# Evolutionary Computation

Lee Altenberg

*The Konrad Lorenz Institute for Evolution and Cognition Research, Martinstrasse 12, Klosterneuburg, Austria A3400*

**Abstract**

Evolutionary computation is a method of solving engineering problems using algorithms that mimic Darwinian natural selection and Mendelian genetics, applied especially to optimization problems that are difficult to solve from first principles. Earliest beginnings were in the 1950s, and by the mid 1990s it had developed as an academic field with its own journals, conferences, and faculty. Several phenomena discovered in evolutionary biology were also discovered in parallel in evolutionary computation, including the evolvability problem, genetic modification, constructive neutral evolution, and genetic robustness. The related field of artificial life focuses on computational systems in which replication, natural selection, and ecological interactions are all emergent.

*Keywords:* crossover, encoding, evolutionary algorithm, evolvability, genetic algorithm, genetic operator, genetic programming, No Free Lunch theorems, objective function, optimization, representation, search space, selection operator, simulated annealing

## 1. Glossary

**A-Life** "Artificial Life" — computational systems in which replication, natural selection, and ecological interactions are not set by the designer but emerge from the dynamics of the system.

**EA** "Evolutionary Algorithm" — an algorithm (sequence of mathematical operations on a data set, a computer program) that incorporate analogs of (1) natural or artificial selection and (2) variation generation.

**EC** "Evolutionary Computation" — the name for the entire research and engineering fields centered on evolutionary algorithms.

**EDA** "Estimation of Distribution Algorithm" — An EA where the dynamics are not on populations but rather on models of probability distributions

*Email address:* altenber@hawaii.edu ()
*URL:* dynamics.org/Altenberg/ ()

over the search space, which are updated with the goal of concentrating probability on the fittest candidate solutions.

**EP** "Evolutionary Programming" — evolutionary algorithms where the search space is a set of finite-state automata, originated by Fogel (1964).

**ES** "Evolution Strategies" — a class of evolutionary algorithms originated by Rechenberg (1964) and Schwefel (1965) where the search space is Euclidean space and the variation operators are Gaussian random perturbations.

**GA** "Genetic Algorithm" — an evolutionary algorithm that incorporates recombination as a variation operator, originated by Holland (1975), emphasizing homologous crossover on bit string encodings of the search space.

**GP** "Genetic programming" — an evolutionary algorithm in which the objects being evolved are executable data structures, originated by Koza (1990), where programs are represented as parse trees of operator and variable symbols, and the principal variation operator is exchange of subtrees between programs evolved in a population.

**NFL** "No Free Lunch" Theorems — the result from Wolpert and Macready (1995) that all search algorithms have equal average performance when averaged over all problem sets.

**Encoding** A representation of points in the search space that may change from a native representation to an alternate representation, such as encoding real numbers as a binary sequence.

**Fitness** In EC, "fitness" is used as a shorthand for "value of the objective function," rather than the population genetics usage where it means the expected number of offspring.

**Fitness Proportional Selection** Selection in which the expected number of offspring of a candidate solution is proportional to the value of its (possibly rescaled) objective function.

**Objective Function** A measure of how good a candidate solution is to producing a desired outcome.

**Representation** The way that candidate solutions in a search space are mapped to the data structures that represent them in the algorithm.

**Search space** The collection of all candidate solutions to be searched through by the evolutionary algorithm.

**Simulated Annealing** An evolutionary algorithm with a population of one parent plus one mutant offspring (1+1 EA), where the offspring replaces a less fit parent with probability one, and it replaces a more fit parent with probability $e^{-(w_P - w_O)/T}$, where $w_P$ is the parent fitness, $w_O$ is the offspring fitness, and $T$ is a "temperature" which is successively lowered during a run of the search (the "annealing").

## 2. Body text

### *2.1. Introduction*

Evolutionary computation is an approach to engineering and optimization in which solutions, instead of being constructed from first principles, are instead *evolved* through processes modeled after the elements of Darwinian evolution. Evolutionary computation is one of the principal methods in what is called "nature-inspired computing". Nature-inspired computing, which also includes artificial neural networks, swarm intelligence, and fuzzy logic, has also been called "soft computing", or "computational intelligence" to distinguish it from symbolic artificial intelligence. Technically, evolutionary computation is as an example of heuristic search, i.e. search by trial and error, where in EC the 'trials' are candidate solutions, and the 'error' is the measurement of how far a trial is from a desired outcome. The error is used to select which trials are to be used to generate further trials. The fundamental rule-of-thumb is that the best chance to further reduce the error is to generate new trials by making modifications to the previous trials that had the lowest errors.

The *evolutionary algorithm* is the main object of interest in evolutionary computation. There is a problem to be solved, and the solution is conceived to lie somewhere in a space of possible candidate solutions—the *search space*. The evolutionary algorithm searches for good solutions in the search space using this typical structure:

1. **Initialization**: Randomly generate a population of samples from the search space.
2. **Iteration**:
   (a) **Evaluation**. Compute the value of the objective function for each sample.
   (b) **Selection Operator**: Use the values of objective function computed for the evaluated samples to select the samples to be used in the next step 2c.
   (c) **Variation Operators**. Apply variation operators to the selected samples to transform them into additional samples from the search space.
3. **Termination**: If the termination criteria are met, halt the computation; if not, return to step 2a.

