For example, beings (as well as possible beings and things impossible of being) can be understood in the context of possible worlds. A "possible world" is a sufficiently complete description of possible states of affairs described through chains of propositions. We may observe that things impossible of being, such as a square circle, have in them mutually inconsistent required core attributes; they cannot be realised in any possible world. Possible beings would exist in at least one possible world were it actualised.

For instance, a contingent being B that depends on C might exist in a world W and not in a closely neighbouring one W' if C is present in W but not W'; C thus being an enabling, necessary causal factor for B. By contrast, a necessary being F will exist in all possible worlds, showing itself to be a framework element for such a world.

A key insight is that for any world W to be distinct from W' it requires some factor A in W that is absent in W'. We may then partition the factors of W as $W = \{A|\neg A\}$. After partitioning, we will have two distinct groups—the factor A and all of the factors which are not A. The null set corresponds to zero. Each particular set in the partition can be counted as the number one, and the combination of both partitions (even in a single world where A is an empty set) is two. Thus, for any particular possible world W, the quantities 0, 1, 2 are necessarily present. Taking the von Neumann construction, immediately we find $\mathbb{N}$, thence (using additive inverses) $\mathbb{Z}$, so also (taking ratios) $\mathbb{Q}$ and (summing convergent power series) $\mathbb{R}$; where $\mathbb{Z}$ provides unit-stepped mileposts in $\mathbb{R}$. That is, a structured core of quantities will be present in any W, and we may regard mathematics as the study of the logic of structure and quantity. Extensions to the hyperreals $\mathbb{R}^*$ follow by construction of some H that has as reciprocal $h = \frac{1}{H}$ closer to 0 than $\frac{1}{n}$ for any n in $\mathbb{N}$. Therefore, relationships and linked operations across such quantities will also be present, or may be constructed as needed. Illustrating, after Abraham Robinson (Robinson, 1966), hyperreals allow calculus to be treated as extensions of algebra in $\mathbb{R}^*$.

Thus, while bare distinct identity and coherence focused on quantities will not *cause* things by the inherent potential or action of such entities, they instead are logical constraints on being and are tied to what can or must be or cannot be or happens not to be. So, too, we may see that the abstract logic model worlds that we may construct then lead to key entities that if necessary are framework to any possible world; thus applicable to our common world. By contrast, if certain quantities and relationships are merely part of the contingencies of some W'' that is close enough to our own, they may provide adequate analogies for modelling.

As a result, we have good reason to expect that mathematical reasoning and core entities will in many cases be highly relevant to and have powerful predictive power for our common world.

Robinson, Abraham (1966). *Non-Standard Analysis*. Amsterdam: North-Holland Publishing Company.

Wigner, Eugene (1960). "The Unreasonable Effectiveness of Mathematics in the Natural Sciences". In: *Communications in Pure and Applied Mathematics* 13.1, pp. 1–14. DOI: 10.1002/cpa.3160130102.

Independence Conservation and Evolutionary Algorithms

Eric Holloway

DOI: 10.33014/issn.2640-5652.2.1.holloway.2

Levin's Law of Independence Conservation

Leonid Levin's 1984 article (Levin, 1984) is the first to this author's knowledge to prove a fully stochastic conservation of information law. Levin titled his law 'independence conservation' which he considered fairly obvious, describing it as "Torturing an uninformed witness cannot give information about the crime!"

Levin's law is not well known, which is unfortunate since the more commonly known conservation laws are focused either only on the random or deterministic case. Levin's law is remarkable because it unifies both the random and deterministic cases, showing that the combination also cannot result in information increase.

The second remarkable thing about his law is how easy it is to prove, given some preliminaries about algorithmic information.

Algorithmic Information Theory Background

First is required the notion of algorithmic information, which is defined on bitstrings. Algorithmic information is the length of the shortest program that generates a particular bitstring.

$$K(x) := \min_{y|\mathcal{U}(y)=x} |y|. \tag{1}$$

The shortest program is itself known as the *elegant program* for that particular bitstring. Each bitstring has a unique elegant program.

$$y^* := \arg \min_{y|\mathcal{U}(y)=x} |y|. \tag{2}$$

All programs either terminate after a fixed amount of time, or never terminate. All the programs in question are known as *prefix free*, which means that no terminating program begins another terminating program.

