Ubiquity Symposium

Evolutionary Computation and the Processes of Life

Opening Statement

by Mark Burgin and Eugene Eberbach

Editor's Introduction

Evolution is one of the indispensable processes of life. After biologists found basic laws of evolution, computer scientists began simulating evolutionary processes and using operations discovered in nature for solving problems with computers. As a result, they brought forth evolutionary computation, inventing different kinds operations and procedures, such as genetic algorithms or genetic programming, which imitated natural biological processes. Thus, the main goal of our Symposium is exploration of the essence and characteristic properties of evolutionary computation in the context of life and computation.

Peter J. Denning
Editor

Ubiquity Symposium

Evolutionary Computation and the Processes of Life

Opening Statement

by Mark Burgin and Eugene Eberbach

Investigation of the essence and inherent traits of computation as a basic technological process in contemporary society have attracted many researchers (cf., for example, Denning [1]; and Dodig-Crnkovic and Burgin [2]). Some did this in an informal setting based on computational and research practice, as well as on philosophical and methodological considerations. Others strived to build adequate mathematical models, assuming that similar to any mature science, computer science could efficiently study its complex objects only by means of mathematics. However, despite all the interest in this problem, the conception of computation remains too vague and ambiguous. For instance, in the computer science community, there is no consensus whether computation is a technological process or it exists in nature or whether Turing machines give an absolute model for computation or there are computing devices more powerful than Turing machines. As we are still far from a sufficient understanding of computation, and we know even less about evolutionary computation, it looks reasonable for the participants of our Symposium to base their answers on the following methodological considerations.

Assuming evolutionary computation is a kind of computation, it is possible to base our understanding of evolutionary computation on one of the three premises. First, it is possible to assume we know/understand what computation is and to explain what specific characteristics differentiate evolutionary computation, finding its distinctions from the general case of computation. Second, it is possible to explore the problem from scratch, aiming at independent portrayal of evolutionary computation. Third, it is possible to elaborate an understanding of evolutionary computation based on some definition of computation chosen by the author. Note that genetic computations are a particular case of evolutionary computations.

In any case, engaging with the question is more valuable than finding a definitive answer. For instance, we can start our discussion with the suggestion of Peter Denning to define evolutionary computation as the computations unfolding under the direction of algorithms that adapt their rules to conditions measured in the computation.

Different structures can be used to answer the main question of our Symposium and related questions formulated at the end of this statement. Such an answer can be a rather general and informal description, reflecting opinions of the author. Another type of an answer can be given as an exact definition, formal or informal. Even a formal definition can have several forms. It may be a system of (formalized) defining properties or a mathematical model of system that performs evolutionary computations. So, the participants are able to choose the form of exposition they prefer.

With this in mind, our opening statement is not an instruction of what to write or how to write, but is a reflection to simulate grounded commentary and creative reactions. The participants may not agree with everything in the opening statement or with what the other participants would like to say. Our goal is to find out important values and better understanding of evolutionary computation, specifying its role in our world saturated with information technology.

History of Evolutionary Computation

The origins of evolutionary computation controlled by evolutionary algorithms date back to the 1940s and 1950s. In the beginning, there were theoretical ideas and results. The great mathematician John von Neumann sought to model one of the most basic life's processes—reproduction—by designing self-reproducing automata, which were later called cellular automata [3, 4]. At the Hixon Symposium in 1948, von Neumann discussed the idea of self-replicating machines, which operated in a very simple environment and had uniform components, each of which was a finite automaton organized in a two-dimensional array. For the building blocks for their physical realization, he decided on computer chips. Later Arbib [5], Codd [6], Gardner [7], Holland [8], and Langton [9] simplified the construction of von Neumann. Independently of von Neumann, Turing [10] proposed to use what is now called genetic algorithms in his unorganized machines.

