Two Attempts to Formalize Counterpossible Reasoning in Deterministic Settings

Nate Soares and Benja Fallenstein

Machine Intelligence Research Institute, Berkeley, USA
{nate,benja}@intelligence.org

Abstract. This paper motivates the study of counterpossibles (logically impossible counterfactuals) as necessary for developing a decision theory suitable for generally intelligent agents embedded within their environments. We discuss two attempts to formalize a decision theory using counterpossibles, one based on graphical models and another based on proof search.

Keywords: decision theory, counterpossibles, logical counterfactuals, CDT, UDT

1 Introduction

What does it mean to "make good decisions"? To formalize the question, it is necessary to precisely define a process that takes a problem description and identifies the best available decision (with respect to some set of preferences[1]). Such a process could not be *run*, of course; but it would demonstrate a full understanding of the question.

The difficulty of this question is easiest to illustrate in a deterministic setting. Consider a deterministic decision procedure embedded within a deterministic environment (e.g., an algorithm operating in a virtual world). There is exactly one action that the decision procedure is going to select. What, then, "would happen" if the decision procedure selected a different action instead? At a glance, this question seems ill-defined, and yet, this is the problem faced by a decision procedure embedded within an environment.

Philosophers have studied candidate procedures for quite some time, under the name of *decision theory.* The investigation of what is now called decision theory stretches back to Pascal and Bernoulli; more recently decision theory has been studied by Lehmann [7], Lewis [9], Jeffrey [6], Pearl [12] and many others. Unfortunately, the standard answers from the literature do not allow for the description of an idealized decision procedure, as discussed in Section 2. Section 3 introduces the notion of "counterpossibles" (logically impossible counterfactuals) and motivates the need for a decision theory using them. It goes on to discuss

[1] For simplicity, assume von Neumann-Morgenstern rational preferences [13], that is, preferences describable by some utility function. The problems discussed in this paper arise regardless of how preferences are encoded.

two attempts to formalize such a decision theory, one using graphical models and another using proof search. Section 4 concludes.

2 Counterfactual Reasoning

The modern academic standard decision theory is known as "causal decision theory," or CDT. It is used under the guise of "potential outcomes" in statistics, economics and game theory, and it is used implicitly by many modern narrow AI systems under the guise of "decision networks."

Pearl's calculus of interventions on causal graphs [12] can be used to formalize CDT. This requires that the environment be represented by a causal graph in which the agent's action is represented by a single node. This formalization of CDT prescribes evaluating what "would happen" if the agent took the action a by identifying the agent's action node, cutting the connections between it and its causal ancestors, and setting the output value of that node to be a. This is known as a *causal intervention*. The causal implications of setting the action node to a may then be evaluated by propagating this change through the causal graph in order to determine the amount of utility expected from the execution of action a. The resulting modified graph is a "causal counterfactual" constructed from the environment.

Unfortunately, causal counterfactual reasoning is unsatisfactory, for two reasons. First, CDT is underspecified: it is not obvious how to construct a causal graph in which the agent's action is an atomic node. While the environment can be assumed to have causal structure, a sufficiently accurate description of the problem would represent the agent as arising from a collection of transistors (or neurons, or sub-atomic particles, etc.). While it seems possible in many cases to draw a boundary around some part of the model which demarcates "the agent's action," this process may become quite difficult in situations where the line between "agent" and "environment" begins to blur, such as scenarios where the agent distributes itself across multiple machines.

Secondly, CDT prescribes low-scoring actions on a broad class of decision problems where high scores are possible, known as *Newcomblike problems* [11]. For a simple example of this, consider a one-shot Prisoner's Dilemma played by two identical deterministic agents. Each agent knows that the other is identical. Agents must choose whether to cooperate (C) or defect (D) without prior coordination or communication. If both agents cooperate, they both achieve utility 2. If both defect, they both achieve utility 1. If one cooperates and the other defects, then the defector achieves 3 utility while the cooperator achieves 0.[2]

[2] This scenario (and other Newcomblike scenarios) are multi-agent scenarios. Why use decision theory rather than game theory to evaluate them? The goal is to define a procedure which reliably identifies the best available action; the label of "decision theory" is secondary. The desired procedure must identify the best action in all settings, even when there is no clear demarcation between "agent" and "environment." Game theory informs, but does not define, this area of research.