The problem to be solved usually determines in an obvious way what the search space is, and what the objective function is. For example, if one is trying to find the maximum value of $f(x, y) = \sin(x^2 - 2x - 4)\cos(-3y + y^2 + 1)$ on the intervals $-1 \leq x \leq 1$ and $0 \leq y \leq 1$, the search space is the simply the intervals, and the objective function is $f(x, y)$ itself. To give a more elaborate example, when trying to evolve artificial neural networks that implement a desired map between inputs and outputs, the search space is the set of weights and topology

of the network connections. The objective function is a measure of how closely a candidate map matches a desired map, using a measure of closeness such as mean squared error over a test set of inputs.

Exploration of the search space is performed by using previously sampled and evaluated points to generate new points to be evaluated. The points in the search space are concretely represented by some data structure in the algorithm, and variation operators that randomly perturb these data structures generate the new points.

The classical variation operators are mutation and crossover (i.e. recombination). Mutation is implemented by altering the state of symbols in a string, such as flipping a bit in a binary string from a 0 to a 1 or vice versa. Crossover is implemented by recombining parts of one parent with parts of another. Crossover may be analogous to homologous chromosomal recombination when the positions in the data structure are fixed, as in a fixed-length bit string, or in a fixed set of real-valued parameters. But there are other search spaces where it is impossible to make the analogy to linear chromosomes. For example, in the Traveling Salesman problem, the search space consists of all tours of a set of cities. Taking parts of one tour and recombining them with parts of another tour is likely to produce a path that repeats some cities and misses others. Special "mutation" operators that ensure that a path is a tour are needed.

Choosing which samples to apply the variation operators to is the task of the selection operator. Here, a population structure needs to be introduced. In what is called the "1+1 EA," the population consists of a single sample of the search space. A new sample is generated by applying a variation operator to that sample, to generate a population of size 2. The values of the objective function for the two samples are evaluated and compared, and the sample with the better objective function is selected to generate the next sample. Larger populations may be used, in which case they may maintain enough variation so that recombination may be used as a variation operator. Many different selection operators are possible in large populations. Both the Wright-Fisher multinomial sampling model and the Moran sampling model have been used.

The selection operators that carry out the computational analog of selective breeding may in fact be the exact same operators used in agriculture, such as culling, or ranking.

### *2.2. History of Evolutionary Computation*

Evolutionary computation developed in three phases: the first phase consisted of numerous independent experiments in evolutionary algorithms in the 1950s and 1960s. Principal researchers during this period were Nils Aall Barricelli in the early 1950s, George E. P. Box and Alex S. Fraser in the late 1950s, and Hans J. Bremermann in the early 1960s.

The second phase began in the late 1960s and early 1970s where, out of these multiple lines of investigation emerged three lineages that persisted and produced academic offspring: Lawrence Fogel (1964) established *evolutionary programming*, Ingo Rechenberg (1964) and Hans-Paul Schwefel (1965) established

*evolution strategies*, and John H. Holland (1975) established *genetic algorithms*. Deriving from the genetic algorithm lineage, John Koza (1990)established the field of *genetic programming*.

The third phase came when these four lineages, which had established their own conferences, merged in the mid 1990s into a unified evolutionary computation community. Out of this convergence the field gained its first journals, *Evolutionary Computation* at MIT Press in 1993, *IEEE Transactions on Evolutionary Computation* in 1997, and *Genetic Programming and Evolvable Machines* at Springer in 2000. Before the advent of these journals, research in evolutionary computation appeared mainly in the proceedings of the International Conference on Genetic Algorithms, the Evolutionary Programming conferences, the Parallel Problem Solving from Nature conferences in Europe, and scattered articles throughout academic journals. Because the journal citation index Web of Science does not currently index many of these sources, a large body of the founding literature in evolutionary computation remains invisible to it.

The founding papers in the field from 1953 to 1997 have been collected by David B. Fogel (1998) into a single volume, *Evolutionary Computation: The Fossil Record*.

The history of evolutionary computation differs markedly from that of the field of *artificial intelligence*. Artificial intelligence had a specific birth as a new field in a proposal for a 1956 Dartmouth conference organized by Dartmouth mathematician John McCarthy, Marvin Minsky at Harvard, Nathaniel Rochester at IBM, and Claude Shannon at Bell Labs (McCarthy et al., 1955 (2006). John Holland, it should be noted, was one of the invitees. In the founding of evolutionary computation, the Ivy League played almost no role, and the research took root in diverse state universities and industrial research labs in the United States, and universities in Europe and Australia.

Evolutionary computation has developed largely independently from the established field of theoretical population genetics. Yet interchanges between the evolutionary computation and evolutionary biology communities occurred as early as 1980 in Europe during a workshop on the "Evolution of Evolutionary Mechanisms" held in Göttingen, West Germany (Wagner, 1981), while in the United States, interactions between the two communities were fostered by the advent of the Santa Fe Institute, in which John Holland from the University of Michigan and evolutionary theoretician Marcus W. Feldman from Stanford University were key participants. Paixão et al. (2015) have made a recent effort enhance the flow of theoretical results between fields by creating a modeling framework that encompasses both evolutionary algorithms and computational models of evolution.

