MAX-PLANCK-INSTITUT FÜR INFORMATIK

Genetic Algorithms within the Framework of Evolutionary Computation

Proceedings of the KI-94 Workshop

Jörn Hopf (Editor)

MPI–I–94–241 August 94

Im Stadtwald
66123 Saarbrücken
Germany

Genetic Algorithms within the Framework of Evolutionary Computation

Proceedings of the KI-94 Workshop

Jörn Hopf (Editor)

MPI–I–94–241 August 94

Author's Address

Jörn Hopf (Editor)
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
F. R. Germany
e-mail: hopf@mpi-sb.mpg.de

Preface

It is a matter of fact that in Europe evolution strategies and in the U.S.A. genetic algorithms have survived more than a decade of non-acceptance or neglect. It is also true, however, that so far both strata of ideas have evolved in geographical isolation and thus have not led to recombined offspring. Now it is time for a new generation of algorithms which make use of the rich gene pool of ideas on both sides of the Atlantic.

It is certain that today there are three different schools whose roots have developed independently from each other:

- Evolutionary Programming (EP)
- Evolution Strategies (ESs)
- Genetic Algorithms (GAs)

Genetic Programming (GP) and Classifier Systems (CSs) are both special subbranches of the GA philosophy.

First roots of EP were established with *Artificial Intelligence through Simulated Evolution* by L. J. Fogel, A. J. Owens and M. J. Walsh, 1966, of ESs with *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* by I. Rechenberg, 1973, and *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* by H.-P. Schwefel, 1977, and of GAs with *Adaption in Natural and Artificial Systems* by J. H. Holland, 1975, and *An Analysis of Behavior of a Class of Genetic Adaptive Systems* by K. De Jong, 1975.

Only since 1990 contacts between these different schools take place regularly. In 1991, on the fifth International Conference on Genetic Algorithms the generic term *Evolutionary Algorithms (EAs)* for EP, ES and GA was agreed on. The journal *Evolutionary Computation* published by MIT Press since 1993 is supported by all three groups.

EC does not abandon traditional methods. None of the EAs would do the job better or even as good as those. EC should be taken into consideration if good old and well underpinned methods either do not exist, are not applicable, or fail.
Evolutionary approaches play a considerable role in Artificial Life, a divergence from classical Artificial Intelligence, control, planning, combinatorial optimization and many other areas. This workshop surveys the state of the art, presents examples and case studies as well as attempts at systematization. In addition, due to this workshop, a research area which has already tackled real-world problems with considerable success receives more attention.

This year the first workshop on Evolutionary Computation (EC) will take place on the German Annual Conference on Artificial Intelligence (*KI-94*).
It starts with two invited lectures: *Quantitative Experimental Studies of Darwinian Evolution* by Christof K. Biebricher, Max-Planck-Institute for physical Chemistry, one of the leading researcher in evolution and *The Science of Breeding and its Application to the Breeder Genetic Algorithm* by Heinz Mühlenbein, German National Research Center for Computer Science (GMD), a leading researcher who covers both, fundamental research and real applications. This is followed by two tutorials: *Artificial Life and Selforganisation* by Wolfgang Banzhaf, Dortmund University and *Genetic Programming* by Frank Klawonn, University of Braunschweig.
I hope that this gives a little overview and motivation for more reasarch in this field. On the basis of a very good national and international participation our workshop includes the following main topics:

- Theory of Evolutionary and Genetic Computation
- Biological Principles

- Genetic Programming
- Artificial Life
- Genetic Algorithms for Neural Network Design
- and many Applications

Finally, I want to thank the authors for their contributions, as well as all those, especially Hans-Jürgen Appelrath (University of Oldenburg), Wolfgang Banzhaf (Dortmund University), Volker Claus (University of Stuttgart), Heinz Mühlenbein (GMD, St. Augustin) and Lothar Thiele (University of Saarbrücken), who all helped me in organizing this workshop.
I have to thank Hans-Paul Schwefel (Dortmund University), who gave me historical information for this preface and support to the workshop even though he will not be able to take part.
It is desireable and already foreseeable that the part of AI presented with this workshop will gain ground on this side of the Atlantic, too.

Saarbrücken, August 1994 Jörn Hopf

Contents

Quantitative Experimental Studies of Darwinian Evolution

Christof K. Biebricher
Max–Planck–Institute for physical Chemistry
37077 Göttingen, Germany

Evolution is a highly efficient self-organization process where optimal solutions are obtained by iterative mutation and selection. The underlying molecular phenomena for the biological reproduction, mutation and selection have been identified: The genetic program is encoded as a linear sequence of nucleotides in the genomic double helical DNA, where the two strands are held together by the base-pairing of complementary nucleotides. DNA is reproduced by unwinding of the double helix and completion of both strands to double helices. The accuracy of copying is limited and occasional errors or mutations take place.

The genetic program is decoded by a complicated decoding apparatus. Eventually, this 'expression' of the genotype leads to certain observable properties of the organism in the pertaining environment, i.e. to a ph enotype whose 'fitness' is evaluated globally. During reproduction, the genetic program and the decoding apparatus must be copied. Darwinian evolution is not bound to living organisms, but takes place in any process comprising metabolism, reproduction and mutation. Indeed, the only system where evolution could be studied experimentally has been a system (Spiegelman et al., 1965) where RNA molecules are replicated by an enzyme called replicase. For the quantitative description of the evolution process in this system we used the expressions in the quantitative theory of molecular evolution developed by Eigen and coworkers (Eigen & Schuster, 1977; Eigen & Biebricher, 1988). In this system, evolution can be studied in the test tube, because the sequence complexity is low, the mutations rates are high, the population sizes are very high, and the expression of the genotype into the phenotype is simple: the phenotype of a species is simply its efficiency in replication.

The mechanism of RNA replication has been investigated in detail and is well understood (Biebricher et al., 1981, 1982,1983, 1984, 1985, 1991). Replicase binds a single-stranded RNA template, synthesizes a complementary replica and releases the template. While replicase is present in excess, the RNA grows exponentially. When the replicase is saturated with template, linear growth results. Single-stranded complementary strands may react to double strands that are unable to replicate. Different species growing in the same solution compete with each other and the selection behaviour can be investigated. In the exponential growth phase, RNA species are selected for their fecundity, i.e. the overall replication rate. In the early linear growth phase, replicase is limited and species that bind replicase most efficiently have the highest selection value. At still higher RNA concentrations, species are also selected for minimizing the loss by double strand formation. Quantitive selection values can be calculated predicting precisely the complicate selection behaviour.

Mutations occur during replication and the mutants produced compete with each other. A mutant distribution called quasispecies is formed where each mutant has a certain relative population according to its rate of formation by mutation and by its selection value. Due to compensating mutations, a quasispecies contains multi-error mutants that are nearly selectively neutral. When the environment changes, the quasispecies adapts rapidly by selecting the best mutant and building a new quasispecies around it. Evolution is very effective in leading to the optimum in a contiguous mountainous region in the fitness landscape, but can not reach the global optimum when this optimum is lying in a well-separated mountain. By raising the population size and by increasing the step size of mutational jumps, the optimization can be improved.

References

[1] Biebricher, C.K., Eigen, M. & Luce, R.

(1981) *J. Mol. Biol.* **148**, 391–410.

[2] Biebricher, C.K., Diekmann, S. & Luce (1982) *J. Mol. Biol.* **154**, 629–648.

[3] Biebricher, C.K., Eigen, M. & Gardiner, W.C. (1983) *Biochemistry* **22**, 2544–2559.

[4] Biebricher, C.K., Eigen, M. & Gardiner, W.C. (1984) *Biochemistry* **23**, 3186–3194.

[5] Biebricher, C.K., Eigen, M. & Gardiner, W.C. (1985) *Biochemistry* **24**, 6550–6560.

[6] Biebricher, C.K., Eigen, M. & Gardiner, W.C. (1991) In: *Biologically inspired Physics* (Peliti, L., ed.) NATO ASI Series B, Vol. 263, pp. 317–337, Plenum Press, New York.

[7] Eigen, M. & Biebricher, C.K. (1988) In: *RNA Genetics* Vol. III: Variability of RNA Genomes (Domingo, E., Ahlquist, & Holland, J.J., eds.) pp. 211–245. CRC Press, Boca Raton, Fl.

[8] Eigen, M. & Schuster, P. (1977) *Naturwissenschaften* 64, 541–565.

[9] Spiegelman, S., Haruna, I., Holland, I.B., Beaudreau, G., & Mills, D.R. (1965) *Proc. Natl. Acad. Sci. USA* 54, 919–927.

A Predictive Theory of the Breeder Genetic Algorithm

Heinz Mühlenbein, Dirk Schlierkamp-Voosen
GMD Schloss Birlinghoven
D-53757 Sankt Augustin1
e-mail: muehlen@gmd.de

Abstract–The Breeder Genetic Algorithm BGA models artificial selection as performed by human breeders. We show how the response to selection equation and the concept of heritability can be applied to predict the behavior of the BGA. The theoretical results are obtained under the assumption of additive gene effects. For general fitness landscapes advanced statistical techniques for estimating the heritability are used to analyze and control the BGA.

1 Introduction

Evolution of natural organisms is based on three major components - reproduction, variation and selection. Some reproductions of natural organisms occur with "failures" called mutations. A more systematic variation of the genetic material happens in sexual reproduction. Each parent contributes half of its genetic material to the offspring. This method of variation is called recombination. The offspring will be identical to the parents if the parents are genetically equal.

Variation is necessary to allow selection to work. Selection in nature is very difficult to define precisely. The term was introduced by Darwin [5] very informally. "*The preservation of favourable variations and the rejection of injurious variations, I call Natural Selection.*" But how can an observer predict which are the favorable variations? The favorable variations are the variations which are preserved! The variations can only be judged after they have competed in the "struggle for life." Natural selection is no independent force of nature, it is the result of the competition of natural organisms for resources.

In contrast, in the science of breeding the above problem does not exist. The selection is done by human breeders. Their strategies are based on the assumption that mating two individuals with high fitness more likely produces an offspring of high fitness than two randomly mating individuals. The *Breeder Genetic Algorithm* **BGA** introduced in [13] is based on the science of breeding. The science is part of applied statistics. A major component is the regression of parent and offspring.

In this paper we deal mainly with a rather simplified model. We assume additive gene contributions and uniform crossover. Nevertheless five parameters are needed to describe the initial state of the population and the selection process. The necessary parameters are

- the population size N
- the initial frequency of the desirable allele p_0
- the number of loci n
- the mutation rate m
- the intensity of selection I

For this model we have computed the expected number of generations until convergence. It would be futile to investigate the model with all five parameters variable. Therefore we will investigate the model with one or more parameters fixed. The outline of the paper is as follows.

First we will investigate evolution without selection, also called *genetic drift* ($I = 0$). If there is no mutation the population will converge to a unique genotype. In section three we will analyze selection and recombination in large populations. The analysis is based on the *response to selection equation* and on the concept of *heritability*. Then the major theoretical results are summarized.

The above theory gives a clear picture about the behavior of the major evolutionary components. For the breeder genetic algorithm this theory plays the same role as the ideal gas theory for classical thermodynamics. The "ideal gas" in evolutionary algorithms are simple additive fitness functions. For these simple fitness functions the behavior of the breeder genetic algorithm is already complex.

In section 5 it is shown how the theory can be extended to arbitrary fitness functions. The key problem is to estimate the heritability. In section 6 the theory is applied to a number of fitness functions.

The theory presented here is based on classical livestock breeding and population genetics. Some of the results presented in this paper are also of interest for population genetics. Our models are restricted to haploid organisms. But in this area our models and equations are sometimes more precise than the ones used in population genetics. Examples are the analysis of genetic drift and the analysis of the genetic variance. For a recent survey about predicting the response to selection in livestock productions see [17].

2 Evolution without selection - genetic drift

It has been known in population genetics for quite some time that a finite population converges to a single genotype, even if selection is not applied. The mutation rate is assumed to be negligible. The fixation of the population is a result of its finite size. This effect has been called *genetic drift* by Wright [18]. The importance of genetic drift for explaining evolution in nature has been emphasized by Kimura [8]. He developed a neutral theory of molecular evolution, claiming that natural selection is not as important for evolution as previously surmized. Kimura used a very complex diffusion equation approach to quantify genetic drift [4]. We will generalize his results. Two chance models will be distinguished

1. no selection, no recombination
2. no selection, but with recombination

The first model is just sampling with replacement. The second model is an adaptation of Mendel's genetic chance model to haploid organisms. For the analysis of genetic effects the following cases will be distinguished if necessary:

- one gene with two alleles
- n genes each with two alleles
- n genes with an infinite number of alleles

The last case roughly models the genetic representation used by the BGA for continuous functions of n variables. In all cases, recombination is done by randomly choosing an allele from one of the parents. For binary representations this recombination scheme is called *uniform crossover* [16].

The next three theorems have been derived in [2]. The proofs are based on a Markov chain analysis for one gene with two alleles. The formulas have been obtained by numerically fitting the data.

Theorem 1 *Let there be a gene with two alleles. Let half of the initial population have allele* 0*, the other allele* 1*. Then in a randomly mating population of size N without mutation and recombination, the expected number of generations until equilibrium GEN_e is given by*

$$E(GEN_e) \approx 1.4 \cdot N \tag{1}$$

If the number of genes or the number of alleles is very large, GEN_e is only slightly larger. This is shown in the next theorem.

Theorem 2 *Let the number of genes or alleles be large enough, that the genotypes of the initial population are all different from each other. Then in a randomly mating population of size N without mutation and recombination, the expected number of generations until equilibrium GEN_e is approximately*

$$E(GEN_e) \approx 2 \cdot N \tag{2}$$

N	16	32	6 4	128	256	512
GEN_e	29.4	60.3	128.0	245.2	546.0	1131.1
SD	16.7	33.5	72.1	121.1	294.9	736.5

Table 1: Gen_e for a large number of genes

In table 1 numerical results from simulations are given. They are averages over 10,000 runs. Note the very large standard deviation SD.

The theorems are in agreement with the results of Crow and Kimura [4]. They obtained for diploid chromosomes twice as large values, i.e $GEN_e = 2.8N$ and $GEN_e = 4N$.

The next theorem gives the convergence time if recombination is applied. It is restricted to binary representations. This theorem is new.

Theorem 3 *Let each gene have two alleles. Let the size of the chromosome be n, the size of the population be N. Let the initial population be randomly generated. Then for a randomly mating population with no selection, but with uniform crossover, the expected number of generations until equilibrium is approximately*

$$E(GEN_e) \approx 1.4 \cdot N \cdot (0.5ln(n) + 1)^{1.1} \tag{3}$$

Table 2 gives some results of BGA simulations. One clearly observes that GEN_e increases linearly with the popsize N and only logarithmically with the size of the problem n. This result shows that recombination is not able to substantially reduce the influence of genetic drift. We will later show that genetic drift is indeed an important factor if small selection intensities are used.

	N		
n	16	32	64
32	67.1	131.0	261.9
64	77.6	160.2	334.2
512	107.7	224.0	475.4
1024	123.6	247.6	504.3
4096	141.6	289.0	

Table 2: GEN_e for different n and $N = 16, 32, 64$ with recombination (two alleles)

The results for an infinite number of alleles case are qualitatively similar. To summarize some results obtained by simulations, they show that GEN_e scales as $N \cdot ln(n)$, similar to the binary case. It seems that the value of GEN_e for an infinite number of alleles is about the value of GEN_e for the binary case with twice as many genes. The popsizes are held equal.

In the next section we will analyze selection and recombination in large populations.

3 Response to selection

In this section we summarize the theory presented in more detail in [13],[14]. The change produced by selection that mainly interests the breeder is the *response to selection*, which is symbolized by R. R is defined as the

difference between the population mean fitness of generation $t+1$ and the population mean of generation t. $R(t)$ measures the expected progress of the population.

$$R(t) = M(t+1) - M(t) \tag{4}$$

where $M(t)$ denotes the average of the population at generation t. Breeders measure the selection with the *selection differential*, which is symbolized by S. It is defined as the difference between the mean fitness of the selected parents $M_s(t)$ and the mean fitness of the population.

$$S(t) = M_s(t) - M(t) \tag{5}$$

Breeders often use *truncation selection* or *mass selection*. In truncation selection with threshold T, the T % best individuals will be selected as parents. T is normally chosen in the range 10% to 50%.

The prediction of the response to selection starts with

$$R(t) = b_t \cdot S(t) \tag{6}$$

b_t is called *realized heritability* in quantitative genetics. The breeder either measures b_t in previous generations or estimates b_t by different methods. Two popular methods based on the regression of parents to offspring will be explained later. It is normally assumed that b_t is constant for a certain number of generations. This leads to

$$R(t) = b \cdot S(t) \tag{7}$$

There is no genetics involved in this equation. It is simply an extrapolation from direct observation. The prediction of just one generation is only half the story. The breeder (and the GA user) would like to predict the cumulative response R_s for s generations of his breeding scheme.

$$R_s = \sum_{t=1}^{s} R(t) = b \sum_{t=1}^{s} S(t) \tag{8}$$

The response to selection is the product of the heritability and the selection differential. For predicting the response to selection b and the selection differential have to be estimated.

If the fitness values are normal distributed, the selection differential $S(t)$ in truncation selection is approximately given by

$$S(t) = I \cdot \sigma_p(t) \tag{9}$$

where σ_p is the phenotypical standard deviation. I is called the *selection intensity*. The formula is a feature of the normal distribution. A derivation can be found in [3].

The science of artificial selection consists of estimating b and $\sigma_p(t)$. We just cite the following theorem [13]. It was proven for the ONEMAX function under the assumption that $\sigma_p(t)$ has a binomial distribution

$$\sigma_p(t) = \sqrt{n \cdot p(t) \cdot (1 - p(t))}$$

$p(t)$ is the probability of the advantageous allele in the population at generation t.

Theorem 4 *Let the breeder genetic algorithm be run with uniform crossover. If the population is large enough that it converges to the optimum and if the selection intensity I is greater than 0, then the response to selection is given for the ONEMAX function by*

$$R(t) = I \cdot \sqrt{n \cdot p(t)(1 - p(t))} \tag{10}$$

The number of generations needed until equilibrium is approximate

$$GEN_e = \left(\frac{\pi}{2} - arcsin(2p_0 - 1)\right) \cdot \frac{\sqrt{n}}{I} \tag{11}$$

$p_0 = p(0)$ denotes the probability of the advantageous bit in the initial population.

We next compare the analytical results with simulations. In figure 1 the mean fitness versus the number of generations is shown for three popsizes $N = 1024, 256$, and 64. The selection intensity is $I = 0.8$, the size of the problem $n = 64$. The initial population was generated with $p_0 = 1/64$.

A closer look at the simulation results show that the phenotypic variance is slightly less than given by the binomial distribution. The empirical data is better fitted if the following estimate is used

$$\tilde{\sigma}_p(t) = \frac{\pi}{4.3}\sqrt{n \cdot p(t) \cdot (1 - p(t))} \tag{12}$$

Using this variance one obtains the equations

$$\tilde{R}(t) = \frac{\pi}{4.3} \cdot I \cdot \sqrt{n \cdot p(t)(1 - p(t))} \tag{13}$$

$$G\tilde{E}N_e = \frac{4.3}{\pi}\left(\frac{\pi}{2} - arcsin(2p_0 - 1)\right) \cdot \frac{\sqrt{n}}{I} \tag{14}$$

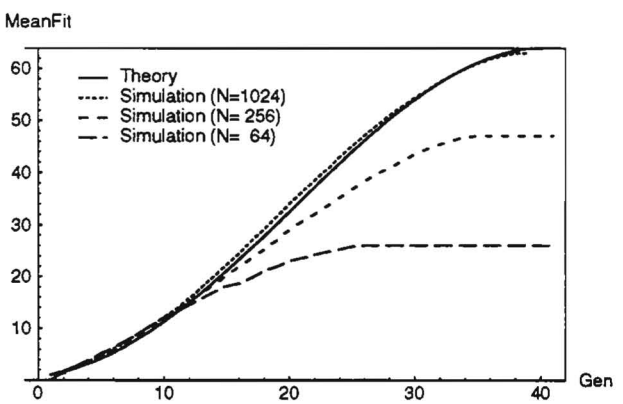


Figure 1: Mean fitness for various N ($T = 0.5, p_0 = 1/64$). $N = 64$ converges first.

The fit of equation 13 and the simulation run with $N = 1024$ is very good. For $N = 256$ and $N = 64$ the population does not converge to the optimum. These popsizes are less than the *critical popsize* $N^*(I, n, p_0)$. The critical popsize is defined to be the minimum popsize that the BGA converges with high probability to the optimum. The problem of determining the critical popsize will be discussed later.

We have not been able to prove a similar theorem for an infinite number of alleles. The difficulty lies in estimating the variance of the population. We will give some simulation results in the next section.

For proportionate selection which is used by the simple genetic algorithm we extend the theorem already proven in [13].

Theorem 5 *For a genetic algorithm using proportionate selection the selection differential is given by*

$$S(t) = \frac{\sigma_p^2(t)}{M(t)} \tag{15}$$

For the ONEMAX function of size n the response to selection can be computed from

$$R(t) = 1 - p(t) \tag{16}$$

If the population is large enough, the number of generations until $p(t) = 1 - \epsilon$ is given for large n by

$$GEN_{1-\epsilon} \approx n \cdot ln\frac{1 - p_0}{\epsilon} \tag{17}$$

p_0 is the probability of the advantageous allele in the initial population.

Proof *We will only prove 17. For $ONEMAX(n)$ we have $R(t) = S(t)$. As before we approximate the variance by the variance of the binomial distribution*

$$\sigma_p^2(t) \approx np(t)(1 - p(t)) \tag{18}$$

Because $M(t) = np(t)$, equation 16 is obtained. From $R(t) = n(p(t+1) - p(t))$ we get the difference equation

$$p(t+1) = \frac{1}{n} + (1 - \frac{1}{n})p(t) \tag{19}$$

This equation has the solution

$$p(t) = \frac{1}{n}\left(1 + (1 - \frac{1}{n}) + \cdots + (1 - \frac{1}{n})^{t-1}\right) + (1 - \frac{1}{n})^t p_0$$

This equation can be simplified to

$$p(t) = 1 - (1 - \frac{1}{n})^t(1 - p_0)$$

By setting $p(GEN_{1-\epsilon}) = 1 - \epsilon$ equation 17 is easily obtained.

This theorem shows the problem of proportionate selection. It selects too weak if the population approaches the optimum.

Both theorems of this section assume large popsizes. Due to space limitations we will just summarize our major theoretical results in the next section..

4 Summary of the major theoretical results

In this section we will just survey the major results which can be found in [9],[13],[14],[2],[1]. They are valid for fitness functions with additive gene effects. Let n denote the number of genes, N the size of the population.

We first consider populations with recombination and no mutation. Any finite population of size N will converge to a single genotype, even if selection is not applied. This effect is called *genetic drift*. The number of generations until convergence GEN_e is surprisingly low.

$$GEN_e \propto N \cdot ln(n) \quad no\_sel, rec, no\_mut \tag{20}$$

We now turn to truncation selection. If the size N of the population is larger than the *critical popsize* N^*, the minimum popsize to converge to the optimum with high probability, then we have

$$GEN_e \propto \frac{\sqrt{n}}{I} \quad trunc\_sel, rec, no\_mut, N \geq N^* \tag{21}$$

Note that GEN_e is independent of N. The estimation of the critical popsize is very difficult. The dependence of N^* from I is nonlinear. Simulations have shown that N^* increases for large selection intensities and for small selection intensities [14]. For small selection intensities this behavior seems surprisingly. But the reason is the *genetic drift* which reduces the variance of the population. We conjectured

$$N^* = \sqrt{n} \cdot ln(n) \cdot f_1(p_0) \cdot f_2(I) \tag{22}$$

Proportionate selection as used in the simple GA [7] selects too weak when the variance of the population gets small. The expected number of generations $GEN_{1-1/n}$ until the favorable allele is distributed in the population with probability of $1 - 1/n$ is given by

$$GEN_{1-1/n} \propto n \cdot ln(n) \quad prop\_sel, rec, no\_mut, N \gg 0 \tag{23}$$

This number is much larger than with truncation selection. The analysis of recombination in small populations is difficult. We have shown in [14] the results in phase diagrams relating the posize and GEN_e. The phase diagrams can be divided into two areas. The border is given by the critical popsize N^*.

We now turn to populations using only mutation. Mutation is a random search operator especially efficient in small populations. The most important result concerns the mutation rate. The mutation rate is defined as the probability of mutating a gene.

Rule of thump: *The mutation rate $m = 1/n$ where n is the size of the chromosome is almost optimal [10].*

For the above mutation rate the expected number of generations GEN_{opt} until the optimum is found has been computed for the $(1+1)$–strategy (one parent, one offspring; the better of the two survives).

$$GEN_{opt} \propto n \cdot ln(n) \quad sel, no\_rec, mut, N = 2 \tag{24}$$

Mutation in large population is inefficient. The scaling remains the same as for $N = 2$. But it is still twice as efficient as proportionate selection with recombination [14].

$$GEN_{1-1/n} \propto n \cdot ln(n) \qquad sel, no\_rec, mut, N \gg 0 \quad (25)$$

For binary fitness functions, populations using either recombination or mutation are able to locate the optimum. Moreover, the asymptotic order of the number of trials needed (FE_{opt}), seems to be the same, namely $O(n \cdot ln(n))$. For recombination this number is obtained by multiplying GEN by the critical popsize N^*. Therefore the question which of the two operators is more efficient is difficult to answer. The comparison needs an exact expression for N^*, which we have not yet obtained. But we can easily make a qualitative comparison. The major difference between mutation and recombination is their dependence on p_0, the percentage of the desired allele in the initial population.

Let us take $p_0 = 1 - 1/n$ as example. Then just one bit of a chromosome is wrong on the average. Mutation will need about $O(n)$ trials to change the incorrect bit. Uniform crossover of two strings, each with one bit wrong, will generate the optimum string with probability 1/4, independent of the size of the problem. Therefore the critical popsize N^* is also independent of n. Thus recombination is much more efficient than mutation. But the determination of the exact N^* is also difficult in this simple case. It will need on the average 4 trials to generate the optimum. But the probability that a popsize of 4 will not generate the optimum is $0.75^4 = 0.31$. It needs 16 trials in order to obtain the optimum with 99% probability.

If we take $p_0 = 1/n$ the situation is reversed. Here only one bit is correct on the average. Now mutation is much more efficient than recombination which needs a huge popsize in order to locate the optimum. It is obvious that mutation is more successful than recombination when far from the optimum. Recombination has too few building blocks to generate better offspring. But recombination is more effective than mutation near the optimum. Here the success of a mutation is the lowest.

A more detailed comparison between mutation and recombination, also by means of a competition between populations can be found in [12]. We now turn to general fitness functions.

5 Statistics and genetics

In this section we will present two methods for estimating the *heritability*. The first one will use the concept of *regression of offspring to parent* and the second one the concept of *genetic variance*. Both methods have been of great importance in the development of statistics and population genetics. Therefore we will first give a short historical survey.

Genetics represents one of the most satisfying applications of statistical methods. Modern statistics starts with Galton and Pearson who found at the end of the last century a striking empirical regularity. On the average a son is halfway between his father and the overall average height for sons. They used data from about 1000 families. In order to see this regularity Galton and Pearson invented the scatter diagram, regression and correlation [6].

Independently Mendel found some other striking empirical regularities like the reappearance of a recessive trait in one-fourth of the second generation hybrids. He made up a chance model involving what are now called *genes* to explain his rules. He conjectured these genes by pure reasoning - he never saw any.

At first sight, the Galton-Pearson results look very different from Mendel's, and it is hard to see how they can be explained by the same biological mechanism. Indeed Pearson wrote an article in 1904 claiming that his results cannot be derived by Mendel's laws. About 1920 Fisher, Wright and Haldane more or less simultaneously recognized the need to recast the Darwinian theory as described by Galton and Pearson in Mendelian terms. They succeeded in this task, but unfortunately much of the original work is abstruse and very difficult to follow. The difficulty lies in the exact definition of *genetic variance* and its connection to *heritability*. We will in this section adapt the classical methods to haploid chromosomes. Furthermore we will precisely define the concepts.

The first theorem connects the realized heritability $b_t = R(t)/S(t)$ with the regression coefficient between *midparent* and offspring. Let f_i, f_j be the phenotypic values of parents i and j, then

$$\bar{f}_{i,j} = \frac{f_i + f_j}{2}$$

is called the midparent value. Let the stochastic variable $\bar{F}$ denote the midparent value.

Theorem 6 *Let $F(t) = (f_1, \ldots, f_N)$ be the population at generation t, where f_i denotes the phenotypic value of individual i. Assume that an offspring generation $O(t)$ is created by random mating, without selection. If the regression equation*

$$o_{ij}(t) = a(t) + b_{\bar{F}O}(t) \cdot \frac{f_i + f_j}{2} + \epsilon_{ij} \qquad (26)$$

with

$$E(\epsilon_{ij}) = 0$$

is valid, where o_{ij} is the fitness value of the offspring of i and j, then

$$b_{\bar{F}O}(t) \approx b_t \qquad (27)$$

Proof *From the regression equation we obtain for the expected averages*

$$E(O(t)) = a(t) + b_{\bar{F}O}(t) M(t)$$

Because the offspring generation is created by random mating without selection, the expected average fitness remains constant

$$E(O(t)) = M(t)$$

Let us now select a subset as parents. The parents will be randomly mated, producing the offspring generation. If the subset is large enough, we may use the regression equation and obtain for the averages

$$M(t+1) = a(t) + b_{\bar{F}O}(t) \cdot M_s(t)$$

Here $M(t+1)$ is the average fitness of the offspring generation produced by the selected parents. Subtracting the above equations we obtain

$$M(t+1) - M(t) = b_{\bar{F}O}(t) \cdot (M_s(t) - M(t))$$

This proves $b_{\bar{F}O}(t) = b_t$.

The importance of regression for estimating the heritability was discovered by Galton and Pearson. They computed the regression coefficient rather intuitively by scatter diagrams of midparent and offspring [6]. The problem of computing a good regression coefficient is solved by the theorem of Gauss-Markov. We just cite the theorem. The proof can be found in any textbook on statistics [15].

Theorem 7 *A good estimate for the regression coefficient of midparent and offspring is given by*

$$b_{\bar{F}O}(t) = \frac{cov(O(t), \bar{F}(t))}{var(\bar{F}(t))} \tag{28}$$

The covariance of O and $\bar{F}$ is defined by

$$cov(O(t), \bar{F}(t)) = \frac{1}{N} \sum_{i,j} (o_{i,j} - av(O(t))) \cdot (\bar{f}_{i,j} - av(\bar{F}(t)))$$

av denotes the average and var the variance. Closely related to the regression coefficient is the correlation coefficient $cor(\bar{F}, O)$. It is given by

$$cor(\bar{F}(t), O(t)) = b_{\bar{F}O}(t) \cdot (\frac{var(\bar{F}(t))}{var(O(t))})^{1/2}$$

The above theorem enables us to estimate the heritability by a second method. It works as follows. For a large sample population F the offspring have to be created by random mating. Then the regression coefficient $b_{\bar{F}O}$ can be computed by equation 28. This procedure is more robust than dividing $R(t)$ by $S(t)$. First, it works also in the case of small selection intensity. Second, the trustworthiness of the computation can be estimated by statistical techniques.

By the above method an average value for the heritability is computed. The average is taken over the whole domain. For the breeder genetic algorithm we decided to proceed slightly differently. The regression coefficient is only computed for the *selected parents* and their offspring. This local approximation makes it possible to compute regression coefficients which depend on the given population and the local fitness landscape.

The next theorem shows the connection between *midparent* and *parent* regression.

Theorem 8 *Midparent and parent regression are connected by*

$$b_{FO}(t) = 0.5 \cdot b_{\bar{F}O}(t) \tag{29}$$

$$cor(F(t), O(t)) = \sqrt{\frac{1}{2}} cor(\bar{F}(t), O(t)) \tag{30}$$

Proof *We have*

$$cov(O(t), \bar{F}(t)) = cov(O(t), F(t))$$

$$var(\bar{F}(t)) = 0.5 \cdot var(F(t))$$

From (28) the theorem is obtained.

We now describe a method for estimating the covariance. This method connects a microscopic genetic chance model and the macroscopic phenotypic covariance. It is restricted to discrete genes. In this paper we only give the necessary definitions and the fundamental theorem. The interested reader is refered to [1] where the proof can be found. A detailed computation is given for a diploid chromosome with two genes in [4].

Let a haploid chromosome with n binary genes x_i be given, $f(\mathbf{x})$ its fitness. Let the genetic chance model be defined by *uniform crossover*. This model can be considered as Mendel's chance model restricted to haploid chromosomes. We will decompose the fitness value $f(\mathbf{x})$ recursively into an additive part and interaction parts. Let $p(\mathbf{x})$ denote the probability of $\mathbf{x}$, $p(\mathbf{x}|x_i)$ the conditional probability of $\mathbf{x}$ given x_i. First we extract the average.

$$f(\mathbf{x}) = av(f) + r_0(\mathbf{x}) \tag{31}$$

Then we extract the first order (additive) part from the residual $r_0(\mathbf{x})$.

$$r_0(\mathbf{x}) = \sum_{i=1}^{n} f_{(i)}(x_i) + r_1(\mathbf{x}) \tag{32}$$

where $f_{(i)}(x_i)$ are given by

$$f_{(i)}(x_i) = \sum_{\mathbf{x}|x_i} p(\mathbf{x}|x_i) r_0(\mathbf{x}) = \sum_{\mathbf{x}|x_i} p(\mathbf{x}|x_i) f(\mathbf{x}) - av(f)$$

Here $\sum_{\mathbf{x}|x_i}$ means that the i-th locus is fixed to the value x_i. The $f_{(i)}(x_i)$ minimize the quadratic error $\sum_{\mathbf{x}} p(\mathbf{x}) r_1(\mathbf{x})^2$.
If $r_1(\mathbf{x}) \not\equiv 0$, we can proceed further to extract the second order terms from $r_1(\mathbf{x})$:

$$r_1(\mathbf{x}) = \sum_{\substack{(i,j) \\ i<j}} f_{(i,j)}(x_i, x_j) + r_2(\mathbf{x}) \tag{33}$$

where

$$f_{(i,j)}(x_i, x_j) = \sum_{\mathbf{x}|x_i,x_j} p(\mathbf{x}|x_i, x_j)\ r_1(\mathbf{x})$$

$$= \sum_{\mathbf{x}|x_i,x_j} p(\mathbf{x}|x_i, x_j)\ f(\mathbf{x}) - f_{(i)}(x_i) - f_{(j)}(x_j)$$

If we have n loci, we can iterate this procedure $n-1$ times recursively and finally we get the decomposition of f as

$$f(\mathbf{x}) = \bar{f} + \sum_i f_{(i)}(x_i) + \sum_{(i,j)} f_{(i,j)}(x_i, x_j) + \cdots + \sum_{\substack{(i_1,\ldots,i_{n-1}) \\ i_1<\ldots<i_{n-1}}} f_{(i_1,\ldots,i_{n-1})}(x_{i_1}, \ldots, x_{i_{n-1}}) + r_{n-1}(\mathbf{x})$$

Let V_k for $k = 1$ to $n-1$ be defined as

$$V_k = \sum_{\substack{(i_1,\ldots,i_k) \\ i_1<\ldots<i_k}} \sum_{x_{i_1},\ldots,x_{i_k}} p(x_{i_1},\ldots,x_{i_k}) f_{(i_1,\ldots,i_k)}(x_{i_1},\ldots,x_{i_k})^2, \tag{34}$$

and

$$V_n = \sum_{\mathbf{x}} p(\mathbf{x}) r_{n-1}(\mathbf{x})^2 \tag{35}$$

We are now able to formulate the fundamental theorem.

Theorem 9 *Let the population be in linkage equilibrium i.e.*

$$p(\mathbf{x}) = \prod_{i=1}^{n} p_i(x_i) \tag{36}$$

Then the variance of the population is given by

$$var(F) = V_1 + V_2 + \cdots + V_{n-1} + V_n \tag{37}$$

The covariance of midparent and offspring can be computed from

$$cov(\bar{F}, o) = \frac{1}{2}V_1 + \frac{1}{4}V_2 + \cdots + \frac{1}{2^n}V_n = \sum_{k=1}^{n} \frac{1}{2^k} V_k \tag{38}$$

From theorems 7 and 9 we obtain

Corollary 1 *If the fitness function is additive that is,* $f(\mathbf{x}) = \sum_i f_i(x_i)$, *then*

$$cor(\bar{F}, O) = \sqrt{1/2} \qquad b_{\bar{F}O} = 1 \tag{39}$$

The above theorem plays an important role in the science of breeding. Breeders conjecture that the *additive genetic variance* V_1 is the most important factor of the heritability. The higher order interactions contribute much less to the heritability. Therefore they can be neglected. We will test this conjecture in a forthcoming paper.

Numerically, decomposing the variance is computationally far too expensive to be of use for the breeder genetic algorithm. But the regression technique is very simple to implement. We will show in the next section that the regression technique can be used to control and guide the breeder genetic algorithm.

6 Numerical applications of the theory

From statistics and population genetics it is known that the regression coefficient should be a reliable estimate for heritability in the case of continuous fitness functions and large populations. Therefore as a first example we take the minimization of the hypersphere. The BGA for continuous functions has been described in [13]. It uses a floating point representation. In figure 2 scatter diagrams of midparent and offspring at generation 1 and 30 are shown. In this example only discrete recombination is used, no mutation. It is easily seen that the whole population is moving towards the global minimum, which is 0 in this example. The regression coefficient is almost exactly one in both diagrams as predicted by the theory.

In figure 3 the numerical values of the two different estimates for the heritability are shown ($R(t)/S(t)$ and the regression coefficient). Both estimates oscillate around 1 as predicted. The correlation coefficient is about 0.5.

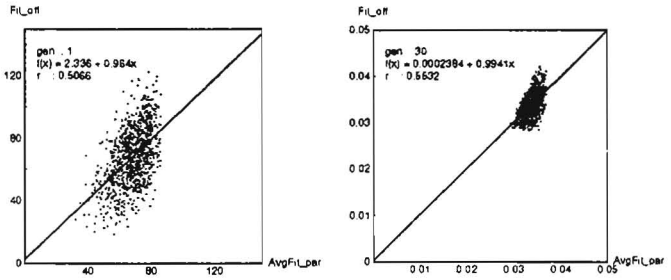


Figure 2: Scatter diagrams for generations 1 and 30 for the hypersphere. Only discrete recombination is used (N=1024,T=0.5).

This is less than the maximum value possible, which is $\sqrt{0.5}$. The reason for this difference is the selection. The selection reduces the variance of the parents and therefore the correlation coefficient.

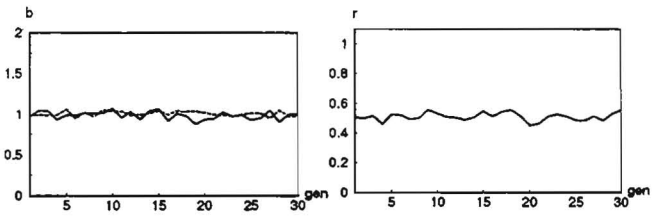


Figure 3: Heritability estimates (regression coefficient solid line, $R(t)/S(t)$ dashed line) and correlation coefficient r for the hypersphere (N=1024,T=0.5).

We just report the results for a simulation run without selection. In this case the $R(t)/S(t)$ estimator cannot be used because S(t) is about 0. The regression coefficient can be computed as usual and remains 1. Furthermore the correlation coefficient is about $\sqrt{0.5}$ as predicted by the theory.

The above results are not restricted to simple unimodal functions. As the next example we take the highly multimodal function which is known as Schwefel's function $F7$ [13].

$$F_7 = \sum_{1}^{n} -x_i \, sin\left(\sqrt{|x_i|}\right) \qquad -500 \le x_i \le 500 \tag{40}$$

The theory predicts that the multimodality of this function can be considered more or less as noise for the BGA. It should have no major influence on the regression coefficient. Indeed, with random mating, the regression coefficient is 1 and the correlation coefficient between midparent and parent is about $\sqrt{0.5}$, just as for the hypersphere. Figure 4 shows a real BGA simulation run with selection, recombination *and* mutation. One clearly observes that the search is first driven by recombination, then by mutation. From generation 17 on, the regression coefficient substantially differs from the ratio estimator $R(t)/S(t)$. Now the search is mainly driven by the random operator mutation. The BGA mutation scheme is described in [13].

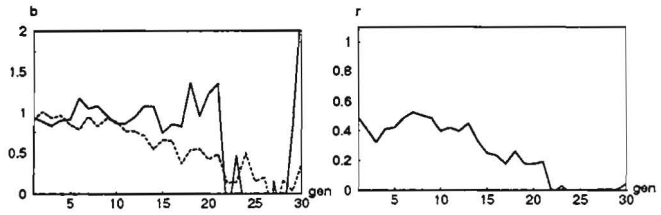


Figure 4: Heritability estimates b with mutation and recombination ($N = 256$). The correlation coefficient r drops to zero. The regression coefficient (solid line) and the ratio estimator (dashed line) are almost equal at the beginning. Then the ratio $R(t)/S(t)$ goes to zero whereas the regression coefficient remains high till generation 22.

Next we turn to binary functions. We take as examples

- ONEMAX(n)
- PLATEAU(20,3)
- DECEP(10,3)

$PLATEAU(20,3)$ has a string length n of 60. An increase in fitness is allocated only if three consecutive bits at loci 1,3,6,.. are 1's. In each case, the fitness is increased by 3. $DECEP(10,3)$ is the deceptive function defined by Goldberg [9].

In figure 5 the results of a BGA run are shown for $ONEMAX(64)$ with a truncation threshold of $T = 0.5$ and uniform crossover, but without mutation. The two heritability estimates coincide fairly well. They are about 1, as predicted. The correlation coefficient is about 0.5 till generation 14. This is less than the correlation coefficient without selection, which is $\sqrt{0.5}$. At the end of the run the correlation coefficient increases. This behavior indicates that the genotypes of the selected parents are becoming very similar. Therefore the offspring are very similar to both parents.

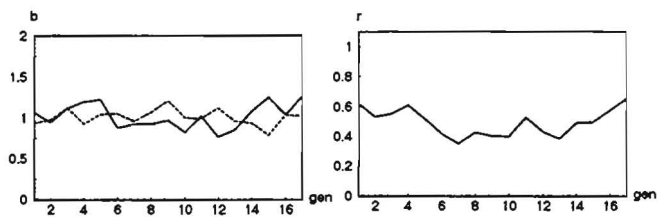


Figure 5: Heritability b estimates (regression coefficient solid line, $R(t)/S(t)$ dashed line) and correlation coefficient r with recombination only for ONEMAX(64) ($N = 128, T = 0.5$)

Our next example is the PLATEAU function. We will discuss PLATEAU(20,3) and PLATEAU(20,5). PLATEAU(20,5) has a plateau of size 5, therefore it is more difficult to optimize. Without selection the regression coefficients for the two functions are about 0.7 and 0.4, the correlation coefficients are about 0.5 and 0.3. In figure 6 we have used a truncation threshold of $T = 0.5$. For both functions the regression coefficients are substantially higher than without selection. This indicates that selection is very effective for this fitness function. But note that the realized heritability $R(t)/S(t)$ is considerably smaller than the regression coefficient. For PLATEAU(20,5) it substantially increases during the run.

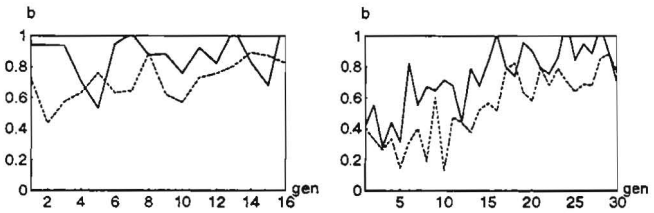


Figure 6: Heritability b estimates (regression coefficient solid line) for PLATEAU(20,3) and (20,5)

The last example is the deceptive function DECEP(10,3). This function is called deceptive, because the search is guided into the local optimum $(0,0,0)$. The global optimum is at $(1,1,1)$. Without selection, the regression coefficient is about 0.5 and the correlation coefficient about 0.35. This is shown in figure 7.

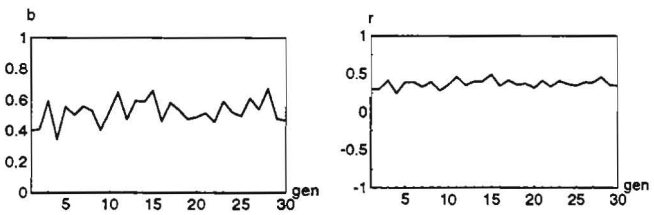


Figure 7: Heritability b and correlation r estimate with recombination for DECEP(10,3), no selection ($N = 256$)

The behavior radically changes with selection. If selection is applied, both the regression coefficient and the ratio estimator become erratic. Half of the time they are negative. This shows selection with this fitness function works against crossover and vice versa.

For binary functions the heritability can also be estimated by decomposing the genetic variance. We have already used this method for the ONEMAX function. But the numerical implementation for the general case is prohibitive. The method of decomposing the variance will numerically be useful if the first term, the additive genetic variance V_1 is sufficient for estimating the heritability. We must postpone this investigation.

To summarize this section: *The theory presented is especially applicable for continuous functions. For many continuous fitness functions the regression coefficient will be 1, the maximum possible. For binary functions the regression coefficient and the realized heritability give useful information about the complexity of the fitness landscape and how to guide the search of the breeder genetic algorithm.*

7 Conclusion

Efficient evolutionary algorithms for optimization should be based on the science of breeding rather than on natural selection. The breeder genetic algorithm BGA con-

nects the theory of genetic algorithm with classical population genetics and statistics. Some of the results, already known in the science of breeding, have been extended or made more precise. Several possible improvements need further study. One example is to use nonlinear regression techniques for estimating the heritability in complex fitness landscapes.

We believe, that it has been a big mistake in the theory of genetic algorithms, that researchers tried to develop a new theory without looking into the theory already developed in population genetics. It took the most famous statisticians and population genetics researchers almost half a century to derive at the theory presented here. The so called schema theorem [7] which is the starting point of the conventional GA theory is either a tautology or it is incorrectly used. The fundamental theorem of section 5 is a generalized version of Fisher's fundamental theorem of natural selection. It correctly describes the development of a genetic population. By comparing this theorem with the schema theorem one easily detects why the schema theorem has no predictive power.

The BGA solves the problem of how to scientifically breed a population. We hope that genetic algorithm research in the future concentrates on the real problem remaining - how to find a good representation for the given application. For combinatorial optimization problems the representation problem is discussed in [9],[11].

Acknowledgement: Part of this research has been funded by the REAL WORLD COMPUTING programme under the project SIFOGA.

References

[1] H. Asoh and H. Mühlenbein. Estimating the heritability by decomposing the genetic variance. Technical report, GMD, Sankt Augustin, 1994.

[2] H. Asoh and H. Mühlenbein. On the mean convergence time of genetic populations without selection. Technical report, GMD, Sankt Augustin, 1994.

[3] M. G. Bulmer. *"The Mathematical Theory of Quantitative Genetics"*. Clarendon Press, Oxford, 1980.

[4] J. F. Crow and M. Kimura. *An Introduction to Population Genetics Theory*. Harper and Row, New York, 1970.

[5] Ch. Darwin. *The Origins of Species by Means of Natural Selection*. Penguin Classics, London, 1859.

[6] D. Freedman, R.Pisani, R. Purves, and A. Adhikkari. *Statistics second edition*. W.W. Norton, New York, 1991.

[7] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[8] M. Kimura. *The neutral theory of molecular evolution*. Cambridge University Press, Cambridge University Press, 1983.

[9] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337, San Mateo, 1991. Morgan-Kaufman.

[10] H. Mühlenbein. How Genetic Algorithms Really Work: Mutation and Hill-climbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 15–26, Amsterdam, 1992. North-Holland.

[11] H. Mühlenbein. Parallel Genetic Algorithms in Combinatorial Optimization. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research*, pages 441–456, New York, 1992. Pergamon Press.

[12] H. Mühlenbein and D. Schlierkamp-Voosen. Analysis of Selection, Mutation and Recombination in Genetic Algorithms. *Neural Network World*, 3:907–933, 1993.

[13] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization. *Evolutionary Computation*, 1:25–49, 1993.

[14] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm. *Evolutionary Computation*, 1:335–360, 1994.

[15] C.R. Rao. *Linear Statisticcal Inference and Its Application*. Wiley, New York, 1973.

[16] G. Syswerda. Uniform crossover in genetic algorithms. In H. Schaffer, editor, *3rd Int. Conf. on Genetic Algorithms*, pages 2–9, San Mateo, 1989. Morgan Kaufmann.

[17] E. Verrier, J.J. Colleau, and J.L. Foulley. Methods for predicting response to selection in small populations under additive genetic models: a review. *Livestock Production Science*, 29:93–114, 1991.

[18] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proc. 6th Int. Congr. on Genetics*, pages 356–366, 1932.

Artificial Life and Selforganisation

Wolfgang Banzhaf

Department of Computer Science, Dortmund University

Baroper Str. 301, 44221 Dortmund, GERMANY

banzhaf@tarantoga.informatik.uni-dortmund.de

Foreword

The following article reports on work presented at the 4th International Workshop on Artificial Life, held from July 6 - 8, 1994 at MIT in Cambridge, Massachusetts. It will appear in *Proceedings of ALIFE IV*, R. Brooks and Pattie Maes (Eds.), MIT Press, Cambridge, MA, 1994 and refers to the second part of my talk.

Abstract

We discuss a system of autocatalytic sequences of binary numbers. Sequences come in two forms, a 1-dimensional form (operands) and a 2-dimensional form (operators) that are able to react with each other. The resulting reaction network shows signs of emerging metabolisms. We discuss the general framework and examine specific interactions for a system with strings of length 4 bits. A self-maintaining network of string types and parasitic interactions are shown to exist.

1 Introduction

Sequences of binary numbers are the most primitive form of information storage we know today. They are able to code any kind of man-made information, be it still or moving images, sound waves and other sensory stimulations, be it written language or the rules of mathematics, just to name a few. As the success of von-Neumann computers has shown over the last 50 years, binary sequences are also sufficient to store the commands that drive the execution of computer programs. In fact, part of the success of the digital computer was due to the universality of bits and their interchangeability between data and programs.

It is not far-fetched to expect that the physical identity between operators (programs) and operands (data) may also play an essential role in self-organisation. We have proposed to consider a simple self-organising system [1], in which sequences of binary numbers are able to react with each other and sometimes even to replicate themselves. This ability of binary strings was a result of the proposition to consider binary strings similar to sequences of nucleotides in RNA. RNA sequences which presumably stood at the cradle of life [2, 3], seem capable of self-organisation and come in at least two alternative forms, a one-dimensional genotypic form and a two or three-dimensional phenotypic form. We proposed to consider binary strings in analogy and to provide for a second, folded and operative form of strings. Technically, we considered as this alternative a two-dimensional matrix form that is able to perform operations on other one-dimensional binary strings.

2 Reactions between binary strings

The fundamental ideas of this model have been outlined elsewhere (see ref. [1],[4],[7] for details). Here we only give a brief overview of what has been learned so far.

Let us consider sequences

$$\vec{s} = (s_1, s_2, ..., s_i, ..., s_N)\,. \tag{1}$$

of binary symbols $s_i \in \{0, 1\}, i = 1, ..., N$ organised in 1-dimensional strings.

Then we ask the question: Does there exist an alternative form of these strings, that is (i) reversibly transformable into the form (1), and is (ii) operative on form (1)? The answer is surprisingly simple and well known from mathematics: Yes, there are operators with the above capabilities, known as matrices.

Thus, we require the existence of a mapping $\mathcal{M}$

$$\mathcal{M} : \vec{s} \rightarrow \mathcal{P}_{\vec{s}} \tag{2}$$

which transforms $\vec{s}$ into a corresponding 2-dimensional matrix form $\mathcal{P}_{\vec{s}}$ of the sequence which should be unique and reversible. This mapping is simply a spatial reorganisation of the information contained in a sequence and

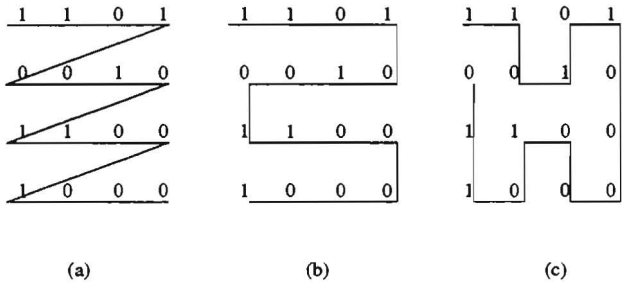


Figure 1: Some two-dimensional compactly folded forms of a string in an example with $N = 16$ binary numbers: $\vec{s} =$ (1101001011001000).
(a): non-topological folding, (b) and (c): topological foldings.

may be termed a *folding*, in close analogy to the notion used in molecular biology.

The most compact realization of such a 2-dimensional form would be a quadratic matrix. For a string with a quadratic number of components $N, N \in \mathcal{N}_{sq}$ with $\mathcal{N}_{sq} = \{1, 4, 9, 16, 25, ...\}$, the procedure is straightforward: Any systematic folding (examples are shown in Figure 1) would do. Since folding is not yet very sophisticated, and different configurations may be obtained by a renumbering of string components, we shall consider here the topological folding of Figure 1 (b) only.

In the more general case of N being a non-quadratic number, different generalizations are reasonable. Here we shall only discuss a compact folding [1] in non-square matrices, where

$$N_1 \times N_2 = N. \tag{3}$$

In order to treat non-quadratic cases similar to the quadratic case, a bias should be used in the direction of the most compact solution, i.e.

$$N_i = \sqrt{N} + \varepsilon_i, \qquad i = 1, 2 \tag{4}$$

with $\mid \varepsilon_i \mid$ as small as possible.

Table 1 gives the resulting 2-dimensional form for strings up to $N = 10$. One can see that strings with a length corresponding to a prime number are somewhat special as they do not allow any compactification in the 2-dimensional form.

The interaction between a 2-dimensional form of a string and a 1-dimensional form can be considered a reaction between the two strings. As an example, let us assume an operator $\mathcal{P}_{\vec{s}}$ was formed from string $\vec{s}$. This operator might now "react" with another string, $\vec{s}'$, producing thereby a new string $\vec{s}''$:

$$\mathcal{P}_{\vec{s}}\ \vec{s}' \Rightarrow \vec{s}'' \tag{5}$$

The notion here is that some sort of raw material (analogous to energy-rich monomers in Nature) is continuously supplied to allow the ongoing production of new strings based on the information provided by the cooperation of $\mathcal{P}_{\vec{s}}$ and $\vec{s}'$.

