The RBF-Gene model

Virginie LEFORT, Carole KNIBBE, Guillaume BESLON, Joël FAVREL
e-mail : {vlefort, cknibbe, gbeslon}@prisma.insa-lyon.fr, joel.favrel@insa-lyon.fr

INSA-IF/PRISMa, 69621 Villeurbanne CEDEX, France

Abstract. We present here the Radial Basis Function Gene Model (RBF-Gene) as a new approach of evolutionary computation. This model introduces a quasi-realistic notion of gene into artificial genomes. Thus, it enables artificial organisms to evolve both on a functional basis (i.e. to enhance their fitness) and on a structural basis (i.e. to permanently reorganize their genome). This approach enables us to introduce new reproduction operators, thus giving the chromosome the opportunity to evolve its own "evolvability". This allows a better convergence. In this paper, both principles and first results are presented, showing the great interest of this model.

1 Introduction

Although Genetic Algorithms (GAs) are directly inspired of Darwinian principles, they use an over-simplistic model of chromosome and genotype to phenotype mapping. Following different authors, we argue that this simplification leads to a lack of performance, mainly (but not only) because the chromosome structure directly constrains the evolution process.

We present here the Radial Basis Function Gene Model (RBF-Gene) as a new approach of evolutionary computation. This model introduces a quasi-realistic notion of gene into artificial genomes: The genetic sequence is less constrained than it is in classical GAs (since genes are detected thanks to start sequences inside the chromosome). As a consequence, the RBF-Gene model enables us to introduce new variation operators working on large chromosomal segments (duplication, deletion, translocation...). It enables artificial organisms to evolve both on a functional basis (i.e. to enhance their fitness) and on a structural basis (i.e. to organize permanently their genome).

The influence of the chromosome structure will be discussed in sections 2 and 3, first in classical GAs and then in biology. Section 4 presents the RBF-Gene model, which includes different biologically inspired mechanisms enabling structural evolution in order to favor the convergence. This model has been tested on the abalone data set [1] (a regression task) and results are discussed in section 5.

2 Convergence and structural evolution in genetic algorithms

Darwin explained many years ago the evolution principles: For him, there are two main processes in evolution which are variation and selection of the fittest (in fact, extinction of the least fit individuals).

Classical GAs [2] are directly inspired by this vision and by the first studies on DNA in the early 50s. They are used in parametric optimization where the goal is to find the best set of parameters in order to optimize (minimize or maximize) a function. GAs operate on a population of individuals by analogy with natural evolution. Each individual is represented by a chromosome (the genotype) and represents a possible solution to the problem: The phenotype. In the traditional fixed-length algorithm, all the parameters are coded sequentially on the chromosome. The genotype is then the string representing the chromosome (a bit string in most cases) and the translation is the process that converts it into a phenotype.

A fitness function is defined to assign to each individual a fitness value. The fitter the individual, the better the solution it represents and the greater its probability to have descendants: This is the selection operator. Reproduction then allows amplification and variation from mutation and crossover: The former flips bits, while the latter exchanges bit sequences between two parents.

Since 1975, we think, thanks to Holland and his Building Blocks Hypothesis [3] that GAs do not work directly on the whole genetic sequence but on schemata. GAs first build short, good schemata and aggregate them and build longer blocks until a solution is found.

Since short building blocks are easier to find and maintain than long ones, the structure of the chromosome (i.e. the number of parameters and their order on the chromosome) directly influences the performance. Therefore, there are two types of linkage between parameters: Epistatic linkage (based on the dependence of the parameters) and physical linkage (based on the proximity on the chromosome). The more different these linkages on the chromosome, the harder the problem.

Indeed, in classical GAs the chromosome structure is given: The number of "genes" is the number of parameters needed and their order, without specific knowledge, can be considered as randomly chosen. So, there is little chance for it to match epistatic linkage. Thus, building blocks can be hard to find or will not be found at all, and the algorithm prematurely converges on local optima.