Algorithmic mutual information is the length of this program if we are also provided another bitstring as input, subtracted from the length if we are not provided the extra input bitstring.

$$I(x : y) := K(y) - K(y|x). \tag{3}$$

Unfortunately, this basic definition of algorithmic mutual information is only symmetric under a logarithmic error, because we have to mark where one bitstring starts and the other ends. This requires a number of bits logarithmic on the size of the shortest bitstring, which is x in this case.

$$I(x : y) - I(y : x) = O(log(x)). \tag{4}$$

We can improve the definition of algorithmic mutual information to be completely symmetric under a constant that is independent of the bitstrings we are looking at, which in other words means we don't have to worry about the constant and the algorithmic mutual information is symmetric as far as we are concerned. This improvement is to use the elegant program of the input bitstring instead of the bitstring itself.

$$I^*(x : y) := K(y) - K(y|x^*). \tag{5}$$

Since the elegant program halts once it has generated the input bistring, we know we can start on the next bitstring, so we avoid having to encode the bitstring length. This saves us from having to use the logarithimic term.

$$I^*(x : y) - I^*(y : x) = O(1). \tag{6}$$

One final fascinating point on algorithmic information is that we can also use it to create a *universal distribution*.

$$\mathbf{m}(x) := 2^{-K(x)}. \tag{7}$$

"Universal" means is that we have a distribution that provides the highest probability for every bitstring possible, within a multiplicative constant, under the assumption that we are only dealing with computable generating sources for the bitstrings.

$$\mathbf{m}(x) \geq p(x) * O(1). \tag{8}$$

The computable distribution assumption is a reasonable assumption for dealing with physical phenomena, since (as far as we know) everything physical can be modeled to theoretically perfect accuracy with enough computational resources.

Proving Levin's Deterministic Law

Alright, so now onto proving Levin's law.

We first start with a simple lemma, that providing more information can only decrease conditional algorithmic information. In other words, the more we know about y, the less information we need to describe y.

$$K(y|x) \geq K(y|x, z) + O(1). \tag{9}$$

We now introduce another simple lemma that with a program to generate x, namely z which we execute with Turing machine U to generate x,

$$x = U(z), \tag{10}$$

we can generate both x and z. Thus, the joint information is the same between z and x, z.

$$K(z) = K(x, z) + O(1). \tag{11}$$

This also means to generate the triple $\{y, x, z\}$ we only need y and z.

$$K(y, x, z) = K(y, z) + O(1). \tag{12}$$

Since it is the case that

$$K(y|x, z) = K(y, x, z) - K(x, z). \tag{13}$$

Then performing replacements to Equation 13 with Equations 11 and 12, we get

$$\begin{aligned} K(y|x, z) &= K(y, z) - K(z) && (14)\\ &= K(y|z). && (15) \end{aligned}$$

Combining Equations 9 and 14 shows us that x can never tell us more about y than z.

$$\begin{aligned} K(y|x) &\geq K(y|x, z) && (16)\\ &= K(y|z). && (17) \end{aligned}$$

We can then use Equation 16 to show running a program f on i does not increase mutual information with y. The notation $U(f.i)$ to mean we've run program f with input i using a universal Turing machine U.

First we decompose the mutual information.

$$\begin{aligned} I^*(f(i) : y) &= I^*(U(f.i) : y) && (18)\\ &= K(y) - K(y|U(f.i)). && (19)\\ & && (20) \end{aligned}$$

Now, we set $z = f.i$ and $x = U(z) = U(f.i)$, and then apply Equation 16 to Equation 18.

$$\begin{aligned} I^*(f(i) : y) &= I^*(x : y) && (21)\\ &= K(y) - K(y|x) && (22)\\ &\leq K(y) - K(y|x, z) && (23)\\ &= K(y) - K(y|z) && (24)\\ &= I^*(z : y) && (25)\\ &= I^*(f, i : y). && (26) \end{aligned}$$

Giving the final concise result,

$$I^*(f(i) : y) \leq I^*(f, i : y). \tag{27}$$

This Equation 27 states that executing function f on input i does not produce any more information about y than the function and input before they are executed. In other words, running a program doesn't produce any information.

