Genetic Algorithms and Evolutionary Computation

Ms.Nagori Meghna,
Faculty of EngineeringGovernment EnggCollege Aurangabad, -431432, India

Ms. KaleJyoti
Faculty of Engineering MGM'c Engg College Nanded, -411402, India

Abstract
Generally speaking, genetic algorithms are simulations of evolution, of what kind ever. In most cases, however, genetic algorithms are nothing else than probabilistic optimization methods which are based on the principles of evolution. Using simulation and Genetic Algorithms to improve cluster tool performance, Mooring Pattern Optimization using Genetic Algorithms. This paper is designed to cover a few important application aspects of genetic algorithm under a single umbrella.
Key words:
Genetic Algorithm

Introduction

Creationists occasionally charge that evolution is useless as a scientific theory because it produces no practical benefits and has no relevance to daily life. However, the evidence of biology alone shows that this claim is untrue. There are numerous natural phenomena for which evolution gives us a sound theoretical underpinning. To name just one, the observed development of resistance - to insecticides in crop pests, to antibiotics in bacteria, to chemotherapy in cancer cells, and to anti-retroviral drugs in viruses such as HIV - is a straightforward consequence of the laws of mutation and selection, and understanding these principles has helped us to craft strategies for dealing with these harmful organisms. The evolutionary postulate of common descent has aided the development of new medical drugs and techniques by giving researchers a good idea of which organisms they should experiment on to obtain results that are most likely to be relevant to humans. Finally, the principle of selective breeding has been used to great effect by humans to create customized organisms unlike anything found in nature for their own benefit. The canonical example, of course, is the many varieties of domesticated dogs (breeds as diverse as bulldogs, Chihuahuas and dachshunds have been produced from wolves in only a few thousand years), but less well-known examples include cultivated maize (very different from its wild relatives, none of which have the familiar "ears" of human-grown corn), goldfish (like dogs, we have bred varieties that look dramatically different from the wild type), and dairy cows (with immense udders far larger than would be required just for nourishing offspring).Critics might charge that creationists can explain these things without recourse to evolution. For example, creationists often explain the development of resistance to antibiotic agents in bacteria, or the changes wrought in domesticated animals by artificial selection, by presuming that God decided to create organisms in fixed groups, called "kinds" or *baramin*. Though natural microevolution or human-guided artificial selection can bring about different varieties within the originally created "dog-kind," or "cow-kind," or "bacteria-kind" (!), no amount of time or genetic change can transform one "kind" into another. However, exactly how the creationists determine what a "kind" is, or what mechanism prevents living things from evolving beyond its boundaries, is invariably never explained.

But in the last few decades, the continuing advance of modern technology has brought about something new. Evolution is now producing practical benefits in a very different field, and this time, the creationists cannot claim that their explanation fits the facts just as well. This field is computer science, and the benefits come from a programming strategy called *genetic algorithms*.

What is a genetic algorithm

- Methods of representation
- Methods of selection
- Methods of change
- Other problem-solving techniques

Concisely stated, a genetic algorithm (or GA for short) is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a *fitness function* that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA

Manuscript received December 5, 2010
Manuscript revised December 20, 2010

being to improve them, but more often they are generated at random.The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem.

The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered.

As astonishing and counterintuitive as it may seem to some, genetic algorithms have proven to be an enormously powerful and successful problem-solving strategy, dramatically demonstrating the power of evolutionary principles. Genetic algorithms have been used in a wide variety of fields to evolve solutions to problems as difficult as or more difficult than those faced by human designers. Moreover, the solutions they come up with are often more efficient, more elegant, or more complex than anything comparable a human engineer would produce.

Methods of representation

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space".

This technique was used, for example, in the work of Steffen Schulze-Kremer, who wrote a genetic algorithm to predict the three-dimensional structure of a protein based on the sequence of amino acids that go into it. Schulze-Kremer's GA used real-valued numbers to represent the so-called "torsion angles" between the peptide bonds that connect amino acids. (A protein is made up of a sequence of basic building blocks called amino acids, which are joined together like the links in a chain. Once all the amino acids are linked, the protein folds up into a complex three-dimensional shape based on which amino acids attract each other and which ones repel each other. The shape of a protein determines its function.) Genetic algorithms for training neural networks often use this method of encoding also.