Obviously, such a behavior would be avoided if the sequence structure could be modified/optimized during the evolutionary process or *by* the evolutionary process: If the structure is not suitable, the algorithm should change it so as to ready itself for future evolution [4]. For instance, moving two functionally related genes closer may improve the efficiency of future crossing-over.

Unfortunately, evolvability [4] cannot be defined in simple GAs since the mapping from the genotype to the phenotype is based on the sequence structure (number of parameters, coding length, order of the genes along the sequence...).

The rules for the mapping are known and fixed once and for all at the beginning of the evolution cycle.

Many algorithms have tried to permit the structure to change dynamically in order to learn the epistatic linkage (dependences between genes) so as the two linkages (epistatic and physical) match. For instance, there is the well-known Messy GA [5] in which the chromosome is a sequence of pairs {allele:value}. The Messy GA introduces locus-independence and variable number of "genes" (alleles can be over- or under-specified).

However, it cannot be considered as a complete "structure-free" algorithm: The number of alleles is known and predefined. Moreover, it has to use specific, global, mechanisms to give a value to under-specified alleles and to choose the value of over-specified ones. Nevertheless, Messy GA and its alternatives give promising results, thus proving that secondary order evolution may strongly helps the convergence.

3 Some interesting biological features

Structural evolution is impossible in classical GAs because the genotype and the phenotype are interdependent. On the opposite, in biology, both levels can evolve separately. Thus, the structure of the chromosome is free to evolve. In this section, we will describe some biological processes that enable such a behavior.

3.1 Genotype-phenotype mapping

In living beings, the mapping between the genotype (the genes on the chromosome) and the phenotype (what an individual is) is not done at once. First, DNA is used to create proteins. Then, these proteins biochemically interact to build the phenotype.

In fact, the first step is a double-process. The DNA is first transcribed into RNAs. Then, proteins are translated from RNA by use of a genetic code. To a first approximation, the quantity of a protein "depends" on the transcription process while its function "depends" on the translation process.

The genetic code determines each protein sequence thanks to a translation table. In particular, in the standard genetic code, there is one "start" codon[1] (indicating the beginning of the translated regions) and three "stop" codons (indicating the end). The other codons indicate which amino acids are contained in the protein and their order. Thus, we can defined a *gene* as a coding sequence (from a start to the next stop) whatever its function is or may be.

Thanks to the translation process, most genes are coding for similar objects: Proteins[2]. Their function does not depend on their locus (i.e. their position along the chromosome)[3]. The proteins, by their interactions and biochemistry, interact and contribute to build the global being.

[1] A small sequence of consecutive nucleotides, 3 in biology

[2] Actually, in our model, we will only consider the proteins.

[3] The locus may however influences the expression level of the gene and therefore the quantity of a protein.

This “two-steps” mapping is the basic mechanism enabling the structure to evolve: Different genotypes (with different structures) can lead to the same protein “pool”, thus leading to the same phenotype. Consequently, different sequences can be equivalent on a functional basis. They can only be compared according to their evolvability.

3.2 Operators

Since the phenotype does not depend on the chromosome structure, variation operators may be used that modify this structure. Indeed, we can see in a living being a lot of operators modifying it.

There are three kinds of operators:

Local operators that locally modify the chromosome by deleting, inserting or switching a nucleotide or a small number of nucleotides.

Large operators that globally modify the structure of the chromosome by deleting, duplicating or moving large amount of genetic material. They can affect more than one gene in a single operation.

External operators that use another being’s genetic material to create a new genotype (crossing-over for sexual reproduction, lateral transfer for asexual reproduction).

All these operators may modify the structure. Even local operators can destroy/add a “start” or a “stop” codon, then destroying/adding a protein.

As the structure is free to evolve, these operators can modify the genotype and eventually promote future evolution. But these operators cannot be efficient without a primordial feature of the genetics: The non-coding sequences.