The actions of the two agents will be identical by assumption, but neither agent's action causally impacts the other's: in a causal model of the situation, the action nodes are causally separated, as in Figure 1. When determining the best action available to the left agent, a causal intervention changes the left node without affecting the right one, assuming there is some fixed probability p that the right agent will cooperate *independent* of the left agent. No matter what value p holds, CDT reasons that the left agent gets utility $2p$ if it cooperates and $2p + 1$ if it defects, and therefore prescribes defection [8].

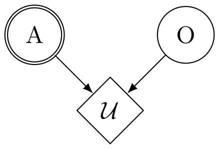

Fig. 1. The causal graph for a one-shot Prisoner's Dilemma. A represents the agent's action, O represents the opponent's action, and $\mathcal{U}$ represents the agent's utility.

Indeed, many decision theorists hold that it is in fact rational for an agent to defect against a perfect copy of itself in a one-shot Prisoner's Dilemma, as after all, no matter what the opponent does, the agent does better by defecting [5, 9]. Others object to this view, claiming that since the agents are identical, both actions must match, and mutual cooperation is preferred to mutual defection, so cooperation is the best available action [1]. Our view is that, in the moment, it is better to cooperate with yourself than defect against yourself, and so CDT does not reliably identify the best action available to an agent.

CDT assumes it can hold the action of one opponent constant while freely changing the action of the other, because the actions are causally separated. However, the actions of the two agents are *logically* connected; it is impossible for one agent to cooperate while the other defects. Causal counterfactual reasoning neglects non-causal logical constraints.

It is a common misconception that Newcomblike scenarios only arise when some other actor is a *perfect* predictor (perhaps by being an identical copy). This is not the case: while Newcomblike scenarios are most vividly exemplified by situations involving perfect predictors, they can also arise when other actors have only partial ability to predict the agent [10]. For example, consider a situation in which an artificial agent is interacting with its programmers, who have intimate knowledge of the agent's inner workings. The agent could well find itself embroiled in a Prisoner's Dilemma with its programmers. Let us assume that the agent knows the programmers will be able to predict whether or not it will cooperate with 90% accuracy. In this case, even though the programmers are imperfect predictors, the agent is in a Newcomblike scenario.

In any case, the goal is to formalize what is meant when asking that agents take "the best available action." Causal decision theory often identifies the best action available to an agent, but it sometimes fails in counter-intuitive ways, and therefore, it does not constitute a formalization of idealized decision-making.

3 Counterpossibles

Consider the sort of reasoning that a human might use, faced with a Prisoner's Dilemma in which the opponent's action is guaranteed to match our own:

> The opponent will certainly take the same action that I take. Thus, there is no way for me to exploit the opponent, and no way for the opponent to exploit me. Either we both cooperate and I get \$2, or we both defect and I get \$1. I prefer the former, so I cooperate.

Contrast this with the hypothetical reasoning of a reasoner who, instead, reasons according to causal counterfactuals:

> There is some probability p that the opponent defects. (Perhaps I can estimate p, perhaps not.) Consider cooperating. In this case, I get \$2 if the opponent cooperates and \$0 otherwise, for a total of \$$2p$. Now consider defecting. In this case I get \$3 if the opponent cooperates and \$1 otherwise, for a total of \$$2p + 1$. Defection is better no matter what value p takes on, so I defect.

Identifying the best action requires respecting the fact that identical algorithms produce identical outputs. It is not the *physical output of the agent's hardware* which must be modified to construct a counterfactual, it is the *logical output of the agent's decision algorithm.* This insight, discovered independently by Dai [4] and Yudkowsky [14], is one of the main insights behind "updateless decision theory" (UDT).

