Why We Do Not Evolve Software? Analysis of Evolutionary Algorithms

Roman V Yampolskiy

Department of Computer Engineering and Computer Science, J.B. Speed School of Engineering, University of Louisville, Louisville, KY, USA.

Evolutionary Bioinformatics
Volume 14: 1–11

DOI: 10.1177/1176934318815906
SAGE

ABSTRACT: In this article, we review the state-of-the-art results in evolutionary computation and observe that we do not evolve nontrivial software from scratch and with no human intervention. A number of possible explanations are considered, but we conclude that computational complexity of the problem prevents it from being solved as currently attempted. A detailed analysis of necessary and available computational resources is provided to support our findings.

KEYWORDS: Darwinian algorithm, genetic algorithm, genetic programming, optimization

RECEIVED: October 15, 2018. **ACCEPTED:** November 6, 2018.

TYPE: Original Research

FUNDING: The author(s) received no financial support for the research, authorship, and/or publication of this article.

DECLARATION OF CONFLICTING INTERESTS: The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

CORRESPONDING AUTHOR: Roman V Yampolskiy, Department of Computer Engineering and Computer Science, J.B. Speed School of Engineering, University of Louisville, Duthie Center for Engineering, 215, 222 Eastern Pkwy, Louisville, KY 40208, USA.
Email: roman.yampolskiy@louisville.edu

> We point out a curious philosophical implication of the algorithmic perspective: if the origin of life is identified with the transition from trivial to non-trivial information processing – e.g. from something akin to a Turing machine capable of a single (or limited set of) computation(s) to a universal Turing machine capable of constructing any computable object (within a universality class) – then a precise point of transition from non-life to life may actually be undecidable in the logical sense. This would likely have very important philosophical implications, particularly in our interpretation of life as a predictable outcome of physical law.[1]

Introduction

On April 1, 2016, Dr Yampolskiy posted the following to his social media accounts: "Google just announced major layoffs of programmers. Future software development and updates will be done mostly via recursive self-improvement by evolving deep neural networks." The joke got a number of "likes" but also, interestingly, a few requests from journalists for interviews on this "developing story." To nonexperts, the joke was not obvious, but why? Why don't we evolve software? A quick search produced no definitive answers, and so this article was born.

In 1859, Charles Darwin[2] and many scholars before him[3,4] have proposed theories to explain the origins of complex lifeforms via natural selection and modification. Scientific theories are algorithms which given as input starting conditions make statistically accurate predictions of the future state of the system. For example, computer simulations of continental drift give us positions of continents at some time t. Yampolskiy emphasizes importance of such simulations:

> A scientific theory cannot be considered fully accepted until it can be expressed as an algorithm and simulated on a computer. It should produce observations consistent with measurements obtained in the real world, perhaps adjusting for the relativity of time scale between simulation and the real world. In other words, an unsimulatable hypothesis should be considered significantly weaker than a simulatable one. It is possible that the theory cannot be simulated due to limits in our current computational capacity, hardware design, or capability of programmers and that it will become simulatable in the future, but until such time, it should have a tentative status.[5]

Simulations of Darwinian algorithm on a computer are known as evolutionary algorithms (EAs) and have been around since the early days of computer science,[6,7] with popular sub-fields such as genetic algorithms (GA), genetic programming (GP), evolutionary strategy (ES), and artificial life (AL). Currently, the state of performance in all of the above-mentioned areas is orders of magnitude less complex than what we observe in the natural world, but why?

A number of seminal papers have been published attempting to formalize Darwin's biological theory from the point of view of computational sciences. Such works essentially see biological evolution as a computational process running on a carbon-based substrate, but which can be run on other substrates. Valiant in his work on evolvability[8] treats Darwinian evolution as a learning process over mathematical functions and attempts to explain quantitatively which artifacts can be evolved with given resources, and which cannot. Likewise, Chaitin[9,10] in his work on metabiology attempts to develop an abstract fundamental mathematical theory of evolution. Wolfram in his, "a New Kind of Science,"[11] attempts to show how rules of computational universe of simple programs can be used to explain some of the biological complexity we observe. Livnat and Papadimitriou[12] analyze sex as an algorithm, in their work on the theory of evolution viewed through the lens of computation.

It is interesting to do a thought experiments and try to imagine what testable predictions Charles Darwin would have made, had he made his discovery today, with full knowledge of modern bioinformatics and of computer science. His predictions may have included the following: (1) simulations of evolution will produce statistically similar results at least with respect to complexity of artifacts produced and (2) if running EAs for as long as possible continued to produce nontrivial

outputs, scientists would run them forever. Likewise, he would be able to make some predictions, which would be able to falsify his theory, such as (1) representative simulations of evolution will not produce similar results to those observed in nature, (2) researchers will not be able to evolve software or other complex or novel artifacts, and (3) there will not be any projects running EAs long term because their outputs would quickly stop improving and stabilize. With respect to the public and general cultural knowledge, it would be reasonable to predict that educated people would know the longest-running EA and the most complex evolved algorithm. Similarly, even schoolchildren would know the most complex digital organism ever evolved.

In the rest of the article, we evaluate the state-of-the-art in relevant published research to see how the above-mentioned predictions and counterfactuals hold up and what we can say about the foundational question of this article. We analyze potential explanations for the current observations of progress in the domain of EAs and look at computational resources, required and available, as the main source of limitations and future opportunities for evolving software.

Evolutionary Computation

Inspired by the Darwin's theory[2] of biological evolution, evolutionary computation attempts to automate the process of optimization and problem-solving by simulating differential survival and reproduction of individual solutions. From the early 1950s, multiple well-documented attempts to make Darwin's algorithm work on a computer have been published under such names as Evolutionary Programming,[13] Evolutionary Strategies,[14] Genetic Algorithms,[15] Genetic Programming,[16] Genetic Improvement,[17] Gene Expression Programming,[18] Differential Evolution,[19] Neuroevolution,[20] and Artificial Embryogeny.[21] Although numerous variants different in their problem representation and metaheuristics exist,[22-25] all can be reduced to just 2 main approaches—GA and GP.