3.3 Non-coding sequence

On the chromosome, each gene is delimited by a start and a stop codon (and promoter/terminator sequences). The natural DNA contains a lot of non-coding sequences. They are of three types :

Intergenic regions These are non-coding regions *between* genes. Depending on the gene definition, some of these regions have a regulation function but many have no (known) function.

Intragenic regions (or introns) These are non-coding regions *inside* a gene. The introns are spliced out of the RNA before the translation occurs. They are only present in the eukaryotic chromosomes but not in the prokaryotic ones.

Pseudogenes These are segments of DNA similar to a functional gene (i.e. the sequence “makes sense”) but with some changes that prevent their transcription and/or their translation. They do not contribute to the fitness so they are not subject to selection pressure and as such mutate quickly. As a pseudogene mutates, its similarity to a functional gene progressively disappears. In some cases, pseudogenes can be “reactivated” after mutation, thus creating new genes.

These sequences play an important role as they improve the operation of the genetic operators. Indeed, with the non-coding sequences, the operators can operate on "building blocks" more efficiently: The non-coding sequences separate the basic blocks (or the genes) and facilitate extern and large operators.

3.4 Putting more biology in genetic algorithms

As we have seen, in GAs the non-convergence often comes from the interdependence between the chromosome structure and the phenotype. On the opposite, in biology, both levels appear to be less dependent. That is why it may be interesting to add some more natural aspects in genetic algorithms in order to improve evolvability [6–9]. However, the three biological features listed above (mapping, operators and non-coding sequences) are all of important matter. If one of these features is missing, this will be harder or even impossible.

With the Virtual Virus (VIV), Burke *et al.* [10] proposed to introduce these three biological features into a genetic algorithm: The number of genes and their sizes are free to evolve, there are non-coding sequences and large operators.

The VIV uses a genetic code to find and translate the coding sequences into words. The goal is to find three words representing three main features of the viruses. The size of a gene is linked to the size of the word it codes. Moreover, the length of coding and non-coding sequences are free to evolve thanks to biologically-inspired operators.

However, in the VIV, only the best three genes are used to compute the fitness of the individual: Burke *et al.* introduced a "daemon" that chooses which genes to keep or not. Such an entity – which is obviously biologically unrealistic – is not very different from the "over-specified" rules used in the Messy GA. It contradicts the basic principles of the mapping independence since it introduces global rules (keeping the best gene) at the local level (the gene sequence). Moreover, exactly as in the Messy GA, the alleles number is fixed and predefined. Thus, the VIV cannot be considered as a real structure-free algorithm.

4 The RBF-Gene algorithm

The non-convergence of the genetic algorithms is often due to the differences between the epistatic linkage and the physical linkage. In order to improve the results, the structure of the chromosome must be free to evolve. We propose here a new model of genetic algorithm based on these three main biological features that may improve genotype evolvability.

First, we will present the main ideas of the algorithm and then the specific implementation we have used: The RBF-Gene. We will then discuss our first results on a regression benchmark.

4.1 Principles

As seen above, the evolvability of the chromosome structure mainly rests on the existence, between the genotype and the phenotype, of an intermediate coding level (i.e. the proteins pool). Applied to genetic algorithms, such an intermediate level would allow us to compute a phenotype whatever the structure of the chromosome is.

To implement this initial idea, we have to choose what will be the basic elements composing this intermediate level. Inspired by neural networks (RBF layered networks [11]), we propose a radical change in the problem approach: Instead of trying to fit a predefined parametric function by means of its global formula, we discover the shape of this function with a linear combination of an unspecified number of basic functions[4], each of them with a fixed number of parameters. Applied to GAs, it means that each "gene" will code for one of these basic functions (called "kernels") exactly as, in biology, the genes code for proteins.

Consequently, the traditional fixed number of parameters is replaced by a variable number of kernels, depending on the number of "genes". The phenotype is obtained by the interactions of the different kernels in a linear combination. In short terms, all problems that can be defined in term of function approximation can be resolved with the algorithm (function identification, non-linear regression, classification. . .).