[1] Compact foldings do not have any spacing between adjacent string elements

Length	Compact folded form
1	(s_1)
2	$(s_1\ \ s_2)$
3	$(s_1\ \ s_2\ \ s_3)$
4	$\begin{pmatrix} s_1 & s_2 \\ s_4 & s_3 \end{pmatrix}$
5	$(s_1\ \ s_2\ \ s_3\ \ s_4\ \ s_5)$
6	$\begin{pmatrix} s_1 & s_2 & s_3 \\ s_6 & s_5 & s_4 \end{pmatrix}$
7	$(s_1\ \ s_2\ \ s_3\ \ s_4\ \ s_5\ \ s_6\ \ s_7)$
8	$\begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ s_8 & s_7 & s_6 & s_5 \end{pmatrix}$
9	$\begin{pmatrix} s_1 & s_2 & s_3 \\ s_6 & s_5 & s_4 \\ s_7 & s_8 & s_9 \end{pmatrix}$
10	$\begin{pmatrix} s_1 & s_2 & s_3 & s_4 & s_5 \\ s_{10} & s_9 & s_8 & s_7 & s_6 \end{pmatrix}$

Table 1: Compact topological string folding with length up to $N = 10$. Each folding comes also with the transposed matrix.

A typical example of an interaction is given in Figure 2 for the simple case of strings of the same quadratic length N. $\vec{s}'$ might be considered as concatenated from $\sqrt{N}$ segments with length $\sqrt{N}$ each. The operator $\mathcal{P}_{\vec{s}}$ acts on each of these segments sequentially, and performs semi-local operations. In this way, it moves down the string in steps of size $\sqrt{N}$ until it has finally completed the production of a new string $\vec{s}''$.

The particular algorithm for assembling new components "0" and "1" into strings that we have examined in more detail, is:

$$s'_{i+k\sqrt{N}} = \sigma\left[\sum_{j=1}^{j=\sqrt{N}} P_{ij} s_{j+k\sqrt{N}} - \Theta\right] \tag{6}$$

$$i = 1, ..., \sqrt{N} \qquad k = 0, ..., \sqrt{N} - 1$$

with $\sigma[\ ]$ being the squashing function

$$\sigma[x] = \begin{cases} 1 & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{7}$$

and Θ used as an adjustable threshold. Eq. (6) may be interpreted as a combination of Boolean operations, applied separately in each segment k of the string if $\Theta = 1$.

The consistent generalization of eq. (6) for interaction of non-quadratic strings and for strings of different length is straightforward: Suppose a matrix of size $N_1 \times N_2$ is interacting with a string of length N_3. The operator locally interacts with N_1 elements of the second string in order to generate one component of the new string. This operation will be repeated N_2 times, then the operator moves on to interact with the next

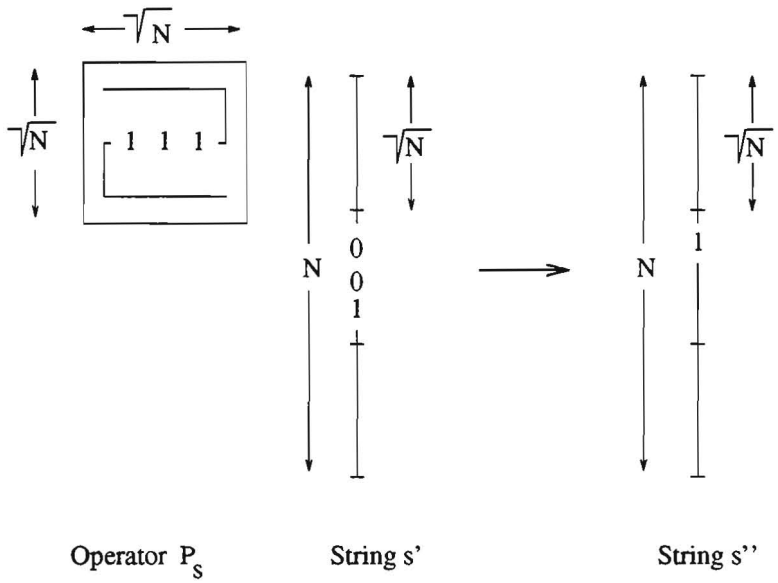


Figure 2: An operator $\mathcal{P}_{\vec{s}}$ of matrix dimension $\sqrt{N} \times \sqrt{N}$ (derived from string $\vec{s}$) acts upon a string $\vec{s}'$ consisting of $\sqrt{N}$ segments of length $\sqrt{N}$ each to produce a new string $\vec{s}''$.

N_1 elements of the second string. The newly produced string will thus consist of N_4 elements with

$$N_4 = \lceil \frac{N_3}{N_1} \rceil \times N_2. \tag{8}$$

where $\lceil x \rceil$ are Gaussian parentheses giving the next larger integer to x.

In mathematical terms, the interaction reads:

$$s''_{i+kN_2} = \sigma \left[\sum_{j=1}^{j=N_1} P_{ij} s'_{j+kN_1} - \Theta \right] \tag{9}$$

with

$$i = 1, ..., N_2 \qquad k = 0, ..., \lceil \frac{N_3}{N_1} \rceil - 1.$$

This interaction is generally length-changing — either resulting in a shorter or a longer product strings. The particular direction of this length-change depends on the relation of N_1 to N_2: If $N_1 > N_2$ then $N_4 < N_3$, and the new string is shorter. If, however, $N_1 < N_2$ then $N_4 > N_3$, and the new string is longer.

The different types of possible reactions between strings are listed in Table 2. For a given N, say $N = 4$, the reactions form a reaction network, and we shall observe in the next Section the behavior of such a network.

3 Dynamics of a sample system

Every reaction vessel is only able to keep a finite number of strings, say M. The reactions discussed in Section 2, however, continuously produce new strings. Therefore, a competitive dynamics has to be implemented by providing for an overflow mechanism for the reaction vessel. Since at present we would like a well-stirred reaction vessel without any spatial structure for reactions, the removal of strings will be a random process, hitting each sort of strings with a probability proportional to its concentration. For each newly produced string, one string is removed from the vessel. Whereas this random process does not influence in any way the constitution of the vessel content, due to different reaction channels producing new strings, a change in the composition of the content will happen over time.

Reactants	Product	Description
$s + s'$	s''	Heterogeneous reaction
$s + s'$	s	Replication
$s + s'$	s'	Replication
$s + s$	s'	Heterogeneous self-reaction
$s + s$	s	Self-replication

Table 2: Characterization of different polymerization reactions.

There are, however, some potentially "lethal" strings in such systems. A string is said to be lethal if it is able to replicate in an unproportionally large number in almost any ensemble configuration. For eq. (6), this happens to be the case for two self-replicating string types [2], $s^{(0)} = (0, 0, ..., 0, 0)$ and $s^{(2^{N-1})} = (1, 1, ..., 1, 1)$. The former is able to replicate with every other string, the latter with most of the other strings.

[2] We shall name strings with decimal numbers corresponding to their binary sequence

In order to balance this tendency of the system we prohibit production of $s^{(0)}$ and discourage production of $s^{(2^{N-1})}$. In other words, $s^{(0)}$ will not be added to the vessel, if the reaction product should be $s^{(0)}$. Instead, a randomly selected string will be copied. We deal with $s^{(2^{N-1})}$ in a more gentle way by providing a means of non-deterministic string removal due to decay processes. The fewer the number of "1"'s a string contains, the more stable it becomes. The chance to decay therefore depends on the string feature

$$I^{(k)} = \sum_{i=1}^{N} s_i^{(k)}, \qquad k = 1, ..., M. \tag{10}$$

$I^{(k)}$ measures the number of "1"'s in string k and determines a probability

$$p^{(k)} = (I^{(k)}/N)^n \tag{11}$$

which determines whether a string should be removed. Usually, we set the parameter n to $n = 1$. In any case, the decay probability of $s^{(2^{N-1})}$ is 1. Once a string decays, its place might be filled
(i) with a later reaction product or
(ii) with a copy of a randomly selected string in the vessel. The latter method has the advantage of allowing a constant string number M in the vessel and is adopted here.

One sweep through the algorithms hence consists of the following steps:

STEP 1:

Generate M random binary strings of length N each

STEP 2:

Select a string and fold it into an operator by forming a compact matrix

STEP 3:

Select another string and apply the operator generated in STEP 2

STEP 4:

Release the new string, the old string and the operator (as string) into the reaction vessel, provided it is not an $s^{(0)}$. Otherwise go to STEP 2.

STEP 5:

Remove one randomly chosen string in order to compensate for the addition of a string in STEP 4

STEP 6:

Select one string and substitute it according to the probability of (11) with the copy of a randomly selected string

STEP 7:

Go to STEP 2

M sweeps through this algorithm are called a generation.

For a discussion of the system's dynamic behaviour we use as observables the concentrations $x_i(t)$ of all the different string types $s^{(i)}$ with:

$$x_i(t) = m_i(t)/M \tag{12}$$

where $m_i(t)$ is the number of actual appearances of string type $s^{(i)}$ in the vessel at time t.

If we run a system by seeding it with an initial composition of M random strings, we regularly observe a transition into a (mostly fixed point) attractor. Due to different rates of production of different sorts, an initial composition will change until an equilibrium is reached. During the transition, new sorts are produced, already present sorts disappear, and every now and then a coexistence between sorts is reached for some time. As long as new sorts are created by interactions between already present sorts, the network has to reorganise itself in order to incorporate the newly emerging reaction channels between the different sorts. After some time, however, no new string sorts arrive, and the system reaches a steady state. Thus, the system behaves as one of the metabolic networks that are discussed in Bagley et. al. [5, 6]. As long as we have a small number of sorts, we can easily describe the system by a set of deterministic differential equations for the time development of string sort concentrations.

Deterministic rate equations were derived in [1] and are given here as a summary: We assume continuous non-random concentration functions $y_i(t)$ of the different string types $i, 1 \leq i \leq n_S$, which are considered to approximate the time averaged concentrations $<x_i>_t$:

$$y_i(t) \cong <x_i>_t, \qquad 0 \leq y_i(t) \leq 1 \tag{13}$$

The deterministic rate equations in $y_i(t)$ read:

$$\dot{y}_i(t) = A(t)y_i(t) + \left[B_i y_i(t) + \sum_{k \neq i}^{n_S} C_{ik} y_k(t) - D_i \right] y_i(t) +$$

$$\sum_{j,k \neq i}^{n_S} W_{ijk} y_j(t) y_k(t) - \frac{y_i(t)}{\sum_k y_k(t)} \Phi(t) \tag{14}$$

where B_i, C_{ik}, W_{ijk} are coupling constants derived from a reaction table containing all sorts $1...n_S$. D_i determines a selection term

$$D_i = p^{(i)} \tag{15}$$

and $A(t)$ reflects the addition of strings due to random copies

$$A(t) = \sum_{i,j}^{n_S} a_{ij} y_i(t) y_j(t) + \sum_{i}^{n_S} D_i y_i(t) \tag{16}$$

where

$$a_{ij} = \begin{cases} 1 & \text{if the reaction of } s^{(i)} \text{ and } s^{(j)} \text{ produces } s^{(0)} \\ 0 & \text{otherwise .} \end{cases} \tag{17}$$

Operator	*String*														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	2	3	2	3	0	1	0	1	2	3	2	3
2	0	1	1	0	0	1	1	2	2	3	3	2	2	3	3
3	1	1	1	2	3	3	3	2	3	3	3	2	3	3	3
4	0	4	4	0	0	4	4	8	8	12	12	8	8	12	12
5	1	4	5	2	3	6	7	8	9	12	13	10	11	14	15
6	0	5	5	0	0	5	5	10	10	15	15	10	10	15	15
7	1	5	5	2	3	7	7	10	11	15	15	10	11	15	15
8	4	0	4	8	12	8	12	0	4	0	4	8	12	8	12
9	5	0	5	10	15	10	15	0	5	0	5	10	15	10	15
10	4	1	5	8	12	9	13	2	6	3	7	10	14	11	15
11	5	1	5	10	15	11	15	2	7	3	7	10	15	11	15
12	4	4	4	8	12	12	12	8	12	12	12	8	12	12	12
13	5	4	5	10	15	14	15	8	13	12	13	10	15	14	15
14	4	5	5	8	12	13	13	10	14	15	15	10	14	15	15
15	5	5	5	10	15	15	15	10	15	15	15	10	15	15	15

Table 3: Reactions table for the simulations of a $N = 4$ system. It was generated using a variant of (6) with topological folding.

Finally, $\Phi(t)$ is a flow term that enacts competition between the various string sorts $s^{(i)}$ by enforcing constancy of the overall sum of concentrations.

The reaction table listing the interactions between string types (cf. Table 3) can be used to derive interaction graphs for various situations. In Figure 3 we have depicted all interaction graphs that can be generated from Table 3 if we start the reaction vessel with one out of 2^{N-1} string types (here $N = 4$). Functionally identical graphs are not depicted. Figure 3 illustrates the variety of interactions emerging from a start with different string types. It ranges from self-replication over parasitic interaction to entire metabolisms. From an interaction graph it is evident, what kind of attractor may be approached.

The dynamics of the parasitic interactions of Figure 3 is examined by integrating eq. (14). Figure 4 - 6 show the results of a simulation. The transition of the string composition is clearly visible. In [1, 4] we have shown that simulations on the reaction level agree completely with the integration of rate equations used here.

A simple metabolism emerges if we do not start with one sort only, but with two or more from the outset. Figure 7 shows the interaction graph of this self-maintaining network of reactions. This graph is somewhat special as each reaction channel is of nearly equal strength. A search through the space of all combinations of 2 initial sorts uncovers that the self-replicator $s^{(12)}$ plays some special role. Usually, as soon as even a spurious concentration of $s^{(12)}$ is present, together with one other sort (except $s^{(1)}$), the metabolic attractor emerges. Figure 8 gives two examples.

It is interesting to note that there are many closed subsets of elements within even a simple $N = 4$ system. In Tables 4, 5 we give a complete list of them, ordered according to their complexity in terms of participating string sorts. Following [8], a closed subset is defined as the set $\mathcal{A}^*$ of elements from the ensemble of string types $\mathcal{N}_S = \{s^{(1)}, s^{(2)}, ..., s^{(2^{N-1})}\}$,

$$\mathcal{A}^* \subseteq \mathcal{N}_S \tag{18}$$

that might be produced by all different sequences of n reactions,

$$R_n(\mathcal{A}) = \cup_{i=0}^{n} r_n(\mathcal{A}) \tag{19}$$

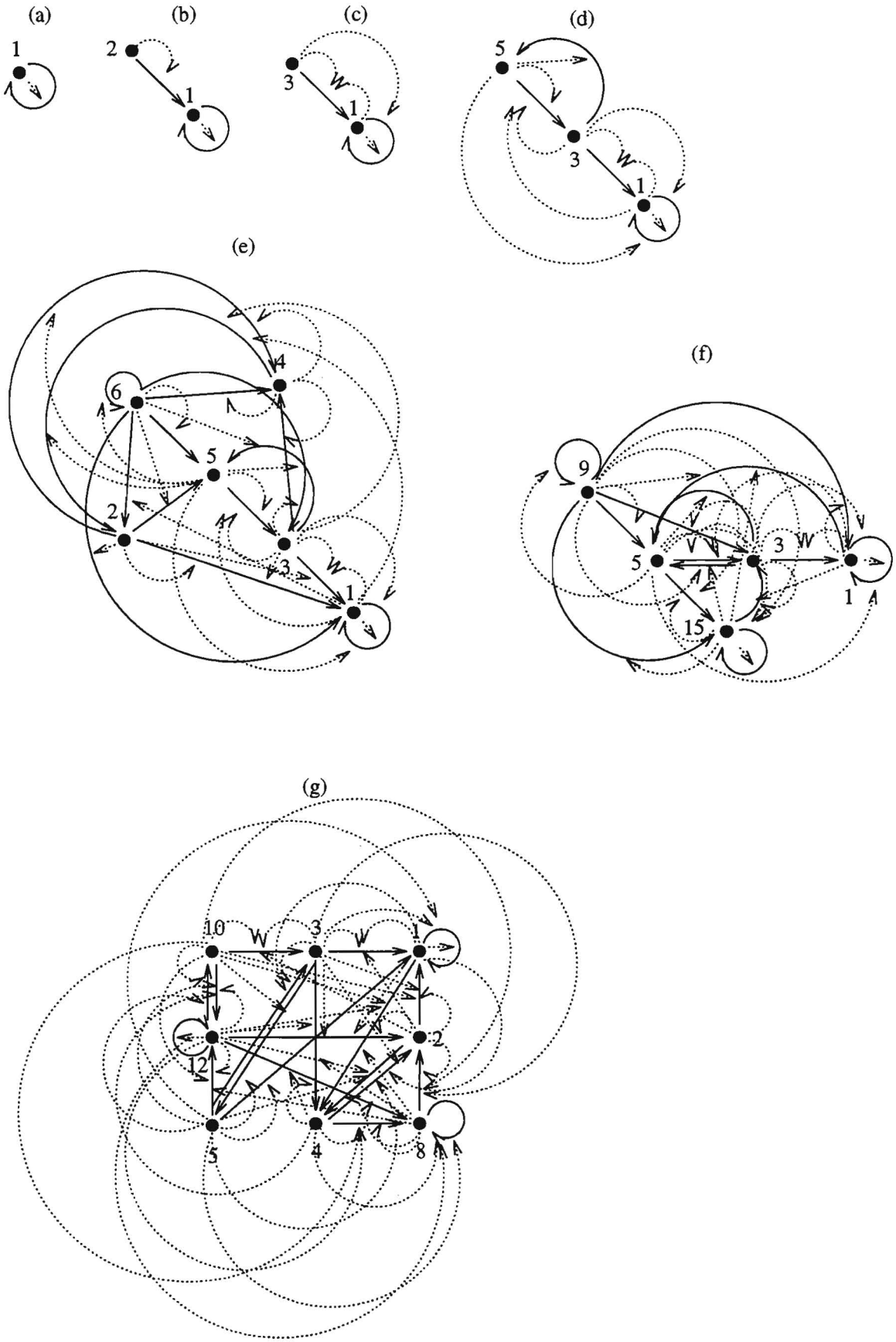


Figure 3: Interaction graphs of the system with $N = 4$. These graphs include all string sorts that are produced if the upper left string sort is used as the one and only initial sort in the reaction vessel. Solid lines connect the two string sorts participating in a reaction. Dashed lines indicate operator sort. (a) Self-replicator, also realized by sorts $s^{(7)}, s^{(12)}$ and $s^{(15)}$. (b) - (f): Simple and complicated parasitic interactions, (b) also realized by pairs $(s^{(11)}, s^{(15)}), (s^{(13)}, s^{(15)})$ and $(s^{(14)}, s^{(15)})$.

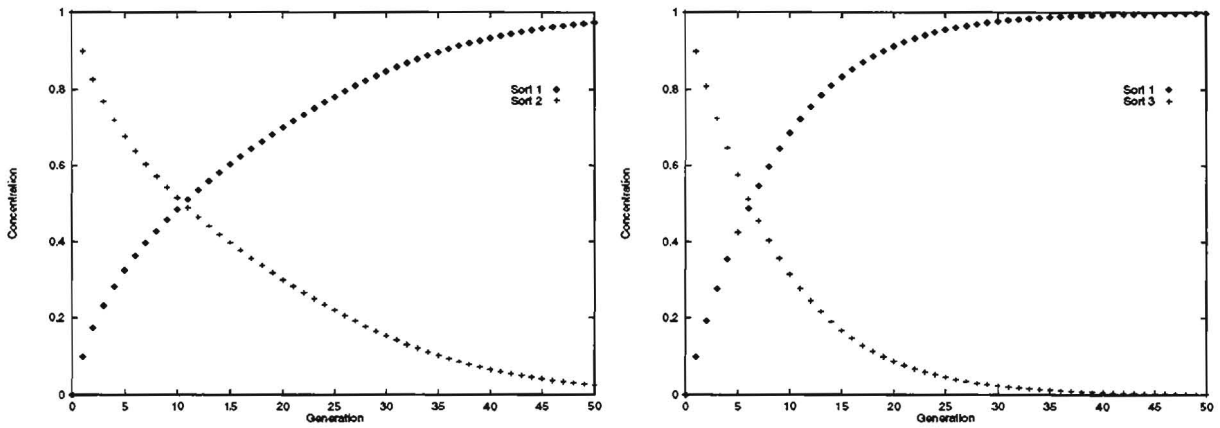


Figure 4: Dynamics of the interaction graph Figure 3(b) and (c).

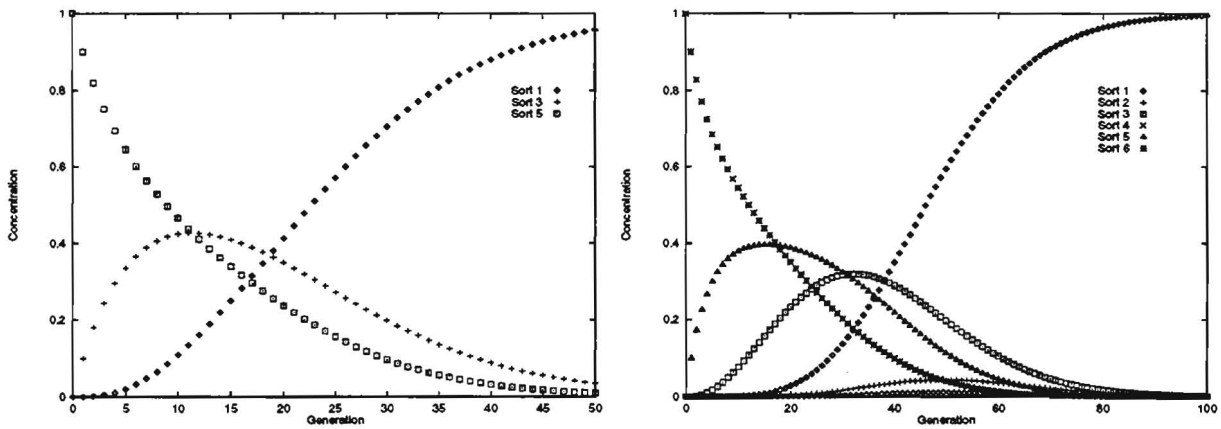


Figure 5: Dynamics of the interaction graph Figure 3(d) and (e).

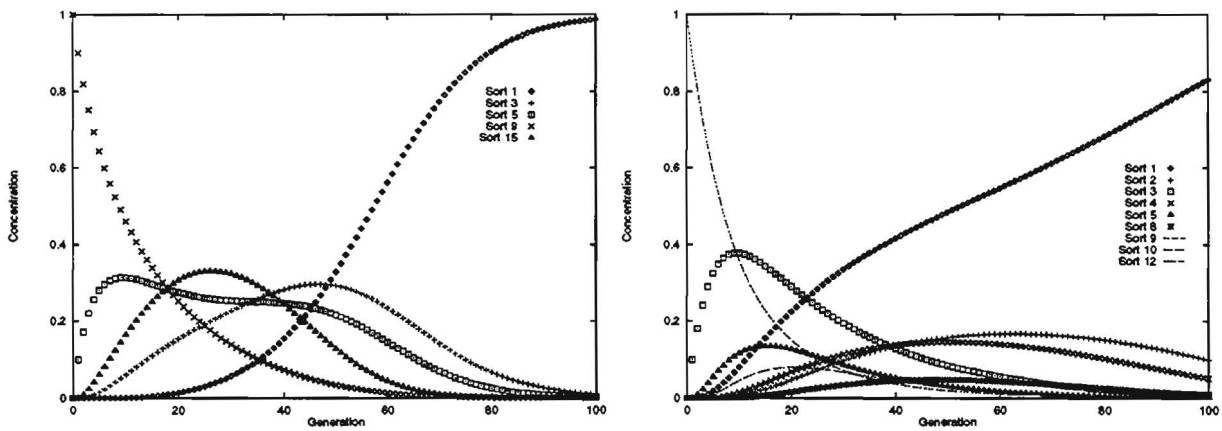


Figure 6: Dynamics of the interaction graph Figure 3(f) and (g).

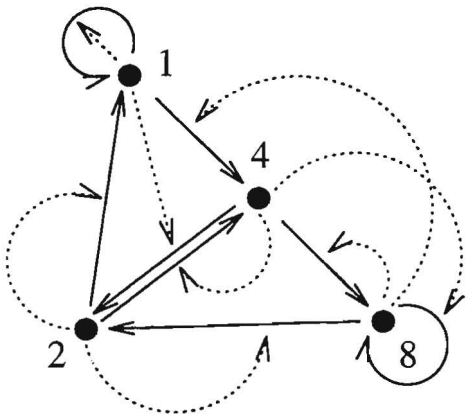


Figure 7: Reaction graph of the metabolism of $N = 4$.

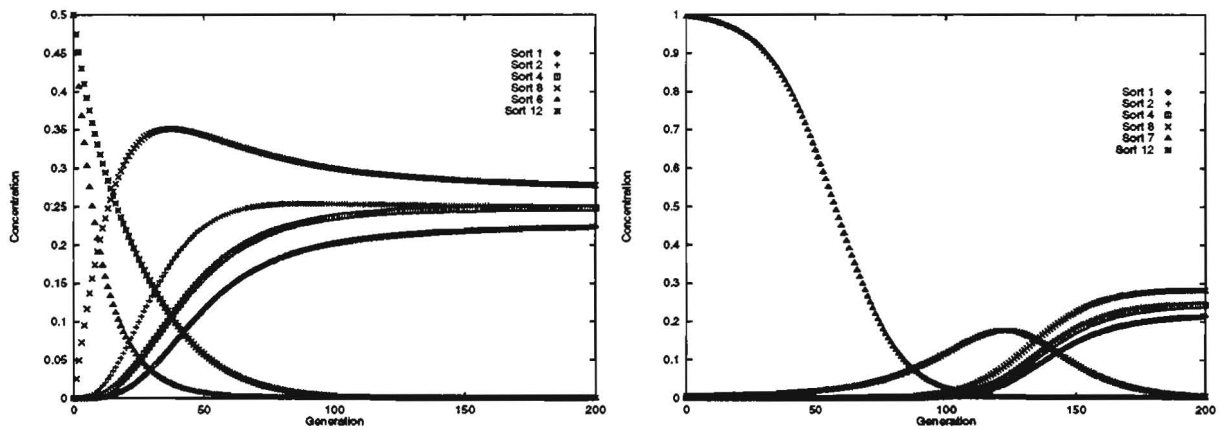


Figure 8: Dynamics of the interaction graph Figure 7. Left: Equal concentration of $s^{(6)}$ and $s^{(12)}$, at the outset; Right: High concentration of $s^{(7)}$, low concentration of $s^{(12)}$ at the outset.

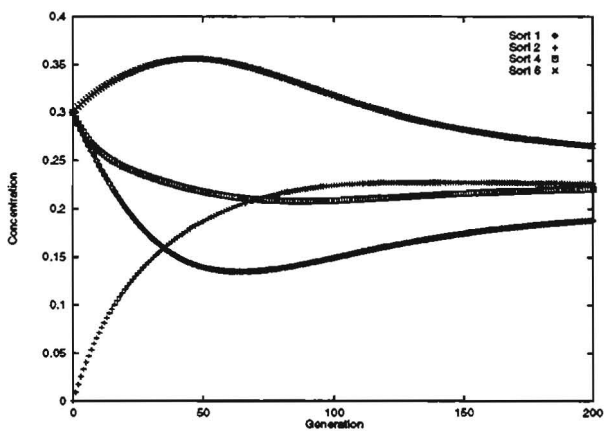


Figure 9: Dynamics of the interaction graph Figure 7. Starting with three different string sorts.

starting from an initial set $\mathcal{A} \subseteq \mathcal{N}_S$, for $n \rightarrow \infty$, with

$$r_0(\mathcal{A}) = \mathcal{A} \tag{20}$$

$$r_n(\mathcal{A}) = \cup_{i=0}^{n-1} r_i(\mathcal{A}) \circ r_{n-i-1}(\mathcal{A}) : \tag{21}$$

$$\mathcal{A}^* = \lim_{n \rightarrow \infty} R_n(\mathcal{A}). \tag{22}$$

Closed subsets are important organisational structures, especially in the light of the fact, that we can only populate part of sequence space, once the component number strings increases.

We should keep in mind, that we have dealt here with a system consisting of 4-bit components. The complexity of interactions in such a simple system as Figure 3 demonstrates, is astonishing. We expect the two basic behavioral classes, parasitic interaction and metabolism, to emerge in a variety of forms in systems with longer strings.

4 Evolution

As we have seen, the dynamics in this small system quickly settles into one of its attractor states. The question, however, arises, whether there is a perspective for evolution, that is, for a sequential exploration of possibilities. For evolution to happen, an occasional mutation of one string into another should lead to a cascade of newly produced string types, that lead to a new equilibrium. We have shown this to happen in a $N = 9$ system [7], and will adopt the results learned there.

We have been using a mutation as a motor for occasional change. A mutation hits each string with a probability depending on its size. We define q to be the probability that one element of a randomly selected string changes to another symbol, here "0" to "1" and vice versa. Since each element may be hit, this is a length dependent change and the probability that at least one error occurs in a string is $Q(1) = Nq$, with the provision that $q << \frac{1}{N}$. Evidently, this mutation probability depends linearly on the concentration of string sorts in the reaction vessel. That is to say, a more successful string sort will spawn more variations. Two-bit mutations are then $Q(2) = (Nq)^2$ where we neglect the fact that sometimes back-mutation may happen. In Nature, at least on instance of this type of mutations occurs in mutations caused by cosmic radiation.

Mutation does open up new transformation pathways between string sort, something Bagley et al. term a stochastic metadynamics [6].

Suppose we start our system by sort $s^{(7)}$. Since this is a self-replicating string sort, nothing interesting will happen, unless the mutation process introduces one of its nearest neighbors $s^{(3)}, s^{(5)}, s^{(6)}, s^{(15)}$. The reaction table shows, that the appearance of $s^{(6)}$ will have no consequence, whereas the appearance of $s^{(3)}, s^{(5)}, s^{(15)}$ allows the system to switch to another attractor. Figure 10, left, shows the effect of introducing $s^{(5)}$. As a result, the interaction graph of Figure 3 (d) comes into play, and $s^{(1)}$ dominates. Figure 10,right, is the evolution from selfreplicator $s^{(15)}$ to the metabolism consisting of $s^{(1)}, s^{(2)}, s^{(4)}, s^{(8)}$. This has been achieved by introducing $s^{(12)}$, a two-bit mutation from $s^{(15)}$, in spurious concentration.

5 Conclusion

We have examined a very simple self-organising system. The main idea was to introduce a second form of the information carriers of our system, the sequences of binary numbers. This has been accomplished by using an operative matrix form for the strings. We then have defined a particular interaction between matrices and strings and considered the interaction itself as some sort of a reaction with input and output. The low-level ("atomic") computations in the system have thus been likened to chemical reactions in the real world.

It has been shown that closed subsets of strings exist which can be considered as organisations. Under the assumption of one particular folding, these subsets of strings might be studied in their 2-dimensional matrix form alone, effectively yielding an interesting class of mathematical objects that are closed under the proposed non-linear interaction.

We also dealt with the dynamics of the competitive system naturally emerging, with reactions going on between different species of strings. As in other artificial systems [5, 6, 9, 10, 11] an attractor state was reached relatively quickly, beyond which nothing interesting happened any more. However, we already demonstrated powerful evolutionary effects brought about by the inclusion of a mutation or the potential of length changing interactions. Systems with longer strings will certainly possess different metabolic networks, and it is clear that the behavioral flexibility in such systems will be enormous.

1	2	3	4	5	6	7	8
1	(1,2)	(1,2,3)	(1,2,3,4)	(1,2,3,4,5)	(1,2,3,4,5,6)	(1,2,3,4,5,6,7)	(1,2,3,4,5,8,10,12)
(4)	1,3	(1,2,4)	(1,2,4,8)	(1,2,3,4,8)	(1,2,3,4,5,7)	1,3,5,7,11,13,15	1,3,5,7,9,11,13,15
7	(4,8)	1,3,5	1,3,5,7	(1,2,4,8,12)	(1,2,3,4,8,12)		
(8)	7,15	(4,8,12)	1,3,5,15	1,3,5,7,15			
15	(8,12)	7,11,15	7,11,13,15	1,3,5,9,15			
	13,15	13,14,15		7,11,13,14,15			
	14,15						

Table 4: Closed subsets of elements with up to 8 members. First column: Self-replicators. In parenthesis: Subsets which occasionally produce the destructor.

9	11	13	15
(1,2,3,4,5,8,10,12,15)	(1,2,3,4,5,6,8,9,10,12,15)	(1,2,3,4,5,7,8,10,11,12,13,14,15)	(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

Table 5: Closed subsets with more than 8 members. All subsets occasionally produce the destructor.

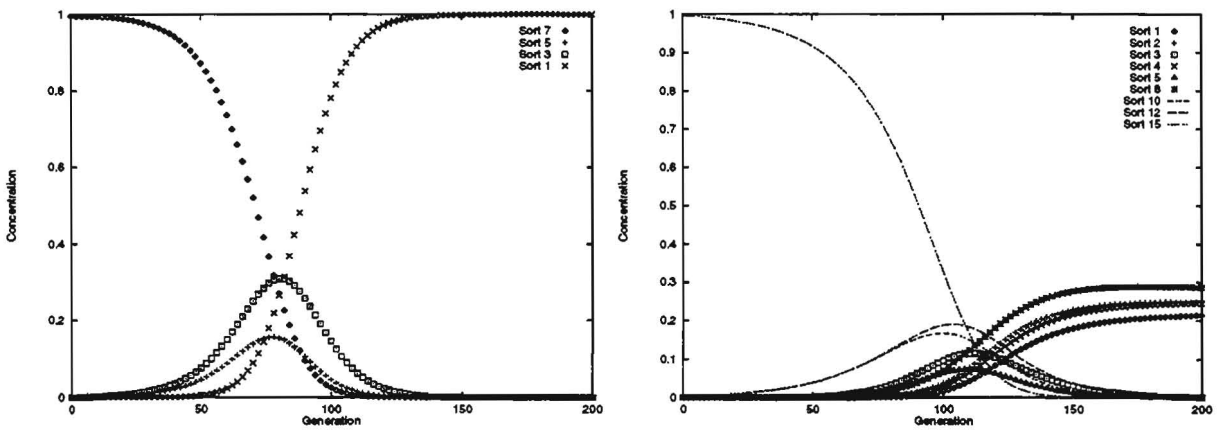


Figure 10: Evolutionary dynamics. Left: A 1-bit mutation causes $s^{(5)}$ to appear. This leads to $s^{(1)}$ as the dominant string sort after the transition. Right: A 2-bit mutation causes $s^{(12)}$ to appear. The result is the emergence of the metabolism of Figure 7.

Acknowledgement

I have enjoyed discussions with Dr. Walter Fontana. Mr. Helge Baier has provided me with the data of Table 4 and 5.

References

[1] W. Banzhaf, *Self-replicating sequences of binary numbers*, Computers and Mathematics, **26** (1993) 1

[2] M. Eigen, *Steps toward Life: a perspective on evolution*, Oxford University Press, 1992

[3] M. Eigen, P. Schuster, *The Hypercycle - A principle of natural self-organization, Part A-C*, Naturwissenschaften **64** (1977) 541 and **65** (1978) 7, 341

[4] W. Banzhaf, *Self-replicating sequences of binary numbers — Foundations I and II*, Biological Cybernetics, **69** (1993) 269 and 275

[5] R.J. Bagley, J.D. Farmer, *Spontaneous Emergence of a Metabolism*, in: C.G. Langton, C. Taylor, J.D.Farmer, S. Rasmussen (Eds.), *Artificial Life II.* Addison-Wesley, Redwood City, CA, 1991, 93

[6] R.J. Bagley, J.D. Farmer, W. Fontana, *Evolution of a Metabolism*, in: C.G. Langton, C. Taylor, J.D.Farmer, S. Rasmussen (Eds.), *Artificial Life II.* Addison-Wesley, Redwood City, CA, 1991, 141

[7] W. Banzhaf, *Self-replicating sequences of binary numbers — Foundations III*, to be published

[8] W. Fontana, L. Buss, *"The arrival of the Fittest": Toward a Theory of Biological Organization*, Bulletin of Mathematical Biology, submitted

[9] W. Fontana, *Algorithmic Chemistry*, in: C.G. Langton, C. Taylor, J.D.Farmer, S. Rasmussen (Eds.), *Artificial Life II.* Addison-Wesley, Redwood City, CA, 1991, 159

[10] J.D. Farmer, S.A. Kauffman, N.H. Packard, *Autocatalytic Replication of Polymers*, Physica **D22** (1986) 50

[11] S. Rasmussen, C. Knudsen, R. Feldberg, *Dynamics of Programmable Matter*, in: C.G. Langton, C. Taylor, J.D.Farmer, S. Rasmussen (Eds.), *Artificial Life II.* Addison-Wesley, Redwood City, CA, 1991, 211

Genetic Programming

Frank Klawonn
Department of Computer Science
University of Braunschweig
Bültenweg 74/75
D–38106 Braunschweig, Germany
Tel. (+49)(531)391-3293, Fax (+49)(531)391-5936, E-Mail klawonn@ibr.cs.tu-bs.de

Abstract

Evolutionary computation comprises problem solution techniques that are based on ideas inspired by the process of natural evolution. Genetic programming, one of the branches of evolutionary computation, provides a framework in which solutions to a problem are generated in the form of a strategy in a LISP–like notation, instead of a parameterized form, which is used by most of the algorithms in evolutionary computation. This paper gives a short introduction to the basic techniques in genetic programming and a brief overview of possible applications.

Keywords: Evolutionary computation, genetic programming

1 Introduction

The process of natural evolution provides the basic ideas for evolutionary computation techniques. Most of these techniques encode the possible solutions to a problem in the form of a real–valued vector, like in evolution strategies or sometimes in evolutionary programming, or in the form of a string of fixed length. Such codings are well suited for parameter optimization and for problems where solutions have a canonical representation in very restricted and fixed form.

However, for many complex problems the solution cannot be given in a parameterized form, but only in terms of a strategy or a computer program. In order to apply the ideas of evolutionary computation to the evolution of optimal strategies or programs, an appropriate formal framework has to be found. In [2, 6] modified genetic algorithms are applied in order to learn the rule base of a fuzzy controller in the form of if–then rules. Such a rule base can be interpreted as a simple program or strategy. However, these rules are still very restrictive and a more general framework for formulating solution strategies is needed.

J.R. Koza, who developed the genetic programming paradigm [7, 8], chose for various reasons LISP as the language in which the solutions have to be encoded. Of course, the genetic operators have to be adapted to the framework of LISP–programs as chromosomes.

The paper is organized as follows. After a short introduction to a LISP–like notation in section 2, we explain the basic methodology in genetic programming in section 3. The fourth section reviews briefly some application areas of genetic programming. Although the theoretical analysis of genetic programming is still in a very early state, we point out some approaches in this direction in section 5.

2 Coding of a Solution in a LISP–Like Style

What kind of programming language should be chosen for coding possible solutions? It should have a very simple syntax and substructures or subroutines must be easily identifiable. LISP (LISt Processing) offers these features.

Therefore, let us shortly review the basic constructs of LISP, slightly modified for our purposes. In LISP one distinguishes between symbols for functions and for terminals (constants or variables). Besides these symbols only brackets are needed. Let $\mathcal{F}$ and $\mathcal{T}$ denote the set of function symbols and terminals, respectively.

An S–expression or a list is defined as follows.

1. t is a list for any $t \in \mathcal{T}$.
2. If $f \in \mathcal{F}$ is a function symbol for a function of n arguments and $\ell_1, \ldots, \ell_n$ are lists, then $(f \ \ell_1 \ \ldots \ \ell_n)$ is also a list.
3. No other strings than those defined in 1. and 2. are lists.

We can associate to each list ℓ a function in the following canonical way.

1. If $\ell = t$ where $t \in \mathcal{T}$, then we identify ℓ with the function taking the constant value t, whenever t is a terminal standing for a constant. In the case that stands for a variable, then ℓ is the identity function (in that variable).
2. If ℓ is of the form $\ell = (f \ \ell_1 \ \ldots \ \ell_n)$ as described above and the list ℓ_i is associated with the function $g_i(x_1^{(i)}, \ldots, x_{k_i}^{(i)})$ where $i = 1, \ldots n$, then we identify ℓ with the function

$$f\left(g_1(x_1^{(1)}, \ldots, x_{k_1}^{(1)}), \ldots, g_n(x_1^{(n)}, \ldots, x_{k_n}^{(n)})\right).$$

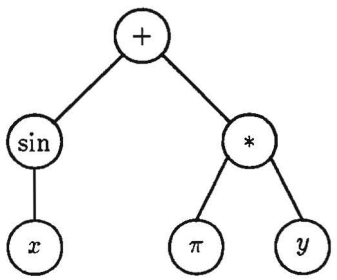


Figure 1: The tree representation of the list (1).

Example. Assume $\mathcal{F} = \{+, *, -, \sin\}$ and $\mathcal{T} = \{x, y, z, 0, 1, \pi\}$. Then the list

$$(+ \quad (\sin \quad x) \quad (* \quad \pi \quad y)) \qquad (1)$$

is associated with the function

$$f(x, y) \quad = \quad \sin(x) + \pi \cdot y.$$

Note that we understand the functions $+, *$, and $-$ as functions of exactly two arguments, i.e. we do not allow a list like $(+ \quad x \quad y \quad z)$. In order to express the function $f(x, y, z) = x + y + z$, we have to use either $(+ \quad x \quad (+ \quad y \quad z))$ or $(+ \quad (+ \quad x \quad y) \quad z)$. This restriction makes the definition of genetic operators more easy.

sin is of course a function of one argument.

It is not necessary that the functions in $\mathcal{F}$ operate on real numbers. Boolean functions, functions on strings, or on arbitrary abstract domains are also thinkable. However, what we require is that each function in $\mathcal{F}$ is defined for any possible input. If this is not the case like for example for the logarithm, one has to choose an arbitrary fixed value as the output for those values for which the function is not defined. It might be reasonable to enrich the set of terminals $\mathcal{T}$ with an extra constant *error*, which is chosen as output if a function is not defined somewhere.

The representation of a list as a string over the alphabet of function symbols, terminals, and brackets is very convenient for storing them in a computer. But for the way in which the list was defined as a composition of functions, trees seem to be more suitable. A list can be transformed into a tree in the following way. The nodes are the terminals and function symbols appearing in the list. A node representing a function symbol is connected to its 'argument' nodes. In this way, the terminals become the leaves of the tree. The tree corresponding to the list (1) is shown in figure 1.

The tree representation of a list is also appropriate for genetic programming, since in this coding genetic operators can be defined easily.

3 Description of Genetic Programming

The principal idea in the genetic programming paradigm is to evolve a population of LISP–programs or lists as they are introduced in the previous section in order to find a solution to a given problem. Based on a problem dependent fitness function that evaluates the members of the population with respect to their capabilities to solve the problem, selection can be defined in the usual manner. Even more than in genetic algorithms, the most important genetic operator for genetic programming is crossover. Of course, crossover has to be redefined for lists to guarantee that the recombination of two list yields again a list. The principle scheme of genetic programming is as follows.

1. Choose suitable sets $\mathcal{F}$ and $\mathcal{T}$ of functions and terminals.
2. Create a random initial population of lists.
3. Determine the fitness of the lists in the population. The fitness of a list should reflect how well its corresponding function solves the given problem.
4. Carry out selection based on fitness.
5. Apply crossover.

Repeat steps 3. – 5. until the termination criterion is satisfied.

The termination criterion is as usual in evolutionary computation defined on the basis of a maximal number of generations, desired quality of the best chromosome (list), no improvements of the fitness in the last generations, etc.

In the following, we describe more detailed the single steps to be carried out when solving a problem by genetic programming.

3.1 Function Set and Terminal Set

First of all, suitable finite sets of a functions, and terminals have to be determined. It is clear that at least the input variables for the problem (if there are any) have to be included in the set of terminals. But in addition usually some constants are also needed. If Boolean functions are considered then the only constants 0 and 1 can be added to the terminals. However, for real–valued functions it is impossible to include all real numbers in the set of terminals. In this case only a few 'representative' numbers should be considered as terminals. Other constants can be represented indirectly by iterated application of functions from the function set to those constants that belong to the terminals.

The closure property of the function set and the terminal set is very important in genetic programming. It requires that any terminal and any value or data type returned by any function is accepted by each function of the function set as its argument. This seems to be quite a restrictive postulate. But it can be easily met by extending partially defined functions simply by choosing an arbitrary fixed output for those arguments for which a function is not defined. For example, the division operation may yield the usual quotient unless the denominator is zero. Then we define for instance the zero as the result of the division. Another possibility is to introduce the value *undefined*, which then can also appear as an argument of function leading again to the result *undefined*. Various approaches to the solution of this problem are discussed in Chapter 6 of [7]. Another idea using types is proposed in section 3.6.

Besides the closure property, the choice of the functions and terminals should guarantee for sufficiency, i.e. the lists that can be constructed from the functions and terminals must be capable of solving the given problem, or at least include a satisfactory approximate solution. For instance, when one is looking for a Boolean function, it is well known that the function set {AND,OR} is not sufficient for constructing an arbitrary Boolean function, in opposition to the set {NAND}. Sufficiency is, of course, a must. But it is not required that the function set is minimal in the sense that without one of the functions the sufficiency property would be lost. Extraneous functions may lead to solutions that can be better interpreted by a human. For example, expressing a Boolean function only on the basis of NAND usually does not help to understand the character of the function. Nevertheless, it cannot be recommended to use too many extraneous functions and especially extraneous terminals, since it leads to larger search spaces.

3.2 The Fitness Function

The fitness of a list should indicate how well or how bad its corresponding function solves the given problem. The *raw fitness* of a list is usually defined on the basis of a number of test cases. The test cases should be a representative set of inputs. For each test case the desired result and the output given by the function associated with the list are compared and the (absolute value of the) error is calculated. The sum of these errors is the raw fitness of the list. For some problems there might be no fixed desired output. Instead of this, the solution is required to obtain as many score points as possible. In this case the sum over all score points collected in the test cases gives the raw fitness.

The *standardized fitness* is defined in such a way that the goal is the minimization of the standardized fitness which should only take non–negative values. If the raw fitness measures the errors then the raw fitness can be chosen as the standardized fitness. If the raw fitness counts the scoring points, the standardized fitness can be defined as the maximal number of obtainable scoring points minus the raw fitness.

Koza [7, 8] proposes to introduce the *adjusted fitness* which transforms the standardized fitness into the unit interval by

$$a(\ell) \quad = \quad \frac{1}{1+s(\ell)}$$

where $s(\ell)$ and $a(\ell)$ are the standardized and the normalized fitness values of the list (chromosome) ℓ.

Finally, the *normalized fitness* $n(\ell,t)$ of the list ℓ in the population of generation t, on which selection is based, is given by

$$n(\ell,t) \quad = \quad \frac{a(\ell)}{\sum_{\ell' \in P(t)} a(\ell')}$$

where $P(t)$ is the population of generation t.

3.3 The Initial Population

The initial population is created by producing a number of random lists. In principal, this is done by selecting randomly one of the elements of the set $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$. If a function $f \in \mathcal{F}$ of n arguments was chosen then again n elements of $\mathcal{C}$ have to be selected at random. The process is continued recursively, until finally only terminals were selected and thus the creation of a random list is completed.

This simple procedure can lead to lists whose corresponding trees have a high depth and might be very unbalanced. To avoid this effect, a maximal initial depth of the trees is specified (usually 6) and for each depth from two to the maximal initial depth equally many trees are generated randomly. Moreover, 50% of the trees should be full, i.e. the length of the path from the root to each leaf is the same for all leaves of the tree. Finally, duplicate lists should be avoided.

A typical population size is 500, however, for very complex problems a larger population size is recommended.

3.4 Selection

Usually roulette wheel selection based on the normalized fitness is carried out. When dealing with larger populations it might enhance the performance of genetic programming when fitter lists gain an additional bonus during selection.

3.5 Crossover

Crossover is the main genetic operator within the framework of genetic programming. The crossover operator has to be modified with respect to lists. Allowing arbitrary crossover points would in general lead to incorrect lists. The idea is that crossover should exchange 'sub-functions' of lists, meaning that in the tree representation subtrees are exchanged. Therefore, crossover is better explained on the level of trees than on the level of lists. When two lists are selected for crossover, for each of them a node in their tree representation is chosen randomly. In each list the chosen node marks a subtree. These subtrees are then exchanged by crossover.

Consider the list (1) with its corresponding tree shown in figure 1 and the list

$$(\sin \quad (+ \quad (\sin \quad (+ \quad x \quad y)) \quad z)) \qquad (2)$$

with its associated tree illustrated in figure 2.

Assume that these two list were selected for crossover and that for list (1) the $*$–node was chosen as crossover point whereas for list (2) the second (lower) sin–node was chosen. In figures 3 and 4 the respective nodes are marked by a dashed box and the induced subtrees are framed in a box.

After crossover the subtrees are exchanged resulting in the trees illustrated in figures 5 and 6 that correspond to the lists

$$(+ \quad (\sin \quad x) \quad (\sin \quad (+ \quad x \quad y)))$$

and

$$(\sin \quad (+ \quad (* \quad \pi \quad y) \quad z)),$$

respectively.

It is very important to note that in opposition to ordinary genetic algorithms, the application of the crossover

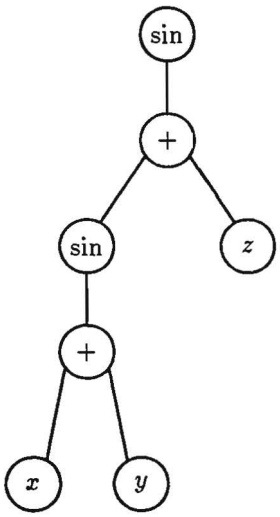


Figure 2: The tree representation of the list (2).

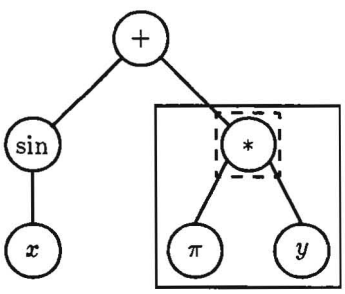


Figure 3: The crossover point and the corresponding subtree in list (1).

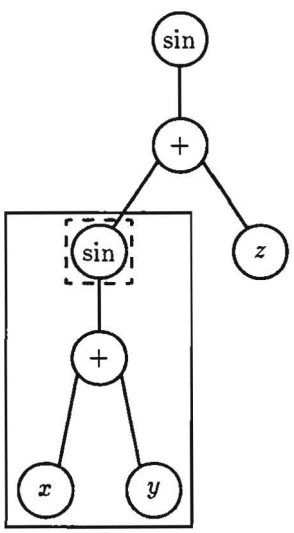


Figure 4: The crossover point and the corresponding subtree in list (2).

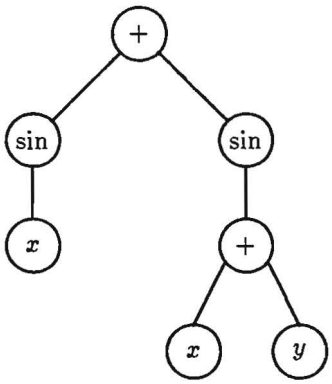


Figure 5: The first tree after crossover.

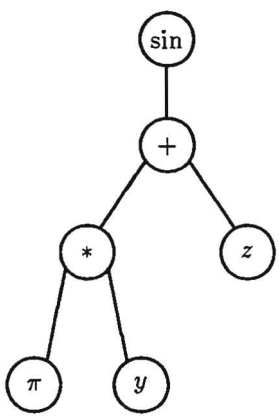


Figure 6: The second tree after crossover.

operator to two identical lists (trees), does here in general lead to two new lists that do both not coincide with their parents. Therefore, where in genetic algorithms the mutation operator is needed to maintain a certain variety in the gene pool, in genetic programming a rich variety is already produced by crossover alone. Therefore, crossover (together with selection) is considered as the primary operator in genetic programming, whereas mutation as well as other additional genetic operators are of minor importance. In the next subsection we shortly discuss some of these secondary operators.

Typically, a crossover rate of 90% is chosen. The crossover point is usually selected in such a way that the terminal nodes, i.e. the leaves, get only a 10% chance of being chosen. The remaining 90% of the crossover points are distributed equally among the internal nodes representing functions.

Crossover can create more and more complex lists. To reduce the size of the search space it is recommended to limit the depth of trees and to cancel a crossover operation when a tree is generated that exceeds the maximal allowed length of for instance 17 nodes.

3.6 Other Operators and Modifications

The definition of mutation for trees is straight forward. An arbitrary node is chosen randomly and the whole subtree below this node is replaced by a random tree.

Permutation is inspired by the inversion operator for genetic algorithms. A random internal (function) node is chosen and its arguments are permutated randomly.

More important than mutation and permutation is editing which simplifies lists on the basis predefined rules like

$$\text{Replace (NOT (NOT } x\text{)) by } x.$$

Editing leads to shorter solutions and helps also protecting good but clumsily stated solutions like (NOT (NOT (NOT (NOT x)))), coding simply the solution x, against destruction by crossover.

Another way to avoid the disruptive effects of crossover is the application of encapsulation which replaces a subtree by a single newly defined node which cannot be destroyed by crossover. Kinnear [4] discusses a similar concept called module acquisition which not only encapsulates a subtree or composed function, but also adds it to a module library which can be used by other lists by module acquisition.

We introduced types into genetic programming for applications in medical diagnosis. Here we have the problem of a large number of different measurement values of different types (real, integer, boolean, and even linguistic values). Moreover, it is very common to deal with missing values in medical diagnosis. To overcome these problems and to obtain solutions that are easier to interpret, we defined types for the arguments of the functions and admitted only those list in which each function had its correct types as arguments.

4 Some Application Areas

Genetic programming is a very flexible technique and can therefore be applied in many fields. In this section we mention some standard application areas of genetic programming. Large collections of examples are included in [7, 9] where also details can be found.

4.1 Symbolic Regression

A very common problem is that of identifying an unknown function which reflects some input–output behaviour on the basis of some given data. The idea of symbolic regression is to find a suitable function as a composition of some basic functions like polynomials or trigonometric functions. These functions and some standard operations like addition, multiplication etc. are then considered as the function set for genetic programming. The terminals should include the input variables and very few typical constants. The lists engendered by genetic programming are then interpreted as input–output functions. The fitness of such a list is defined on the basis of the sum of (squared) errors that is produced by the corresponding input–output function when it is applied to the given data set. Symbolic regression can also be used for solving differential or integral equations. The lists or the respective functions are interpreted as possible solutions of the equations. The fitness is again computed on the basis of the errors caused by the list as a solution for some typical points.