The GAs are used to evolve optimized solutions to a particular instance of a problem such as Shortest Total Path,[26] Maximum Clique,[27] Battleship,[28] Sudoku,[29] Mastermind,[24] Light Up,[30] Graph Coloring,[31] integer factorization,[32,33] or efficient halftone patterns for printers[34] and so are not the primary focus of this article. The purpose of GPs, from their inception, was to automate programming by evolving an algorithm or a program for solving a particular class of problems, for example, an efficient[35] search algorithm. Software design is the type of application most frequently associated with GPs,[36] but work in automated programming is also sometimes referred to as "real programing," "object-oriented GP," "traditional programming," "Turing-equivalent (TE) programming," or "Turing-complete GP."[37,38]

The subfield of computation, inspired by evolution in general, and GP paradigm, established by John Koza in 1990s, in particular, are thriving and growing exponentially as evidenced both by the number of practitioners and of scientific publications. Petke et al[17] observe ". . . enormous expansion of number of publications with the Genetic Programming Bibliography passing 10,000 entries . . . By 2016 there were nineteen GP books including several intended for students . . ." Such tremendous growth has been fueled, since early days, by belief in capabilities of EAs and our ability to overcome obstacles of limited compute or data as illustrated by the following quotes:

> We will (before long) be able to run genetic algorithms on computers that are sufficiently fast to recreate on a human timescale the same amount of cumulative optimization power that the relevant processes of natural selection instantiated throughout our evolutionary past . . .[39]

> As computational devices improve in speed, larger problem spaces can be searched.[40]

> "We believe that in about fifty years' time it will be possible to program computers by means of evolution. Not merely possible but indeed prevalent."[41] "The relentless iteration of Moore's law promises increased availability of computational resources in future years. If available computer capacity continues to double approximately every 18 months over the next decade or so, a computation requiring 80 h will require only about 1% as much computer time (i.e., about 48 min) a decade from now. That same computation will require only about 0.01% as much computer time (i.e., about 48 seconds) in two decades. Thus, looking forward, we believe that genetic programming can be expected to be increasingly used to automatically generate ever-more complex human-competitive results."[42]

> The production of human-competitive results as well as the increased intricacy of the results are broadly correlated to increased availability of computing power tracked by Moore's law. The production of human-competitive results using genetic programming has been greatly facilitated by the fact that genetic algorithms and other methods of evolutionary computation can be readily and efficiently parallelized. . . . Additionally, the production of human-competitive results using genetic programming has facilitated to an even greater degree by the increased availability of computing power, over a period of time, as tracked by Moore's law. Indeed, over the past two decades, the number and level of intricacy of the human-competitive results has progressively grown. . . . there is, nonetheless, data indicating that the production of human-competitive results using genetic programming is broadly correlated with the increased availability of computer power, from year to year, as tracked by Moore's Law.[42]

> . . . powerful test data generation techniques, an abundance of source code publicly available, and importance of nonfunctional properties have combined to create a technical and scientific environment ripe for the exploitation of genetic improvement.[40]

State-of-the-Art With Respect to Predictions

To establish the state-of-the-art in evolutionary computation we examined a number of survey papers[42,43] and seminal results[44-49] looking at produced human-competitive results, as they are meant to represent the greatest accomplishments of the field. Although, on the surface, the results may seem

impressive, deeper analysis shows complete absence of success in evolving nontrivial software from scratch and without human assistance. It is of course necessary to be precise about what it is we are trying to measure or detect, as to avoid disagreements resulting from ambiguity in terms being used.

It may be difficult to formally specify what makes a piece of software nontrivial, but intuitively attractive measure of length of the program expressed as the number of lines of code is not a sufficient indicator of complexity, as it could have an extremely low Kolmogorov[50] complexity. Inspired by the Turing test,[51,52] which is based on inability to distinguish output from a person and a computer, we propose defining nontrivial software as such which would take an average experienced human programmer at least a full hour of effort to produce if given the same problem to solve. If the solution source code could be produced with significantly less effort (eg, 1 minute), it may not be sufficiently complex and the problem may be deemed trivial for our purposes. Our approach to specifying triviality would exclude "Hello World" and most other toy programs/problems from consideration, which is exactly what we wanted to achieve as the main benefit from being able to evolve software would come from ability to replace full-time programmers.

Regarding the other 2 conditions, they are much easier to specify. From "scratch" means that we are not starting with an existing version of a program (but are happy to rely on existing APIs, subject to the nontriviality of all newly produced code). Without human assistance can be interpreted to mean that the programmer is working alone, or a team of programmers is working an equivalent amount of time, for example, 2 programmers would each need at least 40 minutes to solve the problem, which implies a small communication overhead.

Reading early claims about capabilities of EA feels just like reading early predictions from artificial intelligence (AI) literature.[53] Some early success is projected into the future by assuming that the same rate of progress continues and it is claimed that complete success is just years away. However, just like with early AI, the claims are inflated, unsupported, overly optimistic, phrased in deceptive and metaphoric language, and the solutions do not scale to the real-world problems. Perhaps, an EA "winter" is long overdue. Here is how Koza presents the state of the field in 1994:

> . . . in this article, we will present a number of examples from various fields supporting the surprising and counter-intuitive notion that computers can indeed by programmed by means of natural selection. We will show, via examples, that the recently developed genetic programming paradigm provides a way to search the space of all possible programs to find a function which solves, or approximately solves, a problem.[16]

After 16 years, he reports results of what he calls an "extraordinary long experiment":

> An additional order-of-magnitude increase was achieved by making extraordinarily long runs on the largest machine (the 1,000-node machine). . . . The length of the run that produced the two patentable inventions was 28.8 days—almost an order-of-magnitude increase (9.3 times) over the overall 3.4-day average for typical runs of genetic programming that our group had been making at the time.[42]

One quickly realizes that most improvements in the field simply come from using more compute to search progressively larger parts of the solutions space, a result similar to the one expected for random search algorithm.

Here is an example of overhyped and ambiguous reporting of results, from recent work on EA. Researchers Becker and Gottschlich[40] go from naming their paper—"AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms" to abstract "AI Programmer, that can automatically generate full software programs requiring only minimal human guidance." To claiming that "Using AI Programmer, we were able to generate numerous complete software programs." Finally, in experimental results they state what they managed to produce "A generated program that outputs 'hello'" or performs addition operation.[40] But even that is a bit of a hype, "Rather than starting with 'Hello World,' we first had AI Programmer create a more simplistic program that simply output 'hi.' It was successfully after 5,700 generations . . ."[40] Even this trivial 1-liner was not a clean success. "The generated program fulfilled its requirement to output the target text, but interestingly included subsequent random characters, which contained parsing errors, including nonmatching brackets."[40] An identical program but the one printing "I love all humans" took 6 057 200 generations.[40]

Perhaps, it is unfair to pick on this particular article, which is only available as an unreviewed preprint, but we selected it because it is highly representative of the type of work frequently published in GP, and its extremeness makes problems clear to identify. If its title was "Brute Forcing Strings," it would be a reasonable work on that subject, but like so many others, authors claim to "Autonomously Creating Software Programs" using evolutionary computation, a claim which is never substantiated in any published literature on this subject. We are not alone in our skepticism; many others have arrived at exactly the same conclusions:

> We examine what has been achieved in the literature, and find a worrying trend that largely small toy-problems been attempted which require only a few line of code to solve by hand.[38]

> "A literature review has revealed that only a small fraction of the papers on GP deal with evolving TE computer programs, with the ability to iterate and utilize memory, while the majority of papers deal with evolving less expressive logical or arithmetic functions."[38] "We conclude that GP in its current form is fundamentally awed, when applied to the space of TE programs."[38] "Computer code is not as robust as genetic code, and therefore poorly suited to the process of evolution, resulting in a insurmountable landscape which cannot be navigated effectively with current syntax based genetic operators. Crossover, has problems being adopted in a computational setting, primarily due to a lack of context of exchanged code. A review of the literature reveals that evolved