### *2.3. Variety in Evolutionary Computation Techniques*

Designers of evolutionary algorithms have explored a very large space of techniques in the hope of improving the performance of evolutionary algorithms over the very wide variety of engineering problems to which they have been applied. Explorations include the values of algorithm parameters such as mutation and

crossover rates, selection strength, and population size, encodings, spatial population structure, generation overlap, niching, fitness sharing, multi-objective optimization methods, feasibility constraints, etc. The theory of evolutionary dynamics has yet to reach the point where one can convert the information one has about an optimization problem into the choices of which techniques and parameters to use. For specific models however, progress is being made in obtaining rigorous theoretical results (portal papers include Oliveto et al. (2007); Valiant (2009)).

Evolutionary algorithms employ a great palette of special operators which may have no analogy in biology. One area under extensive development is multi-objective evolutionary optimization. Another example that has burgeoned into a research program is the method called *estimation of distribution* algorithms (EDAs) introduced by Mühlenbein and Paass (1996). An early method was to merge all of the genomes in the population into a single gene pool and sample the alleles at each locus to generate a new genotype, so that the entire population serves as the parent (Syswerda, 1993). EDAs derive from an abstract view about what the operators of selection, mutation, and recombination are actually doing: they produce (implicitly) a probability distribution from which the genotypes of offspring are sampled. EDAs produce these probabilities explicitly, as a dynamical system on probability models whose parameters are adjusted by the objective function values of sampled candidate solutions, in the hope of concentrating the probability on the best candidate solutions. EDAs readily incorporate techniques from machine learning to generate the probability models (Lozano et al., 2006; Pelikan et al., 2015).

A significant development in evolutionary computation came with the advent of genetic programming, because the data structures representing candidate solutions are no longer of fixed length, but can grow in complexity. Genetic programming evolves code structures that are themselves executable. Programs can be represented as parse trees, and pairs of parse trees can be recombined by exchanging their subtrees. Figure 1 shows an illustration of how a mathematical function can be represented as a parse-tree in genetic programming, and how subtree exchange between two copies of a program (in this case, for a Gaussian distribution) produces an offspring that encodes a different function. Many other techniques have been developed for variation operators in genetic programming, for example, by placing the executable structures back one step from the variation operators through "cellular encoding" (Gruau, 1994) or "developmental encoding" (Mouret and Tonelli, 2014), in which the executable structures are constructed from the encoding through an ontogenic development process.

### *2.4. Contrasts Between Evolutionary Computation and Organic Evolution*

The motivation behind the development of evolutionary algorithms is optimization — finding approximate or exact solutions to diverse problems within the constraints of the computational resources available. The amount of computation required to find an optimum or to achieve a desired level of approximation is therefore a central concern. Such rate of improvement per computation can be considered a measure of *evolvability*. In organismal evolution, everything is

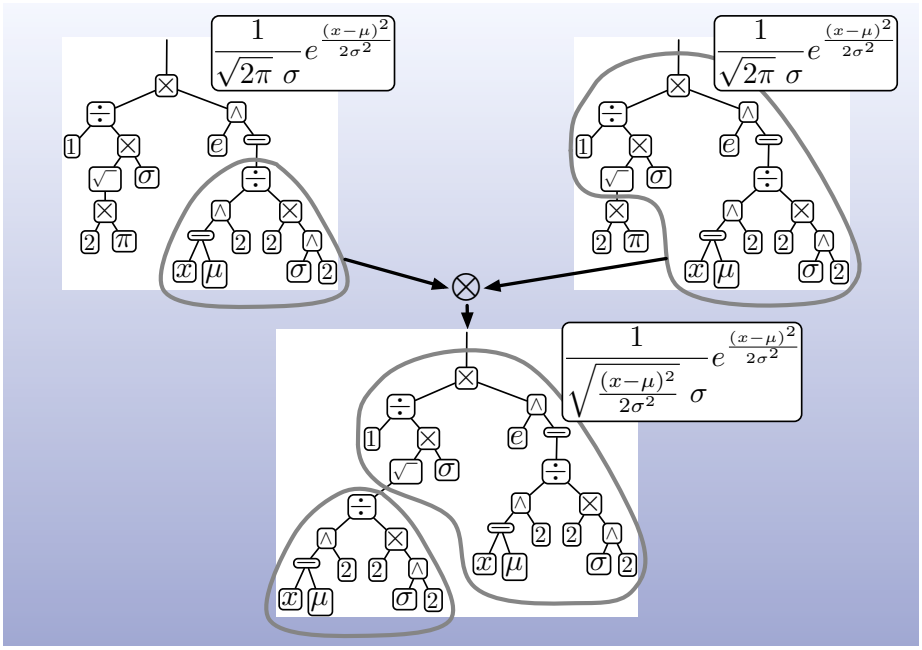

Figure 1: Genetic programming subtree exchange between two parse trees. Two identical parent trees encode the algorithm for the Gaussian normal density. Their recombinant offspring encodes a new mathematical expression.

conditioned on the survival of the lineage, as pointed out by Palmer and Feldman (2012), and evolvability may conflict with survival if it is tied to increased production of deleterious variation, or leads to the evolution of short-sighted traits that sabotage long-term survival (Nunney, 1989; Altenberg, 2005) or reduces total population size (Frank, 2013). Lineage survival is precluded as an element of evolutionary optimization because the algorithms are fundamentally search algorithms, and a 'population' is simply the set of previously sampled points from which new samples are to be generated. In other words, population persistence is guaranteed by construction. Evolvability is thus the primary performance goal of evolutionary algorithms.