4.2 Optimal Control

Related to symbolic regression is the problem of optimal control where we are looking for a transfer function of a controller. In some cases there might be data available obtained from observing an operator. Then the control problem reduces to symbolic regression. If only the process itself or a simulation of it is available, fitness has to be defined on the basis of how well the transfer function defined by a list can cope with the process or the simulation.

4.3 Planning, Emergent Behaviour, and Game–Playing

In opposition to symbolic regression and optimal control, planning, emergent behaviour, and game–playing are usually based on strategies that are not described in terms of a real–valued input–output function. The strategies are usually more complex and thus a problem dependent function set has to be defined for genetic programming. Also the definition of fitness may vary strongly from problem to problem.

5 Theoretical Analysis of Genetic Programming

Although there are a lot of activities in evolutionary computation directed to theoretical analysis, for many algorithms such a theoretical analysis is missing completely or only a few investigations were made that give some hints about the power of the methods. Most results were of course obtained in the older fields of genetic algorithms and evolution strategies.

For the young discipline of genetic programming only a few attempts for a theoretical analysis were made now, which do not explain the behaviour of genetic programming satisfactory.

Koza [7] discussed the notion of schemata in genetic programming on an informal basis. He defines a schema as a subtree. Although his arguments seem to be intuitively appealing, he does not carry out any computation regarding schemata and does not even mention Mühlenbein'ss criticism [10] of wrong interpretations of the schema theorem.

Altenberg [1] examines the capability of evolvability for genetic programs, i.e. the ability of a population to produce variants fitter than any yet existing. He points out that the high recombination rates in genetic programming do usually not support the evolvabilty, although they might be necessary for finding good solutions quickly.

Most of the investigations on genetic programming are more experimental as for example in [4, 5] than based on a rigorous theoretical background. Until now one must rely on heuristics, when one wants apply genetic programming. Nevertheless, the successful applications show that genetic programming is a technique worth a closer examination.

References

[1] L. Altenberg, The Evolution of Evolvability in Genetic Programming. In: [3], 47–74.

[2] J. Hopf, F. Klawonn, Learning the Rule Base of a Fuzzy Controller by a Genetic Algorithm. In: R. Kruse, J. Gebhardt, R. Palm (eds.), Fuzzy Systems in Computer Science. Vieweg, Braunschweig (1994), 63–74.

[3] K.E. Kinnear (ed.), Advances in Genetic Programming. MIT Press, Cambridge, Massachusetts (1994).

[4] K.E. Kinnear, Alternatives in Automatic Function Definition: A Comparison of Performance. In: [3], 119–141.

[5] K.E. Kinnear, Fitness Landscapes and Difficulty in Genetic Programming. Proc. 1st IEEE Conference on Evolutionary Computation, Orlando (1994), 142–147.

[6] J. Kinzel, F. Klawonn, R. Kruse, Modifications of Genetic Algorithms for Designing and Optimizing Fuzzy Controllers. Proc. 1st IEEE Conference on Evolutionary Computation, Orlando (1994), 28–33.

[7] J.R. Koza, Genetic Programming: On the Programming of Computers by the Means of Natural Selection. MIT Press, Cambridge, Massachusetts (1992).

[8] J.R. Koza, The Genetic Programming Paradigm: Genetically Breeding Populations of Computer Programs to Solve Problems. In: [11], 203–352.

[9] J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, Massachusetts (1994).

[10] H. Mühlenbein, Evolution in Time and Space – The Parallel Genetic Algorithm. In: G. Rawlins (ed.), Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo (1991), 316–337.

[11] B. Souček and The IRIS Group (eds.), Dynamic, Genetic, and Chaotic Programming. Wiley, New York (1992).

Genetic Programming and Redundancy

Tobias Blickle* and **Lothar Thiele†**
Lehrstuhl für Mikroelektronik
Universität des Saarlandes
D-66041 Saarbrücken

Abstract

The *Genetic Programming optimization method (GP)* elaborated by John Koza [Koza, 1992] is a variant of Genetic Algorithms. The search space of the problem domain consists of computer programs represented as parse trees, and the crossover operator is realized by an exchange of subtrees. Empirical analyses show that large parts of those trees are never used or evaluated which means that these parts of the trees are irrelevant for the solution or redundant. This paper is concerned with the identification of the redundancy occuring in GP. It starts with a mathematical description of the behavior of GP and the conclusions drawn from that description among others explain the "size problem" which denotes the phenomenon that the average size of trees in the population grows with time.

1 Introduction

A growing range and variety of problems are solved using the Genetic Programming Paradigm. Especially in control systems the ability of GP to generate symbolic solutions makes it an interesting optimization tool for a plenty of problems. But to achieve good solutions in reasonable time a lot of parameters have to be adjusted, and often only heuristics can help to set these parameters.

It is well known that the representation of the possible solutions as parse trees allows the trees to grow with time, often without improving the current best solution ("size problem" or "bloating"). This problem is often addressed in connection with some kind of "schema theory" for GP: to overcome the decrease in efficiency caused by bloating, some mechanism is introduced to keep track of the frequency and saliency of subtrees (named "schema") [Tackett, 1994; Rosca and Ballard, 1994]. These informations are used to select "better" subtrees for crossover to achieve a better convergence.

Another method to avoid "oversized" solutions is to add some penalty to the fitness function for too big trees:

$$fit(i) = \alpha\,\text{rawfit}(i) + \beta\,\text{sizeof}(i)$$

If the fitness is to be minimized, larger trees are "punished", because they get a higher fitness than smaller ones. The adjustment of the parameters α and β is the main problem of this approach because they depend heavily on the problem and the rawfit function. Examples can be found in [Koza, 1992; Kinneth E. Kinnear, 1993]. Conor Ryan recently suggested a special case of multi-modal fitness functions named "Pygmies and Civil Servants" [Ryan, 1994] where he uses two criteria to determine the fitness: one list keeps track of the best rawfit individuals, in a second list the sorting criterion is the size of the individuals.

In this paper both the phenomenon of bloating and the unsatisfying convergence of the search process are explained by means of the redundancy in the trees. An exact analysis of these phenomena requires a mathematical description of the evolutionary process during a generation. This description is given in the next section where also a simple theorem for convergence is derived. The subsequent section elucidates some effects of redundancy and explains the "size problem". Section 4 suggests a control mechanism to reduce redundancy. A discussion of the results and a comparison with other approaches follows in Section 5.

2 Generational Behavior of GP

To describe the behavior of trees during a GP run one needs an analysis of both the reproduction and the crossover phase. In the following we assume that reproduction and crossover are done sequentially: first a reproduction phase creates an intermediate population and crossover is then performed on the fraction p_c of this intermediate population to get the population for the next generation. This kind of description differs form that of the Genetic Programming Paradigm in [Koza, 1992] but is mathematically equivalent.

2.1 Reproduction Phase

Definition 2.1 *The* reproduction rate $s'(f)$ *denotes the expected number of individuals with fitness value f after the reproduction phase. $s(f)$ denotes the number of individuals with fitness value f before reproduction.*

The reproduction rate s' depends on the fitness value, the fitness distribution of the population, and the reproduction method chosen. Using this definition a classi-

*blickle@ee.uni-sb.de
†thiele@ee.uni-sb.de

fication of the reproduction methods is possible. The ratio $\frac{s'(f)}{s(f)}$ gives the expected number of copies of one individual with fitness f. A reasonable selection method should of course favour good individuals by assigning them a ratio $\frac{s'(f)}{s(f)} > 1$ and punish bad individuals by a ratio $\frac{s'(f)}{s(f)} < 1$.

Well known is for example the reproduction rate for fitness proportionate selection $s'_{fp}(f) = s(f)\frac{f}{\frac{1}{M}\sum_{T\in P} fit(T)}$ where M is the size of the population P and $fit(T)$ represents the fitness value of the individual T (see e.g. [Goldberg, 1989]).

We concentrated our work on tournament selection [Goldberg and Deb, 1990], that works as follows: Randomly choose a certain number t of individuals (with or without replacement) from the population and take only the best individual for the next population. This process is repeated as often until the desired population size is reached. The tournament size t strongly affects the behaviour of the reproduction method.

The expected number of best individuals for tournament selection (with replacement) can be exactly derived (see [Rheinert, 1994]).

Theorem 2.1 *Let M be the population size, $s(f_b)$ the number of current best individuals with fitness f_b. Then the expected number of best individuals after performing tournament reproduction with tournament size t is*

$$s'(f_b) = M\left(1 - \left(1 - \frac{s(f_b)}{M}\right)^t\right) \tag{1}$$

Proof: For each tournament t individuals are randomly chosen from the population. The probability that no best individual participates in such a tournament is given by $\left(1 - \frac{s(f_b)}{M}\right)^t$, as $1 - \frac{s(f_b)}{M}$ is the probability to select a non-best individual. The probability that at least one best individual is in a tournament is then $1 - \left(1 - \frac{s(f_b)}{M}\right)^t$. In this case the best-fit individual will win the tournament. As exactly M tournaments are held the expected number of best individuals is given by Equation 1. □

Theorem 2.2 *For a small number $s(f_b) \ll M$ of best individuals with fitness f_b the reproduction rate of tournament with tournament size t is given by*

$$s'(f_b) \approx t\, s(f_b) \tag{2}$$

Proof:

$$\begin{aligned} s'(f_b, P) &= M\left(1 - \left(1 - \frac{s(f_b)}{M}\right)^t\right) \\ &\approx M\, t\frac{s(f_b)}{M} = t\, s(f_b) \end{aligned}$$

□

The choice of the tournament size depends on the population size and the current number of individuals with best fitness. If the tournament size is too high there will be too many best individuals and the GP will perform badly and get stuck at a local optimum.

2.2 Crossover Phase and Redundancy

Definition 2.2 *The edge A in tree T is called* redundant *if for all values of the leaves (terminals) the function represented by tree T is independent of the subtree located at edge A.*

Note:

- If the edge A is redundant it follows immediately that all edges in the subtree located at edge A are redundant, too.
- The redundancy of en edge A in general depends on the context.
- All nodes located at redundant edges are *redundant nodes.*
- The non-redundant nodes are also called "atomic" [Tackett, 1994].

Definition 2.3 *The* proportion of redundant edges *in a tree T is given by*

$$p_r(T) = \frac{number\ of\ redundant\ edges\ in\ T}{number\ of\ all\ edges\ in\ T} \tag{3}$$

Definition 2.4 *The* redundancy class T^* *is the set of all trees T that only differ from subtrees at redundant edges, i.e. for any two trees $T_1, T_2 \in T^*$, T_1 can be transformed into T_2 only by changing subtrees at redundant edges of T_1.*

A tree can only belong to one redundancy class, and all members of the class have the same fitness value denoted by $fit(T^*)$.

Now it is easy to compute the probability $p_s(T)$ that a tree T of the redundancy class T^* survives crossover, i.e. it remains in the class T^*.

Theorem 2.3 *Let p_c be the probability of crossover and T a tree of the redundancy class T^*. The probability of tree T to remain in class T^* after crossover is given by*

$$p_s(T) \geq 1 - p_c + p_c p_r(T) \tag{4}$$

Proof: With probability $1 - p_c$ the tree does not participate in crossover and with probability $p_c p_r(T)$ the crossover is performed at a redundant edge. In both cases the tree T remains in the class T^*. The "$\geq$" sign takes into account that a tree also survive if crossover is performed at a non-redundant edge when a "matching" subtree is inserted. But the probability of that event to occure is very small. □

The average probability of an arbitrary member of the redundancy class T^* to survive crossover is

$$p_s(T^*) \geq 1 - p_c + p_c p_r(T^*) \tag{5}$$

with $p_r(T^*) = \frac{\sum_{T\in T^*} p_r(T)}{|T^*|}$ the average proportion of redundant edges in the trees of class T^*.

It follows from Equation 4 that within a class the more redundant trees are more likely to survive. On the other hand it is very difficult to predict whether the redundancy of a tree will increase or decrease once a redundant crossover site has been chosen as this depends on the size of the new subtree being inserted. But this size is independent of the redundancy, so in average the selection mechanism will dominate and the redundancy within a class is expected to increase.

Theorem 2.4 *Let $p_r(T^*)$ be the average redundancy of the redundancy class T^* before reproduction. The average redundancy $p_r(T'^*)$ after reproduction is in average independent of the reproduction method and $p_r(T'^*) = p_r(T^*)$.*

Proof: As the reproduction of any tree of the class T^* is independent of the redundancy and each tree has the equal reproduction rate $\frac{s'(fit(T^*))}{s(fit(T^*))}$ no tree is given preference to. Therefore, the redundancy will in average remain constant after reproduction. □

As all statistical considerations in this paper Theorem 2.4 is only valid in average and the effective redundancy during a GP run may significantly differ.

2.3 Generational Behavior

Now we are able to describe the behavior of the Genetic Program over a whole generation.

Theorem 2.5 *(Generational Behavior) Let f be the fitness value of the trees of the redundancy class T^* with $|T^*|$ members and $s(f)$ the total number of individuals with fitness value f. The expected number of members of T^* in the next generation is given by*

$$|T'^*| \geq |T^*| \frac{s'(f)}{s(f)} (1 - p_c + p_c p_r(T^*)) \tag{6}$$

Proof: $|T^*| \frac{s'(f)}{s(f)}$ gives the expected number of members of the redundancy class T^* after reproduction and before crossover. Using this and combining 5 with Definition 2.1 Equation 6 is obtained. Actual $p_r(T^*)$ denotes the average redundancy in the class T^* before reproduction but it is used in Equation 6 as average redundancy after reproduction. But in average these two values are equal, as shown by Theorem 2.4. □

Theorem 2.5 shows that the redundancy classes with high redundancy will be given preference to. Conclusions from that will be drawn in Section 3.

In the following two different trees are compared.

Theorem 2.6 *Let T_1^* and T_2^* be two redundancy classes with the same fitness value f but a different redundancy ($p_r(T_2^*) = p_r(T_1^*) + \alpha, \alpha > 0$). The ratio r of trees of class T_1^* to trees of class T_2^* after crossover is given by*

$$r \approx \frac{|T_1^*|}{|T_2^*|} \frac{p_s(T_1^*)}{p_s(T_1^*) + p_c \alpha} \tag{7}$$

Proof:

$$\begin{aligned} r &\approx \frac{|T_1^*| \frac{s'(f)}{s(f)} p_s(T_1^*)}{|T_2^*| \frac{s'(f)}{s(f)} p_s(T_2^*)} \\ &= \frac{|T_1^*| p_s(T_1^*)}{|T_2^*| p_s(T_2^*)} \\ &= \frac{|T_1^*|}{|T_2^*|} \frac{p_s(T_1^*)}{p_s(T_1^*) + p_c \alpha} \end{aligned}$$

□

If we use Theorem 2.6 in the special case of only one individual in each redundancy class (i.e. we look at the moment of appearance of these trees) it is possible to trace the ratio r for k generations :

$$r(k) \approx \left(\frac{p_s(T_1^*)}{p_s(T_1^*) + p_c \alpha} \right)^k \tag{8}$$

It is assumed that the proportion of redundant edges in average remains constant for the trees of the same class T^* over the generations. An exact reflection must consider that this proportion varies (increases) with time, e.g. $p_s(T_1^*)$ and α are a function of k. But the basic statement of Equation 8 remains valid: the ratio will approximately rise exponentially in k.

2.4 Convergence

"Good convergence" is one of the most desirable properties of an optimization method. To achieve this, it is demanded that the current best individuals in a population shall be in the generation at least once. By this the current best solution will never be lost.

Theorem 2.7 *(Convergence Theorem) Let T_b^* be a redundancy class with current best fitness f_b and redundancy $p_r(T_b^*)$, $s(f_b)$ the total number of individuals with fitness f_b and $s'(f_b)$ the reproduction rate. At least one member of the class T_b^* will in average be contained in the next generation if the following inequality holds:*

$$|T_b^*| \frac{s'(f_b)}{s(f_b)} (1 - p_c + p_c p_r(T_b^*)) \geq 1 \tag{9}$$

Proof: This is directly obtained from Theorem 2.5. □

If this statement is specialized for the case that only one best individual shall survive, regardless from which redundancy class, the following inequality is obtained

$$\frac{s'(f_b)}{s(f_b)} (1 - p_c + p_c \overline{p_r(T_b)}) \geq 1 \tag{10}$$

with $\overline{p_r(T_b)} = \frac{1}{s(f_b)} \sum_{T_i | fit(T_i) = f_b} p_r(T_i)$ the average redundancy of all trees with fitness f_b.

Limiting oneself to a further specialization of Equation 10 with $\overline{p_r(T_b)} = 0$, i.e. the current best trees are free of redundancy, the condition that at least one best-fit individual survives is tightened to

$$\frac{s'(f_b)}{s(f_b)} (1 - p_c) \geq 1 \tag{11}$$

This inequality gives a lower bound on the adjustment of the reproduction rate. Using tournament reproduction the tournament size should be $t = \frac{s'(f_b)}{s(f_b)} \geq \frac{1}{1 - p_c}$.

The parameters of a GP run ($\frac{s(f_b)}{s(f_b)}$ and p_c) should be adjusted according to this theorem to achieve good convergence.

3 Effect of Redundancy

In the previous section a mathematical description of the Genetic Programming was carried out by introducing the term of redundancy. Now some effects of this theory are derived and experimentally verified. The 6-multiplexer problem from [Koza, 1992] served as test problem. The

6-multiplexer problem was chosen because it allows to determine the total redundancy in the trees. To achieve this sequentially in each edge of every tree a boolean NOT was inserted. If the insertion does not change the function table of the tree this edge is redundant according to Definition 2.2. This is a time consuming mechanism but an easy way to determine the total redundancy.

In the experiments the population size was $M = 200$, and the probability of crossover $p_c = 0.9$. The reproduction method was tournament, and the maximum tree size was limited to 50 nodes. All data given are the average over 100 runs. Note that the fitness measure is the number of errors in the function table of a tree, i.e. the optimal individual has a fitness value of 0.

According to Theorem 2.5 the redundancy in the trees of a certain (not only the best) fitness value will increase over time, because the trees are treated equal during reproduction phase but more redundant trees have a higher chance to survive crossover. This expectation is verified by Figure 1: in this single run, the GP was stuck at a local optima from generation 4 to 15 and an increasing redundancy is observed for all fitness values.

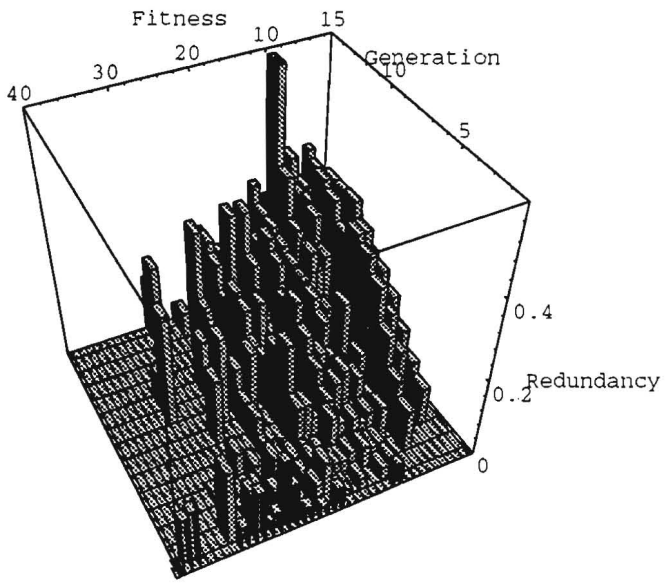


Figure 1: The redundancy increases with time (6-multiplexer problem, tournament size $t = 10$).

Another way to observe the increasing redundancy is to continue a GP run even if a best solution is found. [Koza, 1992] reported the phenomenon that after some generations there are almost 70-80% best-fit individuals, but almost all best trees are different. This can be explained straightforward with redundancy: the trees are all different but belong to only a small number of different redundancy classes. A very high redundancy makes the trees robust against crossover. Figure 2 shows the course of redundancy versus time.

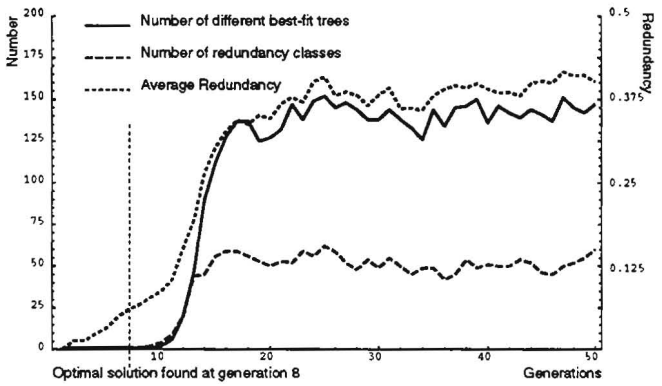


Figure 2: Redundancy versus time after occuring of a best-fit individual

For a given fitness value this increasing redundancy can only be achieved by increasing the tree size, because some "minimal size" in the atomic part of the tree is necessary to reach this fitness. By this the tree size will grow with time and the "size problem" can be explained with redundancy. Figure 5 shows that the average tree size for a given fitness value is smaller, if a higher reproduction rate is chosen. A comparison with the results in [Tackett, 1994] is given in the next section. So if the redundancy can be reduced the average tree size will be reduced at the same time.

One way to achieve a reduction of redundancy is offered by Theorem 2.5: If the reproduction rate s' is too low Theorem 2.5 implies that non-redundant trees have a too small chance to survive and redundancy-free trees will often vanish. Therefore a higher reproduction rate should lead to lower redundancy. The results of the corresponding experiments with tournament reproduction are shown in Figure 3 for a tournament size t of 2, 5 and 10. It can be seen that the redundancy decreases with increasing selection pressure, e.g. the redundancy at generation 50 for tournament size 2 is almost twice as high as the redundancy using tournament size 10. As the convergence time is very different for various selection pressure the relation between the average redundancy in a population versus the average population fitness gives a more meaningful description. In Figure 4 it can be seen that the higher the selection pressure the lower the redundancy. Note that a fitness value of zero is optimal.

The increasing redundancy makes it less likely to choose a non-redundant edge as crossover site and thereby hinders the evolution of new individuals. It follows from this consideration that the probability to escape a potential local optimum decreases with time.

If two partial solutions with the same fitness score but different redundancy occur at the same time, the less redundant might be lost after few generations (Equation 8). If the GP is not able to combine the two partial solutions during a limited time the loss of important genetic material (i.e. one partial solution) is probable.

Looking at the redundancy from an individual's point of view it is advantageous to maximize the redundancy, because it increases the probability of the individual to

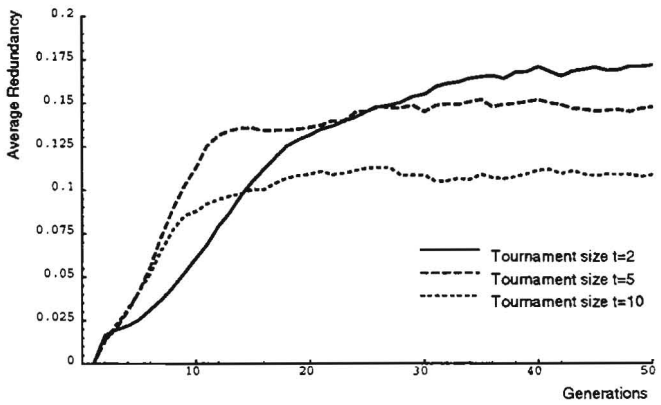


Figure 3: Increasing selection pressure leads to decreasing redundancy

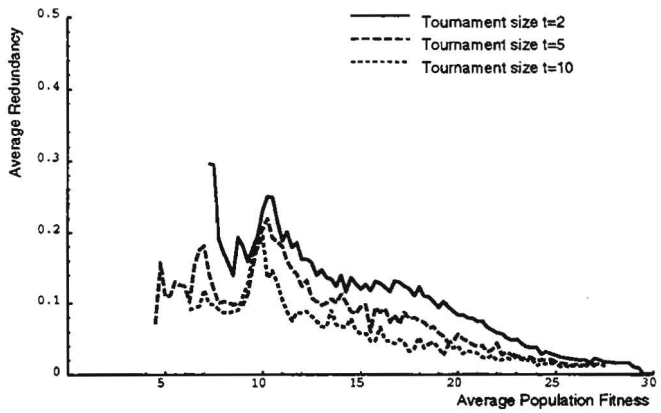


Figure 4: Average population fitness versus redundancy with tournament reproduction

survive. From the viewpoint of GP as an optimization method, the redundancy is unfavorable because it aggravates the optimization process.

These considerations show the importance of redundancy for the performance of the Genetic Programming optimization method. One suggestion to control the redundancy is given in the next section.

4 Control of Redundancy by Marking

The idea of marking is to avoid crossover at redundant edges by marking all edges that are traversed during evaluation of the fitness function: First the marking flags of all nodes are cleared and if a node is evaluated during the fitness calculation the corresponding flag is set. After calculating the fitness function, only at redundant nodes the flags are still cleared. The crossover is then restricted to edges with the flag set, i.e. non-redundant edges. By this useless crossover sites are avoided.

The additional time consumption caused by setting the marking flag is very low and the additional memory demands is one bit per node for the marking flag. The method implies that some subtrees are not evalu-

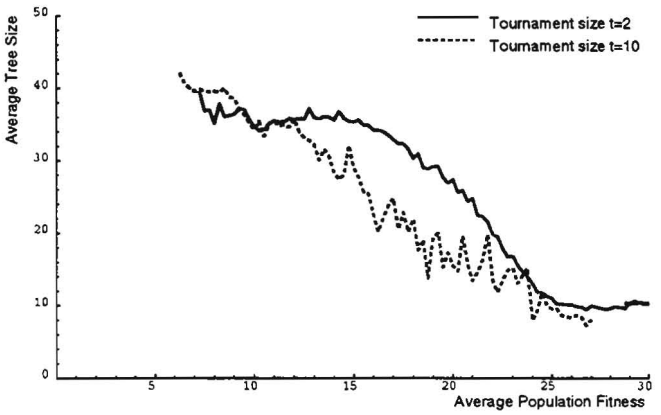


Figure 5: Higher selection pressure leads to smaller tree sizes for the same fitness value ($maxtreesize = 50$)

ated if some "run-time" condition is met. That can only be achieved by a suited implementation of the GP. Also the "total" redundancy can not be discovered, because it depends on the ordering of the evaluation of the subtrees. Consider for example the boolean **AND** function and the tree **AND(FALSE,SUBTREE)** where **FALSE** is a tree that always returns the value false and **SUBTREE** is an arbitrary tree . If the implementation evaluates the first subtree at first, the second subtree **SUBTREE** is not traversed because the result will be false anyway. By this the **SUBTREE** will be discovered as redundant. But if the implementation evaluates the second argument at first, in some cases **SUBTREE** will be true in some cases false. So both subtrees will be evaluated and the redundancy will not be discovered.

The marking method also depends strongly on the function set used to solve the optimization problem.

Experimentally we investigated the effect of the marking method for three problems taken from [Koza, 1992].

For the 6-multiplexer problem the performance was almost doubled, as can be seen in Figure 6. The parameter setting was the same as in the experiment described above with a tournament size of 10. This problem is well suited for the marking method because the usage of the function set **IF AND OR NOT** allows an easy discovering of the redundant edges.

Using the marking method for the *truck backer upper* problem an improvement in convergence of 20 % was measured. The third example was the *artificial ant* problem, where the almost no improvement occurred.

5 Discussion

In Section 3 we explained the size problem using redundancy. In [Tackett, 1994] Walter Tackett states that the average growth in size is proportional to selection pressure, and he gives an example of a deceptive problem solved with several selection methods to verify his statement. We think that it is better to analyze the dependency between the average population fitness and the average tree size, rather than between the time (number of generations) and the average tree size as done by

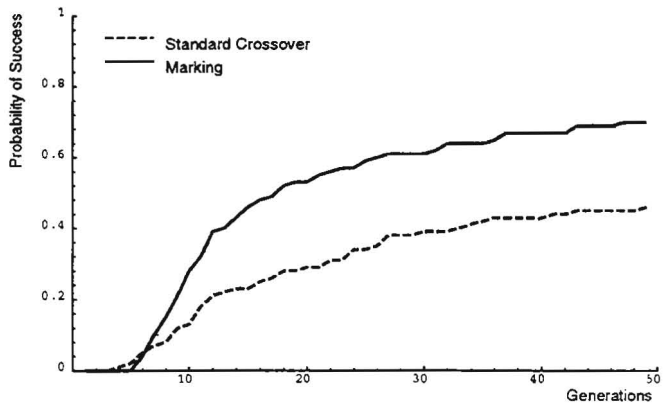


Figure 6: Improving performance of the 6-multiplexer problem using marking

Tackett. As a higher selection pressure mostly leads to higher fit individuals at an earlier generation and higher fit individuals mostly are bigger it is natural that the average tree size will grow faster under higher selection pressure. But the relation between average fitness and size has a higher meaningfulness. Figure 5 shows that under higher selection pressure (tournament size 10) the average tree size for given average fitness value is smaller than under lower selection pressure (tournament size 2). Analyzing the tree size over time yields the same results that Tackett has reported, but we do not think that this correlation gives a valuation of a good parameter setting.

Another observation reported by Tackett is that no "bloating" at all occurs if there is no selection. This can easily be deduced from the theory of redundancy: without selection *all* edges in all trees are redundant (according Def. 2.2). From $p_r(T) = 1$ follows $p_s(T) = 1$ for all trees, i.e. every tree will "survive" crossover.

6 Conclusions

In this paper the main emphasis was put on the mathematical description of the behavior of the Genetic Programming optimization method by introducing the term of redundancy. The theory showed that high fit and high redundant trees will spread exponentially in the population. The conclusions drawn gave an explanation of the "size problem" and showed a dependency between selection pressure and redundancy.

Further the new crossover operator "marking" was introduced that can improve the convergence by avoiding redundant crossover sites. Though this method may not be suited for many problems it demonstrates the influence of redundancy on the evolvability of a program.

References

[Goldberg and Deb, 1990] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. Technical Report 90007, The Clearinghouse for Genetic Algorithms, Department of Engineering Mechanics, The University of Alabama, Tuscaloosa, AL 35487–2908, July 1990.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[Kinneth E. Kinnear, 1993] Jr. Kinneth E. Kinnear. Generality and difficulty in genetic programming: Evolving a sort. In Stefanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 287–294, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[Koza, 1992] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[Rheinert, 1994] Pascal R. Rheinert. Die Genetische Programmierung. Master's thesis, Lehrstuhl für Mikroelektronik, Universität des Saarlandes, D-66041 Saarbrücken, 1994.

[Rosca and Ballard, 1994] Justinian P. Rosca and Dana H. Ballard. Genetic programming with adaptive representations. Technical Report 489, The University of Rochester, Computer Science Department, Rochester, New York 14267, 1994.

[Ryan, 1994] Conor Ryan. Pygmies and civil servants. In Jr. Kinneth E. Kinnear, editor, *Advances in Genetic Programming.* The MIT Press, Cambridge, Massachusetts, 1994.

[Tackett, 1994] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs.* PhD thesis, Faculty of the Graduate School, University of Southern California, 1994.

Racial Harmony in Genetic Algorithms

Conor Ryan*

University College Cork, Ireland

1 Introduction

This paper extends the work orginally presented in (Ryan,1994) which described a new selection scheme, the Pygmy Algorithm, designed for multi-objective problems. The extensions not only improve the performance of the Pygmy Algorithm, but show that, in certain cases, the mating of phenotypically similar parents can benefit evolution. The extended Pygmy Algorithm also demonstrates the power of evolving the parameters that control a population in parallel with the evolution of the population. Finally, the paper also examines the issue of taking parallels not only from nature, but also from sociological phenomenon.

2 Background - Pygmies and Civil Servants

The Pygmy Algorithm was orginally applied to the problem of evolving minimal sorting networks, which must not only be able to sort numbers, but also in as few steps as possible. The Pygmy Algorithm employs two fitness functions and maintains two lists of parents, one for each criterion of the problem. Traditional methods which use a single fitness function and try to balance the reward for each criterion force the implementor to choose how important each criteria is and run the risk of individuals trading off parts of the fitness function for each other. Neither of these problems arise with the Pygmy Algorithm.

One of the lists contains very efficient individuals, known as *Civil Servants*, while the other contains very short individuals, known as *Pygmies*. When selecting parents for a new individual, the Pygmies and Civil Servants were treated as though they were morphologically different, one parent being drawn from each list. As a population evolved, it was found that each list contributed pressure to direct the evolution in its particular direction, i.e. Civil Servants ensured that individuals were efficient at sorting, while Pygmies exerted pressure towards individuals being short. It was found that, when using the Pygmy Algorithm for selection, individuals could not trade off various parts of the overall fitness function for other, less important parts, and that populations using the Pygmy Algorithm were more likely to converge on an optimal solution than those that use a fitness function which is a sum of the two criteria.

To aid comparison between the work presented here and previous work, the same problems will be examined in this paper.

* This work is supported in part by Memorex Telex Ireland Limited. Thanks to Gordon Oulsnam for suggestions on this paper.

3 Gender or Race?

Due to the fact that all the individuals were looked upon as belonging to one of two genders, inbreeding between individuals with similar performance was prevented. It was still possible, however, for individuals to breed with close relatives - siblings perhaps, or even a parent of the opposite sex. In the Pygmy Algorithm, such incestuous behaviour did not appear to cause problems because of the diversity of parents maintained in each of the two lists.

The nature of multi-objective problems is such that individuals must solve two or more smaller problems in order to solve the main problem. However, despite the fact that the Pygmy Algorithm maintained groups of individuals who were good at each of the sub-problems, neither group explicitly attempted to produce individuals who excelled at its own sub-problem, simply because individuals in the same group could not mate with each other.

One of the motives behind this work is to investigate whether or not it would be better to try to solve the main problem together with each sub-problem all in parallel. The only way a GA can solve a problem is, of course, through evolution, so individuals in the same lists, up to now physically unable to do so, would have to be permitted to mate. For this reason, individuals are no longer of a given gender, but are assigned a *race*, which allows every individual to mate with every other, but still makes an individual's membership of a list readily identifiable.

3.1 Racial Preference Factor

If the original Pygmy Algorithm is taken to be using races, then each race would *always* outbreed. The descriptive name chosen for an individual's tendency to outbreed is *Racial Preference Factor* (R.P.F.) - which is simply a measure of the probability that an individual will choose an individual from the *other* race when selecting a mate. In the case of the original Pygmy Algorithm, individuals display behaviour characteristic of having an RPF of 100%, i.e. always outbreed.

Initial experiments were designed to investigate whether or not it was worthwhile using RPFs of different values - which would permit inbreeding within each race to a certain degree. As there was no way of knowing in advance which value of RPF (if any) would be the optimal, several experiments varying the value of the RPF were carried out. The first experiment was to evolve a minimal sorting network with six inputs using a population of 100 individuals. The RPF was varied from 0% (always inbreed) to 100% (always outbreed), and yielded the results as shown in figure 1.

Although experiments for *each* value of RPF were repeated on 3000 different initial populations - the same 3000 for each value to aid comparison - there was no one value for the RPF which was obviously better than the rest. However, an important result was that all of the higher results were in the range 20% to 40%, which shows that always outbreeding in this case was not the optimal strategy, and that inbreeding to quite a significant extent improves performance. To what extent is, at this stage, still unclear.

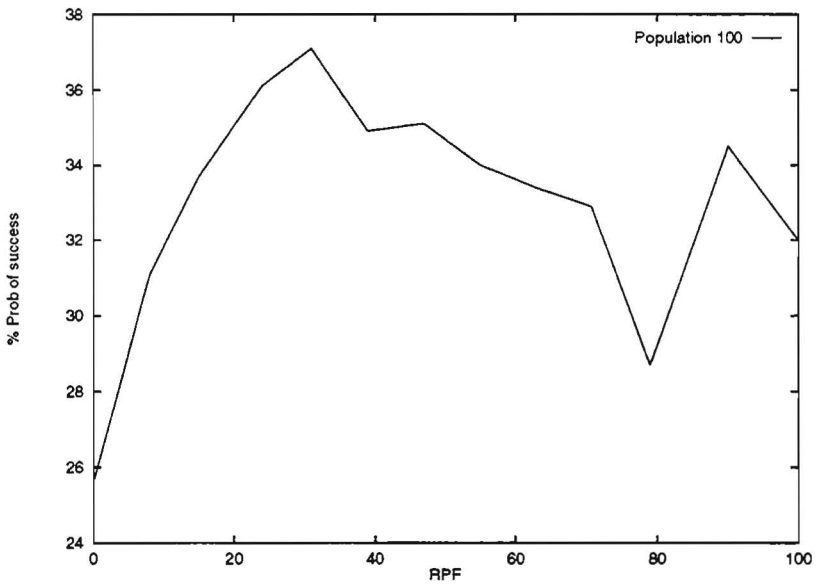


Figure 1 : Varying the RPF with a population of 100

To further test the result of there being no optimal value for the RPF, the same experiments were carried out on a number of other population sizes, ranging from 150 to 300 individuals, and yielded the results as in figure 2.

These experiments served to add to the confusion over the value for the RPF: not only were there several different possible optimal values, but these also changed when the population size changed. The only consistent results were that always inbreeding, not surprisingly, yielded poor results, and that a RPF of around 80% also tended to produce poor results.

The conclusion can only be that there is no single optimal value for the RPF, rather that it is dependent on the initial state of the population. It is certain that an RPF of somewhere between 15% and 75% would be best, but such vagueness would be useless when using the Pygmy Algorithm on different problems - how could one choose the value of RPF in advance? If there is some uncertainty about the optimal value, as there is here, which value should one choose?

4 Tuning the RPF

The conclusions of the previous section show how dependent on the initial state of a population the RPF is. An ideal situation would be to choose a separate value RPF for each population, which would be dependent on that population.

A similar view was taken by (Baeck,1991) when trying to select an optimal value for mutation rate for a population. Trying to find an optimal value for mutation rate (De Jong,1975), (Grefenstette,1986), (Schaffer et al,1989) yielded

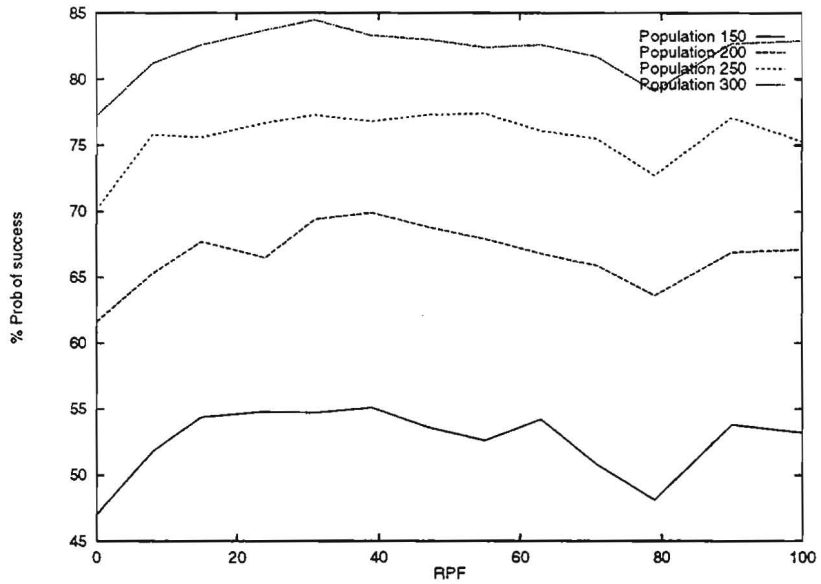


Figure 2 : Varying the population over a number of RPF values

much the same conclusions as the early experiments in this paper on RPF: the optimal value of mutation rate varies from problem to problem, and even from population to population within a single problem. The approach taken was to incorporate mutation rate as part of an individual's genes and allow it to *evolve* as the population evolved. Like other genes, an individual's mutation rate could be subjected to crossover and mutation. This strategy meant that each individual had its own mutation rate which it would examine when testing to see if mutation was to be performed. Baeck found that this improved the performance of his GAs over those GAs for which he arbitrarily selected a value for mutation rate.

Applying this strategy to RPF, each individual was assigned its own personal RPF which reflected its attitude to outbreeding. This attitude was shaped by the experience of its parents and ancestors - an individual who was the product of outbreeding would be more inclined to outbreed, reasoning that if it worked for its parents, then it should help it produce children with good performance.

4.1 Meta-GA

Three approaches were taken to the tuning of RPF. All three involved individuals having their own RPF which could evolve in the same manner as any other gene. The three approaches are as follows :

- **Species Average (SA)**
- **Racial Average (RA)**
- **Individual Average (IA)**

The SA experiments maintained a single value for RPF which was simply the average of the entire species. While this does incorporate the overhead of calculating the average of the parent population - 20% of the entire population - it does have the advantage of allowing one to keep track of the effective RPF as the population evolves.

The RA experiments maintained *two* values for RPF - one for each race, with each RPF being the average of that race. As a run progressed, the two values deviated considerably from each other and even changed at different rates, showing that, depending on the current state of the population, different amounts of inbreeding and outbreeding suited each race.

Finally, the IA experiments maintained a seperate RPF value for each individual, and individuals did not consult or examine the RPF of other individuals when choosing a mate. This approach is the closest to that of (Baeck,1991) but had the slight disadvantage of making it impossible to figure out what exactly was happening to the value of the RPF.

To maintain a balance, half of the parents were chosen from each race, and these parents then chose from which race they wanted their mate.

The results are shown in figure 3.

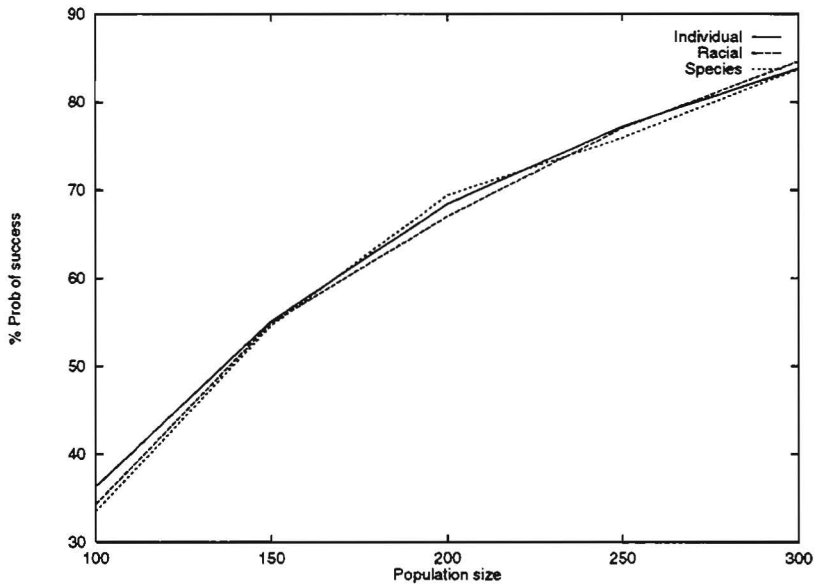


Figure 3 : Evolving the RPF

Despite the advantages of being able to track the effective value of RPF in both the SA and RA experiments, the IA yielded the best results, and at the smallest computational cost. Clearly, allowing individuals their own RPF is the best method.

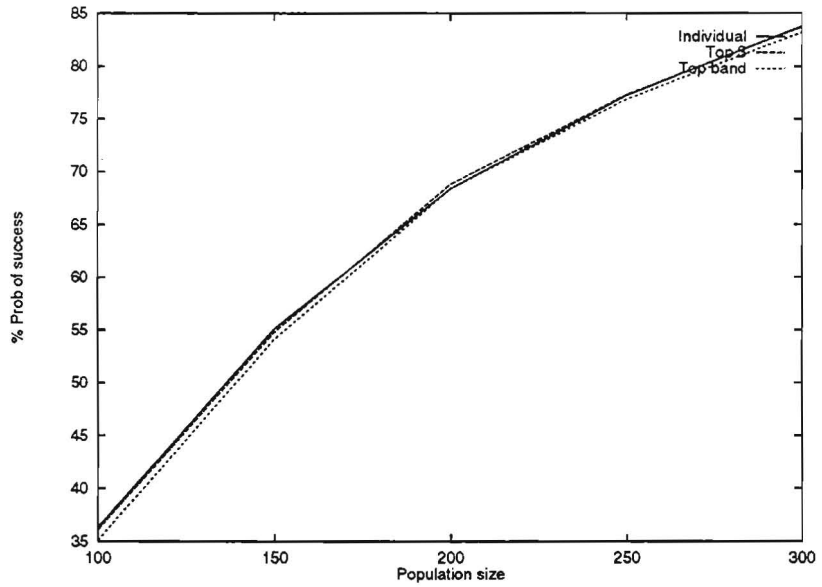


Figure 4 : A comparison of evolving RPF and fixed RPF

Figure 4 shows the best of the evolving RPF experiments compared to two of the fixed RPF results, against an average of the top three results for each population and against the top band for each population, typically in the region 15% to 75%. Although the results appear practically identical from the graph, the evolving RPF, henceforth known simply as "IA", slightly outperformed all of the fixed experiments, with the bonus that using IA did not involve many runs to try and find the optimal value for RPF. The IA experiments did not perform better than the best result found by brute force, but as the values for brute force involved some 39,000 experiments it was felt that using an average of the top results gave a fair enough impression.

4.2 Sociological Modelling

So far, as in all previous implementations of meta-GA, the RPF of an individual is looked upon solely as being a genetic feature. However, because of what this paper is modelling - the behaviour of individuals in races - it was felt that treating the RPF as an *attitude* rather than simply as a phenotypic trait would be more appropriate. As well as usig RPF, individuals used a variety of methods for calculating their own RPF. Some individuals were incapable of making up their own minds and simply followed the prevailing opinion, the same as SA above, while others were a bit more tribal in their attitudes, following the general opinion of their race, in the same manner as RA. A final, independently-minded group were the same as IA, in that they made up their own minds when deciding their RPF.

Taking RPF to be an attitude loses none of the ability to perform the experiments outlined above, but, like any opinion, RPF can be influenced or swayed by other opinions, and this observation led to another suite of experiments.

- Free Choice Model(FCM).
- Independence from Prevailing Opinion.
- Influence by other individuals.
- Opinion reinforcement.

The first set of experiments, the free choice model, influenced by (Todd,1991), allowed individuals the choice of whether or not to accept another individual as a mate. This was implemented as below:

1. Select *father* from one race.
2. Select, according to father's RPF, which race to choose a mate from.
3. Select individual (*mother*) probabilistically from that race.
4. Test, according to mother's RPF, if she wants to mate with an individual from the father's race.
5. If she accepts the father's overtures, then mate, otherwise select another mother.

If, after trying nine attempts, an individual cannot persuade any others to mate with him, he is deemed too unattractive and is rejected.

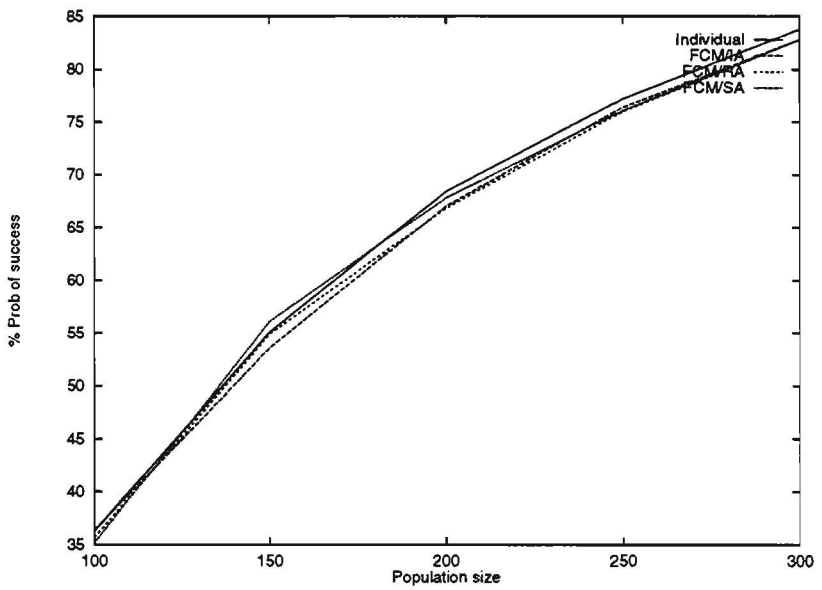


Figure 5 : A comparison of the Free Choice Model and IA

Figure 5 above shows that allowing individuals free choice of whether or not to mate with a potential suitor didn't give any improvement over the original IA experiments. Several other experiments were tried, varying from allowing individuals of type RA and SA some degree of independence from the racial or species average, to permitting individuals to influence each other to some extent. As the most interesting results have all come from the IA type experiments, only the extensions to these will be discussed.

4.3 Influential Partners

Like any opinion, RPF can also be subject to change. In this section, individuals were influenced by (potential) partners, which allowed their RPF to vary depending on that of those around them. The implementation was as follows:

1. Select *father* from one race.
2. Select, according to father's RPF, which race to choose a mate from.
3. If father's RPF>mother's RPF, then let her effective RPF be the geometric mean of the two.
4. Test, according to mother's *effective* RPF, if she wants to mate with an individual from the father's race.
5. If she accepts the father's overtures, then mate, otherwise select another mother.

The reasoning behind this experiment is rooted more in sociological than biological thought. If the probability of the father choosing the mother is greater than the probability of her accepting his advances, then her RPF is adjusted upwards - reflecting the influence his enthusiasm has on her. Two different versions of this experiment were run - in the first, the mother's RPF returned to its initial value after each mating, whereas in the second, the value of her RPF remained at the new value, reflecting a situation where a mate had a lasting effect on her.

The rather easily led individuals in this experiment did not fare too well relative to the IA experiments, as can be seen in Figure 6, and resulted in lower performance under every circumstance. Clearly, relying on other individuals for information about how to behave to other races does not help the society as a whole.

4.4 Opinion Reinforcement

The final experiment concentrated solely on the IA experiment, as this had yielded the best results so far. Again, individuals were allowed to change their RPF during their lifetime, and any changes were permanent. All changes were based on their *own* experiences and, based on the results of the experiments in which individuals were influenced by each other, did not concern themselves with the opinions of anybody else. This was implemented as follows:

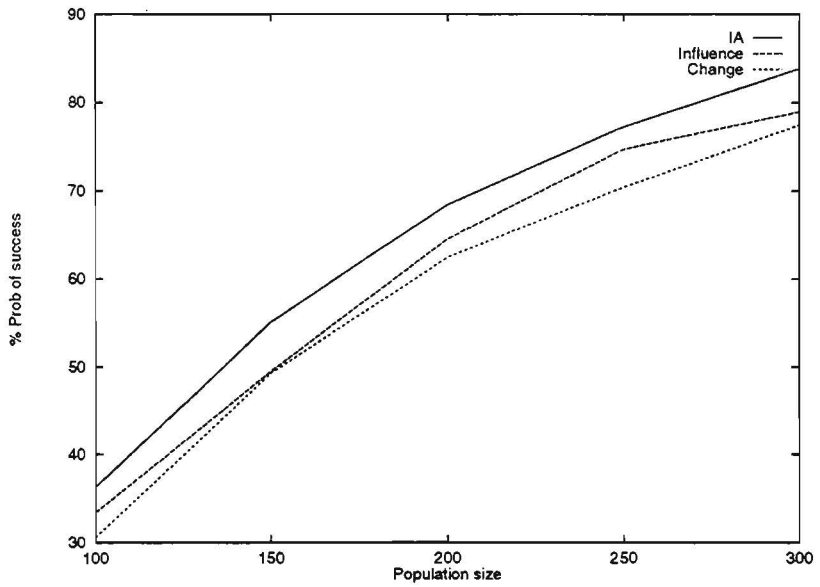


Figure 6 : A comparison of the influential models and IA

1. Select father from one race.
2. Select, according to father's RPF, which race to choose a mate from.
3. Produce child.
4. Test child.
5. If child is fit enough to enter parent population, adjust father's RPF so he is more likely to make the same decision the next time he mates, otherwise adjust the RPF so the father is less likely to make the decision.

Of all the experiments which exploited the fact that RPF was an opinion, the Opinion Reinforcement fared best, giving the same performance as IA. This serves to confirm that allowing individuals to make up their own minds, whether through evolution or from personal experience, leads to better performance than either forcing a value on them, or by letting them influence each other.

5 Conclusion

An extension to the Pygmy Algorithm, the use of races, has been introduced and is shown to outperform the Pygmy Algorithm in multi-objective functions. The increase in performance is due solely to the fact that individuals are allowed to inbreed when it suits them, and hence allow the algorithm to break up the problem and concentrate on solving separate parts of it in parallel.

Also demonstrated is the power of meta-GA, the evolving of the control parameters of a GA, and it is shown that when in doubt about the value of a

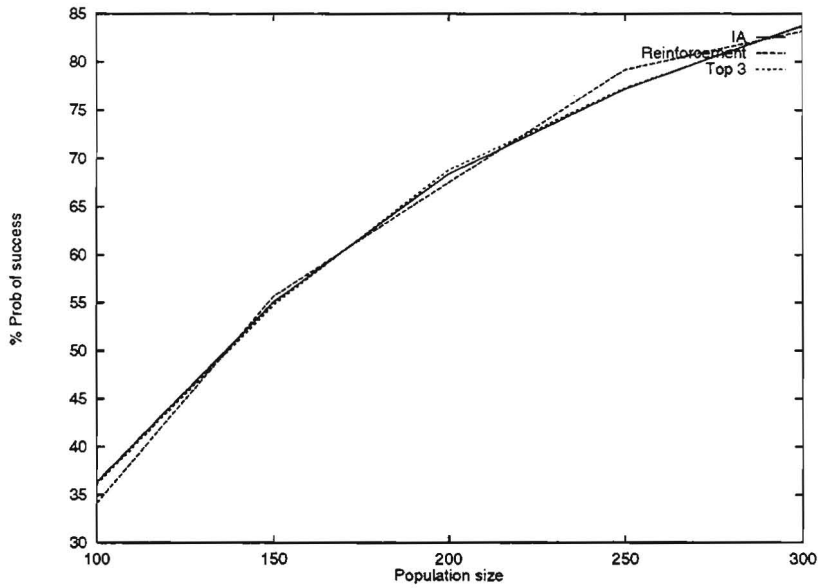


Figure 7 : A comparison of the Reinforcement Model, IA and the top 3.

parameter, it is better to evolve it, which allows the parameter to change as the population changes.