> programs contain at most two nested loops, indicating that a glass ceiling to what can currently be accomplished."[38]
>
> A full understanding of open-ended evolutionary dynamics remains elusive.[54]
>
> There are many problems that traditional Genetic Programming (GP) cannot solve, due to the theoretical limitations of its paradigm. A Turing machine (TM) is a theoretical abstraction that express the extent of the computational power of algorithms. Any system that is Turing complete is sufficiently powerful to recognize all possible algorithms. GP is not Turing complete.[55]

Even a survey of GP community itself produced the following feedback regarding current problems being worked on:

> "Far too many papers include results only on simple toy problems which are often worse than meaningless: they can be misleading";
>
> "(we should exclude) irrelevant problems that are at least 20 years old";
>
> "get rid of some outdated, too easy benchmarks";
>
> "the standard 'easy' Koza set should not be included"
>
> "[it is] time to move on."[37]

In practice, GPs are used in the same way as GAs, for optimization of solutions to particular problems or for function optimization[37,38,55–58] or for software improvement.[59]

Regarding Darwin's hypothetical predictions raised in the introduction, we can state the following:

> **Prediction**. Simulations of evolution will produce statistically similar results at least with respect to complexity of artifacts produced. **Status.** False as of 2018.
>
> **Prediction.** If running EAs for as long as possible continued to produce nontrivial outputs, scientists would run them forever. **Status.** False as of 2018.
>
> **Prediction.** Representative simulations of evolution will *not* produce similar results to those observed in nature. **Status.** True as of 2018.
>
> **Prediction.** Researchers will *not* be able to evolve software or other complex or novel artifacts. **Status.** True as of 2018.
>
> **Prediction.** There will *not* be any projects running EAs long-term because their outputs would quickly stop improving and stabilize. **Status.** True as of 2018.
>
> **Prediction.** With respect to the public and general cultural knowledge, it would be reasonable to predict that educated people would know the longest-running EA, and the most complex evolved algorithm. **Status.** False as of 2018.
>
> **Prediction.** Similarly, even schoolchildren would know the most complex digital organism ever evolved. **Status.** False as of 2018.

Looking at outcomes from the made predictions, we observe that all predictions are false as of 2018 and all counterfactuals are true as of the same year as long as we look only at nontrivial products of evolutionary computations. We are not evolving complex artifacts, we are not running EAs for as long as possible, we are not evolving software, and the public is unaware of most complex products of evolutionary computation. On close examination, all "human-competitive" results turn out to be just optimizations, never fully autonomous programming leading to novel software being engineered.

Possible Explanations

A number of possible explanations for "Why we don't evolve software?" could be considered. We tried to be as comprehensive as possible in this section, but it is possible that we have not considered some plausible explanations:

- Incompetent programmers—It is theoretically possible, but is highly unlikely, that out of thousands of scientists working on evolutionary computation, all failed to correctly implement the Darwinian algorithm.
- Nonrepresentative algorithms—Some[55] have suggested that EAs do not accurately capture the theory of evolution, but of course that would imply that the theory itself is not specified in sufficient detail to make falsifiable predictions. If, however, such more detailed specifications are available to GP believers, it is up to them to implement them as computer simulations for testing purposes, but no successful examples of such work are known and the known ones have not been successful in evolving software.
- Inadequate fitness functions—Fitness function for a complex software product is difficult to outline and specify and may be as complex (or even more complex) as the software we want to evolve as it has to consider all the possible use cases and pass all unit tests. This may be the Achilles heel of GP, but it is also an objection to feasibility of programming in general and GP in particular, as both have to convert software specification into the source code. If human programmers and biological evolution succeed with such constraints, so should Darwinian simulations.
- The Halting problem—Turing proved[60] that it is impossible to determine whether an arbitrary program halts, but this is also a problem for human programmers and could be easily addressed by placing time limits on considered solutions.
- Program correctness—If we require evolved software to be provably correct, this would present a problem as GP does not verify produced designs but only tests them against specific unit tests. Likewise, we cannot rely on automated software verification as it is still an unsolved problem[5] in the general case. This is not really a problem

as most of the human-written software is never proven to be correct and only a small portion of software engineering process relies of formal specification and Test Driven Development.

- Inappropriate solutions—Literature on EA is full of examples[61] of surprising creativity of Darwinian algorithm resulting in solutions which match the letter of design specifications but not the spirit. This is similar to human-produced software and numerous examples of ways in which such software fails the goals of the initial design.[62]
- Insufficient complexity of the environment (not enough data, poor fitness functions)—It is possible that the simulated environment is not complex enough to generate high complexity outputs in evolutionary simulations. This does not seem correct as Internet presents a highly complex landscape in which many self-modifying computer viruses roam.[63] Likewise, virtual world such as Second Life and many others present close approximations to the real world and are certainly more complex than early Earth was:

> A skeptic might insist that an abstract environment would be inadequate for the evolution . . ., believing instead that the virtual environment would need to closely resemble the actual biological environment in which our ancestors evolved. Creating a physically realistic virtual world would require a far greater investment of computational resources than the simulation of a simple toy world or abstract problem domain (whereas evolution had access to a physically realistic real world "for free"). In the limiting case, if complete microphysical accuracy were insisted upon, the computational requirements would balloon to utterly infeasible proportions.[39]

Requiring more realistic environmental conditions may result in an increase in necessary computational resources, a problem addressed in the next bullet.

- Insufficient resources (compute, memory)—From the history of computer science, we know of many situations (speech recognition, NN training), where we had a correct algorithm but insufficient computational resources to run it to success. It is possible that we simply do not have hardware powerful enough to emulate evolution. We will address this possibility in section "Computational Complexity of Biological Evolution and Available Compute."
- Software design is not amenable to evolutionary methods—Space of software designs may be discreet with no continues path via incremental fitness to the desired solutions. This is possible, but this implies that original goals of GP are unattainable and misguided. In addition, because a clear mapping exists between solutions to problems and animals as solutions to environmental problems, this would also imply that current explanation for the origin of the species is incorrect.[64]
- Darwinian algorithm is incomplete or wrong—Finally, we have to consider the possibility that the inspiration behind evolutionary computation, the Darwinian algorithm itself is wrong or at least partially incomplete. If that was true, computer simulations of such algorithm would fail to produce results comparable with observations we see in nature and a search for an alternative algorithm would need to take place. This would be an extraordinary claim and would require that we discard all the other possible explanations from this list.

Perhaps, we can learn some answers from similar historical conundrums. Earliest work on artificial neurons was done in 1943 by McCulloch and Pitts,[65] and although research on Artificial Neural Networks (ANN) continued,[66] until 2010 it would have been very logical to ask: "Why don't artificial neural networks perform as well as natural ones?" Today, deep neural networks frequently outperform their human counterparts,[67,68] but it may still be helpful to answer this question about NN, to see how it was resolved. Stuhlmüller succinctly summarizes answer given by Ghahramani:

> Why does deep learning work now, but not 20 years ago, even though many of the core ideas were there? In one sentence: We have more data, more compute, better software engineering, and a few algorithmic innovations . . .[69]

Consequently, the next section looks at this very likely explanation in detail.

