psychology and animal behavior. Its goals were expressed as a utility function over this world model, which was sufficient for its designed purpose. All went well until one day, a prankster decided to "upgrade" the robot's world model to be based on modern particle physics. This unfortunately caused the robot's utility function to instantly throw a domain error exception (since its inputs are no longer the expected list of macroscopic objects and associated properties like shape and color), thus crashing the controlling AI.

According to Peter de Blanc, who used the phrase "ontological crisis" to describe this kind of problem,

> Human beings also confront ontological crises. We should find out what cognitive algorithms humans use to solve the same problems described in this paper. If we wish to build agents that maximize human values, this may be aided by knowing how humans re-interpret their values in new ontologies.

I recently realized that a couple of problems that I've been thinking over (the nature of selfishness and the nature of pain/pleasure/suffering/happiness) can be considered instances of ontological crises in humans (although I'm not so sure we necessarily have the cognitive algorithms to solve them). I started thinking in this direction after writing this comment:

> This formulation or variant of TDT requires that before a decision problem is handed to it, the world is divided into the agent itself (X), other agents (Y), and "dumb matter" (G). I think this is misguided, since the world doesn't really divide cleanly into these 3 parts.

What struck me is that even though the world doesn't divide cleanly into these 3 parts, *our models* of the world actually do. In the world models that we humans use on a day to day basis, and over which our utility functions seem to be defined (to the extent that we can be said to have utility functions at all), we do take the Self, Other People, and various Dumb Matter to be ontologically primitive entities. Our world models, like the coin collecting robot's, consist of these macroscopic objects ruled by a hodgepodge of heuristics and prediction algorithms, rather than microscopic particles governed by a coherent set of laws of physics.

For example, the amount of pain someone is experiencing doesn't seem to exist in the real world as an XML tag attached to some "person entity", but that's pretty much how our models of the world work, and perhaps more importantly, that's what our utility functions expect their inputs to look like (as opposed to, say, a list of particles and their positions and velocities). Similarly, a human can be selfish just by treating the object labeled "SELF" in its world model differently from other objects, whereas an AI with a world model consisting of microscopic particles would need to somehow inherit or learn a detailed description of itself in order to be selfish.

To fully confront the ontological crisis that we face, we would have to upgrade our world model to be based on actual physics, and simultaneously translate our utility functions so that their domain is the set of possible states of the new model. We

currently have little idea how to accomplish this, and instead what we do in practice is, as far as I can tell, keep our ontologies intact and utility functions unchanged, but just add some new heuristics that in certain limited circumstances call out to new physics formulas to better update/extrapolate our models. This is actually rather clever, because it lets us make use of updated understandings of physics without ever having to, for instance, decide exactly what patterns of particle movements constitute pain or pleasure, or what patterns constitute oneself. Nevertheless, this approach hardly seems capable of being extended to work in a future where many people may have nontraditional mind architectures, or have a zillion copies of themselves running on all kinds of strange substrates, or be merged into amorphous group minds with no clear boundaries between individuals.

By the way, I think nihilism often gets short changed around here. Given that we do not actually have at hand a solution to ontological crises in general or to the specific crisis that we face, what's wrong with saying that the solution set may just be null? Given that evolution doesn't constitute a particularly benevolent and farsighted designer, perhaps we may not be able to do much better than that poor spare-change collecting robot? If Eliezer is worried that actual AIs facing actual ontological crises could do worse than just crash, should we be very sanguine that for humans everything must "add up to moral normality"?

To expand a bit more on this possibility, many people have an aversion against moral arbitrariness, so we need at a minimum a utility translation scheme that's principled enough to pass that filter. But our existing world models are a hodgepodge put together by evolution so there may not be any such sufficiently principled scheme, which (if other approaches to solving moral philosophy also don't pan out) would leave us with legitimate feelings of "existential angst" and nihilism. One could perhaps still argue that any *current* such feelings are premature, but maybe some people have stronger intuitions than others that these problems are unsolvable?

Do we have any examples of humans successfully navigating an ontological crisis? The LessWrong Wiki mentions loss of faith in God:

> In the human context, a clear example of an ontological crisis is a believer's loss of faith in God. Their motivations and goals, coming from a very specific view of life suddenly become obsolete and maybe even nonsense in the face of this new configuration. The person will then experience a deep crisis and go through the psychological task of reconstructing its set of preferences according the new world view.

But I don't think loss of faith in God actually constitutes an ontological crisis, or if it does, certainly not a very severe one. An ontology consisting of Gods, Self, Other People, and Dumb Matter just isn't very different from one consisting of Self, Other People, and Dumb Matter (the latter could just be considered a special case of the former with quantity of Gods being 0), especially when you compare either ontology to one made of microscopic particles or even less familiar entities.

But to end on a more positive note, realizing that seemingly unrelated problems are actually instances of a more general problem gives some hope that by "going meta" we can find a solution to all of these problems at once. Maybe we can solve many ethical problems simultaneously by discovering some generic algorithm that can be used by an agent to transition from any ontology to another?

(Note that I'm not saying this *is* the right way to understand one's real preferences/morality, but just drawing attention to it as a possible alternative to other more "object level" or "purely philosophical" approaches. See also this previous discussion, which I recalled after writing most of the above.)

Two More Decision Theory Problems for Humans

(This post has been sitting in my drafts folder for 6 years. Not sure why I didn't make it public, but here it is now after some editing.)

There are two problems closely related to the Ontological Crisis in Humans. I'll call them the "Partial Utility Function Problem" and the "Decision Theory Upgrade Problem".

Partial Utility Function Problem

As I mentioned in a previous post, the only apparent utility function we have seems to be defined over an ontology very different from the fundamental ontology of the universe. But even on it's native domain, the utility function seems only partially defined. In other words, it will throw an error (i.e., say "I don't know") on some possible states of the heuristical model. For example, this happens for me when the number of people gets sufficiently large, like 3^^^3 in Eliezer's Torture vs Dust Specks scenario. When we try to compute the expected utility of some action, how should we deal with these "I don't know" values that come up?

(Note that I'm presenting a simplified version of the real problem we face, where in addition to "I don't know", our utility function could also return essentially random extrapolated values outside of the region where it gives sensible outputs.)

Decision Theory Upgrade Problem

In the Decision Theory Upgrade Problem, an agent decides that their current decision theory is inadequate in some way, and needs to be upgraded. (Note that the Ontological Crisis could be considered an instance of this more general problem.) The question is whether and how to transfer their values over to the new decision theory.

For example a human might be be running a mix of several decision theories: reinforcement learning, heuristical model-based consequentialism, identity-based decision making (where you adopt one or more social roles, like "environmentalist" or "academic" as part of your identity and then make decisions based on pattern matching what that role would do in any given situation), as well as virtual ethics and deontology. If you are tempted to drop one or more of these in favor of a more "advanced" or "rational" decision theory, such as UDT, you have to figure out how to transfer the values embodied in the old decision theory, which may not even be represented as any kind of utility function, over to the new.

Another instance of this problem can be seen in someone just wanting to be a bit more consequentialist. Maybe UDT is too strange and impractical, but our native model-based consequentialism at least seems closer to being rational than the other decision procedures we have. In this case we tend to assume that the consequentialist module already has our real values and we don't need to "port" values from the other decision procedures that we're deprecating. But I'm not entirely sure this is safe, since the step going from (for example) identity-based decision making to heuristical model-based consequentialism doesn't seem *that* different from the step between heuristical model-based consequentialism and something like UDT.

Using vector fields to visualise preferences and make them consistent

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

This post was written for Convergence Analysis by Michael Aird, based on ideas from Justin Shovelain and with ongoing guidance from him. Throughout the post, "I" will refer to Michael, while "we" will refer to Michael and Justin or to Convergence as an organisation.

Epistemic status: High confidence in the core ideas on an abstract level. Claims about the usefulness of those ideas, their practical implications, and how best to concretely/mathematically implement them are more speculative; one goal in writing this post is to receive feedback on those things. I'm quite new to many of the concepts covered in this post, but Justin is more familiar with them.

Overview

This post outlines:

- What vector fields are
- How they can be used to visualise preferences
- How utility functions can be generated from "preference vector fields" (PVFs)
- How PVFs can be extrapolated from limited data on preferences
- How to visualise inconsistent preferences (as "curl")
- A rough idea for how to "remove curl" to generate consistent utility functions
- Possible areas for future research

We expect this to provide useful tools and insights for various purposes, most notably AI alignment, existential risk strategy, and rationality.

This post is structured modularly; different sections may be of interest to different readers, and should be useful in isolation from the rest of the post. The post also includes links to articles and videos introducing relevant concepts, to make the post accessible to readers without relevant technical backgrounds.

Vector fields and preferences

A vector represents both magnitude and direction; for example, velocity is a vector that represents not just the speed at which one is travelling but also the direction of travel. A vector field essentially associates a vector to each point in a region of space. For example, the following image (source) shows the strength (represented by arrow lengths) and direction of the magnetic field at various points around a bar magnet:

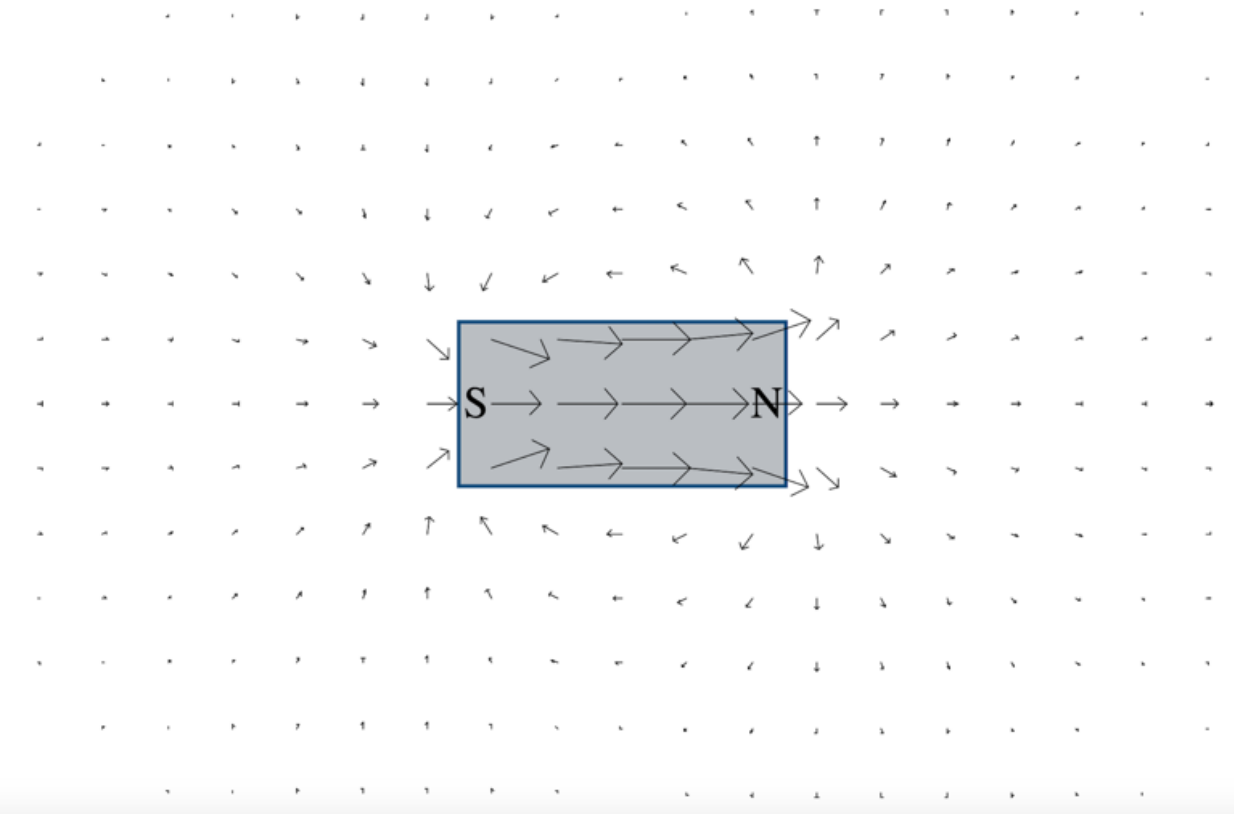


Figure 1.

Another common usage of vector fields is to represent the direction in which fluid would flow, for example the downhill flow of water on uneven terrain (this short video shows and discusses that visualisation).

We believe that vector fields over “state spaces” (possible states of the world, represented by positions along each dimension) can be a useful tool for analysis and communication of various issues (e.g., existential risk strategy, AI alignment). In particular, we’re interested in the idea of representing preferences as “preference vector fields” (PVFs), in which, at each point in the state space, a vector represents which direction in the state space an agent would prefer to move from there, and how intense that preference is.[1] (For the purposes of this post, “agent” could mean an AI, a human, a community, humanity as a whole, etc.)