In terms of sheer computational power, computer-based evolutionary algorithms cannot compare with the number of evolutionary operations taking place in the biosphere (reproduction, mutation, recombination, natural selection). There are an estimated $3 \times 10^{27}$ cells of the ocean cyanobacterium *Prochlorococcus marinus* alive at any one time (Flombaum et al., 2013) (on the order of the number of atoms in a ton of gold), and with a once-daily cell division, a genome size of $2 \times 10^6$ base pairs, and an estimated mutation rate of $5.4 \times 10^{-7}$ mutations per base pair (Osburne et al., 2011), there are some $3 \times 10^{27}$ mutations generated daily. That is enough to sample all of the 4-mutation neighbors of a parental sequence. The raw information content of organismal genomes, which

ranges on the order of $10^5$ to $10^{10}$ bits, is matched by very few instances of the data structures searched with evolutionary algorithms. The current record is set by some special examples in which genetic algorithms have been performed on one-billion-bit strings (Iturriaga and Nesmachnow, 2012).

A run of an evolutionary algorithm typically begins with a population constructed from uniformly sampled random points in the search space. In organic evolution, by contrast, the initial condition was the origin of life itself. At some point life evolved into organisms with DNA-based genomes, but none of the scenarios for the origin of nucleotide-based genomes propose that the initial sequences were uniformly sampled random sequences. So the initial stage in a run of an evolutionary algorithm, where recombination, mutation, and selection are being performed on a population of random sequences, has no parallel in organic evolution, where the processes are always observed in populations that have been evolving for a very long time.

In evolutionary algorithms, there is a clean separation between logical parts of the algorithm: the code for the objective function and the variation and selection operators is entirely separate from the code for the data structures representing the search space, and the entire algorithm is contained in application code separate from the operating system and hardware of the computer. In organic evolution we see the exact opposite: these logical elements are all physically enmeshed—the molecular machinery that produces mutation and recombination and organismal survival and reproduction is all a product of the information in the genome, cytoplasm, and whole organism, acting in the context of its environment. Some efforts have gone in to blurring the logical separations in evolutionary algorithms, notably the encoding of mutation and recombination rates into the genomes of the individual (see *Genetic Modifiers*, below). The field of *artificial life* distinguishes itself from evolutionary algorithms in that it places a priority on getting reproduction, replication, mutation, recombination, fitness, spatial movement, and ecological interactions to all emerge from the computational system instead of being written by hand in separate parts of the program, emulating the enmeshment of logical levels found in biology (see *Artificial Life*, below).

### *2.5. Contrasts Between Evolutionary Computation and Computational Models of Evolution*

When evolutionary algorithms are compared with computational models of evolution, the most obvious difference is that computational models of evolution place a priority on biological realism, while evolutionary algorithms place a priority on computationally efficient optimization. As "nature inspired" computing, EC has drawn upon many biological and population processes as a source of ideas to try out in algorithmic form, but little priority is given to keeping them biologically realistic.

In the drive toward more biological realism, computational models of evolution have gone beyond the rather limited set of constructs used traditionally in population genetics models, such as additive, multiplicative, low-order epistatic, random, and Gaussian fitness functions. Kauffman and Levin (1987)

introduced the more structured, tunably-rugged NK fitness landscapes. Knibbe et al. (2007) included insertions, deletions, and duplications in a model with a complex genotype-phenotype map. Computational models of RNA folding (Schuster et al., 1994), protein folding (Lobkovsky et al., 2011), and gene regulatory networks (Wagner, 1994) have provided biologically grounded fitness landscapes for exploring evolutionary dynamics. More recently, laboratories have been able to obtain fitness estimates for small regions of fitness landscapes, and computational models of evolution on these empirical fitness landscapes have been utilized to explore and even predict evolution (Visser and Krug, 2014).

The fitness landscapes encountered in evolutionary algorithms are derived from a vast variety of real engineering problems, which may be seen as "wild" rather than constructed. Evolution on these fitness landscapes ranges beyond the standard population genetics models and the special cases explored thus far in computational models of evolution. They may therefore have value as models to expand the palette of observed evolutionary phenomena.

### *2.6. Parallel Discoveries of Evolutionary Phenomena in Evolutionary Computation*

There are several phenomena that were discovered in the field of evolution in parallel with their discovery in the evolutionary biology community. Here, several principal examples will be given, under the names given to the phenomena in the evolutionary biology literature.

**The Problem of Evolvability.** The earliest experiences with evolutionary algorithms showed that the combination of Darwinian selection with random variation does not necessarily work to produce adaptation. Friedberg (1958); Friedberg et al. (1959) attempted to evolve computer programs through selection and variation operators, but found that the process frequently stagnated. Conrad (1974a) pointed out that in computer programs, "slight changes in such a rule (e.g. in the pattern of internal inputs) result in radical changes in the behaviour of the system." In organisms, the physical nature of molecular dynamics makes gradual changes in function possible, and Conrad even proposed that gradualism could itself be increased in evolution (Conrad, 1974b).

But even where gradualism is abundant, epistasis, multi-modality, and deception can stymy evolutionary algorithms from finding global optima or even good approximations. Good performance of evolutionary algorithms was found early on to depend critically on the encoding or representation of the search space, and on parameters of the algorithm such as mutation rate, population size, and selection operators.

A key theoretical breakthrough was the paper, "No Free Lunch Theorems for Search" by Wolpert and Macready (1995, 1997). They showed that over the space of all problems, all search algorithms have the same average performance. The only way that a search algorithm such as an evolutionary algorithm can have superior performance is if the set of problems is

matched to the algorithm, or said in another way, that the algorithm has implicit knowledge about the search space. Concretely, the variation operators have to be able to generate candidate solutions with ever better objective function values as the population evolves. This depends on how the variation operators acting on the representations relate to the objective function (Altenberg, 1995).