Computational Complexity of Biological Evolution and Available Compute

In the biological world, evolution is a very time-consuming process with estimates for the appearance of early life pointing to some 4 billion years ago and each new generation taking minutes for simple life-forms like bacteria and about 20 years for more complex species, like *Homo sapiens*. Given the timescales involved, it is impossible to replicate full-scale evolution in experimental settings, but it may be possible to do so in computer simulations, by generating new offspring in matter of milliseconds and by greatly expediting necessary fitness evaluation time, potentially reducing a multibillion year natural process to just a few years of simulation on a powerful supercomputer. Others have thought about the same:

> *What algorithm could create all this in just* 10^{12} *steps?* The number 10^{12}—one trillion—comes up because this is believed to be the number of generations since the dawn of life $3.5 \cdot 10^9$ years ago (notice that most of our ancestors could not have lived for much more than a day).[12]

Hamiltonian complexity[70] studies how hard is it to simulate a physical system, where "hard" means that the computational resources required to approximate behavior of the system grow too quickly with the size of the system being simulated, so that no computer can perform the task in reasonable time.[70]

Specifically, in the context of EAs, research effort to establish bounds and improve efficiency is known as evolutionary algorithm theory (EAT).[71] In this section, we will attempt to estimate the computational power of evolution in biosphere, analyze computational complexity of bioinspired EAs, and finally compare our findings with the available and anticipated computational resources; all in the hopes of understanding if it is possible to replicate evolution on a computer, in practice.

Similar attempts have been made by others, for example, Shulman and Bostrom wanted to figure out computational requirements necessary to evolve AI:

> The argument from evolutionary algorithms then needs one additional premise to deliver the conclusion that engineers will soon be able to create machine intelligence, namely that we will soon have computing power sufficient to recapitulate the relevant evolutionary processes that produced human intelligence. Whether this is plausible depends both on what advances one might expect in computing technology over the next decades and on how much computing power would be required to run genetic algorithms with the same optimization power as the evolutionary process of natural selection that lies in our past. One might for example try to estimate how many doublings in computational performance, along the lines of Moore's law, one would need in order to duplicate the relevant evolutionary processes on computers.[39]

By looking at total number of generations, population sizes, DNA storage[72–75] and computation and involved neural information processing it is possible to arrive at broad estimates of computational power behind biological evolution.

> In this way, the biosphere can be visualised as a large, parallel supercomputer, with the information storage represented by the total amount of DNA and the processing power symbolised by transcription rates. In analogy with the Internet, all organisms on Earth are individual containers of information connected through interactions and biogeochemical cycles in a large, global, bottom-up network.[76]

> We have various methods available to begin to estimate the power of evolutionary search on Earth: estimating the number of generations and population sizes available to human evolution, creating mathematical models of evolutionary "speed limits" under various conditions, and using genomics to measure past rates of evolutionary change.[39]

- With respect to the estimates of the storage capabilities of the biosphere we have: "The total amount of DNA contained in all of the cells on Earth is estimated to be about 5.3 x 10^{37} base pairs,[76] equivalent to 1.325 x 10^{37} bytes of information."[77]

> Modern whole-organism genome analysis, in combination with biomass estimates, allows us to estimate a lower bound on the total information content in the biosphere: 5.3×10^{31} ($\pm 3.6 \times 10^{31}$) megabases (Mb) of DNA. Given conservative estimates regarding DNA transcription rates, this information content suggests biosphere processing speeds exceeding yottaNOPS values (10^{24} Nucleotide Operations Per Second).[76]

> Finding the amount of DNA in the biosphere enables an estimate of the computational speed of the biosphere, in terms of the number of bases transcribed per second, or Nucleotide Operations Per Second (NOPS), analogous to the Floating-point Operations Per Second (FLOPS) metric used in electronic computing. A typical speed of DNA transcription is 18–42 bases per second for RNA polymerase II to travel along chromatin templates . . . and elsewhere suggested as 100 bases per second . . .. Precisely how much of the DNA on Earth is being transcribed at any one time is unknown. The percentage of any given genome being transcribed at any given time depends on the reproductive and physiological state of organisms, and at the current time we cannot reliably estimate this for all life on Earth. If all the DNA in the biosphere was being transcribed at these reported rates, taking an estimated transcription rate of 30 bases per second, then the potential computational power of the biosphere would be approximately 10^{15} yottaNOPS (yotta = 10^{24}), about 10^{22} times more processing power than the Tianhe-2 supercomputer . . ., which has a processing power on the order of 10^5 teraFLOPS (tera = 10^{12}).[76]

- To estimate neural information processing of nature, we need to look at the processing power of all neurons in the biosphere:

> There are some 4-6*10^{30} prokaryotes in the world today, but only 10^{19} insects, and fewer than 10^{10} human (pre-agricultural populations were orders of magnitude smaller). However, evolutionary algorithms require not only variations to select among but a fitness function to evaluate variants, typically the most computationally expensive component. A fitness function for the evolution of artificial intelligence plausibly requires simulation of "brain development," learning, and cognition to evaluate fitness. We might thus do better not to look at the raw number of organisms with complex nervous systems, but instead to attend to the number of neurons in biological organisms that we might simulate to mimic evolution's fitness function. We can make a crude estimate of that latter quantity by considering insects, which dominate terrestrial biomass, with ants alone estimated to contribute some 15-20% of terrestrial animal biomass. Insect brain size varies substantially, with large and social insects enjoying larger brains; e.g., a honeybee brain has just under 10^6 neurons, while a fruit fly brain has 10^5 neurons, and ants lie in between with 250,000 neurons. The majority of smaller insects may have brains of only a few thousand neurons. Erring on the side of conservatively high, if we assigned all 10^{19} insects fruit-fly numbers of neurons the total would be 10^{24} insect neurons in the world. This could be augmented with an additional order of magnitude, to reflect aquatic copepods, birds, reptiles, mammals, etc., to reach 10^{25}. (By contrast, in pre-agricultural times there were fewer than 10^7 humans, with under 10^{11} neurons each, fewer than 10^{18} total, although humans have a high number of synapses per neuron.) The computational cost of simulating one neuron depends on the level of detail that one wants to include in the simulation. Extremely simple neuron models use about 1,000 floating-point operations per second (FLOPS) to simulate one neuron (for one second of simulated time); an electrophysiologically realistic Hodgkin-Huxley model uses 1,200,000 FLOPS; a more detailed multicompartmental model would add another 3-4 orders of magnitude, while higher-level models that abstract systems of neurons could subtract 2-3 orders of magnitude from the simple models. If we were to simulate 10^{25} neurons over a billion years of evolution (longer than the existence of nervous systems as

> we know them) in a year's run time these figures would give us a range of 10^{31}-10^{44} FLOPS.[39]

