3

Evolving Solutions: The Genetic Algorithm and Evolution Strategies for Finding Optimal Parameters

Thomas McTavish and Diego Restrepo

University of Colorado at Denver and Health Sciences Center
Aurora, CO 80045, USA
{Thomas.McTavish,Diego.Restrepo}@uchsc.edu

Summary. This chapter provides an introduction to evolutionary algorithms (EAs) and their applicability to various biological problems. There is a focus on EAs' use as an optimization technique for fitting parameters to a model. A number of design issues are discussed including the data structure being operated on (chromosome), the construction of robust fitness functions, and intuitive breeding strategies. Two detailed biological examples are given. The first example demonstrates the EA's ability to optimize parameters of various ion channel conductances in a model neuron by using a fitness function that incorporates the dynamic range of the data. The second example shows how the EA can be used in a hybrid technique with classification algorithms for more accuracy in the classifier, feature pruning, and for obtaining relevant combinations of features. This hybrid technique allows researchers to glean an understanding of important features and relationships embedded in their data that might otherwise remain hidden.

3.1 Introduction

Evolutionary algorithms (EAs) provide a means of finding an optimal solution to models of data or systems. This chapter provides an overview and some general design principles of two classes of EAs: The *genetic algorithm* (GA), and *evolution strategy* (ES). We will also briefly discuss *genetic programming*.

Unlike many areas of biology and computer science, as a form of biologically inspired computing, EAs give computer scientists and biologists an opportunity to speak the same language. EAs demonstrate how the concepts and mechanics of genetics and biological evolution, which has spawned an extreme diversity of life forms, can also be used to find solutions to fit the niche of an abstract computational problem. EAs and evolutionary computational strategies therefore go far beyond enabling models of artificial life and biological evolution and have proved useful in several disparate areas of finance, engineering, and science.

T. McTavish and D. Restrepo: *Evolving Solutions: The Genetic Algorithm and Evolution Strategies for Finding Optimal Parameters*, Studies in Computational Intelligence (SCI) **122**, 55–78 (2008)
www.springerlink.com

While some researchers developed computational models of evolution in the 1950s and 60s, it was primarily the book by John Holland, *Adaptation in Natural and Artificial Systems* [1], that described the GA and how evolutionary principles could solve a large number of abstract problems. Concurrently, students at the Technical University of Berlin were demonstrating the applicability of ES to optimize aircraft wing parameters in wind tunnels [2].

EAs are therefore somewhat old in terms of computational techniques. They are fairly straightforward to understand and implement; and they are also applicable to a broad number of disparate problems in science and engineering. However, most biologists have not heard of the genetic algorithm or an evolutionary algorithm, and if they have, from their names they assume the algorithms are used to assemble genomes or phylogenetic trees. We will discuss the algorithms in the context of how they can be used to find optimal sets of parameters to describe all sorts of data—even biological problems unrelated to evolution or genetics.

3.2 Searching Parameter Space

To provide a deeper understanding of data and enable predictions for future data, a frequent goal of scientific and engineering efforts is to describe data in terms of a parameterized model. For example, if two-dimensional data could be modeled as a line of the form $y = mx + b$, the data is given as the points (X, Y) and the parameters of the model are m and b.

The difficulty of describing data with a model is that if the model↔data mapping is nonlinear, then one cannot simply solve the problem. One has to *search* through the space of possible parameter values and choose the best set that reasonably describes the data. Sampling all possible parameter sets is often an impossible task, even when the range of parameters is small or finitely discretized. For this reason, approximating methods have to be employed, allowing the search to sample *parameter space* in such a way to hopefully find a globally optimal solution in the process.

When a model is tested against the data, it can report its error. Parameter space can therefore map to an *error landscape* which provides an intuition of how regions in parameter space are likely to perform. A search method therefore employs *heuristics* using the error from any number of points in parameter space to help direct the search toward a global minimum, the solution with least error. The difficulty for most methods is avoiding being tricked into pursuing a local minimum, a solution that is not the global minimum, but where the neighbors in parameter space deliver worse solutions. Avoiding local minima requires adequate sampling of parameter space and the ability to pop out of a local minimum if a proposed solution becomes trapped. The genetic algorithm provides a good strategy for searching parameter space effectively, combining the elements of following a gradient but also with the

ability to jump out of local minima and sample disparate areas in parameter space.

3.3 The Evolutionary Algorithm

The EA is a search technique that employs the ideas of organisms evolving and being naturally selected for to fill a niche, and applies these principles to a number of proposed solutions that are allowed to modify and borrow from each other in such a way that they evolve to fit data.

In pseudocode, an EA can be simply described as

```
Evolutionary-Algorithm()
1  Start with an initial population of possible solutions.
2  Repeat for some number of generations or until a solution is found,
3      Select those individual solutions which will reproduce
4      Breed a new generation of candidate solutions from those individual
           solutions using mutation and recombination
```

Let us look deeper at this simple, but effective search method. This requires the introduction and definition of several terms.

3.3.1 The individual

An *individual* is simply a proposed solution. Each individual has a *genotype* and a *phenotype*. An individual's genotype is the set of values for the parameters being optimized. An individual's phenotype are those parameter values plugged into the model.

The chromosome

GAs and ESs follow the algorithm outlined above. A primary difference between GAs and ESs begins with the way each algorithm encodes the parameters being optimized which subsequently determines the type of operations on those parameters that can be performed. In both cases, the data structure of the genotype is called the *chromosome*.

In the classical GA, the chromosome is a binary string of 0s and 1s, the presence of each *gene* represented as a bit. If the model can be set up to look for the best combinations of objects, the presence of the object would be determined by a single gene. However, as in biology where several genes define a feature, a given parameter may be described by several genes. For example, if a parameter can take on one of 16 possible values, we might encode this parameter in 4 bits ($2^4 = 16$). Therefore, for use in GAs, real values (floating point numbers and integers) have to be converted to binary representations. For example, if a parameter was allowed to vary between 0.0 and 1.0 and this

parameter was represented in 4 bits, the binary value 0110 might correlate with 0.267, one of the 16 discretized values in the set $[0.0, 0.067, ...0.933, 1.0]$.

Another common use for the GA is to obtain an optimal sorting for some list. In this case, a chromosome is the permutation. For example, in the list `ABCDEF`, the permutation `BEDAFC` would be a valid chromosome. Here the string is obviously not binary, but contains indexes to items in a list.

Encodings of real values, each parameter as a single gene on the chromosome, is the domain of ES. (There may be a bit of confusion that arises with the term "real-valued" or "real-coded" GA. From the classical definitions of the GA and ES, real-valued GAs are actually ESs. Much of the literature, however, combines GA and ES under one umbrella, simply calling them both GA). Chromosomes of ESs/real-valued GAs is straightforward: They are typically represented as a vector of floating point or integer values, for example, [0.4 6].