To illustrate this, the following PVF shows a hypothetical agent’s preferences over a state space in which the only dimensions of interest are wealth and security.[2][3]

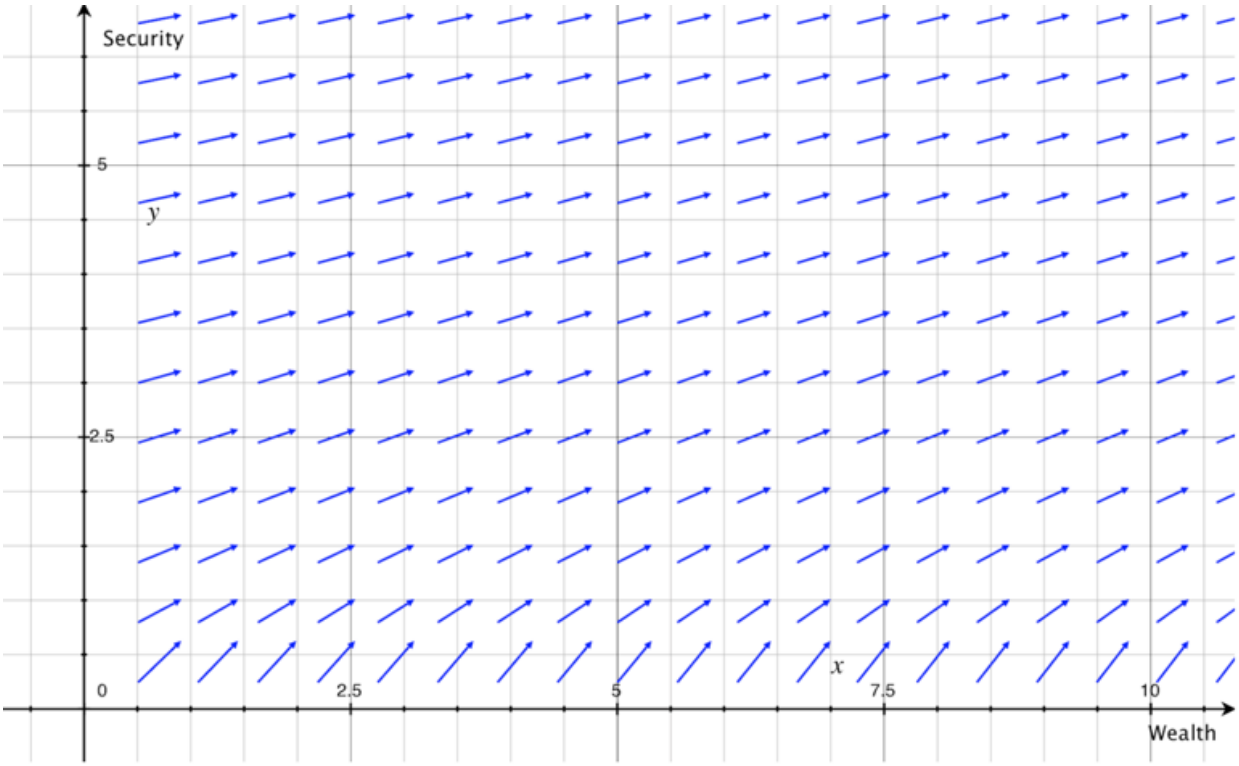


Figure 2.

The fact that (at least over the domain shown here) the arrows always point at least slightly upwards and to the right shows that the agent prefers more wealth and security to less, regardless of the current level of those variables. The fact that the arrows are longest near the x axis shows that preferences are most intense when security is low. The fact that the arrows become gradually more horizontal as we move up the y axis shows that, as security increases, the agent comes to care more about wealth relative to security.

Not only preferences

In a very similar way, vector fields can be used to represent things other than preferences. For example, we might suspect that for many agents (e.g., most/all humans), preferences do not perfectly match what would actually make the agent happier (e.g., because of the agent being mistaken about something, or having separate systems for reward vs motivation). In this case, we could create a vector field to represent the agent's preferences (represented by the blue arrows below), and another to represent what changes from any given point would increase the agent's happiness (represented by the green arrows).

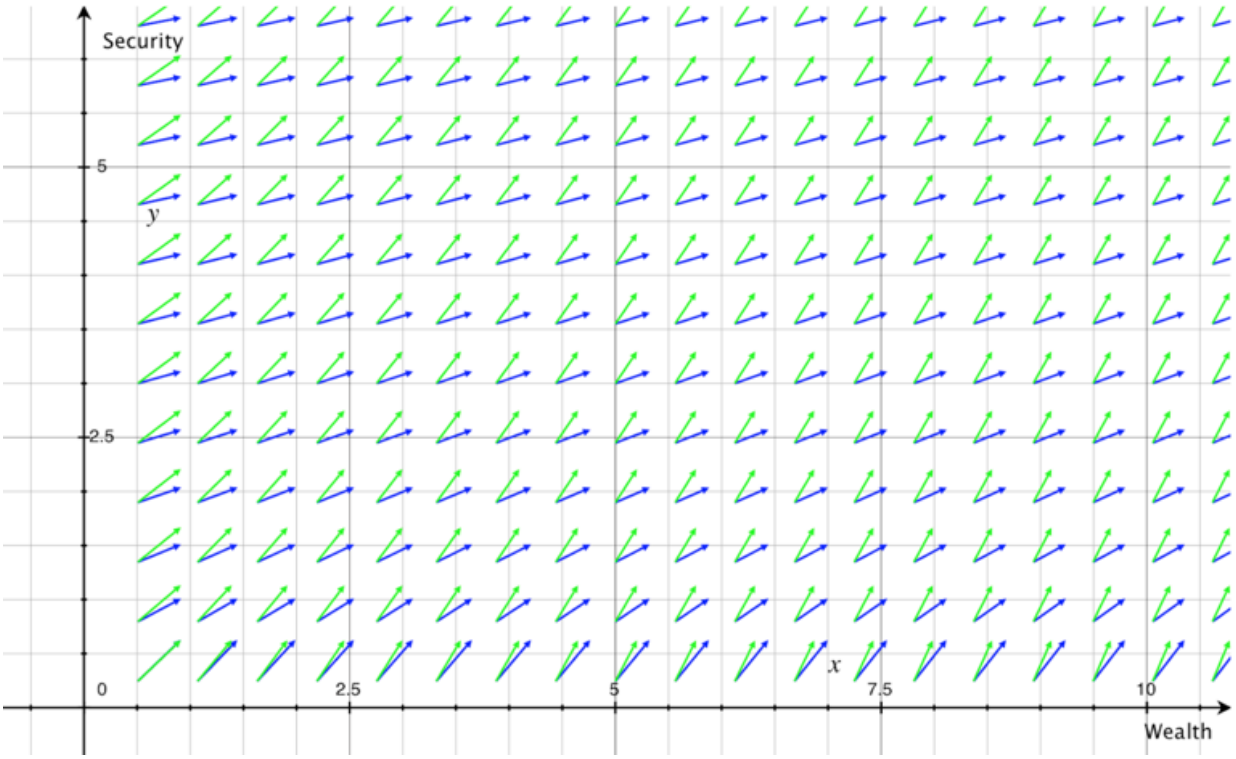


Figure 3.

This method of layering vector fields representing different things can be used as one tool in analysing potential clashes between different things (e.g., between an agent's preferences and what would actually make the agent happy, or between an agent's *beliefs* about what changes would be likely at each state and what changes would *actually* be likely at each state).

For example, the above graph indicates that, as wealth and/or security increases (i.e., as we move across the x axis and/or up the y axis), there is an increasing gap between the agent's preferences and what would make the agent happy. In particular, security becomes increasingly more important than wealth for the agent's happiness, but this is not reflected in the agent's preferences.

(Note that, while it does make sense to compare the direction in which arrows from two different vector fields point, I haven't yet thought much about whether it makes sense to compare the lengths Grapher shows for their arrows. It seems like this is *mathematically* the same as the common problem of trying to compare utility functions across different agents, or preferences across different voters. But here the functions represent different things within the *same* agent, which may make a difference.)

Gradients and utility functions

When a vector field has no "curl" (see the section "Curl and inconsistent preferences" below), the vector field can be thought of as the gradient of a scalar field.[4] (A scalar field is similar to a vector field, except that it associates a *scalar* with each point in a region of space, and scalars have only magnitude, rather than magnitude *and direction*.) Essentially, this means that the arrows of the vector field can be thought of

as pointing “uphill”, away from low points and towards high points of the associated scalar function. If the vector field represents preferences, higher points of the scalar function would be where preferences are more satisfied, and lower points are where it is less satisfied; thus, the scalar function can be thought of as the agent’s utility function.[5] (The same basic method is often used in physics, in which context the scalar function typically represents scalar potential.)

Below is one visualisation of the scalar field representing the utility function of the agent from the previous example (based on its preferences, not on what would make it “happy”), as well as the related vector field. Colours towards the red end of the spectrum represent higher values of the scalar field. It can be seen that the arrows of the vector field point away from blue areas and towards red areas, representing the agent’s preference for “climbing uphill” on its utility function.

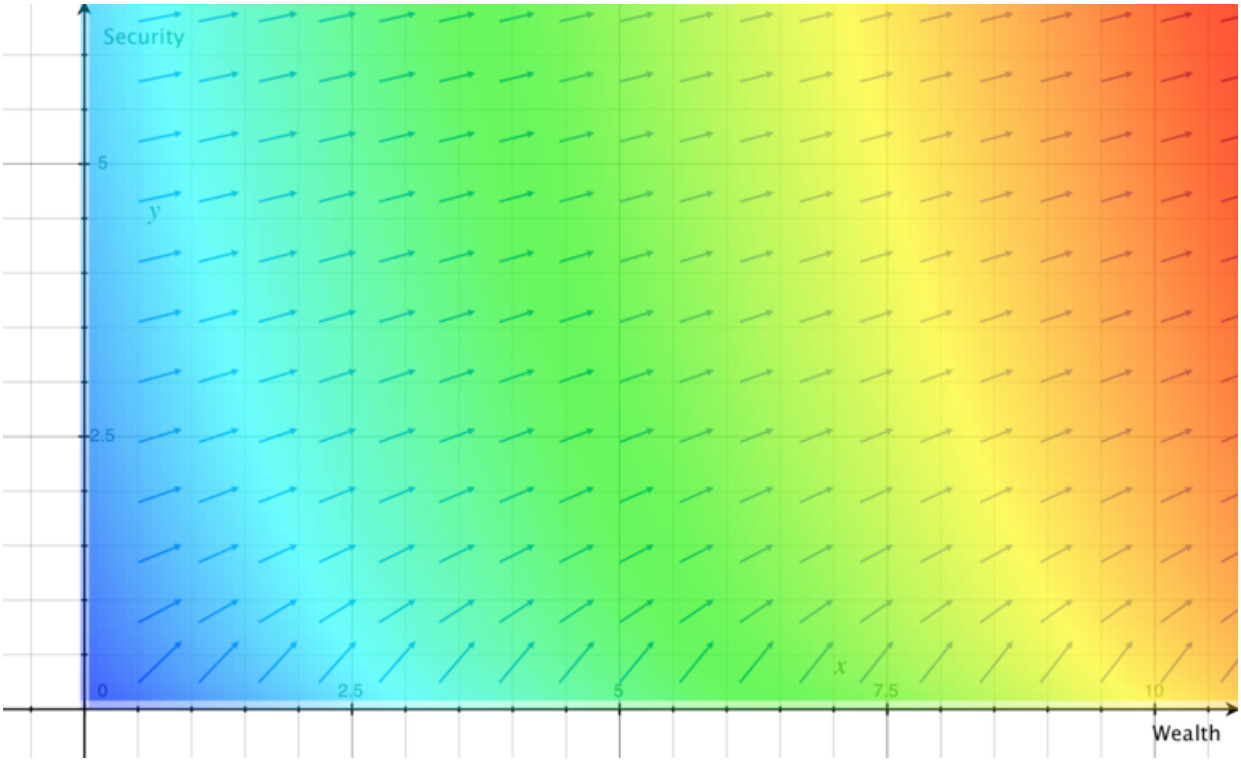
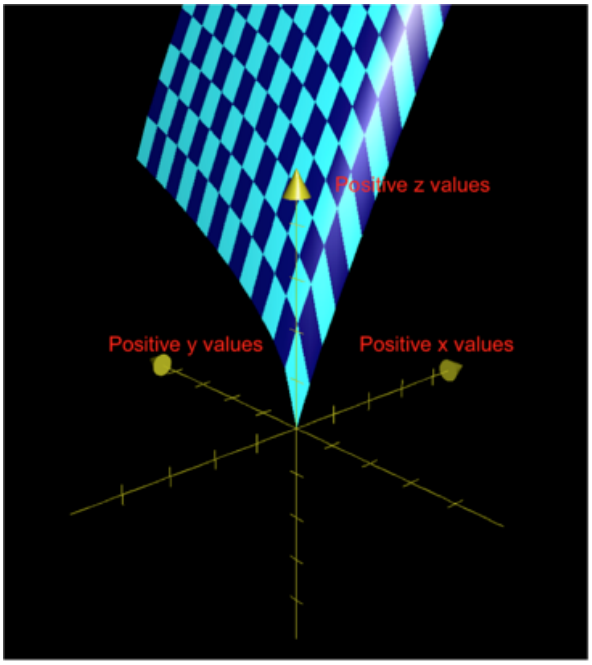
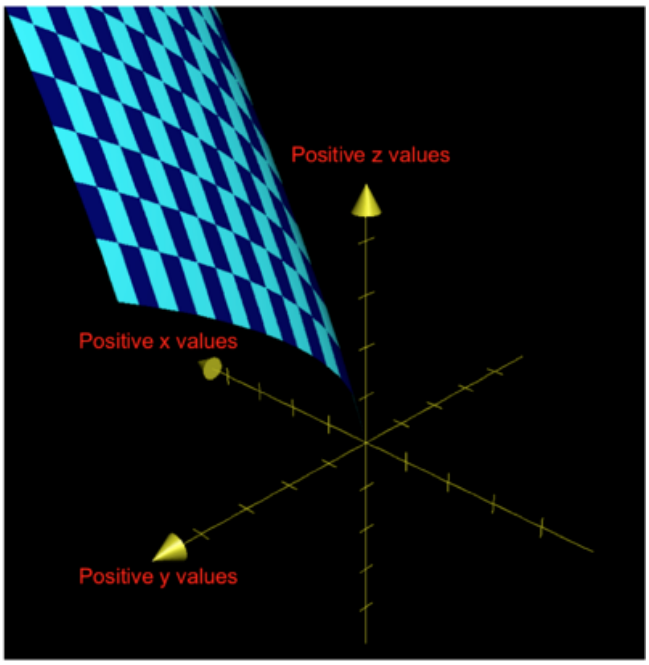


Figure 4.

The scalar field can also be represented in three dimensions, as values on the z dimension, which are in turn a function of values on the x and y dimensions. This is shown below (from two angles), for the same agent. (These graphs are a little hard to interpret from still images on a 2D screen, at least with this function; such graphs can be easier to interpret when one is able to rotate the angle of view.)





Figures 5a and 5b.