The final set of experiments investigated were those that involved sociological modelling on a coarse level and, in a curious parallel with human societies, experimental results showed that situations where individuals are allowed make up their own minds and judge from their own experiences produce much better performing societies than those where individuals are easily led, and give more weight to the prevailing opinion of their society than to their own feelings.

References

Baeck, T. : *Self-adaptation in Genetic Algorithms* in Proceedings of the First European Conference on Artificial Life. MIT Press (1992)

De Jong, K. : *An analysis of athe behaviour of a class of genetic adaptive systems* PhD thesis, University of Michigan, Ann Arbor, MI. (1975)

Eshelman, L. : *Preventing premature convergence in Genetic Algorithms by preventing incest* in ICGA4, R. K Belew, L.B. Booker, Eds. San Mateo, CA : Morgan Kaufmann. (1991)

Goldberg, D and Richardson, J : *Genetic algorithms with sharing for multimodal function optimisation* in ICGA2 J. J. Grefenstette, Ed. San Mateo, CA : Morgan Kauffmann. (1987).

Grefenstette, J.J. : *Optimization of control parameters for genetic algorithms* in IEEE Transactions on Systems, Man and Cybernetics, SMC-16(1):122-128. (1986)

Ryan C. : *Pygmies and Civil Servants* in Advances in Genetic Programming, K. Kinnear Jr., Ed. MIT Press. (1994)

Schaffer, J.D. et al : *A study of control parameters affecting online performance of genetic algorithms for function optimization* in ICGA3 J.D. Schaffer, Ed. CA : Morgan Kaufmann.

Todd, P. and Miller, J. : *On the sympatric origin of the species: Mercurial Mating in the Quicksilver model* in ICGA4, R. K Belew, L.B. Booker, Eds. CA : Morgan Kaufmann. (1991)

This article was processed using the LaTeX macro package with LLNCS style

Selectively Destructive Re-start

Jonathan Maresky, Yuval Davidor*, Daniel Gitler, Gad Aharoni and **Amnon Barak**
Department of Computer Science,
Hebrew University of Jerusalem, Jerusalem 91904, Israel
Email Correspondence: yuval@wisdom.weizmann.ac.il

Abstract

This paper motivates for, and introduces a macro genetic operator called selectively destructive re-start. This operator uses a convergence evaluation function to determine when computing resources would likely be better used in re-initializing the population and beginning the search process in a new area of the search space. Genes are re-initialized according to some probability; this has the result of keeping some genetic information from the converged run, at the same time as introducing new genetic diversity. The use of this new operator is demonstrated for two problems of Quadratic Assignment and Dynamic Control.

1 Introduction

A *Genetic Algorithm* (GA) is a search process that is motivated by natural selection and other concepts from natural evolution. While its search is not guaranteed to produce an optimal solution, it is often characterized by finding good solutions in a reasonable time, using a reasonable amount of resources. Thus, any comparison of the performance of two genetic algorithms over a problem domain depends on both the quality of solution and the execution resources.

Consequently, algorithms may be compared on the basis of solution quality while holding the number of evaluations constant, or on the basis of resources used until a certain quality of solution is found.

In both cases, resources may be wasted by an algorithm searching an area not containing a solution of sufficient quality, where any possible improvement in the solution quality is not justified by the resources used.

The *macro* genetic operator *re-start* - well-known to practitioners and researchers in the field - observes the level of genetic convergence of a population and decides when resources would be better utilized in re-starting the search in a new area, with a new population.

This involves a tradeoff between the probability of discovering sufficiently better individuals within the current area of the search space, and the use of extra resources in creating a new population and improving it to a similar level as the previous converged population.

An extension to the re-start operator is proposed here, which allows for different rates of population re-initialization. On re-starting the population after convergence, we introduce a probability of re-initialization, for each gene of an individual. We call the resulting operator *selectively destructive re-start*. This operator improves on the unselective version of re-start considerably although this improvement is highly dependent on the selection rate parameter.

The remainder of this paper is organized as follows: the next section motivates for and explains the re-start operator, and describes the approaches used for comparing GAs using re-start. Section 3 introduces the selectively destructive re-start operator. The fourth section describes the test-bed of problems, analyses the differences between the two re-start operators on these problems, and briefly presents results. Section 5 discusses related and future work, while the last section draws our conclusions.

2 The Re-start Operator

A genetic algorithm generally consists of a number of iterations of a loop which produces new individuals and inserts them into the population, subject to certain constraints (see Figure 1).

```
Initialize population
While termination condition false
{
    Produce new individuals,
    Insert these into the population
}
Report on results
```

Figure 1: *Outer loop structure of a Genetic Algorithm*

In terms of the inner loop, a *Generational Replacement* GA produces many new individuals during one inner loop, while a *Steady State* GA produces few new individuals, relative to the size of the population. This work is based on an elitist Steady State GA with unsophisticated real number crossover, mutation and termination

*Schema Ltd., 61a Hadar St., Herzlia

conditions.

Regardless of the approach, the normal behaviour of the GA is such that the amount of genetic diversity decreases as fitter individuals are discovered; the area of search becomes more concentrated and the possibility of chancing on new genetic material is limited. This is called *convergence*: the GA has likely settled on some local optimum in the search space of the problem.

In order to deal with this characteristic, we utilize the genetic operator *re-start* which, in order to continue the GA's search, introduces new genetic information and moves the GA into another solution region.

Re-start is a *macro* genetic operator. In contrast to the *micro* genetic operators of *crossover*, *mutation* and others, which function at the gene and chromosome level in producing new individuals, the re-start operator functions at the population level.

The re-start operator uses an evaluation function to decide when the population has likely converged. The population is then re-initialized, and the convergence process begins again.

Why is this operator necessary? As a population's genotypic diversity decreases, it is less likely to produce strikingly different individuals, and may be wasting resources searching an area of which the genetic possibilities have been exhausted. At some point it is necessary to abandon the population and utilize computing resources more efficiently (see [3], [4] and [7]). It has been shown, moreover, that a GA with optimal population size and the appropriate number of re-starts, exhibits better performance than the same GA functioning without the re-starts, and an appropriately larger population size([6]).

The re-start operator functions *outside* the main loop of the GA (see Figure 2) and is reliant on some definition of convergence.

```
While termination condition false
{
    Initialize population
    While population has not converged,
      and termination condition is false
    {
        Produce new individuals,
        Insert these into the population
    }
}
Report on results
```

Figure 2: *Outer loop structure of a Genetic Algorithm with* re-start *operator*

Determining the "when" of re-start, the extent of genetic convergence, is exceedingly costly: this would involve the variance analysis of every gene within the population. However, this may be estimated by observing the relative improvement of the population's best individual, also called the *best-of-generation* individual, as a function of the number of iterations. A relatively steep curve indicates much exploration of new genetic material, while genetic convergence tends to imply a flattish curve. In Figure 3, it is fairly easy to determine when it is necessary to re-start the execution.

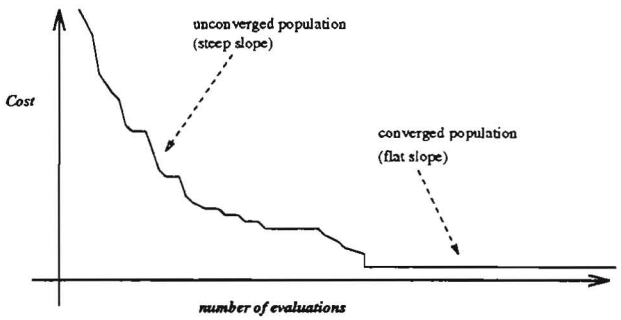


Figure 3: *Using the cost improvement curve to determine convergence*

The calculation of the extent of convergence is a function of:

- The number of evaluations since the last improvement of the best-of-generation individual. Longer gaps between updates implies greater convergence.
- The relative magnitude of the improvement. Larger improvements often affect the process more than smaller improvements.
- The population size. A larger population will generally widen the gap between best-of-generation updates.

Note that the evaluation function is relatively problem-specific.

Note also that convergence does not imply that the best-of-generation individual would not improve. Rather, it is probable that the resources used in attempting to improve the best-of-generation individual would be better used in re-starting the population and continuing the execution from there.

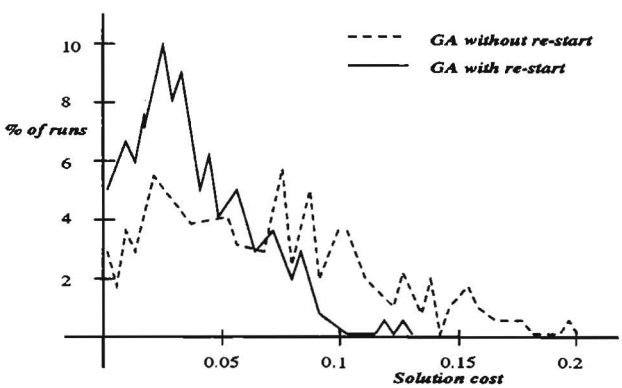


Figure 4: *Probability distributions of GAs with and without re-start*

For example, we compared the functioning of the GA with re-start against the original GA, on the *bal-14* problem. The probability distribution graph of solution costs is shown in Figure 4.

Running for 20000 evaluations, the GA with re-start produced a mean result of 0.0428, with standard deviation 0.026. The normal GA without re-start, produced a mean result of 0.0697 with standard deviation 0.045 ($n = 250$). Despite the extra resources invested in re-starting the new population, the re-start operator produces better, more stable results.

It is possible to approximate the behaviour of the GA with re-start by observing the probability distribution of the GA without re-start. Each sub-run, from reinitialized population to convergence, constitutes a unit within the distribution, and is independent of sub-runs before and after. Thus the re-start operator effectively re-starts the execution with a new initial population, whose outcome is unrelated to the previous outcome.

3 The Selectively Destructive Re-start Operator

We investigated an extension to the operator: instead of completely destroying the genetic material present in the population at convergence and thereby wasting the results of much computation, the re-start probabilistically reinitializes genes in the creation of new individuals.

The resulting operator is called *selectively destructive re-start* (SDR) because it does not completely destroy, and re-initialize, the converged population before beginning the convergence process, in the manner of the normal re-start operator; a percentage of the converged genes will survive untouched to begin the next convergence stage. The resulting GA's structure is outlined in Figure 5.

```
While termination condition false
{
    Initialize population:
      for each gene, for each individual,
      with probability = selection rate,
      the gene is reinitialized
    While population has not converged,
      and termination condition is false
    {
        Produce new individuals,
        Insert these into the population
    }
}
Report on results
```

Figure 5: *GA with* selectively destructive re-start *operator*

This percentage is the *rate of selectivity* of the operator: the higher the rate, the more genes are reinitialized.

It is fairly easy to compare GAs which use re-start and SDR: the difference in the re-initialization process is hidden from any comparison.

4 Results

4.1 Problems

Two problems are selected for testing. One, *bal-14*, a Quadratic Assignment problem, and one, *JM1*, a Dynamic Control problem.

The Quadratic Assignment problem ([1]) involves 14 dimensional quadratic minimization, which has many applications in dynamics (though the dimensionality may vary).

Given a structure which is not dynamically balanced around its rotational axis Z (centre of gravity is not on the Z axis), and n locations in the structure in which mass can be added or subtracted, find that combination of mass changes that minimizes the yaw and pitch movements and the total mass added.

$$\min(C\Delta W + (1 - C) \parallel M \parallel)$$

where ΔW is the net weight added, C is some constant, and $\parallel M \parallel$ is the resulting Z right angle moment.

The upper and lower bounds for the point masses, the cylindrical coordinates z_i, r_i and φ_i (representing distance along the Z axis, radius, and angle from some reference radius, respectively) for each of the n points, and the initial mass of the structure are given.

At low dimensions, the problem may be solved using the Simplex method which guarantees optimality. However, numerical problems are encountered at high dimensions which cause convergence to a local optimum and a GA approach becomes attractive.

Each individual is coded as containing a chromosome of 14 real numbers, a total mass, moment, cost and a fitness value. The solution space size is of the order of 10^{287}.

The Dynamic Control problem is from [5]:

$$\min\left(x_N^2 + \sum_{k=0}^{N-1}(x_k^2 + u_k^2)\right)$$

where

$$x_{k+1} = x_k + u_k, k = 0, 1, ..., N - 1,$$

where x_0 is the initial state, $x_k \in \mathcal{R}$ is a state, and $\vec{u}$ is the solution vector.

We call the following problem *JM1*: a fixed domain of $(-200, 200)$ is assumed for each u_i, with $x_0 = 100$ and $N = 45$. In similar fashion to the *bal-14* problem, each individual is represented by a 45-ary vector of real numbers, a cost and a fitness value.

4.2 Analysis

In order to compare GAs using re-start, the following approaches were used:

- In the *evaluational* approach, the number of evaluations is held constant, and the GAs are compared by using the mean result obtained.
- The *good-enough* approach searches until an acceptably good solution is found, and GAs are compared by observing the mean number of evaluations required to arrive at the solution.

The new operator was tried on the above test-bed of functions and proved successful, in comparison to the original re-start operator, albeit at different selectivity rates for different problems.

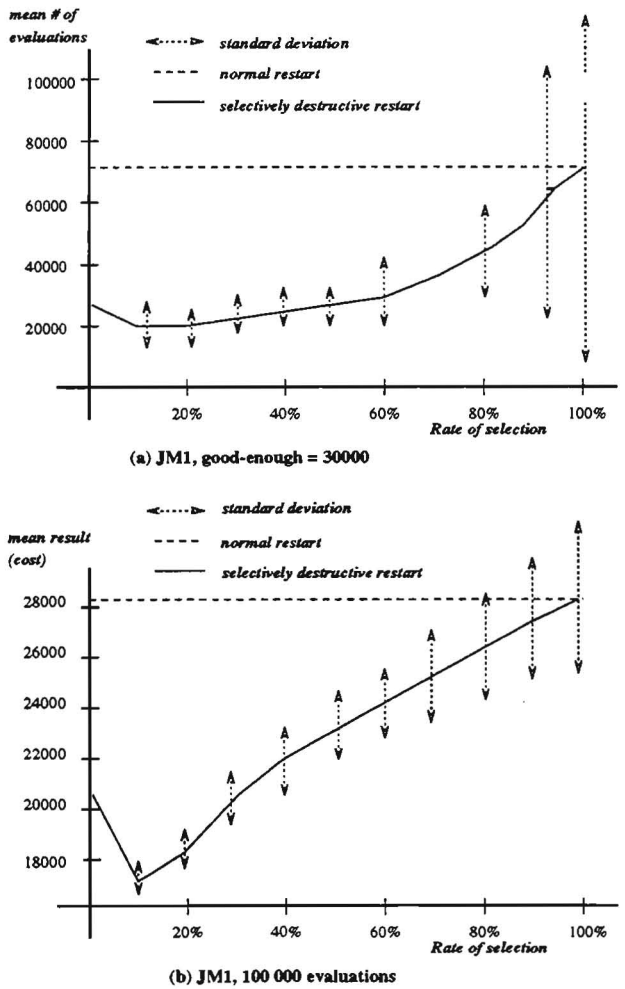


Figure 6: *Comparative performance of different re-start operators on JM1, using the* good-enough *and* evaluational *approaches*

Figures 6 and 7 compare the mean and standard deviation results of the normal re-start operator and the SDR operator on the different problems, under different selection rates, using both the *good-enough* and the *evaluational* approaches. Figure 8 gives specific results.

The monotonic *JM1* problem was chosen specifically for its behaviour under the normal re-start operator. Figure 6 shows that a GA using re-start (100% selection rate) is less effective than a GA without re-start (0% selection rate). Thus, even where using the re-start operator may not be advisable, the SDR operator improves on the core GA not using re-start.

The choice of a selection rate of 15% results in an improvement of 72% over the number of evaluations needed to produce a *good-enough* result using the normal re-start operator. The corresponding standard deviation is less than one tenth of the standard deviation of the re-start operator.

Using the *evaluational* approach, the optimal selection rate is closer to 10%, and results in an large mean solution improvement over the re-start operator GA, using 100000 evaluations. The standard deviation is also considerably lower than that of the re-start GA.

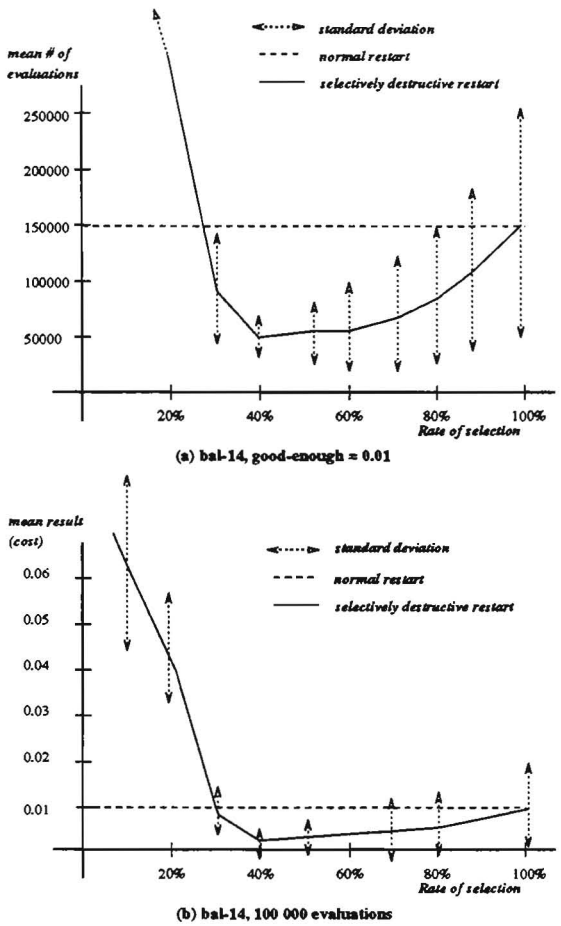


Figure 7: *Comparative performance of different re-start operators on bal-14, using the* good-enough *and* evaluational *approaches*

For the 14-dimensional *bal-14* problem (Figure 7), a 40% rate of selection is preferred using the *good-enogh* approach: it results in a performance improvement of over 65% compared to the re-start approach's mean number of evaluations used, and the stability also improves considerably.

Using the *evaluational* approach, the same selection rate produces a mean result which is one third of the re-start GA's mean result. The standard deviation is also improved.

We noticed that low selection rates on the Quadratic Assignment problem produce very poor results - the slight change in genetic material as a result of re-start introduced by the SDR operator is not sufficient to move the algorithm to a new search area.

The new operator results in impressive savings in resource utilization: it finds *good-enough* solutions quicker,

and produces considerably better and more stable results given a fixed number of evaluations. These *evaluational* results seem more striking, but it is not possible to quantify the improvement, because of the nonlinearity of this relationship. Note also that the optimal selection rate is not necessarily the same for *good-enough* and *evaluational* executions (see Figure 8).

Problem	(1)	(2)	(3)	(4)	(5)	(6)
JM1	*19950(5478)*	71525(66728)	15%(72%)	*17376(382)*	28306(2369)	10%
bal-14	*52656(32117)*	152002(144413)	40%(65%)	*0.0031(0.0047)*	0.013(0.01)	40%

(1) Good-enough: mean evaluations (std. deviation) - Selectively Destructive Re-start
(2) Good-enough: mean evaluations (std. deviation) - Re-start
(3) Best Selection rate for good-enough approach (improvement)
(4) Evaluational: mean result (std. deviation) - Selectively Destructive Re-start
(4) Evaluational: mean result (std. deviation) - Re-start
(6) Best Selection rate for evaluational approach

Figure 8: *Optimal selection rates for different problems ($n = 250$)*

Different problems have different optimal rates of selection; additionally, one problem may have different optimal rates of selection using different comparison approaches. It is an open question which feature of the problem influences this parameter.

In terms of computing resources used, in most cases the SDR operator uses slightly more resources than the normal re-start operator: apart from a call to the random number generator to determine if a gene is to be re-initialized, another random number call is necessary to produce the new value, although fewer stores are used by the new operator.

Selectively destructive re-start is superior to normal re-start and provides us with an improved method of renewing genetic diversity in genetic algorithm search. Intuitively, the complete reinitialization of the population "forgets" the previous solutions, and the chance of arriving at the same result, or even the same general area of search, is minimal.

In contrast, the use of a selective rate of reinitialization keeps the previous solution within the convex hull of the new population, even though it is unlikely that the previous solution be a member of the new population. The new run will in most cases improve on the previous result, in contrast to the normal re-start operator which has no memory of previous solutions.

This *a priori* analysis is mostly borne out by observation of executions of the model running the SDR operator on different populations: the function of best result, per new re-started population, against number of re-starts is almost monotonically improving, whereas the normal re-start produces the expected erratic curve of statistically independent intermediate results.

The apparent monotonicity of the new operator is not provable, however. The stochastic nature of the process will produce unexpected results in response to the slightest difference in a crossover or mutation, exactly as in nature.

5 Related and Future Work

Eshelman ([2]) introduced a similar operator as part of his nontraditional genetic algorithm. His operator, called *restart* or *partial initialization*, is used as a more global substitute for mutation, where mutation is moved from inside the reproduction-recombination loop to outside of the loop. It was not analyzed separately from Eshelman's other new operators, with the result that its performance improvement has been mostly unknown.

In understanding the reasons behind the superiority of selectively destructive re-start over normal re-start, we have not attempted a theoretical analysis. A more thorough mathematical investigation of the genetic reasons for this new operator's workings is necessary.

6 Conclusions

The re-start operator provides an effective mechanism for continuing the execution of a converged population. A simple and computationally inexpensive extension to the operator produces the selectively destructive re-start operator, where the selection of an effective rate of re-initialization results in impressive improvements in execution performance as well as solution quality. At low selection rates, where the converged population is re-started by changing relatively few genes, certain problem domains react adversely and have difficulty in improving their result. Additionally, selectively destructive re-start benefits problems where the normal re-start operator provides no improvement over the GA without re-start.

References

[1] Davidor, Y. An ECOlogical Model for Evolutionary Computing. In *The Japanese Journal for Systems, Control and Information*, 1993, vol. 37, no. 8.

[2] Eshelman L.J. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *Foundations of Genetic Algorithms* (Edited by Gregory J.E . Rawlins), 1991.

[3] Goldberg, D.E. Optimal initial population size for binary-coded genetic algorithms. *Technical Report No. 85001*, TCGA, 1985.

[4] Goldberg, D.E. Sizing populations for serial and parallel genetic algorithms. In *Proc. 3rd Int. Conf. on Genetic Algorithms* (Arlington, VA), 70–89, 1989.

[5] Janikow C.Z., Michalewicz Z. An Experimental Comparison of Binary and Floating Point Representation in Genetic Algorithms. In *Proc. 4th Int. Conf. on Genetic Algorithms* (San Diego, CA), 31–36, 1991.

[6] Nakano R., Davidor Y., Yamada T. Optimal population size under constant computation cost. In print: *Parallel Problem Solving from Nature 1994*, Springer-Verlag.

[7] Robertson G.G. Population size in classifier systems. In *Proc. 5th Int. Conf. on Machine Learning* (Los Altos, CA), 142–152, 1988.

A Comparison of Different Evolutionary Algorithms on the Quadratic Assignment Problem

Volker Nissen, University of Goettingen, Germany. E-mail: vnissen@uni-goettingen.de

Abstract. The Quadratic Assignment Problem (QAP) is a tough standard operations research problem with relevance to different practical applications, particularly in facility layout. In this paper it serves as a vehicle to compare the performance of different types of Evolutionary Algorithms (EAs). A migration model Genetic Algorithm (MMGA), an Evolution Strategy (CES) and Evolutionary Programming (CEP) are compared on a set of QAPs. The strictly mutation-based CES- and CEP-heuristics generally outperform the crossover-based GA-approach. This may be explained by reference to recent results concerning the relation of fitness landscape and design of evolutionary operators. CES and CEP are competitive to implementations of Tabu Search and Simulated Annealing as described in the literature. The results demonstrate that even highly epistatic problems can successfully be approached with pure (non-hybridized) EAs when the search operators are designed accordingly.

1 Introduction

The Quadratic Assignment Problem is one of the toughest combinatorial optimization problems. It can be formalized as follows [5]: Given a set $N = \{1, 2 \ldots n\}$ and real numbers c_{ik}, a_{ik}, b_{ik} for $i, k = 1, 2 \ldots n$, find a permutation φ of the set N which minimizes:

$$Z = \sum_{i=1}^{n} c_{i\varphi(i)} + \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{\varphi(i),\varphi(k)}$$

where (assuming a facility layout problem)

- n : number of facilities/locations
- c_{ij} : fixed cost of locating facility j at location i
- a_{ik} : cost of transfering a material unit from location i to location k
- b_{jl} : flow of material from facility j to facility l.

The QAP is characterized by a high degree of epistasis (non-linear interaction between solution elements). Even swapping the assignment of two facilities might affect the quality of virtually all other assignments, depending on the flow-matrix. As a generalization of the TSP the QAP is NP-hard, and only moderately sized problem instances (approx. $n = 18$) can be solved to optimality with exact algorithms within reasonable time limits. One, therefore, concentrates on developing effective QAP-heuristics. Extensive reviews of the QAP and associated solution techniques can be found in [15, 5].

The QAP is of practical relevance in such diverse areas as facility layout, machine scheduling and data analysis. Vollmann and Buffa [28] introduced the concept of flow-dominance, measuring the variation of values in the flow matrix. It is given by $100 * std.dev./mean$ of the matrix elements. Simply stated, high flow-dominance indicates that few facilities with high interaction tend to dominate the problem. Burkard and Fincke [4] were the first to prove the asymptotic behavior of large randomly generated QAPs. This means that the relative difference between the worst and the optimal solution becomes arbitrarily small with a probability tending to 1 as the problem size tends to infinity. Hence, for any QAP-heuristic test-suite design becomes an issue. Here, a set of 7 QAPs varying in size between $n = 15$ and $n = 64$ with different structure (flow-dominance) have been employed. They were taken from the QAP-library collected by Burkard et al. [6].

In this paper, the performance of different variants of Evolutionary Algorithms (EAs) is compared on a test-suite of QAPs. EAs are general purpose search and optimization techniques with heuristic character that imitate basic principles from evolution theory. They differ from more conventional optimization methods in the following aspects:

- They operate on a string or vector representation of the decision variables.
- They, generally, process a set (population) of solutions (individuals), exploring the search space from many different points simultaneously.
- In their search operators, EAs imitate the evolutionary mechanisms of replication, variation and selection of individuals.

- They require only information on the quality (fitness) of solutions derived from objective function values but no auxiliary knowledge such as derivatives. However, incorporating available domain knowledge in solution representation, initialization, operators or decoding function may substantially increase the competitiveness of an EA at the cost of a reduced scope of application.
- Stochastic elements are deliberately employed. This means no pure random search, though, but an intelligent exploration of the search space.

The most prominent EA-variants are:

- Genetic Algorithms [11],
- Evolution Strategies [24],
- Evolutionary Programming [9].

Readers unfamiliar with these techniques are refered to Bäck and Schwefel [2] for a concise overview and Nissen [19] for a more detailed presentation.

Few authors developed evolutionary approaches to solve QAPs. Typically, when high-quality results were achieved, the EA had been massively hybridized with other problem solving techniques such as Simulated Annealing (SA) [3, 18] It is, therefore, difficult to assess the performance of 'pure' EAs (i.e. non-hybridized) on the QAP. This question is of particular interest when one recalls the recent debate on the power of Genetic Algorithms in combinatorial optimization.

Here, in a set of experiments the following types of EAs are compared on the QAPs:

- migration model Genetic Algorithm (MMGA),
- Combinatorial Evolutionary Programming (CEP),
- Combinatorial Evolution Strategy (CES).

Since none of the EA-types was originally intended for combinatorial optimization certain adaptations of the methodology were unavoidable. However, in the authors view all EA-types should be treated as variants under a common evolutionary paradigm, and they together constitute a set of design elements from which problem-specific EAs should be constructed that best solve the actual problem. So adaptations of the basic EA-concepts to combinatorial problems are even desirable.

2 Description of the implemented EA-variants

2.1 A Migration Model Genetic Algorithm (MMGA)

The migration model Genetic Algorithm uses a straightforward permutation coding (figure 1) as solution representation. This representation is also employed by the other two evolutionary approaches to be discussed later.

Assuming a facility layout problem, each position on a solution vector represents a facility location and integer values are assigned to these positions as numbered facilities. With this information, objective function values can be calculated from the given problem matrices and are taken as fitness information for the implemented EAs.

MMGA is a Genetic Algorithm which processes 6000 individuals (solution trials) in 200 subpopulations of 30 individuals each. Thereby, selection and recombination of mating partners are performed only locally to prevent premature convergence. All subpopulations evolve independently. During each GA-cycle the worst individual in each subpopulation is replaced (steady-state GA).

I apply deterministic binary tournament selection to determine mating partners. This means, two individuals are randomly drawn from a subpopulation, and the better is kept as the first mating partner. This is repeated to determine the second mating partner. The recombination-operator is partially matched crossover (PMX) as introduced by Goldberg and Lingle [10], since it is a rather conservative form of crossover and tends to inherit absolute string positions. In my implementation, strings are treated as rings to overcome the bias against end-positions. The better of the two offspring is kept.

For PMX to be effective identical mating partners must be avoided. Therefore, a subpopulation is blocked from further genetic action when identical mating partners are consecutively drawn twice. It is then assumed that this subpopulation has reached a state of relative convergence. Once all subpopulations are blocked, randomly drawn individuals are exchanged between randomly determined subpopulations (emigration phase). 750 such exchanges are performed, refreshing the subpopulations with locally new solutions. Afterwards all subpopulations are re-

leased and the GA-process continues as before.

A copy of the best solution in the entire population is always kept in a seperate storage and updated whenever a better solution has occurred.

This solution is output as the final MMGA-result and can optionally be improved by a hillclimber (2-Opt). 2-Opt is a simple local search heuristic that sequentially considers pairwise exchange (2-swap) between the positions of facilities. The swap is made whenever this results in a lower objective function value and the search starts again from the new solution. This procedure continues until no exchange in the current solution results in a further improvement.

Figure 2 gives an overview of MMGA in pseudo-code. As for the other evolutionary heuristics presented here, the MMGA strategy parameters are determined experimentally and no claim is made as to their optimality. The parameters are kept constant over all testproblems since a heuristic that requires problem-specific tuning of strategy parameters is not particularly user-friendly.

2.2 An approach based on Evolutionary Programming (CEP)

In this approach 25 children are generated from five parents in each generation by copying each parent five times and mutating the resulting offspring. No crossover is applied following standard practice. Mutation is based on 2-swaps of solution elements. In standard Evolutionary Programming mutation of a solution element means adding a normally distributed random variable where the standard deviation depends on the parent's fitness value. This is sensible when the global optimum is known in advance and fitness measures some error term such as the squared difference of current objective function value and global optimum. This is not the case for the QAP. However, to keep the basic idea of standard Evolutionary Programming CEP utilizes the parental fitness value when determining the mutation intensity (number of 2-swaps on the current solution). Additionally, the current generation number influences the mutation intensity. More formally, the number of 2-swaps *exchange#* during mutation of a particular offspring is determined by the following formula:

$$
\begin{aligned}
exchange\# &= \text{round}(\text{abs}(N(0,\sigma))) + 1 \\
\text{with } \sigma &= (\ln(\ln(maxgen/gen) + 1)) \cdot \\
&\quad \cdot \left(\frac{fit}{worstfit}\right)^2 \cdot \alpha
\end{aligned}
$$

where:

$N(0,\sigma)$:	normally distributed random variable with expectation 0 and standard deviation σ
$maxgen$:	maximal number of CEP-generations
gen :	current CEP-generation
fit :	parental fitness value of current child
$worstfit$:	worst objective function value in startpopulation
α :	scalar ($\alpha = \frac{2}{3}$).

Note that mutation intensity is high at the beginning of the search process and decreases on a logarithmic scale. The current state of the optimization influences the mutation intensity, firstly, by the factor *maxgen/gen* that decreases solely with time. Secondly, the parental solution quality in relation to the worst element of the startpopulation is taken into consideration. As a general tendency, the offspring of a mediocre parent will undergo a stronger mutation than children of a good parent (minimization!). Sigma decreases with continued optimization, focussing the search process. This may be compared with the cooling schedule of Simulated Annealing. The constant term in the above formula ensures that at least one 2-swap is performed to avoid a premature termination of the search process.

Following mutation, five new parents are selected from the population of parents and children (30 individuals). This is done via a stochastic tournament in which the fitness (objective function value) of each individual is compared to the fitness of seven randomly determined "opponents". An individual scores a "win" when its fitness is at least as good as the fitness of the opponent. The best five individuals resulting from a ranking based on individual wins (not fitness) are taken to be the new parents for the next CEP-generation. Thus, imitating biological processes, selection is stochastic (as in MMGA) and even bad individuals have a certain chance of surviving the competition. The best solution based on fitness is always kept in a seperate storage and updated when required. It

is the final CEP-result that is output at the end of each run. An optional postprocessing with the 2-Opt local hillclimber is possible.

Figure 3 presents the general outline of CEP as pseudo code. Again, the strategy parameters of CEP are determined experimentally. They are held constant over all QAP-experiments.

2.3 A combinatorial variant of Evolution Strategy (CES)

CES was first presented in [20]. It uses a simple population concept and is basically a $(1,\lambda)$-ES. This means, in each generation λ children are generated from one parent solution by first copying the parent and then randomly swapping integer values on the coding string via mutation. The mutation operator from standard-ES, which is based on adding normally distributed random variables to the elements of the current solution, is not adequate for such a permutation problem since it would yield invalid results. Crossover of solutions is not employed. λ equals 50 for the smaller test problems NUG15, NUG20 and ELS19. For the other problems λ equals 100. The parent is eliminated after each generation. This allows for an occasional deterioration of the parent solution. The concept has experimentally proved slightly more successful than having parent and children compete for survival.

The number of 2-swaps during mutation is randomly chosen to be either one or two. It can occasionally be zero however, should the algorithm by chance choose the same position for a swap twice. This then preserves the parent solution. In accordance with the standard ES-scheme, it would have been possible to use a normally distributed random variable for adapting the number of swaps during mutation, but it was decided to keep the number of swaps small. Intensive swapping of facilities generally deteriorates the objective function value of a given QAP-solution due to massive interactions between the facilities. Generating normally distributed random numbers is also more time consuming.

The best child becomes the new parent solution (deterministic selection). If its fitness value is not better than the former parent's value, a counter is increased. The counter is reset to zero whenever a CES-generation is successful. After *round (n/10+2)* consecutive unsuccessful generations, an empirically determined value, a procedure called destabilization is executed. This is a non-standard operator. The concept of destabilization in the context of ES is also described in [1] but realized differently. During this phase, the counter is set to zero and λ children are created with increased mutation intensity. The number of swaps randomly lies in the interval $[3,..,8]$ now. Thereby, individuals which differ more strongly from previous solutions are generated and the search shifts to a new area in the solution space. This helps to escape from local optima and counters the strong selection pressure in CES. Again, CES determines the best child to become the new parent. Procedure destabilization is then terminated and the search continues as before until the termination criterion holds.

As for the other EAs, CES starts from a randomly generated (rather poor quality) initial solution. The best solution discovered by CES is stored separately and is continuously updated during the search. This is the final result of the heuristic. It can optionally be improved with a local hillclimbing technique (2-Opt). Figure 4 gives an overview of CES in pseudo code. The CES parameters are empirical and constant over all QAP-experiments again.

3 Empirical Results

MMGA, CEP and CES were run on seven test-problems taken from [22] (NUG15, NUG20, NUG30), [26] (STE36a, STE36c), [8] (ELS19) and [25] (SKO64) with numbers of facilities n varying between 15 and 64. NUG15, NUG20, NUG30, and SKO64 are randomly generated problems with low flow-dominance. The other three appear to be practical applications with high (STE36a, STE36c) and very high (ELS19) flow-dominance values. Hence, the seven problems are very different in size and structure making them a good test-suite. The search space size (number of solution alternatives) varies from roughly $1.31 \cdot 10^{12}$ for NUG15 to $1.27 \cdot 10^{89}$ for SKO64. Ten trials are performed in each experiment. Results are given in tables 3-5. Data for generation 0 refers to the starting solutions used.

The well-known conventional combinatorial heuristic 2-Opt serves as a benchmark to compare the quality of solutions generated by CES with a more traditional QAP-heuristic (table 2). The starting solutions of CES and 2-Opt are identical. MMGA, CEP, CES and 2-Opt were all implemented in Pascal on a workstation IBM RS 6000/320 H. Also, for orientation purposes,

results of the excellent TABU search implementation (TS) of Taillard [27] are given (table 1), but one should bare in mind that TS was run on different hardware (parallel transputer system) and the evaluations of individual solutions in TS have a lower time complexity as in the heuristics presented in this article.

Examining the results for MMGA first (table 3) it is clear that the GA-approach is neither competitive with TS nor with 2-Opt when both solution quality and CPU-requirements are taken into consideration. An exception to be discussed later is ELS19. The general increase in solution quality due to continuing search efforts with rising generation number is no surprise. However, the number of solution evaluations and CPU-requirements rise only underproportionally in case of the larger test problems. This means that with time subpopulations are blocked more frequently here, and the emigration operator loses part of its power. Processing the MMGA-result with an additional 2-Opt hillclimber is useful for short MMGA-runs to overcome the well-known weakness of GA in finetuning the final solution.

Because of the chosen 2-swap based search operators in CEP and CES both heuristics allow for an especially efficient form of solution evaluation that cannot be applied to the PMX-based MMGA (see [19] for details). The number of CEP- and CES-generations are chosen as to produce roughly equivalent numbers of solution evaluations during runs, since the time complexity to evaluate individual solutions is very similar in both heuristics.

Results for CEP and a hybrid, where the CEP-solution is postprocessed by 2-Opt, are given in Table 4. Generally, the optimal solution is approached in an asymptotic manner, while, simultaneously, the reliabilty of the optimization process increases with rising generation number as can be seen in the reduced standard deviation. For ELS19 the paradoxical situation occurs that the shortest run produces the second best CEP-result. One should bare in mind, though, that depending on the max. generation number the search process for CEP proceeds differently for runs of varying length. This is due to the influence of the factor *maxgen/gen* on the mutation intensity.

Very good or optimal values are quickly identified for the smaller test problems NUG15 and NUG20 as can in the Best-Gen. column. Contrary to this, CEP is weaker on ELS19, the problem instance with the highest flow-dominance. This phenomenon will be discussed later.

Improving the final CEP-result by 2-Opt is useful only for the largest testproblem SKO64. In all other cases CEP-solutions are already (at least nearly) locally optimal. The hybrid strategy of CEP and 2-Opt may be described as hill-finding (CEP) and hillclimbing (2-Opt). That such a combination is sensible can be seen when comparing with the pure 2-Opt results (table 2). After 2500 generations the hybrid produces on average much better solutions with greater reliability (lower std.dev.) and lower CPU-requirements. Examining the additional CPU-seconds for 2-Opt, it is interesting to note, how search-load shifts from the hillclimber to CEP as the length of the CEP-run increases.

On large problem instances CEP is more efficient than 2-Opt when solution quality and CPU-time are considered simultaneously. Compared to TS the hybrid of CEP and 2-Opt produces results on a similar level.

Various authors have asserted that in complex search spaces low selection pressure is appropriate to avoid premature convergence of an EA (e.g. [13]). For CEP, selection pressure can be influenced by altering the ratio of parents to children as well as by varying the number of individual tournaments during the stochastic selection phase. Results for the second alternative are given in table 6. More results can be found in [19]. CEP is mildly influenced by the number of individual tournaments during selection. The most successful selection pressure is still comparatively high (Seven tournaments for each individual = 23 percent of the entire population). This is slightly surprising and demonstrates the difficulty to give general recommendations for strategy values that are independent of the particular EA-design and the application in question.

CES, as the last EA-variant to be discussed here, quickly identifies good solutions on all seven problem instances (table 5). Not surprisingly, average solution quality and standard deviation improve with the number of generations because more solution trials are performed. Destabilization proves to be an important heuristic element. It counters the negative effects (tendency to premature convergence) of the extreme selection pressure in CES. The average number of destabilizations can be calculated from the number of function evaluations, because without destabilization there would

only be $\lambda \times$ *generations* $+ 1$ function evaluations. On larger QAPs there is a tendency for fewer destabilization phases. One reason for this phenomenon is the way the allowed number of consecutive unsuccessful CES-generations before destabilization is computed. The larger n the higher this maximum value. Also, as the problem dimension increases, it becomes easier to leave a local optimum. But because the size of the search space ($n!$) rises drastically, though, the search process requires more time to identify high quality solutions than on smaller problem instances.

CES quickly produces on average far better solutions with more reliability than 2-Opt. Even after only 100 generations CES often generates better mean values, thereby challenging the speed-advantage of more conventional heuristics on larger problems. It also converges to better solutions with higher reliability as shown in table 7. CES also competes well with TS in terms of solution quality and efficiency.

Table 5 shows that postprocessing CES-results with a local hillclimber can lead to improved solutions on larger problems. As with CEP, to achieve results of very high quality, though, it is necessary to allow for a reasonably long search phase of the evolutionary component before invoking 2-Opt. Also, on smaller problems the CES-solutions are usually (nearly) locally optimal so that a 2-Opt becomes superfluous.

CES appears to be an effective, easily implementable heuristic that yields good results without problem-specific parameter tuning on QAPs of very different size and structure. It has acceptable CPU-requirements even on a serial computer. In [21] the authors extend the CES-heuristic to include practically relevant constraints of facility layout.

Finally, comparing the algorithmic performance of MMGA, CEP and CES on the QAPs leads to the following results and conclusions:

The strictly mutation-based ES- and EP-approaches outperform the crossover-based GA on all instances but the problem with the highest flow-dominance value ELS19 (see efficiency comparison in figures 5, 6 and 7). This may be explained by reference to results of Manderick et al. [17] and Lipsitch [16] concerning the relation of fitness landscape and design of search operators. On the one side, when flow-dominance is low or medium, a highly multimodal QAP fitness-landscape results from the epistatic nature of the problem. Only small changes from parents to children, as in the 2-swap based mutation, lead to a certain correlation of their fitness values. Even such comparatively conservative crossover-forms as PMX frequently produce behavior close to random search by changing solutions too much at a time when population diversity is reasonably high. In general, the fitness landscape for CEP and CES is smoother than for MMGA due to the different search operators employed. However, some search behavior close to random search is also present during the early phases of CEP and it is deliberately employed during the destabilization phases of CES. On the other side, when flow-dominance values are very high, 2-swap-based heuristics have difficulties in overcoming pronounced local suboptima, so that crossover is advantageous. All three EAs discussed here can in principle be parallelized. For more details see [19].

4 Conclusions

Overall, the combinatorial variant of an Evolution Strategy (CES) is the best EA presented here. The non-standard destabilization operator in the ES-implementation is useful in overcoming local optima when selection pressure is high as in CES. It is also effective on problem instances with high flow-dominance that prove difficult for other heuristics purely based on pairwise exchange of solution elements.

Despite the high degree of epistasis of the application, CES and CEP (possibly combined with an additional 2-Opt) are successful optimization techniques for Quadratic Assignment Problems. CEP and in particular CES are competitive to implementations of Tabu Search (or Simulated Annealing) as described in the literature. Contrary to some TS-implementations, they require no tuning of strategy parameters on individual problem instances, thus making the EAs particularly user-friendly.

On larger problem-instances a postprocessing of the final EA-solution with a simple 2-Opt hillclimber can improve the quality of results slightly at very low cost. One has to be careful, however, to allow for sufficient exploration of the search space by the EA first before the hillclimber is invoked.

While the general notion is that problems of mild epistasis are most suited for EAs [7], the results demonstrate that even highly epistatic problems, such as the QAP, can successfully be approached with pure EAs when one takes care

to design the search operator accordingly (preference for small changes).

At first glance, the empirical results also seem to indicate that pure, non-hybridized GA are not very successful for tough combinatorial problems. This may be so. One must remember, though, when assessing the performance especially of MMGA, that the design flexibility of EAs allows for many more variants of evolutionary approaches to be implemented than could possibly be tested here. Moreover, non of the three EA-mainstream techniques was originally invented to solve combinatorial optimization problems. Thus, I have freely adapted the various EA-types to suit the needs of this application. Nevertheless, for the QAP is is fair to conclude that crossover is not required for an EA to be successful.

A more detailed description of the experimental setup, results of sensitivity analysis as well as results for a messy-GA approach based on Goldberg et al. [12] is given in Nissen [19], together with other applications of EAs, and an in-depth presentation of all major EA-types, including hybrid-systems.

References

[1] Ablay, P.: Optimieren mit Evolutionsstrategien. Spektrum der Wissenschaft (1987) 7, 104-115.

[2] Bäck, T.; Schwefel, H.-P.: An Overview of Evolutionary Algorithms for Parameter Optimization. In: Evolutionary Computation 1 (1993) 1, 1-23.

[3] Brown, D.E.; Huntley, C.L.; Spillane, A.R.: A Parallel Genetic Heuristic for the Quadratic Assignment Problem, In: Schaffer, J.D. (ed.) Proceedings of the Third International Conference on Genetic Algorithms, San Mateo/CA: Morgan Kaufmann 1989, 406-415.

[4] Burkard, R.E.; Fincke, U.: The Asymptotic Probabilistic Behaviour of Quadratic Sum Assignment Problems. Zeitschrift für Operations Research 27 (1983), 73-81.

[5] Burkard, R.E.: Locations with Spatial Interactions: The Quadratic Assignment Problem. In: Mirchandani, P.B.; Francis, R.L. (eds.) Discrete Location Theory, New York: John Wiley & Sons 1990, 387-437.

[6] Burkard, R.E.; Karisch, S.; Rendl, F.: QAPLIB - A Quadratic Assignment Problem Library. EJOR 55 (1991), 115-119.

[7] Davidor, Y.: Epistasis Variance: Suitability of a Representation to Genetic Algorithms. In: Complex Systems 4 (1990), 369-383.

[8] Elshafei, A.N.: Hospital Layout as a Quadratic Assignment Problem. Operational Research Quarterly 28 (1977) 1 ii, 167-179.

[9] Fogel, D. B.: Evolving Artificial Intelligence. Dissertation, University of California, San Diego 1992.

[10] Goldberg, D. E.; Lingle Jr., R.: Alleles, Loci, and the Traveling Salesman Problem. In: Grefenstette, J.J. (ed.): Proceedings of an International Conference on Genetic Algorithms and Their Applications, Hillsdale/NJ: Lawrence Erlbaum 1985, 154-159.

[11] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Reading/MA: Addison-Wesley 1989.

[12] Goldberg, D.E.; Korb, B.; Deb, K.: Messy Genetic Algorithms: Motivation, Analysis, and First Results. TCGA Report 89003, University of Alabama, The Clearinghouse for Genetic Algorithms, Tuscaloosa 1989.

[13] Hoffmeister, F.; Bäck, T.: Genetic Algorithms and Evolution Strategies: Similarities and Differences. Technical Report SYS-1/92, University of Dortmund 1992.

[14] Koopmans, T.C.; Beckmann, M.J.: Assignment Problems and the Location of Economic Activities. Econometrica 25 (1957), 53-76.

[15] Kusiak, A.; Heragu, S.S.: The Facility Layout Problem. EJOR 29 (1987), 229-251.

[16] Lipsitch, M.: Adaptation on Rugged Landscapes Generated by Iterated Local Interactions of Neighboring Genes. In: Belew, R.K.; Booker, L.B. (eds.): Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo/CA: Morgan Kaufmann 1991, 128-135.

[17] Manderick, B.; Weger, M. de; Spiessens, P.: The Genetic Algorithm and the Structure of the Fitness Landscape. In: Belew, R.K.; Booker, L.B. (eds.): Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo/CA: Morgan Kaufmann 1991, 143-150.

[18] Mühlenbein, H.: Parallel Genetic Algorithms, Population Genetics and Combi-

natorial Optimization. In: Schaffer, J.D. (ed.) Proceedings of the Third International Conference on Genetic Algorithms, San Mateo/CA: Morgan Kaufmann 1989, 416-421.

[19] Nissen, V.: Evolutionäre Algorithmen. Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten. Wiesbaden: DUV 1994.

[20] Nissen, V.: Solving the Quadratic Assignment Problem with Clues from Nature. IEEE Transactions on Neural Networks: Special Issue on Evolutionary Programming 5 (1994) 1, 66-72.

[21] Nissen, V.; Krause, M.: Constraint Combinatorial Optimization with an Evolution Strategy. to appear in the Proceedings of 4. Dortmunder Fuzzy Tage (6.-8. June 1994).

[22] Nugent, E.N.; Vollmann, T.E.; Ruml, J.: An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. Operations Research 16 (1968), 150-173.

[23] Sahni, S.; Gonzales, T.: P-complete Approximation Problem. ACM Journal 23 (1976), 556-565:

[24] Schwefel, H.-P.: Numerical Optimization of Computer Models. Chichester: John Wiley & Sons 1981.

[25] Skorin-Kapov, J.: Tabu Search Applied to the Quadratic Assignment Problem. ORSA Journal on Computing 2 (1990) 1, 33-45.

[26] Steinberg, L.: The Backboard Wiring Problem. SIAM Review 3 (1961), 37-50.

[27] Taillard, E.: Robust TABU Search for the Quadratic Assignment Problem. Parallel Computing 17 (1991), 443-455.

[28] Vollmann, T.E.; Buffa, E.S.: The Facility Layout Problem in Perspective. Management Science 12 (1966) 10, B450-468.

Appendix A: Figures

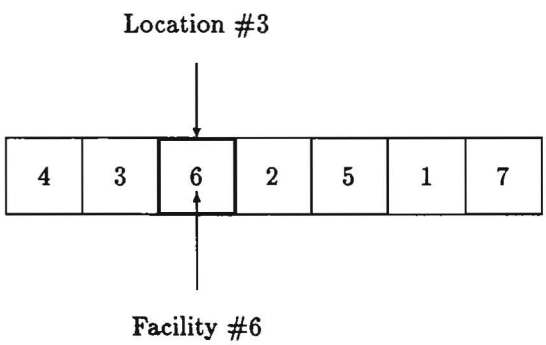


Figure 1: Solution representation in MMGA, CEP, and CES (n=7).

```
5 start
10 generate 200 subpopulations of 30 individuals each randomly
15 choose best solution
20 generationcounter = 1
25 repeat
30     subpopulationcounter = 1
35     repeat
40         if subpopulation not blocked, then
45             tournament selection of two non-identical parents
               (PMX-partners, max. 2 attempts)
50             if identical PMX-partners consecutively drawn twice,
               then block subpopulation, else
55                 PMX ⇒ 2 offspring
60                 replace worst individual of subpopulation by best offspring
65                 update best solution if required
70                 end
75         increment subpopulationcounter
80     until subpopulationcounter > 200
85     if all subpopulations blocked, then
90         exchange 750 randomly determined individuals between
           randomly determined subpopulations (emigration)
95         release all subpopulations again
100        reduce generationcounter by 1
105    increment generationcounter
110 until max. generation
115 output best solution
120 improve solution by 2-Opt (optional)
125 output improved solution (optional)
130 stop
```

Figure 2: MMGA pseudo code

```
5     start
10    generate and evaluate random population of five parents
15    population = parents
20    worstfit = worst objective function value of
      startpopulation (* used for mutation *)
25    best solution = best start-individual
30    repeat
35        copy each parent 5 times (* replication *)
40        determine number of 2-swaps for each child;
          mutate and evaluate children
45        hold seven tournaments for each of the 30 individuals
          against randomly determined opponents in the population
          and keep number of individual wins
50        sort population in descending order based on wins (* quicksort *)
55        choose five top individuals of this ranking as new parents
60        compare best individual based on objective function value
          from entire population (30) with best solution and update
          best solution if required
65    until max. generation
70    output best solution
75    improve solution by 2-Opt (optional)
80    output improved solution (optional)
85    stop
```

Figure 3: CEP pseudo code

```
5     start
10    generate and evaluate random starting solution
15    best solution = start solution
20    failurecounter = 0
25    repeat
30        save the objective function value of parent
35        copy parent to produce 100 children
40        determine number of 2-swaps from interval [1,2] for each child
          and mutate
45        evaluate children
50        choose best child based on objective function value to be
          new parent
55        if new parent not better than old parent then increment
          failurecounter
          else
              set failurecounter = 0
              if new parent better than best solution
              then update best solution
60        if failurecounter = round (n/10) + 2 then go to 1000
65    until max. generation
70    output best solution
75    improve solution with 2-Opt (optional)
80    output improved solution (optional)
85    stop

1000  procedure destabilization
1005  copy parent to produce 100 children
1010  determine increased number of 2-swaps from interval [3,8]
      for each child and mutate
1015  evaluate children
1020  choose best child based on objective function value to be
      new parent
1025  if new parent better than best solution
      then update best solution
1030  set failurecounter = 0
1035  end procedure
```

Figure 4: CES pseudo code

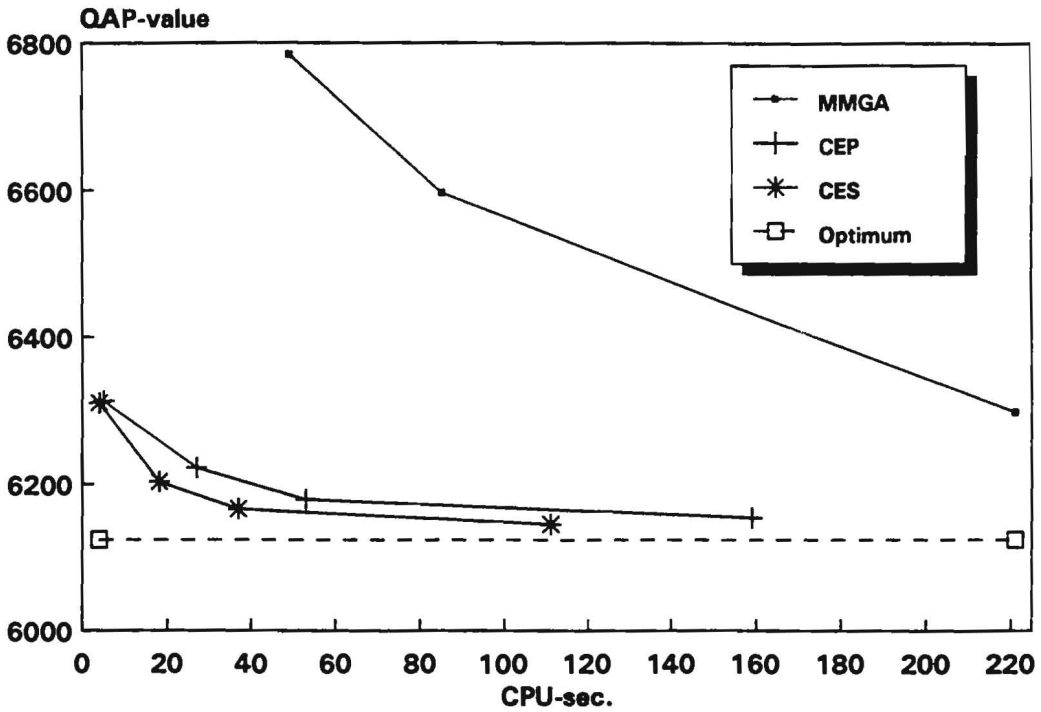


Figure 5: Efficiency graph for the testproblem NUG30.