Many real-valued parameters can be set up to be represented as binary strings. There is broad debate about which is better: Traditional GAs that convert parameters to binary or ESs which maintain the real-value representation. GA↔ES comparisons of real-valued parameters often conclude that ES implementations are faster and more accurate [3, 4]. If you do not have the ability to try both approaches, it is probably easiest to avoid issues with binary translation and stick to an ES when evolving real-valued parameters.

3.3.2 Population genetics

If we have a notion of how well a phenotype describes the data, then through the agglomeration of several phenotypes, we should be able to assemble some heuristics to discover and navigate the error landscape. Indeed, that is what EAs do. Their heuristic is natural selection. An EA works to progressively select better solutions through the combination and modification of those solutions that were better than other solutions previously evaluated. That is, from the *population* of potential solutions in one *generation*, the EA selects those individual solutions that were better than others and which also sample disparate areas in parameter space (line 3 in the algorithm), and from that subset, combines aspects from those individuals or makes modifications to those solutions so that the solutions in the next generation might be even closer to an optimum solution (line 4). In so doing, the EA implicitly maintains a history to help direct its search. Let us evaluate each line of the algorithm in more detail.

3.3.3 Line 1: Start with an initial population of possible solutions

It is typical to begin the search with a random population of individuals, each individual with a unique genotype. That is, each parameter of the genotype is randomly assigned, drawn from that parameter's range and probability

distribution. We therefore begin with a shotgun approach of some number of sample solutions in parameter space.

Most implementations of EAs maintain the same population size throughout their operation. Studies that have measured the optimal size of the population in GAs indicate that this number should be about the size of the length of the binary string encoding, but depending on the mutation and crossover operators, complexity of the error landscape, complexity of the model, and if the number of parameters is large, this guideline becomes variable [5]. While an EA can often jump out of local minima, it may still get trapped. Therefore, it may take a large population to help ensure better coverage of the error landscape and increase the likelihood that the global minimum is found.

3.3.4 Line 2: Repeat for some number of generations or until a solution is found,

In most cases, due to noise or simplicity of the model, models will never exactly describe all of the data. For this reason, the error will nearly always be nonzero, even for the global minimum. A terminating condition is therefore necessary to prevent the infinite search for zero error. Such a condition might be to report the best solution after N generations and/or to run until the error is within an acceptable range.

3.3.5 Line 3: Select those individual solutions which will reproduce

Natural selection states that those individuals of a species that are more fit to reproduce will have a reproductive advantage over their contemporaries. An EA also employs the notion of fitness. Fitness is the reciprocal of error, so an EA seeks to maximize fitness or, equivalently, to minimize error. An EA minimizes error by repeatedly biasing the selection of the solutions it will operate on to form new candidate solutions toward the fitter solutions in the current generation.

3.3.6 The fitness function

An individual's fitness is quantified through the *fitness function* which is a measure of how close a phenotype is to the optimum solution. The fitness function is also known as the *evaluation function* or *objective function*. The phenotype is compared to the data, and in the simplest case, a scalar value quantifies how well the model matches the data.

Errors can be more complicated than simple distance measures of the phenotype's model output to the data. For example, say our model contains the variables A and B. If we know A and B can never both be high because of some physical or biological law, then even if high values of A and B plugged into our model describes our data well, then it is wrong. In short, the fitness function can be improved with an understanding of the model parameters.

A clear understanding of the data is also important. How much noise does it contain? Is there bias in the data? Should some features be weighted more heavily than others? Are some features correlated? Are there enough data points for a thorough analysis? Can the data be transformed into other spaces (for example, time-domain data into the spectral domain) for other measures? Obviously, the better we understand the data and the parameters of the model, the better we can construct a more robust fitness function.

Care should be taken so that the error function can create broad gradients and minimize local minima. Consider a binary measure—that either a solution works or does not. The error landscape created in this case is like a golf putting green, nothing but flat terrain and then a small hole. In this case, the error function provides no information to guide the search process. Any optimization algorithm would be as effective as an exhaustive search, or worse! Broader gradients can be obtained, then, through more complex descriptors and larger measurements of error. If the EA is not converging toward a solution, the error landscape is probably too flat and/or choppy with local minima. If this is the case, it is common to experiment with different population sizes, breeding strategies, and recombination operators (discussed below), but it may be more effective to experiment with a different error measure.

3.3.7 Beyond the fitness function: Other selection considerations

The role of the fitness function is to provide a measure of how each individual compares with others in the generation with respect to the global objective. However, consider the goal of the algorithm: To find a global minimum (which implicitly means avoiding local minima). The best candidates are therefore those solutions which minimize error *while sampling disparate and broad areas of parameter space.* In biological natural selection, the genes or regulators of genes that are more fit can rather quickly infiltrate a population. This is something we want to partially avoid in the EA to prevent the search from falling into local minima.

As an example, say we have a population of 100 individuals and that we choose the 20 phenotypes with the least error to act on. Selection of the top n individuals like this is known as *truncation selection*. (The new generation will contain 100 individuals (offspring) which are modifications of the 20). If those 20 genotypes are clustered in parameter space, their offspring will also be clustered in this same area. This may persist for several generations or even indefinitely. Steps to avoid such *population convergence* or *crowding* while performing truncation selection can include having a high mutation rate and broad mutation range (Section 3.3.11) or the introduction of new random individuals into the breeding pool. Such aggressive steps may be unwanted while still being ineffective at popping out of local minima. For these reasons, other selection techniques are more commonly used.

Many EA implementations stochastically select individuals using *roulette-wheel* or *tournament selection*. Those solutions with higher fitness are biased

to be selected over those that have low fitness, but low-fitness individuals still have the possibility of being selected. The reason one may want to include some low-fitness individuals in the breeding population is that it can be assumed that these low-fitness individuals are far from the other individuals in parameter space. Therefore, they maintain variation in the new generation and help ensure that disparate areas in parameter space are being sampled.

Diversity maintenance in the population can also be obtained through *sharing* [6, 7] which decreases the fitness of an individual if it is close to other individuals in parameter space. There are also other methods that deal with crowding by replacing individuals only if the children are indeed better solutions than their parents [8, 9]. This technique keeps the original shotgun sampling diffuse, delaying convergence, but does not explicitly instill diversity. Additionally, multiple populations could be allowed. The idea is that each population is likely to converge to a different local minimum, or *niche*. If the populations are allowed to interact sparsely, exchanging very few individuals and then only every n^{th} generation, then the aspects from each population might collectively combine to form a new niche closer to the global optimum.

In summary, selection needs to balance the fitness of the solution with other population characteristics to sample multiple areas in parameter space.