**Genetic Modifiers** In 1967, Nei (1967) introduced a model for the evolution of recombination in which a modifier locus controls the rate of recombination between two other loci under viability selection. Feldman (1972) gave the first evolutionary stability analysis of the model. Simultaneously, on the evolutionary computation side, Reed et al. (1967) included genetic modification for mutation and crossover rates and mutation size in their genetic algorithms. Rechenberg (1973) began to incorporate genetic control of the "strategy parameters" for his evolutionary algorithms, which included mutational step sizes, and this was termed "self-adaptation" by Schwefel (1987).

**Constructive Neutral Evolution.** An early discovery in the field of genetic programming was that the size of evolved programs would keep growing in time, even though there was no explicit selection for larger programs. This was dubbed "bloat". One of the explanations proposed in Langdon and Poli (1997) to account for code bloat in genetic programming was that the number of long programs that could implement a given function was much greater than the number of short programs. Offspring produced by exchanging subtrees of parent programs would often have identical fitness to the best parent, and so evolution along neutral networks was possible. Longer programs would evolve as an entropic phenomenon of evolution along neutral networks. As described by Langdon and Poli (1997):

> In general variable length allows many more long representations of a given solution than short ones of the same solution. Thus (in the absence of a parsimony bias) we expect longer representations to occur more often and so representation length to tend to increase. That is fitness based selection leads to bloat.

On the evolutionary biology side, Covello and Gray (1993) proposed that RNA editing may have evolved through a mechanism that Stoltzfus (1999) named "constructive neutral evolution," in which needlessly complex mechanisms evolve through a process of neutral evolution simply because there may be a much greater number of complex ways to produce a phenotype than simple ways. Stoltzfus (1999) includes in this list the phenomenon of "scrambled genes" in ciliates, and the process independently called "subfunctionalization" by Force et al. (1999) in which independently functioning modules within a gene become separated into multiple genes, another essentially entropic process. Echoing the explanation by Langdon and Poli (1997) of bloat, we find the explanation by Stoltzfus (1999) for gene scrambling:

Given such a buffer against the otherwise adverse effects of micronuclear gene rearrangements, a long-term net increase in scrambling would be expected, simply because there are many more scrambled than unscrambled configurations.

**The Evolution of Genetic Robustness.** As described above, program bloat is one of the emergent properties discovered in evolving computer programs through genetic programming. Before Langdon and Poli (1997) proposed their entropic mechanism for bloat, an earlier hypotheses was that the code bloat gave programs a "defense against crossover" (Singleton and Keenan, 1993). They discovered that much of the code in bloated programs could be deleted or exchanged without consequence for the program's behavior. The bloated programs were thus phenotypically robust to the variation operator, and the extra code diverted the crossover operator from hitting sensitive parts of the program. Altenberg (1994) proposed that inherently-neutral code would proliferate in the long-term evolution of populations of programs. Nordin and Banzhaf (1995) examined the distribution of fitness effects of subtree exchange (Figure 2) and found that, indeed, the proportion subtree exchanges with no effect on program performance (neutral crossovers) increased several-fold during the population evolution.

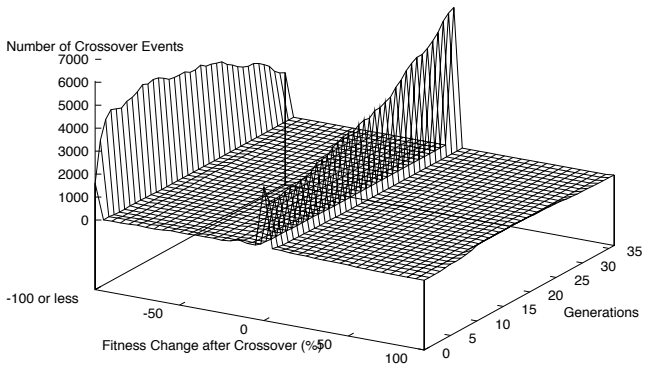


Figure 2: Distribution of fitness effects of recombination during an evolutionary run with genetic programming, from Nordin and Banzhaf (1995). The proportion of neutral crossovers increases during the run.

On the evolutionary biology side, interest in computational models of RNA folding revealed the existence of large mutationally-connected networks of RNA sequences that folded into the same secondary structure, leading to questions about how evolution would proceed on such neutral networks. In 1999, two papers (Bornberg-Bauer and Chan, 1999; Nimwegen et al., 1999) showed that evolution on neutral networks could move a population to genotypes with greater probability of producing neutral mutants, the same

outcome of increased genetic robustness observed in genetic programming by Nordin and Banzhaf (1995).

### *2.7. Artificial Life*

The field that has come to be known as "artificial life" (Langton, 1984, 1986) also focuses on computational systems that exhibit Darwinian evolution. But the motivations behind artificial life and the design of its evolving systems depart significantly from the field of evolutionary computation. In it most ambitious form, the goal of artificial life is to create new "instances of life" in addition to the actual biological world—i.e. systems that exhibit all the essential features of life—in order to advance our understanding of life by having more than the single "data point" of terrestrial living systems. The means to this goal is to create artificial systems that exhibit the principal features of life: structures whose activities result in the maintenance and reproduction of the structures (autopoeisis (Varela et al., 1974)), ecological interactions, and reproduction with variation that allows Darwinian evolution to occur. Another widespread goal is to design artificial systems that exhibit an unbounded increase in complexity in time.

