

# Evolutionary Computation: A Perpetual Motion Machine for Design Information?

By Robert J. Marks II

*Evolutionary computing, modeled after Darwinian evolution, is a useful engineering tool. It can create unexpected, insightful and clever results. Consequently, an image is often painted of evolutionary computation as a free source of intelligence and information. The design of a program to perform evolutionary computation, however, requires infusion of implicit information concerning the goal of the program. This information fine tunes the performance of the evolutionary search and is mandatory for a successful search.*

## Computational Intelligence

Fifty years ago, Ross W. Ashby asked "Can a Chess Machine Outplay Its Maker?" (BRITISH JOURNAL FOR THE PHILOSOPHY OF SCIENCE, 1952). We know today that it can. A more relevant question is "can a computer program generate more information than it is given?" Evolutionary computing, on the surface, seems to be a candidate paradigm. As with all "something for nothing" claims, this is not the case.

Pioneers of evolutionary computing in the 1960s proposed that computer emulation of evolution overcame the difficulty of demonstrating Darwinian evolution in the biology lab. Proof of Darwinian evolution "has from the beginning been handicapped by the fact that no proper test has been found to decide whether such evolution was possible and how it would develop under controlled conditions." (N.A. BARRICELLI, ACTA BIOTHEORETICA, 1962.) "In general, it is usually impossible or impracticable to test hypotheses about evolution in a particular species by the deliberate setting up of controlled experiments with living organisms of that species. We can attempt to partially to get around this difficulty by constructing models representing the evolutionary system we wish to study, and use these to test at least the theoretical validity of our ideas." (J.L. CROSBY, "COMPUTERS IN THE STUDY OF EVOLUTION", SCI. PROG. OXF., 1967.)

## Engineering Design

Evolutionary computation is used today largely in engineering design and problem solving. Design begins with establishing a goal or design objective. From a favorite list of paradigms, a viable model is chosen. Design consists of identification of parameter values within the chosen model. Design has been defined as "the judicious manipulation of mid-range values" within the confines of a model (RANDALL JEAN, 2005). Search algorithms do this with the aid of a computer.

Consider the simple example of designing a recipe for boiling an egg. Our questions include the following.

1. Do we place the eggs in cold water and bring to a boil, or place the eggs in boiling water? (two choices)
2. How long do we boil the eggs?
3. Do we remove the pan from the heat and let the water cool, place the eggs on a dish to cool, or immediately place the eggs in cold water? (three choices)

At step (1) there are two choices, and at step (3), three choices. For the duration of boiling in step (2), let's assume there are choices in fifteen second intervals from 30 seconds to three minutes: 0:30, 0:45, 1:00, ..., 3:00. That's eleven choices of time intervals. The total number of possible recipes is therefore  $2 \times 11 \times 3 = 66$ . We have defined a search space, but have not yet defined what our design criterion is, namely, what is the optimal recipe? Suppose I taste the egg and rate it from one to 100 in terms of taste. This measure, assigned to each of the 66 recipes, is the fitness of the recipe. Anything above a 90 will meet the design criterion. The design goal is identification of a recipe that meets the design criterion.

Assume you have never cooked and have absolutely no idea which recipe is best. We apply Bernoulli's principle of insufficient reason which states that, in the absence of any prior knowledge, we must assume that all the recipes have an equal probability of being best. One recipe must be assumed as good as another. To find the optimal recipe, all 66 would need to be tried. One approach to find a decent recipe is trial and

error. If trial and error could be done on computer, the tests could be done quickly. Suppose we can emulate the boiling of the egg and the fitness of the result on a computer. Then we could determine the optimal recipe quickly by evaluating all 66 recipes. Looking at all possible solutions is called exhaustive search. Unfortunately, search problems scale poorly and this is not possible for even reasonably sized problems. If we have, instead of three, 100 variables and each variable has ten possible outcomes, the number of elements in the search space becomes  $10^{100}$  (i.e., 10 multiplied by itself 100 times), which is a larger number than there are atoms in the universe. Exhaustive search is not possible in such cases.