Method

This video provides one clear explanation of the actual method for determining the scalar function that a curl-free vector field can be thought of as the gradient of (though the video is focused on cases of 3D vector fields). That video describes this as finding the "potential"; as noted earlier, when the vector field represents preferences, the utility function can be thought of as analogous to the "potential" in other cases.

Personally, as a quick method of finding the scalar function associated with a 2D vector field, I used the following algorithm, from the first answer on this page:

> DSolve[{D[f[x, y], x] == [X COMPONENT OF THE VECTOR FIELD], D[f[x, y], y] == [Y COMPONENT OF THE VECTOR FIELD]}, f[x, y], {x, y}]

I input the algorithm into a Wolfram Cloud notebook, which seems to be free to use as long as you create an account. (As noted in the answer on the linked page, this algorithm will come back with no solution if the vector field has curl. This makes sense, because this general approach cannot be used in this way if a field has curl; this is explained in the section "Curl and inconsistent preferences" below.) Finally, I double-checked that the function was a valid solution by using this calculator to find its gradient, which should then be the same as the original vector field.

Extrapolating PVFs (and utility functions) from specific preference data

In reality, one rarely knows an agent's actual utility function or their full PVF. Instead, one is likely to only have data on the agent's (apparent) preferences at *particular points* in state space; for example, the extent to which they wanted more wealth and more security when they had $10,000 of savings and a "4/5" level of security.

One can imagine extrapolating a full preference vector field (PVF) from that data. We do not know of a precise method for actually doing this (we plan to do more research and thought regarding that in future). However, conceptually speaking, it seems the process would be analogous to fitting a regression line to observed data points, and, like that process, would require striking a balance between maximising fit with the data and avoiding overfitting.

For an example (based very loosely on Figure 3 in this article), suppose that I know that Alice prefers car A to Car B, Car B to Car C, Car C to Car D, and Car D to Car A (i.e., to Alice, A>B>C>D>A).[6] I also know the weight (in thousands of pounds) and perceived "sportiness" (as rated by consumers) of the four cars, and am willing to make the simplifying assumption that these are the only factors that influenced Alice's preferences. I could then create a plane with weight on the x axis and sportiness on the y axis, show the position of the four cars in this space, and represent Alice's preferences with arrows pointing from each car towards the car Alice would prefer to that one, as shown below:[7]

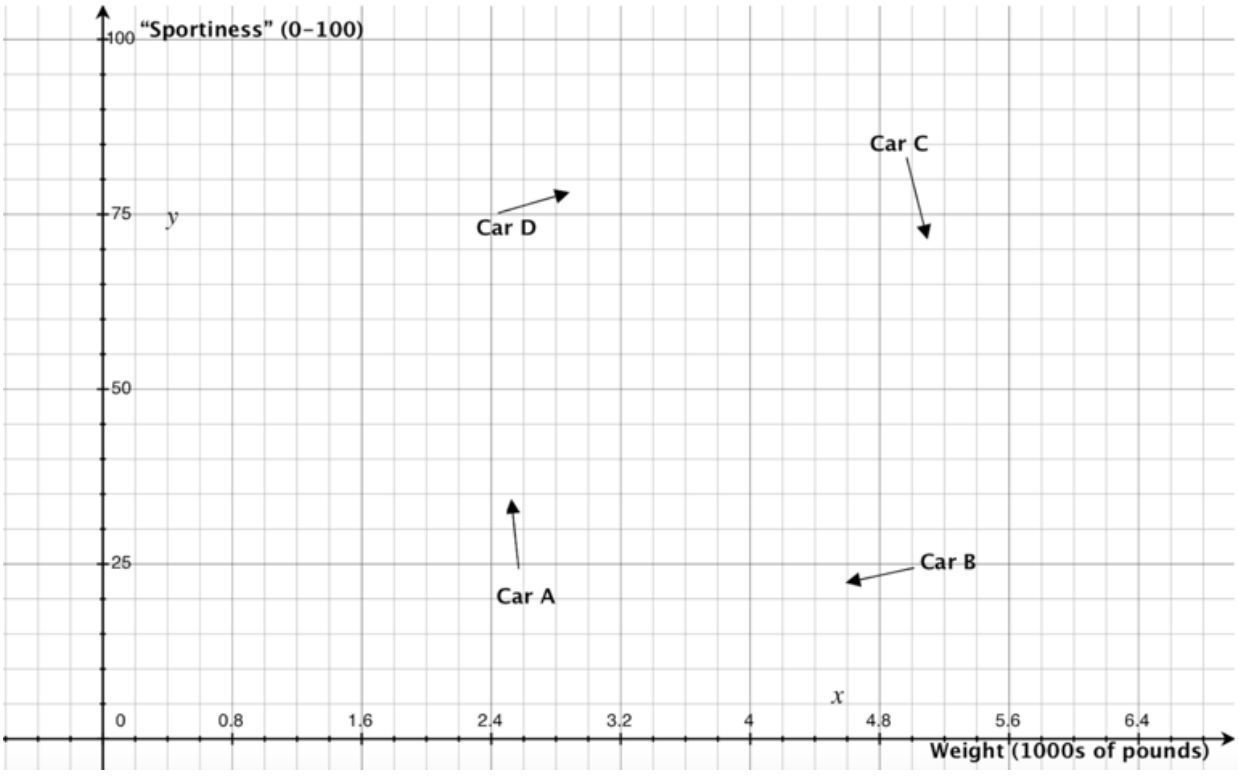


Figure 6.

I could then infer a PVF that (1) approximately captures Alice's known preferences, and (2) suggests what preferences Alice would have at any other point in the plane (rather than just at the four points I have data for). In this case, one seemingly plausible PVF is shown below, with the length of each blue arrow representing the strength of Alice's preferences at the associated point. (This PVF still shows Alice's known preferences,

but this is just for ease of comparison; those known preferences are not actually part of the PVF itself.)

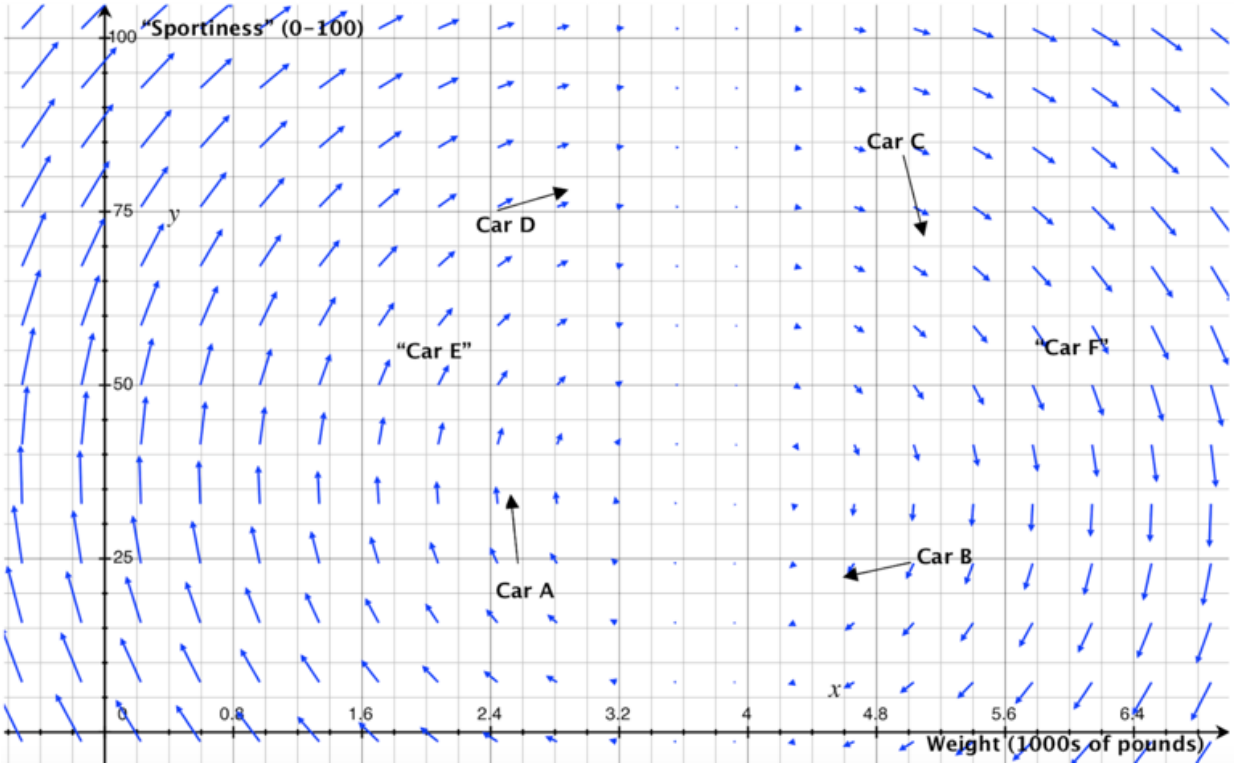


Figure 7.

This PVF allows us to make predictions about what Alice's preferences would be even in situations we do not have any empirical data about. For example, this PVF suggests that if Alice had the hypothetical car E (with a weight of ~2000 pounds and sportiness of ~55), she would prefer a car that was heavier and was higher for sportiness. In contrast, the PVF also suggests that, if she had the hypothetical car F (with a weight of ~6000 pounds and sportiness of ~55), she would prefer a car that was heavier and was rated *lower* for sportiness.

Of course, these predictions are not necessarily accurate. One could likely create many other PVFs that also "appear" to roughly fit Alice's known preferences, and these could lead to different predictions. This highlights why we wish to find a more precise/"rigorous" method to better accomplish the goal I have conceptually gestured at here.

It's also worth noting that one could extrapolate an agent's utility function from limited preference data by first using the method gestured at here and then using the method covered in the previous section. That is, one could gather some data on an agent's (apparent) preferences, extrapolate a PVF that "fits" that data, and then calculate what (set of) scalar function(s) that vector field is the gradient of. That scalar function would be the agent's extrapolated utility function.

However, as noted earlier, this method only works if the PVF has no "curl", so it would not work in the case of Alice's preferences about cars. I will now discuss what I mean by "curl", what implications curl has, and a rough idea for "removing" it.

Curl and inconsistent preferences

In the example above, to Alice, A>B>C>D>A. This is a case of intransitivity, or, less formally, circular or inconsistent preferences. This is typically seen as irrational, and as opening agents up to issues such as being "money pumped". It seems that Alice would be willing to just keep paying us to let her trade in one car for the one she preferred to that one, and do this *endlessly* - going around and around in a circle, yet feeling that her preferences are being continually satisfied.

So another pair of reasons why representing preferences as vector fields is helpful is that doing so allows inconsistencies in preferences:

1. to be directly seen (if they are sufficiently extreme)
2. to be calculated as the vector field's curl

This video introduces the concept of curl. Returning to the visualisation of vector fields as representing the direction in which water would flow over a certain domain, curl represents the speed and direction an object would spin if placed in the water. For example, if there is a strong clockwise curl at a certain point, a stick placed there would rotate clockwise; if there is no curl at a point, a stick placed there would not rotate (though it still may move in some direction, as represented by the vector field itself).

Note that the concepts of curl and inconsistency will also apply in less extreme cases (i.e., where an agent's preferences do not *only* "chase each other around in circles").

As noted earlier, when a vector field has curl, one cannot find its gradient. In our context, this seems logical; if an agent's preferences are inconsistent, it seems that the agent cannot have a true utility function, and that we can't assign any meaningful "height" to any point in the 2D state space. Consider again the example of Alice's preferences for cars; if we were to interpret meeting her preferences as moving "uphill" on a utility function, she could keep arriving back at the same points in the state space and yet be at different "heights", which doesn't seem to make sense.

Removing curl to create consistent utility functions

It seems that agents frequently have intransitive preferences, and thus that their PVFs will often have some curl. It would therefore be very useful to have a method for "removing curl" from a PVF, to translate an intransitive set of preferences into a transitive set of preferences, while making a minimum of changes. This new, consistent PVF would also then allow for the generation of a corresponding utility function for the agent.[8]

We believe that this process should be possible. We also believe that, if developed and confirmed to make sense, it could be useful for various aspects of AI alignment (among other things). In particular, it could help in:

- extrapolation of a consistent "core" (and corresponding utility function) from inconsistent *human* preferences (which could then inform an AI's decisions)
- adjustment of an *AI's* inconsistent preferences (either by engineers or by the AI itself), with a minimum of changes being made

We have not yet implemented this process for removing curl. But we believe that the Helmholtz theorem should work, at least for PVFs in 3 or fewer dimensions (and we believe that a higher dimensional generalization probably exists). The Helmholtz theorem:

> states that any sufficiently smooth, rapidly decaying vector field in three dimensions can be resolved into the sum of an irrotational (curl-free) vector field and a solenoidal (divergence-free) vector field; this is known as the Helmholtz decomposition or Helmholtz representation. (Wikipedia)

This irrotational (curl-free) vector field would then be the consistent projection (in a CEV-like way) of the agent's preferences (from which the agent's utility function could also be generated, in the manner discussed earlier).

Uncertainties and areas for further research

The following are some areas we are particularly interested in getting comments/feedback on, seeing others explore, or exploring ourselves in future work:

- Are there any flaws or misleading elements in the above analysis? (As noted earlier, this is essentially just an initial exploration of some tools/concepts.)
- To what extent do the methods used and claims made in this post generalise to higher-dimensional spaces (e.g., when we wish to represent preferences over more than two factors at the same time)? To what extent do they generalise to graphs of states that don't correspond to any normal geometry?
- Is there an existing, rigorous/precise method for extrapolating a PVF from a limited number of known preferences (or more generally, extrapolating a vector field from a limited number of known vectors)? If not, can a satisfactorily rigorous/precise method be developed?
- Are there meaningful and relevant differences between the concepts of curl in vector fields and of intransitivity, inconsistency, irrationality, and incoherence in preferences? If so, how does that change the above analysis?
- Is it possible to "remove curl" in the way we want, in the sort of situations we're interested in (in particular, not only in three dimensions)? If so, how, specifically?
- What other implications do the above ideas have? E.g., for rationality more generally, or for how to interpret and implement preference utilitarianism. (Above, I mostly just introduced the ideas, and hinted at a handful of implications.)
- What other uses could these "tools" be put to?