3.3.8 Line 4: Breed a new generation of candidate solutions from those individual solutions using mutation and recombination

Individuals chosen to be operated on go through a process of *reproduction* to form a new generation of candidate solutions. While the *breeding strategy* includes the subset selected through the fitness function and the previously mentioned selection criteria, it also directs reproduction. Reproduction can include the cloning of a previous solution, the blending of previous solutions, or the modification of previous solutions in making a new candidate. The breeding strategy therefore includes methods for choosing which previous solutions will be cloned, which previous solutions merge and how, and which previous solutions are modified and how.

3.3.9 Elitism

The method of choosing the fittest n individuals from the breeding pool as the ones to be cloned is known as *elitism*. These elite solutions from the previous generation proceed to the next generation without any modification. Since their phenotype has already been evaluated, their fitness function does not need to be calculated again, so few computational resources need to be devoted to them.

The reason you may want to have a few individuals in each generation form an elite, is that they guarantee that the progeny in the new generation will have error at least as small as the previous generation. That is, there is no guarantee that operations on the previous solutions that form new candidate

solutions will result in decreased error for any of the new solutions. In fact, like biological natural selection itself, it is the rare modification that results in less error, so without forming an elite, it may be possible to create a new generation that has individuals who do not have less error than their parents. Therefore, by maintaining a small set of the "best so far" in each generation, we guarantee that those individuals will remain in the gene pool.

3.3.10 Recombination

Recombination involves the merging of parameters of two solutions from the previous generation into new solutions known as *offspring* or *children* in the new generation. In the simplest case, the selection of pairs is entirely random within the population of those selected to breed, including elites. Pairing can be more directed, however. For example, incest prevention prohibits the mating of similar individuals in parameter space to maintain diversity [10]. At the same time, if individuals are too different from each other, their children may likely be poor solutions. In short, it is often best to employ several breeding strategies.

Recombination often employs the typical meiotic form of genetic *crossover* where offspring will receive some number of genes from one parent and the remainder from the other parent (Figures 3.1 and 3.2 middle). It is normal to generate two offspring and for the offspring to be complements of each other. This keeps the population size constant and also provides balance of sampling in parameter space.

Generally speaking, crossover is more important than mutation in the GA whereas the converse is true for ES. This is primarily because crossover and mutation are performed differently and have different effects in each.

In the GA, crossover operates to quickly weed out which bits should be 1 and which bits should be 0. Here's how. Firstly, we can say, obviously, that if a gene is the same in each parent, then crossover of that bit has no effect. Therefore, children will only *possibly differ* from their parents at those genes that are different between the parents. After obtaining the fitness of the children, we have the potential of gaining a lot of information. We know the fitness of the parents and the bits that are different between them. We also know the fitness of the children who only vary by a different combination of those same bits. Crossover therefore functions to compare the roles of those bits that are different between the parents and begins to characterize the appropriate settings of those bits. We potentially glean the most information when crossover occurs such that half of the different genes go to one child and half go to the other. This is the motivation behind the *uniform crossover* [11] and the *half-uniform crossover* (HUX) [12].

Another common GA crossover operator is the *one-point crossover* where one child will consist of the left portion of one parent and the right portion of the other parent (Figure 3.1 right). The crossover point is usually randomly selected and can therefore bisect a parameter that may span some number

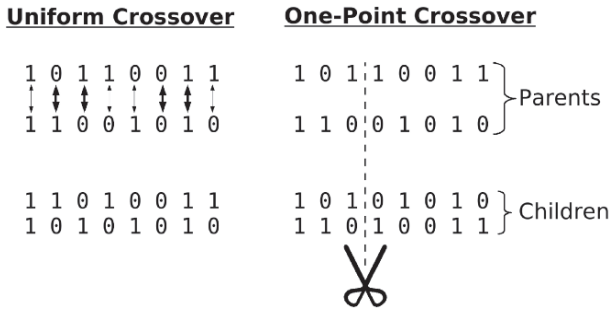

Fig. 3.1. Two types of crossover in the GA. Parents are in the top rows and children are in the bottom. In uniform crossover, each bit has the same probability of flipping. Bold arrows show where a bit flip occurred. Bold arrows to the right have no bearing in the children since the parents have the same bits set at those positions. In one-point crossover, the top child receives the left portion of the top parent and the right portion of the bottom parent. The converse is true for the bottom child.

of genes. The idea behind one-point crossover can be extended to allow n-point crossovers. When utilizing point crossovers, one has to be careful of the placement of the parameters on the chromosome. Genes that are close to each other will tend to crossover together. Therefore, if genes are known to vary together, they should be placed close to each other on the chromosome.

In ES/real-valued GA crossover, children also receive some set of parameters from one parent and the remaining set from the other (Figure 3.2 middle). The role of crossover in real-valued strategies, however, is to find the strongest parameter values that are contributing to each parent's fitness and to incorporate those into the children. It assumes that two parents probably contribute differently to the error—that some genes are good at reducing error in one parent and other genes are probably the ones largely responsible for reducing error in the other parent. With the right swapping of genes, then, a child could potentially reduce the error to even a greater degree.

The reason crossover is not such a prevalent director of evolution in real-valued scenarios is that it is really difficult to determine the combination of parameters to swap that might actually yield fitter offspring. The most common crossover operators randomly select which genes to swap, but real-valued encodings offer a number of customized scenarios. For example, one could easily employ a crossover operator to swap the n most different genes between two parents. The motivation for such an approach is that the most disparate genes between two parents might best describe their different ways of reducing error. Another crossover operator applied on another set of parents might swap their most similar genes to hone those parameters. (Also see Section 3.4.1). It is common to employ a few crossover techniques on a generation, but only one crossover operation is typically applied on a set of parents.

A more complicated form of real-valued crossover is the *blended crossover* [13]. With the blended crossover (also known as BLX-α), a new candidate parameter could be 25% of parent P_A and 75% of parent P_B. Indeed, this is the case for Gene A in the bottom child in the bottom portion of Figure 3.2. Conversely, to maintain balance in parameter space, the top child is 75% of P_A and 25% of P_B.

One needs to be careful when naïvely applying blended crossover, however. It is easy to think that blending within the range of $0 - 100\%$ is the best way to merge two values of a parameter. Blending a parameter in this fashion, though, will work to constrain the range of the parameter. At the population level, this blending will push a given parameter toward the centroid of the gene's value for that generation. Over generations, the centroid might shift, but the range of the variable will continually shrink. To avoid introducing such bias, it is necessary to expand the range of the blend to something like -50% to 150%. This is the case for Gene D. The top child of the bottom row is 150% of the top parent and -50% of the bottom parent. Conversely, the bottom child is -50% of the top parent and 150% of the bottom parent. While this may seem inappropriate, it maintains the dynamic range of the parameter, the gene constricting between some parents and expanding between others.