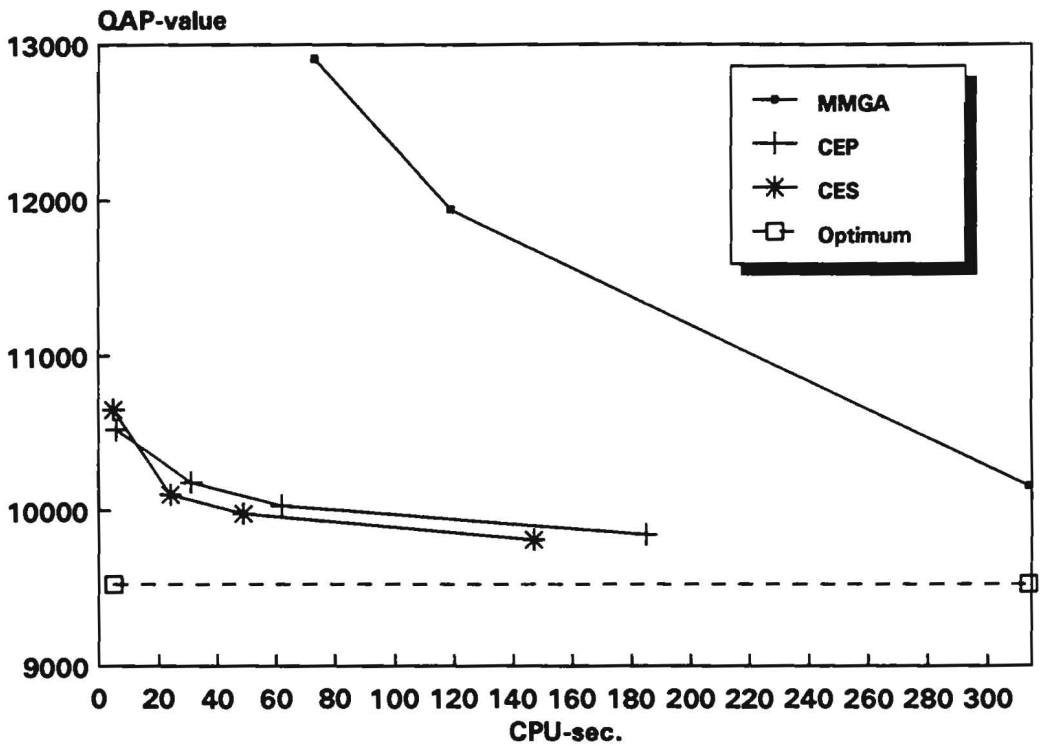


Figure 6: Efficiency graph for the testproblem STE36a.

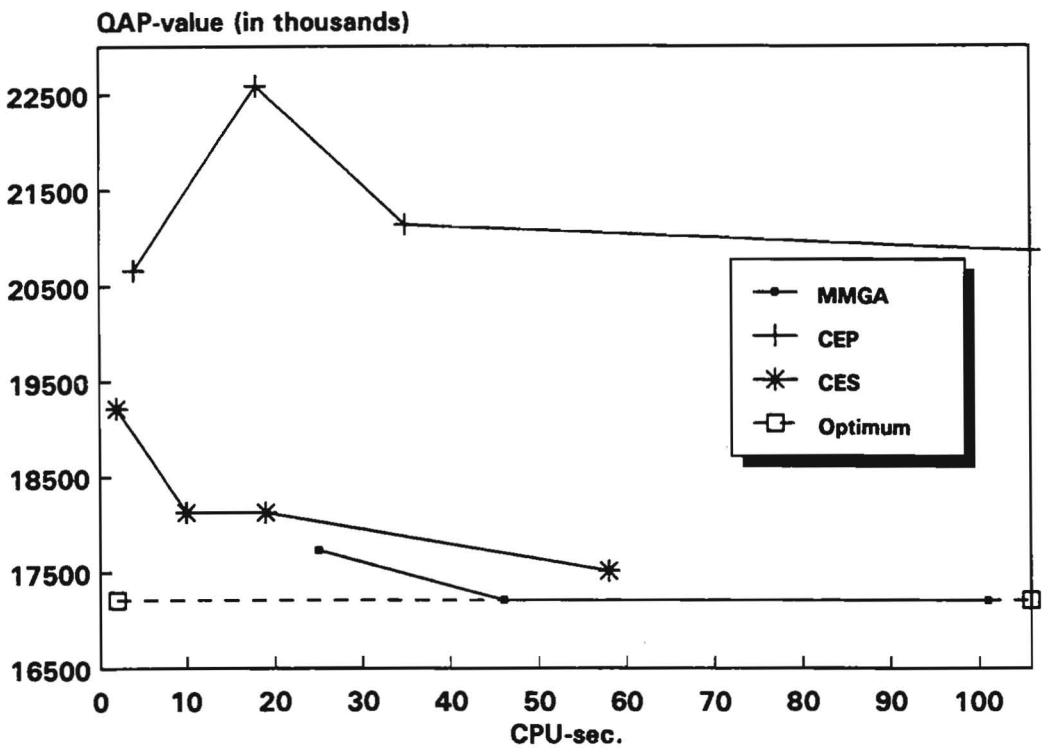


Figure 7: Efficiency graph for the testproblem ELS19.

Appendix B: Tables

Testproblem	Best known solution	Average objective function value* after i TS-iterations					
		$i = 4n$	AFE†	$i = n^2$	AFE‡	$i = 1000$	AFE◇
NUG15	1,150	1,162	6,300	1,152	23,625	1,150	105,000
NUG20	2,570	2,606	15,200	2,580	76,000	2,573	190,000
NUG30	6,124	6,228	52,200	6,148	391,500	6,142	435,000
SKO64	48,498	49,225	516,096	48,692	8,257,536	48,837	2,016,000
ELS19	17,212,548	21,154,221	12,996	19,174,778	61,731	17,780,562	171,000
STE36a	9,526	10,145	90,720	9,869	816,480	9,917	630,000

AFE = average function evaluations.
*The values are approximate. Taillard gives pars per mille above best known solutions.
† $4n \cdot n \cdot (n-1)/2$.
‡ $n^2 \cdot n \cdot (n-1)/2$.
◇ $1000 \cdot n \cdot (n-1)/2$.

Table 1: Numerical results of the TS-implementation by Taillard [27]. TS was run on a 10-node T800C-G20S transputer system. CPU-time per iteration is given by approx. $6,2 \cdot n^2 \mu$s.

Test-problem	Best known solution	Mean	Std.dev.	Ø-solution evaluations	Ø-CPU (sec.)
NUG15	1,150	1,194.4	24.69	745.7	0.06
NUG20	2,570	2,670.6	35.90	2,001.6	0.22
NUG30	6,124	6,321.8	66.93	10,321.9	2.19
SKO64	48,498	49,524.0	321.27	188,649.8	241.57
ELS19	17,212,548	21,798,726.0	2,571,051.42	3,593.8	0.36
STE36a	9,526	10,447.4	265.18	17,838.8	5.20
STE36c	8,239.1	8,902.6	318.18	20,188.4	5.89

Ø-solution evaluations = average number of solution evaluations.

Table 2: 2-Opt results (on IBM RS 6000/320 H).

		MMGA					+ 2-Opt	
Testpr./ b.k.sol.	Gene-ration	Mean	Std.dev.	Best gen.	Ø-sol.-eval.	Ø-CPU (sec.)	Mean	Add. CPU
NUG15/	0	1317.0*	16.35					
1150	500	1181.0	8.06	416.1	65933	16.49	1167.8	0.01
	1000	1154.0	4.47	804.4	111030	29.25	1154.0	0.01
	3000	1154.0	4.47	804.4	182117	99.92	1154.0	0.01
NUG20/	0	2969.2*	21.23					
2570	500	2713.0	20.90	439.1	67754	26.35	2648.4	0.06
	1000	2635.0	16.81	937.1	113419	46.09	2623.4	0.03
	3000	2608.0	22.91	1283.3	195976	145.16	2608.0	0.02
NUG30/	0	7343.2*	78.80					
6124	500	6784.8	52.38	378.1	68192	49.36	6297.0	1.12
	1000	6596.8	60.53	957.5	112037	84.64	6294.0	0.85
	3000	6297.8	67.42	2220.6	243544	220.54	6266.6	0.29
SKO64/	0	56235.0*	195.86					
48498	500	54033.8	189.69	405.5	74437	251.30	49385.0	212.01
	1000	53411.0	178.22	905.5	115277	392.46	49637.6	186.20
	3000	50768.6	409.58	2893.7	307058	1092.36	49556.2	96.83
	10000	50562.0	455.36	3633.5	555534	2506.98	49554.2	101.43
ELS19/	0	26373875.2*	1502325.23					
17212548	500	17735390.4	279257.96	457.9	68992	24.96	17384514.2	0.07
	1000	17215921.4	6648.02	916.9	121786	46.22	17212548.0	0.02
	3000	17212548.0	0.00	944.9	251729	101.12	17212548.0	0.02
STE36a/	0	16805.0*	217.72					
9526	500	12907.6	365.34	475.2	72280	72.73	10423.8	2.97
	1000	11940.4	324.72	941.2	114427	119.39	10182.4	2.58
	3000	10158.6	286.58	2628.4	286285	313.68	10042.0	0.42
STE36c/	0	13826.3*	125.23					
8239.1	500	10878.1	194.80	478.5	71456	77.20	8672.6	3.21
	1000	10111.6	120.06	920.1	113693	127.45	8642.3	3.07
	3000	8697.9	180.04	2664.6	290367	340.89	8530.0	0.76

*Datum refers to the best start-individual in each of the ten runs.
Testpr. / b.k.sol. = Testproblem / best known solution.
Best gen. = average generation of the best solution found.
Ø-sol.eval. = average number of solution evaluations.
Add. CPU (sec.) = average additional CPU requirements for 2-Opt.

Table 3: MMGA results (on IBM RS 6000/320 H).

CEP							+ 2-Opt	
Testpr./ b.k.sol.	Gene-ration	Mean	Std.dev.	Best gen.	Ø-sol.-eval.	Ø-CPU (sec.)	Mean	Add. CPU
NUG15/ 1150	0	1497.2*	53.96					
	500	1160.8	10.81	219.8	12505	3.14	1160.8	0.01
	2500	1155.2	3.92	788.9	62505	15.67	1155.2	0.01
	5000	1154.2	3.84	1281.1	125005	31.35	1154.2	0.01
	15000	1151.2	0.98	3688.1	375005	93.94	1151.2	0.01
	50000	1151.0	1.00	5886.2	1250005	313.14	1151.0	0.01
NUG20/ 2570	0	3241.0*	99.83					
	500	2608.2	26.75	356.3	12505	3.79	2607.8	0.02
	2500	2590.6	15.52	1423.3	62505	18.88	2590.2	0.02
	5000	2582.4	9.79	1834.0	125005	37.68	2582.4	0.02
	15000	2579.8	10.06	2761.7	375005	112.98	2579.8	0.02
	50000	2571.8	5.40	10699.7	1250005	376.47	2571.8	0.02
NUG30/ 6124	0	7898.2*	124.93					
	500	6312.8	48.64	406.7	12505	5.38	6280.0	0.26
	2500	6221.6	42.93	2061.0	62505	26.58	6215.2	0.15
	5000	6178.6	18.97	4102.9	125005	53.07	6168.6	0.14
	15000	6153.6	14.14	10784.4	375005	159.01	6149.0	0.14
	50000	6142.8	11.32	34062.7	1250005	529.59	6139.6	0.11
SKO64/ 48498	0	58052.2*	305.84					
	500	50385.2	302.67	477.6	12505	20.83	49581.8	52.52
	2500	49681.6	189.36	2193.4	62505	103.06	49189.4	44.30
	5000	49374.4	152.40	3481.9	125005	205.12	49016.2	29.01
	15000	49157.2	96.59	12184.8	375005	614.28	48920.6	27.60
	50000	49062.6	82.64	35895.3	1250005	2056.00†	48820.6	26.20†
	250000	48912.6	34.42	196225.8	6250005	10300.00†	48687.0	24.30†
ELS19/ 17212548	0	48655026.0*	5080681.91					
	500	20668140.2	2350732.40	245.9	12505	3.55	20668140.2	0.02
	2500	22587649.8	2688617.62	384.5	62505	17.63	22587649.8	0.02
	5000	21142317.0	1928873.91	546.7	125005	35.24	21142317.0	0.02
	15000	20871435.2	2437603.24	1601.2	375005	105.67	20871435.2	0.02
	50000	20375940.4	1882263.08	5714.2	1250005	349.22	20375940.4	0.02
STE36a/ 9526	0	19825.6*	560.74					
	500	10522.0	276.68	403.9	12505	6.25	10351.8	0.65
	2500	10183.2	259.50	1782.2	62505	30.96	10119.2	0.41
	5000	10031.6	243.80	2357.9	125005	61.81	9992.6	0.39
	15000	9845.4	141.48	11728.9	375005	185.19	9825.8	0.29
	50000	9923.0	106.98	23667.8	1250005	617.58	9614.4	0.27
STE36c/ 8239.1	0	16281.4*	464.31					
	500	8807.1	144.06	396.8	12505	6.30	8720.5	0.62
	2500	8553.3	102.28	1555.0	62505	31.19	8506.3	0.39
	5000	8544.5	264.34	3413.4	125005	62.30	8513.7	0.38
	15000	8443.0	97.70	7640.5	375005	186.60	8425.4	0.29
	50000	8327.4	81.90	34411.4	1250005	621.47	8310.9	0.26

*Datum refers to the best start-individual in each of the ten runs.
†Appox. values.
Testpr. / b.k.sol. = Testproblem / best known solution.
Best gen. = average generation of the best solution found.
Ø-sol.eval. = average number of solution evaluations.
Add. CPU (sec.) = average additional CPU requirements for 2-Opt.

Table 4: CEP results (on IBM RS 6000/320 H).

	CES						+ 2-Opt	
Testpr./ b.k.sol.	Gene-ration	Mean	Std.dev.	Best gen.	Ø-sol.-eval.	Ø-CPU (sec.)	Mean	Add. CPU
NUG15/	0	1564.2	79.80					
1150	200	1162.0	10.24	90.2	10811	1.46	1160.0	0.01
	1000	1153.2	3.49	415.7	54446	7.36	1153.2	0.01
	2000	1150.8	0.98	712.1	108976	14.73	1150.8	0.01
	6000	1150.2	0.60	1583.2	327086	44.23	1150.2	0.01
	20000	1150.0	0.00	2225.0	1090281	147.26	1150.0	0.01
NUG20/	0	3442.0	73.10					
2570	200	2626.4	23.78	89.5	10641	2.10	2623.6	0.03
	1000	2591.8	13.22	607.5	53276	10.47	2591.4	0.02
	2000	2586.8	13.48	1014.7	106626	20.96	2584.2	0.03
	6000	2574.0	6.20	2311.0	320096	62.93	2572.8	0.02
	20000	2570.0	0.00	5447.8	1067451	209.63	2570.0	0.02
NUG30/	0	8127.4	182.19					
6124	100	6310.2	66.92	73.8	10291	3.62	6283.8	0.18
	500	6202.4	46.38	317.2	52281	18.49	6195.4	0.11
	1000	6165.6	23.13	651.8	104701	37.03	6158.8	0.13
	3000	6144.8	10.09	1789.2	314691	111.39	6144.4	0.10
	10000	6135.0	9.31	4680.6	1050181	372.35	6134.8	0.09
SKO64/	0	58947.2	737.28					
48498	100	50195.6	284.02	95.7	10001	19.08	49607.6	42.09
	500	49565.4	277.49	458.9	50271	95.85	49332.0	25.20
	1000	49379.6	181.38	761.0	100781	192.30	49193.0	21.40
	3000	49044.2	123.12	2344.0	302591	583.12	48924.8	17.56
	10000	48906.0	80.61	6423.0	1009211	1927.53	48804.2	13.42
	50000	48815.2	52.04	31158.5	5045471	9640.00†	48697.6	16.60†
ELS19/	0	59725819.2	9244110.12					
17212548	200	19218336.8	2094638.95	136.7	10486	1.92	19209755.4	0.02
	1000	18128394.0	1398977.87	451.5	52916	9.71	18128394.0	0.02
	2000	18128394.0	1398977.87	451.5	105851	19.41	18128394.0	0.02
	6000	17517830.0	915846.00	1127.5	317621	58.21	17517830.0	0.02
	20000	17517830.0	915846.00	1127.5	1058841	193.87	17517830.0	0.02
STE36a/	0	22754.8	1930.78					
9526	100	10650.0	203.66	72.2	10171	4.79	10548.0	0.54
	500	10103.0	110.50	402.3	51461	24.37	10079.2	0.29
	1000	9979.0	111.46	801.0	102981	48.78	9944.6	0.29
	3000	9811.8	103.26	1879.5	309561	146.80	9797.2	0.26
	10000	9701.2	87.92	6628.2	1032511	489.23	9689.8	0.21
STE36c/	0	18890.2	1804.01					
8239.1	100	8923.2	175.76	82.6	10131	4.79	8859.4	0.57
	500	8713.1	187.60	325.5	51301	24.43	8698.5	0.25
	1000	8581.9	165.04	755.3	102901	49.07	8568.3	0.30
	3000	8402.9	93.40	1638.7	309141	147.59	8392.7	0.23
	10000	8337.7	85.76	7268.3	1031451	492.26	8332.0	0.26

†Appox. values.
Testpr. / b.k.sol. = Testproblem / best known solution.
Best gen. = average generation of the best solution found.
Ø-sol.eval. = average number of solution evaluations.
Add. CPU (sec.) = average additional CPU requirements for 2-Opt.

Table 5: CES results (on IBM RS 6000/320 H).

Average objective functions values for different selection pressure			
Gen.	$w = 5$	(Standard) $w = 7$	$w = 10$
500	6351.4	6312.8	6294.0
1000	6343.0	6234.2	6265.2
2500	6276.8	6221.6	6249.6
5000	6253.8	6178.6	6262.2
15000	6237.4	6153.6	6182.2
50000	6210.8	6142.8	6186.4

Gen. = Generation
w = number of individual competitions in the stochastic selection

Table 6: Varying the selection pressure for CEP on NUG30.

10 × CES (10,000 generations)				1700 × 2-Opt			
Best	Worst	Mean	Std. dev.	Best	Worst	Mean	Std. dev.
6124	6150	6135.0	9.31	6128	6702	6351.0	83.94

Table 7: Comparison of CES and 2-Opt solution quality at approx. identical CPU-requirements (IBM RS 6000/320 H) on NUG30. All starting solutions are randomly generated.

CONSTRUCTIVIST ARTIFICIAL LIFE: The constructivist-anticipatory principle and functional coupling

Alexander Riegler

Department of Theoretical Biology
Althanstrasse 14
A–1090 Vienna, Austria

and

Department of Software Technology
Resselgasse 3/2/188
A–1040 Vienna, Austria

Email: riegler@eimoni.tuwien.ac.at
Fax: ++43 1 31336 700

Abstract

Both the system theory of evolution and the epistemology of radical constructivism provide fertile inspiration for enhancements of artificial life. Within this paper I will demonstrate that (a) one can move the emphasis on sensory information processing to a more expectation-driven algorithm; and (b) that a separation between the operational closed brain on the one hand and sensors and motor elements on the other hand will enable the study of cognitive mechanisms independent of the actual environment.

1. Introduction

1.1 Constructivist Artificial Life (*CLife*)

Constructivist artificial life is an enhancement of artificial life models with mechanisms provided by (radical/cognitive) constructivism (sections 2.3 and 2.4) and the system theory of evolution (section 2.2). CLife employs expectation-driven behavior, hierarchical structure of fuzzy representational elements, and a separation between cognitive processes within an agent and its sensor and effector surfaces.

1.2 Active Perception

According the constructivist point of view, autonomous agents have their own hypotheses about the world which do not necessarily correspond directly with physical events. This concept reverses the traditional bottleneck architecture of perception: No longer is the entire available 'information from outside' used to control the behavior of a agent. Instead, agents are viewed as autonomous entities which construct their own 'reality'[1]. This *constructivist-anticipatory principle* (cf. section 3.4) is implemented by chaining and ramifying fuzzy interval schemata. Once a hypothesis is selected, the algorithm needs to test only a few perceptual events when the condition parts of the rules require it. This reduces the computational need for performance.

1.3 Separation between cognitive apparatus and physiology

A priori assumptions and anthropocentric ascriptions (e.g., what is food, what is a predator, etc.) are avoided. CLife strictly separates the cognitive apparatus from the physiology of sensors and motors (see Fig. 1 and section 4). This implements the *operational closure of the brain* (as described by radical constructivism). Due to this modular structure the cognitive apparatus can easily be exchanged for different experiments and tasks.

1.4 Hierarchical Representation

System theory of evolution provides the functional couplings within representational units. Such adaptive predispositions lead to a *canalization of development*, where building blocks of representational units and their hierarchical relationship increase the speed of evolution by orders of magnitudes. I will

1. As argued in [12], the notion of reality may be differentiated into: Realität, which connotes the ontologically given environment every realist makes reference to, and Wirklichkeit, which designates the "constructed" world in our minds, as the constructivist position proposes. Thus, Wirklichkeit connotes a sequence of "effects" (perturbations) which may appear at any time and place. The reference elements for "knowledge acquisition", which we term "phenomena" or "facts", are therefore spatially and temporally constrained configurations of effects.

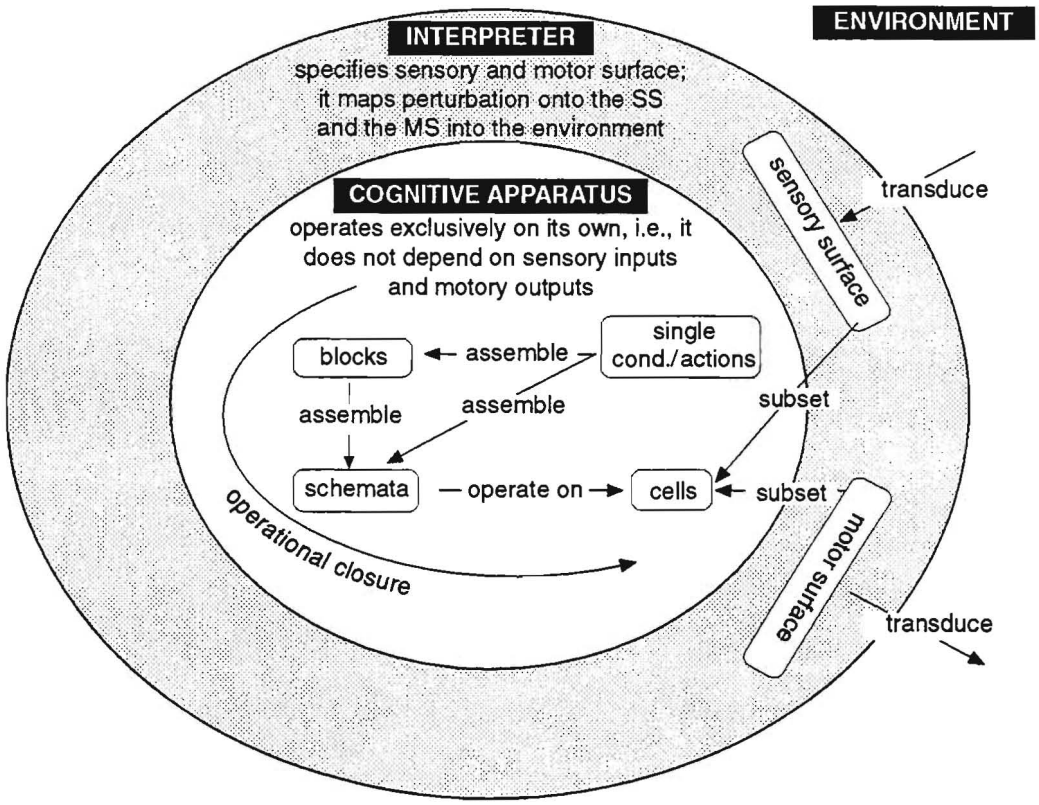


Figure 1: Operational closure within the cognitive apparatus. SS denotes the sensory surface, MS is the motor surface.

point out that the drawbacks of interdependent and hierarchical systems (*genetic loads*), as we know from biological evolution, can be avoided within artificial systems (for an overview see section 3.5).

2. Theoretical Background

2.1 Building Blocks of Behavior in Biology

Konrad Lorenz [6] used the notion of *fixed action patterns* to connote the basic building blocks of behavior. He characterized them as rather stereotypic patterns. Many flight behaviors are of this kind of behavior. The egg-retrieving behavior of the greylag goose provides an example for a slightly improved version of a fixed action behavior: If an egg falls out of the nest the incubating goose rolls the egg back into the nest with its bill. If the egg is removed midway through this action pattern the bird continues until its bill has reached the border of the nest. It seems that during this process the goose neglects environmental events: if an action pattern has been triggered, the processing of sensor information is reduced until the pattern terminates. During egg retrieving it is necessary to compensate for any sideward rolling of the egg. This behavior is called a *behavioral sequence* and consists of the interaction between effectors and sensor feedback. We can compare this behavior with walking in a dark tunnel: whenever we bump against the wall we slightly change the direction to suit the circumstances. In CLife, such action chains are easily implemented: they are just a sequence of procedures whose execution is ultimately stopped by a criterion to be satisfied. In biology, such a criterion is called a *consummatory act* or, if it is merely a state, an end situation.

In most cases fixed action patterns are triggered by stimuli from the environment, but they are independent from these stimuli during their execution[2]. Such *sign stimuli* correspond to conditions in CLife. A group of conditions, which I refer to as concepts in CLife, can be identified as *sign situations*, i.e., a certain configuration of sign stimuli.

The distinction between phylogenetic and ontogenetic elements in CLife mirrors the difference between an *innate releasing mechanism* (IRM) and an *acquired releasing mechanism* (ARM) in bio-

2. Since this definition does not apply to all fixed patterns of action, the notion of *modal action pattern* was introduced. It connotes a specific characteristic, spatially and temporally ordered pattern of behavior, which can be identified by its typical proceeding.

logical systems. Releasing mechanisms can be characterized as neurosensory filter mechanisms related to certain patterns of behavior. They specify the response to sign stimuli by selecting out ineffective stimuli. In CLife, this filtering is turned bottom-up in that individuals do not filter *out* irrelevant information. Rather, they actively test the conditions within their schemata, which decreases computational costs for computing all perturbations from the environment.

2.2 The System Theory of Evolution

According to the system theory of evolution, living beings are hierarchically organized and may therefore be characterized by super- and sub-systems. Any explanation is incomplete as long as only a few elements of organization levels are examined [15].

The mechanisms explained within the system theory of evolution shed a new light on evolutionary processes which are not in a line with traditional Darwinistic ideas about evolution: from a system theoretical point of view, the framework (*Gefüge*, i.e., the construction and functional conditions) of an organism itself is already defined as conditions of *internal selections*.

The relation between system-internal and external selection is comparable to the relationship between selection within an industrial plant and selection on the market: In a plant there are issues such as standardization and management that increase productivity:

> [I]t would be disastrous for a company... to have to rely only upon its customers to find out whether the engine was properly put into a car or whether the cylinders are equal in size. [11]

2.2.1 *Order Principles*

In his morphological analysis of living systems, Riedl [11] outlined the order behind these systems. He noted that the evolution of organisms is canalized and attempted to answer several open questions that could not be answered with the traditional synthetical theory of evolution. We must distinguish four basic order principles of living systems which are responsible for the internal selection in organisms.

- **Norms** connote the uniformity of structures. Therefore, in CLife simple uniform elements (conditions and actions) are used.
- **Interdependence** of norm elements.
- Dependent structures are mutually subordinated to form ranks, grades, or classes, i.e., a **hierarchical** organization. In CLife, uniform elements may be arranged to form blocks of higher order.
- **Tradition** adds a time scale to the previous principles. No organic state exists without tribute to its ancestry, so that all building states are subsequent series of coordinations.

2.2.2 *Functional Coupling*

An increase of evolutionary speed can be achieved by *functional coupling*, a mechanism that obeys the principle of interdependence and hierarchy. If we examine the evolution of joints we recognize that the joint socket and the condyle must evolve in the same direction and to the same degree in order to maintain functionality[3]. Functional couplings have the advantage of dramatically increasing the chances of adaptation by several orders of magnitude. In biological systems, clustering of independent genomes is irreversible, yielding constraints and canalizations which render adaptations to changing environments more difficult. Such *genetic loads* [11] are the reason why a giraffe has the same number of neck bones as a dolphin although it could use more in order to increase its pliancy. CLife has to answer the question: Can artificial systems gain the advantages of functional coupling without inheriting the disadvantage of genetic loads?

2.3 Epistemological Constructivism

Epistemological Constructivism (as formulated by Heinz von Foerster [3] and Ernst von Glasersfeld [4][4]) primarily asks for what we know about the world. The main point is the concept of the observer, i.e., starting with the assertion that observing is the only access to the 'world'. This is based on the fact that an observer is a so-called operationally closed system, wherein nervous signals are unspecified, e.g., visual stimuli affect the same kind of internal signals as tactile ones[5]. Following an example of Maturana and Varela [8], the situation is analogous to a navigator in a submarine: He relys on the readings from his instruments when he operates the levers and buttons. The instruments do sense "something" outside the submarine, but this fact is completely irrelevant to the navigator. His only task is to maintain certain relationships between the

3. In the terminology of Genetic Algorithms this would be called context preserving.
4. For a more detailed description see [12].
5. Since observing—in the sense of having experiences—is a coherent coordination of actions in a community of observers, Radical Constructivism is *not* a solipsistic philosophy.

indicators constant, independent of what this indicators measure and what effect the buttons and wheels have. Within CLife, the cognitive apparatus does not know anything about an environment. It merely operates on cells whose "semantics" (for an observer of CLife) are defined within the interpretative shell around the cognitive apparatus. This shell maps perturbations from the environment onto the cells, but this is irrelevant for the operation of the cognitive apparatus.

Furthermore, Maturana and Varela argue that the ratio sensors:brain:motors in human beings are $10:10^6:1$ (the "Heinz von Foerster ratio"). This does not imply that sensors and effectors are unimportant, but emphasizes that behavior is not just a set of pure taxes and reflexes. Sensors only perturb the brain but do not determine behavior. The way schemata are processed in CLife employs this *constructivist–anticipatory principle*: Sequences of conditions and concepts may only test the state of internal cells in order to generate action patterns. During the execution of action patterns the state of single perceptive cells may be tested in order to direct or to halt the action patterns.

2.4 Cognitive Constructivism

Cognitive Constructivism (e.g., Jean Piaget [10]) emphasizes the cognitive development of beings, especially human beings. Its starting point is a psychological one: the to some extent mentally 'naked' child[6]. Hence, cognition must not be perceived as a static ability but rather as a dynamic process that has its origin in the sensorimotor stage of early childhood.

In the sensorimotor stage, within which cognition is linked to the content of specific sensory inputs or motoric actions, the key question is how cognitive creatures obtain 'symbolic ideas' about their world. For the coordination of cognitive schemata, increasing *intermodal* coordination is very important—e.g., grasping implies the concurrent use of visual and tactile sensors. Working with intermodal experience, the infant constructs the idea of invariant, permanent objects. Generally, there are 3 levels of coordination: (1) In the *monomodal* mode, cognition is guided mainly by a single modality, as is the case within most Artificial Intelligence systems; (2) *Multimodality* connotes the sequential non–overlapping usage of different modalities, as shown in animals up to and including reptiles (see example below); but only (3) *intermodality* enables the interaction of different modalities we find in animals from the mammal level upwards.

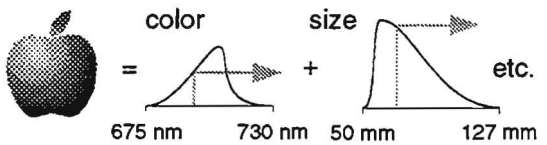


Figure 2: The fuzzy "concept" of apple. The differences in the height of the Gaussian curves mirror the importance of each curve. In this example, size contributes more than color to the concept.

In CLife, the notion of *concept* mirrors the intermodal characteristics of things. See Fig. 2 for an illustration: In order to classify an object as a member of the concept "apple", it is necessary to test whether its size, its color, and so on are within a certain range.

3. The Cognitive Apparatus

This component is often interpreted as the connection[7] between sensors and effectors. The constructivist aspect of CLife, however, requires separating the cognitive apparatus from the physiology of the senses (see Fig.1): The nervous system in biological systems is operationally closed. It does not distinguish between perturbations from outside and internal perturbations, since nervous signals are unspecified with regard to their origin. As shown by Maturana et al. [9], there is no correlation between physical stimulus on the outside of the agent and the subjectively experienced perception. Indeed, signals from the perceptual surface are only triggers to the cognitive apparatus. The nervous system is organized as a closed network of interacting neurons within which each state of relative neuronal activity leads to another state of relative neuronal activity [7].

In CLife, neurons are substituted with *schemata* which include elements from fuzzy logic. From their point of view, they do not deal with information from the environment either, but operate exclusively on a set of cells[8] which are all equivalent to

6. This does not imply the child's cognition to be a *tabula rasa* when it is born. Of course, the child inherits the phylogeny of its ancestors.

7. As shown by Valentin Braitenberg [2], such connections need not be complicated in order to exhibit complex emergent behavioral patterns. Simple connections between two sensors and two locomotion elements exhibit forms of attractive behavior, such as photo- and chemotaxis.

8. The notion of "cell" has no biological implications. A cell is merely a storage of a number with a unique address.

them. The advantages over neural networks, which seem to be biologically more plausible, are evident: It is much easier to track the chain of rules (*schemata*) in order to *explain why* a particular behavioral pattern has ultimately emerged. In neural nets, one can only observe patterns of activity that lead to some behavior. Each rule uses a more sophisticated computation than a single neuron in a net. In combination with the constructivist–anticipatory methodology (see section 3.4), there is no strict demand for parallelism. Finally, rules may serve as better vehicles for some kind of Piagetian schema mechanism (cf. section 2.4), as their data structures are quite similar to schemes.

Due to its operational closure of the cognitive apparatus need not (and indeed does not) assume any outside world in order to function. It consists of a number of uniform cells which are consequently numbered so that they can be addressed by manipulating mechanisms. These mechanisms are enhanced schemata which consist of a condition and an action component. Each of these components may be constituted (a) by single query elements or action elements, and (b) by groups of such single elements. I will refer to the groups of single queries as *concepts*.

3.1 Conditions and concepts

In section 2, I pointed out the reasons to use concepts and gave an illustration (cf. Fig. 2). Generally, the appearance of things are somewhat fuzzy in that an object that is very light-red and measures five centimeters is less likely to be an apple than another object with glaring red color and a height of eight centimeters. Due to this fact, I chose a modified kind of schema[9] that fulfills this requirement. The basic element of the condition part of schemata are *fuzzy interval conditions* (see Fig. 3 for a simplified version): The value to be tested is matched against an interval which is defined as a Gaussian curve with left and right boundaries and an optimal peak. If the value is exactly the optimum, then the query answers with the optimal response. Any deviation from the optimum will lead to lower response until the value is outside the interval. In this case, the answer is zero. It should be emphasized that within the cognitive apparatus the value of cells has no special meaning. The set of cells is just a set of variables, each of which is set to a certain number.

Each condition has a unique number and is stored in either the phylogenetic or ontogenetic

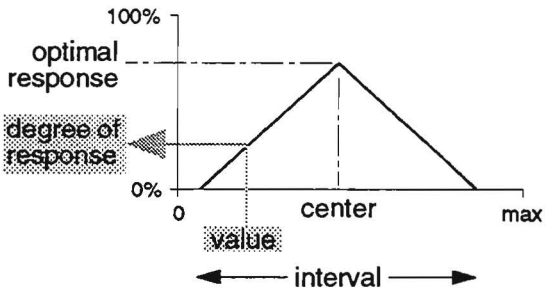


Figure 3: A single query is a fuzzy interval. Due to performance reasons, a triangular approximation with left = right is used instead of a Gaussian curve as described in the text.

condition library. The *phylogenetic condition library* (pcLib) represents the common knowledge which has accumulated during evolution[10]. Theoretically, every organism can access this library and use its entries as building blocks for its behavior. The pcLib is created and enlarged by phylogenetic mechanisms, i.e., mutation and crossover. The *ontogenetic condition libraries* (ocLib) reflect the individual experiences and are therefore private to each individual.

At the second level of hierarchy, conditions may be grouped to form condition blocks or *concepts* as described above. Each concept consists of indices that refer to entries of the libraries at the first level of hierarchy. This allows the usage of a condition in several concepts without multiplying the condition[11]. As for conditions, there are phylogenetic and ontogenetic concepts. While phylogenetic concepts exclusively refer to phylogenetic conditions, ontogenetic condition blocks may combine both kinds of conditions. This is due to the fact that every experience may build upon inherited elements. Both kinds of concepts are collected within libraries: The *phylogenetic condition block libraries* (pCBLib) collect all phylogenetic concepts. Analogously, the same holds true for the *ontogenetic condition block library* (oCBLib).

From the system theoretical point of view, the introduction of condition blocks implements a functional coupling in that the activation of a single block in turn activates several action elements that are associated with this block.

9. See [13] for a more detailed description of the fuzzy interval schema.

10. Philosophically, we may refer to it as the *Kantian aprioris*.

11. This is similar to the usage of calling–by–reference in higher programming languages: The subprocedure only receives a pointer to the variable instead of the variable itself.

3.2 Actions and Action Patterns

Action patterns are groups of single actions. For instance, an escape behavior may include "turn around!" and "start running!" as single actions. Due to its operational closure the cognitive apparatus does not know anything about turning around or running[12]. It merely manipulates the values of the cells (all cells being viewed as equal). Indeed, there are only four kinds of action an action element can perform (see Fig. 4):

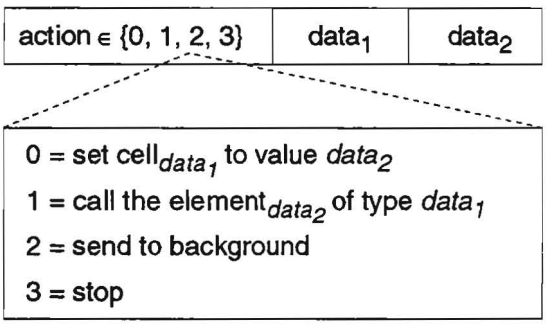


Figure 4: A single action element which can (0) manipulate the value of a cell, or (1) establish a hierarchical organization by calling another structure element, (2) initiate background tasks, or (3) stop the execution of a action sequence (both in foreground and background).

- SET a certain cell to a certain value;
- CALL another structural element (such as another action element or action pattern) as a subroutine, i.e., after the execution of the subroutine has ended, the execution of the calling action pattern continues;
- send the execution of the current action sequence into BACKground. This feature enables the simultaneous execution of several tasks. That is, the agent may look for food (waiting for a certain cell to be set within a certain range) while walking around (background task).
- STOP the execution of the action pattern, which also causes the stoppage of the execution of any other patterns that have called the current pattern as subroutine. This action is also the only possibility to stop a background action sequence.

Note that only the first instruction is effective in the sense that it manipulates something. The background operator enables parallelism within a CLife model. The call and stop operators are only provided to establish a recursive and hierarchical relationship between action patterns and other structural elements.

As in the case with conditions and concepts, action elements are also stored in libraries. The *phylogenetic action library* (pALib) contains the evolutionarily created actions, while the *ontogenetic action libraries* (oALib) consist of individual experiences. At the second level of hierarchy, *phylogenetic* and *ontogenetic action block libraries* (pABLib, oABLib) collect all groups of action elements by referring to the entries of libraries at the first level. Again, this implements a functional coupling in that the activation of a single block in turn activates several action elements.

3.3 Schemata

The actual representational elements are schemata. Schemata have a condition and an action part. Each consists of reference pointers to libraries either on the grouped element level or on the single element level. Individuals phylogenetically inherit schemata in form of indices which point to the phylogenetic schema library, i.e., the innate knowledge. Genetic operators change the indices during the simulated evolution.

3.4 Constructivist-Anticipatory Schema-Processing

The constructivist–anticipatory schema-processing algorithm (*CASP*) works as follows. The behavior is controlled by schemata which, once invoked, ask for sensory or internal data *only* when it becomes necessary [14]. In other words, the algorithm neglects environmental events *except* the current action pattern demands to check a certain cell's value (which, from the viewpoint of the interpreter shell, may be a sensor or an internal value). This leads to a significant decrease in performance costs since the simulation algorithm need not provide the full environmental information to the agent at every time step[13]. The outline of the algorithm—which performs the process of assimilation in the Piagetian sense—is as follows:

- Polling, i.e., pattern matching triggers a first schema. Pattern matching is successful when the conditions of a schema, i.e., the sum of the

12. Attributing a meaning to the cells takes place in the interpretative shell (see section 4).

13. Of course, this type of processing does not render feature extraction obsolete (especially in a more complex structured environment than in the example environment of section 5.2). Rather than computing intensive reduction of complexity at every time step, feature extraction is only necessary to trigger a hypothesis or if a condition asks for a particular feature.

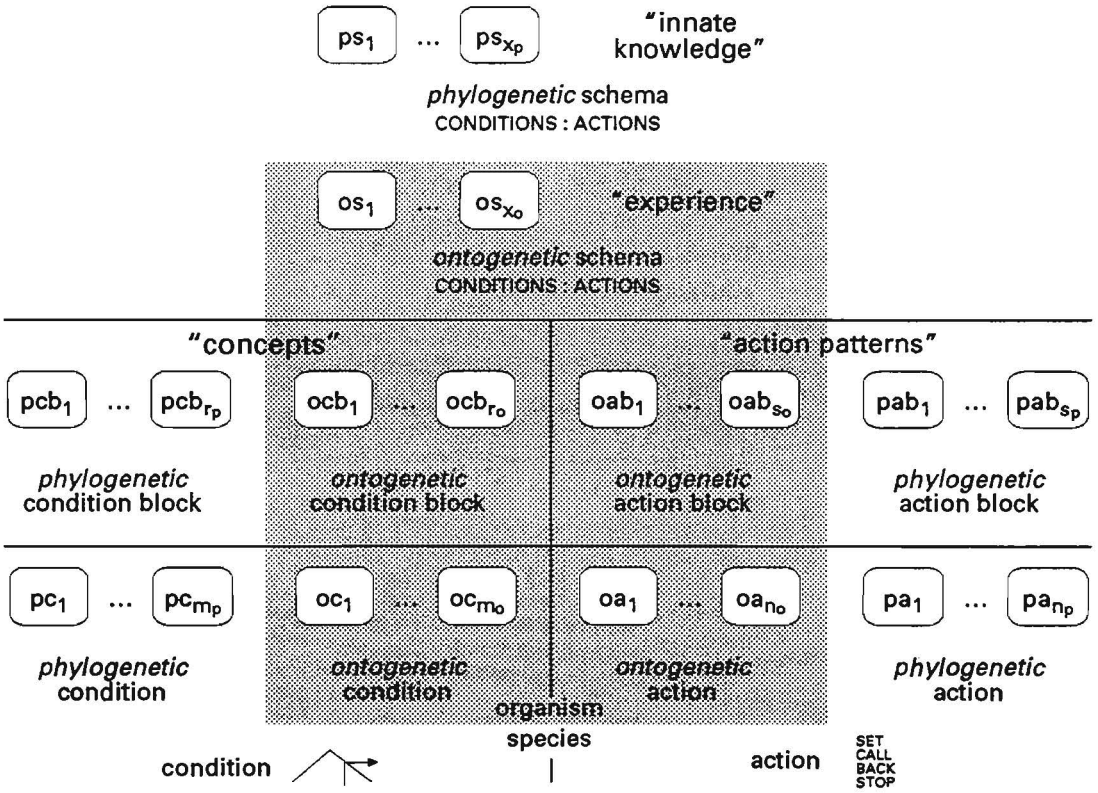


Figure 5: The hierarchical organization of CLife. On each level, all elements of the same sort are summarized in a library so that elements on the next level only need to refer to the library index of elements they consist of: Ontogenetic blocks may consist of both phylo- and ontogenetic elements, while phylogenetic blocks are built up only by phylogenetic components. The condition and action part of a schema may contain both single conditions/actions and condition/action blocks. Note that in addition to this diagram there are also phylogenetic reflexes that implement a kind of interrupt.

fuzzy matching results of each condition in the schema, are satisfied to a certain percental degree, i.e., the global variable *threshold*.

- Subsequently, the action part is executed; it includes setting cells, calling other elements (both actions and conditions, as well as blocks and schemata, thus forming an action sequence), sending the current action sequence to background, or stopping the execution. If a condition is required to be fulfilled during the execution of an action pattern, this checks if the pattern is still on the right track.
- After a foreground action sequence has terminated the algorithm again tries to trigger another schema.

3.4.1 Interrupt Handling

So far, constructivist-anticipatory schema-processing implements an exclusively expectation-driven behavior. But how does an agent respond to unexpected stimuli, e.g., a suddenly appearing situation which threatens the agent's life? For this purpose, we need some sort of interrupt which stops the execution of any action pattern immediately and starts a short action instead, e.g., a jump action to escape the dangerous situation.

To attain these requirements the CASP algorithm polls a small set of phylogenetic single conditions each time cycle. Each of these conditions corresponds to a situation of highest priority that requires an immediate response. Each condition is associated with an action which is an appropriate response to the given situation. This implements a kind of interrupt line. For example, a dangerous obstacle appears in front of the agent and causes the interpreter shell to change the value of a certain cell. The interrupt element that asks for the value of that specific cell now becomes activated and changes the value of another cell. The content of the second cell is translated by the interpreter shell as a backward jump the agent has to perform in order to escape the obstacle. After this interrupt (or *reflex* in biological terms) the CASP algorithm again polls possible interrupts (e.g., the obstacle may still be in

front of the agent). If no interrupt is triggered the algorithm polls normal schemata.

3.4.2 *Types of behavior*

We may now distinguish between 3 types of behaviors that mirror the biological equivalents as described in section 2.1. The more complex a behavioral type is the more it contributes to higher cognitive abilities of the agent. On the other hand, simple types are much more rapidly executed and have higher priority since they have the function of preserving the agent's life.

- *Interrupt elements* (or simple reflexes) consists of a single condition and a single action. As they have the highest priority and their activation has precedence over all other types, they implement interrupt lines in the otherwise completely expectation-driven CASP algorithm.
- *Fixed action patterns* without "checkpoints". These are stereotypic patterns which, once triggered, are executed without any further reference to environmental events.
- *Action sequences* with free positions which require the agent to examine whether the pattern is still appropriate in the current situation.

3.5 The Overall Hierarchy

The hierarchical structure of schemata within CLife is shown in Fig. 5. The actual representational elements are schemata. Individuals phylogenetically inherit schemata in form of indices which point to the phylogenetic schema library, i.e., the *innate knowledge*. Indices are ordinary numerical values (such as integers) which range from 1 to the number of entries of this library. Genetic operators as known from genetic algorithms change the indices during the simulated evolution.

3.6 Escaping genetic loads

How can artificial systems gain the advantage of structural coupling without inheriting the disadvantage of inflexibility? The key is to *break open* a phylogenetic element, i.e., to make grouping of elements reversible[14]. New individuals only have the phylogenetic repertoire of behavior schemata. Newly added ontogenetic schemata may consist of:

- ontogenetic elements whose characteristics are found by trial–and–error or more sophisticated learning algorithms (see below);
- new arrangements of phylogenetic elements;
- ontogenetic copies of phylogenetic elements within which one or more values are changed (broken open). For instance, an ontogenetic copy of the concept "apple" may expand the interval for the color so that it also includes yellow apples. From this point on, the experiences of the individual will decide whether the new concept is superior to the traditional phylogenetic one. The advantage of breaking open phylogenetic elements is to escape canalizations which in the long run make adaptations to new environments more difficult.

3.7 Ontogenetic Learning

From the viewpoint of constructivism, success and failures are not ontological, observer-independent criteria but are only real in the domain of reality brought forth by operationally coherent actions. To make a mistake merely means that the reality expected by the observer is different from the reality within which the 'unsuccessful' action takes place. For these reasons, no explicit fitness function except survival need be chosen (which is indeed the only criterion in an ecological system).

Thus, there is no need for any rewarding at all. Instead, schemata compete in the following way: Each schema is labeled with a measure of the schema's *degree of generality*, i.e., the sum over all interval differences in its condition part. The wider a condition interval the higher its generality. If the conditions of two or more schemata are satisfied, the algorithm chooses the schema with the lower generality. The *frequency* of calls is stored so that schemata which have been called more often than others have a higher priority. Also, there is some kind of *random noise* that influences the decision between competing schemata.

3.8 Phylogenetic Learning

The genotype of each individual consists of indices that refer to the respective phylogenetic libraries. This requires a higher-cardinality alphabet, namely the set of integers. Each genome is prefixed with a tag that indicates the kind of library it refers to. Due to this characteristic, CLife uses a position-independent encoding, i.e., genes code for phenotypic elements regardless of where they are located.

CLife emphasizes the importance of introns. It can be argued that such introns provide a better

14. This is comparable to the compression and expansion feature of Angeline's module acquisition approach [1]

chance to capture a part of the genotype intact if operators such as mutation and crossover are applied. This property of preservation is especially important in later generations where genetic operators can easily destroy sequences of the genotype with above-average fitness. Each gene is assigned with a weight that indicates its mutability. The weights decrease the chance of the respective genes being altered through mutation or crossing over. This is the exact reverse of introns in biological systems, where the only possibility to achieve different degrees of mutability within a genotype is to replicate genes. Introns are also important to maintain established chains between condition and action elements, blocks, and schemata.

4. Interpreter Shell

Within this shell, the sensory and motor surfaces are defined as subsets of the cells of the cognitive apparatus. This is done by mapping certain environmental perturbations onto the sensory surface, and by mapping the values of the cells within the motor subset into environmental effects (cf. Fig. 1).

4.1 The Sensory Surface

This component is capable of perceiving environmental perturbations of various kinds. In a typical artificial life environment this includes visual, acoustic, tactile, olfactory and proprioceptive perturbations. A technical task may use only a subset of these modalities.

4.2 The Motor Surface

The effectors can be thought of as locomotion, acoustic and visual utterances (in order to develop various types of communication through building consensual domains between several agents), laying trails (which will offer the possibility to establish social relationships between agents), and mating (in ALife environments).

5. Environments

In this section I provide two examples of environments to test the CLife approach.

5.1 Classical Artificial Intelligence Environment

Tic-Tac-Toe provides a simple environment for testing the CLife model with respect to its ability to build up condition and action hierarchies, i.e., functional couplings. A player has 9 sensors, each connected to the corresponding field of the 3×3 squares. These sensors deliver values between 0 and 2 depending on whether the field is empty (0), belonging to oneself (1), or belonging to the opponent (2). The interpreter shell writes the values of the sensors into the first 9 cells of the cognitive apparatus that consists of $9 + n + 9$ cells, where n is a variable which may be varied in different runs. The last 9 cells are interpreted as effectors which place a piece on the board. Due to the rules of Tic-Tac-Toe these "effector cells" may only be written once.

As a first step, the system creates a set of random conditions, actions, blocks, and schemata, each schema assigning a random degree of generality. Now the CASP algorithm compars the condition parts of the schemata with the current situation—initially there is an empty board. The triggered schema may then write values in various cells. The interpreter now reads the value of the first changed effector cell of the cognitive apparatus and places a piece according to this cell. There are interesting variants of this configuration: (a) The sensor cells are restricted to boolean values which indicate whether a field is empty or not but which say nothing about the owner of a piece. To determine the owner the agent must refer to the state of the effector cells. This implements proprioceptive knowledge; (b) In addition to (a), the state of the board is considered as 9bit integer (where 0 refers to an empty board and 511 represents a board where all pieces are set) rather than using 9 sensors and 9 effectors. Comparing the results from this experiment with the original representation throws light on integer representation.

Games are played generation after generation. Each agent contributes to the phylogenetic library by adding new rules which can potentially be used by its descendants. Each schema keeps book on the number of times it is called; analyses of these frequencies explain the usefulness of the phylogenetically evolved hierarchy.

5.2 Artificial Life Environment

Originally, CLife modelling was designed for artificial life environments, i.e., computational ecosystems. Its purpose is to investigate the emergence of higher cognitive structures from building invariants in simple sensorimotor beings. The environment within which the creatures act is assumed to be two-dimensional and both spatially and temporally discrete. A strict one-object-per-position rule is followed, i.e., a creature can move to an already occupied position if the creature either eats the object (whether non-living or living) or moves it to an adjoining space.

ceedings of the Third Conference on Artificial Life.

[2] Braitenberg, V. (1984), *Vehicles. Experiments in Synthetic Psychology. Cambridge*, MA: MIT Press

[3] Foerster, H. von (1973), On Constructing a Reality. In: Preiser (ed.) *Environmental Research Design*, Vol. 2. Stroudsburg: Dowden, Hutchinson & Ross, pp. 35-46

[4] Glasersfeld, E. von (1988), An Exposition of Radical Constructivism. in: Donaldson (ed.), *Texts in Cybernetic Theory*. American Society for Cybernetics

[5] Klopfer, P. (1968) *Ökologie und Verhalten*. Stuttgart: G. Fischer

[6] Lorenz, K. Z. (1937) Über die Bildung des Instinkbegriffes. *Naturwissenschaften* 25, pp. 289-331

[7] Maturana, H. R. & Varela, F. J. (1979), *Autopoiesis and Cognition: The Realization of the Living*, Boston: Reidel.

[8] Maturana, H. R. & Varela, F. J. (1984) *The Tree of Knowledge.*

[9] Maturana, H. R., Uribe, G. & Frenk, S. (1968) *A Biological Theory of Relativistic Color Coding in the Primate Retina*, Archivos de Biología y Medicina Experimentales, Santiago, Suplemento No. 1

[10] Piaget, J. (1954), *The Construction of Reality in the Child*. New York: Ballentine

[11] Riedl, R. (1978) *Order of Living Systems*.London: John Wiley.