Later experiments and exploration involved what is now called Artificial Life, or ALife. In it researchers study natural life by creating artificial systems that possess some of the properties of life, such as evolution. In particular, Barricelli [11] wrote a computer program to simulate evolution on a grid of cells because he and other researchers realized they could not give an

analytic answer to the question of what configuration would result from applying the rules repeatedly. Numbers resided in each cell and migrated to neighboring cells based on a set of rules. When two numbers came to the same cell, they competed for survival. Barricelli was interested in studying the emergent properties that would arise from such a simulation. He found even with very simple rules for propagating throughout the environment, certain numeric patterns would evolve and could only persist when other patterns were also present. Barricelli's first experiments already demonstrated something akin to artificial symbiosis. Another famous example of evolutionary simulation is Axelrod's prisoner's dilemma simulations [12].

In evolutionary simulations, researchers simulate some process and watch how it evolves. Then they draw conclusions. Thus, in evolutionary simulation, computers work according to the "clock paradigm" [13] when obtained results are considered only as intermediate, and it is supposed that to get better results it is necessary to make more computational cycles. Super-recursive algorithms provide a mathematical model of this paradigm.

Evolutionary algorithms were also used in the works of Friedman on evolutionary robotics [14]; Box on simulating evolution for industrial plant productivity with the goal to compare different policies and see which ones produce the best results in the long term [15]; and Friedberg, Dunham and North on evolving computer programs [16, 17]. This started application of evolutionary computations to self-programming robots, which can improve their performance by adapting their rules in response to feedback from the environment. It is interesting that success of self-adapting machines inspired Kleene [18] to formulate a conjecture that an algorithm that changes (improves) itself while working can have higher computing or decision power than Turing machines. However, this conjecture was disproved in Burgin's 1992 paper [19], where reflexive Turing machines were introduced as a generic model for programs (algorithms) that change (improve) themselves while they are working. At the same time, Burgin explained successful application of evolutionary self-modifying algorithms proving that self-modifying algorithms are much more efficient than conventional algorithms, being able to outperform any conventional algorithm [19].

Random variation in evolutionary computations can occur in many forms. Even as early as the 1950s, researchers were experimenting with different means for simulating sexual recombination between multiple solutions for studying genetic systems (cf., for example, Fraser [20]). The common form of recombination was a crossover operator that took parts of one solution and matched them up with parts of another.

Genetic algorithms became popular after Holland modelled adaptive systems [21]. Other forms of evolutionary algorithms were suggested by Bremermann, who used a blending recombination with parameters from multiple solutions were averaged and used in evolutionary optimization [22]. When coupled with mutation operators that act on only a single solution, rather than pairs or higher-order couplings, it is possible to generate a robust and often surprisingly efficient search of many complicated solution spaces.

Other forms of recombination were offered in the 1960s, including Lawrence Fogel's suggestion for mating finite state machines as predictors for future environments in generating artificial intelligence [23, 24].

Artificial evolution became a widely recognized optimization method as a result of the work of Rechenberg [25, 26] and Schwefel [27], who employed evolution strategies for evolving physical devices in order to solve complex engineering problems.

Kaufman worked with evolving mathematical systems [28], while Burgin explored evolving solutions to games [29]. Conrad used evolutionary computation for simulating ecosystems [30].

Later new ideas and structures came to the area of evolutionary computation, such as evolving artificial neural networks studied by Yao [31], artificial immune systems, ant colony optimization studied by Dorigo and Stuetzle [32], and particle swarm intelligence studied by Bonabeau, Dorigo, and Theraulaz [33]. Evolutionary computation became a part of emerging computation [34, 35].