As far as the "genetic sequence" is concerned, we no longer consider the sequence as a predefined parameters list. Our chromosome contains variable-length coding and non-coding sequences. The latter has no influence on the phenotype while the former directly contributes to it by coding for one kernel. Coding and non-coding sequences are simply differentiated thanks to two genetic sequences: The "start" sequence and the "stop" sequence. In between, the sequence will be analyzed thanks to a "genetic code" in order to compute the kernel parameters: For example, for a triangular function, its mean, its height and the size of its base; For a Gaussian kernel, the mean, the standard deviation and the height.

The genetic code is a translation table which assigns a pair[5] {parameter:value} to each codon (i.e. fixed size subsequence). All the values obtained for a single parameter are used to compute the final parameter value (for instance, by summing the different values or using a variable-length binary code). Since the number of codons for a parameter is variable, the length of each gene may also be variable.

The length of each coding/non-coding sequence may be variable, and so may the number of kernels: therefore the length of the chromosome may vary. Moreover, since there is no global rule to analyze the sequence, it can be modified by any operator.

Indeed, we can use biologically-inspired operators: Switches, local insertions and deletions, large operators, crossover, transfer. . . The chromosome size, the

[4] The linear combination of these elementary functions must have the property to be a "universal approximator". That is, all continuous and bounded functions can be approximated with a finite sum of these basic functions (e.g. sinusoid, sigmoid, triangular, rectangular or Gaussian functions, . . .).

[5] Contrary to the Messy GA, our alleles represent functions and not directly parameters. Each one has a fixed number of parameters. As the number of alleles is variable, we have a variable number of parameters.

number of kernels, the locus of the genes and the size of the coding sequences are then free to evolve. Moreover, since genes are separated by non-coding sequences, the large operators can be very efficient: They can duplicate/delete/move some genes without affecting the other ones. Non-coding sequences may also contain pseudo-genes which may help to find new kernels "from scratch".

4.2 The implementation used : RBF-Gene

In this paper, we propose a specific implementation of this algorithm, named RBF-Gene. It can be used to approach any bounded $\mathbb{R}^n$ to $\mathbb{R}$ function from data points (i.e. for regression or classification tasks). We say that we have n input dimensions and one output dimension.

Our kernels are here Radial Basis Functions (RBF, in other words Gaussian functions). Such kernels have two parameters: The mean vector μ ($\mu \in \mathbb{R}^n$) and the standard deviation σ. A third parameter is used to compute its weight W in the linear combination. So, we have $n + 2$ reals to determine by kernel.

We use a specific "genetic code" to extract these parameters out of the sequence. In our model, the bases are characters and a codon is a single character. We have defined 2 special codons "start" and "stop". The other codons are associated to one parameter and to one value (0 or 1). We have thus $2(n + 2) + 2$ different codons.

To compute the value of a kernel parameter, we extract the sequence of codons associated with it. Their values give a binary sequence that can be translated into a real value thanks to a variable-size Gray code [12] (it ensures that to switch from any value i to $i + 1$ we only need one mutation). The Gray code gives us a result between 0 and 1. Then, the value is mapped onto the interval corresponding to the bounds of the parameter.

Given this sequence model, we initialized the population with a fixed number of individuals, each of them with a random sequence. So the number of kernels and their parameters values are random. Then, the RBF-Gene algorithm uses a reproduction loop similar to the classical GA's. However, since the sequence can evolve in structure, we introduce different variation operators (see figure 1):

- Local operators (on one nucleotide): Switch, deletion, insertion
- Large operators (on many nucleotides): Translocation, large deletion, duplication
- External operators: 1-point or 2-points crossover, transfer.

5 Experiments: The abalone regression task

5.1 Conditions

We used the algorithm on a regression benchmark: The abalone data set [1]. The goal is to find the age of an abalone given eight inputs: Sex, length, diameter, height, whole weight, shucked weight, viscera weight and shell weight. The data set contains 4177 experimental points.