As stated by a founder of the field, Langton (1986):

> The ultimate goal of the study of artificial life would be to create 'life' in some other medium, ideally a virtual medium where the essence of life has been abstracted from the details of its implementation in any particular hardware. We would like to build models that are so life-like that they cease to be models of life and become examples of life themselves.

The root document of the field is von Neumann (1966) "Theory of Self-Reproducing Automata." It spawned numerous efforts to construct cellular automata in which configurations of cell states would self-reproduce through the automaton dynamics. Langton (1984) discovered such a cellular automaton, called the "Langton loop." A different approach was the successful construction by Ray (1993) of a virtual machine, Tierra, where a specific configuration of instructions would compute its own replication. By including error prone operators, the self-reproducing configuration could evolve. The evolutionary dynamics of Tierra produced emergent phenomena such as parasitism, where mutants parasitized other programs for their own replication, and a rich structure of ecological interactions. Several other virtual machine systems have been subsequently developed, including Avida (Adami and Brown, 1994) and Amoeba (Pargellis, 1996).

In these systems, sets of instructions in a virtual machine language perform the computation that replicates the set of instructions. Replication is not performed by separate variation or selection operators. Thus, replication emerges from the computational dynamics. Selection—the differential survival and replication of structures in the virtual machine—is also emergent from the computation, when some sets of instruction (the "digital organisms") outcopy

others. Mutation, however, is typically hand-coded into the construction of the virtual machine by making some instructions probabilistic. Also, until recently, the programmer had to design by hand the initial self-replicating configuration of instructions that founds the population. But LaBar et al. (2015) searched through 3 billion randomly generated programs in the Avida system and discovered 170 programs that produce self-replication.

Another distinction between evolutionary computation and artificial life is that the latter do not employ objective functions for use by selection operators. There is no "goal" or outcome to optimize; the digital organisms are not "candidate solutions" to some exogenously defined problem as is usually the case with evolutionary algorithms. Selection in A-Life systems could be called "natural" in that differential survival and reproduction of the digital organisms is an emergent outcome of the computations in the virtual machine. The emergent properties of artificial life systems such as Avida have made them useful for exploring the dynamics of evolution, for example the phenomenon of "survival of the flattest" (Wilke et al., 2001), the evolution of the germline/soma division (Goldsby et al., 2014), and the evolution of complexity (Lenski et al., 2003).

## 3. List of Relevant Web Pages

Journals in Evolutionary Computation:

*Artificial Life*, MIT Press, `http://www.mitpressjournals.org/toc/artl/1/4`

*Evolutionary Computation*, MIT Press, `http://www.mitpressjournals.org/loi/evco`

*Genetic Programming and Evolvable Machines*, Springer, `http://www.springer.com/computer/ai/journal/10710`

*IEEE Transactions On Evolutionary Computation* (TEVC) `http://cis.ieee.org/ieee-transactions-on-evolutionary-computation.html`

*Swarm and Evolutionary Computation*, Elsevier, `http://www.journals.elsevier.com/swarm-and-evolutionary-computation/`

## 4. Biography

Lee Altenberg is a Senior Fellow at the Konrad Lorenz Institute for Evolution and Cognition Research. As an undergraduate at the University of California, Berkeley, he was interested in complex systems phenomena in nature, and decided that evolutionary theory would provide a rich field for discovery. He studied theoretical population genetics with Glenys Thomson, and studied with Marcus W. Feldman at Stanford University for his doctorate, introducing his advisor to the work of John H. Holland. In his postdoctoral work at Stanford and Duke University he developed early mechanistic models of evolution that

shaped the genotype-phenotype map in the direction of modularity and the evolution of evolvability. He introduced an analytical framework for the evolution of the distribution of fitness effects and evolvability from population genetics into the field of evolutionary computation. He served as Associate Professor in the Department of Information and Computer Sciences at the University of Hawai'i at Manoa. His recent focus has been on obtaining new results in spectral theory to answer questions of complex information transmission in evolution.

## 5. References


Adami, C., Brown, C. T., 1994. Evolutionary learning in the 2D artificial life system "Avida". In: Artificial Life IV. Vol. 1194. MIT Press, Cambridge, MA, pp. 377–381.

Altenberg, L., 1994. The evolution of evolvability in genetic programming. In: Kinnear, K. E. (Ed.), Advances in Genetic Programming. MIT Press, Cambridge, MA, Ch. 3, pp. 47–74.

Altenberg, L., 1995. The Schema Theorem and Price's Theorem. In: Whitley, D., Vose, M. D. (Eds.), Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA, pp. 23–49.

Altenberg, L., 2005. Evolvability suppression to stabilize far-sighted adaptations. Artificial Life 11 (4), 427–444.

Bornberg-Bauer, E., Chan, H. S., 1999. Modeling evolutionary landscapes: mutational stability, topology, and superfunnels in sequence space. Proceedings of the National Academy of Sciences U.S.A. 96 (19), 10689–10694.

Conrad, M., 1974a. Evolutionary learning circuits. Journal of Theoretical Biology 46 (1), 167–188.

Conrad, M., 1974b. Molecular information processing in the central nervous system. In: Conrad, M., Guttinger, W., Cin, M. D. (Eds.), Physics and Mathematics of the Nervous System. Springer, pp. 82–107.