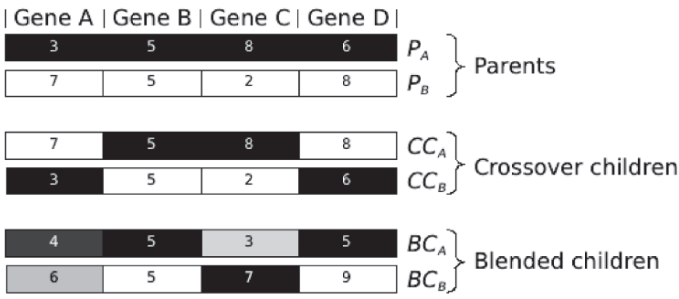

Fig. 3.2. Recombination of real values via crossover (middle) and blended crossover (bottom). The genes from the top parent, P_A, are colored black and the genes from the bottom parent, P_B, are white. In normal crossover of children (CC), genes are randomly swapped between the parents. In blended crossover (BC), a gene in child BC_A will contain $(-50 + n)\,\%$ of P_A and $(150 - n)\,\%$ of P_B where n is a random variable. Conversely, this same gene in child BC_B will contain $(-50 + n)\,\%$ of P_B and $(150 - n)\,\%$, n being the same value as applied to child BC_A. Grayscale values of the gene indicate the relative contribution of each parent, black or white, to the child's gene.

It is important to note that not all parents have to breed. The crossover rate is typically subject to experimentation and may, in fact, be quite low. In this case, parents can clone themselves and then mutate to introduce variability into the children.

3.3.11 Mutation

A *mutation* is a modification to a gene. The reason for introducing mutation is to add variation to the population, giving the algorithm other sample points to measure, and to knock individuals out of local minima.

In the GA, a common simple mutation operator is the uniform mutator. Each bit of the chromosome has a low (usually about 1%) probability of flipping at each generation.

One problem with binary chromosomes is the Hamming cliff problem [14]. Consider the strings 01111 and 10000. If these bits represented one numeric integer, the value of these strings varies by 1. (01111 in binary equals 15 and 10000 in binary equals 16). Also consider the strings 01111 and 11111. These strings vary by one bit, but their values are different by 16. (01111 in binary equals 15 and 11111 in binary equals 31). To help ensure that a single point mutation has minimal, and in a sense, incremental response, it is rare to give parameters their normal binary representations. Instead, it is common to encode binary representations using Gray encoding [15] which makes it so that all adjacent numbers, like 15 and 16, only require one bit change.

There are a number of mutation strategies one can employ with ES. A common, simple mutation strategy is to let each gene have some probability of mutating and when it does, modify that real parameter by some amount—usually the addition of a draw from a normal (Gaussian) distribution centered at zero. Of course, uniform and other probability distributions can also be used. The smaller the mutation, the better it can pursue a gradient and the larger, the better it can pop out of local minima. It may therefore make sense to employ a number of mutators, some with tight *mutation range* and some with a broad range, each mutator with a different *mutation rate*.

In ES, mutation strategy is often tightly coupled with *replacement strategy*, choosing when to remove an individual from the breeder pool. Because many mutations may be deleterious, resulting in poorer fitness in the children, a common replacement strategy in ES is to discard parents from the breeder pool only when they produce fitter offspring.

There is nothing to say that binary representations cannot also employ mutation strategies used in ES. Given that several bits may encode a parameter, it would be easy to decode the bits to capture the value of the parameter, mutate the value as in ES, and then re-encode the binary representation.

3.3.12 Finding optimal parameters *and* the model

In many cases, it may not only be parameters of the model that we wish to optimize, but it may also include the operators of the model itself! In this case, EAs can also be used. This is the domain of *genetic programming* (GP). In this case, chromosomes are computer programs, usually assemblies of tree-like structures. Each terminal node (leaf) in the tree is a parameter and each branching node is an operator (like add, subtract, multiply, divide, log,

exponent, etc.). Operators are subject to mutation, changing from one type to another, but also branches (edges) between nodes are subject to appearing and being removed.

As a simple example of a genetic program, suppose a model to describe the data is $y = 3x + 2$. Figure 3.3 illustrates an optimal genetic program's chromosome to describe this data. Given (X, Y) input data pairs, the EA would figure out the "2" and "3" leaves as well as the "+" and "*" internal nodes to best-fit the data.

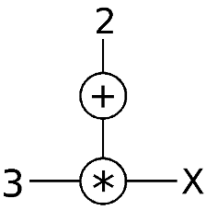

Fig. 3.3. Genetic programming example. The model and parameters, $y = 3x + 2$.

It is, however, difficult to know how to evolve a program toward an end goal. In natural systems, the goal is and always has been survival and reproduction, but the factors influencing survival and reproduction are constantly changing as a given species evolves to exploit one niche after another. If we wanted to evolve a very large sea creature from an ancestral fish, would we first have it develop lungs and walk on land? Nature demonstrates through the blue whale that evolution does not take a linear course!

GPs, like natural systems, are subject to phylogenetic constraints. The developmental program of horses is such that they can never evolve wings. Analogously, GP may be able to compose programs that bifurcate between "birds" and "horses" but are never able to devise a pegasus because to do so would require specific fundamental changes to the program which would render the individual so unfit compared to the horses and birds already in the population.

3.4 Advanced topics

We now concentrate on further strategies to improve the running time of an EA and/or its results.

3.4.1 Errors as vectors

Error is typically reported as one number—the sum of all of the discrete measures of error. Instead, the error of a phenotype could be reported as a vector

in *error-parameter space* where each of these discrete measures represents a separate dimension. This can perhaps be best understood with an example. Say that my data is in the frequency domain, and I am searching for parameters that can deliver a specific frequency and a specific amplitude. As a single number, error would be reported as the sum of the distance from the target frequency plus the distance to the target amplitude. If error is instead represented as a vector in error parameter space, one dimension could be error in frequency and one dimension could be error in amplitude. Doing this can improve the search significantly because it can direct the phenotypes we are interested in pairing for reproduction.

Consider the three vectors in Figure 3.4. These all have the same magnitude so in the traditional error landscape, they would have equal weight. Vector A, however, has no error in amplitude and vector C has no error in frequency. When error is expressed as vectors, we can better target breeding strategies. In this case, vectors A and C are good candidates for breeding because we would like to take whatever it is in A that makes it hit the target amplitude and whatever it is in C that makes it find the target frequency and hopefully discover offspring that can hit both the target frequency and amplitude. We could also preferentially select parameters that were more dissimilar to be selected for crossover. Additionally, EAs are not bound by natural laws of sexual reproduction. If we know one individual reduces error along one dimension, another reduces error in another, and a third reduces error in yet another dimension, then we may seek to breed all three parents.