[12] Riegler, A. (1992), Constructivist Artificial Life, and Beyond. Paper presented at the *Workshop on Autopoiesis and Perception*, Dublin City University Aug. 1992.

[13] Riegler, A. (1994), Fuzzy Interval Stack Schemata for Sensorimotor Beings. To appear in: *Proceedings of the From Perception to Action (PerAc'94) Conference*. Lausanne, Sept. 7-9, 1994. IEEE Computer Society Press.

[14] Sjölander, S. (1993) Some cognitive breakthroughs in the evolution of cognition and consciousness, and their impact on the biology of language. In: *Evolution and Cognition* vol. 3, no. 1

[15] Wuketits, F. M. (1987) Evolution als Systemprozeß. In: Siewing, R. (Ed.) *Evolution*. 3rd edition. Stuttgart, New York: Gustav Fischer (UTB 708)

On genetic algorithms for the packing of polygons

Stefan Jakobs
sjakobs@Pool.Informatik.rwth-aachen.de
RWTH Aachen
Lehrstuhl C für Mathematik,
Templergraben 55, 52062 Aachen, Germany

1 Introduction

In the steel industry problems frequently occur when the need to stamp polygonal figures from a rectangular board arises. The aim is to maximize the use of the contiguous remainder of the board. Similar problems exist in the textile industry, when clothes are cut out of a rectangular piece of material.

In order to solve these problems let us consider the following simpler approach. Given a finite number of rectangles r_i, i=1..n, and a rectangular board, an **orthogonal packing pattern** requires by definition a disjunctive placement of the rectangles on the board in such a way that the edges of r_i are parallel to the x- and y-axes, respectively. The computation of the orthogonal packing pattern with minimal height is called **orthogonal packing problem** (OPP).

The magnitude of the search space of the orthogonal packing problem is infinit, because for every time you move a rectangle in a packing pattern in a possible direction, a new packing pattern is created. In order to effectively reduce the number of possible orthogonal packing patterns the so called **bottom-left-condition** (BL-condition) is introduced. The orthogonal packing pattern fulfills the BL-condition if no rectangle can be shifted further to the bottom or to the left. In addition, the complexity of the problem is NP-complete.

2 Genetic Algorithm

The data structure is important for the genetic algorithm (GA). The first genetic algorithms worked with bit-strings. Over the last few years, GAs have been developed which work on the basis of different structures of data. Here each packing pattern is represented by a permutation π. The permutation represents the sequence in which the rectangles are packed. The advantage of this data structure is the facile creation of new permutations by changing the sequence. A consequence of the variable data structure is the fact that every permutation has to be assigned to a unique packing pattern. This decoding of the genotype is realized by a deterministic algorithm called **BL-algorithm**.

3 Extension to polygons

One approach for the extension to polygons is based on the use of a deterministic algorithm to convert the permutation of polygons into a packing pattern. The cost of existing algorithms is greater than $\mathcal{O}(n^2)$. In the GA for each step one permutation has to be converted. For this reason it is not advisable to use this approach. The **embedding-shrinking algorithm** offers a faster alternative. It consists of three steps:

Step 1: Embed the polygons into rectangles.
Step 2: Apply the GA to the embedded rectangles.
Step 3: Move the polygons in the packing pattern closer together

4 Conclusions

The aim is not the presentation of an optimal packing algorithm. It adresses the problem of improving deterministic packing algorithms. In practice the combination of deterministic and genetic algorithms provides a possible escape out of local minima. A further advantage is the fast and easy implementation of the combination. If a deterministic packing algorithm based on permutation is known, the algorithm could be improved by the genetic algorithm presented here.

References

[1] B. S. Baker, E.G. Coffman, R. L. Rivest: Othogonal Packings in Two Dimensions. SIAM Journal on Computing, Vol. 9 No. 4 (1980) 846–855

[2] D. J. Brown: An improved BL lower bound. Inform. Process. Letters. 11, No. 1 (1980) 37–39

[3] E.G. Coffman jr., M. R. Garey, D. S. Johnson: Approximation algorithms for bin-packing - an updated survey. Approximation Algorithms for Computer System Design (1984) 49–106

[4] R. J. Fowler, M. S. Paterson, St. L. Tanimoto: Optimal packing and covering in the plane are NP-complete. Inf. Process. Letters 12, No. 3 (1981) 133–137

[5] M. Haims: On the optimum two-dimensional allocation problem. Ph.D.-Dissertation, Dep. of Elec.

Engrg., New York University, Bronx, Tech. Rep. (1966)

[6] K. De Jong: An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan (1975)

[7] Zbigniew Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag (1992)

[8] Josef Nelißen: Die Optimierung zweidimensionaler Zuschnittprobleme. Schriften zur Informatik und Angewandten Mathematik, RWTH Aachen, Nr. 150 (1991)

[9] D.D.K.D.B. Sleator: Times optimal algorithm for packing in two dimensions. Information Processing Letters, Vol.10, No. 1 (1980) 37–40

[10] J. Terno, R. Lindemann, G. Scheithauer: Zuschnittprobleme und ihre praktische Lösung. Harri Deutsch-Verlag (1987)

An Evolutionary Heuristic for the Minimum Vertex Cover Problem

Sami Khuri
San José State University
Dept. of Mathematics & Computer Science
One Washington Square
San José, CA 95192–0103
U.S.A.

Thomas Bäck
University of Dortmund
Dept. of Computer Science, LS XI
D–44221 Dortmund
Germany

Introduction

The minimum vertex cover problem (mvcp) belongs to the class of NP-hard problems. The maximum clique problem can be reduced to it [3]. Thus, the search for an optimal solution is intractable (unless of course P = NP). Due to its numerous applications, especially in various matching problems, the problem is not abandoned. The goal is to find heuristics: approximation algorithms that have polynomial running times that return near-optimal solutions.

In this work we describe the results of applying a genetic algorithm [6, 4] to the mvcp. The latter is a highly constrained combinatorial optimization problem. Unlike traditional approaches that use domain-specific knowledge, and specialized genetic operators, the approach presented here makes use of a graded penalty term incorporated in the fitness function to penalize infeasible solutions. The fitness function itself is quite simple and needs to be added to GENEsYs, the genetic algorithm software package we use in this work. This package is based on Grefenstette's widely used GENESIS [5].

Following the formal introduction of the mvcp, the best known heuristic algorithm for that problem is introduced. The study then focuses on the genetic-based heuristic. Several problem instances are used with both algorithms and the results are compared. Our work concludes with some observations about our findings, and some suggestions on the use of evolutionary heuristics for other combinatorial optimization problems.

The Minimum Vertex Cover Problem

The mvcp of an undirected graph $G = (V, E)$ where V is the set of vertices and E denotes the set of edges, consists in finding the smallest subset $V' \subseteq V$ such that $\forall \langle i, j \rangle \in E$, we have $i \in V'$ or $j \in V'$ (or both). V' is said to be a vertex cover of G. The following is a formal definition of the mvcp in which we make use of Stinson's terminology for combinatorial optimization problems [14]:

Problem instance: A graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges. An edge between vertices i, j is denoted by the pair $\langle i, j \rangle \in E$. We define the *adjacency matrix* (e_{ij}) according to

$$e_{ij} = \begin{cases} 1 & , \text{ if } \langle i, j \rangle \in E \\ 0 & , \text{ otherwise} \end{cases}$$

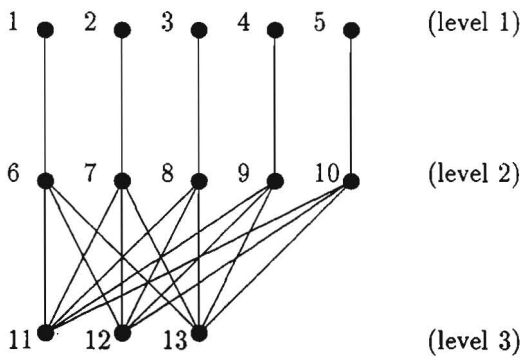


Figure 1: Construction of the regular graph after Papadimitriou and Steiglitz.

Feasible solution: A set V' of nodes such that $\forall \langle i, j \rangle \in E$: $i \in V' \vee j \in V'$.

Objective function: The size $|V'|$ of the vertex cover V'.

Optimal solution: A vertex cover V' that minimizes $|V'|$.

Since we are interested in covering all the edges of the given graph by using as few nodes as possible, one might be tempted to use a greedy-based heuristic to tackle the mvcp. The algorithm consists in repeatedly selecting a vertex of highest degree (the node that covers as many of the remaining edges as possible), and removing all of its incident edges. This is not a good strategy as was demonstrated by Papadimitriou and Steiglitz ([10], p. 407). They considered regular graphs, each of which consists of three levels. The first two levels have the same number of nodes while the third level has two nodes less than the number of nodes found on the previous two levels. More precisely, each graph consists of $n = 3k + 4$ ($k \geq 1$) nodes; $k + 2$ nodes on the first level, labeled $1, \ldots, k + 2$; followed by $k + 2$ nodes on the second level, labeled $k + 3, \ldots, 2k + 4$, while the k nodes of the third level are labeled $2k + 5, \ldots, 3k + 4$.

The regular graph for $k = 3$ can be found in [10] (p. 407) and is reproduced in figure 1.

A minimum vertex cover is obtained by choosing all the nodes of the second level, but the greedy strategy would start with the nodes of the third level, since these have highest degree. Consequently, the greedy strategy

mvcp100-01		mvcp100-02		mvcp100-03		mvcp100-04		mvcp100-05		PS100	
$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N
53	1	55	34	55	77	55	96	55	99	34	65
54	1	57	3	56	1	67	1	83	1	66	35
55	3	59	2	59	6	68	1				
56	6	60	3	63	3	75	1				
57	4	61	10	64	3	90	1				
58	4	62	1	66	1						
59	9	63	8	67	1						
60	4	64	1	68	1						
61	6	65	1	70	1						
62	12	66	6	71	1						
63	5	67	6	74	1						
> 63	45	> 67	25	> 74	4						
$\bar{f} = 62.61$		$\bar{f} = 62.75$		$\bar{f} = 57.62$		$\bar{f} = 55.80$		$\bar{f} = 55.28$		$\bar{f} = 45.20$	

Table 1: Experimental results obtained by the genetic algorithm for five random graphs of size $n = 100$ with edge density $d = 0.1$ ("mvcp100-01"), $d = 0.2$ ("mvcp100-02"), $d = 0.3$ ("mvcp100-03"), $d = 0.4$ ("mvcp100-04") and $d = 0.5$ ("mvcp100-05") and the regular graph of size $n = 100$ from Papadimitriou and Steiglitz ("PS100").

finds a solution of size $2k+2$ (since it has to select $k+2$ nodes in addition to the nodes of the third level), while the optimal solution has size $k+2$.

It can be shown that the greedy algorithm never produces a solution which is more than $\ln(n)$ times the optimum, where n is the number of vertices ([9], p. 323). More precisely, the algorithm performs with a relative error $\frac{\ln(n)}{n}$ (which grows as fast as $\ln(n)$) ([10], p. 408).

The following simple algorithm is surprisingly the best approximation algorithm known for the mvcp ([9], p. 301). It can be shown that the size of the vertex cover it returns is guaranteed to be no more than twice the size of an optimal vertex cover ([1], p. 968).

ALGORITHM 1 (vercov)

$C := \{\ \}$;
  {C contains the vertex cover being constructed}
$E' := E$;
while $E' \neq \emptyset$ **do**
  randomly choose $\langle u, v\rangle \in E'$;
  $C := C \cup \{u, v\}$;
  $E' := E' - \{\langle x, y\rangle \mid x = u \vee y = v\}$;
    {remove from E' every edge incident on
     either u or v};
od
return C;

To compare the performance of the above algorithm with the genetic algorithm, a cover V' is represented by a binary vector $\vec{x} = x_1 x_2 \ldots x_n$ where $x_i = 1$ if the i^{th} node is in V', and $x_i = 0$ if it is not. Using this representation, we developed the following fitness function to be minimized by the genetic algorithm:

$$f(\vec{x}) \quad = \quad \sum_{i=1}^{n} \left(x_i + n \cdot (1 - x_i) \cdot \sum_{j=i}^{n} (1 - x_j) e_{ij} \right)$$

The term $\sum_{i=1}^{n} x_i$ of $f(\vec{x})$ determines the size of the potential vertex cover represented by $\vec{x}$, while the term $n \cdot \sum_{i=1}^{n} \sum_{j=i}^{n} (1-x_i) \cdot (1-x_j) \cdot e_{ij}$ penalizes sets V' that are not covers by adding a penalty of magnitude n for each edge e_{ij} for which $i \notin V'$ and $j \notin V'$. Consequently, the second term drops to zero for feasible solutions.

The fitness function was developed according to the following design principles that are important for a successful penalty function approach [11, 13, 7]:

- The penalty should be graded, i.e., fitness values should improve as solutions approach (in terms of the Hamming distance) feasible regions of the search space.
- Infeasible binary vectors are guaranteed to yield fitness values which are inferior to fitness values of even the worst feasible solutions.

Experimental Results

The experiments reported in this section are performed by using a genetic algorithm with a population size $\mu = 50$, a mutation rate $p_m = 1/n$, crossover rate $p_c = 0.6$, proportional selection, and two-point crossover. As reported in [2, 12], the latter is expected to perform better than the traditional one-point crossover. In order to apply the genetic algorithm to the minimum vertex cover problem, no component of this general genetic algorithm — except, of course, the fitness function — has to be modified. This fact reflects the wide applicability

mvcp100-01		mvcp100-02		mvcp100-03		mvcp100-04		mvcp100-05		PS100	
$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N
80	6	80		80		80	1	80		66	100
82	17	82	1	82		82		82	1		
84	23	84	2	84	1	84	1	84			
86	20	86	4	86	6	86		86	5		
88	17	88	18	88	12	88	5	88	7		
90	14	90	17	90	15	90	13	90	10		
92	1	92	31	92	18	92	24	92	21		
94	2	94	20	94	28	94	34	94	27		
96		96	4	96	16	96	19	96	17		
98		98	1	98	4	98	3	98	9		
100		100		100		100		100	3		
$\bar{f} = 86.46$		$\bar{f} = 89.22$		$\bar{f} = 92.22$		$\bar{f} = 92.96$		$\bar{f} = 93.12$			

Table 2: Experimental results obtained by the vercov heuristic for five random graphs of size $n = 100$ with edge density $d = 0.1$ ("mvcp100-01"), $d = 0.2$ ("mvcp100-02"), $d = 0.3$ ("mvcp100-03"), $d = 0.4$ ("mvcp100-04") and $d = 0.5$ ("mvcp100-05") and the regular graph of size $n = 100$ from Papadimitriou and Steiglitz ("PS100").

and robustness of genetic algorithms when compared to problem-specific heuristics.

For the experimental tests, random graphs of size $n = 100$ with different edge densities d, $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, are used. For example, an edge density of $d = 0.1$ means that an edge is placed between two nodes with a probability of 0.1. For each of these problems, a total of $N = 100$ runs of the genetic algorithm is performed. The results are summarized in table 1 for the best results that were encountered during the 100 runs for each test problem. For each problem instance, we record the different fitness values that were obtained and their frequencies. Furthermore, the average fitness value $\bar{f}$ over all 100 runs is indicated at the bottom of the table. The total number of function evaluations per single run is chosen to be $2 \cdot 10^4$, such that only an extremely small fraction of the search space is tested by the genetic algorithm.

In addition to the randomly constructed graphs, the regular graphs introduced by Papadimitriou and Steiglitz ([10], pp. 406–409) and described in the previous section of this paper, are used to compare the behavior of the genetic algorithm with the vercov heuristic.

Recall that these graphs contain $n = 3k + 4$ ($k \geq 1$) nodes distributed on three levels. They can be scaled up by choosing high values for k. We choose problem instances of the regular graph of sizes $n = 100$ ($k = 32$) and $n = 202$ ($k = 66$).

For each of the problems a total of $N = 100$ independent runs of the vercov heuristic is performed, and the results are summarized in table 2.

The same experiments were also performed for graphs of size $n = 200$ in order to test the behavior of the genetic algorithm as well as the vercov heuristic for an even larger problem size. In this case, the genetic algorithm was allowed to run for $4 \cdot 10^4$ function evaluations per experiment. The corresponding results obtained by the genetic algorithm are shown in table 3, while the results from the vercov heuristic are presented in table 4.

From these tables, a number of interesting conclusions can be drawn. For the random graphs, which were used to compare the genetic algorithm and the vercov heuristic, it is known by construction that an optimum of quality 55 (respectively 110) exists, which is likely to be the global optimum. This optimum is found by the genetic algorithm at least once within the $N = 100$ runs that were performed.

Moreover, the randomly constructed problems quickly become simpler for the genetic algorithm when the edge density is increased. For an edge density of $d = 0.5$, the genetic algorithm almost surely finds the global optimum of the problems.

In case of the regular graph after Papadimitriou and Steiglitz, the genetic algorithm is able to find the global optimum in about 2/3 of all runs, while the remaining runs identify a solution of quality $2k + 2$.

On the other hand, the vercov heuristic performs poorly. For the randomly constructed problems, the heuristic finds best solutions of quality 80 (for graphs of size 100) and 172 (for graphs of size 200), respectively. Surprisingly, the average performance of the vercov heuristic decreases as the edge density is increased, i.e., the problems become even harder for the vercov heuristic while they become much simpler for the genetic algorithm when the edge density increases.

For the regular graph instances, the vercov heuristic fails in all of the 100 runs that were performed for each of the graphs. Recall that the graphs are constructed so as to force the vercov heuristic into yielding solutions of quality $2k + 2$. The random choice of edges in the while-loop of the algorithm implies that nodes from either layers one and two, or from layers two and three

mvcp200-01		mvcp200-02		mvcp200-03		mvcp200-04		mvcp200-05		PS202	
$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N
110	2	110	55	110	95	110	99	110	100	68	60
113	1	120	10	128	1	140	1			134	40
116	4	121	7	129	6						
117	2	122	2	130	2						
119	3	123	7	134	1						
120	1	125	1								
121	2	126	1								
122	3	127	1								
124	3	129	1								
125	6	132	2								
126	2	134	1								
> 126	71	> 134	12								
$\bar{f} = 132.50$		$\bar{f} = 119.07$		$\bar{f} = 111.01$		$\bar{f} = 110.30$		$\bar{f} = 110.00$		$\bar{f} = 108.00$	

Table 3: Experimental results obtained by the genetic algorithm for five random graphs of size $n = 200$ with edge density $d = 0.1$ ("mvcp200-01"), $d = 0.2$ ("mvcp200-02"), $d = 0.3$ ("mvcp200-03"), $d = 0.4$ ("mvcp200-04") and $d = 0.5$ ("mvcp200-05") and the regular graph of size $n = 202$ from Papadimitriou and Steiglitz ("PS202").

of the graph are involved in the solution constructed, which in turn implies that two layers have to be part of the solution found by the heuristic [10].

Conclusion

In this work, we have demonstrated that genetic algorithms can be used in a straightforward way to find good approximate solutions of the minimum vertex cover problem. Moreover, the results found by the genetic algorithm are better than those obtained from the best known traditional heuristic, the vercov algorithm. These findings, in addition to other good results obtained on different, highly constrained combinatorial optimization problems such as the subset sum [7], minimum tardy task [7], and multiple knapsack problems [8], give strong evidence that genetic algorithms can yield good solutions for a wide range of hard combinatorial optimization problems for which solutions can be represented by binary strings.

Acknowledgements

The first author was partially supported by the California State University research award GS850-851. The second author gratefully acknowledges support by the German BMFT, grant 01IB403A, within the project EVOALG.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1990.

[2] L. J. Eshelman, R. A. Caruna, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 10–19. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[3] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* Freemann & Co., San Francisco, CA, 1979.

[4] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.

[5] J. J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.

[6] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[7] S. Khuri, Th. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In D. Cizmar, editor, *Proceedings of the 22nd Annual ACM Computer Science Conference*, pages 66–73, Phoenix, 1994. ACM Press, New York.

[8] S. Khuri, Th. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, editors, *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 188–193. ACM Press, New York, 1994.

[9] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, Reading, MA, 1994.

[10] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice Hall, 1982.

mvcp200-01		mvcp200-02		mvcp200-03		mvcp200-04		mvcp200-05		PS202	
$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N
172	2	172		172		172		172		134	100
174	3	174		174	1	174		174			
176	13	176	1	176	1	176	1	176	1		
178	14	178	1	178	3	178	4	178	1		
180	23	180	5	180	5	180	1	180	5		
182	14	182	11	182	5	182	9	182	5		
184	15	184	18	184	17	184	8	184	5		
186	7	186	16	186	19	186	16	186	13		
188	5	188	24	188	16	188	16	188	11		
190	3	190	15	190	16	190	16	190	21		
192	1	192	4	192	9	192	15	192	19		
> 192	0	> 192	5	> 192	8	> 192	14	> 192	19		
$\bar{f} = 180.98$		$\bar{f} = 186.46$		$\bar{f} = 186.92$		$\bar{f} = 188.06$		$\bar{f} = 189.16$			

Table 4: Experimental results obtained by the vercov heuristic for five random graphs of size $n = 200$ with edge density $d = 0.1$ ("mvcp200-01"), $d = 0.2$ ("mvcp200-02"), $d = 0.3$ ("mvcp200-03"), $d = 0.4$ ("mvcp200-04") and $d = 0.5$ ("mvcp200-05") and the regular graph of size $n = 202$ from Papadimitriou and Steiglitz ("PS202").

[11] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[12] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[13] A. E. Smith and D. M. Tate. Genetic optimization using a penalty function. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 499–505. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[14] D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms.* The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 2nd edition, 1987.

Genetic Algorithm for near optimal Scheduling and Allocation in High Level Synthesis

Sri-Krishna Aditya Magdy Bayoumi Chid Lursinsap

The Center for Advanced Computer Studies,

University of Southwestern Louisiana,

Lafayette, LA 70503, USA

Abstract

Scheduling data flow graphs is a critical task in high-level synthesis and is an NP-Complete problem. In this paper we present the application of Genetic Algorithm, a general optimization technique, to the problem of scheduling and Allocation. The method schedules under a variety of constraints like resource, time, interconnect constraints. It can schedule multicycle, chained and mutually exclusive operations. With minor modifications to cost functions we can take care of functional pipelining.

*Also, if a better heuristic for the problem is found we can always integrate that heuristic with this method with a guarantee that this method performs equally well or better than the heuristic found. This technique can be easily parallelized which makes it very promising for application in large designs. Complexity of the procedure is O(S*N) where S is number of control steps and N is number of nodes*

1. Introduction

Input to a *High-Level Synthesis* system is a high level specification of a digital system which is called as behavior of the system, along with constraints on hardware, timing etc., Output of a *High-Level synthesis* system is the structure which realizes the specified behavior of the system.

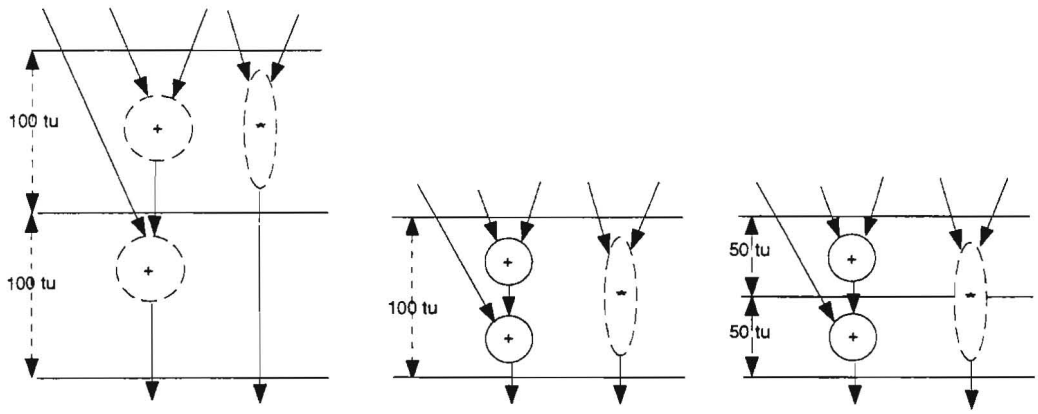
The tasks involved in the high level synthesis are

- *Compilation* — Behavioral description in a hardware description language like VHDL is compiled into optimized data flow and/or control flow graphs. For optimization, several techniques like dead-code elimination etc., are borrowed from compilation of programming languages.
- *Scheduling* —is the assignment of each operation to a control step which generally corresponds to a clock cycle in synchronous systems. This is considered to be the most critical task in high level synthesis.
- *Allocation & binding* — Allocation is determining the number of each type of hardware modules, registers and buses required. Binding is the assignment of each operation to a specific functional unit (hardware resource), assignment of variables to registers and transfers to buses and is termed data path synthesis.
- *Controller* generation for the resulting schedule.
- Output the structural representation of the design.

Scheduling without operation chaining and multicycle operations: Let us assume addition operations take 50 time units and multiplication operations take 100 time units. In fig 1.a we need to have a clock cycle time of 100 time units, time taken by the longest operation. Here the implementation needs two control cycles and total execution time will be 200 time units.

Scheduling with operation chaining: When the consecutive running time of several operations is less than the clock cycle time we can schedule all of them in a single clock cycle and this is called operation chaining. Here (fig-1.b) the two add operations can be executed in a single cycle of 100 time units as their total execution time is less then or equal to the clock cycle time. Schedule takes only 100 time units and one control step (instead of two control steps as in earlier schedule) which simplifies controller.

Scheduling with multicycle operations: If we reduce the clock cycle time to 50 time units then multiplication operation takes two clock cycles. The schedule (fig-1.c) requires two control steps and we need a total execution time of 100 time units. Here the multiplication operation is scheduled in two control steps and hence takes two clock cycles.



(a) Scheduling without operator chaining or multicycle operation Process takes two control steps of 100 time units each. i.e., total process time is 200 time units.

(b) Scheduling with operator chaining. This results in only one control step of 100 time units - Results in simpler control and faster operation.

(c) Scheduling with multicycle operations. Processing time is same as case (b) i.e., 100 time units. But has two control steps of 50 time units each.

Figure 1

Bus requirement of a schedule: Minimum bus requirement of a schedule is the maximum number of concurrent data transfers taking place in the schedule.

Register requirement of a schedule: For a scheduled control data flow graph (CDFG) the register requirement is given by the maximum number of graph edges crossing a control step boundary.

2. Genetic Algorithms

Paulin et al., [1] give an excellent survey of scheduling techniques in the context of high level synthesis, along with their own force directed scheduling methods. Application of general optimization methods like simulated annealing for scheduling/allocation problem can be found in SALSA[2] and in [3]. Genetic Algorithm has been applied to scheduling in [4] wherein two strings are used to encode the problem. Our approach which uses only one string encoding is different from this work.

Genetic algorithms, like simulated annealing are stochastic search algorithms for optimization problems. Excellent introduction to genetic algorithm theory and applications can be found in [5–6]. To apply genetic algorithms to a optimization problem we should have

- A way of encoding the solutions of the problem —Here Chromosome is a string of genes. A position on the chromosome can be occupied by one of the genes from a set of possible genes (called allelles) for that position. It is possible that presence of one gene at a particular position and another gene at another position make the genotype (solution) illegal. That means a certain combination of genes may be illegal.
- A way of creating an initial population — This may be a random generation of solutions to the problem or solutions provided by other algorithms or a combination of both.
- A way of evaluating the fitness of each solution (genotype) of the population.
- A way of generating the next generation by the members of the present population by
  - ☐ Cloning — Certain members of the present generation may be allowed to survive for the next generation or copied into the next generation.
  - ☐ Crossover —Two parents from the present generation are selected , with an *exponentially increasing probability of selection for fitter parents.* Chromosomes of each parent exchange part of their information and form two new chromosomes which constitute two new children into the next generation. The parents may or may not be mutated (altering of chromosomal information) before crossing them with a low probability.
- Determine the way of chromosomal information exchange (crossover), rate of crossover, percentage of cloning, mutation rates.
- Termination conditions

3. Chromosomal Representation

We have integer based representation rather than a binary representation for the chromosome. To illustrate the representation let us consider the control flow graph in fig-2(a). Let us assume that we are considering a time constraint of five control steps, hardware constraints of one adder unit and one multiplier unit and we are trying to get an optimal schedule where bus and register costs are

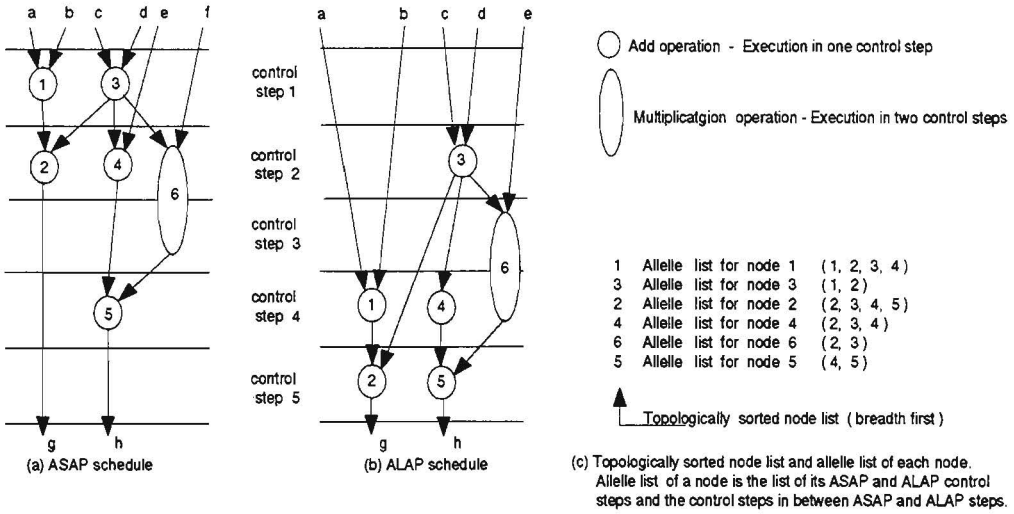


Figure 2

minimum. It may not be possible to meet all the constraints in which case the algorithm should indicate this possibility.

We will find ASAP (fig-2.a) schedule and ALAP (fig-2.b) schedule for the given CDFG and note down the ASAP step and ALAP step for each of the nodes of the graph. Then the possible allelle values for a node are its ALAP and ASAP steps and the control steps in between. An *Allelle list* of a node can be any permutation of the corresponding *allelle set*. In any legal schedule, a node can be scheduled into one of the control step corresponding to a member of the allelle list, if such a scheduling does not contradict precedence relations as indicated by the graph. Allelle lists forms the units of information exchange during crossover of two schedules.

Nodes of the graph are topologically sorted such that in the sorted list a node is entered only after all its predecessors are already entered. Fig-2.c shows one such sorting of the nodes.

4. Initial Population Generation

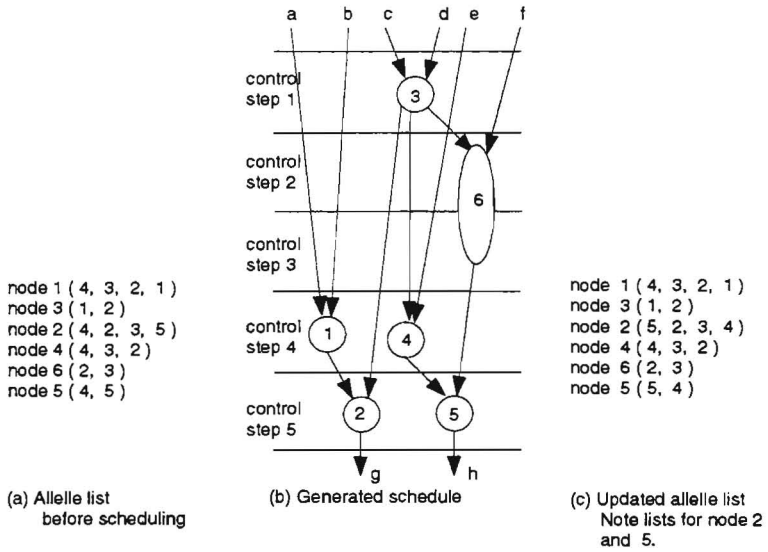
If the population of a generation is decided as N, then N copies of the sorted node list along with allelle set for each of the nodes are made. The allelle set of each node is randomly permuted in each of the copy.

A *Scheduler* takes each of these permuted copies and forms corresponding schedules. Scheduler builds schedules which meet precedence constraints (hard constraints) always and tries to meet the hardware constraints (soft constraints) as far as possible. Timing constraints because of the nature of the method are always met.

Scheduler scans the node list (which is sorted in topological order) and schedules each node in order. For each node it scans corresponding allelle list from left to right. It schedules the node to the first allelle (control step) in the list which meets the precedence constraints and also hardware constraints. If it is not possible to meet the hardware constraints, scheduling will be done to the first allelle in the list which does not contradict any precedence relations. Then that allelle is interchanged with the first allelle in the allelle list. Thus after scheduling is complete, the first allelle in each of the allelle list reflect the control step to which the corresponding nodes are scheduled.

For example let us assume a node X has a associated allelle list (a, b, c, d, e), and after scanning the allelle list, it was found that c is the first allelle in the list to which scheduling of X, meets precedence and hardware constraints. Then we schedule the node X into control step c and update the allelle list as (c, b, a, d, e). As soon as we schedule a node into a control step we should also update the corresponding control step's usage of hardware and the freedom of the subsequent nodes in the sorted list of nodes. Fig-3 shows a random permutation of the allelle list, the corresponding schedule developed by the scheduler and updating of the allelle list to reflect the actual schedule.

After scheduling all the nodes the scheduler checks to see if any of the control step is empty i.e., none of the nodes have been scheduled to that control step. In that case all the nodes scheduled to control steps later to this empty control step are "pulled-up" by one control step. For example a node X has an associated allelle list (after scheduling) of (12, 14, 10, 11, 13) and is scheduled to control step 12. Suppose if the control step 5 is found to be empty, then all the nodes scheduled to control step 6 or more have to be pulled up by one control step. Therefore node X has to be rescheduled to control step 11 and its allelle list updated as (11, 14, 10, 12, 13), by interchanging positions of 11 and 12 in the list. If there are more empty control steps in the schedule the process has to be repeated and this process of filling the empty control steps starts from the empty steps at the end of the schedule to those

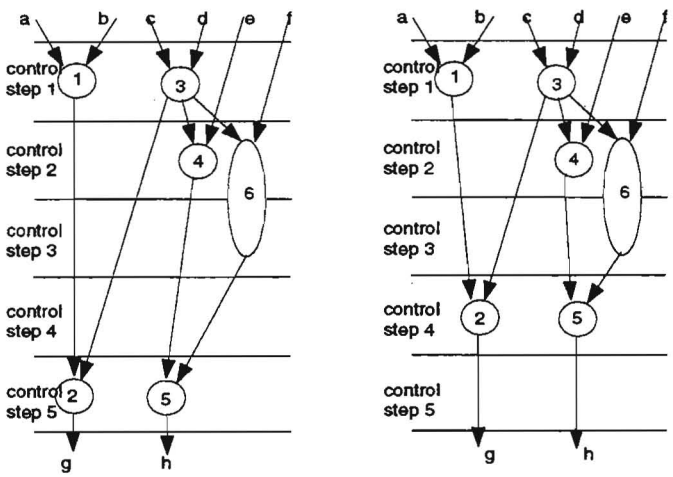


(a) Allelle list before scheduling

(b) Generated schedule

(c) Updated allelle list Note lists for node 2 and 5.

Figure 3



(a) A schedule and its corresponding allelle list

(b) Schedule modified to fill the empty control step 4 and the updated allelle list. Note changes in the allele lists of nodes 2 and 5.

Figure 4

towards beginning of the schedule. For the given example this is illustrated in fig-4.

We can seed initial population with ASAP , ALAP or schedules generated by other heuristics and it is guaranteed that genetic algorithm generates schedules *at least* as good as the best of these schedules. In our trials, we seeded the initial population with only ASAP, ALAP schedules and randomly generated schedules. We did not make use of any other heuristics.

5. Cost Calculation for schedules

After creating schedules in each generation we have to find the cost of each of the schedules. Measuring fitness of a schedule depends on what we are trying to optimize.

For example, if we are trying to get an optimized schedule for some hardware constraints and time constraints, then the following can be a good cost function for the schedule.

Let

X = Number of control steps allowed.
Y = Actual length of the schedule in number of control steps.
P = Penalty for generating shorter schedules.
B = Cost of each bus.
b = Actual number of busses used for the schedule.
R = Cost of each register.
r = Actual number of registers used for the schedule.
C_i = Cost of the hardware unit of type i.
H_i = Allowed number of hardware units of type i.
${}_ch_i$ = Actual number of hardware units of type i, used in control step c.

Then cost of the schedule is equal to

$$(X - Y) * P + B * b + R * r + \sum_i \sum_c (H_i - {}_ch_i) * C_i$$

The cost of the schedule is the objective function that has to be minimized. It should be noted that schedules shorter than the maximum specified length should be penalized as shorter schedules have a higher probability of breaking hardware constraints.

6. Sorting, Cloning, Crossover and Mutation

At each generation we calculate the cost of each schedule and we sort the schedules of the generation, in ascending order of their costs. If two schedules have same cost, schedules having longer length come before the schedules of smaller length, to encourage the schedules to exactly meet the time constraints rather than developing short stubby schedules. It is possible to use other criterion like cost of registers, or cost of buses etc., as a secondary field for sorting (in addition to cost of the schedule as the primary sorting field) to encourage schedules which are better in those aspects.

To produce next generation, a certain percentage of present population is calculated and we copy that many top schedules of the present generation to next generation. This is *Cloning.*

The remaining members of the next generation are formed by *cross over* of the present generation members. The probability of participation in crossover for a member of present generation increases exponentially with its fitness (which is inversely proportional to cost of schedule). For this purpose we use a random number generator with an exponential distribution.

Generation of exponential distribution: If RNG is output of a uniformly distributed random number generator, then X , a random variable with exponential distribution is given by

$$X = \perp \; -10 * log_e(RNG) \; \perp$$

This random number generator generates number one with high probability and the numbers two and onwards the probability of generation exponentially decreases. The numbers after thirty are practically never generated. Thus if we use a population size of 100, only the top thirty members participate in crossover.

To choose two parents from the sorted population list, we run the random number generator (one with exponential distribution) twice. Suppose the numbers generated are *X* and *Y*, then the parents chosen for crossing are the *Xth* and *Yth* schedules in the sorted population list. These two parents exchange the genetic information by swapping allelle lists at specific nodes determined by a random number generator of uniform distribution, which generates a string of 0s and 1s. Length of this string is equal to number of nodes. This is illustrated in fig-5 for the example we considered above. This type of crossover is called 0–1 crossover. It should be noted that in crossover we are not exchanging the genes themselves but permutations of genes corresponding to the node involved in crossover.

As each child is generated we can make it undergo mutation also, which is choosing one or more of its nodes with a very low probability and randomly permuting the allelle lists at those selected nodes. The process is repeated until the required number of children for the next generation are created. Scheduler then processes each children and generates schedules.

Then the process of finding cost of each schedule, sorting, building next generation by cloning, crossover and mutation is repeated for a predetermined fixed number of iterations or until a whole generation converges towards solutions of equal fitness.

Complexity of the procedure is of the order of O(P*G*S*N) where P is the population size, G is the number of generations, S is number of control steps and N is the number of nodes in the data flow graph. Since P and G are constants the procedure is of the complexity of O(S*N).

7. Experimental Results

The algorithm is implemented in C language and benchmark studies were done on SUN SPARC workstation. The benchmark we used is the popular fifth-order elliptic wave filter. This benchmark can be particularly challenging for a stochastic algorithm when scheduling for 21 control

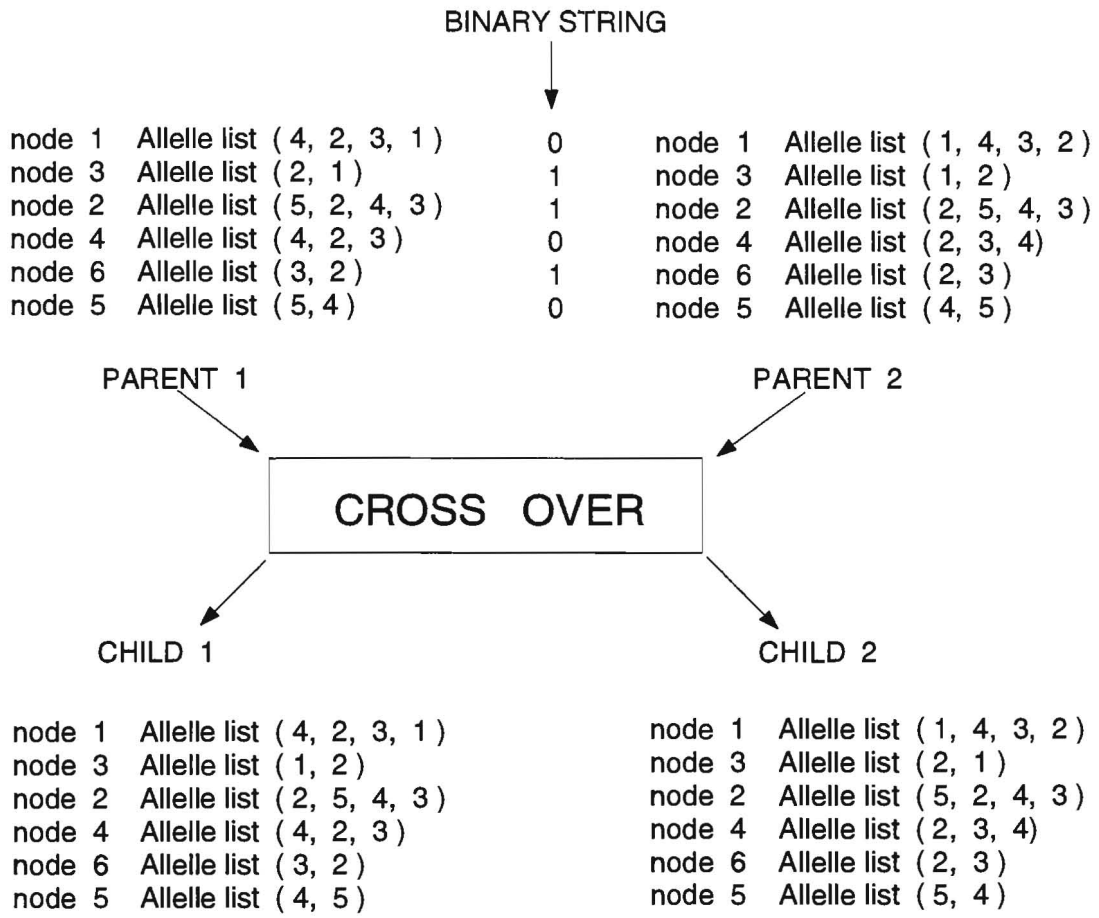


Figure 5

System	17 control steps	18 control steps	19 control steps	21 control steps
HAL	Add - 3 Mul - 3 Reg - 12 Bus - NA	Add - 3 Mul - 2 Reg - 12 Bus - NA	Add - 2 Mul - 2 Reg - 12 Bus - 6	Add - 2 Mul - 1 Reg - 12 Bus - NA
Genetic Algorithm	Add - 3 Mul - 3 Reg - 11 Bus - 6	Add - 3 Mul - 2 Reg - 11 Bus - 6	Add - 2 Mul - 2 Reg - 11 Bus - 4	Add - 2 Mul - 1 Reg - 11 Bus - 4

Table 1

steps as seven multiplication operations have to be scheduled end to end, if we have only one multiplier.

We used a population size of 100, cloning ratio of 50% and 50 generations. The algorithm, in all the runs took nearly 25 CPU seconds for scheduling this benchmark irrespective of whether we are scheduling to 17, 18, 19 or 21 control steps.

Table–1 gives the results obtained and the comparison with HAL [1]. Genetic algorithm gave better results than HAL in terms of registers and busses used for the schedule.

8. Conclusion

This paper presented, application of genetic algorithms for scheduling/allocation problem with timing, hardware and mutual exclusion constraints. The Genetic Algorithm is a robust optimization technique and by integrating problem specific knowledge to genetic algorithm we have obtained promising results. The approach can be easily parallellized. We are extending this work to schedule CDFGs with loops. Even though there are integer programming methods which always generate optimal solutions for this problem they will not be suitable for large graphs because of their exponential time complexity.

9. References

1. Paulin G.P, Knight J.P — "Force_Directed Scheduling for the Behavioral Synthesis of ASICs", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, June 1989, pp 661–679.
2. J. Nestor and G.Krishnamoorthy — "SALSA: A new approach to scheduling with timing constraints", *Proceedings of the ICCAD-90*, November 1990.
3. Srinivas Devadas and Richard Newton — "Algorithms for hardware allocation in data path synthesis", *IEEE Transactions on CAD*, July 1989, pp 768–781.
4. N. Wehn et al., — "A Novel Scheduling/Allocation Approach for Data Path Synthesis based on Genetic Paradigms" in *Logic and Architecture Synthesis, eds.*, P. Michel and G. Saucier, Elsevier Science Publishers B. V. (North-Holland).
5. David Edward Goldberg — *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley Publishers, 1989.
6. Lawrence Davis, editor — *Genetic Algorithms and Artificial Ssystems*, Morgan Kaufmann Publishers, 1987.

GAP - A Knowledge-Augmented Genetic Algorithm for Industrial Production Scheduling Problems

Ralf Bruns
Universität Oldenburg, Fachbereich Informatik
Postfach 2503, D-26111 Oldenburg (FRG)

1 Introduction

The task of production scheduling is the temporal and capacity oriented scheduling of a set of orders for manufacturing products. The manufacturing of a product is accomplished by the execution of a set of operations in a predefined sequence on certain machines. The goal of scheduling is the construction of a schedule (temporal assignment of production processes to machines), which minimizes a chosen evaluation function. Apart from some theoretical cases of little practical importance, the determination of an optimal solution to a scheduling problem belongs to the class of NP-complete problems. In addition to the combinatorial complexity, when dealing with real-world scheduling problems the requirements imposed by numerous details of the particular application domain, e.g. alternative machines, make the scheduling task even more difficult. In spite of a large number of developed scheduling methods (in Operations Research and, more recently, in Artificial Intelligence), only a few practical applications have entered into everyday use in industrial reality.

Almost all previous GA-based approaches to scheduling have only addressed simplified versions of scheduling problems so far, e.g. flow-shop problems, and have made use of a domain-independent problem representation, e.g. the list of all orders to be scheduled [Cleveland 89, Syswerda 91]. However, in addressing more complex scheduling problems the previous domain-independent representation schemes seem to have the disadvantage that the genetic algorithm is restricted to perform a search only on a part of the search space, e.g. only on the space of all permutations of orders. The rest of the search task has to be accomplished by a transformation procedure which has to search for information not provided by the representation scheme.

2 GAP - A Genetic Algorithm for Scheduling

The objective of the project GAP (Knowledge-augmented Genetic Algorithm for Production Scheduling [Bruns 93]) is the development of a genetic algorithm for the effective solution of real-world production scheduling problems. The approach is based on the augmentation of the genetic algorithm with problem-specific knowledge of the application domain.

A new direct representation of candidate solutions to the scheduling problem has been designed: the complete and consistent schedule itself is used as an individual. Thus, the GAP-algorithm operates directly on a population of schedules. This complex representation contains all information relevant for the description of a

schedule, i.e. all operations of all orders with associated machine assignments and time intervals of production as well as the selected process plan for each order.

Knowledge-augmented crossover and mutation operators have been designed to take advantage of the information included in the non-standard representation scheme.

The advanced crossover operator chooses a partial schedule of each selected parent schedule and creates a new offspring schedule by combining the two partial schedules. A suitable partial schedule of the first parent is determined by the random selection of a subset of the orders, their machine assignments and production intervals build a consistent partial schedule for the offspring. In the next step the assignments of the missing orders are chosen from the second parent and have to be inserted into the offspring schedule currently under creation. Occurring capacity conflicts are solved by delaying the machine assignments of orders. The performance of the system was further improved by the integration of heuristics for the selection and combination of the partial schedules.

In order to consider all alternatives specified in the scheduling problem, three separate operators have been developed for mutation, which randomly alter (a) the selected process plan of an order, (b) the selected machine of an operation, or (c) the selected time interval of an operation.

To circumvent the problem of illegal solutions each operator creates offspring schedules in a manner that guarantees that all constraints specified in the scheduling problem remain satisfied. In order to consider all constraints the genetic operators have (parts of) the functionality of knowledge-based scheduling algorithms. By means of the operators an offspring inherits from its parent schedules the machine assignment and the production interval (perhaps deferred) of each operation and, implicitly, the selected process plan of each order. Thus, all relevant scheduling information is subject to inheritance and the genetic algorithm can operate on the entire search space. Moreover, the integration of additional requirements of real-world applications, e.g. cleaning times, is possible due to the direct representation scheme and the knowledge-augmented genetic operators.

3 Experimental Results

Extensive experiments with an extremely complex instance of a real-world problem were performed with the GAP-system and a genetic algorithm based on a domain-independent representation (individual = list of orders). All experiments with different parameters showed the same typical behaviour, namely that the problem-specific GAP-approach generated much better schedules than the domain-independent one.

References

[Bruns 93] Bruns, R.: "Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling", ICGA-93.

[Cleveland 89] Cleveland, G.A., Smith, S.F.: "Using Genetic Algorithms to Schedule Flow-Shop Releases", ICGA-89.

[Syswerda 91] Syswerda, G., Palmucci, J.: "The Application of Genetic Algorithms to Resource Scheduling", ICGA-91.

Cognitive Filtering of Information by Genetic Algorithms

Max Höfferer, Bernd Knaus, Werner Winiwarter

Institute of Applied Computer Science and Information Systems
Liebiggasse 4/3, A-1010 Vienna, Austria
e-mail: {mh, bk, ww}@ifs.univie.ac.at

Abstract. An adaptive Information Filtering System is described which extracts e-mail messages from on-line resources. Adaptation is obtained by applying (1) linguistic analysis to get consistent representations of the contents of interesting e-mails, (2) an evolutionary algorithm for prioritizing morphologically parsed messages and (3) a monitor to simulate a user's cognitive behavior.

1. Introduction

More and more users of *e-mail* and other on-line communication systems are faced with the problem of selecting relevant information in a space of different information-sources. Participants of on-line conferences, members of office-information systems, and network-diary makers get in trouble with an endless stream of messages. To overcome this *information overload problem* [1] information filtering techniques are being developed to deliver information to users.

State of the art IFS, e.g. *Information Lens System* [2], *EDS Template Filler* [3] or *Isceen* [4] offer a static behavior. A cognitive information system should have the ability to learn and adapt itself to the user's behavior. First steps in this direction have been proposed in [5] where interface agents employ AI techniques to provide active assistance to a user with computer-based tasks.

The system presented - *Cognitive Information Filtering System* (*CIFS*) - applies genetic adaptation to learn from user feedback and user behavior. CIFS distils e-mails from the input stream depending on user's interests and evaluation judgements which are used to rank e-mail information.

With regard to efficiency the linguistic analysis of the e-mails is performed by use of a cascading architecture. The filtering proceeds by stepwise refinement of analysis techniques leading to a consecutive reduction of the problem of space. Therefore, only a very small fraction of the incoming e-mails must be analysed by time-consuming linguistic methods.
This paper is organized as follows. In Section 2 and Section 3 we will introduce the basic concepts of information filtering and cognitive models. The linguistic analysis applied performed by the indexer/parser module is presented in Section 4. Finally, Section 5 gives a view of the architecture of the cognitive information filtering system and an example of the actual filtering process.

2. Information filtering

Information Filtering (IF) describes the processes of distribution and delivery of information to users of communication systems. Information filters assist users in finding relevant information but are also used to target information to potentially interested users. Information Filtering Systems (IFS) handle primarily unformated textual data like documents, semi-structured like electronic-mail messages (*e-mail*), NetNews articles, NewsWire stories or more complex structures like hypertext documents containing voice, graphics, and pictures. IFS process streams of incoming data based on descriptions of a single user or groups of users. These user "profiles" typically describe long-term interests [6] and individually depend on the fact how the user reacts on an incoming stream of information. The user can either select information items (positive kind of filtering) or remove items (negative kind of filtering).

The following state of the art prototype IFS are compared in Table 1.

Scisor
The System for Conceptual Information Summarization, Organisation, and Retrieval (SCISOR) [7] is a prototype system that performs text analysis and question answering on financial news selecting and analysing stories about corporate mergers and acquisitions from an on-line financial service (dow jones). The filter component selects stories about mergers and acquisitions and performs the lexical analysis of names, dates, numbers, and other special inputs. The natural language components, using a combination of language-driven and conceptual analysis, process the take-over texts, identifying key roles such as targets, suitor, and price, as well as other features like financing, company products, or legal complications. The result of this analysis is a single representation of each story that the program adds to a central knowledge base. The conceptual retrieval component accesses information in this knowledge base by analysing questions and matching them to the story representations stored in the knowledge base.

Inference Network
The Inference Net model [8] is based on Bayesian inference networks. The network consists of an object network and a query network. The object network contains object nodes (o_j's) and concept representation nodes (r_m's) that contain a specification of the conditional probability associated with that node $P(r_m|o_j)$, given its set of parent object nodes. An incoming stream of objects is compared to profiles at the same time by computing for every incoming object the probabilities associated with all profile nodes. The object is filtered by removing it from the stream for a given profile or selecting an object for a profile.

Information Lens System

Working with the Information Lens System [9], [10] e-mail senders must specify a finite list of receivers and usually know how these people are. The system permits senders to structure their messages by templates which represent a particular message type and header components. Message receivers are allowed to construct filters - by using a - that find messages matching certain structures. Filters are built up of production rules where the condition part contains selection specifications (e.g. message types or characteristics) for different message fields and the action classifies messages in specific folders or deletes messages. A disadvantage of Lens is that it works well in small environments like companies but not in global environments. Typical senders are often not willing to structure their messages and know very little or nothing who or how many readers receive their messages.

Infoscope System

The Infoscope system [1] is an extension to parts of the Information Lens System and consists of three parts:

(1) A graphically based user interface for accessing news messages. This "browser tool" allows users to learn the existing structure of baskets containing selected e-mail messages. Messages are organized into conversations that consist of a message node, all responses to the message in that node, and similarly all responses to the responses until leaf nodes are reached.