Covello, P. S., Gray, M. W., 1993. On the evolution of RNA editing. Trends in Genetics 9 (8), 265–268.
URL doi:10.1016/0168-9525(93)90011-6

Feldman, M. W., 1972. Selection for linkage modification: I. Random mating populations. Theoretical Population Biology 3, 324–346.

Flombaum, P., Gallegos, J. L., Gordillo, R. A., Rincón, J., Zabala, L. L., Jiao, N., Karl, D. M., Li, W. K., Lomas, M. W., Veneziano, D., et al., 2013. Present and future global distributions of the marine cyanobacteria prochlorococcus and synechococcus. Proceedings of the National Academy of Sciences 110 (24), 9824–9829.

Fogel, D. B. (Ed.), 1998. Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway, NJ.

Fogel, L. J., 1964. On the organization of intellect. Ph.D. thesis, University of California, Los Angeles.

Force, A., Lynch, M., Pickett, F. B., Amores, A., Yan, Y. L., Postlethwait, J., 1999. Preservation of duplicate genes by complementary, degenerative mutations. Genetics 151, 1531–1545.

Frank, S. A., 2013. Microbial evolution: regulatory design prevents cancer-like overgrowths. Current Biology 23 (9), R343–R346.

Friedberg, R. M., 1958. A learning machine: Part i. IBM Journal of Research and Development 2 (1), 2–13.

Friedberg, R. M., Dunham, B., North, J. H., 1959. A learning machine: Part II. IBM Journal of Research and Development 3 (3), 282–297.

Goldsby, H. J., Knoester, D. B., Ofria, C., Kerr, B., 2014. The evolutionary origin of somatic cells under the dirty work hypothesis. PLoS Biology 12 (5).

Gruau, F., 1994. Genetic micro programming of neural networks. In: Kinnear, Jr., K. E. (Ed.), Advances in Genetic Programming. MIT Press, Cambridge, MA, pp. 495–518.

Holland, J. H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Iturriaga, S., Nesmachnow, S., 2012. Solving very large optimization problems (up to one billion variables) with a parallel evolutionary algorithm in CPU and GPU. In: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on. IEEE, pp. 267–272.

Kauffman, S. A., Levin, S., 1987. Towards a general theory of adaptive walks on rugged landscapes. Journal of Theoretical Biology 128, 11–45.

Knibbe, C., Coulon, A., Mazet, O., Fayard, J.-M., Beslon, G., 2007. A long-term evolutionary pressure on the amount of noncoding dna. Molecular Biology and Evolution 24 (10), 2344–2353.

Koza, J. R., June 1990. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Tech. Rep. CS-TR-90-1314, Stanford University, Department of Computer Science, Stanford, CA.

LaBar, T., Adami, C., Hintze, A., 2015. Does self-replication imply evolvability? In: Proceedings of the European Conference on Artificial Life 2015. pp. 595–602.

Langdon, W., Poli, R., 1997. Fitness causes bloat. In: 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC2). pp. 1–10.

Langton, C. G., 1984. Self-reproduction in cellular automata. Physica D: Nonlinear Phenomena 10 (1), 135–144.

Langton, C. G., 1986. Studying artificial life with cellular automata. Physica 22D, 120–149.

Lenski, R. E., Ofria, C., Pennock, R. T., Adami, C., 2003. The evolutionary origin of complex features. Nature 423 (6936), 139–144.

Lobkovsky, A. E., Wolf, Y. I., Koonin, E. V., 2011. Predictability of evolutionary trajectories in fitness landscapes. PLoS Comput. Biol 7 (12), e1002302.

Lozano, J. A., Larrañaga, P., Inza, I., Bengoetxea, E. (Eds.), 2006. Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Vol. 192 of Studies in Fuzziness and Soft Computing. Springer.

McCarthy, J., Minsky, M. L., Rochester, N., Shannon, C. E., 1955 (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. AI Magazine 27 (4), 12.

Mouret, J.-B., Tonelli, P., 2014. Artificial evolution of plastic neural networks: A few key concepts. In: Kowaliw, T., Bredeche, N., Doursat, R. (Eds.), Growing Adaptive Machines. Vol. 557 of Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 251–261.
URL http://dx.doi.org/10.1007/978-3-642-55337-0_9

Mühlenbein, H., Paass, G., 1996. From recombination of genes to the estimation of distributions i. binary parameters. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (Eds.), Parallel Problem Solving from Nature — PPSN IV. Vol. 1141 of Lecture Notes in Computer Science. Springer Berlin

Heidelberg, pp. 178–187.
URL http://dx.doi.org/10.1007/3-540-61723-X_982

Nei, M., 1967. Modification of linkage intensity by natural selection. Genetics 57, 625–641.

Nimwegen, E., Crutchfield, J. P., Huynen, M., 1999. Neutral evolution of mutational robustness. Proceedings of the National Academy of Sciences U.S.A. 96, 9716–9720.

Nordin, P., Banzhaf, W., 1995. Complexity compression and evolution. In: Eshelman, L. (Ed.), Genetic Algorithms: Proceedings of the Sixth International Conference. Morgan Kaufmann, San Francisco, pp. 310–317.

Nunney, L., 1989. The maintenance of sex by group selection. Evolution 43 (2), 245–257.

