Wei Dai's Updateless Decision Theory

Tyrrell McAllister

Abstract

Wei Dai developed an approach to decision theory called Updateless Decision Theory (UDT). This note describes two versions, both due to Dai, of the UDT formalism.

This note is just a more-pedantic version of Wei Dai's original descriptions of Updateless Decision Theory (UDT). The first section gives the original version of UDT [2], which Dai now calls "UDT1". In the next section, I give a similar description of UDT1.1, an improvement due to Dai [1]. In both cases, I've stripped out almost all motivation.

1 UDT1

Let $P = \langle P_1, P_2, \ldots \rangle$ be a sequence of programs. To specify a UDT1 algorithm, one must specify the following:

- a set $\mathbf{X}$ of inputs that the UDT1 algorithm might receive;
- a set $\mathbf{Y}$ of outputs that the UDT1 algorithm might produce in response to an input;
- for each program P_i in P, a set $\mathbf{E}_i$ of execution histories that P_i might produce (so far as the algorithm or its writers know);
- a utility function $U\colon \mathbf{E} \to \mathbb{R}$, where $\mathbf{E} \coloneqq \mathbf{E}_1 \times \mathbf{E}_2 \times \cdots$;
- a **mathematical intuition** M, which is a function that takes three arguments, namely
  - an input $X \in \mathbf{X}$,
  - an output $Y \in \mathbf{Y}$, and
  - a sequence $E = \langle E_1, E_2, \ldots \rangle \in \mathbf{E}$,

and returns a probability $M(X, Y, E)$ that E_i would be the execution history of P_i for all i if the UDT1 algorithm were to return output Y upon receiving input X[1].

(Note that, since $M(X, Y, E)$ is a probability, the function M must be nonnegative and normalized for each choice of $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$. That is, for every $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$, we must have that $\sum_{E \in \mathbf{E}} M(X, Y, E) = 1$.)

Suppose that the algorithm receives input $X \in \mathbf{X}$. To determine its response, the algorithm computes, for every possible output $Y \in \mathbf{Y}$, the expected utility

$$EU_X(Y) \coloneqq \sum_{E \in \mathbf{E}} M(X, Y, E)\, U(E)$$

of producing output Y. The algorithm then produces the output $Y^* \in \mathbf{Y}$ that maximized EU_X. That is, the algorithm returns that output $Y^* \in \mathbf{Y}$ such that

$$EU_X(Y^*) = \max_{Y \in \mathbf{Y}} EU_X(Y).$$

2 UDT1.1

Dai later suggested an improvement to UDT, which he dubbed "UDT1.1" (the original UDT now to be called "UDT1").

A UDT1.1 algorithm is specified in the same way as a UDT1 algorithm, with the following exceptions:

- An additional set needs to be specified — namely, a set $\mathbf{I}$ of input-output mappings that the algorithm may (for all it knows) be implementing. Thus, an element $f \in \mathbf{I}$ is a function $f\colon \mathbf{X} \to \mathbf{Y}$.
- The mathematical intuition function M now has a different domain: it now takes only *two* arguments, namely
  - an input-output mapping $f \in \mathbf{I}$ and
  - a sequence $E = \langle E_1, E_2, \ldots \rangle \in \mathbf{E}$,

  and returns a probability $M(f, E)$ that E_i will be the execution history of P_i for all i if the UDT1.1 algorithm were to implement the input-output mapping $f\colon \mathbf{X} \to \mathbf{Y}$.

[1] I believe that Wei Dai generally supposes that M has complete access to the source codes of the programs P_i. This isn't strictly necessary for the UDT1 formalism, however. The UDT1 algorithm only needs to know enough about the P_i to "intuit" probabilities for possible execution histories as functions of how the algorithm responds to inputs. Since Dai does not go into precisely how the mathematical intuition works, the extent of its access to the source codes of the P_i is not part of the UDT1 formalism

The UDT1.1 algorithm determines its output as follows. It first performs a calculation that does not depend on any input: The algorithm computes, for every possible input-output mapping $f \in \mathbf{I}$, the expected utility

$$EU(f) \coloneqq \sum_{E \in \mathbf{E}} M(f, E)\, U(E)$$

of implementing f. Let $f^* \in \mathbf{I}$ be the input-output mapping that maximized EU, so that

$$EU(f^*) = \max_{f \in \mathbf{I}} EU(f).$$

Then, upon receiving input X, the algorithm returns $f^*(X)$. (In fact, as long as the algorithm remembers which function is f^*, the algorithm doesn't need to perform any more expected-utility calculations.)

References

[1] Wei Dai, *Explicit Optimization of Global Strategy (Fixing a Bug in UDT1)* [online], URL: `http://lesswrong.com/lw/1s5/explicit_optimization_of_global_strategy_fixing_a/` [cited 23 July 2011].

[2] ———, *Towards a New Decision Theory* [online], URL: `http://lesswrong.com/lw/15m/towards_a_new_decision_theory/` [cited 23 July 2011].