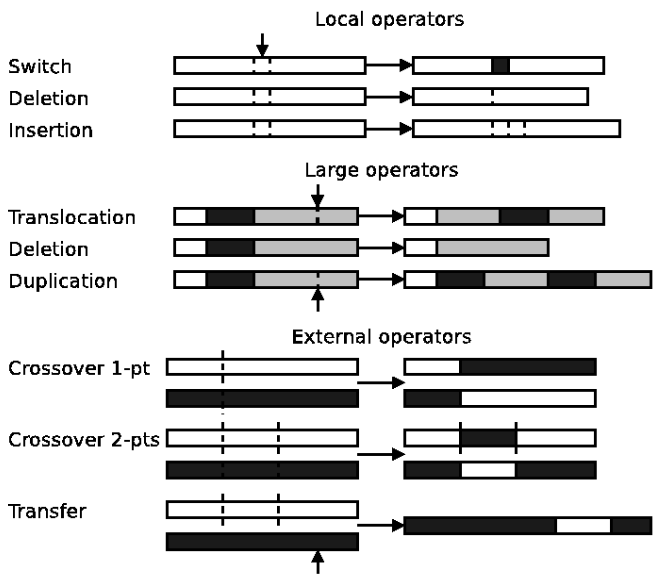


Fig. 1. The operators used in the RBF-Gene

The fitness function used is the root mean squared error:

$$fitness = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (\widehat{y_i} - y_i)^2}$$

with N the number of points, $\widehat{y_i}$ the value given by the algorithm as the output for the i^{th} point and y_i the real value of the i^{th} points.

In order to compare learning and generalization, we use two set of points: the first one (the learning set) is used for the computation of the fitness, the second one (the validation set) is used to monitor the algorithm. We use a 10-fold cross-validation: The data set is divided into ten subsets and the model is trained on all subsets but one. The remaining subset is used as the validation set. This process is repeated ten times, each subset being used for validation once.

We have done 10 runs of 5000 generations each (each with 10 fold cross-validation) so as to obtain 100 results. We use a population size of 100 individuals, with an initial genome size of 200 characters.

The rates of the operators are inspired from the studies of De Jong [13] and Brindle [14]. To obtain the next generation, we use a rank-based roulette-wheel selection and a one-point crossover in 60% of the cases. We do not use here two-points crossover.

We use local operators with a probability of 5x10^{-5} per nucleotide to be switched, deleted or to add a new one before it. The translocation and deletion have a global rate of $2x10^{-2}$ to affect once a chromosome while replicating (and $(0.02)^2$ to affect twice this chromosome, $(0.02)^3$ to be done three times...). Duplication and transfer have a global rate of $1x10^{-2}$. With these rates, there is no bias on the average size of the chromosome (as both duplication and transfer increase size and deletion decreases it in the same proportion).

5.2 Results on abalone

We can compare our results with those in Table 1 from [15]. In each simulation (of 100), we keep the best individual at the 5000^{th} generation. We obtained, for them, a average root mean squared error on training set of 2.1280 (with a standard deviation of 0.04) and on the validation set of 2.1604 (with a standard deviation of 0.10). Our algorithm obtains similar or better results than the others methods listed here.

Table 1. The different fitness

Algorithm	**Training data**	**10 fold cross-validation**
Predicting the mean	3.2238	3.2244
Multi-variate linear regression	2.1916	2.2147
Single level decision tree	2.7313	2.7522
M5' Model tree	2.0385	2.1296
RBF-Gene model	2.1280	2.1604