Now the main area of evolutionary computation applications are: (1) search methods that work well heuristically but don't need exponential time; (2) simulations of populations to see what patterns emerge over time; and (3) comparisons of policies by using simulations to assess their effects. To achieve these goals four main approaches are used: genetic algorithms, genetic programming, evolution strategies, and evolutionary programming. Additional approaches include ant colony optimization, particle swarm optimization, co-evolution, artificial immune systems, evolutionary robotics, evolvable hardware, behavior engineering, evolutionary artificial neural networks, evolutionary multiobjective optimization, artificial life, classifier systems, DNA-based computing, and some fields of bioinformatics. Applications of evolutionary computation are vast and diverse. They include solutions of intractable (hard and NP-complete) optimization problems, machine learning, data mining, neural network training, robotics, control, electronic circuit design, games, economics, network design, pattern recognition, genome and protein analysis, DNA-based computing, evolvable hardware, and many others.

However, in spite of a diversity of useful applications, evolutionary computation theory is still very young and incomplete [36, 37, 38, 39, 40]. Studied theoretical topics include convergence in the limit (elitist selection, Michalewicz's contractive mapping, convergence rate (Rechenberg's 1/5 rule), building block analysis (schema theorems), best variation operators (no free lunch theorem). Very little has been known about expressiveness or computational power of evolutionary computation and its scalability. Conventional computation has many models. One of the most popular is Turing machine. In contrast to this, until recently evolutionary computation did not have a theoretical model able to represent practice in this domain. As a result, many properties of evolutionary computation processes and results could not be precisely evaluated, studied or even found by researchers. Only recently a rigorous mathematical foundations of evolutionary computation has been created [41, 42, 43, 44, 45, 46] although they provide only the beginning of a rigorous mathematical theory of evolutionary computations. In this theory, evolutionary automata play the role similar to the role of Turing machines, finite automata, and other mathematical models in the general theory of computation.

Duality Between Computation and Algorithms

Answering the question—what is evolutionary computation—it is necessary to take into account the existing duality between computations and algorithms. At first, researchers in the emerging computer science were oriented on exploration and definition of *algorithm*. This orientation became especially strong when in 1930s mathematicians began to believe they had found a comprehensive and in some sense, absolute definition in the form of a Turing machine or equivalent schemas, such as partial recursive functions or post transformative systems. Computation was considered as something automatically attached to an algorithm. However, other models of information systems appeared, which were more powerful than Turing machines. This caused confusion in minds of the researchers and being unable to abandon such a comfortable model as Turing machine in the role of the absolute boundary for algorithms, they started to talk about computations and their models instead of algorithms. However, algorithm and computation are intrinsically connected. According to Denning, in the 1970s Dijkstra explained that an algorithm is a static description of a computation, which is a dynamic state sequence evoked from a machine by the algorithm [1]. Later a more systemic explication of this relation was elaborated. Namely, computation is a process of information transformation, which is controlled by an algorithm, while an algorithm is a system of rules for a computation [47].

Note that a computer program is an algorithm written in (represented by) a programming language. This shows that an algorithm is an abstract structure and it is possible to realize one algorithm as different programs. Moreover, many people think neural networks perform computations without algorithms. However, this is not true because neural networks algorithms have representations that are very different from traditional representations of algorithms as systems of rules/instructions. The neural networks algorithms are represented by neuron weights and connections between neurons. This is similar to hardware representation/realization of algorithms in computers.

This duality between computation and algorithms explains why evolutionary computations have been always connected to evolutionary algorithms. There are two understandings of evolutionary algorithms. In a broad sense, evolutionary algorithms are rules that control any kind of evolutionary computations. This relation explicates duality between evolutionary computations and evolutionary algorithms. In a restricted sense, evolutionary algorithms use selection to operate on populations of individuals that are evolved using mutation and crossover, which are similar to natural genetic processes.

Note that even though now computers that can be modeled by Turing machines simulate evolutionary computation, Turing machines don't give a complete and thorough model of evolutionary computation. Actually evolutionary algorithms are super-recursive algorithms, i.e., algorithms from classes more powerful than Turing machines. More exactly, because mathematical results based on Markov chains indicate that it is possible to make evolutionary algorithms converging to the best possible solutions in the limit [48, 49], evolutionary algorithms are limit algorithms in the sense of [50] i.e., algorithms the result of which is the limit of partial results obtained in the computation.