A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution. One example of this technique is Hiroaki Kitano's "grammatical encoding" approach, where a GA was put to the task of evolving a simple set of rules called a context-free grammar that was in turn used to generate neural networks for a variety of problems

The virtue of all three of these methods is that they make it easy to define operators that cause the random changes in the selected candidates: flip a 0 to a 1 or vice versa, add or subtract from the value of a number by a randomly chosen amount, or change one letter to another. In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one subtree with another.

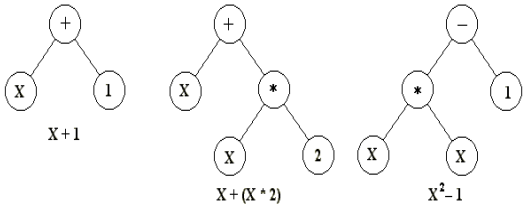

Figure 1: Three simple program trees of the kind normally used in genetic programming. The mathematical expression that each one represents is given underneath.

Methods of selection

There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation, but listed below are some of the most common methods. Some of these methods are mutually exclusive, but others can be and often are used in combination.

Elitist selection: The most fit members of each generation are guaranteed to be selected. (Most GAs do not use pure elitism, but instead use a modified form where the single best, or a few of the best, individuals from each generation are copied into the next generation just in case nothing better turns up.)

Fitness-proportionate selection: More fit individuals are more likely, but not certain, to be selected.

Roulette-wheel selection: A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. (Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen.)

Scaling selection: As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.

Tournament selection: Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

Rank selection: Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution.

Generational selection: The offspring of the individuals selected from each generation become the entire next generation. No individuals are retained between generations.

Steady-state selection: The offspring of the individuals selected from each generation go back into the pre-existing gene pool, replacing some of the less fit members of the previous generation. Some individuals are retained between generations.

Hierarchical selection: Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more computationally expensive fitness evaluation.

Methods of change

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation. There are two basic strategies to accomplish this. The first and simplest is called *mutation*. Just as mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code. The second method is called *crossover*, and entails choosing two individuals to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. Common forms of crossover include *single-point crossover*, in which a point of exchange is set at a random location in the two individuals' genomes, and one individual contributes all its code from before that point and the other contributes all its code from after that point to produce an offspring, and *uniform crossover*, in which the value at any given location in the offspring's genome is either the value of one parent's genome at that location or the value of the other parent's genome at that location, chosen with 50/50 probability.

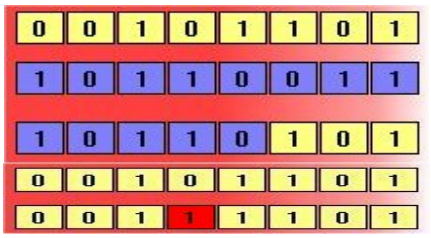

Figure 2: Crossover and mutation.

Other problem-solving techniques

With the rise of artificial life computing and the development of heuristic methods, other computerized problem-solving techniques have emerged that are in some ways similar to genetic algorithms. This section explains some of these techniques, in what ways they resemble GAs and in what ways they differ.

- **Neural networks**

A neural network, or neural net for short, is a problem-solving method based on a computer model of how neurons are connected in the brain. A neural network consists of layers of processing units called nodes joined by directional links: one input layer, one output layer, and zero or more hidden layers in between. An initial pattern of input is presented to the

input layer of the neural network, and nodes that are stimulated then transmit a signal to the nodes of the next layer to which they are connected. If the sum of all the inputs entering one of these virtual neurons is higher than that neuron's so-called *activation threshold*, that neuron itself activates, and passes on its own signal to neurons in the next layer.