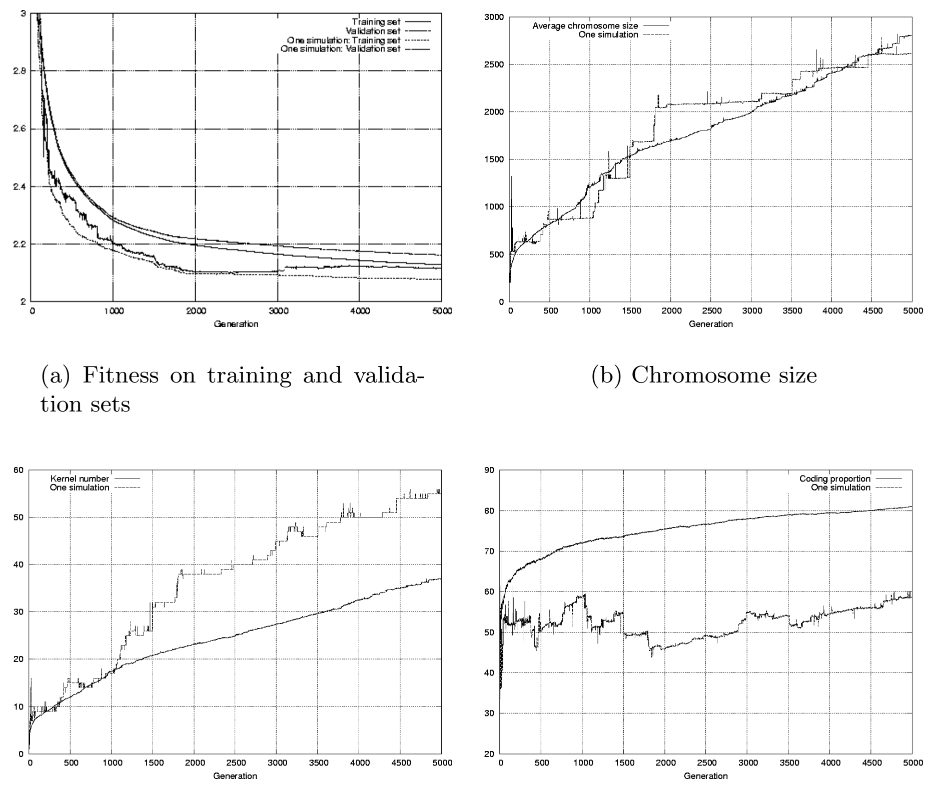
Obviously, the RBF-Gene algorithm generates good results. But the important aspect of the algorithm is the possibility to have second order evolution (i.e. to reorganize its genome while evolving functionally). So, some other indicators are interesting like evolution of sequence length, structure, number of kernels, coding proportion...

For each simulations (of 100), we keep the best individuals. Results at the last generation are presented in Table 2 and the evolution of functional and structural indicators are shown in Figure 2.

We can see two stages: First, the chromosome grows in length adding new kernels and then, stabilizes its length, optimizing the existing kernels.

An important fact is the evolution of the size. At first sight, we could think that it will grow more and more. In practice, nearly all chromosomes stabilize their size around 3000 characters (the initial one is 200 characters) representing 37 kernels[6]. Actually, for an individual, a big size is not detrimental (for the same coding sequences) and may favor the discovery of new genes. But, for the population and the evolution, a big size is penalizing: Mutations will be more

[6] Actually, figure 2b does not show this stabilization because, some chromosomes still grow after 5000 generations. In particular, one simulation diverges in length with a final size of more than 50000 characters and 300 kernels.



(a) Fitness on training and validation sets

(b) Chromosome size

(c) Kernel number

(d) Coding proportion

Fig. 2. Evolution of performance and structural indicators during the 5000 first generations. For each indicator, both the average values for the bests individuals (at each generation) and a typical run are presented.

numerous and keeping the existing structure will be harder so the evolution process will be *globally* penalized. In other words, long genomes are less *evolvable*. We cannot see such a phenomenon in the VIV as length always increases (it can be explained by the fact that only three genes are coding and that outside these genes a mutation is not deleterious).

The coding and non-coding sequences are also of important matter. At the beginning, as the chromosomes are random, there are few coding sequences (on average 30%). At the end of the runs, the coding sequences represent 82% of the chromosome. In the first generations, the percentage of the coding sequences grows faster as the algorithm adds a lot of new kernels. Then, the algorithm improves the existing kernels, modifying/enlarging its coding sequences to be more precise but the number of kernels remains steady (about 37 kernels).