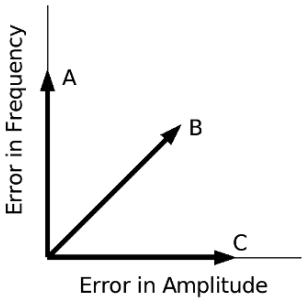

Fig. 3.4. Error represented as vectors in error parameter space. Error is measured in distance from target frequency along the Y axis and distance from the amplitude target along the X axis. A, B, C are individual phenotypes with the same magnitude of error.

Expressing errors as vectors can also make for more robust selection criteria. I may have a phenotype that reduces error in one dimension better than any other phenotype, but gives horrible error along other dimensions. Even though this phenotype has an overall high error, I may choose to keep

this phenotype in my reproductive pool simply because it may describe one dimension better than any other phenotype.

Treating errors as vectors is the subject of Multi-Objective Evolutionary Algorithms (MOEAs) which are covered in more detail in Chapter 4.

3.4.2 Evolution of evolution

While EAs can drastically improve search time by pruning parameter space, several factors such as population size, crossover operators, crossover rates, mutators, and mutation rates will impact the speed and accuracy of the results. We are unlikely, however, to know what the best operators and settings for the EA might be. In a sense, it would be nice to throw the kitchen sink at the problem and somehow let the EA determine appropriate settings. This can be accomplished by maintaining with a child, the operations that gave rise to it. Therefore, when comparing the fitness of individuals in a generation, we can also compare the successful and unsuccessful operators. For example, if we note that small mutations gave rise to fitter individuals than those that received large mutations, then the mutation range may be deemed too aggressive and dynamically change. Likewise, if a particular crossover operator yields fitter children than another crossover operator, we can bias toward that stronger operator. When the EA modifies itself from such acquired history, this is known as *self-adaptation*. A popular mechanism in ES for self-adaptation is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [16, 17].

3.4.3 Been there, done that

It may become apparent that an individual or population of individuals is stuck in a local minimum. Rather than letting them continue to exhaust computational resources and corrupt the gene pool, it may be time for them to die. One can annihilate these candidate solutions by discarding parents that do not produce fitter offspring after n generations. This is known as *aging*. It may also be important to annihilate all of the individuals in the region so that they do not continue to find the same local minimum. Furthermore, it may be useful to dynamically tag this region in parameter space as off-limits so that new individuals do not migrate down the same valley.

3.4.4 Lamarckism

Recombination in the natural world is not goal-oriented. Operations randomly happen and the fitter characteristics are simply more likely to proceed to the next generation. There is nothing to prevent us, however, from employing Lamarckism and employing operators which are goal directed in EAs. This is the domain of *memetic algorithms* which allow children to perform local searches before undergoing mutation and recombination by the EA.

As an example, the EA could work with individuals of local minima. That is, with each generation, individual solutions could follow gradient descent or some greedy, localized search until they became trapped in a local minimum. At that point, the EA could inject mutations and recombination to these local minima. The presumption with this technique is that through the ensuing crossover and mutation of several local minima, better local minima will likely be found in each subsequent generation.

3.4.5 Where EAs fail

A number of factors can make an EA unable to converge on a solution. We have talked about the importance of robust fitness measures and various breeding strategies. The complexity of the model and the interdependence of the variables may also cause a very rough fitness landscape. It may be, however, that the formulation of the problem is *deceptive*. That is, there is a hill surrounding a tiny steep hole that is the global optimum. Samples near the top of the hill will migrate away from the global optimum. Resolving deceptive problems can include the application of different fitness measures, data transformations, model modifications, or dropping the EA for another optimization technique.

3.5 Examples

3.5.1 Optimizing parameters in a computer simulation model

Mitral cells are neurons in the main olfactory bulb responsible for propagating signals from olfactory sensory neurons to pyramidal cell neurons in the olfactory cortex. Like many neurons, active mitral cells fire action potentials with a certain periodicity and often synchronize with other mitral cells. Additionally, it has been reported in the literature that the resting membrane potential of isolated mitral cells which are not receiving any other synaptic input often exhibit subthreshold oscillations [18]. That is, when they are not firing, the difference of electrical charge inside the cell compared to outside the cell fluctuates in a rhythmic fashion. When this charge is centered at -65 mV, the frequency of oscillations averages 13 Hz. At a more positive charge centered at -59 mV, just below firing threshold, the frequency of oscillations is 39 Hz. The amplitude of the oscillations averages 1.85 mV. These findings are summarized in Table 3.1. It is presumed that transmembrane ion channel proteins are responsible for such electrical fluctuations, but the presence and density of such channels have not been empirically determined.

Two neuroscientists, Rubin and Cleland, took an existing mitral cell computational model [19] and predicted that the addition of six particular ion channels could give rise to subthreshold oscillations [20]. They could not say, however, what the conductance (i.e. density) of each channel should be to mimic this oscillatory behavior. Constraining the variability of each channel's

Table 3.1. Subthreshold Oscillation Data. (Amplitude = 1.85 mV for all samples).

Membrane Potential (mV)	Subthreshold Frequency
-65	13
-64	20
-63	26
-62	31
-61	35
-60	38
-59	40

density to a small range, they ran over 60,000 simulations with different values of the channels' densities. They then chose the set which elicited the closest response. Because each ion channel is mathematically modeled with differential equations, each simulation takes a number of seconds to compute. The problem is also therefore highly nonlinear. We were curious to see if an EA which allowed for a broader density range would find the same or even more accurate channel densities while reducing the number of simulations.

This mitral cell model was built with the NEURON simulation package [21] so we modified NEURON to incorporate the Open BEAGLE EA library [22]. The goal was to determine the six channel densities which could match the data as summarized in Table 3.1. Therefore, the genotype was the density of each of these six channels. The phenotype was the behavior of the computational neuron model with each particular set of channel densities plugged into it. A fit individual should behave such that when the cell's membrane has a base potential of -65 mV it exhibits a subthreshold oscillation frequency of 13 Hz, and when its membrane potential has a base of -64 mV it should oscillate at 20 Hz, etc. We therefore wanted 7 samples from each individual where each sample would uniquely correspond with a row in Table 3.1. Each of the 7 samples reported an error value of how close it matched its corresponding row and the overall error for the individual was the sum of these values.

Because each set of ion channel densities impacts membrane dynamics differently, it is not known how much current is necessary to depolarize the individual so that its base membrane potential can be -65 mV, then -64 mV, etc. Yet, we wanted 7 samples from each individual which matched the data. We therefore created 11 possible samples by injecting 11 incremental steps of depolarizing current. We then set this up as a *weighted bipartite graph / optimal assignment* problem and employed the *Kuhn-Munkres (Hungarian) algorithm* [23] to choose the 7 best matches of these 11 steps. Figure 3.5 provides an illustration of a matching and our error function (described below).