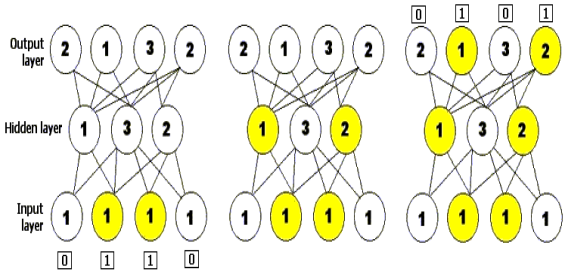

Figure 3: A simple feedforward neural network, with one input layer consisting of four neurons, one hidden layer consisting of three neurons, and one output layer consisting of four neurons. The number on each neuron represents its activation threshold: it will only fire if it receives at least that many inputs. The diagram shows the neural network being presented with an input string and shows how activation spreads forward through the network to produce an output.

Hill-climbing

Similar to genetic algorithms, though more systematic and less random, a hill-climbing algorithm begins with one initial solution to the problem at hand, usually chosen at random. The string is then mutated, and if the mutation results in higher fitness for the new solution than for the previous one, the new solution is kept; otherwise, the current solution is retained. The algorithm is then repeated until no mutation can be found that causes an increase in the current solution's fitness, and this solution is returned as the result. (To understand where the name of this technique comes from, imagine that the space of all possible solutions to a given problem is represented as a three-dimensional contour landscape. A given set of coordinates on that landscape represents one particular solution. Those solutions that are better are higher in altitude, forming hills and peaks; those that are worse are lower in altitude, forming valleys. A "hill-climber" is then an algorithm that starts out at a given point on the landscape and moves inexorably uphill.) Hill-climbing is what is known as a *greedy algorithm*, meaning it always makes the best choice available at each step in the hope that the overall best result can be achieved this way. By contrast, methods such as genetic algorithms and simulated annealing, discussed below, are not greedy; these methods sometimes make suboptimal choices in the hopes that they will lead to better solutions later on.

Simulated annealing

Another optimization technique similar to evolutionary algorithms is known as simulated annealing. The idea borrows its name from the industrial process of *annealing* in which a material is heated to above a critical point to soften it, then gradually cooled in order to erase defects in its crystalline structure, producing a more stable and regular lattice arrangement of atoms. In simulated annealing, as in genetic algorithms, there is a fitness function that defines a fitness landscape; however, rather than a population of candidates as in GAs, there is only one candidate solution. Simulated annealing also adds the concept of "temperature", a global numerical quantity which gradually decreases over time. At each step of the algorithm, the solution mutates (which is equivalent to moving to an adjacent point of the fitness landscape). Finally, the temperature reaches zero and the system "freezes"; whatever configuration it is in at that point becomes the solution. Simulated annealing is often used for engineering design applications such as determining the physical layout of components on a computer chip

A brief history of GAs

The earliest instances of what might today be called genetic algorithms appeared in the late 1950s and early 1960s, programmed on computers by evolutionary biologists who were explicitly seeking to model aspects of natural evolution. (Mitchell 1996, p.2). By 1962, researchers such as G.E.P. Box, G.J. Friedman, W.W. Bledsoe and H.J. Bremermann had all independently developed evolution-inspired algorithms for function optimization and machine learning, but their work attracted little followup..

The next important development in the field came in 1966, when L.J. Fogel, A.J. Owens and M.J. Walsh introduced in America a technique they called *evolutionary programming*. In this method, candidate solutions to problems were represented as simple finite-state machines; like Rechenberg's evolution strategy, their algorithm worked by randomly mutating.

As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments; most notably, Holland was also the first to explicitly propose crossover and other recombination operators. However, the seminal work in the field of genetic algorithms came in 1975, with the publication of the book Adaptation in

Natural and Artificial Systems.. That same year, Kenneth De Jong's important dissertation established the potential of GAs by showing that they could perform well on a wide variety of test functions, including noisy, discontinuous, and multimodal search landscapes . These foundational works established more widespread interest in evolutionary computation. By the early to mid-1980s, genetic algorithms were being applied to a broad range of subjects, from abstract mathematical problems like bin-packing and graph coloring to tangible engineering issues such as pipeline flow control, pattern recognition and classification, and structural optimization.