---

1. It appears some prior work (e.g., this and this) has explored the use of vector fields to represent preferences. Unfortunately, I haven't yet had time to investigate this work, so there may be many useful insights in there that are lacking in this post. ↩

2. Of course, there are often far more than two key factors influencing our preferences. In such cases, a vector field over more dimensions can be used instead (see here for an introduction to 3D vector fields). I focus in this post on 2D vector fields, simply because those are easier to discuss and visualise. We expect many of the ideas and implications covered in this post will be similar in

higher dimensional vector fields, but we aren't yet certain about that, and intend to more carefully consider it later. ↩

3. For both this example and most others shown, the precise equations used were chosen quite arbitrarily, basically by trying equations semi-randomly until I found one that roughly matched the sort of shape I wanted. For those interested, I have screenshots of all equations used, in their order of appearance in this post, here. To create the visuals in this post, I entered these equations into Grapher (for those interested in trying to do similar things themselves, I found this guide useful). I discuss below, in the section "Extrapolating PVFs (and utility functions) from specific preference data", the issue of how to actually generate realistic/accurate PVFs in the first place. ↩

4. It's possible that here I'm conflating the concepts of conservative, irrotational, and curl-free vector fields in a way that doesn't make sense. If any readers believe this is the case, and especially if they believe this issue changes the core ideas and implications raised in this post, I would appreciate them commenting or messaging me. ↩

5. Technically, the vector field is the gradient of a *class of* functions, with the functions differing only in their constant term. This is because gradient only relates to *differences* in height (or roughly analogous ideas, in higher-dimensional cases), not to absolute heights. One can imagine raising or lowering the entire scalar function by the same constant without affecting the gradient between points. (I show in this document examples of what this would look like, while in this post itself I keep all constants at 0.) Thus, in one sense, a PVF does not fully specify the associated utility function representation, but the constant can be ignored anyway (as utility functions are unique up to positive affine transformations). ↩

6. I have purposefully chosen a set of circular (or "intransitive") preferences, as the next session will use this example in discussing the problem of circularity and how to deal with it. ↩

7. Note that, in this example, I am not assuming any knowledge about the *strength* of Alice's preferences, only about their direction. As such, the length of the arrows representing Alice's known preferences has no particular meaning. ↩

8. In conversation with Justin, Linda Linsefors mentioned having had a somewhat similar idea independently. ↩

Inferring utility functions from locally non-transitive preferences

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.
This is a linkpost for https://universalprior.substack.com/p/inferring-utility-functions

As part of the AI Safety Camp, I've been diving a bit deeper into the foundations of expected utility theory and preference learning. In this post, I am making explicit a connection between those two things that (I assume) many people already made implicitly. But I couldn't find a nice exposition of this argument so I wrote it up. Any feedback is of course highly welcome!

Preference utilitarianism and the Von-Neumann-Morgenstern theorem.

At the risk of sounding drab, I briefly want to revisit some well-trodden territory in the realm of expected utility theory to motivate the rest of this post.

When we are thinking about the question of how to align advanced AI with human values, we are confronted with the question of whether we want to capture "How humans act" or "How humans *should* act". Both approaches have virtue, but if we're being ambitious[1] we probably want to aim for the latter. However, our understanding of "How humans *should* act" is still rather confused and ready-made solutions to "plug into" an AGI are not available.

Rather than tackling the entire problem at once, we might focus on one particularly well-formalized portion of ethics first. A possible answer to the question "How should I act?" comes from preference utilitarianism where we focus on satisfying everyone's *preferences* (as revealed by their choices). I say that this portion of ethics is well-formalized because, it turns out, if you are being reasonable about your preferences, we can represent them succinctly in a "utility function," $u(A) \in \mathbb{R}$. This utility function has the neat property that for two possible options, L and M, you *prefer* option M over option L iff $u(M) > u(L)$.

This is the famous "von-Neumann-Morgenstern theorem" which sits at the heart of an approach of "doing good better".

Oskar Morgenstern and John von Neumann

Now given that the utility function lives in the realm of mathematics, there is a seemingly natural strategy to use it to steer AI. Get everyone's utility functions, combine them into a target function, and then let the AI pick the actions that increase the target function the most. If we *have* to pick a single number to maximize, there is a case to be made that utility is one of the best single numbers we can hope for.

Sounds good. Where is the catch?

The futility of computing utility.

Let's start by trying to write down a utility function. **The proof of the von-Neumann-Morgenstern is *constructive***, i.e., it doesn't only guarantee the existence of a utility function, it also shows us the way to get

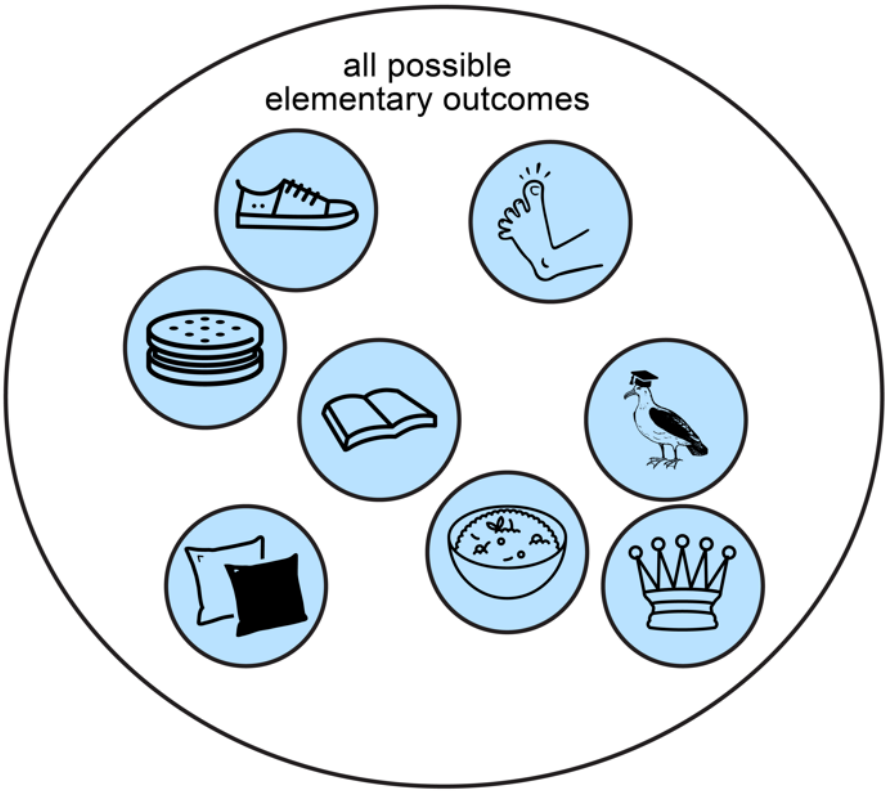
there. Here is a step-by-step guide:

1. Write down all the possible elementary outcomes that we might want to know the utility of.

Ah. Yeah. That's going to be a problem.

"All the things" is a lot of things. We might (and will, in this post) limit ourselves to a toy domain to make some progress, but that will be a poor substitute for the thing we want: all the possible outcomes affecting all existing humans. We might not think of some outcomes because they appear too good to be true. (Or too weird to be thinkable.) We might even want to include those outcomes *in particular,* as they might be the best option that nobody realized was on the table. But if we can't think of them, we can't write them down[2].

(Perhaps there is a way out. An essential nuance in the first step is writing down "all the possible *elementary* outcomes." We don't need to consider all the outcomes immediately. We only need to consider a set from which we can construct the more complicated outcomes. We need a basis of the space of possibilities. That's already infinitely easier, and we're *always guaranteed* to find a basis[3]. Hopefully, we can find a basis of a system that is rich enough to describe all relevant outcomes and simple enough to allow for linear interpolation? Of course, a semantic embedding of natural language comes to mind, but the imprecision of language is probably a deal-breaker. Perhaps the semantic embedding of a formal language/programming language is more appropriate[4]?)



A toy domain containing all the possible elementary outcomes one can possibly think about.

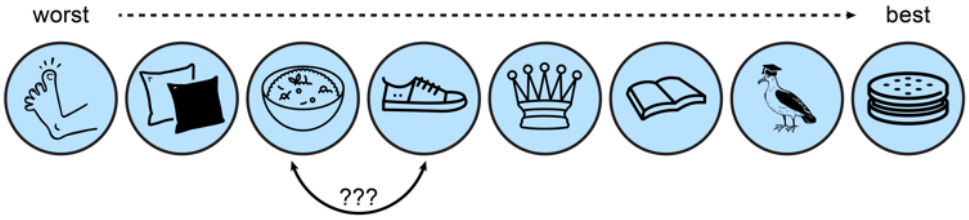
Let's use a toy domain, call this an open problem, and move on.

2. Order all the elementary outcomes from worst to best.

For some inexplicable reason, my intro to computer science course at uni has prepared me for this exact task, and this task alone. We have a bunch of great algorithms for sorting the elements of a list by comparing them with each other. If we're moderately unlucky, **we can hope to sort N-many outcomes with** $N \log N$ **comparisons**. After sorting thousands of elementary outcomes, we pick the worst (bumping your toe against a chair[5]) and the best (ice cream sandwich) and assign them utility 0 and 1. For 1000 elementary

outcomes, we could do with as little as 3000 comparisons which you could knock out in an afternoon. For 100000 elementary outcomes, we're talking 500000 comparisons which will keep you busy for a while. But it's for the survival of humankind, so perhaps it's fine! This could work!

Unless... somebody screws up. Those 3000 comparisons have to be *spot on*. **If you accidentally mess up one comparison, the sorting algorithm might not be able to recover**[6]**.** And since we're working with humans, some errors are guaranteed. Can you confidently say that you prefer a mushroom risotto over a new pair of shoes? Thought so.



New shoes sound fantastic, but that mushroom risotto looks de-li-cious!

But now comes the real killer.

3. Do a sequence of psychophysics experiments where humans indicate where the exact probabilistic combination of the worst- and the best possible outcome is equivalent to an intermediate outcome.

After sorting all outcomes from worst to best, we offer you a sequence of lotteries for each intermediate outcome (mushroom risotto): "Would you rather accept a 10% chance of stubbing your toe and a 90% chance of an ice cream sandwich, or a guaranteed mushroom risotto?" At some point (45% stubbing your toe), your answer will change from "Yes" to "No," and then we know we are very close to finding your utility for mushroom risotto (in this case, a utility of ~0.45).

I was halfway through setting up MTurk, but let's be realistic - this will not work. Allais paradox aside, I don't have the patience to set this up, so who will have the patience to go through this? Of course, we should have seen this coming. **Just because a proof is "constructive" doesn't mean that we can apply it in the messy real world.** Getting a utility function out of a human will require some elbow grease.

Human fallibility and reward modeling.

Let's shuffle some symbols. How do we account for all the human messiness and the tendency to make mistakes? A standard trick is to call it "noise"[7]. Given a *true* estimate of utility, $\bar{u}(A)$, we might write the perceived utility at any given time as a random variable, $u(A)$, distributed around the true value,

$$u(A) \sim N(\bar{u}(A), \sigma^2)$$

Given two outcomes, O_1 and O_2, the probability of assessing the utility of O_2 higher than the utility of O_1 is distributed as the difference of two Gaussians, which is again a Gaussian:

$$P(u(O_2) > u(O_1)) = \mathrm{erf}(\bar{u}(O_2) - \bar{u}(O_1))$$

(assuming independence of O_2 and O_1). The error function (lovingly called erf) is nasty because it doesn't have a closed-form solution. But it *does* have a close relative that looks almost the same and is a lot nicer

mathematically, and that has a much more pleasant-sounding name than "erf"...

I'll replace the erf with a stereotypical logistic function, S,

$$\text{erf}(\bar{u}(O_2) - \bar{u}(O_1)) \approx S(\bar{u}(O_2) - \bar{u}(O_1)) = 1/(1 + e^{-(\bar{u}(O_2) - \bar{u}(O_1))})$$

Much nicer. Even though the logistic function has largely fallen out of favor as an activation function in machine learning, its grip on psychophysics is unbroken.

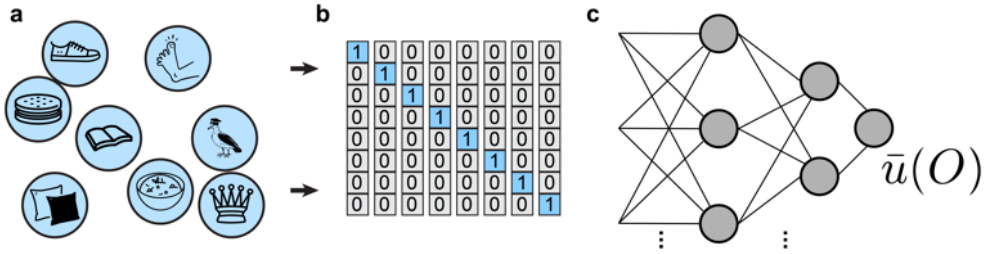
Now that we have the mathematical machinery in place, we need to calibrate it to reality. A natural choice is to take the cross-entropy between our machinery, $P(u(O_2) > u(O_1))$, and the *actual* comparisons provided by humans, $O_2 > O_1$ or $O_1 > O_2$,

$$L = -(O_2 > O_1) \times \log(P(u(O_2) > u(O_1))) - (O_1 > O_2) \times \log(P(u(O_1) > u(O_2)))$$

$$= \log(1 + e^{\bar{u}(O_{bad}) - \bar{u}(O_{good})})$$

Surprise[8]! The resulting loss function is also used for the reward modeling that I discussed in a previous post[9]. The researchers who originally proposed this technique for learning human preferences say that it is similar to the Elo rating system from chess and the "Bradley & Terry model." I find the motivation of reward modeling as an approximation of the von-Neumann-Morgenstern theorem a lot more Romantic, though.

Having uncovered this connection, a natural strategy for inferring a utility function through training a neural network with comparisons of pairs of elements from the domain presents itself.