As Darwinian algorithm is inherently probabilistic, it is likely that many runs of the algorithm are required to have just one of them succeed, just like in the case of biological evolution.[78] The number of such simultaneous runs can be estimated from the total size of the search space divided by the average individual computational resources of each run. In the special case of biological evolution, evolving intelligent beings:

> The observation selection effect is that no matter how hard it is for human-level intelligence to evolve, 100% of evolved civilizations will find themselves originating from planets where it happened anyway. . . . every newly evolved civilization will find that evolution managed to produce its ancestors.[39]

So even a successful evolutionary run, with fixed computational resources, does not indicate that used compute would be sufficient in a similar experiment, as subsequent runs may not produce similar results. As Shulman and Bostrom[39] put it, "However, reliable creation of human-level intelligence through evolution might require trials on many planets in parallel, with Earth being one of the lucky few to succeed." Conceivably, "Evolution requires extraordinary luck to hit upon a design for human-level intelligence, so that only 1 in 10^{1000} planets with life does so."[39] Hanson elaborates,

> Many have recognized that the recent appearance of intelligent life on Earth need not suggest a large chance that similarly intelligent life appears after a similar duration on any planet like Earth. Since Earth's one data point has been subject to a selection effect, it is consistent with any expected time for high intelligence to arise beyond about a billion years. Few seem to have recognized, however, that this same selection effect also allows the origin of life to be much harder than life's early appearance on Earth might suggest.[79]

Evolutionary algorithm theory attempts to estimate computational requirements theoretically necessary to run different variants of the Darwinian algorithm. Such estimates are usually made with respect to the size of the input problem, which is difficult to formalize with respect to software generation:

> It is difficult to characterize the complexity of a problem specific to a method of programming. Holding all things constant, you measure what must change as the size of the input instance increases. It is even more difficult to describe the complexity of a problem that can be solved by a program that is *itself* the output of a program, as is the case with the typical GP. In general, this type of question cannot be answered. What *can* be done however, is to compare the information content of a program with the information content of its output and in this way provide a bound on the complexity of that output.[36]

Specifically, "Though it is impossible to classify the complexity of a problem that can be solved by the output program in advance, it *is* possible to relate the amount of information contained in the output program to the GP itself. By applying the theorems from Kolmogorov complexity, it can be shown that the complexity of the output program of a GP using a pseudo random number generator (PRNG) can be bound above by the GP itself.

Theorem. For all strings x, y, if x is the shortest program that outputs y, that is, $K(y) = |x|$, then $K(x) \geqslant K(y) + c$.

Proof. Let x be the shortest program (by definition, incompressible) that outputs y. That is, $K(y) = |x|$. Suppose $K(y) > K(x)$. By substitution, $|x| > K(x)$, which is impossible as x was defined as incompressible."[36]

Next, we attempted to include best estimates for Darwinian algorithm complexity found in literature:

> The performance of an EA is measured by means of the number of function evaluations T it makes until an optimal solution is found for the first time. The reason is that evolutionary algorithms tend to be algorithmically simple and each step can be carried out relatively quick. Thus, a function evaluation is assumed to be the most costly operation in terms of computation time. Most often, results about the expected optimization time $E(T)$ as a function of n are derived where n is a measure for the size of the search space. If a fixed-length binary encoding is used n denotes the length of the bit strings (and the size of the search space equals 2^n).[71]

"[W]ith random mutations, random point mutations, we will get to fitness BB(N) in time exponential in N (evolution by exhaustive search)."[9] There busy beaver function BB(N) = the largest integer that can be named by an N-bit program.

Fitness function evaluation is the most costly procedure in the Darwinian algorithm and is particularly ill defined in the case of software evaluation. How does one formalize a fitness function for something like an operating system, without having to include human users as evaluators? One may be required to rely on human-based genetic algorithms (HBGAs),[80] which would greatly increase time necessary to evaluate every generation and by extension overall simulation time for the run, making it impossible to recapitulate evolution through EAs:

> Essentially, the complexity of an optimization problem for a GA is bound above by the growth rate of the smallest representation [Minimum Chromosome Length—(MCL)] that can be used to solve the problem . . . . This is because the probabilistic convergence time will remain fixed as a function of the search space. All things held constant, the convergence time will grow as the search space grows.[36]

"This means that the size of the search space doubles for every increase in instance size because the number of possible solutions is equivalent to the number 2 raised to the length of the chromosome, 2l."[36]

> By creating a UGP [Universal Genetic Program], we have a single vehicle capable of evolving any program evolvable by a GP. To do this, we treat the first part of the data for the UGP as the *specification* (i.e. the "target" function) for a unique GP. In this way, we can

> implement *any GP*. This does not eliminate the Kolmogorov complexity bound, rather it determines the *hidden constant* in the Kolmogorov complexity bound.[36]

> Because the output complexity includes all individuals from all populations, producing more individuals through larger populations or longer runs must eventually stop producing new solutions because these solutions would necessarily increase the output complexity beyond the finite limit imposed by the GP.[36]

Others have attempted to calculate total "Computational requirements for recapitulating evolution through genetic algorithms."[39]

Given estimates of computational power of biological evolution in the wild and theoretical analysis for computational resources necessary to run a Darwinian algorithm, we will now try to see whether matching compute is currently available, and whether not how soon until it is predicted to be developed. Currently, world's top 10 supercomputers (https://en.wikipedia.org/wiki/TOP500#Top_10_ranking) range from 10 to 125 peta (10^{15}) floating point operations per second of theoretical peak performance. For comparison, Bitcoin network (https://blockchain.info) currently performs around 35 exa (10^{18}) hashes per second, many thousands of times the combined speed of the top 500 supercomputers. Similarly, "Storing the total amount of information encoded in DNA in the biosphere, 5.3×10^{31} megabases (Mb), would require approximately 10^{21} supercomputers with the average storage capacity of the world's four most powerful supercomputers":[76]

> In recent years it has taken approximately 6.7 years for commodity computers to increase in power by one order of magnitude. Even a century of continued Moore's law would not be enough to close this gap. Running more or specialized hardware, or longer runtimes, could contribute only a few more orders of magnitude.[39]

In this section, we looked at estimated computational power of biological evolution and theoretical computational complexity of Darwinian algorithm. In both cases, we found that required computational resources greatly exceed what is currently available and what is projected to be available in the near future. In fact, depending on some assumptions we make regarding multiverse,[81] quantum aspects of biology,[82] and probabilistic nature of Darwinian algorithm such compute may never be available. Mahoney arrives at a similar realization:

> The biosphere has on the order of 10^31 cells (mostly bacteria) . . . with 10^6 DNA base pairs each, encoding 10^37 bits of memory. Cells replicate on the order of 10^6 seconds, for a total of 10^48 copy operations over the last 3 billion years. If we include RNA transcription and protein synthesis as computing operations, then the evolution of humans required closer to 10^50 operations. By contrast, global computing power is closer to 10^20 operations per second and 10^22 bits of storage. If we were to naively assume that Moore's Law were to continue increasing computing power by a factor of 10 every 5 years, then we would have until about 2080 before we have something this powerful.[83]