We can remove Bernoulli's principle of insufficient reason from the search problem only through infusion of information into the search process. The information can be explicit. For the egg example, knowledge of chemistry tells us that placing the boiled eggs in cold water retards the chemical reaction which will ultimately make the eggs smell like sulfur. Assuming a sulfur smell will detract from the fitness, we can eliminate one of the search variables and reduce the search to 44 recipes. Alternately, the information can be implicit. You may know, for example, that of ten famous egg boilers, two place the raw eggs in cold water and eight in boiling water. This information can guide your search of recipes initially to those with a greater chance of meeting the design criterion.

### **The Need for Implicit Information**

Purely theoretical considerations suggest that, given a fast enough computer and sufficient time, a space can be successfully searched to find the optimal solution. But this is the myth of "monkeys at a typewriter" The story, theoretically plausible, says that if enough monkeys pound out random letters long enough, all of the great texts in history will eventually result. If enough monkeys pound out random letters for a long enough time, all of the great texts, such as Moby Dick (1,170,200 characters), Grimms Tales (1,435,800 characters) and the King James Bible (3,556,480 letters not including spaces) will eventually result. The finiteness of the closed universe, however, prohibits this.

Looking for a single solution in a large unstructured search space is dubbed a "needle in a haystack" problem. In moderately large cases, it simply can't be done. Choosing randomly from a 26 letter alphabet, the chances of writing the KJB is  $26^{3,556,480} = 3.8 \cdot 10^{5,032,323}$ . This is a number so large it defies description. If all the matter in the universe ( $10^{58}$  kg) were converted to energy ( $E = mc^2$ ) ten billion times per second since the big bang (20 billion years) and all this energy were used to generate text at the minimum irreversible bit level (i.e.,  $\ln(2) kT = 2.9 \cdot 10^{-21}$  joules per bit), then about  $10^{88}$  messages as long as the KJB could be generated. If we multiply this by the number of atoms in the universe ( $10^{78}$  atoms), we have  $10^{166}$  messages, still dwarfed by the required  $10^{5,032,323}$ .

Let's try a more modest problem: the phrase  
IN\*THE\*BEGINNING\*GOD\*CREATED

(We could complete the phrase with "the heaven and the earth," but the numbers grow too large.) Here there are 27 possible characters (26 letters and a space) and a string has length 28 characters. The odds that this is the phrase written by the monkeys is  $27^{28} = 1.20 \cdot 10^{40}$  to one. This number isn't so big that we can't wrap our minds around it. The chance of a monkey typing 28 letters and typing these specific words is the same as choosing a single atom from over one trillion short tons of iron. [Using Avogadro's number, we compute 2728 atoms (1 mole per  $6.022 \cdot 10^{23}$  atoms) (55.845 grams per mole) (1 short ton per 907,185 grams) =  $1.22 \cdot 10^{12}$  short tons.]

Quantum computers would help by reduction of the equivalent search size by a square root (HO et al., IEEE TRANS. AUT. CONT., MAY 2003, P.783), but the problem remains beyond the resources of the closed universe. Information must be infused into the search process.

Searching an unstructured space without imposition of structure on the space is computationally prohibitive for even small problems. Early structure requirements included gradient availability, the dependence of the optimal solution on the second derivative of the fitness, convexity, and unimodal fitness functions.

(BREMNERMAN et al. "GLOBAL PROPERTIES OF EVOLUTION PROCESS, 1966; NASH, "SUMT (REVISITED)", OPERATIONS RESEARCH, 1998.) More recently, the need for implicit information imposed by design heuristics has been emphasized by the no free lunch theorems (WOLPERT, ET AL., IEEE TRANS. EVOLUTIONARY COMPUTATION, 1997.) which have shown, "unless you can make prior assumptions about the ... [problems] you are working on, then no search strategy, no matter how sophisticated, can be expected to perform better than any other" (HO op. cit.) No free lunch theorems "indicate the importance of incorporating problem-specific knowledge into the behavior of the [optimization or search] algorithm." (WOLPERT, op. cit.)