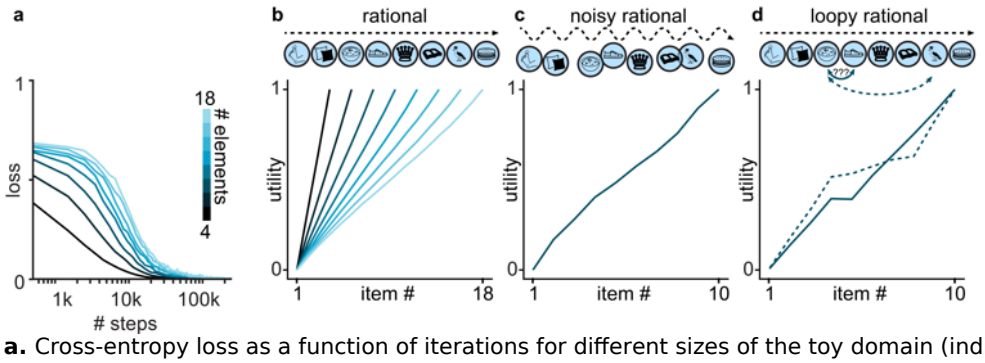


Unordered elements from the domain of “things we might care about” (**a**) are translated into an orthonormal basis of a high dimensional vector space (**b**) and transformed into a scalar output through a multilayer perception (**c**).

Can this work? It doesn’t involve MTurk for now, so I am happy to try!

A natural representation of utility functions.

I found that the neural network achieves near-zero loss on the comparisons after 20k steps (panel **a**). Runtime appears to increase linearly with the number of elements[10]. The resulting utility functions are monotonic and increase by an (approximately) equal amount from one item to another (panel **b**). This demonstrates that given enough time, a multilayer perceptron can sort a list.



a. Cross-entropy loss as a function of iterations for different sizes of the toy domain (indicated by color). **b.** Predicted utility of different items after 250k iterations of a “rational" (i.e., a total order) teaching signal. **c.** Predicted utility of items after 250k iterations of a “noisy rational” (i.e., 90% chance of total order, 10% chance of inverting preference) teaching signal. **d.** Predicted utility of items after 250k iterations of a “loopy rational” (i.e., total order, except for elements 3 and 4, or 3 and 7, which are connected in a loop) teaching signal. In **b** to **c**, the predicted utility is normalized between 0 and 1.

Some might say that I used several hours of compute time on a Google Colab GPU *just* to sort lists, but that would be rather uncharitable. My primary motivation for this approach (human tendency to make mistakes) bears fruits in the following experiment. When I add noise to the choice procedure (resulting in 10% “random” choices), the network is still able to recover the appropriate ordering (panel **c**). And, even more remarkable, **when I make the choice procedure loopy (i.e., nontransitive), the network can still recover a reasonable approximation of what the utility function looks like (panel d)!**

This last set of experiments is exciting because introducing loops leads to nonlinear utility functions that are squashed together in the vicinity of the loop. Intuitively, if outcomes #3 and #4 are impossible to distinguish reliably, this might indicate that their utility is indeed very similar. The exciting possibility is that step 3 of the procedure above (psychometric calibration of utilities) could automatically be satisfied when options are sufficiently similar and sometimes confused[11].

Concluding thoughts and what’s next?

As mentioned at the beginning of this post, I suspect that this way of interpreting preference learning was already clear to some, but I hope that making the connection explicit is useful. I also find it great to see that **we can recover a consistent utility function from inconsistent preferences** (panel d of last figure). There are a lot more experiments I want to run on this toy domain to probe the limits to which preference orderings can be turned into utility functions:

- What is the largest number of elements we can sort with a given architecture? How does training time change as a function of the number of elements?
- How does the network architecture affect the resulting utility function? How do the maximum and minimum of the unnormalized utility function change?
- Which portion of possible comparisons needs to be presented (on average) to infer the utility function?
- How far can we degenerate a preference ordering until no consistent utility function can be inferred anymore?

But independent of those experiments, there are some fascinating directions that I plan to explore in a future post. Now that I have a natural way to induce utility functions, I think I can further explore the utility monster and some of the classic literature on (un-)comparability of utility functions. I also really want to write a proper treatment of value handshakes, which is a topic in dire need of exploration.

I'd be curious to hear any additional ideas for what to try, as well as comments on how feasible this approach sounds/where it's likely to break down.

1. ^

   Some argue that narrow solutions are not stable in any case.

2. ^

   I suspect that if Goodhart creeps into the argument, this is around the place where it happens.

3. ^

if we're willing to embrace Zorn.

4. ^

   Has anybody ever looked at that? What happens when I subtract Fizzbuzz from the Fibonacci sequence?

5. ^

   A very close "win" over S-risk scenarios.

6. ^

   And the worst-case runtime shoots up to ∞ as soon as you have the probability of errors). In this situation, Bogosort might just be the most reliable solution - at least it won't terminate until it's done.

7. ^

   Joke aside, it's an interesting question which factors need to come together to make something "noise." I guess the central limit theorem can help if there are many independent factors, but that's never really the case. The more general question is under which circumstances the residual factors are random and when they are adversarial.

8. ^

   Please ignore that it's already written in the section title.

9. ^

   The original Leike paper has an equivalent expression that does not write out the logarithm applied to the sigmoid.

10. ^

    My money would be on $O(n \log n)$ or worse, of course.

11. ^

    I'd expect that the difference in utility, $|\bar{u}(O_1) - \bar{u}(O_2)|$, will be proportional to the probability of confusing the order, $P(u(O_1) > u(O_2))$.

Turning Some Inconsistent Preferences into Consistent Ones

cross-posted from niplav.github.io

Epistemic Status

This is still a draft that I was told to already post here, which includes working (but very slow) code for one special case. Hopefully I'll be able to expand on this in the next ~half year.

> Representing inconsistent preferences with specific mathematical structures can clarify thoughts about how to make those preferences consistent while only minimally changing them. This is discussed in the case of preferences over world states, represented by directed graphs; and preferences over lotteries of world states, represented either by infinitely dense graphs, (in some cases) vector fields over probability simplices, or edge-weighted directed graphs. I also present an algorithm for the discrete case based on the graph edit distance. Implications for scenarios such as ontological shifts are discussed.

Turning Some Inconsistent Preferences into Consistent Ones

> A kind of God-made (or evolution-created) fairness between species is also unexpectedly found.

— Yew-Kwang Ng, "Towards Welfare Biology: Evolutionary Economics of Animal Consciousness and Suffering" p. 1, 1995

> Random testing is simple in concept, often easy to implement, has been demonstrated to effectively detect failures, is good at exercising systems in unexpected ways (which may not occur to a human tester), and may be the only practical choice when the source code and the specifications are unavailable or incomplete.

— Tsong Yueh Chen/Fei-Ching Kuo/Robert G. Merkel/T.H. Tse, "Adaptive Random Testing: the ART of Test Case Diversity", 2010

Consider an agent which displays (von Neumman-Morgenstern) inconsistent preferences, for example choosing two incompatible options in the two scenarios in the Allais paradox, or reliably displaying cycles in its actions (detecting which actions are in fact caused by inconsistent preferences, and not just exotic ones from weird abstractions, is considered a separate problem here). We might want to interact with that agent, e.g. trade with it, help it (or exploit it), or generally know how it will act But how to go about that if the agent displays inconsistent preferences? Perhaps it might even be the case that humans are such agents, and find ourselves in a conundrum: we know our preferences are inconsistent and reliably exploitable, and that agents with such preferences reliably fare worse in the world, we might want to change that.

A possible approach to this problem has two steps:

1. Find ways to represent inconsistent preferences with a mathematical structure which can encode all possible violations of the von Neumann-Morgenstern axioms in all their combinations.
2. Then turn those inconsistent preferences into consistent ones, and then inform the agent about these inconsistencies and their optimal resolutions (or, in the case of trying to help the agent, then enacting these preferences in the real world).

Mathematical Formulation of the Problem

Define a set of possible (von Neumann-Morgenstern) inconsistent preferences over a set W of worlds as $\aleph$, and the set of consistent preferences over those worlds as $\daleth$.

Elements from those sets are written as $\succsim \in \aleph$ and $\succeq \in \daleth$.

One way we could approach the problem is by trying to turn those inconsistent preferences consistent, i.e. constructing a function $t : \aleph \mapsto \daleth$ that takes an inconsistent preference $\succsim$ and transforms it into a consistent preference $\succeq$, while retaining as much of the original structure of the preference as possible (it would make little sense if we replaced the original preference relation with e.g. indifference over all options).

Formally, we want to find for some given distance metric $d : \aleph \times \daleth \mapsto R$ a function t so that

$$t = \underset{t}{\operatorname{argmin}}\, d(\succsim, t(\succsim))$$

$$\succeq = t(\succsim)$$

I call this function a **turner**, and sometimes call the results of that function the **set of turnings** (an element from that set is a **turning**). The names mostly chosen for not having been used yet in mathematics, as far as I know, and because I want to be a little extra.

A solution to the problem of turning inconsistent preferences into consistent ones then has these components:

1. A mathematical structure for representing $\aleph$ and $\daleth$
    - Inconsistent preferences over discrete options are represented via directed graphs
    - Inconsistent preferences over lotteries of options are represented via
        - directed graphs over probability simplices
            - potentially more exotic structures such as graphons or results from extremal graph theory might be relevant here, but I haven't investigated these in detail

        - vector fields on probability simplices
        - graphs with edge weights in R
2. A specification for t
    - In the case of discrete options, I propose adding and removing edges from the directed graph
    - In the case of lotteries I don't have yet any clear proposals
3. A specification for d
    - In the case of discrete options, I propose using the graph edit distance
    - In the case of lotteries I don't yet have any definite proposals

Related Work

This work is closely related to the investigations in Aird & Shovelain 2020 (so closely that even though I believe I re-invented the approach independently, it might just be that I had read their work & simply forgotten it), and broadly related to the value extrapolation framework outlined in Armstrong 2022.

Discrete Case

When we have discrete sets of worlds W, we can represent an inconsistent preference over those worlds by using a directed graph $G_{\gtrsim} = (W, E_{\gtrsim} \subseteq W \times W)$. The presence of an edge (w_1, w_2) would mean that $w_1 \gtrsim w_2$, that is w_1 is preferred to w_2.

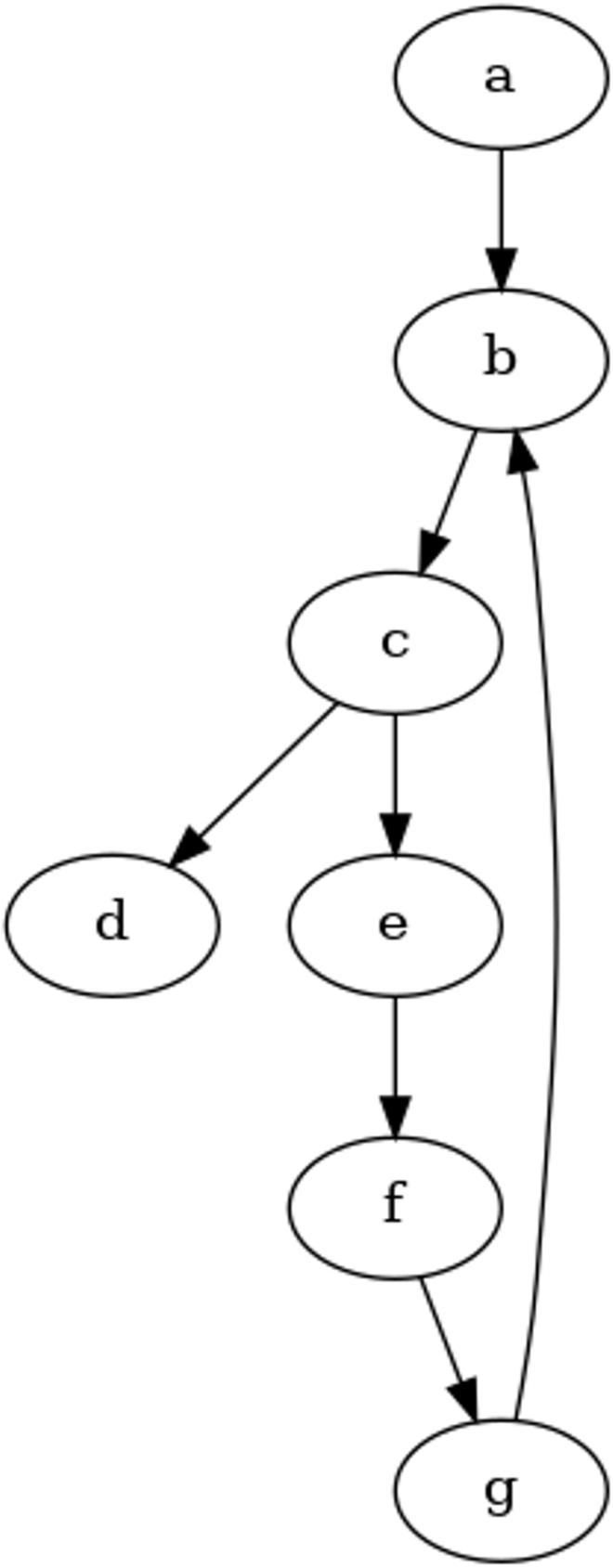
Mathematically, then, $\nleftrightarrow$ is the set of all possible graphs with edges in $W \times W$, that is $\nleftrightarrow = \{(W, E) | E \in P(W \times W))\}$).

The consistent equivalent to an inconsistent preference represented by a directed graph would be a path graph $G_{\geq} = (V, E_{\geq})$ over the same set of vertices W. The method for transforming $G_{\gtrsim}$ into $G_{\geq}$ would be by adding/deleting the minimal number of vertices from $E_{\gtrsim}$.

Mathematically, then ν is the set of transitive closures of all possible path graphs that are encode permutations of W; $\nu = \{(V, E)^{+} | E \in \sigma(W)\}$.