Tentative Topics and Questions for Consideration

- It is natural to compare evolutionary computations to Turing machine computation. Is it the same, slightly different or essentially distinctive?
- What are the differences between evolutionary computation and emergent computation?
- What are the most important tasks in the area of evolutionary computation?
- What are the most important future directions for evolutionary computation?
- How evolutionary computation may help computation in general?

- What are relations between nature-inspired computation and evolutionary computation?
- What is the role of new computational models in the area of evolutionary computation?
- What is the role of theory in the area of evolutionary computation?
- What are relations between natural computation and evolutionary computation? Are there evolutionary computations in nature? Is any natural computation evolutionary?
- Is DNA transcription into RNA an evolutionary computation? Is RNA translation into protein an evolutionary computation?
- Is self-reproduction of von Neumann's automata an evolutionary computation?
- Is it possible to model all life processes with an evolutionary computation? Is it possible to model all life processes, e.g., all kinds of thinking or emotions, with computers?

In his closing statement to the preceding *Ubiquity* symposium, Peter Denning wrote about an emerging consensus that reactive systems are models of interactive computation and are different from Turing modeled systems [51]. Are evolutionary systems also different from Turing machines or conventional algorithms? In particular, are they more powerful than Turing machines? If it is true, then the question is why? If it is not true, then justify this.

To conclude the authors express gratitude to Peter Denning for his useful advice for improving the style of this statement.

References

[1] Denning, P. What is computation? Opening Statement. *Ubiquity.* October 2010.

[2] Dodig-Crnkovic, G., and Burgin, M. (eds.) *Information and Computation*. World Scientific, New York, 2011.

[3] Von Neumann, J. *Theory of Self-Reproducing Automata*, A. Burks (ed.). University of Illinois Press, Champaign, IL, 1966.

[4] Wolfram, S. *Cellular Automata and Complexity*. Addison-Wesley, Boston, 1994.

[5] Arbib, M.A. Simple self-reproducing universal automata. *Information and Control* 9, 2 (1966), 177-189

[6] Codd, E.F. Cellular Automata. Academic Press, New York, 1968.

[7] Gardner, M. The fantastic combinations of John Conway's new solitaire game "life." *Scientific American* 223 (October 1970), 120-123.

[8] Holland, J.H. Studies of the spontaneous emergence of self-replicating systems using cellular automata and formal grammars. In *Automata, Languages, Development*, A. Lindenmayer and G. Rozenberg (eds.). North-Holland, Amsterdam, 1976, 385-404.

[9] Langton, C.G. Self-reproduction in cellular automata. *Physica D* 10 (1984), 134-144.

[10] Turing, A. Intelligent Machinery (1948). In *Collected Works of A.M. Turing*: *Mechanical Intelligence*. Elsevier Science, 1992.

[11] Barricelli, N.A. *Esempi numerici di processi di evoluzione*. *Methodos* (1954),45-68.

[12] Axelrod, R. The evolution of strategies in the iterated Prisoner's Dilemma. In *Genetic algorithms and simulated annealing*. Pitman, London, 1987, 32-41.

[13] Burgin, M. Theory of super-recursive algorithms as a source of a new paradigm for computer simulation. In *Proceedings of the Business and Industry Simulation Symposium*. Washington D.C., 2000, 70-75.

[14] Friedman, G.J. Selective feedback computers for engineering synthesis and nervous system analogy. Master's thesis. UCLA, 1956.

[15] Box, G. E. P. Evolutionary operation: A method for increasing industrial productivity. *Appl. Statistics* VI (1957), 81–101.

[16] Friedberg, R. M. A learning machine. *IBM J.* 2 (1958), 2–13.

[17] Friedberg, R. M., Dunham, B., and J. H. North. A learning machine: Part II. *IBM J.* 3 (1959), 282–287.