UDT identifies the best action by evaluating a world-model which represents not only causal relationships in the world, but also the logical effects of algorithms upon the world. In a symmetric Prisoner's Dilemma, a reasoner following the prescriptions of UDT might reason as follows:

> The physical actions of both myself and my opponent are determined by the same algorithm. Therefore, whatever action this very decision algorithm selects will be executed by both of us. If this decision algorithm selects "cooperate" then we'll both cooperate and get a payoff of 2. If instead this decision algorithm selects "defect" then we'll both defect and get a payoff of 1. Therefore, this decision algorithm selects "cooperate."

Using reasoning of this form, a selfish agent acting according to the prescriptions of UDT cooperates with an identical agent on a symmetric one-shot Prisoner's Dilemma, and achieves the higher payoff.[3]

[3] The agent does *not* care about the utility of its opponent. Each agent is maximizing its own personal utility. Both players understand that the payoff must be symmetric, and cooperate out of a selfish desire to achieve the higher symmetric payoff.

Evaluating a counterfactual outcome in which the *decision algorithm* behaves differently requires evaluating a logically impossible possibility, known as a "counterpossible."[4] As noted by Cohen [3], "the problem of counterpossible conditionals remains very near the center of philosophy."

To our knowledge, there does not yet exist a formal method of evaluating counterpossibles that is suitable for use in decision theory. This paper discusses two early attempts to formalize a decision theory which makes use of counterpossible reasoning.

3.1 Counterpossibles Using Graphical Models

Following Pearl's formalization of CDT (2000), one might be tempted to formalize UDT using a graphical approach. For example, one might attempt to construct a "logical graph" of the one-shot prisoner's dilemma, where each algorithm has its own "logical node," as in Figure 2. To do so, the graphical representation of the environment must encode not only causal relations, but also logical relations.

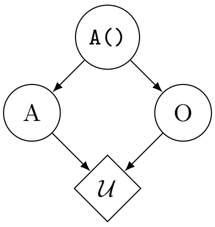

Fig. 2. The logical graph for a symmetric Prisoner's Dilemma where both the agent's action A and the opponent's action O are determined by the algorithm `A()`.

Given a probabilistic graphical model of the world representing both logical and causal connections, and given that one of the nodes in the graph corresponds to the agent's decision algorithm, and given some method of propagating updates through the graph, UDT can be specified in a manner very similar to CDT. To identify the best action available to an agent, iterate over all available actions $a \in A$, change the value of the agent's algorithm node in the graph to a, propagate the update, record the resulting expected utility, and return the action a leading to the highest expected utility. There are two obstacles to formalizing UDT in this way.

[4] Some versions of counterpossibles are quite intuitive; for instance, we could imagine how the cryptographic infrastructure of the Internet would fail if we found that $\mathtt{P} = \mathtt{NP}$, and it seems as if that counterfactual would still be valid even once we proved that $\mathtt{P} \neq \mathtt{NP}$. And yet by the Principle of Explosion, literally any consequence can be deduced from a falsehood, and thus no counterfactual could be "more valid" than any other in a purely formal sense.

The first obstacle is that UDT (like CDT) is underspecified, pending a formal description of how to construct such a graph from a description of the environment (or, eventually, from percepts). However, constructing a graph suitable for UDT is significantly more difficult than constructing a graph suitable for CDT. While both require decreasing the resolution of the world model until the agent's action (in CDT's case) or algorithm (in UDT's case) is represented by a single node rather than a collection of parts, the graph for UDT further requires some ability to identify and separate "algorithms" from the physical processes that implement them. How is UDT supposed to recognize that the agent and its opponent implement the same algorithm? Will this recognition still work if the opponent's algorithm is written in a foreign programming language, or otherwise obfuscated in some way?

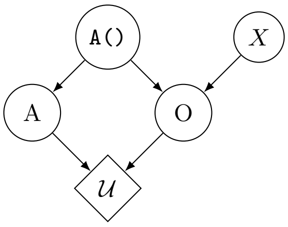

Fig. 3. The desired logical graph for the one-shot Prisoner's Dilemma where agent A acts according to `A()`, and the opponent either mirrors `A()` or does the opposite, according to the random variable X.