Now we could have modeled the data in Table 3.1 with a simple equation and for each individual asked how close it mapped to this target function. However, this could have made it such that an individual fit very well, but only to a small portion of the data. Setting the problem up as an optimal assignment problem ensured adequate coverage of the target data.

For our error measure, we took the Euclidean distance of each of the 11 samples to each of the 7 target rows along three dimensions: 1) The base membrane potential 2) the oscillatory frequency, and 3) the amplitude of the oscillations to 1.85 mV. Because frequency had the most variability and dynamic range, we scaled the other measures by a factor of 3 to provide a better balance between the individually reported error vectors. The sum of the error vectors from a sample to a target row first provided the weight of the edge in the bipartite graph for the optimal assignment. After the optimal assignment, the sum of the matched edges was the error reported for an individual.

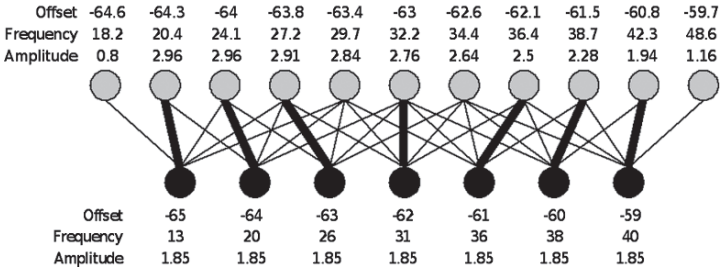

Fig. 3.5. Example of an optimal assignment of 11 sorted samples (top row) to 7 sorted target values (bottom). The weight of the edge is the error of a sample to a target row in Table 3.1 where error is defined as the Euclidean distance $\sqrt{(3 * offset_s - 3 * offset_t)^2 + (freq_s - freq_t)^2 + (3 * amp_s - 3 * 1.85)^2}$ and where s subscripts denote the sample and t subscripts denote the target. The heavy edges denote an optimal assignment or matching. The error reported for an individual to the EA is the sum of these matched edges.

We employed four types of EAs: 1) A binary encoding and single composite error measure, 2) a binary encoding with multiple error measures (i.e. a multiobjective EA) where the sum of the 7 offsets' distances were reported as a value, the sum of the errors in frequency as another value, the sum of the amplitude errors as a third value, and the composite of all three as another value[1] 3) real-valued with a single error measure, and 4) real-valued with multiple error measures. Three trials were performed with each type of EA. The results are highlighted in Figure 3.6 and summarized in Table 3.2. The results show that most EAs were able to converge on solutions that provided slightly better results than Rubin and Cleland's within a couple thousand simulations. The real-valued single error measure was least liable to get stuck in a local minimum, even though one run with a bitstring GA did better than any of the real-valued approaches. Those runs that got stuck could have

[1] Multiobjective EAs deliver the "Pareto front", describing trade-offs between errors. That is, for some error value along one dimension, the MOEA will provide the smallest values for the other errors. MOEAs do not directly reduce the composite score. By including the composite score, the EA could seek to reduce it.

perhaps been assisted with population migration, but that was not employed for these simulations to keep the runs segregated. It is also important to highlight that our results largely resemble Rubin and Cleland's, providing further support for their findings as well as the use of EAs to also solve the problem.

Table 3.2. Summary of model simulations. Optimal coefficients for the six channels from the best individual from each type of simulation. Abbreviations: S is for ion channels at the soma. D is for ion channels in the lateral dendrite. kA is an inactivating Potassium channel, kCa is a Calcium-dependent Potassium channel. NaP is a persistent Sodium channel. Ih is a hyperpolarization-activated cation current. Bottom two rows describe the allowed dynamic range of each variable.

	Evolved channel coefficients						Total	
	S_{kA}	S_{kCa}	D_{NaP}	D_{kA}	D_{kCa}	D_{Ih}	runs	Error
Rubin/Cleland	0.012	0.12	0.00042	0.0074	0.10	0.0024	60000	14.6
Bit/Single	0.010	0.10	0.00042	0.0090	0.10	0.0022	1910	9.87
Bit/Multi	0.016	0.08	0.00039	0.0023	0.13	0.0027	2300	13.9
Real/Single	0.007	0.13	0.00038	0.0040	0.11	0.0025	2910	10.7
Real/Multi	0.020	0.04	0.00041	0.0062	0.11	0.0022	3060	11.4
Min value	0.005	0.01	0.00010	0.0010	0.01	0.0010		
Max value	0.100	0.50	0.00100	0.0500	0.50	0.0100		

3.5.2 Feature reduction and selection

Biological data is frequently laden with several signals and noise. It is often difficult or impossible to extract the important features or combinations of features to explain the data. *Feature reduction* algorithms such as Principal Components Analysis (PCA) transform several features into a few meta-features. Employing such a technique has proven useful in pre-filtering data and as a "wrapper" to clustering and classification algorithms such as K-Means, Artificial Neural Networks (ANNs), and Support Vector Machines (SVMs) to make them more robust [24–26].

By and large, machine learning techniques such as ANNs and SVMs operate as black boxes, filtering and transforming the input data into a particular output classification. They do not, however, make it easy or perhaps even possible to describe the particular input features which give rise to a particular output. While feature reducers can make a stronger classifier, by collapsing several features into a meta-feature, feature reducers often further cloud the ability to reverse-engineer the black box. This is unfortunate because it is especially meaningful when it can also be known what features and combinations of features really explain the output.

EAs have been employed as feature reducers to great effect [24, 25], but they can also be used as a *feature selector*, permitting a reverse-engineering