[18] Kleene, S.C. Mathematical logic: Constructive and non-constructive operations. In *Proceedings of the International Congress of Mathematicians* (Edinburgh, Aug. 14-21, 1958). Cambridge University Press, New York, 1960, 137-153.

[19] Burgin, M. Reflexive calculi and logic of expert systems. In *Creative Processes Modeling By Means Of Knowledge Bases*. Sofia, 1992, 139-160.

[20] Fraser, A.S. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences* v.10 (1957), 484-491.

[21] Holland, J.H. Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM* 9 (1962), 297-314.

[22] Bremermann, H. J. Optimization through evolution and recombination. In *Self-Organizing Systems*. Spartan, Washington, D.C., 1962.

[23] Fogel, L. J. Autonomous automata. *Ind. Res.* 4 (1962), 14–19.

[24] Fogel, L.J., Owens, A.J. and Walsh, M.J. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[25] Rechenberg, I. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment, Farnborough, 1965, 11-22.

[26] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*. Dr.-Ing. Thesis. Technical University of Berlin, Department of Process Engineering, 1971.

[27] Schwefel, H.-P. *Kybernetische evolution als strategie der exprimentellen forschung in der strömungstechnik*. Master's thesis. Technical University of Berlin, 1965.

[28] Kaufman, H. An experimental investigation of process identification by competitive evolution. *IEEE Trans. Systems Science and Cybernetics* SSC3-1 (1967), 11-16.

[29] Burgin, G. H. On playing two-person zero-sum games against nonminimax players. *IEEE Trans. Syst. Sci. Cybern.* SSC-5 (1969), 369–370.

[30] Conrad, M. Computer experiments on the evolution of coadaptation in a primitive ecosystem, Ph.D. dissertation. Stanford University, 1969.

[31] Yao, X. Evolving artificial neural networks. *Proceedings of the IEEE* 87, 9 (1999), 1423-1447.

[32] Dorigo, M., and Stuetzle T. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[33] Bonabeau, E., M., Dorigo, M., and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Oxford, 1999.

[34] Forrest, S. (ed.) *Emergent Computation*. MIT Press, Cambridge, MA, 1991.

[35] Simon, M. *Emergent Computation*: *Emphasizing Bioinformatics*. Springer, 2005.

[36] Fogel, D.B. An Introduction to evolutionary computation. Tutorial. Congress on Evolutionary Computation, Seoul, Korea, 2001.

[37] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, third edition. Springer-Verlag, 1996.

[38] Kennedy, J, Eberhart, R., and Shi, Y. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, 2001.

[39] Michalewicz, Z., and Fogel D.B. *How to Solve It: Modern Heuristics*, second edition. Springer-Verlag, 2004.

[40] Macready, W.G., and Wolpert, D.H. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1 (1997), 67-82.

[41] Eberbach, E. Toward a theory of evolutionary computation. *BioSystems* 82 (2005), 1-19.

[42] Burgin, M., and Eberbach, E. Cooperative Combinatorial Optimization: Evolutionary computation case study. *BioSystems* 91 (2008), 34-50.

[43] Eberbach E., and Burgin M. Evolutionary automata as foundation of evolutionary computation: Larry Fogel was right. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation* (Trondheim, Norway, MAY 18-21). IEEE Press, Piscataway, NJ, 2009, 2149-2156.

[44] Burgin, M., and Eberbach, E. On foundations of evolutionary computation: An evolutionary automata approach. *Handbook of Research on Artificial Immune Systems and Natural Computing*. IGI Global, 2009, 342-360.

[45] Burgin, M., and Eberbach, E. Universality for Turing machines, inductive Turing machines and evolutionary algorithms. *Fundamenta Informaticae* 90 (2009a), 1-25.