### **Sources of Information**

A common structure in evolutionary search is an imposed fitness function, wherein the merit of a design for each set of parameters is assigned a number. The bigger the fitness, the better. The optimization problem is to maximize the fitness function. Penalty functions are similar, but are to be minimized. In the early days of computing, an engineer colleague of mine described his role in conducting searches as a penalty function artist. He took pride in using his domain expertise to craft penalty functions. The structured search model developed by the design engineer must be, in some sense, a good model. Exploring through the parameters of a poor model, no matter how thoroughly, will not result in a viable design. In a contrary manner, a cleverly conceived model can result in better solutions in faster time.

Here is a simple example of structure in a search. Instead of choosing each letter at random, let's choose more commonly used letters more frequently. If we choose characters at random then each character has a chance of  $1/27 = 3.7$  percent of being chosen. In English, the letter "e" is used about 10 percent of the time. A blank occurs 20 percent of the time. If we choose letters in accordance to their frequency of occurrence, then the odds of choosing IN\*THE\*BEGINNING\*GOD\*CREATED nose dives to a five one millionths (0.0005%) of its original size – from  $1.2 \cdot 10^{40}$  to  $5.35 \cdot 10^{34}$ . This is still a large number: the trillion tons of iron has been reduced to 5 and a half million tons. If we use the frequency of digraphs, we can reduce it further. (Digraphs are letter pairs that occur frequently; for instance, the digraph "e\_" where "\_" is a space is the most common pair of characters in English.) Trigraph frequency will reduce the odds more.

### **The Fine Tuning of the Search Space**

As more implicit structure is imposed on the search space, the easier the search becomes. Even more interesting is that, for moderately long messages, if the target message does not match the search space structuring, the message won't be found. (PAPOULIS, PROBABILITY, RANDOM VARIABLES AND STOCHASTIC PROCESSES, 1991.)

The search space fine-tuning theorem. Let a search space be structured with a disposition to generate a type of message. If a target does not match this predisposition, it will be found with probability zero.

This theorem, long known in information theory in a different context, is a direct consequence of the law of large numbers. If, for example, we structure the search space to give an "e" 10 percent of the time, then the number of "e's" in a message of length 10,000 will be very close to 1000. The curious book Gadsby, containing no "e's", would be found with a vanishingly small probability.

Structuring the search space also reduces its effective size. The search space consists of all possible sequences. For a structured space, let's dub the set of all probable sequences that are predisposed to the structure the search space subset. For frequency of occurrence structuring of the alphabet, all of the great novels we seek, except for Gadsby, lie in or close to this subset.

The more structure is added to a search space, the more added information there is. Trigraphs, for example, add more information than digraphs.

### **Diminishing subset theorem.**

As the length of a sequence increases and the added structure information increases, the percent of elements in the search subset goes to zero.

Structuring of a search space therefore not only confines solutions to obey the structure of the space; the number of solutions becomes a diminishingly small percentage of the search space as the message length increases. (IBID.)

### **Final thoughts.**

Search spaces require structuring for search algorithms to be viable. This includes evolutionary search for a targeted design goal. The added structure information needs to be implicitly infused into the search space and is used to guide the process to a desired result. The target can be specific, as is the case with a precisely identified phrase; or it can be general, such as meaningful phrases that will pass, say, a spelling and grammar check. In any case, there is yet no perpetual motion machine for the design of information arising from evolutionary computation.

BIOSKETCH: Robert J. Marks II is Distinguished Professor of Engineering and Graduate Director in the Department of Engineering at Baylor University. He is Fellow of both IEEE and The Optical Society of America. Professor Marks has received the IEEE Centennial Medal. He has served as Distinguished Lecturer for the IEEE Neural Networks Society and the IEEE Computational Intelligence Society. Dr. Marks served as the first President of the IEEE Neural Networks Council (now a Society). He has over 300 publications. Some of them are very good. Eight of Dr. Marks' papers have been reproduced in volumes of collections of outstanding papers. He has three US patents in the field of artificial neural networks and signal processing.