Even given some reliable means of identifying copies of an agent's decision algorithm in the environment, this may not be enough to specify a satisfactory graph-based version of UDT. To illustrate, consider UDT identifying the best action available to an agent playing a Prisoner's Dilemma against an opponent that does exactly the same thing as the agent 80% of the time, and takes the opposite action otherwise. It seems UDT should reason according to a graph as in Figure 3, in which the opponent's action is modeled as dependent both upon the agent's algorithm and upon some source X of randomness. However, generating logical graphs as in Figure 3 is a more difficult task than simply detecting all perfect copies of the an algorithm in an environment.

Secondly, a graphical model capable of formalizing UDT must provide some way of propagating "logical updates" through the graph, and it is not at all clear how these logical updates could be defined. Whenever one algorithm's "logical node" in the graph is changed, how does this affect the logical nodes of other algorithms? If the agent's algorithm selects the action a, then clearly the algorithm "do what the agent does 80% of the time and nothing otherwise" is affected. But what about other algorithms which correlate with the agent's algorithm, despite not referencing it directly? What about the algorithms of other agents which base their decisions on an imperfect model of how the agent will behave? In order to understand how logical updates propagate through a logical graph, we desire a better notion of how “changing” one logical fact can “affect” another logical fact.

3.2 Counterpossibles Using Proof Search

Given some method of reasoning about the effects of $\texttt{A()} = a$ on any other algorithm, a graphical formalization of UDT is unnecessary: *the environment itself is an algorithm* which contains the agent, and which describes how to compute the agent’s expected utility! Therefore, a formal understanding of “logical updating” could be leveraged to analyze the effects of $\texttt{A()} = a$ upon the environment; to evaluate the action a, UDT need only compute the expected utility available in the environment as modified by the assumption $\texttt{A()} = a$.

This realization leads to the idea of “proof-based UDT,” which evaluates actions by searching for formal proofs, using some mathematical theory such as Peano Arithmetic ($\mathcal{PA}$), of how much utility is attained in the world-model if $\texttt{A()}$ selects the action a. As a bonus, this generic search for formal proofs obviates the need to identify the agent in the environment: given an environment which embeds the agent and a description of the agent’s algorithm, no matter how the agent is embedded in the environment, a formal proof of the outcome will implicitly identify the agent and describe the implications of that algorithm outputting a. While that proof does the hard work of propagating counterpossibles, the high-level UDT algorithm simply searches all proofs, with no need to formally locate the agent. This allows for an incredibly simple specification of updateless decision theory, given below.

First, a note on syntax: Square quotes ($\ulcorner \cdot \urcorner$) denote sentences encoded as objects that a proof searcher can look for. This may be done via e.g., a Gödel encoding. Overlines within quotes denote “dequotes,” allowing the reference of meta-level variables. That is, if at some point in the algorithm $a := 3$ and $o := 10$, then the string $\ulcorner \texttt{A()} = \overline{a} \to \texttt{E()} = \overline{o} \urcorner$ is an abbreviation of $\ulcorner \texttt{A()} = 3 \to \texttt{E()} = 10 \urcorner$. The arrow $\ulcorner {\to} \urcorner$ denotes logical implication.

The algorithm is defined in terms of a finite set A of actions available to the agent and a finite sorted list O of outcomes that could be achieved (ordered from best to worst). The proof-based UDT algorithm takes a description $\ulcorner \texttt{E()} \urcorner$ of the environment and $\ulcorner \texttt{A()} \urcorner$ of the agent’s algorithm. $\texttt{E()}$ computes an outcome, $\texttt{A()}$ computes an action. It is assumed (but not necessary) that changing the output of $\texttt{A()}$ would change the output of $\texttt{E()}$.

To demonstrate how the algorithm works, consider UDT evaluating the actions available to a UDT agent in a symmetric prisoner’s dilemma. The list of outcomes is $O := [\,3, 2, 1, 0\,]$ according to the cases where the agent exploits, mutually cooperates, mutually defects, and is exploited, respectively. The set of actions is $A := \{\,\mathrm{C}, \mathrm{D}\,\}$ according to whether the agent cooperates or defects. To identify the best action, UDT iterates over outcomes in order of preference, starting with 3. For each outcome, it iterates over actions; say it first considers C. In the case that $\texttt{A()} = \mathrm{C}$, the agent cannot achieve the outcome 3, so there is no proof of