What are the strengths of GAs?

The first and most important point is that genetic algorithms are intrinsically parallel. Most other algorithms are serial and can only explore the solution space to a problem in one direction at a time, and if the solution they discover turns out to be suboptimal, there is nothing to do but abandon all work previously completed and start over. However, since GAs have multiple offspring, they can explore the solution space in multiple directions at once. If one path turns out to be a dead end, they can easily eliminate it and continue work on more promising avenues, giving them a greater chance each run of finding the optimal solution.

However, the advantage of parallelism goes beyond this. Consider the following: All the 8-digit binary strings (strings of 0's and 1's) form a search space, which can be represented as ******** (where the * stands for "either 0 or 1"). The string 01101010 is one member of this space. However, it is *also* a member of the space 0*******, the space 01******, the space 0******0, the space 0*1*1*1*, the space 01*01**0, and so on. By evaluating the fitness of this one particular string, a genetic algorithm would be sampling each of these many spaces to which it belongs. Over many such evaluations, it would build up an increasingly accurate value for the *average* fitness of each of these spaces, each of which has many members. Therefore, a GA that explicitly evaluates a small number of individuals is implicitly evaluating a much larger group of individuals - just as a pollster who asks questions of a certain member of an ethnic, religious or social group hopes to learn something about the opinions of all members of that group, and therefore can reliably predict national opinion while sampling only a small percentage of the population. In the same way, the GA can "home in" on the space with the highest-fitness individuals and find the overall best one from that group. In the context of evolutionary algorithms, this is known as the Schema Theorem, and is the "central advantage" of a GA over other problem-solving methods.

Due to the parallelism that allows them to implicitly evaluate many schema at once, genetic algorithms are particularly well-suited to solving problems where the space of all potential solutions is truly huge - too vast to search exhaustively in any reasonable amount of time. Most problems that fall into this category are known as "nonlinear". In a linear problem, the fitness of each component is independent, so any improvement to any one part will result in an improvement of the system as a whole. Needless to say, few real-world problems are like this. Nonlinearity is the norm, where changing one component may have ripple effects on the entire system, and where multiple changes that individually are detrimental may lead to much greater improvements in fitness when combined. Nonlinearity results in a combinatorial explosion: the space of 1,000-digit binary strings can be exhaustively searched by evaluating only 2,000 possibilities if the problem is linear, whereas if it is nonlinear, an exhaustive search requires evaluating 2^{1000} possibilities - a number that would take over 300 digits to write out in full.

Fortunately, the implicit parallelism of a GA allows it to surmount even this enormous number of possibilities, successfully findig optimal or very good results in a short period of time after directly sampling only small regions of the vast fitness landscape. For example, a genetic algorithm developed jointly by engineers from General Electric and Rensselaer Polytechnic Institute produced a high-performance jet engine turbine design that was three times better than a human-designed configuration and 50% better than a configuration designed by an expert system by successfully navigating a solution space containing more than 10^{387} algorithms is that they perform well in problems for which the fitness landscape is complex - ones where the fitness function is discontinuous, noisy, changes over time, or has many local optima. Most practical problems have a vast solution space, impossible to search exhaustively; the challenge then becomes how to avoid the local optima - solutions that are better than all the others that are similar to them, but that are not as good as different ones elsewhere in the solution space. All four of a GA's major components - parallelism, selection, mutation, and crossover - work together to accomplish this. In the beginning, the GA generates a diverse initial population, casting a "net" over the fitness landscape. Compares this to an army of parachutists dropping onto the landscape of a problem's search space, with each one being given orders to find the highest peak.) Small mutations enable each individual to explore its immediate neighborhood, while selection focuses progress, guiding the algorithm's offspring uphill to more promising parts of the solution space, However, crossover is the key element that distinguishes genetic algorithms from other methods such as hill-climbers and simulated annealing. Without