Proving Levin's Random Law

Now with the deterministic version out of the way, we can move onto the random version.

The random version asks, what if we generate f randomly, could that result in an information gain? This question is based on the fact that generating f randomly will result in an f with a lot of algorithmic information, since it will be incompressible. So, even though running $U(f.i)$ doesn't give us anything new, the initial selection of f may start us off with a good amount of information about y.

Levin's second step in proving the random law shows this intuition is false. To prove the second law, we will rely on the dominance property of the universal distribution in Equation 8.

$$\begin{aligned} \sum_f p(f) I^*(f, i : y) &= \sum_f p(f) \log_2 \frac{\mathbf{m}(f, i|y)}{\mathbf{m}(f, i)} && (28)\\ &\leq \sum_f p(f) \log_2 \frac{\mathbf{m}(f, i|y)}{\mathbf{m}(f)\mathbf{m}(i)} && (29)\\ &\leq \log_2 \sum_f \frac{p(f)\mathbf{m}(f, i|y)}{\mathbf{m}(f)\mathbf{m}(i)} && (30)\\ &\leq \log_2 \frac{\sum_f \mathbf{m}(f, i|y)}{\mathbf{m}(i)} && (31)\\ &= \log_2 \frac{\mathbf{m}(i|y)}{\mathbf{m}(i)} && (32)\\ &= K(i) - K(i|y) && (33)\\ &= I^*(i : y) && (34) \end{aligned}$$

Which gives us the result that randomly generating an f is not expected to provide any information about y.

$$\sum_f p(f) I^*(f, i : y) \leq I^*(i : y). \quad (35)$$

To wrap up the independency conservation law, we apply Equation 27 to Equation 35.

$$\sum_f p(f) I^*(f(i) : y) \leq \sum_f p(f) I^*(f, i : y) \leq I^*(i : y). \quad (36)$$

Evolutionary Algorithms

What sort of impact, if any, does the law of independency conservation in Equation 36 have on evolutionary algorithms?

One of the simplest implications is that if there is a target area independently designated by y, then it is not possible to randomly evolve population i towards y without any fitness information.

However, there are also implications even if there is fitness information provided for the evolution of i.

Let's say that f represents one round of an evolutionary algorithm applied to i, which consists of the following steps:

1. crossover population
2. vary population
3. select population

i is defined as a population of bitstrings selected at random.

We then define y as the set of bitstrings rated at a certain level of fitness.

Part of f stays constant, and another part is varied randomly, such as mutation and which bitstring sections are crossed. We denote the random part as r.

This gives us a surprising result.

$$\sum_r p(r) I^*(f(i, r) : y) \leq I^*(f, i : y). \quad (37)$$

Equation 37 states that each round of randomized evolution provides no further information about the fitness region y than already existed in the initial conditions of the original population i and the evolutionary algorithm f. Thus, this equation proves that evolutionary algorithms cannot generate algorithmic mutual information, even in regards to regions specified according to fitness.

Levin, Leonid A (1984). "Randomness conservation inequalities; information and independence in mathematical theories". In: *Information and Control* 61.1, pp. 15–37.

CrowdRank: A Simple Ranking Algorithm for Crowdsourced Rating Systems with Uneven Participation

Jonathan Bartlett

DOI: 10.33014/issn.2640-5652.2.1.bartlett.3

Introduction

Public rating systems are difficult to score well. Voting systems tend to simply favor what is already popular. Averaging systems tend to have significant variance if there are not enough people scoring.

For instance, let's say that I run a songwriting contest and have 100 entries. I then put it out to a public vote on the Internet to see who wins. Most people are not going to listen to all 100 songs. If I do a simple "thumbs up" approach and count how many votes a song has, then whichever songwriter has the best existing following will simply tell their fans to vote for them, and it will simply devolve into a popularity contest.

Let's say instead I do a rating system where you can rate a song between 0 and 100. Now, songs by popular artists will actually be negatively weighted because they will have more visibility for negative ratings. It is not hard for a few votes to be all 100s, but it is hard for a thousand votes to be that way. Thus, those who have fewer ratings have an advantage.