Others agree,

> The computing resources to match historical numbers of neurons in straightforward simulation of biological evolution on Earth are severely out of reach, even if Moore's law continues for a century. The argument from evolutionary algorithms depends crucially on the magnitude of efficiency gains from clever search, with perhaps as many as thirty orders of magnitude required.[39]

If ". . . one would have to simulate evolution on vast numbers of planets to reliably produce intelligence through evolutionary methods, then computational requirements could turn out to be many, many orders of magnitude higher still . . ."[39] It is hoped by some that future developments in Quantum Evolutionary Computation[84] will help to overcome some of the resource limitations[85] without introducing negative side effects.[86]

Conclusions

Our analysis of relevant literature shows that no one has succeeded at evolving nontrivial software from scratch; in other words, the Darwinian algorithm works in theory but does not work in practice, when applied in the domain of software production. The reason we do not evolve software is that the space of working programs is very large and discrete. Although hill-climbing heuristic–based evolutionary computations are excellent at solving many optimization problems, they fail in the domains of noncontinuous fitness.[87] This is also the reason we do not evolve complex alife or novel engineering designs. With respect to our 2 predictions, we can conclude that (1) simulations of evolution do not produce comparably complex artifacts and (2) running EAs longer leads to progressively diminishing results. With respect to the 3 falsifiability conditions, we observe that all 3 are true as of this writing. Likewise, neither the longest-running EA nor the most complex-evolved algorithm nor the most complex digital organism are a part of our common cultural knowledge. This is not an unrealistic expectation as successful software programs, such as Deep Blue[88] or Alpha Go,[89,90] are well known to the public.

Others have come to similar conclusions:

> It seems reasonable to assume that the number of programs possible in a given language is so inconceivably large that genetic improvement could surely not hope to find solutions in the 'genetic material' of the existing program. The test input space is also, in the words of Dijkstra, "so fantastically high" that surely sampling inputs could never be sufficient to capture static truths about computation.[17]

> . . . computing science is—and will always be—concerned with the interplay between mechanized and human symbol manipulation, usually referred to as "computing" and "programming" respectively. An immediate benefit of this insight is that it reveals "automatic programming" as a contradiction in terms.[91]

Moreover, more specifically,

> Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, and airfoils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.[92]

Even Koza himself acknowledges that it would be highly surprising if his approach could work:

> Anyone who has ever written and debugged a computer program and has experienced their brittle, highly non-linear, and perversely unforgiving nature will probably be understandably skeptical about the proposition that the biologically motivated process sketched above could possibly produce a useful computer program.[16]

We challenge EA community to prove us wrong by producing an experiment, which evolves nontrivial software from scratch and without human help. That would be the only way in which our findings could be shown to be incorrect. Perhaps, reframing the problem in terms of maximizing negentropy of digital organisms, as suggested by Schrödinger[93], Michaelian,[94] and Ulanowicz and Hannon,[95] with respect to negative energy being a fundamental property of all life-forms may produce better results.

On a positive side, the fact that it seems impossible to evolve complex software implies that we are unlikely to be able to evolve highly sophisticated artificially intelligent agents, which may present significant risk to our safety and security.[96-102] Just imagine what would have happened, if the very first time we ran a simulation of evolution on a computer, it produced a superintelligent agent. Yampolskiy[103] has shown that programming as a problem is AI-complete; if GP can solve programming that would imply that GP = AGI (artificial general intelligence), but we see no experimental evidence for such claim. In fact, it is more likely that once we have AGI, it could be used to create an intelligent fitness function for GP and so evolve software. Genetic programming will not be the cause of AI, but a product of it. However, neuroevolution methods for optimizing deep learning architectures and parameters remain a strong possibility for creation of AGI.

Author Contributions

RVY is the only author contributed everything to this work.

REFERENCES

1. Walker SI, Davies PC. The algorithmic origins of life. *J R Soc Interface*. 2013;10:20120869.
2. Darwin C. *The Origin of Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life [Die Entstehung der Arten durch natürliche Zuchtwahl]*. London, England: Leipzig; 1859.
3. Lamarck JBP. *Philosophie Zoologique*. National Museum of Natural History (Jardin des Plantes); 1809.
4. Chambers R. *Vestiges of the Natural History of Creation*. London: John Spriggs Morss Churchill; 1853.
5. Yampolskiy RV. What are the ultimate limits to computational techniques: verifier theory and unverifiability. *Phys Script*. 2017;92:093001.
6. Fogel LJ, Owens AJ, Walsh MJ. *Artificial Intelligence Through Simulated Evolution*. Oxford, UK: John Wiley & Sons; 1966.
7. Barricelli NA. Symbiogenetic evolution processes realized by artificial methods. *Methodos*. 1957;9:143–182.
8. Valiant LG. Evolvability. *J ACM*. 2009;56:3.
9. Chaitin G. Life as evolving software. In: Zenil H, ed. *A Computable Universe: Understanding and Exploring Nature as Computation*. Singapore: World Scientific; 2013:277–302.
10. Chaitin G. *Proving Darwin: Making Biology Mathematical*. New York, NY: Vintage; 2012.
11. Wolfram S. *A New Kind of Science*. Champaign, IL: Wolfram Media, Inc; 2002.
12. Livnat A, Papadimitriou C. Sex as an algorithm: the theory of evolution under the lens of computation. *Commun ACM*. 2016;59:84–93.
13. Back T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, UK: Oxford University Press; 1996.
14. Mayr E. Behavior programs and evolutionary strategies: natural selection sometimes favors a genetically "closed" behavior program, sometimes an "open" one. *Am Scient*. 1974;62:650–659.
15. Davis L. *Handbook of Genetic Algorithms*. New York, NY: Van Nostrand Reinhold; 1991.
16. Koza JR. Genetic programming as a means for programming computers by natural selection. *Stat Comput*. 1994;4:87–112.
17. Petke J, Haraldsson S, Harman M, White D, Woodward J. Genetic improvement of software: a comprehensive survey. *IEEE T Evol Comput*. 2017;22:415–432.
18. Ferreira C. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Vol 21. Berlin, Germany: Springer; 2006.
19. Storn R, Price K. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *J Global Optim*. 1997;11:341–359.
20. Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567; 2017.
21. Stanley KO, Miikkulainen R. A taxonomy for artificial embryogeny. *Artif Life*. 2003;9:93–130.
22. Yampolskiy RV, Ashby L, Hassan L. Wisdom of artificial crowds—a metaheuristic algorithm for optimization. *J Intell Learn Syst Appl*. 2012;4:98–107.
23. Yampolskiy RV, El-Barkouky A. Wisdom of artificial crowds algorithm for solving NP-hard problems. *Int J Bio-Inspir Com*. 2011;3:358–369.
24. Khalifa AB, Yampolskiy RV. GA with wisdom of artificial crowds for solving mastermind satisfiability problem. *Int J Intell Games Simul*. 2011;6:12–17.
25. Lowrance CJ, Abdelwahab O, Yampolskiy RV. Evolution of a metaheuristic for aggregating wisdom from artificial crowds. Paper presented at: Portuguese Conference on Artificial Intelligence; September 8-11, 2015; Coimbra, Portugal.
26. Hundley MV, Yampolskiy RV. Shortest total path length spanning tree via wisdom of artificial crowds algorithm. Paper presented at: 28th Modern Artificial Intelligence and Cognitive Science Conference (MAICS); April 28-29, 2017; Fort Wayne, IN.
27. Ouch R, Reese K, Yampolskiy RV. Hybrid genetic algorithm for the maximum clique problem combining sharing and migration. Paper presented at: 24th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS); April 13-14, 2013; New Albany, IN.
28. Port AC, Yampolskiy RV. Using a GA and wisdom of artificial crowds to solve solitaire battleship puzzles. Paper presented at: 17th International Conference on Computer Games (CGAMES); July 30-August 1, 2012; Louisville, KY.
29. Hughes R, Yampolskiy RV. Solving Sudoku puzzles with wisdom of artificial crowds. *Int J Intell Games Simul*. 2012;7:24–29.
30. Ashby LH, Yampolskiy RV. Genetic algorithm and wisdom of artificial crowds algorithm applied to light up. Paper presented at: 16th International Conference on Computer Games (CGAMES); July 27-30, 2011; Louisville, KY.
31. Hindi M, Yampolskiy RV. Genetic algorithm applied to the graph coloring problem. Paper presented at: 23rd Midwest Artificial Intelligence and Cognitive Science Conference (MAICS); April 21-22, 2012; Cincinnati, OH.
32. Yampolskiy RV. Application of bio-inspired algorithm to the problem of integer factorisation. *Int J Bio-Inspir Com*. 2010;2:115–123.
33. Mishra M, Pal S, Yampolskiy R. Nature-inspired computing techniques for integer factorization. In: Gujarathi AM, Babu BV, eds. *Evolutionary Computation: Techniques and Applications*. Waretown, NJ: Apple Academic Press; 2016:401.