crossover, each individual solution is on its own, exploring the search space in its immediate vicinity without reference to what other individuals may have discovered. However, with crossover in place, there is a *transfer of information* between successful candidates - individuals can benefit from what others have learned, and schemata can be mixed and combined, with the potential to produce an offspring that has the strengths of both its parents and the weaknesses of neither. This point is illustrated in, where the authors discuss a problem of synthesizing a lowpass filter using genetic programming. In one generation, two parent circuits were selected to undergo crossover; one parent had good topology (components such as inductors and capacitors in the right places) but bad sizing (values of inductance and capacitance for its components that were far too low). The other parent had bad topology, but good sizing. The result of mating the two through crossover was an offspring with the good topology of one parent and the good sizing of the other, resulting in a substantial improvement in fitness over both its parents.
The problem of finding the global optimum in a space with many local optima is also known as the dilemma of exploration vs. exploitation, "a classic problem for all systems that can adapt and learn". If a particular solution to a multiobjective problem optimizes one parameter to a degree such that that parameter cannot be further improved without causing a corresponding decrease in the quality of some other parameter, that solution is called *Pareto optimal* or *non-dominated*).

To determine whether those changes produce an improvement. The virtue of this technique is that it allows genetic algorithms to start out with an open mind, so to speak. Since its decisions are based on randomness, *all* possible search pathways are theoretically open to a GA; by contrast, any problem-solving strategy that relies on prior knowledge must inevitably begin by ruling out many pathways *a priori*, therefore missing any novel solutions that may exist there. Lacking preconceptions based on established beliefs of "how things should be done" or what "couldn't possibly work", GAs do not have this problem. Similarly, any technique that relies on prior knowledge will break down when such knowledge is not available, One vivid illustration of this is the rediscovery, by genetic programming, of the concept of negative feedback - a principle crucial to many important electronic components today,.

What are the limitations of GAs?

Although genetic algorithms have proven to be an efficient and powerful problem-solving strategy, they are not a panacea. GAs do have certain limitations; however, it will be shown that all of these can be overcome and none of them bear on the validity of biological evolution.

The first, and most important, consideration in creating a genetic algorithm is defining a representation for the problem. The language used to specify candidate solutions must be robust; i.e., it must be able to tolerate random changes such that fatal errors or nonsense do not consistently result. There are two main ways of achieving this. The first, which is used by most genetic algorithms, is to define individuals as lists of numbers - binary-valued, integer-valued, or real-valued - where each number represents some aspect of a candidate solution. If the individuals are binary strings, 0 or 1 could stand for the absence or presence of a given feature. If they are lists of numbers, these numbers could represent many different things: the weights of the links in a neural network, the order of the cities visited in a given tour, the spatial placement of electronic components, the values fed into a controller, the torsion angles of peptide bonds in a protein, and so on. Mutation then entails changing these numbers, flipping bits or adding or subtracting random values. In this case, the actual program code does not change; the code is what manages the simulation and keeps track of the individuals, for outputting an oscillating signal. At the end of the experiment, an oscillating signal was indeed being produced - but instead of the circuit itself acting as an oscillator, as the researchers had intended, they discovered that it had become a radio receiver that was picking up and relaying an oscillating signal from a nearby piece of electronic equipment! This is not a problem in nature, however. Those organisms which reproduce more abundantly compared to their competitors are more fit; those which fail to reproduce are unfit.

The solution has been "the evolution of evolvability" - adaptations that alter a species' ability to adapt. For example, most living things have evolved elaborate molecular machinery that checks for and corrects errors during the process of DNA replication, keeping their mutation rate down to acceptably low levels; conversely, in times of severe environmental stress, some bacterial species enter a state of *hypermutation* where the rate of DNA replication errors rises sharply, increasing the chance that a compensating mutation will be discovered. Of course, not all catastrophes can be evaded, but the enormous diversity and highly complex adaptations of living things today show that, in general, evolution is a successful strategy. Likewise, the diverse applications of and impressive results produced by genetic algorithms show them to be a powerful and worthwhile field of study.