Oliveto, P., He, J., Yao, X., 2007. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. International Journal of Automation and Computing 4 (3), 281–293.
URL http://dx.doi.org/10.1007/s11633-007-0281-3

Osburne, M. S., Holmbeck, B. M., Coe, A., Chisholm, S. W., 2011. The spontaneous mutation frequencies of prochlorococcus strains are commensurate with those of other bacteria. Environmental microbiology reports 3 (6), 744–749.

Paixão, T., Badkobeh, G., Barton, N., **c**Cörü**c**s, D., Dang, D.-C., Friedrich, T., Lehre, P. K., Sudholt, D., Sutton, A. M., Trubenová, B., 2015. Toward a unifying framework for evolutionary processes. Journal of Theoretical Biology 383, 28–43.

Palmer, M. E., Feldman, M. W., 2012. Survivability is more fundamental than evolvability. PloS ONE 7 (6), e38025.

Pargellis, A., 1996. The evolution of self-replicating computer organisms. Physica D: Nonlinear Phenomena 98 (1), 111–127.

Pelikan, M., Hauschild, M., Lobo, F., 2015. Estimation of distribution algorithms. In: Kacprzyk, J., Pedrycz, W. (Eds.), Springer Handbook of Computational Intelligence. Springer Berlin Heidelberg, pp. 899–928.
URL http://dx.doi.org/10.1007/978-3-662-43505-2_45

Ray, T. S., 1993. An evolutionary approach to synthetic biology: Zen and the art of creating life. Artificial Life 1 (1_2), 179–209.

Rechenberg, I., 1964. Cybernetic solution path of an experimental problem (Kybernetische Lösungsansteuerung einer experimentellen Forschungsaufgabe). In: Fogel, D. B. (Ed.), Evolutionary Computation: the Fossil Record. IEEE Press, New York. 1998, pp. 301–309.

Rechenberg, I., 1973. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart.

Reed, J., Toombs, R., Barricelli, N. A., 1967. Simulation of biological evolution and machine learning: I. selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing. Journal of theoretical biology 17 (3), 319–342.

Schuster, P., Fontana, W., Stadler, P. F., Hofacker, I. L., 1994. From sequences to shapes and back: A case study in RNA secondary structures. Proceedings of the Royal Society of London B: Biological Sciences 255 (1344), 279–284.

Schwefel, H.-P., 1965. Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Master's thesis, Technical University of Berlin.

Schwefel, H.-P., 1987. Collective phenomena in evolutionary systems. Preprints of the 31st Annual Meeting of the International Society for General System Research, Budapest 2, 1025–1033.

Singleton, A., Keenan, N., 1993. Defense against crossover, discussion in the Genetic Programming Workshop at the Fifth International Conference on Genetic Algorithms.

Stoltzfus, A., 1999. On the possibility of constructive neutral evolution. Journal of Molecular Evolution 49, 169–181.

Syswerda, G., 1993. Simulated crossover in genetic algorithms. In: Whitley, L. D. (Ed.), Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA, pp. 239–255.

Valiant, L. G., 2009. Evolvability. Journal of the ACM (JACM) 56 (1), 3.

Varela, F. G., Maturana, H. R., Uribe, R., 1974. Autopoiesis: the organization of living systems, its characterization and a model. Biosystems 5 (4), 187–196.

Visser, d. J., Krug, J., 2014. Empirical fitness landscapes and the predictability of evolution. Nature Reviews Genetics 15, 480–490.

von Neumann, J., 1966. Theory of Self-Reproducing Automata. University of Illinois Press, Champaign, IL, edited and completed by Arthur W. Burks.

Wagner, A., 1994. Evolution of gene networks by gene duplications: a mathematical model and its implications on genome organization. Proceedings of the National Academy of Sciences U.S.A. 91, 4387–4391.

Wagner, G. P., 1981. Evolution of evolutionary mechanisms: A workshop held at Berlepsch Castle (Göttingen, West Germany), November, 1980. Evolutionary Theory 5, 185–186.

Wilke, C. O., Wang, J. L., Ofria, C., Lenski, R. E., Adami, C., 2001. Evolution of digital organisms at high mutation rates leads to survival of the flattest. Nature 412 (6844), 331–333.

Wolpert, D. H., Macready, W. G., 1995. No free lunch theorems for search. Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM.
URL `citeseer.nj.nec.com/wolpert95no.html`

Wolpert, D. H., Macready, W. G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1, 67–82.

## 6. Selected Further Reading

Adami, C., 1998. Introduction to Artificial Life. Springer, New York.

Banzhaf, W., Nordin, P., Keller, R. E., Francone, F. D., 1998. Genetic Programming: An Introduction. Morgan Kaufmann, San Francisco.

Bentley, P., Corne, D., 2002. Creative evolutionary systems. Morgan Kaufmann.

De Jong, K. A., 2006. Evolutionary Computation: A Unified Approach. MIT press.

Jansen, T., 2013. Analyzing Evolutionary Algorithms: The Computer Science Perspective. Springer.

Poli, R., Langdon, W. B., McPhee, N. F., 2008. A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, (With contributions by J. R. Koza).

Mitchell, M., 2009. Complexity: A guided tour. Oxford University Press.

## 7. Acknowledgments

I thank David B. Fogel and editor Norman Johnson for their helpful comments. This work was supported by the Konrad Lorenz Institute for Evolution and Cognition Research; the Mathematical Biosciences Institute at The Ohio State University, USA, through National Science Foundation Award #DMS 0931642.