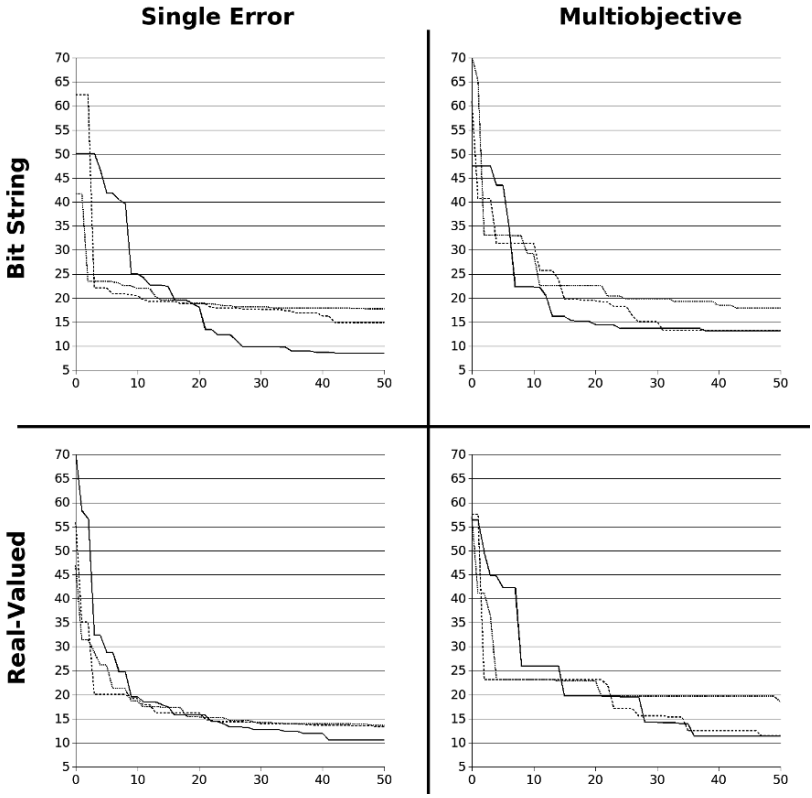

Fig. 3.6. Evolution of 4 different types of EAs to determine the best density values for six ion channels to exhibit subthreshold oscillations in a computational mitral cell neuron model. 3 runs with 60 individuals for 50 generations was performed with each EA type. Generations are plotted on the X-axis. Y-axis is on a logarithmic scale and indicates the error value of the individual from each generation with the lowest error. Solid line indicates the run finishing with the lowest error. The single-error scenarios employed an elite of 3 individuals per generation. Multiobjective scenarios used NSGA-II [8] for their replacement strategy. Tournament selection was used on all scenarios. The bitstring scenarios encoded each parameter with 12 bits and used a uniform crossover operator with a rate of 50% for each bit and with a 30% chance of a given individual being selected for crossover. They also used a uniform mutation operator set at 2%. The bitstring single-error strategy processed about 1925 individuals and the multiobjective strategy processed 2285 individuals in each run. The real-valued scenarios employed a blended crossover (BLX-α) [13] with $\alpha = 0.5$ such that a variable would blend between -50% and 150% of its parents. Individuals had a 30% chance of being selected for crossover with each generation. A Gaussian mutation operator was used with σ values for each parameter set to 20% of the range of the parameter. Each gene of each individual had a 10% chance of being mutated in each generation. The real-valued single-error scenario processed 2910 individuals and the multiobjective strategy processed 3060 in each run.

of the black box clusterer or classifier [26, 27]. When EAs are coupled with clusterers or classifiers, features are the genes being optimized. Therefore, the feature landscape is equivalent to the fitness landscape. A niche in parameter space, then, will describe those combinations of features which give rise to a particular classification. EAs can therefore provide information about input features in ways other feature reducers cannot.

As an example of an EA used for feature selection, Lavine and Vora used the GA to discriminate European honeybees from Africanized honeybees through gas chromatograms of the bees' secretions [27]. Figure 3.7 shows an example of a gas chromatogram. Within each trace are 65 peaks labeled with a letter and a number.

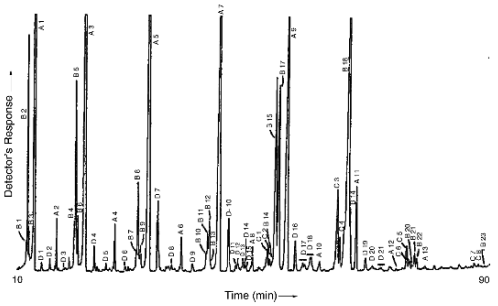

Fig. 3.7. Gas chromatographic trace of the hydrocarbon extracts obtained from the wax gland, cuticle, and exocrine gland of a heavily Africanized forager. A: normal alkanes; B: alkenes; C: dienes; and D: branched chain alkanes. Reprinted with kind permission from Lavine et al. [27].

Figure 3.8 shows a plot of the two principal components of these 65 peaks. European honeybees are labeled with a 1 and Africanized bees are labeled with a 2. What is apparent in the PCA plot is that there is not a clear distinction between the two sets. At least two reasons exist for the lack of separation. Either the data does not contain enough information to segregate the bees into distinct categories, or the existing data contains too many features clouding the most pertinent ones.

Assuming that relevant features were being masked by unnecessary features (noise), Lavine and Vora employed the GA to search for the relevant peaks in the chromatogram that could best discriminate the bees. They let the presence of a peak be determined by a single bit on a 65-bit chromosome. In this capacity, the chromosome effectively served as a filter on the chromatograms to either keep a peak at its original value (gene present) or set the peak to zero (gene not present). The fitness of an individual was assessed by perfoming PCA on 238 bee chromatograms filtered by the chromosome and then scoring the plot by how well it segregated the two classes. Figure 3.8 therefore shows the PCA plot with the chromosome of all ones: [111...111].

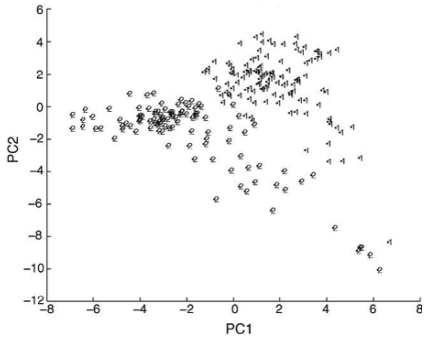

Fig. 3.8. Plot of the two largest principal components of the 65 gas chromatography peaks and 238 European and Africanized honeybee gas chromatograms that comprise the training set. Each bee is represented as a point in the principal component plot: 1 represents European honeybees, and 2 represents moderately and heavily Africanized honeybees. Reprinted with kind permission from Lavine et al. [27].

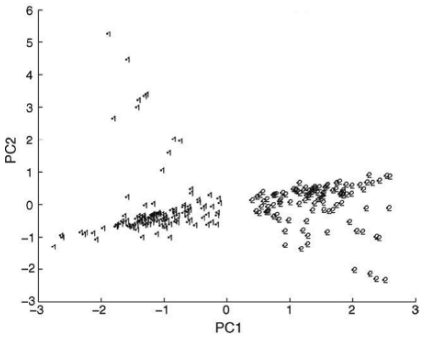

Fig. 3.9. Plot of the two largest principal components developed from the four gas chromatography peaks identified by the pattern recognition GA. Each bee is represented as a point in the principal component plot: 1 represents European honeybees, and 2 represents moderately and heavily Africanized honeybees. Clustering of the honeybees by genotype is evident. Reprinted with kind permission from Lavine et al. [27].

Figure 3.9 shows the result of the chromosome that displayed the best separation of the classes in the PCA plot. In this chromosome, only 4 genes were used (corresponding to peaks B11, B14, B15, and B22) and all other peaks were set to zero. The GA had therefore been able to select the 4 peaks which could discriminate the two classes.