One type of problem that genetic algorithms have difficulty dealing with are problems with "deceptive" fitness functions (Mitchell 1996, p.125), those where the

locations of improved points give misleading information about where the global optimum is likely to be found. For example, imagine a problem where the search space consisted of all eight-character binary strings, and the fitness of an individual was directly proportional to the number of 1s in it - i.e., 00000001 would be less fit than 00000011, which would be less fit than 00000111, and so on - with two exceptions: the string 11111111 turned out to have very low fitness, and the string 00000000 turned out to have very high fitness. In such a problem, all involve controlling the strength of selection, so as not to give excessively fit individuals too great of an advantage.

Finally, several researchers advise against using genetic algorithms on analytically solvable problems. It is not that genetic algorithms cannot find good solutions to such problems; it is merely that traditional analytic methods take much less time and computational effort than GAs and, unlike GAs, are usually mathematically guaranteed to deliver the one exact solution. Of course, since there is no such thing as a mathematically perfect solution to any problem of biological adaptation, this issue does not arise in nature.

Some specific examples of GAs

As the power of evolution gains increasingly widespread recognition, genetic algorithms have been used to tackle a broad variety of problems in an extremely diverse array of fields, clearly showing their power and their potential. This section will discuss some of the more noteworthy uses to which they have been put.

- Acoustics
- Aerospace engineering
- Astronomy and astrophysics
- Chemistry
- Electrical engineering
- Financial markets
- Game playing
- Geophysics
- Materials engineering
- Mathematics and algorithmics
- Military and law enforcement
- Molecular biology
- Pattern recognition and data mining
- Robotics

Creationist arguments

As one might expect, the real-world demonstration of the power of evolution that GAs represent has proven surprising and disconcerting for creationists, who have always claimed that only intelligent design, not random variation and selection, could have produced the information content and complexity of living things. They have therefore argued that the success of genetic algorithms does not allow us to infer anything about biological evolution. The criticisms of two anti-evolutionists, representing two different viewpoints, will be addressed: young-earth creationist

- Some traits in living things are qualitative, whereas GAs are always quantitative
- GAs select for one trait at a time, whereas living things are multidimensional
- GAs do not allow the possibility of extinction or error catastrophe
- GAs ignore the cost of substitution
- GAs ignore generation time constraints
- GAs employ unrealistically high rates of mutation and reproduction
- GAs have artificially small genomes
- GAs ignore the possibility of mutation occurring throughout the genome
- GAs ignore problems of irreducible complexity
- GAs have preordained goals
- GAs do not actually generate new information
- Evolutionary algorithms are therefore incapable of generating true complexity
- evolutionary algorithms, on the average, do no better than blind search.

Conclusion

Even creationists find it impossible to deny that the combination of mutation and natural selection can produce adaptation. Nevertheless, they still attempt to justify their rejection of evolution by dividing the evolutionary process into two categories - "microevolution" and "macroevolution" - and arguing that only the second is controversial, and that any evolutionary change we observe is only an example of the first.

Now, microevolution and macroevolution *are* terms that have meaning to biologists; they are defined, respectively, as evolution below the species level and evolution at or above the species level. But the crucial difference between the way creationists use these terms and the way scientists use them is that scientists recognize that these two are fundamentally the same process with the same mechanisms, merely operating at different scales. Creationists, however, are forced to postulate some type of unbridgeable gap separating the two, in order for them to deny that the processes of change and adaptation we

see operating in the present can be extrapolated to produce all the diversity observed in the living world.

References

[1] "Adaptive Learning: Fly the Brainy Skies." Wired, vol.10, no.3 (March 2002). Available online at http://www.wired.com/wired/archive/10.03/everywhere.html?pg=2.

[2] Altshuler, Edward and Derek Linden. "Design of a wire antenna using a genetic algorithm." Journal of Electronic Defense, vol.20, no.7, p.50-52 (July 1997).