Example

Consider the following directed graph:


a
b
c
d
e
f
g

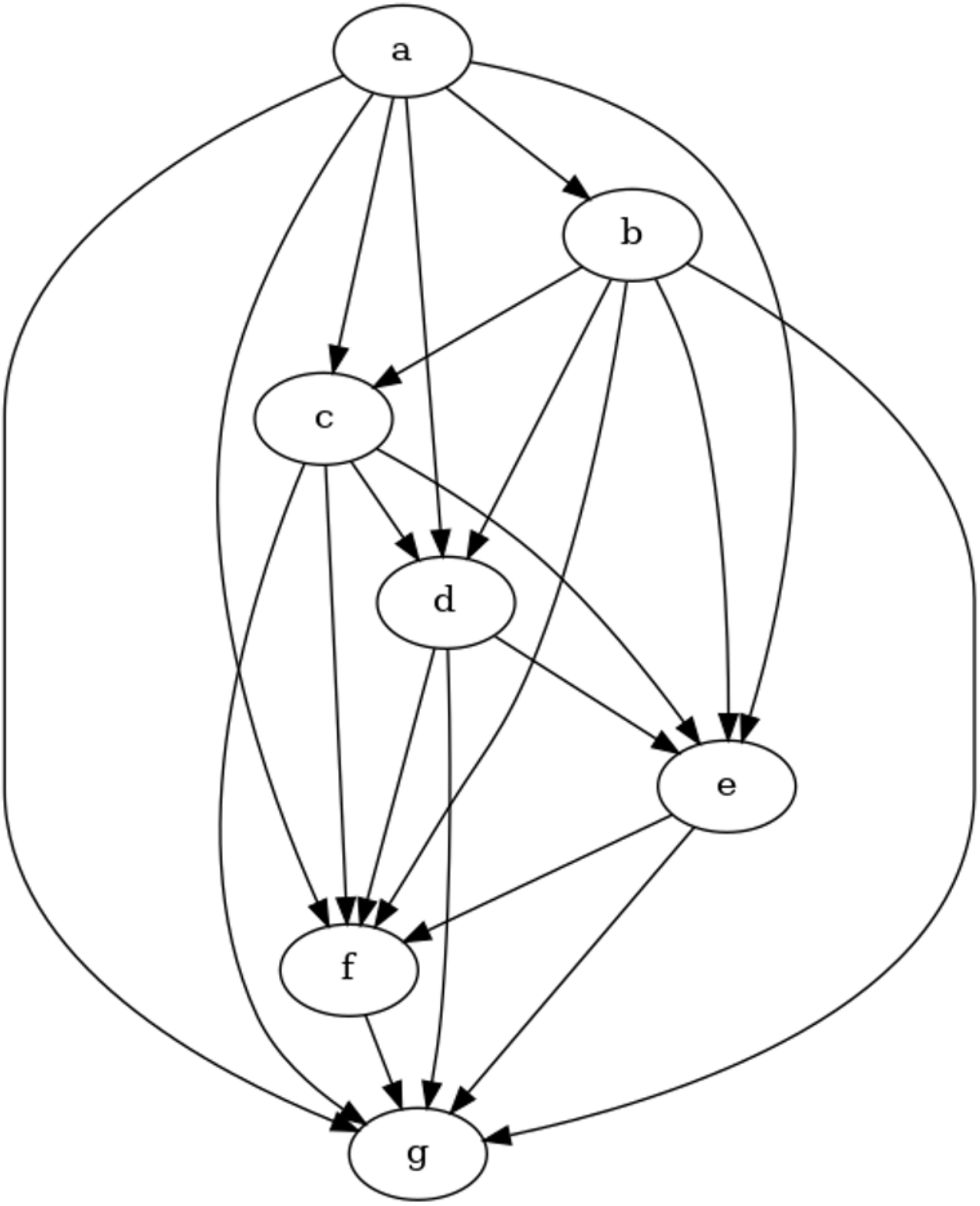
Here, $W = \{a, b, c, d, e, f, g\}$.

An edge from a to b means that a is preferred to b (short $a \gtrsim b$). The absence of an edge between two options means that those two options are, from the view of the agent, incomparable.

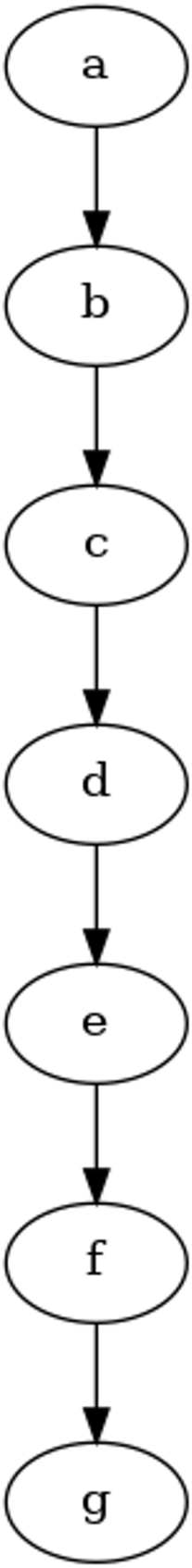
It violates the two von Neumann-Morgenstern axioms for discrete options:

- Completeness is violated because for example options d and e are incomparable (and we don't merely have indifference between these options)
- Transitivity is violated because of the $b \to c \to e \to f \to g \to b$ loop

A possible turned version of these preferences could then be the following graph:



This graph looks quite messy, but it's really just the transitive closure of this graph:


a
b
c
d
e
f
g

Whether this is the "right" way to turn the previous inconsistent preferences depends on the choice of distance metric we would like to use.

Resolving Inconsistencies

In some sense, we want to change the inconsistent preferences as little as possible; the more we modify them, the more displayed preferences we have to remove or change. Since the presence or absence of preferences is encoded by the presence or absence of edges on the graph, removing edges or adding new edges is equivalent to removing or adding preferences (at the moment, we do *not* consider adding or removing vertices: we stay firmly inside the agent's ontology/world model).

Luckily, there is a concept in computer science called the graph-edit distance: a measure for the difference between two graphs.

The set of possible editing operations on the graph varies, e.g. Wikipedia lists

- **vertex insertion** to introduce a single new labeled vertex to a graph.
- **vertex deletion** to remove a single (often disconnected) vertex from a graph.
- **vertex substitution** to change the label (or color) of a given vertex.
- **edge insertion** to introduce a new colored edge between a pair of vertices.
- **edge deletion** to remove a single edge between a pair of vertices.
- **edge substitution** to change the label (or color) of a given edge.

—*English Wikipedia, "Graph Edit Distance", 2021*

Since we do not have labels on the edges of the graph, and have disallowed the deletion or insertion of vertices, this leaves us with the graph edit distance that uses edge insertion and edge deletion.

We can then write a simple pseudocode algorithm for $\succeq = f(\gtrsim)$:

```
turn(G≿=(W, E≿)):
        mindist=∞
        for L in perm(W):
                L=trans_closure(L)
                dist=ged(G≿, R)
                if dist<mindist:
                        R=L
                        mindist=dist
        return R
```

where `perm(W)` is the set of permutations on `W`, `trans_closure(G)` is the transitive closure of a graph `G`, and `ged(G1, G2)` is the graph edit distance from `G1` to `G2`.

Or, mathematically,

$$R = \operatorname*{argmin}_{R \in \sigma(W)} \mathrm{GED}(R^{+}, G_{\gtrsim}))$$

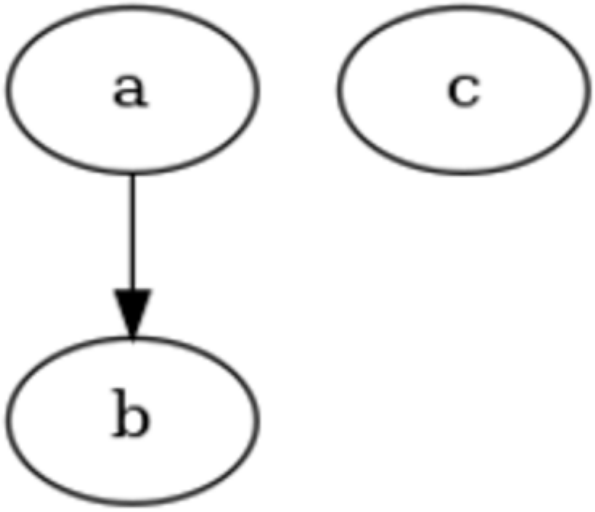
Implementation

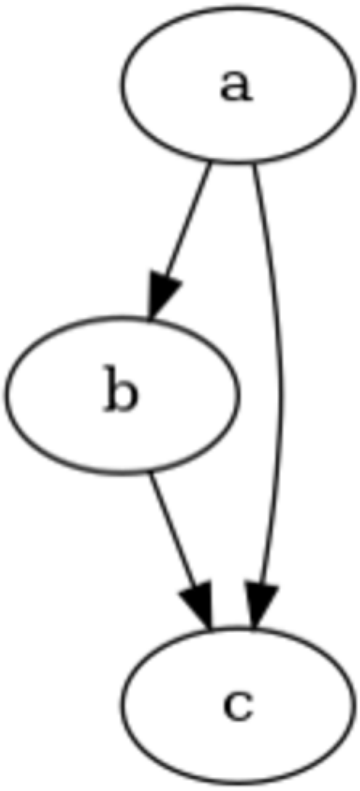
Implementing this in Python 3 using the networkx library turns out to be easy:

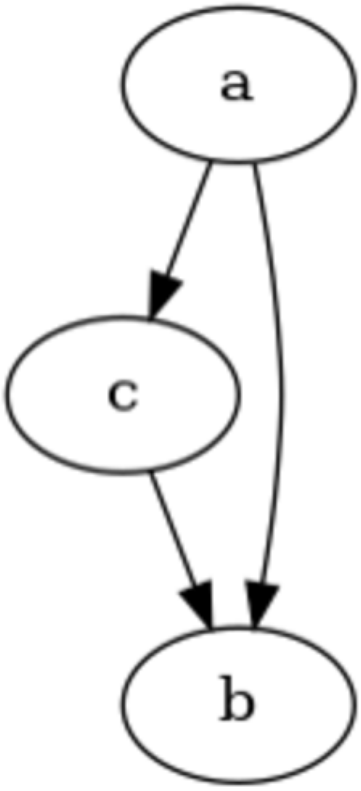
```
import math
import networkx as nx
import itertools as it

def turn(graph):
mindist=math.inf
        worlds=list(graph.nodes)
        for perm in it.permutations(worlds):
                perm=list(perm)
                pathgraph=nx.DiGraph()
                for i in range(0, len(worlds)):
                        pathgraph.add_node(worlds[i], ind=i)
                # The transitive closure over this particular path graph
                # Simplify to nx.algorithms
                for i in range(0, len(perm)-1):
                        pathgraph.add_edge(perm[i], perm[i+1])
                pathgraph=nx.algorithms.dag.transitive_closure(pathgraph)
                # Compute the graph edit distance, disabling node
insertion/deletion/substition and edge substitution
                edge_cost=lambda x: 1
                unaffordable=lambda x: 10e10
                same_node=lambda x, y: x['ind']==y['ind']
                edge_matches=lambda x, y: True
                dist=nx.algorithms.similarity.graph_edit_distance(graph, pathgraph,
node_match=same_node, edge_match=edge_matches, node_del_cost=unaffordable,
node_ins_cost=unaffordable, edge_ins_cost=edge_cost, edge_del_cost=edge_cost)
                if dist<mindist:
                        result=pathgraph
                        mindist=dist
        return result
```

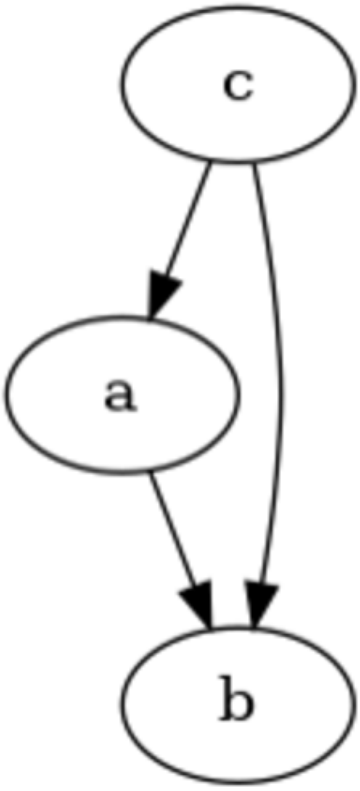
We can then test the function, first with a graph with a known best completion, and then with our example from above.

The small example graph (top left) and its possible turnings are (all others):









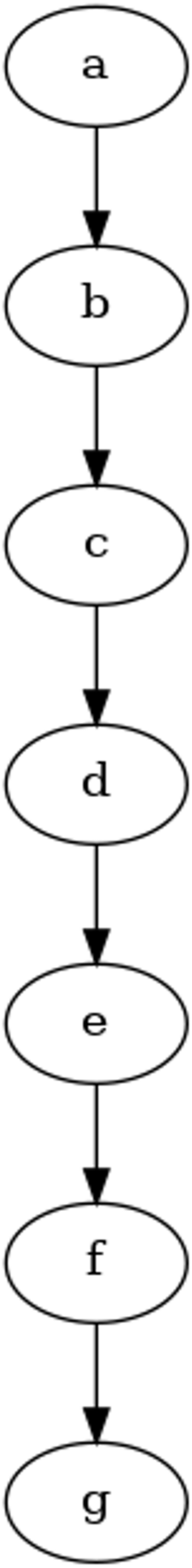

```
>>> smallworld=['a', 'b', 'c']
>>> smallgraph=nx.DiGraph()
```

```
>>> for i in range(0, len(smallworld)):
...     smallgraph.add_node(smallworld[i], ind=i)
>>> smallgraph.add_edges_from([('a', 'b')])
>>> smallre=turn(smallworld, smallgraph)
>>> smallre.nodes
NodeView(('a', 'b', 'c'))
>>> smallre.edges
OutEdgeView([('a', 'b'), ('a', 'c'), ('b', 'c')])
```

This looks pretty much correct.

```
>>> mediumworld=['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> mediumgraph=nx.DiGraph()
>>> for i in range(0, len(mediumworld)):
...     mediumgraph.add_node(mediumworld[i], ind=i)
>>> mediumgraph.add_edges_from([('a', 'b'), ('b', 'c'), ('c', 'd'), ('c', 'e'), ('e',
'f'), ('f', 'g'), ('g', 'b')])
>>> mediumres=turn(mediumworld, mediumgraph)
>>> mediumres.nodes
NodeView(('a', 'b', 'c', 'd', 'e', 'f', 'g'))
>>> mediumres.edges
OutEdgeView([('a', 'b'), ('a', 'c'), ('a', 'd'), ('a', 'e'), ('a', 'f'), ('a', 'g'),
('b', 'c'), ('b', 'd'), ('b', 'e'), ('b', 'f'), ('b', 'g'), ('c', 'd'), ('c', 'e'),
('c', 'f'), ('c', 'g'), ('d', 'e'), ('d', 'f'), ('d', 'g'), ('e', 'f'), ('e', 'g'),
('f', 'g')])
```

This is actually equal to the hypothesized solution from above (below is the non-transitive-closure version):


a
b
c
d
e
f
g

Problems with This Method and its Algorithm

This solution has some glaring problems.