The gas chromatogram looks quite similar to the output of mass spectrometers (MS) where, indeed, EAs have been used frequently in proteomics [24, 28]. For more EA examples in bioinformatics, see the review by Pal, Bandyopadhyay, and Ray [29] which highlight how EAs have been applied to fragment assembly, gene mapping, sequence alignment, gene finding and promoter identification, microarray analysis, molecular structure prediction, and protein-ligand docking problems.

3.6 Summary

In the words of the genetic algorithm's author, John Holland, "Genetic algorithms [are] computer programs that "evolve" in ways that resemble natural selection [that] can solve complex problems even their creators do not fully understand" [30]. There is a rigorous theory behind real-valued and bit-string strategies which describes how sampling individuals and combinations of individuals implicitly samples, to some degree, large regions in parameter space,

but that is beyond the scope of this chapter. That is, even though mutations are performed on single individuals and recombination only combines two or a few individuals, the effect is felt at the population level. Natural selection and sexual reproduction operate to modify the population as a whole, and by operating on the population, the EA employs a broad, collective heuristic to direct its search.

The field of evolutionary computation, while still somewhat young, has a solid foundation of theory, datasets for comparative testing, a plethora of techniques, and a number of software packages available. Several books have been written on EAs. There are also peer-reviewed journals, professional societies, and international conferences on the subject. Therefore, there are a number of resources for exploring the subject further and for employing and modifying an EA in almost any application.

For further information, the book *Genetic Algorithms + Data Structures = Evolution Programs* by Michalewicz [3] provides a rich introduction to the subject. Additionally, the search term "evolutionary algorithm" in Wikipedia [31] or other search engines can point you to several examples, tutorials, and other internet resources. As mentioned before, the review by Pal, Bandyopadhyay, and Ray [29] contains several references to bioinformatics examples. Finally, the search term, "genetic algorithm" in PubMed [32] will present several examples of the GA's use in biomedicine.

EAs have proven useful in several engineering and scientific disciplines including Biology. People have exploited EAs in a number of creative ways and have hybridized them with other techniques. However, there remain many outstanding problems in Biology that may exploit EAs to great effect. When the fitness landscape is complex and unknown, the time to search needs to be reduced, as is often the case in computer simulations and complex models with several variables, or when features need to be reduced or selected in classification and clustering problems, then an EA might provide a solution.

3.7 Acknowledgments

We would like to thank Christian Gagne for his assistance with Open BEAGLE [22] and Michael Hines for his assistance with integrating Open BEAGLE into the NEURON simulation environment [21] for our computational simulations. We would also like to thank Nathan Schoppa for sharing his intimate knowledge of the mitral cell and Larry Hunter for his technical critique and advice. Our simulation work was funded by NIH grants DC004657, DC006070, 5R01-LM008111-03, and DC006640.

References

1. Holland JH (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI

2. Beyer HG, Schwefel HP (2002) Evolution strategies - a comprehensive introduction, Natural Computing 1:3–52
3. Michalewicz Z (1996) Genetic Algorithms + Data Structures = Evolution Programs. Springer
4. Okabe T, Jin Y, Sendhoff B (2005) Theoretical comparisons of search dynamics of genetic algorithms and evolution strategies, In: Evolutionary Computation, 2005. The 2005 IEEE Congress on, vol. 1, 382– 389 Vol.1
5. Alander JT (1992) On optimal population size of genetic algorithms, CompEuro'92'Computer Systems and Software Engineering', Proceedings 65–70
6. Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization, Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents 41–49
7. Holland JH (1992) Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press
8. Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, Proceedings of the Parallel Problem Solving from Nature VI Conference 849–858
9. De Jong KA (1975). An analysis of the behavior of a class of genetic adaptive systems
10. Eshelman LJ, Schaffer JD (1991) Preventing premature convergence in genetic algorithms by preventing incest, Proceedings of the Fourth International Conference on Genetic Algorithms 115–122
11. Sywerda G (1989) Uniform crossover in genetic algorithms, Proceedings of the third international conference on Genetic algorithms table of contents 2–9
12. Eshelman LJ (1991) The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, Foundations of Genetic Algorithms 1:265–283
13. Eshelman LJ, Schaffer JD (1993) Real-coded genetic algorithms and interval-schemata, Foundations of Genetic Algorithms 2:187–202
14. Schaffer JD, Caruana RA, Eshelman LJ, Das R (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization, Proceedings of the third international conference on Genetic algorithms table of contents 51–60
15. Gray F (1953). Pulse code communication
16. Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, 312–317
17. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9:159–195
18. Desmaisons D, Vincent JD, Lledo PM (1999) Control of action potential timing by intrinsic subthreshold oscillations in olfactory bulb output neurons, Journal of Neuroscience 19:10727–10737
19. Davison AP, Feng J, Brown D (2000) A reduced compartmental model of the mitral cell for use in network models of the olfactory bulb, Brain Res Bull 51:393–9
20. Rubin DB, Cleland TA (2006) Dynamical mechanisms of odor processing in olfactory bulb mitral cells, Journal of Neurophysiology 96:555

21. Hines ML, Carnevale NT (1997) The neuron simulation environment, Neural Comp 9:1179–1209
22. Gagne C, Parizeau M (2002) Open beagle: A new versatile c++ framework for evolutionary computation, Late-Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO); New York City 161–168
23. Kuhn HW (2005) The hungarian method for the assignment problem, Naval Research Logistics 52:7–21
24. Li L, Tang H, Wu Z, Gong J, Gruidl M, Zou J, Tockman M, Clark RA (2004) Data mining techniques for cancer detection using serum proteomic profiling, Artificial Intelligence In Medicine 32:71–83
25. Wang M, Zhou X, King RW, Wong STC (2007) Context based mixture model for cell phase identification in automated fluorescence microscopy, BMC Bioinformatics 8:32
26. Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm, Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems] 13:44–49
27. Lavine BK, Vora MN (2005) Identification of africanized honeybees, J Chromatogr A 1096:69–75
28. Jeffries NO (2005) Performance of a genetic algorithm for mass spectrometry proteomics, feedback
29. Pal SK, Bandyopadhyay S, Ray SS (2005) Evolutionary computation in bioinformatics: A review, IEEE Transactions on Systems, Man, and Cybernetics, Part-C
30. Holland JH (1992) Genetic algorithms: Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand, Scientific American 267:66–72
31. Wikipedia. Evolutionary algorithm — Wikipedia, The Free Encyclopedia URL `http://en.wikipedia.org/w/index.php?title=Evolutionary_algorithm&oldid=138840243`
32. NCBI. Pubmed home URL `http://www.ncbi.nlm.nih.gov/sites/entrez?db=PubMed`