[3] Andre, David and Astro Teller. "Evolving team Darwin United." In RoboCup-98: Robot Soccer World Cup II, Minoru Asada and Hiroaki Kitano (eds). Lecture Notes in Computer Science, vol.1604, p.346-352. Springer-Verlag, 1999. .

[4] Andreou, Andreas, Efstratios Georgopoulos and Spiridon Likothanassis. "Exchange-rates forecasting: A hybrid algorithm based on genetically optimized adaptive neural networks." Computational Economics, vol.20, no.3, p.191-210 (December 2002).

[5] Assion, A., T. Baumert, M. Bergt, T. Brixner, B. Kiefer, V. Seyfried, M. Strehle and G. Gerber. "Control of chemical reactions by feedback-optimized phase-shaped femtosecond laser pulses." Science, vol.282, p.919-922 (30 October 1998).

[6] Au, Wai-Ho, Keith Chan, and Xin Yao. "A novel evolutionary data mining algorithm with applications to churn prediction." IEEE Transactions on Evolutionary Computation, vol.7, no.6, p.532-545 (December 2003).

[7] Haas, O.C.L., K.J. Burnham and J.A. Mills. "On improving physical selectivity in the treatment of cancer: A systems modelling and optimisation approach." Control Engineering Practice, vol.5, no.12, p.1739-1745 (December 1997).

[8] Hughes, Evan and Maurice Leyland. "Using multiple genetic algorithms to generate radar point-scatterer models." IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.147-163 (July 2000).

[9] Jensen, Mikkel. "Generating robust and flexible job shop schedules using genetic algorithms." IEEE Transactions on Evolutionary Computation, vol.7, no.3, p.275-288 (June 2003).

[10] Kewley, Robert and Mark Embrechts. "Computational military tactical planning system." IEEE Transactions on Systems, Man and Cybernetics, Part C - Applications and Reviews, vol.32, no.2, p.161-171 (May 2002).

[11] Lee, Yonggon and Stanislaw H. Zak. "Designing a genetic neural fuzzy antilock-brake-system controller." IEEE Transactions on Evolutionary Computation, vol.6, no.2, p.198-211 (April 2002).

[12] Obayashi, Shigeru, Daisuke Sasaki, Yukihiro Takeguchi, and Naoki Hirose. "Multiobjective evolutionary computation for supersonic wing-shape optimization." IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.182-187 (July 2000).

[13] Porto, Vincent, David Fogel and Lawrence Fogel. "Alternative neural network training methods." IEEE Expert, vol.10, no.3, p.16-22 (June 1995).

[14] Rizki, Mateen, Michael Zmuda and Louis Tamburino. "Evolving pattern recognition systems." IEEE Transactions on Evolutionary Computation, vol.6, no.6, p.594-609 (December 2002).

[15] Robin, Franck, Andrea Orzati, Esteban Moreno, Otte Homan, and Werner Bachtold. "Simulation and evolutionary optimization of electron-beam lithography with genetic and simplex-downhill algorithms." IEEE Transactions on Evolutionary Computation, vol.7, no.1, p.69-82 (February 2003).

[16] Srinivas, N. and Kalyanmoy Deb. "Multiobjective optimization using nondominated sorting in genetic algorithms." Evolutionary Computation, vol.2, no.3, p.221-248 (Fall 1994).

[17] Soule, Terrence and Amy Ball. "A genetic algorithm with multiple reading frames." In GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference, Lee Spector and Eric Goodman (eds). Morgan Kaufmann, 2001. Available online at http://www.cs.uidaho.edu/~tsoule/research/papers.html.

[18] Tang, K.S., K.F. Man, S. Kwong and Q. He. "Genetic algorithms and their applications." IEEE Signal Processing Magazine, vol.13, no.6, p.22-37 (November 1996).

[19] Zitzler, Eckart and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the Strength Pareto approach." IEEE Transactions on Evolutionary Computation, vol.3, no.4, p.257-271 (November 1999).