Table 2. Functional and structural indicators at the last generation for the 100 best individuals (the bests of each simulation)

Indicator	**Mean**	**Minimum**	**Maximum**	**Std deviation**
Training fitness	2.1280	2.04	2.29	0.04
Validation fitness	2.1604	1.96	2.41	0.10
Chromosome size	2809.1	724	50393	5203
Kernels number	37.0	11	357	39
Coding proportion	81.6	53.0	96.7	9.5

6 Conclusions and future work

The RBF-Gene algorithm introduces a real independence between the genotype and the phenotype: All the genes are coding and the phenotype is always computable, whatever the number of genes. Moreover, the chromosome structure is dynamically modified during the evolution process to improve future evolution.

Although it is still under study, our model gives promising results, proving the importance of structural evolution. It is the fruit of a profitable collaboration between computer scientists and biologists. With some light modifications, it can be used by biologists to study evolution.

To go further, we have to test the algorithm on more functions and/or more problems. Actually, it can be used to optimize RBF layered neural networks, to do regression task or classification tasks. An interesting extension may be to use it in robotics or to explore other neural networks topologies.

Moreover, the large-scale operators act completely randomly. It seems obvious that a translocation of a gene in one parent can be deleterious for a descendant obtained by crossover. Actually, it will have 0 or 2 instances of these genes. But having twice the same gene causes it to be twice in the linear combination (i.e. with a double weight) thus being deleterious in most cases. So, future works will focus on biologically inspired large-scale operators based on sequence similarity.

References

1. UCI Machine Learning Website (http://www.ics.uci.edu/∼mlearn/MLRepository.html): Abalone data set (consulted in 2003)
2. Goldberg, D.E.: Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley (1989)
3. Forrest, S., Mitchell, M.: Relative building-block fitness and the building-block hypothesis. In Whitley, D.L., ed.: Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993) 109–126
4. Pennisi, E.: How the genome readies itself for evolution. Science **281** (1998) 1131–1134

5. Goldberg, D.E., Deb, K., Kargupta, H., Harik, G.: Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S., ed.: Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1993) 56–64
6. Wu, A.S., Lindsay, R.K.: A survey of intron research in genetics. In Voigt, H.M., Ebeling, W., Ingo, R., Hans-Paul, S., eds.: Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation. Volume 1141., Berlin, Germany, Springer-Verlag (1996) 101–110
7. Wu, A.S., Lindsay, R.K.: A comparison of the fixed and floating building block representation in the genetic algorithm. Evolutionary Computation **4** (1996) 169–193
8. Banzhaf, W.: Genotype-phenotype-mapping and neutral variation - a case study in genetic programming. In Yuval, D., Hans-Paul, S., Reinhard, M., eds.: Parallel Problem Solving from Nature III. Volume 866., Jerusalem, Springer-Verlag (1994) 322–332
9. Kargupta, H.: A striking property of genetic code-like transformations. Technical Report EECS-99-004, Department of Electrical Engineering and Computer Science, Washington State University (1999)
10. Burke, D.S., De Jong, K.A., Grefenstette, J.J., Ramsey, C.L., Wu, A.S.: Putting more genetics into genetic algorithms. Evolutionary Computation **6** (1998) 387–410
11. Simon, H.: Neural Networks, A Comprehensive Fundation. Second edition edn. Prentice Hall (1999)
12. Hinterding, R., Gielewski, H., Peachey, T.C.: The nature of mutation in genetic algorithms. In Eshelman, L., ed.: Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann (1995) 65–72
13. De Jong, K.A.: An analysis of the behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan (1975)
14. Brindle, A.: Genetic algorithms for function optimization. Technical report, Department of Computer Science, University of Alberta, Edmonton (1981)
15. Automatic Knowledge Miner (AKM) Server: Data mining analysis (request abalone). Technical report, AKM (WEKA), University of Waikato, Hamilton, New Zealand (2003)