Algorithm 1: Proof-based UDT

```
Function UDT($\ulcorner E() \urcorner$, $\ulcorner A() \urcorner$):
    Sort the set of outcomes $O$ in nonincreasing preference order;
    for outcome $o \in O$ do
        for action $a \in A$ do
            if $\mathcal{PA}$ proves $\ulcorner A() = \overline{a} \to E() = \overline{o} \urcorner$ then return $a$ ;
    return the lexicographically first action in $A$
```

$\ulcorner \texttt{A()} = \mathrm{C} \to \texttt{E()} = 3 \urcorner$[5]. Next, UDT considers D. If the agent defects, then so does the opponent, so it would get outcome 1, and so there is no proof of $\ulcorner \texttt{A()} = \mathrm{D} \to \texttt{E()} = 3 \urcorner$. So UDT moves on to the next outcome, 2, and considers C. In this case, if the agent cooperates then so will the opponent, so there is a proof of $\ulcorner \texttt{A()} = \mathrm{C} \to \texttt{E()} = 2 \urcorner$, and so UDT selects C.

While this proof-based formalism of UDT is extremely powerful, it is not without its drawbacks. It requires a halting oracle in order to check whether proofs of the statement $\ulcorner \texttt{A()} = \overline{a} \to \texttt{E()} = \overline{o} \urcorner$ exist; but this is forgivable, as it is meant to be a definition of what it means to "choose the best action," not a practical algorithm. However, this formalization of UDT can only identify the best action if there exists a proof that executing that action leads to a good outcome. This is problematic in stochastic environments, and in any setting where $\mathcal{PA}$ is not a strong enough theory to find the appropriate proofs (which may well occur if agents in the environment are themselves searching for proofs about what UDT will prescribe, in order to guess the behavior of agents which act according to UDT).

There is also larger problem facing this formalism of UDT: even in simple examples, the algorithm is not guaranteed to work. Consider a case where the outcomes are $O := [3, 2, 1]$ corresponding in `E()` to the actions $A := \{\texttt{High}, \texttt{Med}, \texttt{Low}\}$. If we ask proof-based UDT to identify the best available action to the agent $\texttt{A()} := \texttt{const Low}$, and it considers the action `Med` before the action `High`, then it will misidentify `Med` as the best available action! This happens because there is a proof that $\texttt{A()} \neq \texttt{Med}$, and so $\texttt{A()} = \texttt{Med} \to \texttt{E()} = 3$ by the principle of explosion. (In fact, this sort of thing can happen whenever there is any action that is provably not taken.)

As discussed by Benson-Tilsen [2], this problem is averted in the important case $\texttt{A()} = \texttt{UDT}(\ulcorner\texttt{E()}\urcorner, \ulcorner\texttt{A()}\urcorner)$ (this fixed point exists, by Kleene's second recursion theorem). In this case, UDT does in fact get the best provably attainable outcome. This follows from the consistency of $\mathcal{PA}$: imagine that a is a action such that $\mathcal{PA}$ proves $\texttt{A()} \neq a$. Then $\mathcal{PA}$ proves that $\texttt{A()} = a$ implies the first outcome in O (which has the highest possible preference), and so UDT must either return a or return another action which implies the first outcome in O—but returning

[5] One must be careful with this sort of reasoning, for if $\mathcal{PA}$ could prove that $\texttt{A()} = \mathrm{D}$ then it could also prove $\texttt{A()} = \mathrm{C} \to \texttt{E()} = 3$ by the principle of explosion. However, in this case, that sort of "spurious proof" is avoided by technical reasons discussed by Benson-Tilsen [2]

a would be a contradiction. Therefore, either UDT will return an action which truly leads to the highest outcome, or there is no action a such that $\mathcal{PA}$ can prove $\texttt{A()} \neq a$, and thus the only proofs found will be genuine implications. Even so, the apparent deficits of UDT at analyzing other algorithms are troubling, and it is not obvious that reasoning about the logical implications of $\texttt{A()} = a$ is the right way to formalize counterpossible reasoning.