34. Yampolskiy R, Anderson P, Arney J, Misic V, Clarke T. Printer model integrating genetic algorithm for improvement of halftone patterns. Paper presented at: Western New York Image Processing Workshop (WNYIPW); September 24, 2004; Rochester, NY.
35. Yampolskiy RV. Efficiency theory: a unifying theory for information, computation and intelligence. *J Discr Math Sci Cryptogr*. 2013;16:259–277.
36. Rylander B, Soule T, Foster J. Computational complexity, genetic programming, and implications. Paper presented at: European Conference on Genetic Programming; April 18-20, 2001; Lake Como, Italy.
37. White DR, McDermott J, Castelli M, et al. Better GP benchmarks: community survey results and proposals. *Genet Program Evol M*. 2013;14:3–29.
38. Woodward JR, Bai R. Why evolution is not a good paradigm for program induction: a critique of genetic programming. Paper presented at: Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary; June 12-14, 2009; Shanghai, China.
39. Shulman C, Bostrom N. How hard is artificial intelligence? evolutionary arguments and selection effects. *J Consciousness Stud*. 2012;19:103–130.
40. Becker K, Gottschlich J. AI Programmer: autonomously creating software programs using genetic algorithms. arXiv preprint arXiv:1709.05703; 2017.
41. Orlov M, Sipper M. FINCH: a system for evolving Java (Bytecode). In: Riolo R, McConaghy T, Vladislavleva E, eds. *Genetic Programming Theory and Practice VIII*. New York, NY: Springer; 2011:1–16.
42. Koza JR. Human-competitive results produced by genetic programming. *Genet Program Evol M*. 2010;11:251–284.
43. Kannappan K, Spector L, Sipper M, et al. Analyzing a decade of human-competitive ("HUMIE") winners: what can we learn? In: Riolo R, Worzel W, Kotanchek M, eds. *Genetic Programming Theory and Practice XII*. Cham, Switzerland: Springer; 2015:149–166.
44. Clune J, Misevic D, Ofria C, Lenski R, Elena S, Sanjuán R. Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLoS Comput Biol*. 2008;4:e1000187.
45. Ray TS. Evolution and optimization of digital organisms. In: Billingsley KR, Derohanes E, Brown H III, eds. *Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers*, Athens, GA: The Baldwin Press; 1991:489–531.
46. David OE, van den Herik HJ, Koppel M, Netanyahu NS. Genetic algorithms for evolving computer chess programs. *IEEE T Evol Comput*. 2014;18:779–789.
47. Altshuler EE, Linden DS. Wire-antenna designs using genetic algorithms. *IEEE Antenn Propag M*. 1997;39:33–43.
48. Bird J, Layzell P. The evolved radio and its implications for modelling the evolution of novel sensors. Paper presented at: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02); May 12-17, 2002; Honolulu, HI.
49. Hauptman A, Sipper M. GP-EndChess: using genetic programming to evolve chess endgame players. Paper presented at: European Conference on Genetic Programming; March 30-April 1, 2005; Lausanne, Switzerland.
50. Kolmogorov AN. Three approaches to the quantitative definition of information. *Probl Inf Trans*. 1965;1:1–7.
51. Turing A. Computing machinery and intelligence. *Mind*. 1950;59:433–460.
52. Yampolskiy RV. Turing test as a defining feature of AI-completeness. In: Yang XS, ed. *Artificial Intelligence, Evolutionary Computing and Metaheuristics*. Berlin, Germany; Heidelberg, Germany: Springer; 2013:3–17.
53. Armstrong S, Sotala K. How we're predicting AI—or failing to. In: Romportl J, Zackova E, Kelemen J, eds. *Beyond Artificial Intelligence*. Cham, Switzerland: Springer; 2015:11–29.
54. Soros LB, Stanley KO. Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. Paper presented at: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems; July 30-August 2, 2014; Manhattan, NY.
55. Teller A. Turing completeness in the language of genetic programming with indexed memory. Paper presented at: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence; June 27-29, 1994; Orlando, FL.
56. Woodward JR, Johnson CG, Brownlee AE. GP vs GI: if you can't beat them, join them. Paper presented at: Proceedings of the Genetic and Evolutionary Computation Conference; July 20-24, 2016; Denver, CO.
57. Poli R, Langdon WB, McPhee N, Koza J. Field guide to genetic programming, 2008, http://www.gp-field-guide.org.uk.
58. Woodward JR. GA or GP? that is not the question. Paper presented at: Congress on Evolutionary Computation (CEC'03); December 8-12, 2003; Canberra, ACT, Australia.
59. Orlov M, Sipper M. Flight of the FINCH through the Java wilderness. *IEEE T Evol Comput*. 2011;15:166–182.
60. Turing A. On computable numbers, with an application to the Entscheidungsproblem. *P Lond Math Soc*. 1936;2:230–265.
61. Lehman J, Clune J, Misevic D, et al. The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities. arXiv preprint arXiv:1803.03453; 2018.
62. Yampolskiy RV, Spellchecker M. Artificial intelligence safety and cybersecurity: a timeline of AI failures. arXiv preprint arXiv:1610.07997; 2016.
63. Nachenberg C. Computer virus-antivirus coevolution. *Commun ACM*. 1997;40:46–51.
64. Yampolskiy RV. On the origin of synthetic life: attribution of output to a particular algorithm. *Phys Script*. 2016;92:013002.
65. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*. 1943;5:115–133.
66. LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521:436–444.
67. Yampolskiy RV. *Artificial Superintelligence: A Futuristic Approach*. New York: Chapman and Hall/CRC; 2015.
68. Yampolskiy R. *Artificial Intelligence Safety and Security*. Boca Raton, FL: CRC Press; 2018.
69. Stuhlmüller A. 50 things I learned at *NIPS* 2016. NIPS, 2016, https://blog.ought.com/nips-2016-875bb8fadb8c.
70. Osborne TJ. Hamiltonian complexity. *Rep Prog Phys*. 2012;75:022001.
71. Jansen T, Zarges C. Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. Paper presented at: Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms; January 5-9, 2011; Schwarzenberg, Austria.
72. Beck MB, Yampolskiy RV. Hiding color images in DNA sequences. Paper presented at: 26th Modern Artificial Intelligence and Cognitive Science Conference (MAICS); April 25-26, 2015; Greensboro, NC.
73. Beck MB, Desoky AH, Rouchka EC, Yampolskiy RV. Decoding methods for DNA steganalysis. Paper presented at: 6th International Conference on Bioinformatics and Computational Biology (BICoB); March 24-26, 2014; Las Vegas, NV.
74. Beck MB, Rouchka EC, Yampolskiy RV. Finding data in DNA: computer forensic investigations of living organisms. Paper presented at: International Conference on Digital Forensics and Cyber Crime; October 25-26, 2012; Lafayette, LA.
75. Beck M, Yampolskiy R. DNA as a medium for hiding data. *BMC Bioinformatics*. 2012;13:A23.
76. Landenmark HK, Forgan DH, Cockell CS. An estimate of the total DNA in the biosphere. *PLoS Biol*. 2015;13:e1002168.
77. Gillings MR, Hilbert M, Kemp DJ. Information in the biosphere: biological and digital worlds. *Trends Ecol Evol*. 2016;31:180–189.
78. Lenski RE, Rose MR, Simpson SC, Tadler SC. Long-term experimental evolution in *Escherichia coli*. I. adaptation and divergence during 2,000 generations. *Am Nat*. 1991;138:1315–1341.
79. Hanson R. Must early life be easy? the rhythm of major evolutionary transitions. Unpublished Manuscript. 1998, http://hanson.gmu.edu/hardstep.pdf.
80. Kosorukoff A. Human based genetic algorithm. Paper presented at: IEEE International Conference on Systems, Man, and Cybernetics; October 7-10, 2001; Tucson, AZ.
81. De Simone A, Guth AH, Linde A, Noorbala M, Salem MP, Vilenkin A. Boltzmann brains and the scale-factor cutoff measure of the multiverse. *Phys Rev D*. 2010;82:063520.
82. Lambert N, Chen Y-N, Cheng Y-C, Li C-M, Chen G-Y, Nori F. Quantum biology. *Nat Phys*. 2013;9:10–18.
83. Mahoney M. Couldn't we eliminate any and all risk of the artificial general intelligence by making its goal to be the most accurate possible advisor? 2015, https://www.quora.com/Couldnt-we-eliminate-any-and-all-risk-of-the-artificial-general-intelligence-by-making-its-goal-to-be-the-most-accurate-possible-advisor.
84. Han K-H, Kim J-H. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE T Evol Comput*. 2002;6:580–593.
85. Sofge DA. Toward a framework for quantum evolutionary computation. Paper presented at: IEEE Conference on Cybernetics and Intelligent Systems; June 7-9, 2006; Bangkok, Thailand.
86. Majot A, Yampolskiy R. Global catastrophic risk and security implications of quantum computers. *Futures*. 2015;72:17–26.
87. Mishra M, Gupta V, Chaturvedi U, Shukla KK, Yampolskiy RV. A study on the limitations of evolutionary computation and other bio-inspired approaches for integer factorization. *Proced Comput Sci*. 2015;62:603–610.
88. Hsu F-H. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton, NJ: Princeton University Press; 2004.
89. Silver D, Huang A, Maddison CJ, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*. 2016;529:484–489.
90. Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge. *Nature*. 2017;550:354.
91. Dijkstra EW. On the cruelty of really teaching computing science. *Commun ACM*. 1989;32:1398–1404.
92. Mewada S, Sharma P, Gautam S. Exploration of fuzzy system with applications. In: Shah NH, Mittal M, eds. *Handbook of Research on Promoting Business Process Improvement through Inventory Control Techniques*. Hershey, PA: IGI Global; 2018:479–498.