Speed (or the Lack Thereof)

Some of you might have noticed that this algorithm is *somewhat inefficient* (by which I mean *absolutely infeasible*).

Since we iterate through the permutations of W, the runtime is O(|W|!) (with the added "benefit" of additionally computing the NP-complete graph edit distance inside of the loop, which is also APX-hard to approximate).

Possible better approaches would involve finding the longest subgraph that is a path graph, or the spanning tree, perhaps the transitive reduction is helpful, or maybe the feedback arc set?

Non-Unique Results

Another, smaller problem is that the algorithm often doesn't have a unique result, as seen in the small example above.

We can compute the set of all possible turnings with some trivial changes to the algorithm:

```
turn_all(G≥=(W, E≥)):
        mindist=∞
        R=∅
        […]
                if dist<mindist:
                        R={L}
                        mindist=dist
                else if dist==mindist:
                        R=R∪{L}
        return R
```

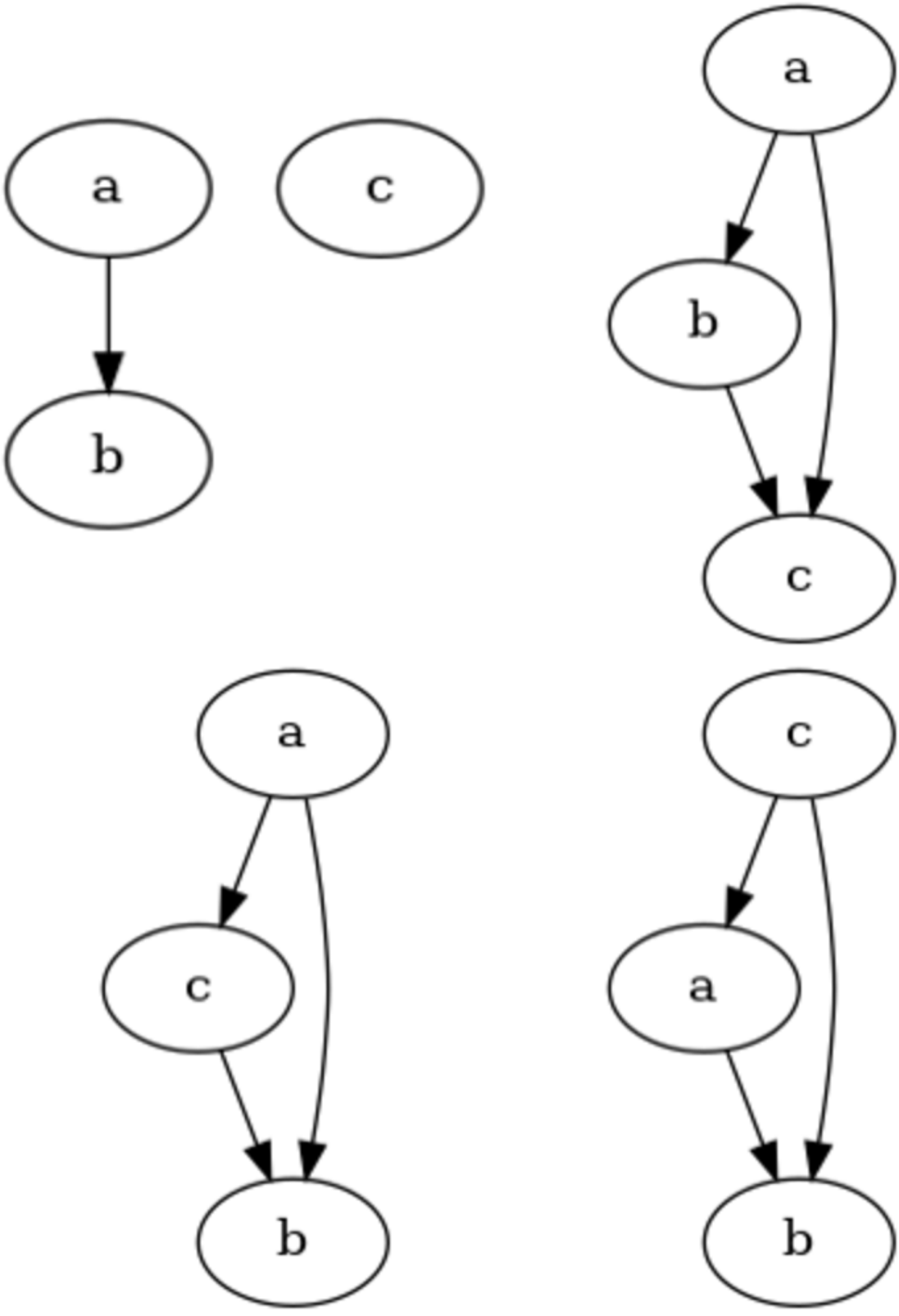
and its implementation

```
def turn_all(graph):
        results=set()
        […]
                if dist<mindist:
                        results=set([pathgraph])
                        mindist=dist
                elif dist==mindist:
                        results.add(pathgraph)
        return results
```

The results, with the small example, are as expected:

```
>>> turnings=list(turn_all(smallworld, smallgraph))
>>> len(turnings)
3
>>> turnings[0].edges
OutEdgeView([('a', 'b'), ('a', 'c'), ('b', 'c')])
>>> turnings[1].edges
```

```
OutEdgeView([('a', 'b'), ('c', 'a'), ('c', 'b')])
>>> turnings[2].edges
OutEdgeView([('a', 'c'), ('a', 'b'), ('c', 'b')])
```



For the big example, after waiting a while for the solution:

```
>>> turnings=list(turn_all(mediumworld, mediumgraph))
>>> len(turnings)
49
```

I will not list them all, but these are less than the 7! = 5040 possible options.

This brings up an interesting question: As we have more and more elaborate inconsistent preferences over more worlds, does it become more likely that they have a unique consistent preference they can be turned to? Or, in other words, if we make the graphs bigger and bigger, can we expect the fraction of inconsistent preferences with a unique turning to grow or shrink (strictly) monotonically? Or will it just oscillate around wildly?

More formally, if we define G_n as the set of graphs with n nodes, and

$U_n = \{G \in G_n | 1 = |\text{turn\_all}(G)|\}$ as the set of graphs with n nodes that have unique path graphs associated with them.

We can further define the set of all graphs wwith n nodes with m turnings as

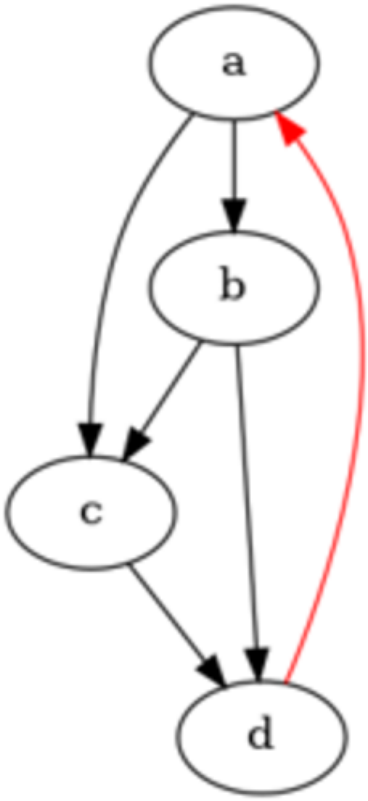
$T\,n, m = \{G \in G_n | m = |\text{turn\_all}(G)|\}$ (of which $U_n = T_{n,1}$ is just a special case).

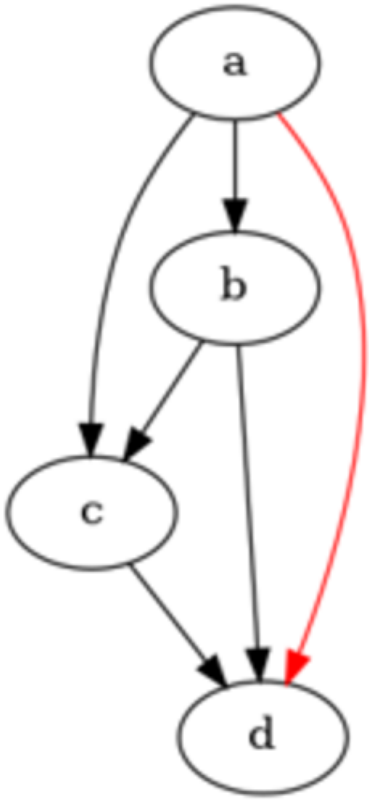
We can call the size of the set of all turnings of a graph the **confusion** of that graph/set of inconsistent preferences: If the graph is already the transitive closure of a path graph, the size of that set is (arguendo) 1: there are no other possible turnings. If the graph contains no edges (with n nodes), the confusion is maximal with n!, the preferences carry the minimal amount of meaning.

Minimal and Maximal Number of Turnings

The minimal number of turnings a graph can have is 1, with a graph-edit distance of 0: any transitive closure of a path graph satisfies this criterion (if your preferences are already consistent, why change them to be more consistent?)

However, those graphs aren't the only graphs with exactly one turning, consider the following graph (left) and a possible turning (right) (with graph-edit distance 1; the changed edge is red, a nice opportunity for some rubrication):





One can easily see that it has exactly one turning, and checking with the code confirms:

```
>>> counter=nx.DiGraph()
>>> counterworld=['a', 'b', 'c', 'd']
>>> for i in range(0, len(smallworld)):
...     smallgraph.add_node(smallworld[i], ind=i)
>>> counter.add_edges_from([('a', 'b'), ('b', 'c'), ('c', 'd'), ('a', 'c'), ('b',
'd'), ('d', 'a')])
>>> counterres=list(turn_all(counter))
>>> len(counterres)
>>> >>> counterres[0].edges
OutEdgeView([('a', 'b'), ('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd'), ('c', 'd')])
```

For a graph with n nodes the maximal number of turnings it is upper-bounded by n!, and a sufficient condition for the graph to have that many turnings is when the graph is the union of a set of complete digraphs with disjoint nodes. For example the graph with 4 nodes and no edges has 24 possible turnings, as does the graph with 4 nodes and two edges $\{(1,2),(2,1)\}$.

We can prove this inductively: When considering a node-labeled graph with n nodes and no edges, the graph edit distance to any path graph variant of that graph is the same, because we always have to add $n-1+n-2+n-3 \ldots 1 = \frac{n-1+(n-1)^2}{2}$ edges to reach any transitive closure of a path graph (by the sum of any arithmetic progression). Let not G° be a graph with n nodes that is solely the union of complete digraphs with

disjoint nodes. When we now pick two nodes u and v from G° and add the edges

$$\{(u,v),(v,u)\} \cup \{(v,x)|(u,x) \in E^\circ\} \cup \{(u,y)|(v,x) \in E^\circ\} \cup \{(x,y)|(u,x) \in E^\circ, (v,y) \in E^\circ\}$$

(that is, we connect u and v, and all their neighbors) to G°, we have necessarily increased the graph-edit distance to any path graph by the same amount, we have symmetrically added edge-pairs that need to be broken in either direction.

Questions

One can now pose several (possibly distracting) questions:

- Does it matter whether we give `turn` a graph G or the transitive closure of G?
- Is there a more efficient algorithm to compute the turning?
  - Can it at least be made exponential?
  - Can we exploit the fact that we're always computing the graph-edit distance to a path-graph?
- As we add more options to our inconsistent preferences, do they become more likely to turn uninuely?
  - That is: Does it hold that $\frac{|U_n|}{|G_n|} < \frac{|U_{n+1}|}{|G_{n+1}|}$?
  - It should be possible to check this for small cases.

Number of Turnings for G_n

- In general, how does the size of U_n develop? What about $T_{n,2}$, or in general $T_{n,m}$?
  - Does the average number of turnings for inconsistent preferences converge to a specific number?
  - That is, what is $\lim_{n\to\infty} \frac{1}{G_n} \sum_{i=1}^{n} T_{n,i}$?
  - I predict 20% on the number monotonically increasing, 50% on monotonically decreasing and 30% on showing no clear pattern.

We can check these empirically! While it would be nice to prove anything about them, it's much nicer to investigate them computationally. This is pretty straightforward: For increasing n, generate G_n, for every $G \in G_n$, compute |turn_all(G)|, save the data in a file somewhere, and do interesting things with that data.