[46] Burgin M., and Eberbach E. Evolutionary automata: Expressiveness and convergence of evolutionary computation. *Computer Journal* (2011). (DOI:10.1093/comjnl/bxr099)

[47] Burgin, M. *Superrecursive Algorithms*. Springer, New York, 2005.

[48] Fogel, D.B. *Evolving artificial intelligence*. Ph.D. dissertation. UC San Diego, 1992.

[49] Rudolph, G. Convergence properties of canonical genetic algorithms. *IEEE Transactions on Neural Networks* 5 (1994), 1994.

[50] Burgin, M. Topological algorithms. In *Proceedings of the ISCA 16th International Conference* "*Computers and their Applications*" (Seattle, WA, Mar. 28-30, 2001). 61-64.

[51] Denning, P. What is computation? Closing Statement. *Ubiquity*. April 2011.

About the Authors

Dr. Mark Burgin received his M.A. and Ph.D. in mathematics from Moscow State University and Doctor of Science in logic and philosophy from the National Academy of Sciences of Ukraine. He was a Professor at Institute of Education, Kiev; at International Solomon University, Kiev; at Kiev State University, Ukraine; and the Head of the Assessment Laboratory, Research Center of Science at the National Academy of Sciences of Ukraine. Currently he is working at UCLA. Dr. Burgin is a member of New York Academy of Sciences and a senior member of IEEE and of the Society for Computer Modeling and Simulation International. He is the chief editor of the journal *Integration*, chief editor of the journal *Information*, an associate editor of *Ubiquity* and of the *International Journal on Computers and their Applications*. Dr. Burgin is doing research, has publications, and taught courses in mathematics, computer science, information sciences, artificial intelligence, philosophy, methodology of science and logic. His research areas in computer science and artificial intelligence include theory of algorithms and computation, complexity, evolutionary computations, mathematical theory of information technology, software metrics, software correctness, concurrent computation and distributed systems, computer modeling and simulation, interactive computation, computational schema theory, algorithmic information theory, general theory of information, mathematical logic, knowledge representation, knowledge acquisition and data mining, knowledge and information dynamics, representational schema theory, non-monotonic reasoning, and inconsistent knowledge systems. His recent publications include such books as *Theory of Information* (2010), *Measuring Power of Algorithms, Computer Programs, and Information Automata* (2010), and *Super-recursive Algorithms* (2005). He originated such theories as the general theory of information, mathematical theory of technology, system theory of time, theory of logical varieties, theory of named sets and neoclassical analysis (in mathematics) and made essential contributions to such fields as foundations of computer science, theory of algorithms and computation, information theory, theory of knowledge, theory of negative probabilities, theory of intellectual activity, and complexity studies.

Dr. Eugene Eberbach is Professor of Practice at Department of Engineering and Science, Rensselaer Polytechnic Institute. He has Ph.D. (1982) and M.Sc. and Eng. (1977) degrees both from Warsaw University of Technology. He held various academic and industrial positions in the U.S., Canada, United Kingdom, and Poland, including the University of Massachusetts Dartmouth, Acadia University, University of Memphis, University College London, Rzeszow University of Technology, WSK “PZL-Rzeszow” and, Applied Research Lab, Penn State University. In the late 1980s he worked on new non-von Neumann fifth generation computer

architectures at University College London. From 1990 onwards he worked on distributed autonomous underwater vehicles with support of ONR. In Canada and the U.S. he introduced Calculus Of Self-Modifiable Algorithms and $-Calculus process algebra for automatic problem solving under bounded resources with support of NSERC and ONR. He proposed two super-Turing models of computation: $-Calculus and Evolutionary Turing Machines. His current work is in the areas of process algebras, resource bounded optimization, autonomous agents and mobile robotics. General topics of interest are new computing paradigms, languages and architectures, distributed computing, concurrency and interaction, evolutionary computing, and neural nets. Dr. Eberbach is the author of more than 150 publications in the above areas and he has been a recipient of 17 external research grants.

DOI: 10.1145/2351436.2351437