93. Schrodinger E. *What Is Life?: The Physical Aspect of the Living Cell and Mind and Matter; Mind and Matter*. Cambridge, UK: Cambridge University Press; 1967.
94. Michaelian K. Entropy production and the origin of life. *J Mod Phys*. 2011;2:595–601.
95. Ulanowicz RE, Hannon B. Life and the production of entropy. *P Roy Soc Lond B Bio*. 1987;232:181–192.
96. Sotala K, Yampolskiy RV. Responses to catastrophic AGI risk: a survey. *Phys Script*. 2015;90:018001.
97. Babcock J, Kramár J, Yampolskiy R. The AGI containment problem. Paper presented at: International Conference on Artificial General Intelligence; July 16-19, 2016; New York, NY.
98. Pistono F, Yampolskiy RV. Unethical research: how to create a malevolent artificial intelligence. Paper presented at: 25th International Joint Conference on Artificial Intelligence (IJCAI-16); July 9-15, 2016; New York, NY.
99. Majot AM, Yampolskiy RV. AI safety engineering through introduction of self-reference into felicific calculus via artificial pain and pleasure. Paper presented at: IEEE International Symposium on Ethics in Science, Technology and Engineering; May 23-24, 2014; Chicago, IL.
100. Ramamoorthy A, Yampolskiy R. Beyond mad? the race for artificial general intelligence. *ITU J*. 2018;1:1–8.
101. Yampolskiy RV. Taxonomy of pathways to dangerous artificial intelligence. Paper presented at: AAAI Workshop: AI, Ethics, and Society. February 13, 2016; Phoenix, AZ.
102. Yampolskiy RV, Fox J. Safety engineering for artificial general intelligence. *Topoi*. 2013;32:217–226.
103. Yampolskiy RV. AI-complete, AI-hard, or AI-easy—classification of problems in AI. Paper presented at: 23rd Midwest Artificial Intelligence and Cognitive Science Conference (MAICS); April 21-22, 2012; Cincinnati, OH.