In code, we first generate all directed graphs with n nodes with a recursive function

```
def all_directed_graphs(n):
        if n<=0:
                return [nx.DiGraph()]
        graphs=all_directed_graphs(n-1)
        newgraphs=[]
        for g in graphs:
                g.add_node(n, ind=n)
                for tosubset in powerset(range(1, n+1)):
                        for fromsubset in powerset(range(1, n)):
```

```
                                gnew=g.copy()
                                for element in tosubset:
                                        gnew.add_edge(n, element)
                                for element in fromsubset:
                                        gnew.add_edge(element, n)
                                newgraphs.append(gnew)
        return newgraphs
```

and start turning:

```
max=16
for i in range(0,max):
        graphs=turn.all_directed_graphs(i)
        for g in graphs:
                print('{0},{1},"{2}"'.format(i, len(turn.turn_all(g)), g.edges))
```

However, my computer quickly freezes and I find out that this is a lot of graphs:

```
>>> [len(list(all_directed_graphs(i))) for i in range(0,5)]
[1, 2, 16, 512, 65536]
```

So the number directed graphs with 5 nodes would be 2^{32} = 4294967296, far too many for my puny laptop. But instead of generating them all, one can just generate a random sample and test on that, using the Erdős–Rényi model, for which networkx has the helpful function `generators.random_graphs.gnp_random_graph` (Wikipedia informs us that "In particular, the case p = ½ corresponds to the case where all $2^{\binom{n}{2}}$ graphs on n vertices are chosen with equal probability."). We have to randomly add reflexive edges (not included in the model, it seems) with probability ½ each, and labels for the nodes, and then we're good to go:

```
samples=256
for i in range(5,lim):
        for j in range(0,samples):
                g=nx.generators.random_graphs.gnp_random_graph(i, 0.5, directed=True)
                for n in g.nodes:
                        g.add_node(n, ind=n)
                        if random.random()>=0.5:
                                g.add_edge(n,n)
                print('{0},{1},"{2}"'.format(i, len(turn.turn_all(g)), g.edges))
```

We now run the script in the background, happily collecting data for us (`python3 collect.py >.https://niplav.github.io/../data/turnings.csv &`), and after a nice round of editing this text go back and try to make sense of the data, which runs squarely counter my expectations:

```
>>> import pandas as pd
>>> df=pd.read_csv('data/turnings.csv')
>>> df.groupby(['0']).mean()
           1
0
1   1.000000
2   1.875000
3   3.941406
4   9.390289
5  21.152344
6  39.885246
```

It seems like the mean number of turnings actually increases with the graph size! Surprising. I'm also interested in the exact numbers: Why *exactly* 3.390289… for the graphs with 4 nodes? What is so special about that number? (Except it being the longitude of the Cathedral Church of Christ in Lagos).

Looking at unique turnings turns (hehe) up further questions:

```
>>> def uniqueratio(g):
...     return len(g.loc[g['1']==1])/len(g)
...
>>> df.groupby(['0']).apply(uniqueratio)
0
1    1.000000
2    0.125000
3    0.089844
4    0.055542
5    0.050781
6    0.016393
dtype: float64
>>> def uniques(g):
...     return len(g.loc[g['1']==1])
>>> df.groupby(['0']).apply(uniques)
0
1       2
2       2
3      46
4    3640
```

Very much to my surprise, searching for "2,2,46,3640" in the OEIS yields *no results*, even though the sequence really looks like something that would already exist! (I think it has a specifically graph-theoretic "feel" to it). But apparently not so, I will submit it soon.

I omit the number of unique turnings for 5 and 6, for obvious reasons (I also believe that the ratio for 6 is an outlier and should not be counted). The number of unique resolutions for the graph with 1 node makes sense, though: Removing the reflexive edge should count as one edge action, but the graph only has one unique resolution:

```
>>> df.loc[df['0']==1]
   0  1        []
0  1  1        []
1  1  1  [(1, 1)]
```

Encoding Inconsistencies

Theory

Assuming that we have a set of axioms that describe which preferences are consistent and which are inconsistent, for the purposes of this text, we want to ideally find a set $\mathcal{N}$ of mathematical structures that

1. can represent preferences that violate each possible subset of those axioms.
    1. Each inconsistent preference should have exactly one element of $\mathcal{N}$ that represents it

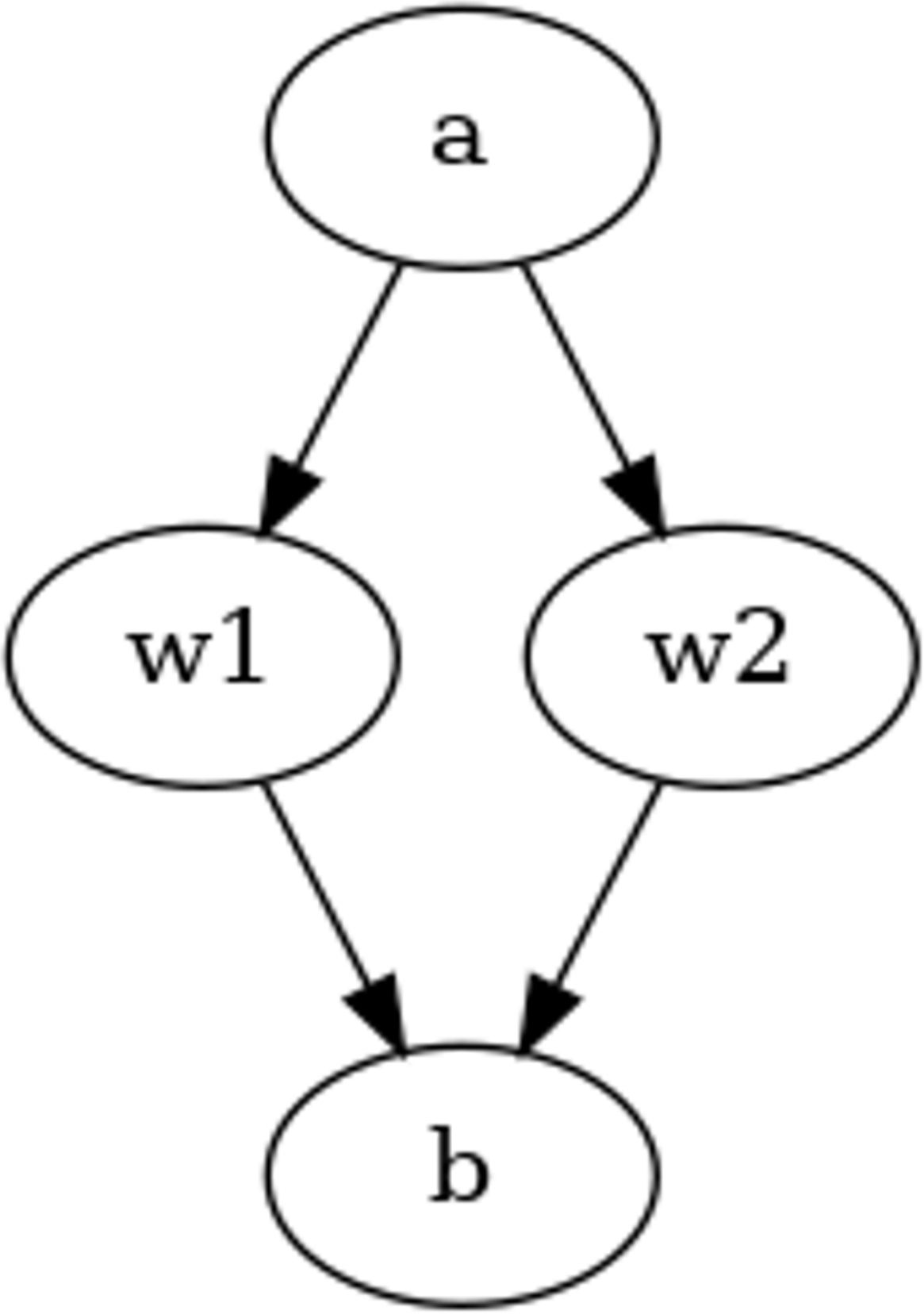
2. has a strict subset $\succ \subset \not\succ$ so that $\succ$ can represent only consistent preferences.

Discrete Case

The two relevant von Neumman-Morgenstern axioms are completeness and transitivity, with a directed graph one can also represent incompleteness and intransitivity.

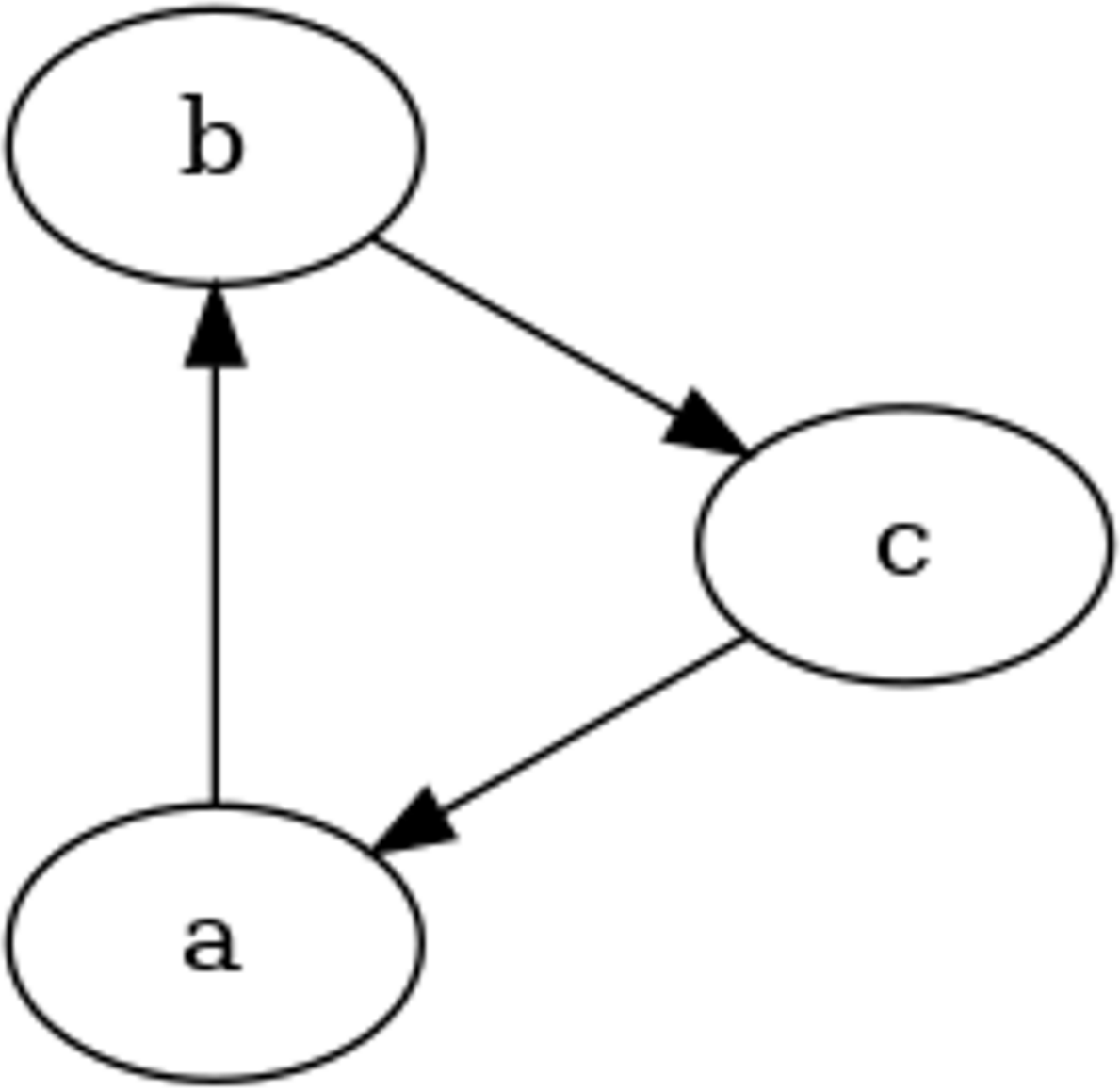
Incompleteness

Incompleteness (or incomparability) between two options w_1, w_2 can be represented by not specifying an edge between the two options, that is $(w_1, w_2) \notin E, (w_2, w_1) \notin E$.


a
w1
w2
b

Intransitivity

Intransitivity can be represented by cycles in the graph:



Non-Encodable Inconsistencies

With option set $\{a, b\}$ have preference $a \succsim b$, with option set $\{a, b, c\}$ have preferences $b \succsim a, a \succsim c, b \succsim c$.

Continuous Case

Incompleteness

- Minima/maxima in the vector field
- Discontinuities
- Undifferentiable points

Intransitivity

Curl in the vector field?

Discontinuity

Can only exist with incompleteness?

Dependence

Discussion

This leads to an interesting ethical consideration: is it a larger change to a preference relation to add new information or remove information?

It is discussed how to incorporate those weights into an algorithm for minimally transforming $G_{\gtrsim}$ into $G_{\geq}$.

Continuous Case

Vector Fields over Probability Simplices

Vector field over the probability simplex over the options (representing local preferences over lotteries).

Resolving Inconsistencies

Find mapping from vector field to another that makes the vector field consistent by minimizing the amount of turning/shrinking the vectors have to perform.

Graphons

?

Look into extremal graph theory.

Implications for AI Alignment

> I've seen six cities fall for this
> mathematics with incompetence
> red flags stand among the trees
> repugnant symphonies
> a billionaires tarantula just ate the ceiling
> thinking it was yet another floor

—Patricia Taxxon, "Hellbulb" from "Gelb", 2020

Ambitious Value Learning

Learn human values, check if known inconsistencies are encoded (to ensure learning at the correct level of abstraction), then make consistent.

Ontological Crises

> Furthermore, there remain difficult philosophical problems. We have made a distinction between the agent's uncertainty about which model is correct and the agent's uncertainty about which state the world is in within the model. We may wish to eliminate this distinction; we could specify a single model, but only give utilities for some states of the model. We would then like the agent to generalize this utility function to the entire state space of the model.

—Peter de Blanc, "Ontological Crises in Artificial Agents' Value Systems", 2010

If you know a mapping between objects from human to AI ontology, you could find the mapping from the (consistent) human probability simplex to the AI simplex?

Discrete Case

A node splits in two or more, or two or more nodes get merged. If the then resulting graph isn't a path graph, it can be turned with the method described above.

Further Questions

- Does every graph G have a unique graph $G^{'}$ so that G is the transitive closure of $G^{'}$?
- There is something interesting going on with lattices (?) over individual transitivity operations

Acknowledgements

Thanks to Miranda Dixon-Luinenburg for finding some typos.