(2) Virtual newsgroups represent areas of special interest to an individual user. The *comp.lang.lisp* node is a Usenet newsgroup and the *comp.lang.lisp.clos* node is a virtual basket containing selected messages filtered from the newsgroups *comp.lang.lisp* and *cu.co.commonloops*. Using this new virtual basket users have a repository for information about *clos* that is displayed using the name semantically attached to that information by the user who defined that basket.

(3) Agents are collections of rule based heuristics that utilize information resulting from the analysis of user behavior to make suggestions to the user. Agents are a feedback mechanism, helping the user modify the message structure based on their own system usage patterns. Relevant messages may be missed by filters because Infoscope does neither analyse the contents of a message nor does the vocabulary in the header field match the vocabulary in the message body.

Iscreen system

Iscreen [4] is a rule-based system for screening text messages. Users provide instructions in the form of rules, which include a list of conditions and actions. Conditions describe values associated with attributes of messages (e.g. who the message is from, what it is about). Actions describe what is to be done with messages that match the specified conditions (e.g. forward, save, delete). A special purpose editor is used to define these rules. The system makes decisions regarding what should be done with them based on rules provided by each of the recipients. To do this it matches conditions specified in rules against the contents and envelopes of messages. Variable references in rules are resolved e.g. if users specify that messages

from their managers are to be saved in an important box, the system is able to determine whether a message is from the users' managers.

LyricTime
LyricTime is a personalized music system [11] in which songs are played at the listener's workstation. A listener profile provides listener specific preference information to the filter. The listener is free to stop and start playing at any time, step forward and backward through the list of selected songs, change the volume and enter his 'mood' (cheerful, romantic, calm, sad, curious). Listener feedback is used to update the profile based on the listener's opinion of songs that have been played.

Pasadena
The Pasadena System [12] is a WAN subscription service that actively and aperiodically queries diverse information sources (e.g. NewsNet news and mailing lists) and maintains a local database of current information items, deleting older information and adding new items continually. Each subscriber has one profile which contains one or more queries. Each query is composed of a comparison text and parameters like list of databases to be searched or pattern for document exclusion/inclusion. The filtering process involves categorisation, exclusion patterns and vector space ranking and text comparing algorithms. Those documents that fall into the categories contained in the query and which do not contain the queries exclusion pattern are ranked by their calculated retrieval status value [15] for each query.

Tapestry
Tapestry [13] is an experimental mail system that supports *collaborative filtering.* Collaborative filtering means that users collaborate to help one another perform filtering by recording their reactions to documents they read. These reactions or annotations e.g. the document was uninteresting, can be accessed by others' filters. Incoming documents are indexed and added to a document store. An annotation store provides storage of annotations associated with documents. A filter component repeatedly runs a batch of user-provided queries over a the set of documents and those documents matching a query are placed in the *little box* of the query's owner. The system applies TQL (Tapestry Query Language - based on SQL) to specify filters as queries. A TQL query is a boolean expression to select those documents that satisfy a user's current need.

Datacycle
The Datacycle system model [14] includes access managers acting on a single large set of shared data items (*storage pump*). They perform retrieval and update operations, complex searches, and support for queries that function as database triggers. Data items are made available to the access managers by repetitive broadcast of the entire storage pump. The broadcast stream is filtered by custom VLSI filters within the access managers.

Criteria	Type of information objects	Constructio n - filter	Rule-based / Query-based	User support for rules/queries	Queries ?
Datacycle	Documents	User		Fuzzy queries	Filter: SQL and Fuzzy queries
Lens	Documents	User	rule-based	Templates	-
Inference Net	Documents	User	query-based	-	-
Infoscope	Documents	User/agents	rule-based	Heuristics	-
Iscreen	Documents	User	rule-based	Conflict detection, explanation	What-If queries
Lyric time	Music	System	rule-based	-	-
Pasadena	Documents	User	query-based	User dialogue	Filter: Queries
Scisor	Documents	User	query-based	-	-
Tapestry	Documents	User/System	query-based	User comments, TQL	Filter: TQL-Queries

Table 1: Information Filtering Systems

IF is closely related to *Information Retrieval* (IR) which is concerned with the representation, storage, organisation, and accessing of information items such as documents [15]. The fundamental problem in IR is to identify the relevant documents from non relevant ones according to a particular user's request. Three domains classifying IR research: indexing, retrieval and evaluation.

The *document representation* or *indexing* process performs the task of assigning information items to documents for purposes of retrieval. An indexing language maps the contents of documents on a textual representation.

The three main retrieval models in IR - *boolean, vectorspace* and *probabilistic model* - differ with respect to the matching process between user queries and document representations.

The *Boolean* model [15] compares queries and document descriptions by exact matching of the index terms with the help of boolean operators. A disadvantage of the exact match model is that the whole document space is divided into two sets of relevant and non relevant documents with a ranking of documents according to a query.

In the *Vector Space* model [15] queries and documents are represented as vectors in a multi-dimensional space and compared with the help of statistical methods e.g. the Cosine, Dice or Jaccard function [16].

The *Probabilistic IR* model estimates the probabilities of a document's relevance by using the Bayes' theorem. The model is based on the *probabilistic ranking principle* (PRP) [17] which states that optimum retrieval is achieved when documents are

ranked according to decreasing values of their probability of relevance with respect to the current query.

Differences between IF and IR are described in Table 2.

	Information Filtering	*Information Retrieval*
System input	dynamic datastream	static database
User goals	long-term periodic desires	short-term intentions
User behavior to incoming data	reacts to	actively searching
Information processing	removing	finding (selection)
Information flow	distribution and organization	representation and organization
Use of the system	repeated	single
Representation of user interests	profiles	queries
Environment	more or less privacy	more or less public
User-groups	undefined	well-defined

Table 2: Information filtering vs. information retrieval

Simple *Keyword Matching* determines whether the user's information interests match the incoming information items of the system.

Before we present the cognitive models incorporated in CIFS, it is most important to state that the process of filtering that the indexing component consists of:

- a lexical scanner,
- a morphological component, and
- a component for generating postings.

3. Cognitive models

In each electronic information source there can be so much detail that the information presented to the reader may be of lower quality and less relevant than traditional approaches. The ability to select relevant information to a user is essential to the viability of such services and requires an individual user model [18]. In our approach we have incorporated the following cognitive aspects to improve our system's ability for filtering e-mails.

Given the diversity of IFS users, the fact that they will not have the same problems or needs, and that the user's level of expertise and interests is likely to change in the cause of time, it is desirable that *profiles* be able to adapt to and support the requirements of individual users.

Monitoring data collection techniques, think-aloud protocols, tape recording of interaction, interviews, and questionnaires are helpful to understand the filtering

process of an individual user [19]. The user's behavior is mapped on the behavior of the system.

The system must also have a model of itself, in order to foresee its possible future actions and thus be able to choose the best way to do. Therefore we apply techniques similar to those used for debugging to give us a trace of system actions during the filtering process. If no user interrupt occurs, in case of a relevant message, the system analyses any observations so that it can choose what to do with the next incoming similar message. Observation is needed to detect system actions during the filtering process. The observation me be passive or active depending on whether it memorizes what happened or makes experiments to find out autonomous new topics that may be interesting for the user. In the first prototype we only use passive observation.
As long as the system has not noticed abnormal behavior, reacts exactly alike without observing itself but simultaneously creates a trace of its actions and results; it may correct what goes wrong immediately or analyze it later. this observation may be continuous or occasional. In our approach we use *continuous observation.*
Similar experimental e-mail assistants like Maxims [20] are learning by continuously "looking over the shoulder" of the user as he/she deals with e-mail [5].

CIFS is a two step learning system. In a first step, the user may specify a catalogue of relevant topics *(interest-domains).* By rating the keywords of each incoming e-mail and assigning them to one or more interest-domains, the system creates a *polarity profile* for each domain out of a set of ratings. In this *(training)*-phase the system learns the basic structure of the user's cognitive style.
Of course, learning has to be an incremental process and the system has to learn on the job. In this *(adaptive)* phase, the *monitor* memorizes all user (re-)actions as situation-action pairs for the genetic algorithm to work with (Chapter 5).
A complete cognitive user model has to represent the user's cognitive style and personality factors, the user's goals and plans, his/her capabilities and preferences, and the user's beliefs and knowledge.

4. Linguistic analysis

Within CIFS linguistic analysis is performed by the indexer/parser module. Additionally, a pre-filter reduces the amount of relevant e-mails. For this purpose, the pre-filter contains a set of keywords and phrases that initially describe the user's current interests. These descriptors weed out e-mails that are not about a *topic* of interest.

4.1. Indexer

As a second step within the filtering process the documents selected from the pre-filter are assigned to the appropriate interest domains on the basis of an indexation module. The document texts are first transformed to a sequential word list. In order to retrieve the individual word boundaries well-approved heuristics are applied and abbreviations, compound words and special formats (e.g. time, date, or currency) are treated correctly by the use of a morphological pre-processor.
After generating the word list, all words that do not contribute to the meaning of the document (e.g. grammatical particles or expletives) are removed on the basis of a stop word list. Now, an index is created which assigns each entry with a list of postings, that is, the positions of its occurrences. In order to check the equality of two words we do not apply an exact string match but we designed a special comparison module which applies simple techniques from morphological analysis to lemmatise the concerned words.
The developed tool is multilingual in the sense that techniques which are valid for any language are strictly separated from language-specific features. Furthermore, the latter are to a high degree mapped to parameters so that the adaptation to a new language can be very easily performed. Although we applied mostly approximate methods, we achieved a very high accuracy without losing too much speed. The following important morphological phenomena are analysed by our comparison module:

- *spelling errors:* Obviously, we do not consider all possibilities for the appearance of spelling errors but restrict ourselves to correct the most frequent error patterns like the insertion of one wrong character, the permutation of two letter or the omission of one character.
- *vowel-gradation:* This is a irregular morphological variation which occurs in many languages during the formation of inflexions (e.g. the ablaut in German) and can be resolved using techniques from spelling error correction.
- *endings and suffixes:* The distinct final part of two words are compared with lists of legal inflexions and derivations which gives also a first evidence for the word category, an information which is essential for later syntactic analysis.
- *elision:* Elision is the omission of the unstressed e-sound, a frequent phenomenon in German and in Scandinavian languages.
- *binding sounds:* Finally, binding sounds tie together the individual parts of the word in the formation of derived or compound words.

For a more detailed discussion of morphological analysis in computational linguistics, especially for inflective languages, we refer to [21].
The resulting document index is matched against the domain descriptors by applying statistical similarity measures adopted from the vector space paradigm of information retrieval [15].

4.2. Parser

Only those documents which are evaluated as relevant to one of the interest domains of the user are subject to a more sophisticated syntactic and semantic analysis. Due to the requirements of information filtering with regard to processing time, natural language analysis can only be performed by *information extraction* and not by *text understanding*. This implies that only fractions of the document text are analysed, the retrieved information is mapped to some target representation, and all other subtle aspects of meaning are left out of consideration [22].
Therefore, a cascaded architecture is required which does not perform a complete linguistic processing for the whole document but narrows the scope by first retrieving text segments of special interest which can then be analysed more carefully. Within our filtering system this task of selecting interesting text segments is performed by the indexing module and is made available to the parser as result of the match against the user's profile.
The contexts of these so-called *trigger words* are further analysed by use of a simple but efficient parsing algorithm in order to detect syntactic constructs (e.g. noun phrases, verb phrases or prepositional phrases). Only straight-forward syntactic analysis is performed, all more tedious linguistic issues are ignored, according to the 'golden rule' of information extraction: 'to do the right amount of syntax, so that pragmatics can take over its share of the load' [22].
Pragmatics is modelled within our knowledge base by the use of frames as conceptual representation scheme. The interesting pieces of information contained in the analysed contexts are mapped to the slots of these frames during semantic analysis [23]. As consequence of our main research objectives, the adaptive behavior of our information filtering tool, this knowledge base cannot be implemented by use of a static structure but on the contrary the represented knowledge must change dynamically in response to changes of the user's interests.
Finally, during discourse analysis all resulting frames are merged to obtain one consistent representation of the contents of an e-mail document. One difficult and important task of this merging process is to unify various interpretations, that is, to eliminate local ambiguities [24]. The final semantic representation is used for valuating the subjective relevance of the document for the user, it models only that part of contents the user is interested in, all other aspects are filtered out.

5. Cognitive Information Filtering System

5.1. Evolutionary Computation

The field of Evolutionary computation (EC) includes research in *genetic algorithms* [25], *evolution strategies* [26], *genetic programming* [27], *artificial life* [28] and several other problem solving strategies, that are based on biological observations, that Charles Darwin called 'The means of natural selection and the survival of the fittest'. These algorithms are thus termed Evolutionary Algorithms (EA) and use

computational models of evolutionary processes as key elements in the design and implementation of problem solving systems [29].
They share a common conceptional base of simulating the evolution of individual structures via processes of *selection, mutation,* and *reproduction*. The processes depend on the perceived performance of the individual structures as defined by the environment. In CIFS we use the EC approach for prioritizing e-mails.

5.2. System Architecture

Based on the previous Sections we designed the system architecture of our Cognitive Information Filtering System displayed in Figure 1. The central component of the system, the *cognitive information filter* contains a population of information objects (words, user usage patterns, phrases) called *e-mail agents*, representing the contents of e-mails. By using a Genetic Algorithm [30] e-mail agents cooperate and compete for correct evaluation of a user's actual interest rating. E-mail agents learn by adjusting their evaluation, thus moving it closer to the user's evaluation. A payoff schema prevents the population from increasing.

The filter is supplemented by the following modules:

• *Pre-filter*
Contains a set of general negative keywords and phrases which weed out e-mails that are not about any topic of interest. Typical examples are sender of the mail or subject categories like 'request to unsubscribe', 'please send me information how to subscribe', 'I am on vacation', a distinct group or a specified time interval.

• *Indexer/Parser*
This component is responsible for the linguistic analysis as described in detail in Section 4. The document text is transformed to an index used for pre-selection of relevant contexts. The resulting text segments are parsed and modelled as frames in order to match them with the corresponding items of the knowledge base.

• *Knowledge Base*
The knowledge base contains the semantic representation of the user profiles which is applied to the assessment of new e-mails. The internal structure consists of frames describing the individual user interests. Their dynamical adaptation is induced by the e-mail agents of the filter component.
• *Monitor*
Records a user's behavior, that is, his/her reaction to incoming e-mails, e.g. deleting, forwarding, storing, replying, printing. Therefore, the monitor provides a feedback mechanism, measuring how effectively the recording of usage patterns predicts current user behavior.

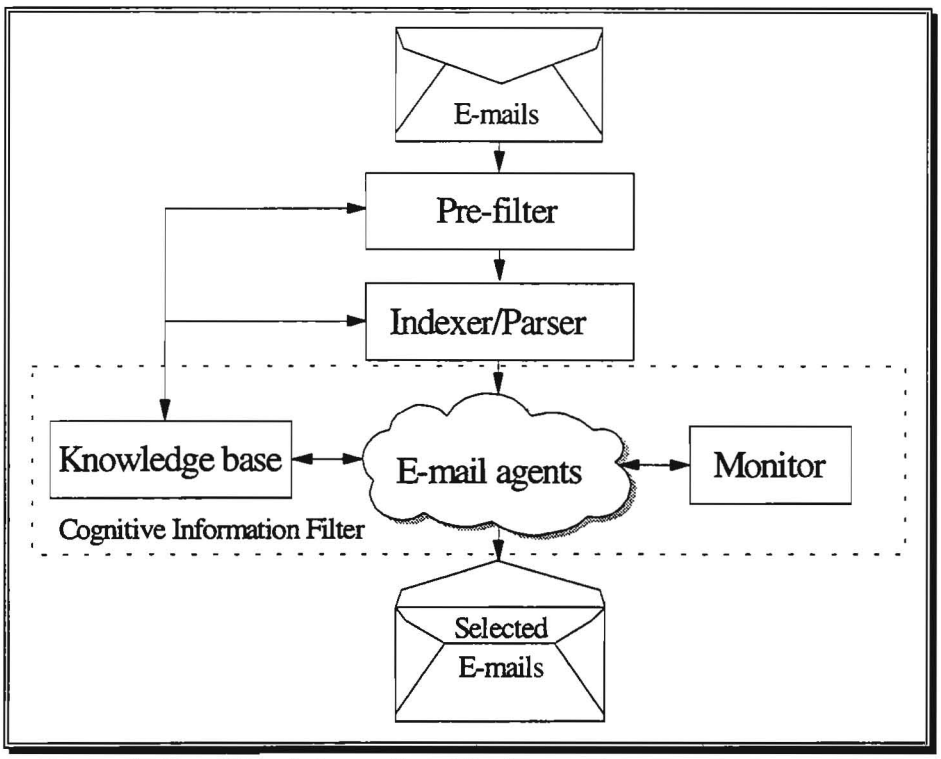


Figure 1: Cognitive information filtering system

5.3 Implementation

In the initial training session of CIFS the user specifies interest domains and categorizes the e-mails accordingly. By use of the indexer module an index is generated out of the e-mail text and presented to the user for rating. He/she may assign one of the following discrete relevance values *very interesting*..[-1,-0.5), *informing*..[-0.5,0), *desinteresting*..0, *boring* .. (0,0.5] and *unexciting* .. (0.5,1] to the individual terms, all others are evaluated with a default value. As additional feature the user is allowed to define synonyms and phrases.

After the collection of a representative sample for each topic (Figure 2), the training session is completed. On the basis of the different assessments, a list of descriptors is created which covers the semantics of the interest domain (Figure 3).

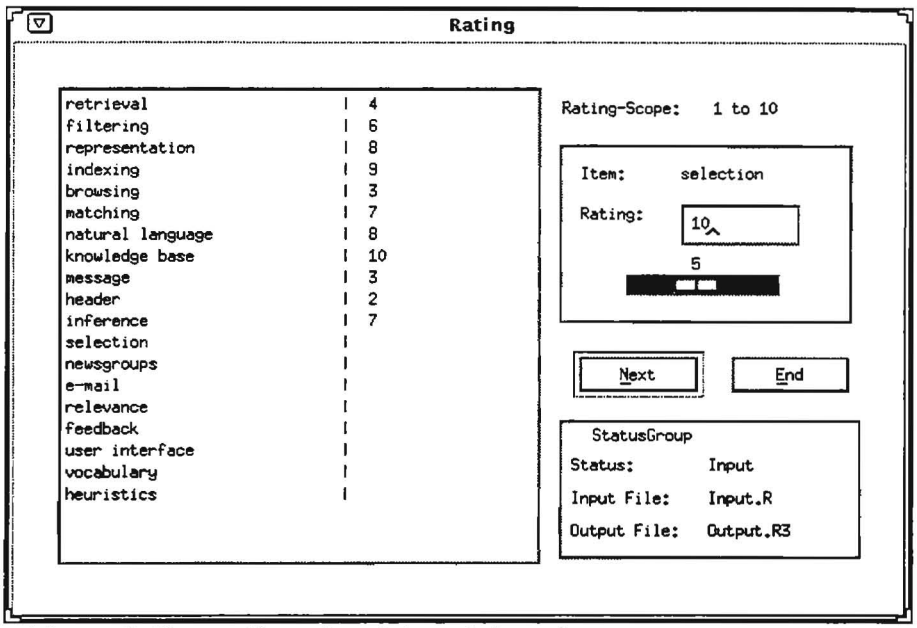


Figure 2: User rating of an interest domain

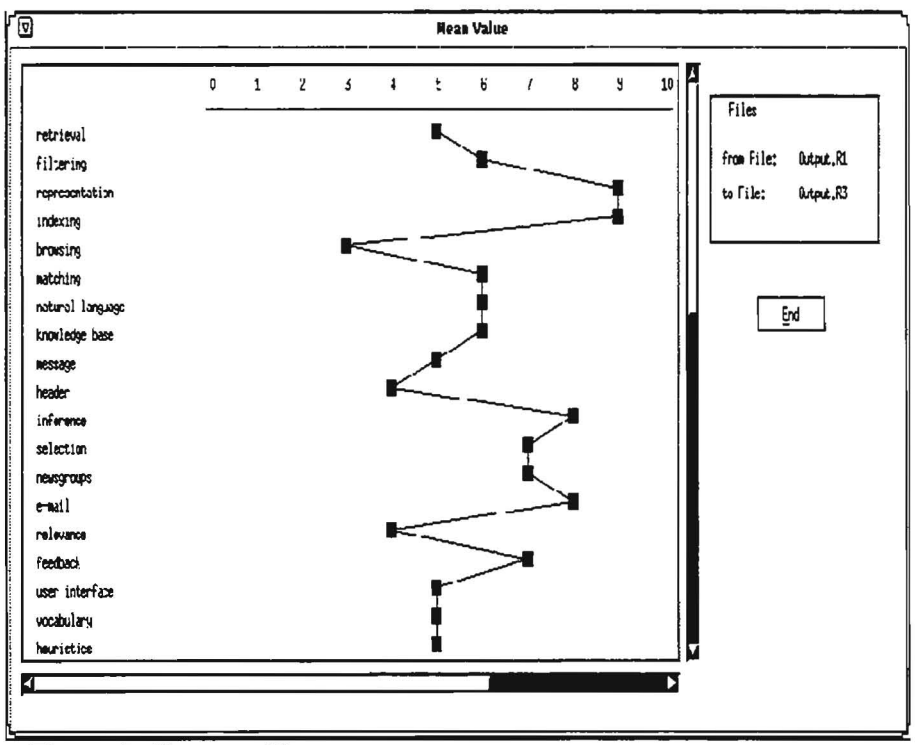


Figure 3: User-profile

The actual filtering process is presented in figure 4.

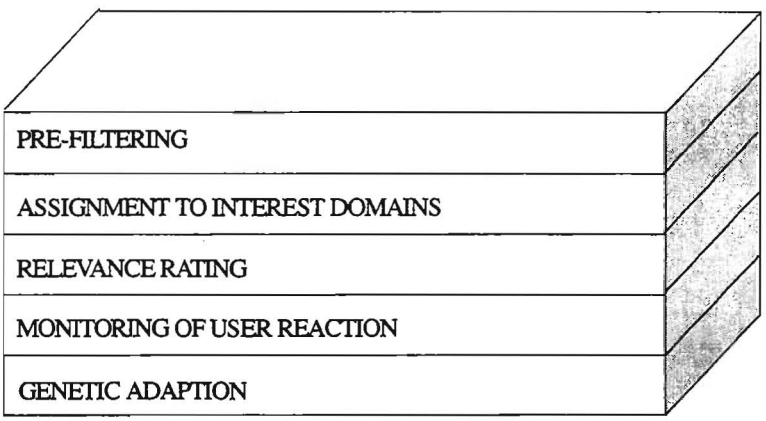


Figure 4: The Filtering Process

1. Pre-filter

E-mail 1 (Figure 5) is deleted because the trigger word *subscribe* is detected.

From genetic-programming-owner@list.Stanford.EDU Wed Jul 20 17:19:41 1994
Date: Wed, 20 Jul 1994 16:32:44 +0100
From: xxxxxxx
To: genetic-programming@cs.stanford.edu
Content-Length: 42
X-Lines: 1
Status: RO

subscribe genetic-programming xxxxxxxx

Figure 5: E-mail 1

E-mail 2 (Figure 6) passes through.

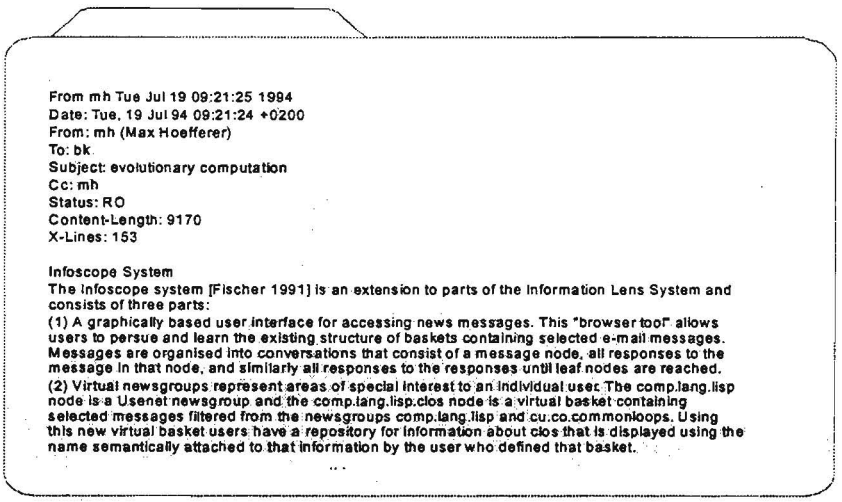


Figure 6: E-mail 2

2. *Assignment to interest domains*
   The descriptors of e-mail 2 are matched against the descriptors of the different domains. The e-mail is assigned to the interest domain which the highest matching score.

3. *Relevance rating*
   Each descriptor of the e-mail is weighted by the corresponding strength from the knowledge base. Based on the list of weighted descriptors the relevance of the e-mail is computed.

4. *Monitoring of user reaction*
   Reactions or sequences of reactions, Looking over the user's shoulder positive (store, forward, print, reply), neutral (view) and negative actions (delete) result in acceptance measure.

5. *Genetic adaptation*
   The learning algorithm in Figure 7 describes the adaptive process that associates user ratings with descriptors to rank incoming e-mails. A population of e-mail agents belongs to a user's domain of interest. The structure of an agent consists of his (initial) strength, a bid [-1,...,+1] and a bid learning rate. The population learns by adjusting their evaluation, moving it closer to the user's evaluation. A payoff schema prevents the population from increasing.

Repeat
(1) For any particular set of e-mail agents, measure the fitness (strength S_i) of each of its descriptions.
(a) After an e-mail is evaluated by a user, agents adjust their fitness:
Calculate for each competing agent the new strength S_i' after payoff:
$S_i' = (S_i + LU)/(1 + L)$ with L .. learning rate and U .. user rating.
(b) Enforce competition among n e-mail agents by the payoff function
$P_i = E/n + 1/n \cdot e_i \cdot F$ F .. fee
error: $e_i = |R - S_i|$ $E = \sum e_i$
(2) After each user evaluation randomly select e-mail agents for reproduction.
Pairwise cross over the fittest agents to produce offsprings.
Until a predefined number of generations is obtained

Figure 7: Genetic adaptation of e-mail descriptions

E-mail agents that run out of strength leave the population for a calculated period of time - at that moment they are not relevant - and get incorporated into the pre-filter. Agents above the average fitness serve

- as 'new' keywords to the prefilter, e.g. the system reacts to the fact that a user has not been interested in a topic for a period of time, or
- remain in the population and get one more chance to be useful.

The knowledge-base contains the semantic representation of the user profiles. Individual interests are mapped to frames. Their dynamic adaptation is induced by e-mail agents.
The monitor is a kind of feedback mechanism, measuring how effectively the history of usage patterns predicts current usage patterns and the probability that an item is needed given the history for such information [31].

6. Conclusion

CIFS supports two general aspects:

- individual user preferences in daily operations with his/her e-mail system, and
- the actual contents of messages that are deemed interesting or uninteresting.

A further advantage is the time saving device to cope with the information overload problem. The current state of implementation is that we completed all modules but do so far without complex frame representations and sophisticated linguistic analysis techniques.

References

[1] G. Fischer, C. Stevens, Information Access in comples, poorly structured information spaces, *Proceedings CHI conference*, pp.63-70, April 1991.

[2] T. Malone, K. Grant, F. Turbak, S. Brobst, M. Cohen, Intelligent Information-Sharing Systems, *CACM* 30(5), pp.390-402, 1987.

[3] H.K. Shuldberg, M. Macpherson, P. Humphrey, J. Corley, Distilling Information from Text: The EDS TemplateFiller System, *JASIS* 44(9), pp.493-507, 1993.

[4] S. Pollock, A Rule-Based Filtering System, *ACM TOIS* 6(3), pp.232-254, 1988.

[5] P. Maes, *Agents that reduce work and information overload, CACM* 37(7), 1994.

[6] N.J. Belkin, W.B. Croft, Information Filtering and Information Retrieval: Two Sides of the Same Coin? *CACM* 35(12), pp.29-38, 1992.

[7] P.S. Jacobs, L.F. Rau, SCISOR: Extracting Information from On-line News, *CACM* 33(11), pp.88-97, 1990.

[8] H.R. Turtle, W.B. Croft, Evaluation of an inference network-based retrieval model, *ACM TOIS* 9(3), pp.187-222, 1991.

[9] T. Malone, K. Grant, F. Turbak, S. Brobst, M. Cohen, Intelligent Information-Sharing Systems, *CACM* 30(5), pp.390-402, 1987.

[10] T. Malone, K. Grant, F. Turbak, K. Lai, D. Rosenblitt, Semistructured messages are surprisingly useful for computer-supported coordination, *ACM TOIS* 5(2), pp.115-131, 1987.

[11] S.Loeb, Architecting Personalized Delivery of Multimedia Information, *CACM* 35(12), pp.39-48, 1992.

[12] H.P. Frei, M.F. Wyle, Retrieval Algorithm Effectiveness in a Wide Area Information Filter, *Proc. 14th Int. Conf. on Res. a. Devel. in Information Retrieval*, pp. 114-122, 1991.

[13] D. Goldberg, D. Nichols, B.M. Oki, D. Terry, Using Collaborative Filtering to Weave an Information Tapestry, *CACM* 35(12), pp.61-70, 1992.

[14] G.E. Herman, G. Gopal, K.C. Lee and A. Weinrib, The Datacycle architecture for very high throughput database systems, *Proceedings of ACM SIGMOD*, ACM, NewYork, 1987.

[15] G. Salton, M.J. McGill, *Introduction to Modern Information Retrieval,* New York: McGraw Hill, 1983.

[16] T. Norault, M. McGill, M.B. Koll, A performance evaluation of similarity measures, document term weighting schemes representations in a boolean system. *Information retrieval Research*, (eds). R.N. Oddy et.al., pp. 57-71, 1981.

[17] S.E. Robertson, The probability ranking principle in IR, *Journal of Documentation,* Vol. 33, pp. 294-304, 1977.

[18] R.B. Allen, User models: theory, method, and practice, *Int. J. Man-Machine Studies* 32, pp.511-543, 1990.

[19] J. Anderson, *Cognitive Psychology and its Implications*, A Series of Books in Psychology, Ed.: R. Atkinson, G. Lindzey, R. Thompson, New York: W.H. Freeman and company, 1985.

[20] Lashkari, Y., Metral, M. and Maes, P., Collaborative Interface Agents, *Proceedings of the Natianal Conference on Artificial Intelligence*, 1994.

[21] W. Winiwarter, A.M. Tjoa. Morphological Analysis in Integrated Natural Language Interfaces to Deductive Databases. *Proceedings of the Fourth International Workshop on Natural Language Understanding and Logic Programming,* Sep. 1993.

[22] J.R. Hobbs, D.E. Appelt, M. Tyson, J. Bear, D. Israel, Description of the Fastus System for MUC-4, *Proc. of the 4th Message and Understanding Conference,* pp.169-177, 1992.
[23] D. Ayuso, S. Boisen, H. Fox, H. Gish, R. Ingria, R. Weischedel, BBN: Description of the PLUM System Used for MUC-4, *Proc. of the 4th Message and Understanding Conference,* pp.268-275, 1992.
[24] A. Meyers, D. de Milster, McDonnell Douglas Electronic System Company: Description of the TexUS System used for MUC-4, *Proceedings of the 4th Message and Understanding Conference,* pp.207-215, 1992.
[25] D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning,* Addison Wesley Inc., Reading, Mass., 1989.
[26] H.P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie,* Birkhäuser Verlag, Basel, Stuttgart, 1977.
[27] J. Koza, *Genetic Programming: On the programming of computers by means of natural selection*, Cambridge, MIT press, 1992.
[28] T.S. Ray, Is it alive, or is it a GA, *Proceedings of the 1991 International Conference on Genetic Algorithms*, pp.527-543, CA, Morgan Kaufmann, 1991.
[29] L.J. Fogel, J.W. Atmar (eds.), *Proceedings of the first Annual Conference on Evolutionary Programming, Evolutionary Programming Society*, San Diego, CA, 1992.
[30] D.E. Goldberg, Genetic and Evolutionary Algorithms Come of Age, *CACM* 37(3), pp.113-119, 1994.
[31] J.R. Anderson, *The Adaptive Character of Thought,* Lawrence Erlbaum Associates: Hillsdale, New Jersey, 1990.

A Term-Based Genetic Code for Artificial Neural Networks

Marek Musial and Tobias Scheffer
(TU Berlin), Schottburger Str. 11 a, D-12305 Berlin, musia@cs.tu-berlin.de
(TU Berlin), Jahnstr. 65, D-12347 Berlin, tobiass@cs.tu-berlin.de

Abstract

We developed a well-structured term-based language for the structural specification of artificial neural networks. The language achieves an intuitive and compact representation even for very large networks, making it interesting as an input language for network simulators. Since it describes neural networks on a logical level, it is very well suited as a "genetic code" for the optimization of network architectures by genetic algorithms, allowing well-controllable mutation operators and a powerful crossover operation that is able to recombine *functional blocks* of any shape instead of destroying them. We define the language formally, give examples of its application and present some results of its use as a genetic code for finding network architectures.

1 Introduction

Although it can be proven that there exists an artificial neural network (ANN) approximating every continuous, bounded function (e.g. Theorem of CYBENKO [1]), no universal algorithm that determines the network's parameters is known. Given an architecture of an ANN, i.e. the number of units and their connectivity, the weights can be adapted by several training algorithms, e.g. *back-propagation* [2].

Yet, the architecture decides whether a set of weights can be found by the BP algorithm. Genetic algorithms (GOLDBERG [3]) can be used to find or optimize network structures. A similar optimization strategy is *evolution strategy* (RECHENBERG [4] [5]). Evolution strategy is better understood and a more extensive theory is available, parts of which can be applied to the similar genetic optimization strategy.

There are two possible ways of optimizing ANNs by means of genetic algorithms: Architecture and weights can be optimized simultaneously (e.g. KOZA and RICE [6]), or only architecture is optimized by a genetic algorithm, while the weights are adapted by a local search strategy like *back-propagation* for each individual (e. g. HARP [7]). Due to the results of the theory of evolution strategy, we focused on the second approach.

1.1 Genetic Algorithms

Genetic algorithms optimize a population of individuals (e.g. ANNs) by (1) evaluating the quality of each individual with respect to the problem, (2) reproduction of individuals with a rate proportional to the quality and (3) mutating and sexually recombining the new individuals.

Although it is possible to define the mutation operators on the phenotype level (UTECHT and TRINT [8]), the individuals are usually represented by a genetic code. The mutation and crossover operators are then defined on that code. This representation of an individual is called its genotype. To evaluate the quality of each member of the population, a mapping from the genotype to the phenotype has to be defined, which can be considered the semantics of the genetic code.

1.2 Genetic Representation

To achieve efficient optimization by genetic algorithms, the principle of strong causality has to be obeyed by the mutation operators as well as by the genetic code, i.e. the mutants of each individual have to be most probably similar to their parent in phenotype, and similar genotypes should lead to phenotypes of similar behaviour, which means smoothness of the quality space.

Various forms of genetic representation of ANNs have been invented. They can be divided into *direct encoding schemes*, which encode complete and detailed information about the network's architecture, and *indirect encoding schemes*, which encode either rules for the generation of the phenotype [9] or only those parts of the network's architecture that are considered to be of relevance.

Direct encoding schemes, the simplest of which is the direct encoding of the interconnection matrix, tend to result in large and redundant genotypes. The term "redundant" here alludes to a crucial characteristic of ANNs: There is only a relatively loose causality between an ANN's architecture and its suitability for a given task, i.e. its quality. The effect of small variations in the network's architecture is in many cases completely concealed by the stochastic noise in the quality space, which is a consequence of the random initial weights in back-propagation training.

Indirect encoding schemes sometimes limit the set of

possible solutions a priori, and the law of strong causality between the genotypes and the phenotypes is often violated, e.g. two similar sets of rules may derive completely different network architectures.

2 The Approach

Our genetic code is based on the idea that the genotype should reflect the *logical structure* of the network as it would be seen by a human engineer. An ANN may consist of subnets or *organs*, self-contained functional units performing a special task. Identical or similar parts can be incorporated several times in the network. If such information about the network's structure is available in the genotype, mutation operators should be possible that vary the structure in steps of well-controllable size, so that effective structural changes are possible while the complete destruction of the network remains improbable.

In addition, we want to be able to define a crossover operator that simulates the advantages of biological, sexual reproduction. That is, it should be capable of combining independently developed subnetworks, like organs, in a common descendant. Therefore, it should in most cases extract functionally correlated parts of a network, including those extending over several layers.

3 The Genetic Code

To achieve this, we designed a recursive, structured, term-based language.

3.1 Syntax

The language is the smallest set T that satisfies:

$n \in T$,	iff $n \in \mathcal{N}$
$\mathsf{par}(t_1, \ldots, t_n) \in T$,	iff $t_1, \ldots, t_n \in T$
$\mathsf{ser}(t_1, \ldots, t_n) \in T$,	iff $t_1, \ldots, t_n \in T, n > 1$
$\mathsf{mul}(n, t) \in T$,	iff $n \in \mathcal{N} \setminus \{1\}, t \in T$
$\mathsf{smul}(n, t) \in T$,	iff $n \in \mathcal{N} \setminus \{1\}, t \in T$
$\mathsf{tf}(r) \in T$,	iff $r \in [0, 1]$

Where $\mathcal{N}$ is the set of positive natural numbers.

3.2 Informal Semantics

A term consists of a function symbol followed by a list of arguments. Depending on the function symbol, the arguments may also be terms, and their semantics are networks. The function symbol determines the arrangement of the argument networks to obtain the resulting ANN. The argument networks may lie parallel or in series:

- "n"-terms anchor the set. They represent n *parallel units* not having any connections between them.
- The arguments of a $\mathsf{par}(t_1, \ldots, t_n)$ term lie *parallel* and are structured networks themselves. There are no edges from any unit within an argument to any unit within another argument of the same par term.
- The arguments of a $\mathsf{ser}(t_1, \ldots, t_n)$ term lie *in series* and are networks again. If neither argument t_i nor argument t_{i+1} is of the type $\mathsf{tf}(r)$, then the output units of block t_i are completely connected to the input units of block t_{i+1}.
- $\mathsf{mul}/\mathsf{smul}(n, t)$ terms allow the *"cloning"* of a block n times and correspond to a par term with n times the argument t. In the smul case, weights are shared by the instances of t. Thus, *feature detectors* can be described.
- The *topological filter* $\mathsf{tf}(r)$, placed between two arguments of a ser term, avoids a complete interconnection between these blocks. Instead, a specific connection structure is generated, with the number of connections actually drawn determined by r. The resolution of topological filter is described in detail in section 3.3.

3.3 Semantics

To depict terms and to apply a set of transformation rules, we translate them into diagrams. Diagrams are graphs with special types of nodes and edges. We consider a graph to be a six-tuple $(V, E, I^V, I^E, i^v, i^e)$, where V is a set of vertices (nodes), $E \subseteq V \times V$ a set of edges, I^V the alphabet of node types, I^E the alphabet of edge types; $i^v : V \to I^V$ assigns each node a type and $i^e : E \to I^E$ assigns each edge a type. The types of edges for a diagram are:

1. *connected-to*: units and blocks (i.e.: terms) can be connected to other blocks or units. We draw *connected-to*-edges as solid lines; but although they are directed edges, we do not care to draw arrows, because the direction is always from the bottom to the top. Such an arrangement is always possible, because we specify *feed-forward* networks only.
2. *member-of*: Since diagrams represent structured terms, blocks (or units) can be incorporated in other blocks. We do not draw these edges at all, instead we draw all members *directly into* their parent blocks.

Each node is assigned a type. The (infinite) alphabet of types is

$$\mathcal{N} \times \{\bullet, \mathsf{par}, \mathsf{tf}(r)\}, \quad r \in [0, 1]$$

The first component of the type indicates the position of the block within its parent, the second determines whether the block is a unit (which is a block, too), a structured non-unit block or a topological filter. If it is a topological filter, the real-valued filter-factor is given. The function $\Pi : V \to \mathcal{N}$ maps every block to its position within its parent. It is defined by

$$\Pi(n) = \pi, \qquad \text{if } i^v(n) = (\pi, \_)$$

Figure 1 shows the translation of terms into diagrams. The translation function can be defined as follows:

We first append unique identifiers to every block of a term, i.e. the rule

$$f(a_1, \ldots, a_n) \longrightarrow f(a_1, \ldots, a_n) : \mathrm{id},$$

with id denoting a unique identifier, is applied wherever possible. As an example, the term $\mathsf{par}(\mathsf{ser}(3, 4), 5)$ is transformed into $\mathsf{par}(\mathsf{ser}(3 : \mathrm{id}_1, 4 : \mathrm{id}_2) : \mathrm{id}_3, 5 : \mathrm{id}_4) : \mathrm{id}_5$. Then the δ function translates the named term into a graph. δ receives a named term and a relative position

of that term within its parent as parameters. At the top level, $\delta(t, \omega)$ indicates that there is no parent for the whole term.

$$\delta(n : \mathrm{id}, \pi) =$$
$$\quad \delta(\mathsf{par}(\bullet : \mathrm{id}_1, \ldots, \bullet : \mathrm{id}_n), \pi)$$
$$\delta(\bullet : \mathrm{id}, \pi) =$$
$$\quad (V, E, I^V, I^E, i^v, i^e), \quad \text{where}$$
$$\quad V = \{\mathrm{id}\},$$
$$\quad E = \{(\mathrm{id}, \mathrm{id})\},$$
$$\quad I^V, I^E \text{ defined as above},$$
$$\quad i^v = (\mathrm{id} \mapsto (\pi, \bullet)),$$
$$\quad i^e = \{((\mathrm{id}, \mathrm{id}) \mapsto \textit{member of})\}$$
$$\delta(\mathsf{par}(t_1 : \mathrm{id}_1, \ldots, t_n : \mathrm{id}_n) : \mathrm{id}) =$$
$$\quad (V', E, I^V, I^E, i^{v'}, i^e), \quad \text{where}$$
$$\quad (V, E, I^V, I^E, i^v, i^e) = \bigcup_i \delta(\mathrm{id}_i, i),$$
$$\quad V' = V \cup \{\mathrm{id}\},$$
$$\quad i^{v'} = i^v \cup \{(\mathrm{id} \mapsto (\pi, \mathsf{par}))\};$$
$$\delta(\mathsf{ser}(t_1 : \mathrm{id}_1, \ldots, t_n : \mathrm{id}_n) : \mathrm{id}, \pi) =$$
$$\quad (V, E', I^V, I^E, i^v, i^{e'}), \quad \text{where}$$
$$\quad (V, E, I^V, I^E, i^v, i^e) = \bigcup_i \delta(\mathrm{id}_i, i),$$
$$\quad E' = E \cup \bigcup_i^{n-1} \{(\mathrm{id}_i, \mathrm{id}_{i+1})\} \cup \bigcup_i^{n} \{(\mathrm{id}_i, \mathrm{id})\},$$
$$\quad i^{e'} = i^e \cup \bigcup_i^{n-1} ((\mathrm{id}_i, \mathrm{id}_{i+1}) \mapsto \textit{connected to})\}$$
$$\qquad \cup \bigcup_i^{n} \{((\mathrm{id}_i, \mathrm{id}) \mapsto \textit{member of})\};$$
$$\delta((\mathsf{s})\mathsf{mul}(n : \_, t : \mathrm{id}), \pi) =$$
$$\quad \delta(\mathsf{par}(t : \mathrm{id}_1, \ldots, t : \mathrm{id}_n), \pi);$$
$$\delta(\mathsf{tf}(r) : \mathrm{id}, \pi) =$$
$$\quad (V, E, I^V, I^E, i^v, i^e), \quad \text{where}$$
$$\quad V = \{\mathrm{id}\}, \quad E = \emptyset,$$
$$\quad I^V, I^E \text{ defined as above}$$
$$\quad i^v = \{(\mathrm{id} \mapsto (\pi, \mathsf{tf}(r)))\}, \quad i^e = \emptyset$$

where $\mathrm{id}_1, \ldots, \mathrm{id}_n$ are new, unique identifiers, derived from the identifier id, and the union of graphs is defined as commonly expected:

$$(V_1, E_1, I^V, I^E, i_1^v, i_1^e) \cup (V_2, E_2, I^V, I^E, i_2^v, i_2^e)$$
$$= (V_1 \cup V_2, E_1 \cup E_2, I^V, I^E, i_1^v \cup i_2^v, i_1^e \cup i_2^e)$$

Figure 2 shows how diagrams can be viewed as graphs. ser terms are drawn from the bottom (input) to the top (output), n-terms are drawn as n dots, the filter-values of topological filters are written into their blocks. For

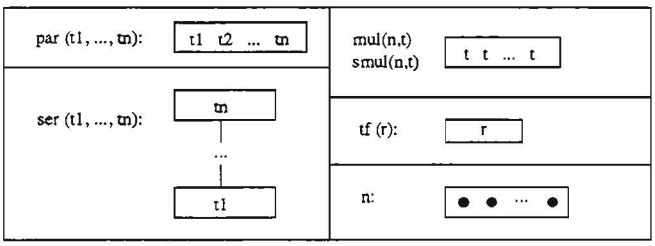


Figure 1: The diagram language

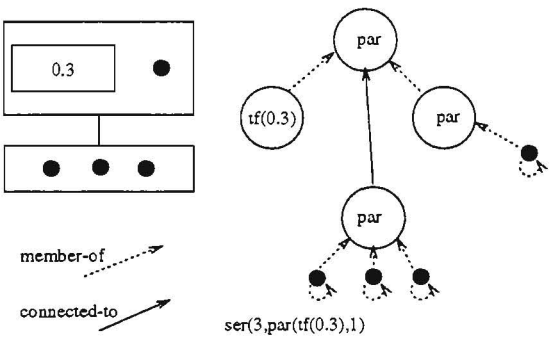


Figure 2: Viewing diagrams as graphs

mul/smul and par terms, the argument terms are drawn into their parents, mul/smul terms are expanded. Note that units are members of themselves.

After this has been done, the *connected-to*-edges need to be resolved, i.e. the member blocks have to get incoming and outgoing edges as well. This is accomplished by a set of diagram transformation rules, which are depicted in figure 3.

R1 This rule determines that topological filters with filter-value 1.0 can be inserted anywhere into a ser term. To guarantee the termination of the transformation process, this rule may only be applied if the "neighbours" are not of the type tf.

R2 defines the connection of two filters in series. The filter with the larger filter-value is discarded.

R3 resolves parallel filters. The filter with the least filter-value is discarded. Thus, edges "can more easily pass through a set of parallel filters", while serial filters block as well as the most powerful of them.

R4 determines what happens to multiple blocks sharing a common filter. The filter is applied to every possible combination of upper and lower neighbours.

R5 defines the resolution of edges. Edges can be resolved if two non-elementary blocks are connected via a filter. If the filter is missing, it can be inserted by R1. The filter and the old edges are removed and each of the n member blocks of the upper neighbour (target) is connected to some of the m lower member blocks (source). From each target block at position π_t, $r \cdot m$ connections to some of the source blocks are established. The connections are grouped around a centre at source position $\frac{\pi_t \cdot m}{n}$. So source blocks "on

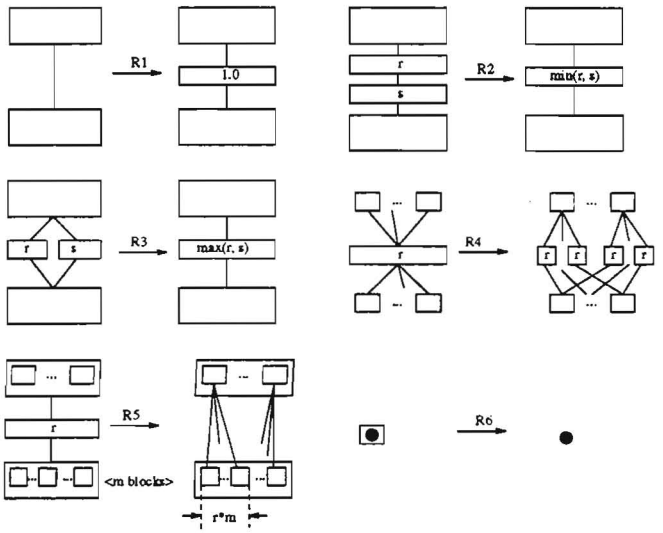


Figure 3: The transformation rules

the left-hand side" are preferably connected to target blocks "on the left-hand side". If $r \cdot m$ is not a natural number, some of the target blocks on the right-hand side get one edge more than their peers on the left so that a total of $n \cdot r \cdot m$ edges is obtained to the nearest whole number.

R6 Blocks containing only a single unit are identified with that unit. This means, all incoming and outgoing edges of such a block are passed to the unit.

The rules R_1 through R_6 can be formalized as follows:

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_1} (V', E', I^V, I^E, i^v, i^e)$$
$$\exists s, t \in V : (s,t) \in E \wedge (s \mapsto (\pi, \mathsf{par})) \in i^v$$
$$((s,t) \mapsto \textit{connected to}) \in i^e$$
$$\wedge V' = V \cup \{\mathrm{id}\}$$
$$\wedge i^{v'} = i^v \cup \{(\mathrm{id} \mapsto \mathsf{tf}(1.0))\}$$
$$\wedge E' = E \setminus \{(s,t)\} \cup \{(s,\mathrm{id}), (\mathrm{id},t)\}$$
$$\wedge i^{e'} = i^e \setminus \{((s,t) \mapsto x)\}$$
$$\cup\{((s,\mathrm{id}) \mapsto \textit{connected to})\}$$
$$\cup\{((\mathrm{id},t) \mapsto \textit{connected to})\}$$

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_2} (V', E', I^V, I^E, i^{v'}, i^{e'})$$
$$\exists s, t, f_1, f_2 \in V : (f_1 \mapsto (\pi_1, \mathsf{tf}(r_1))) \in i^v$$
$$\wedge (f_2 \mapsto (\pi_2, \mathsf{tf}(r_2))) \in i^v$$
$$\wedge t_e = \{(s,f_1), (s,f_2), (f_1,t), (f_2,t)\} \subseteq E$$
$$\wedge i^e(t_e) = \{\textit{connected to}\}$$
$$f_{min} = f_1, \text{ if } r_1 < r_2, \text{ else } f_2$$
$$f_{max} = f_2, \text{ if } r_1 < r_2, \text{ else } f_1$$
$$\wedge V' = V \setminus \{f_{max}\}$$
$$\wedge i^{v'} = i^v \setminus \{(f_{max} \mapsto x)\}$$
$$\bar{E} = \{(s,f_1), (f_1,f_2,), (f_2,t)\}$$
$$\wedge E' = E \setminus \bar{E} \cup \{(s,f_{min}), (f_{min},t)\}$$
$$\wedge i^{e'} = i^e \setminus \{(e \mapsto i^e(e)) | e \in \bar{e}\}$$
$$\cup\{((s,f_{min}) \mapsto \textit{connected to})\}$$

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_3} (V', E', I^V, I^E, i^{v'}, i^{e'})$$
$$\exists s, t, f_1, f_2 \in V : (f_1 \mapsto (\pi_1, \mathsf{tf}(r_1))) \in i^v$$
$$\wedge (f_2 \mapsto (\pi_2, \mathsf{tf}(r_2))) \in i^v$$
$$\wedge t_e = \{(s,f_1), (s,f_2), (f_1,t), (f_2,t)\} \subseteq E$$
$$\wedge i^e(t_e) = \{\textit{connected to}\}$$
$$f_{max} = f_2, \text{ if } r_1 < r_2, \text{ else } f_1$$
$$f_{min} = f_1, \text{ if } r_1 < r_2, \text{ else } f_2$$
$$\wedge V' = V \setminus \{f_{min}\}$$
$$\wedge E' = E \setminus \{(s,f_{min}), (f_{min},t)\}$$
$$\wedge i^{v'} = i^v \setminus \{(f_{min} \mapsto \textit{type})\}$$
$$\wedge i^{e'} = i^e \setminus \{((s,f_{min}) \mapsto \_), ((f_{min},t) \mapsto \_)\}$$

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_4} (V', E', I^V, I^E, i^{v''}, i^{e''})$$
$$\exists f \in V : i^e(f) = (\pi, \mathsf{tf}(r))$$
$$\wedge \exists S, T \subset V : s \in S \Leftrightarrow (s,f) \in E$$
$$\wedge t \in T \Leftrightarrow (f,t) \in E$$
$$\wedge V' = V \setminus \{f\} \cup \{f_1, \ldots, f_{|S| \cdot |T|}\}$$
$$s \in S \wedge t \in T \Leftrightarrow (s, f_{\Pi(s) \cdot \Pi(t)}) \in E^+$$
$$\wedge (f_{\Pi(s) \cdot \Pi(t)}, t) \in E^+$$
$$\wedge E' = E \setminus (S \times \{f\}) \setminus (\{f\} \times T) \cup E^+$$
$$\wedge (a \mapsto \textit{type}) \in i^v \wedge a \in V'$$
$$\Leftrightarrow (a \mapsto \textit{type}) \in i^{v'}$$
$$\wedge i^{v''} = i^{v'} \cup \bigcup_{a \in V' \setminus V} \{a\}$$
$$\wedge ((a,b) \mapsto \textit{type}) \in i^e \wedge (a,b) \in E'$$
$$\Leftrightarrow ((a,b) \mapsto \textit{type}) \in i^{e'}$$
$$\wedge i^{e''} = i^{e'} \cup \bigcup_{(s,t) \in E' \setminus E} \{((s,t) \mapsto \textit{connected to})\}$$

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_5} (V', E', I^V, I^E, i^{v'}, i^{e''})$$
$$\exists p_s, p_t, f \in V : (p_s \mapsto \mathsf{par}) \in i^v$$
$$\wedge (p_t \mapsto \mathsf{par}) \in i^v$$
$$\wedge (f \mapsto \mathsf{tf}(r)) \in i^v$$
$$\wedge S, T \subset V : (s \in S \Leftrightarrow ((s,p_s) \mapsto \textit{member of}) \in i^e$$
$$\wedge \nexists s' \in V : i^e((s,s')) = \textit{connected to}$$
$$\wedge i^e(s', p_s) = \textit{member of})$$
$$\wedge (t \in T \Leftrightarrow ((s,p_t) \mapsto \textit{member of}) \in i^e$$
$$\wedge \nexists t' \in V : i^e(t',t) = \textit{connected to}$$
$$\wedge i^e(t', p_t) = \textit{member of})$$
$$\wedge V' = V \setminus \{f\}$$
$$\wedge E' = E \setminus \{(p_s,f), (f,p_t)\} \cup \bigcup_{t \in T} P(S,T,t,r)$$
where $(s,t) \in P(S,t,r) \Leftrightarrow \Pi(s) \in$

$$\left[\frac{\Pi(t)\cdot|S|}{|T|} - \frac{\lfloor r\cdot|T|\rfloor + v}{2},\right.$$
$$\left.\frac{\Pi(t)\cdot|S|}{|T|} + \frac{\lfloor r\cdot|T|\rfloor + v}{2}\right]$$

where $v = 1$ if $\Pi(t) \geq |S| - \lceil |S|\cdot r\cdot|T|\rceil - |S|\cdot\lfloor r\cdot|T|\rfloor$, else 0

$$\wedge i^{v'} = i^v \setminus \{(f \mapsto \_)\} \cup \bigcup_{a\in V'\setminus V} (a \mapsto \textit{connected to})\}$$

$$\wedge i^{e'} = i^e \setminus \{((p_s, f) \mapsto \_), ((f, p_t) \mapsto \_)\} \cup \bigcup_{(a,b)\in E'\setminus E} ((a,b) \mapsto \textit{connected to}\}$$

$$(V, E, I^V, I^E, i^v, i^e) \xrightarrow{R_6} (V', E, I^V, I^E, i^{v'}, i^e)$$

$$\exists p, u \in V : i^v(p) = (\pi, \mathsf{par}) \wedge i^v(u) = (\pi, \bullet)$$
$$\wedge |\{u' \in V : |((u', p) \mapsto \textit{member of})| = 1\}$$
$$\wedge V' = V \setminus \{u\}$$
$$\wedge i^{v'} = i^v \setminus \{(u \mapsto (\pi, \mathsf{par}))\} \cup \{(u \mapsto (\pi, \bullet)\}$$

Note that the rules R_1 through R_6 are relations rather than functions, for a rule might "match" several parts of a diagram.