A better understanding of counterpossible reasoning may well be necessary in order to formalize UDT in a stochastic setting, where it maximizes expected utility instead of searching for proofs of a certain outcome. Such an algorithm would evaluate actions *conditioned* on the logical fact $\texttt{A()} = a$, rather than searching for logical implications. How does one deal with the case where $\texttt{A()} \neq a$, so that $\texttt{A()} = a$ is a zero-probability event? In order to reason about expected utility conditioned on $\texttt{A()} = a$, it seems necessary to develop a more detailed understanding of counterpossible reasoning. If one deterministic algorithm violates the laws of logic in order to output something other than what it outputs, then how does this affect other algorithms? Which laws of logic, precisely, are violated, and how does this violation affect other logical statements?

It is not clear that these questions are meaningful, nor even that a satisfactory general method of reasoning about counterpossibles actually exists. It is plausible that a better understanding of reasoning under logical uncertainty would shed some light on these issues, but a satisfactory theory of reasoning under logical uncertainty does not yet exist.[6] Regardless, it seems that some deeper understanding of counterpossibles is necessary in order to give a satisfactory formalization of updateless decision theory.

4 Conclusion

The goal of answering all these questions is not to identify practical algorithms, directly. Rather, the goal is to ensure that the problem of decision-making is well understood: without a formal description of what is meant by “good decision,” it is very difficult to justify high confidence in a practical heuristic that is intended to make good decisions.

It currently looks like specifying an idealized decision theory requires formalizing some method for evaluating counterpossibles, but this problem is a difficult one, and counterpossible reasoning is an open philosophical problem. While these problems have remained open for some time, our examination in the light of decision-theory, with a focus on concrete algorithms, has led to some new ideas. We are optimistic that further decision theory research could lead to significant progress toward understanding the problem of idealized decision-making.

[6] A *logically uncertain* reasoner can know both the laws of logic and the source code of a program without knowing what the program outputs.

References

1. Bar-Hillel, M., Margalit, A.: Newcomb's paradox revisited. British Journal for the Philosophy of Science 23(4), 295–304 (1972), http://www.jstor.org/stable/686730
2. Benson-Tilsen, T.: UDT with known search order. Tech. Rep. 2014–4, Machine Intelligence Research Institute (2014), http://intelligence.org/files/UDTSearchOrder.pdf
3. Cohen, D.: On what cannot be. In: Dunn, J., Gupta, A. (eds.) Truth or Consequences, pp. 123–132. Kluwer (1990)
4. Dai, W.: Towards a new decision theory. Less Wrong (2009), http://lesswrong.com/lw/15m/towards_a_new_decision_theory/
5. Gibbard, A., Harper, W.L.: Counterfactuals and two kinds of expected utility. In: Hooker, C.A., Leach, J.J., McClennen, E.F. (eds.) Foundations and Applications of Decision Theory, The Western Ontario Series in Philosophy of Science, vol. 13a. D. Reidel (1978)
6. Jeffrey, R.C.: The Logic of Decision. Chicago University Press, 2 edn. (1983)
7. Lehmann, E.L.: Some principles of the theory of testing hypotheses. Annals of Mathematical Statistics 21(1), 1–26 (1950)
8. Lewis, D.: Prisoners' dilemma is a Newcomb problem. Philosophy & Public Affairs 8(3), 235–240 (1979), http://www.jstor.org/stable/2265034
9. Lewis, D.: Causal decision theory. Australasian Journal of Philosophy 59(1), 5–30 (1981)
10. Lewis, D.: '"Why ain'cha rich?"'. Noûs 15(3), 377–380 (1981), http://www.jstor.org/stable/2215439
11. Nozick, R.: Newcomb's problem and two principles of choice. In: Rescher, N. (ed.) Essays in Honor of Carl G. Hempel, pp. 114–146. No. 24 in Synthese Library, D. Reidel (1969)
12. Pearl, J.: Causality. Cambridge University Press, 1 edn. (2000)
13. Von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press, 1 edn. (1944)
14. Yudkowsky, E.: Timeless decision theory. Tech. rep., The Singularity Institute, San Francisco, CA (2010), http://intelligence.org/files/TDT.pdf