The rules are applied to the diagram by the following strategy:

```
do
    if a rule of R1, R2, R3, R4, R6 is applicable
        apply it
    else apply R5 (if possible)
until no more rules are applicable
```

When no more rules are applicable, all non-elementary blocks can be discarded. The resulting neural network is the largest subgraph with only vertices of the type $(\pi, \bullet)$.

Those units located "at the bottommost edge" of the network are its input units. A unit is an input unit if and only if its related n-term is located in the *first* argument of each surrounding ser term. The output units are identified analogously, considering only the *last* ser-arguments.

The network is uniquely determined by the term, i.e. the transformation process is well-defined. This can be proven since each two rules (except R_5) are commutative.

3.4 Understanding Everything

We shall now translate an example term into a network. The term we will focus on at first is $\mathsf{ser}(1, \mathsf{par}(\mathsf{ser}(2, \mathsf{tf}(0.5), 2), \mathsf{tf}(1.0)), 1)$. We first have to translate it into a diagram. The first function symbol is a ser having three arguments. Following figure 1, the arguments have to be put into series. The first and last arguments are "n"-terms, they represent one unit each. Figure 4a shows the result we have got so far. We now have to translate the par term between the input and

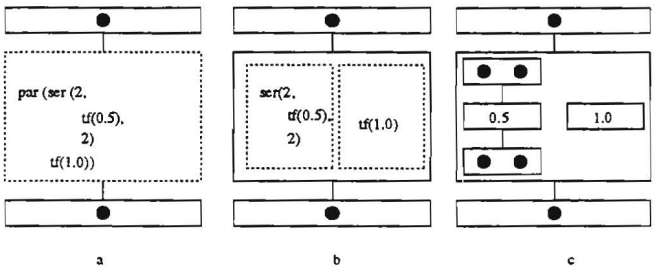


Figure 4: An example diagram

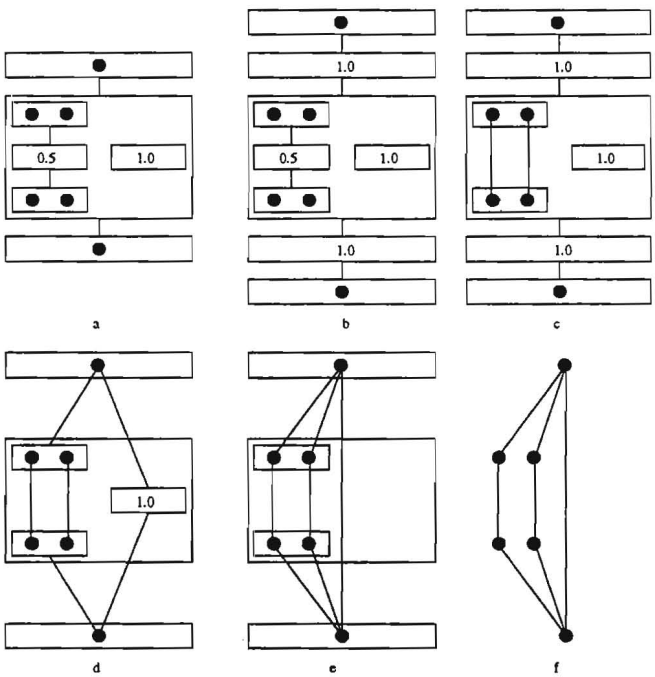


Figure 5: An example translation

output layers. The par term has two arguments, another ser term and a tf term. With respect to figure 1 we have to place them in parallel. Figure 4b depicts the result achieved by now. Finally, the last ser term has to be resolved. We get two n-terms and a filter between them. Figure 4c shows the complete diagram.

Now, the transformation rules R_1 through R_6 have to be applied. Since we can see two edges with no filter on either side of them, we apply R_1 twice and get figure 5b. Now we cannot apply any rule except R_5. So we start resolving the tf(0.5) edge. The source and target blocks contain two members each. That is, a maximum of $2 \cdot 2$ edges could be drawn. A filter-value of 0.5 tells us now to draw only $4 \cdot 0.5 = 2$ edges instead. That makes one edge per member of the target block. The *projection set* P_1 of the left target unit has cardinality 1 and aims at source position $\frac{1\cdot 2}{2} = 1$. The projection set P_2 of the right target unit aims at source position $\frac{2\cdot 2}{2} = 2$. Thus, the left target unit is connected to the left source unit while the right one is connected to the right source unit. The result is shown in figure 5c.

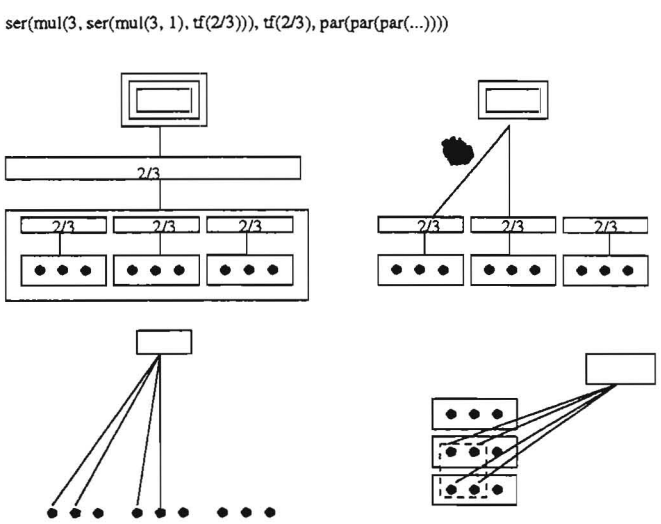


Figure 6: A simple feature detector

Now we resolve the topmost and the bottommost edges. The number of members of the middle block visible to the input layer is two: the bottommost block containing two units and the tf(1.0) filter. The input layer consists of one unit. The filter-value is 1.0, so the full number of two edges have to be drawn. The same holds for the output layer.

Finally, the remaining filter has to be eliminated. Remember that each unit is a member of itself (see section 3.3), so the two units involved are connected via an edge when the filter is resolved. Now the non-elementary blocks are discarded and the result is an artificial neural network.

Topological filters can be used to extract data from a multidimensional receptive field, in which the arrangement of filters defines the dimensionality assumed for the data. We shall now glance at a simple implementation of a *feature detector* that takes a subset of units ordered in a square as input and performs an operation on them. By the same technique, arrays of feature detectors can be implemented that perform identical operations on different but possibly overlapping sections of the input data. Figure 6 shows the resolution of the network. The large filter means "take only two thirds of the input segments", the array of smaller filters means "of each segment (line), only two thirds are of interest". Using mul/smul terms, overlapping detector arrays of any complexity can be expressed. See section 4 for a non-trivial example.

3.5 Bug-Fixing

Not every term results in a suitable network. The following conditions can arise that we regard as context-errors:

- If some filter-values are too small, some units may be *isolated*, i.e. do not have input or do not have output[1].
- The number of input or output units in the derived network will in most cases differ from the number required by the task to be solved.
- Topological filters in the input and output layers do not make sense at all.

Since the above error conditions can be most easily detected during the translation of a term, "bug-fixing" is performed as a side effect of the translation process, mainly by permanently *changing* the term into a similar context-correct one. The following sections describe each case in detail.

3.5.1 Isolated Units

After the transformation into a network has been finished in principle, some units may lack fan-in or fan-out. This is always caused by one or more topological filters carrying too small factors. Hence, we must find one or more filters connected to a parent of the isolated node and ensure that

- these filters themselves have respectively fan-in or fan-out,
- if so, their factors must let pass at least one link per member of the parent of the isolated mode.

This is accomplished by a recursive algorithm, making use of the properties of the filter resolution rule (R_5 in section 3.3). This algorithm is outlined here:

scale ($v \in V$)
  if v is *connected to* a non-tf node $s \in V$
    let $f_{new} \leftarrow \frac{1}{m}$, where s has got m *members*
  else if v is *connected to* a tf node $t \in V$
    scale (f)
    increase f's filter-factor to f_{new}
    revert those parts of the translation process that have been affected by the factor of f, translation has to be resumed here
  else (v has no *connected-to*-edges)
    scale (parent (v)) (where v is *member of* parent (v))
    exit!

Note that the isolation of the node v may not be overcome after a single iteration of scale(v) followed by a partial retranslation if there is more than one offending filter. But the algorithm assures that, if called for an isolated node v, at least one additional *connected-to*-edge will be established afterwards, which will guarantee the correctness of the algorithm.

3.5.2 Number of Input/Output Units

Although there is nothing fundamentally wrong with networks containing any number of input or output units, we must fix the number of input and output units for a given task. We should not restrict the input and output layers to be represented by n-terms of the required value, for the input and output layers have to carry topological information. Hence, mutations and crossovers can affect the number of input or output units, which have to be *rescaled* in return by changing the n-values and mul-/smul-factors in the input and output

[1] Input units are always considered to "have input" and output units never lack output.

regions of the term appropriately. Thus, we try to adjust the *size* of the input and output layers with minimal effect on their *structure*.

Since the number of units in every mul/smul subnetwork is a product, it may happen that a given number of, e.g., input units cannot be reached – especially if it is a prime number. Therefore we split the rescaling process into two phases (let us consider input units only for simplification):

1. **raw-scaling**: All n-values and mul-/smul-factors affecting the number of input units are changed proportionally until the network has *slightly more* input units than required. Formally, *slightly more* means that no single n-value or mul-/smul-factor can be further decreased without resulting in *less* input units than required. All changes performed in the raw-scaling phase are permanently incorporated into the term, so that this process need not be repeated as long as the term remains unchanged.
2. **fine-scaling**: When the term is translated into the diagram graph (see 3.3), all mul/smul terms are expanded as if they were corresponding par terms. After this expansion, no more multiplications take place and the superfluous units are discarded by simply reducing some of the n-values by one. Fine-scaling affects the derived network, ensuring the correct number of input and output units, but it obviously cannot have any effect on the term.

As an example, consider the term mul(3, 10) and a target number of 13 units. Since 13 is prime, raw scaling cannot reach 13 exactly, but it will result in the term mul(2, 7) indicating 14 units. After expansion, the diagram looks like the one for par(7, 7), which can be fine-scaled to par(7, 6) yielding 13 units as required.

Input and output scaling would be rather problematic if the number of input units and that of output units were not independent, i.e. if a single atomic change to the term were able to affect both the number of input and the number of output units. We prevented this by restricting the individuals in our GA experiments to having a ser symbol at the topmost recursive level.

3.5.3 Filters in the Input/Output Layer

Topological filters in the input and output layers are permanently removed from the term. This might leave an "empty" parent symbol, which has to be removed as well, and so on. If less than two independent layers for input and output persist (in the worst case a term contains nothing but filters), the whole term is replaced by ser(n_i, n_o) where n_i and n_o are the required number of input and output units.

3.6 The Derivation of Shared Links

Obviously, the diagram transformation process does not cover the derivation of *shared links*, i.e. links that have a common weight parameter in BP training. Shared links are specified by the use of smul terms. They are derived in an additional pass after the diagram transformation has been completed, and for every pair of units u, v it is already known if there will be a link (u, v) between these units or not. So we just have to choose some groups of links and mark their respective members as *shared*. We now define how this is achieved.

First of all, the smul subterms within a term set up an equivalence relation $\equiv \subset V \times V$ on the set of blocks, including units and filters, in the resulting diagram graph. $v_1 \equiv v_2$ holds if and only if

- either v_1 and v_2 are instances of the argument of the *same* smul block
- or else the parent blocks p_1 of v_1 and p_2 of v_2 are both of type smul and equivalent, $p_1 \equiv p_2$
- or else $\Pi(v_1) = \Pi(v_2) \wedge p_1 \equiv p_2$

This means *corresponding* blocks within the instances of smul-arguments are considered equivalent. In particular, we obtain classes of equivalent *units*, and every unit in such a class shall have the same vector of input edge weights – in principle.

Since the structure of the input region for a number of equivalent units carries topological information, the actual grouping of the links that are to be shared needs to have some topology-preserving properties; e.g., links from one row of a two-dimensional receptive field to a certain unit should be coupled with the links from the corresponding row of the input region of another unit, even if the "rows" involved vary in size.[2] Therefore, we need some more definitions:

The term for the whole network can be seen to uniquely define a total order $\prec$ on the set of units, namely the order in which the units occur in the written-down term after all mul and smul symbols have been expanded into corresponding pars.

Furthermore, we define the *topological distance* $\Delta(u_1, u_2)$ between two units u_1 and u_2 as follows: Consider the shortest way from u_1 to u_2 in the final diagram graph using only *member-of*-edges and visiting all units u_b between u_1 and u_2, $u_1 \prec u_b \prec u_2$, in the order given by $\prec$. This way consists of multiple alternating ascents and descents in the *member-of* tree. The maximum length of such a single ascent or descent, measured by the number of edges involved, determines the *topological distance* $\Delta(u_1, u_2)$ between u_1 and u_2. Figuratively, the topological distance gives the number of *coordinates* that u_1 and u_2 differ in, according to our topological interpretation of the surrounding network region.

Now we can formally express what topology-preservation means to link-sharing: Every input edge (u, v) to each unit v of a given equivalence class α with respect to $\equiv$ is assigned a so-called *share index*, $(u, v) \mapsto ind(u, v)$, so that the following conditions are satisfied:

- The first input edges get the same index for all $v \in \alpha$, i.e. $ind(u_{v1}, v) = \text{const.}$ where u_{v1} is the smallest unit according to $\prec$ being linked to v.[3]
- $u_1 \prec u_2 \Rightarrow ind(u_1, v) < ind(u_2, v)$ for all $(u_1, v), (u_2, v)$.

[2] Such variations are particularly annoying if caused by the fine-scaling of "identical" regions.

[3] The actual selection of this initial index is to ensure that sharing takes place among equivalent units only.

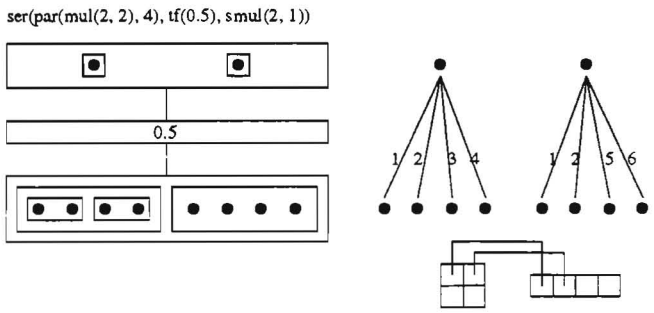


Figure 7: An example of weight sharing

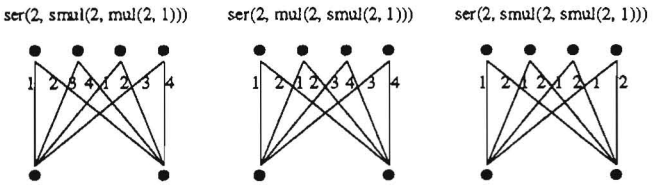


Figure 8: Combining mul and smul symbols

- $ind(u_1, v_1) = ind(u_2, v_2) \wedge ind(u_3, v_1) = ind(u_4, v_2) \Rightarrow \Delta(u_1, u_3) = \Delta(u_2, u_4)$ for all $(u_1, v_1), (u_2, v_2), (u_3, v_1), (u_4, v_2)$. This is what topology-preservation formally means.
- Reducing any one of these share-indices would violate one of the above conditions. This means the share indices are assigned continually unless the previous condition requires a gap.

Finally, all links with the same index are coupled. By the way, all biases remain independent in every case. Note that the above conditions *do not* induce a unique set of indices, but they *do* induce a unique scheme of shared weights!

Figure 7 gives an example of shared links and the idea of topology preservation. There are a "one-dimensional" and a "two-dimensional" receptive field for two *equivalent* units; only the first row of the two-dimensional region corresponds to the one-dimensional region with respect to link sharing.

Figure 8 shows the effects of some combinations of smul and mul symbols on weight sharing. Please refer to the definition of $\equiv$ to comprehend the results in detail.

3.7 Completeness

The bad news is the term language is not complete. Networks exist that cannot be expressed as a term. They are irregular, "non-structured" networks with link crossings that cannot be derived from higher-level *connected-to*-edges in the diagram graph.

But such networks are quite rare in practice, i.e. human network designers usually succeed without using networks of that kind. There seems to be no evidence that a non-structured architecture would be required for an ANN to learn some particular task. Furthermore, the language can be patched so that completeness is achieved

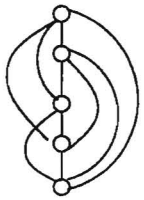


Figure 9: An impossible network

easily. If *named terms* (see section 3.3) are used with a list of irregular edges appended, every graph can be expressed. Figure 9 shows an example of an ANN that cannot be coded as a term. Nevertheless, it can be described by a named term:

$$\text{ser}(1:a, 1:b, 1:c, 1:d, 1:e) \\ < a-c, a-d, a-c, a-e, b-d, b-e, c-e >$$

If named terms are used, the resulting ANN is not always a *feed-forward* network, since cycles of edges can be specified, e.g. $\text{ser}(1:a, 1:b) < b-a >$. The feed-forward condition has to be checked in an extra pass before the resolution rules are applied. We do not believe that using named terms is actually necessary, because non-structured networks can be approximated by structured networks; for example, if only one edge is omitted in figure 9, the network *can* again be expressed as a term. With respect to the Theorem of CYBENKO [1] we have reason to hope that every function that is learned by a non-structured network can be learned by a similar structured network as well. This is additionally supported by the fact that BP is capable of deleting (more precisely: ignoring) drawn connections in principle by their weights converging to zero.

4 Using the Code as an Input Language for Simulators

If a network is to be put into a simulator, the schematic structure is usually known to the user, and thus the network can be easily encoded. The ZIP code reader by LeCun et al. [12] can be considered a suitable real-world problem. The network consists of 1256 units and more than 50,000 edges, most of them with shared weights. The units are arranged in overlapping feature detectors in a regular but non-trivial pattern. While the interconnection matrix takes several megabytes of memory and the BIGNET-specification [13] still fills a couple of pages, the term representation is quite handy:

```
ser(
  par(mul(16, 16)),
  mul(12, ser(tf(0.3125), smul(8, ser(tf(0.3125), smul(8, 1))))),
  tf(0.6667),
  mul(12, ser(tf(0.625), smul(4, ser(tf(0.625), smul(4, 1))))),
  30, 10
)
```

Modelling networks can be made easier and completeness can be assured if *partially named terms* are used. In

partially named terms only those nodes are given a name that are to get an edge of the separate edge-list. A network with three layers of 25 units each and a single shortcut from layer #1 to layer #3 could be expressed in this manner by $\mathsf{ser}(\mathsf{par}(1:a,24),25,\mathsf{par}(1:b,24)) < a-b >$. Note that the same network could be written (more elegantly) with a topological filter of value $\frac{1}{25\cdot 25}$ parallel to the hidden units.

5 The Genetic Algorithm

We used our term code in a genetic algorithm that is to find ANN architectures for given problems. The phenotypes are back-propagation networks with the transfer functions identity for input units and the sigmoid function $x \mapsto 1/(1+exp(-x))$ for hidden and output units. The GA follows the well-known standard scheme evaluate, reduce, select, produce. Below we briefly describe those aspects of the GA that are directly influenced by the characteristics of the genetic code and the problem, i.e. mutation, crossover, and the evaluation of the individuals.

5.1 Mutation Operators

Since our genetic code is obviously much more structured than binary strings, mutation operators are more complicated to write down, but easier to understand and to control. They are a set of term replacement rules, and mutating a genotype means randomly choosing a subterm in the genotype, a mutation rule being applicable to that subterm, and in most cases some additional rule-specific parameters. The application of the selected rule to the selected subterm yields the "mutant".

In the following list of mutation rules, small letters represent numbers or subterms and capital letters represent (possibly empty) sequences of subterms. Rule-specific context-conditions are labelled with ©, and rule-specific parameters are listed after #.

forgetSer:	$\mathsf{ser}(A,x,B) \to \mathsf{ser}(A,B)$
©	$\mathsf{length}(A \circ x \circ B) \geq 3$
#	relative position of x
forgetPar:	$\mathsf{par}(A,x,B) \to \mathsf{par}(A,B)$
©	$\mathsf{length}(A \circ x \circ B) \geq 2$
#	relative position of x
extendSer:	$\mathsf{ser}(A,B) \to \mathsf{ser}(A,x,B)$
extendPar:	$\mathsf{par}(A,B) \to \mathsf{par}(A,x,B)$
#	relative position of x
#	x is a new randomly created term
liftMul:	$\mathsf{mul}(n,x) \to x$
liftSmul:	$\mathsf{smul}(n,x) \to x$
lift1Par:	$\mathsf{par}(x) \to x$
liftSer:	$\mathsf{ser}(A,\mathsf{ser}(B),C) \to \mathsf{ser}(A,B,C)$
liftPar:	$\mathsf{par}(A,\mathsf{par}(B),C) \to \mathsf{par}(A,B,C)$
#	relative position of the subterm to be "lifted"
dropMul:	$x \to \mathsf{mul}(n,x)$
dropSmul:	$x \to \mathsf{smul}(n,x)$
#	n
drop1Par:	$x \to \mathsf{par}(x)$
dropSer:	$\mathsf{ser}(A,B,C) \to \mathsf{ser}(A,\mathsf{ser}(B),C)$
©	$\mathsf{length}(B) \geq 2 \wedge \mathsf{length}(A \circ C) \geq 1$
#	start and end of B within the argument list, obeying the condition
dropPar:	$\mathsf{par}(A,B,C) \to \mathsf{par}(A,\mathsf{par}(B),C)$
©	$\mathsf{length}(B) \geq 1$
#	start and end of B within the argument list, obeying the condition
resizeMul:	$\mathsf{mul}(n,x) \to \mathsf{mul}(r \cdot n, x)$
resizeSmul:	$\mathsf{smul}(n,x) \to \mathsf{smul}(r \cdot n, x)$
resizeN:	$n \to r \cdot n$
©	$r \cdot n \geq 2$ (mul, smul).
©	$r \cdot n \geq 1$ (n).
#	r
changeTf:	$\mathsf{tf}(p) \to \mathsf{tf}(p+r)$
©	$0 \leq p+r \leq 1$
#	r
share:	$\mathsf{mul}(n,x) \to \mathsf{smul}(n,x)$
unshare:	$\mathsf{smul}(n,x) \to \mathsf{mul}(n,x)$
expand:	$\mathsf{mul}(n,x) \to \mathsf{par}(x,\ldots,x)$
twiceSer:	$\mathsf{ser}(A,x,B) \to \mathsf{ser}(A,x,x,B)$
twicePar:	$\mathsf{par}(A,x,B) \to \mathsf{par}(A,x,x,B)$
#	relative position of x
tfInSer:	$\mathsf{ser}(A,B) \to \mathsf{ser}(A,\mathsf{tf}(r),B)$
©	$\mathsf{length}(A) \geq 1 \wedge \mathsf{length}(B) \geq 1$
#	relative position to insert the new filter
#	r

The application probability of each rule as well as the bounds for the randomly chosen parameters in the above list can be specified by the user of the GA. In other words, it is possible to supply the GA with some knowledge of the task to be performed. For instance, nobody would expect that the "optimal" network for xor and the optimal one for backgammon will be similar in size. Thus, the GA should converge faster if the mutation strategy can be adjusted at least to the estimated "size" of the task and the required architecture. Furthermore, there is no reason why it should *fail* to converge even if the controlling parameters chosen are inconvenient.

5.2 Crossover

In contrast to the mutation operators, crossover is extremely simple for our genetic code. If there are two terms a and b to be crossed-over, choose an arbitrary subterm of a and one of b and simply swap them. The only context-condition is that neither a nor b themselves may be chosen as subterms for swapping. The definition of the language ensures that either modified individual is also a term of the language if a and b are.

5.3 Evaluation

The phenotypes are trained by back-propagation in batch mode with the usual total square error cost function and a separate set of test patterns. Termination

criteria are the error on the training set and an upper bound to the number of BP epochs. The following enhancements to "traditional" back-propagation (e.g. [2]) are used:

- Dynamic adaptation of the η (learning rate) and α (momentum) parameters after SALOMON [10]. This method removes the necessity of choosing the "correct" parameters by hand and is capable of adjusting the parameters for varying local properties of the error space.
- Gradient normalization, i.e. the steps in the weight space depend on the gradient's direction but not on its magnitude. This solves the problem that the convergence of traditional BP slows down substantially if the error "landscape" becomes relatively flat, while steep slopes may cause jumps which are much too long.

Our universal quality function q for a single individual is

$$q = \exp(-c_1 et - c_2 np - c_3 nl - c_4 ep - c_5 nn)$$

where

et:	final error on the test set
np:	number of free weight parameters in the network
nl:	number of links in the network
ep:	BP epoch number with the smallest test error
nn:	number of term nodes in the genotype
c_i:	user-defined scaling constants

Component nn refers to the genotype rather than the phenotype; its purpose is to encourage the GA to develop compact representations (note that there might be "redundancies" in the genotype without any influence on the phenotype but affecting the set of possible crossover operations). The exponential function reduces the numerical differences so that the qualities can be reasonably used as relative production probabilities.

6 Results

In this section we present some experiments carried out with the term representation and the genetic algorithm outlined above.

6.1 Identity Function

First we tested our GA on the eight-dimensional identity function as an example of a very "small" problem. The training set consisted of the eight possible patterns with only one unit "high" (1) and the others "low" (0), the test set contained similar but somewhat noisy patterns.[4] We used a population size of 40 parents and 120 offspring. The quality was measured by the test error, $c_1 = 1$, the number of weight parameters and the number of term nodes, $c_2 = c_5 = 0.001$, and the training time, $c_4 = 0.0001$ (refer to section 5.3). This means we were interested in finding a network that learns the identity function and is minimal with respect to the amount of information incorporated. The mutation probability was set to 0.7 and the crossover probability to 0.8. The networks were trained for at most 500 epochs or until the training error dropped below 10^{-10}.

[4]This is a typical training environment for the popular encoder problem and a network with three hidden units. Of course we did not predetermine the network architecture, but the "classic" 8-3-8 encoder would be a suitable solution even though it cannot learn the identity function for all *possible* inputs.

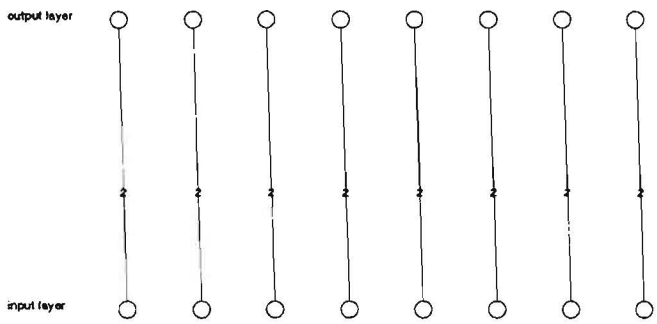


Figure 10: The identity network

In the following we show the milestones of the evolution, represented by the genotypes of the best individuals of each generation. Note that, for most representation schemes known, it would not have been possible to present the evolution process in such a compact form:

Gen.	Best Individual, Comment
1	ser(8, 8) 64 independent edges
2	ser(8, tf(0.125), 8) 8 independent edges
3	ser(smul(4, 2), tf(0.25), smul(4, 2)) 16 edges, 4 connection-weights
4,5	ser(8, tf(0.24), smul(8, 1)) 15 edges, 2 connection-weights
6	ser(8, ser(tf(0.125), smul(8, 1))) optimal phenotype

Apart from the redundant inner ser node, the best individual of the sixth generation is an optimal solution, because our GA insists on independent input and output layers (see section 3.5.2). Figure 10 depicts the resulting network, which contains eight edges with a single shared weight parameter. The share index "2" is printed at the edges.

While the milestones of evolution presented above are in a sense characteristic of our identity experiments, this is not quite the case for the convergence speed. We carried out 6 experiments under equal conditions, with the optimal phenotype shown evolving after 6, 16, 20, 17, 32, and 7 generations, which makes 16.3 generations on average.

6.2 Two-Spirals Problem

The two-spirals problem provides a good test of network topology optimization strategies because it has proven to be very hard to solve by backpropagation-based networks. Actually, some results indicate that *short-cut* connections could be necessary to succeed, and standard

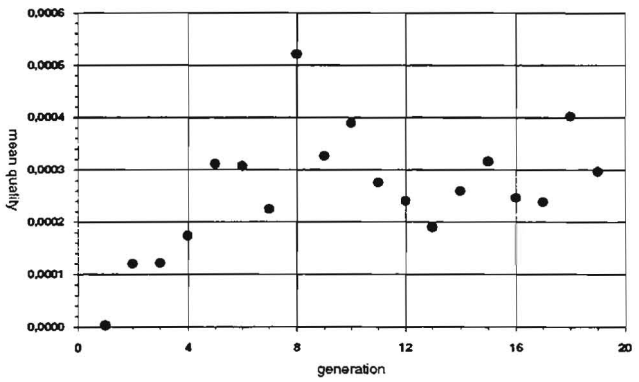


Figure 11: Two-spirals problem – mean quality

BP has been reported to require at least 20,000 cycles for a solution (FAHLMANN, LEBIERE [11]).

The problem can be outlined as follows: There are two interlocking spirals in the input plane, each going around a common centre point three times. The network's task is to decide whether a given point, represented by a pair of real-valued coordinates, belongs to the first or to the second spiral. Hence, two input units are mapped to one output unit.

Our population again contained 40 parents producing 120 descendants. Mutation and crossover probabilities were 0.7 and 0.8 respectively. The networks were trained for a (rather optimistic!) maximum of 12,000 epochs or until a training error of 0.1, which guarantees a hit rate of 100% on the training set. It seems to be favourable to start with a very small learning rate; we chose $\eta = 10^{-4}$. The training set comprised only 100 points, leaving 94 further ones as a test set to examine the networks' ability to generalize. In contrast, up to now all 194 patterns have usually been taken as training cases and generalization neglected.

Figure 11 shows the development of the mean quality of the entire population during the experiment. The "winner" of the experiment evolved in the 8th generation. Its evaluation terminated after 5322 cycles due to the training error, with a minimal test error of 2.16 in epoch 5319. This obviously means a hit rate of 100% on the training set, while the hit rate on the test set (which we unfortunately could not measure directly) might have been 91.5% making 8 misses *at worst*, but probably was in the area of 100% as well. In addition, the test error was almost certainly still decreasing when training terminated. Figure 12 presents the phenotype of our champion. It contains 28 hidden units arranged in only two hidden layers, and it is particularly surprising in that it does not show any *shortcuts* at all.

On the one hand, this network is quite a bit larger than the two-spirals networks proposed in literature, which comprise between 10 and 20 hidden units. On the other hand, we seem to have set up a new landmark with respect to the time back-propagation requires to learn this problem, particularly in view of the generalization results. Furthermore, in some generations a network with

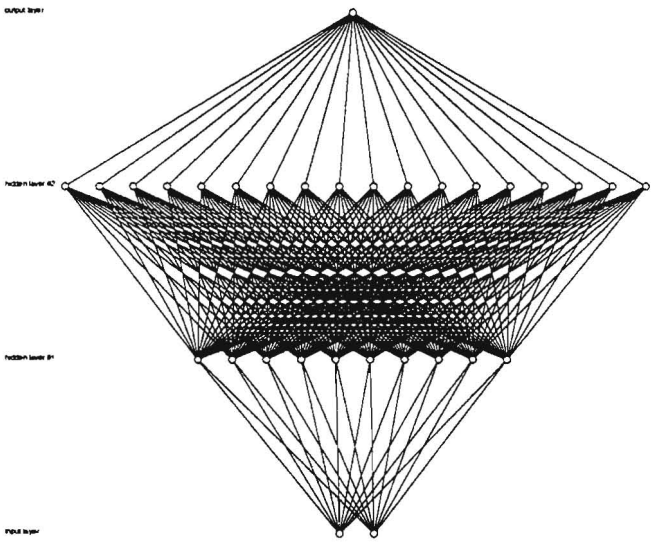


Figure 12: The two-spirals champion

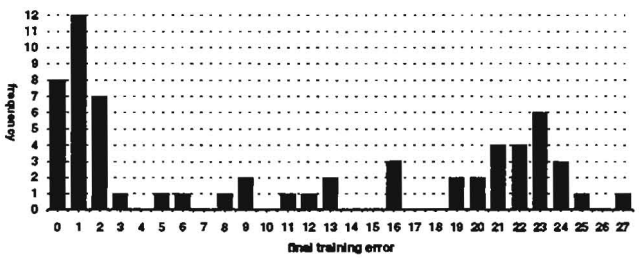


Figure 13: Training the two-spirals champion with enhanced BP

only 19 hidden units and a minimal test error between 3 and 6 was best. In general, solutions with two hidden layers similar in size, but no shortcuts, were clearly favoured by the GA.

When we examined the probability of our winner's successfully learning the task, it turned out that the astonishing results are in part a merit of the BP enhancements in use (see section 5.3). Standard back-propagation did not discover any correlation between input and output in any of 21 trials over 20,000 epochs each. But with parameter adaptation and gradient normalization turned on, 28 out of 63 trials (44%) were quite successful, i.e. showed training errors below 3.0, after at most 20,000 epochs. Figure 13 depicts the frequencies of the final training errors when rounded to integers.

Furthermore, the mean quality diagram confirms a typical problem of the evaluation of neural network architectures, the noise in the quality space as mentioned in the introduction. Although the fittest individual of every generation was ensured to survive, the minimal test error happened to increase again in some generations. We tried to reduce this noise by restricting the initial weights to the relatively small interval $[-0.1; 0.1]$, which appeared to be a suitable compromise between obtain-

ing reproducable results and successful back-propagation training.

7 Conclusion

The results presented show that our term representation enables a genetic algorithm to quickly find a network architecture that solves a given task in principle even if this task is rather difficult with respect to back-propagation strategy. Such a first solution may be more complicated than necessary, but our GA also performed quite well in reducing the networks in a subsequent phase, even though "local optimizations" do not seem to be the principal strong point of this representation scheme.

During our experiments, it turned out that the very compact and handy genotypes constitute an advantage that cannot be appreciated enough: Every genetic algorithm for optimizing network architectures will be controlled by a number of parameters, and the human experimentor has to adjust them in "real-world" applications. Therefore he should have an idea what the GA is doing (wrong?) at any time, which is quite easy to find out with a representation scheme like ours, but will be difficult and arduous if the genotypes are bunches of link specifications, at worst being represented in binary form.

Further investigation on ANN optimization using the term code should focus on problems requiring really large networks (i.e. more than 100 units at least), which would significantly reduce the influence of the random initial population on the convergence speed and the results of the GA.

By the way, in all of our experiments we have never found an indication that the incompleteness of the term representation is a serious problem in this field. Especially, for tasks requiring substantially big networks there can hardly be any *objective* criteria to identify an "optimal" network because of the noisy quality space and the existence of contradictory aspects of quality (e.g. error, network size, and training time).

Finally, even if no genetic optimization is involved at all, we think our representation language could be of great help in fields like simulator experiments or talking about network architectures. This holds particularly if incompleteness is overcome by the use of *partially named terms*.

8 Acknowledgements

We wish to mention that our work in this field was encouraged by Michael Scholz and Eckart Schau of the Department of Applied Computer Science at the Technical University of Berlin, and the publication of this paper was motivated by Iwan Santibàñez-Koref of the Department of Bionics and Evolution Technique. In addition, Christoph Reichert, Eckart Schau, and Michael Scholz provided parts of the evaluation software we used in our experiments.

References

[1] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals, and Systems 2*, pp. 303-314, 1989

[2] D. E. Rummelhart, G. E. Hinton, R. J. Williams: "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in Microstructures of Cognition, Vol I*, MIT Press, Cambridge, 1986.

[3] D. E. Goldberg: *"Genetic Algorithms in Search, Optimization, and Machine Learning"*, Addison-Wesley, Reading, 1989

[4] I. Rechenberg: *"Evolutionsstrategie"*, Frommann-Holzboog, Stuttgart, 1973

[5] I. Rechenberg: *"Evolutionsstrategie '94"*, Frommann-Holzboog, Stuttgart, 1994

[6] J. R. Koza, J. P. Rice: "Genetic Generation of Both the Weights and Architecture for a Neural Network", *IEEE Press, Vol II*, 1991

[7] S. A. Harp, T. Samad, A. Guha: "Designing Application-Specific Neural Network Structure Using the Genetic Algorithm", *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, 1990

[8] U. Utecht, K. Trint: "Mutation Operators for Structure Evolution of Neural Networks", proposed to be published in *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, Jerusalem, 1994

[9] F. Gruau: "Cellular Encoding as a Graph Grammar", *ISM*, 7/93

[10] R. Salomon: "Verbesserung konnektionistischer Lernverfahren, die nach der Gradientenmethode arbeiten", Dissertation, Technical University of Berlin, 1991

[11] S. E. Fahlmann, C. Lebiere: "The Cascade-Correlation Learning Architecture", *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, 1990

[12] Y. Le Cun, B. Boser, J. S. Denker, D. Hendersen, R.E Howard, W. Hubbard, L. D. Jackel: "Backpropagation Applied to Handwritten Zip Code Recognition"; *Neural Computation* 1, pp. 541ff, 1989

[13] A. Zell, N. Mache, R. Hübner, M. Schmalzl, T. Sommer, G. Mamier, M. Vogt: *"SNNS - User Manual"*, University of Stuttgart, Report 8/92

Evolving Neural Networks with Minimal Topology

Ralf Salomon
International Computer Science Institute,
1947 Center St., Suite 600
Berkeley, CA 94704-1198
rs@icsi.berkeley.edu

Abstract

While constructing and training neural networks, one faces the challenging question of how to determine an appropriate topology for the network to be trained. The concurrent process of learning a network and minimizing its topology can be seen as an optimization in a multimodal function. In this article we show how the collective learning procedure, which efficiently optimizes those function, can be successfully applied to evolve a minimal neural network's topology.

1 INTRODUCTION

While constructing and training neural networks, one faces the challenging question of how to determine an appropriate topology for the network to be trained. Usually, one has a certain amount of experience and makes an educated guess for the number of hidden units and hidden layers needed. The number of hidden units, and consequently the number of connections, has an influence on the generalization properties of the trained network and the number of training epochs needed.

In the neural network community, there is a common assumption that a network with too many weights do not generalize well (Le Cun, 1990). However, if the number of connections is below a certain threshold the network is not able to learn the given task. On the other hand, a larger number of hidden units normally decreases the number of training epochs. Thus, one has to find a good compromise which could be difficult beforehand.

In the past, several strategies have been developed to automize the process of determining a neural networks topology. In the skeletonization approach (Mozer, 1989), a *relevance* is assigned to each hidden unit. Once, learning has obtained a specified accuracy it calculates the relevance of all hidden units. Then the procedure removes the hidden unit with the least relevance and continues the training process. The optimal brain damage approach (Le Cun, 1990), involves the computation of a *salience* for each parameter (weight). After a specified criterion is obtained those weights with the lowest saliency are removed and training is proceeded. Yet another common approach starts with a small network and adds new hidden units when no further learning progress is attainable. Thus, this approach is reverse of those described above. An alternative approach is given in (Schiffmann, 1990). This alternative features a genetic algorithm which constructs different topologies at each generation. In a subsequent step, these networks are trained for a specified amount of time and the resulting error is used as a fitness value. By means of the genetic algorithm and backpropagation learning this procedure is able to produce small networks over time.

The approaches described so far have at least one of the following drawbacks. (1) A procedure which removes hidden units or single connections has to start with a sufficiently large network. But what is sufficiently large? If the initial network is too large, training time and removing process are unnecessarily long. (2) Adding hidden units from time to time might result in a network which represents the requested functionality but the final topology could be far away from the minimum. (3) Separating the training from the topology modification process requires a criterion after which the training has to be stopped. An unappropriate choice slows down the overall convergence speed. (4) The hybrid approach proposed by Schiffmann requires the specification of a maximal number of training epochs. Thus, the procedure is not able to find topologies which need more training time than that. In addition, this specification requires a certain degree of problem knowledge a priori.

In the next section, we show how a method, called *collective learning*, can be successfully applied evolving topologies which are very close to the minimum.

2 THE PROCEDURE

In (Salomon, 1994) we proposed the *collective learning* procedure for optimizing multimodal functions which have many local and only one global optimum. Collective learning is a hybrid scheme which is loosely inspired by evolutionary considerations and provides a significant improvement in global convergence compared to other methods as can be seen in Table 1. This table presents a short comparison of the number of function evaluations needed when applying the genetic algorithm variant of collective leaning (CL^{GA}) and PGA (Mühlenbein, 1991) to Rastrigin's function $f(\vec{x}) = 200 + \sum_{i=1}^{20} x_i^2 - 10\cos(2\pi x_i)$. One can see that collective learning is up

	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 400$
PGA	9900	42753	109072	390768	7964400
Collective Learning^{GA}	4600	8650	12650	23750	31150

Table 1: Number of Function Evaluations Needed by Collectice Learning and PGA when Applying to Rastrigin's Highly Multimodal Function $f(\vec{x}) = 200 + \sum_{i=1}^{20} x_i^2 - 10\cos(2\pi x_i)$ for Several Dimensions (20 to 400).

to two orders of magnitude faster than PGA.

The collective learning procedure is based on a genetic algorithm or evolution strategy (Rechenberg, 1973) respectively, and a modified self-adapting backpropagation algorithm (Salomon, 1992) working as a local learning procedure. The basic idea is that the genetic algorithm produces offspring with slightly modified topologies as well as slightly modified learning parameters like learning rate η, momentum α, and that backpropagation allows each offspring to perform local learning steps.

To generate new offspring the procedure (the genetic algorithm or the evolution strategy part) randomly choose one or two parents and constructs a new offspring by means of mutation and cross-over. In addition, the procedure maintains a construction as well as a destruction probability for each object. These probabilities were inherited in the same way as the learning parameters mentioned above. According to these probabilities, the procedure removes and adds connections as well as hidden units. Therefore, a network can grow or shrink. After all modifications are done, the procedure removes all hidden units which have no input or no direct / indirect connection to any output in order to obtain correct and useful topologies.

To establish some biological components, collective learning maintains *one* population of size p. Then, in each generation it produces o offspring, changes their topology, and determines their initial weights as well as local learning parameters by means of mutation and multiple-point crossing-over. Before collective learning makes the necessary selection, it gives each individual time to perform *exactly one local backpropagation step*.

In the subsequent *selection phase* the procedure selects the best p individuals according to the following scheme: For each individual the procedure maintains an *age count* and it calls an individual *young* iff the age count is below a given boundary. In the first stage, the procedure selects all young individuals and then, in the second stage, it uses the current fitness - with respect to the given task - to select further objects. Note that the individuals perform only *one step* and not, as in several other approaches, the total or a major part of the whole learning process. This sort of hybrid optimization process is known as Lamarckian learning (Grefenstette, 1991), and the main idea behind this approach is that young and promising networks with promising features have enough time adapting to the given task before they feel any selection pressure.

A crucial point using a genetic algorithm or evolution strategy as an optimization scheme is the definition of an appropriate fitness function. This is because the fitness value decides which networks are better than others and consequently decides which networks will survive in the next generation. Within the context of evolving minimal topologies the procedure has two distinguished goals. First, the network should posses as less connections as possible, and second, the network's output has to be within a specified accuracy. To this end, the fitness function q has two parameters and is defined as follows:

$$q(me, \#c) = \begin{cases} me + C & \text{if } me > ac \\ me + \#c & \text{otherwise} \end{cases} \quad (1)$$

whereas $\#c$ is the number of connections, C a huge constant (e.g. 1000), ac the specified accuracy and me the maximal error occurring on any output and any pattern. The above definition of the fitness function guarantees that any network which has obtained the specified accuracy has a better fitness value than any other network with more connections.

3 RESULTS AND DISCUSSION

In our experiments we used standard feed forward networks without restrictions to the number of hidden units and hidden layers, and $f(x) = 1/(1+e^{-x})$ as the squashing function. Unless otherwise stated, we used in all trials a population size of $p = 50$, $o = 4$ offspring, and a bound of age = 11 for young objects.

We started our experiments with the 4-x-4 encoder task, where the standard topology consists of 2 fully interconnected hidden units with a total of 22 connections. This task is fairly simple. However, it is worth to study it because this task is well understood and our experiments give further insights in the minimization process. During various trials, the procedure developed different solutions with mostly two or three and occasionally four hidden units. Normally, after a few generations the first solution consists of approximately 50 connections. In the ongoing optimization process, the total number of connections are reduced. Generally, the final solutions consist of 12 to 15 connections.

A typical run can be seen in Figure 1. Here, the x and y axes represent the number of generations and the fitness value respectively. As described above, the fitness value is the sum of the number of connections and the maximal error iff the accuracy is within its specification (i.e. 0.3). One can see that the best network after 860 generations has only 17 connections, after 2700 generations 16 connections and so forth. Finally, after 6500 generations the procedure ends up with a network of merely 12 connections including all bias links. One may wonder, how such a sparse network can perform the desired task. In this case, there are solely four connections from the input to the hidden layer and the two hidden units generate the following four activation patterns: $(0.0, 0.5)$, $(0.5, 1.0)$, $(0.5, 0.0)$, and $(1.0, 0.5)$.

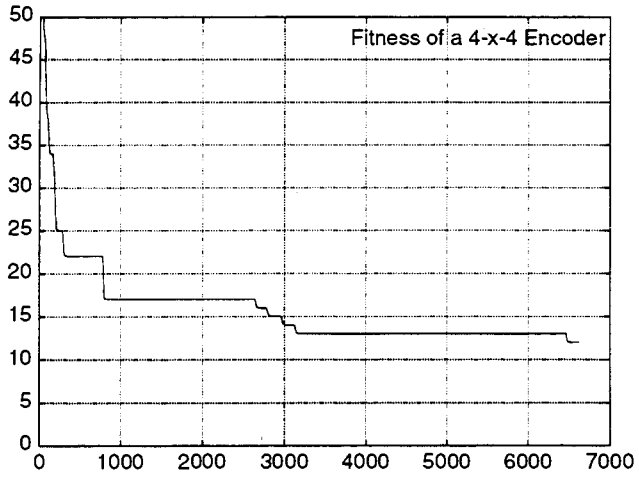


Figure 1: Fitness (Number of Connections plus Maximal Error) of a 4-x-4 Encoder.

Furthermore, we used collective learning to evolve a neural network for a pattern recognition task. Here, the procedure came up with the first solution consisting of 305 connections after 30 generations. In the subsequent optimization process, the procedure needed an additional 330 generations to shrink the network to 106 connections. Finally, after the 750th generation the network was shrunk to 25 connections.

In addition, we applied our procedure to other tasks. For example, if one allows an encoder network having short cuts from the input to the output layer then all hidden units vanish during the optimization process and the final network consists of exactly eight connections (two per each output unit).

We also applied our procedure to the well known exclusive-or (XOR) problem. In this case, the procedure needs approximately 150 generations to construct a network with 8 connections, 183 generations to shrink it to 7 connections, and finally ends up with 6 connections after 900 generations.

4 CONCLUSION

In this article, we have shown how to evolve a neural network's minimum topology by applying the collective learning procedure that is devoted minimizing multimodal functions. The link between multimodal functions and evolving topologies is the following. Removing hidden units or particular connections results in a modified representations in the hidden layer(s). Thus, an *ongoing* learning process can get stuck in a local minima. To overcome these local minima, the collective learning procedure generates different topologies and significantly modifies all connections of new offspring. By means of the described selection process, the procedure eventually overcomes these local optimia. Applied to the 4-x-4 encoder task, the procedure found solutions with merely twelve connections in a resonable amount of time. Currently, we are investigating other topology modification operators as well as different fitness functions. We are also applying the procedure to more realistic tasks.

Acknowledgements

We gratefully acknowledge David Stoutamire for helpful discussions and polishing up and thanks in particular to Jerome Feldman for his support.

References

[1] John J. Grefenstette. Lamarckian learning in multi-agent environments. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufman Publisher, 1991.

[2] Yann le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, pages 598–605. Morgan Kaufmann Publishers, 1990.

[3] Michael C. Mozer. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 107–115. Morgan Kaufmann Publishers, 1989.

[4] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizier. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufman Publisher, 1991.

[5] Ingo Rechenberg. *Evolutionsstrategie.* Problemata. Frommann-Holzboog, 1973.

[6] Ralf Salomon. Improved global convergence by means of collective learning. In *The Third Annual Conference on Evolutionary Programming.* World Scientific Publishing, 1994.

[7] Ralf Salomon and J. Leo van Hemmen. A new learning scheme for dynamic self-adaptation of learning-relevant parameters. In *International Conference on Artificial Neural Networks*, pages 1047–1050. North-Holland, 1992.

[8] Wolfram Schiffmann and Klaus Mecklenburg. Genetic generation of backpropagation trained neural networks. In *International Conference on Parallel Processing in Neural Systems and Computers (ICNC), Düsseldorf, FR Germany*, pages 205–208. Elsevier Science Publishers B.V., 1990.

Simulation of Global Illumination: An Evolutionary Approach

Brigitta Lange, Markus Beyer

Fraunhofer Institute for Computer Graphics
Wilhelminenstr. 7
D–64283 Darmstadt
F.R.G.

Abstract

This paper presents a new approach to optimize the global illumination simulation by Evolutionary Algorithms. It is shown that Evolutionary Algorithms have the potential to achieve good approximations to the solution of the Rendering Equation a multidimensional integral equation modelling radiant light transfer.

In contrast to Monte Carlo evaluation irradiance information gained during sampling is exploited efficiently. Thus the simulation process becomes self–organizing and is not limited to any a priori assumptions on irradiance distribution, which allows the system to adjust optimally to a particular lighting situation and results in a better convergence towards the accurate value of the Rendering Equation.

From a general classification of variance reduction techniques and Evolutionary Algorithms two different evolution models for the optimization of global illumination are derived: one describes the evolution of an optimal sample ray distribution and the other represents an evolutionary subdivision of the integration domain into intervals of optimal confidence.

1 Introduction

In computer graphics the attainable degree of realism strongly depends on the simulation model of global illumination. In general for every elementary surface area within a scene the total irradiance incident from the entire half–space has to be accounted for. In a mathematical formulation this leads to a system of complex integral equations also known as *Rendering Equation* [Kaj86]. Since there exists no closed form analytical solution the Rendering Equation is solved approximately by applying *Monte Carlo methods*.

Monte Carlo integration is a stochastic process, where the integration domain, i.e. the half–space of irradiance, is sampled by a finite set of random rays. The integral is estimated by the weighted average of irradiance values calculated for each sample ray direction. The major problem of Monte Carlo integration is to determine an optimal location and density of samples in order to guarantee some bound on the variance of the estimate, which is equivalent to finding a probability density function that is a good primary estimator for the function

being integrated. In general this is impossible, since the irradiance distribution over the hemisphere above a point on a surface has many local peaks of different magnitude and both, location and magnitude of these local intensity maxima are unknown in advance.

Many well known techniques for variance reduction (e.g. importance sampling, stratified sampling) have been applied to the global illumination problem. Yet even with these techniques, there are many scenes for which current Monte Carlo algorithms fail to yield good approximations; since they have one substantial drawback: they rely on a priori assumptions on irradiance distribution.

Unfortunately the only way to gain information about irradiance distribution is by actually evaluating it. The goal, therefore, is to optimize the simulation process in such a way that this information is exploited effectively allowing the system to adapt itself to the actual irradiance distribution.

The techniques presented here contribute towards a solution to this problem by means of *Evolutionary Algorithms* (EA). In the past Evolutionary Algorithms, which are directed search techniques based on the model of natural evolution, have been applied successfully to many global optimization problems. We decided to approach the global illumination problem by Evolutionary Algorithms because they have the capability of evaluating extremely noisy and discontinuous objective functions without needing any predefined models and provide mechanisms for self–adaptation.

First, we will analyze variance reduction techniques for Monte Carlo simulation of radiant light transfer. Then a short description of Evolutionary Algorithms is given, and two suitable representations for the global illumination problem as evolutionary process, where each implements a different evolution model, are investigated. The outline of the algorithms is described and it is shown, that by Evolutionary Algorithms a better convergence of the estimate towards the solution of the Rendering Equation is achieved, which in turn results in an improvement of image quality.

2 Simulation Model

The production of realistic images requires the ability to simulate the propagation of light in an environment, i.e. the ability to completely account for the global illumination arising from complex interreflections within the environment. Therefore the light transfer model and the simulation algorithm used are the most important characteristics of a realistic renderer. The light transfer model governs the quality of the simulation results. The simulation algorithm affects the simulation speed and the accuracy of the results.

In classical radiation theory radiant light transfer between surfaces can be characterized by an integral equation modelling the radiance of light leaving a point on a surface in a particular direction. The radiance of a surface is the sum of the emitted and the reflected radiance. For a given point on a reflecting surface the total hemisphere of incoming radiation has to be accounted for in order to calculate the emitted radiance. Thus the radiance L_{out} of an elemental

projected surface area $d\omega$ emitted in direction (θ_r, ϕ_r) is obtained by integrating the incident radiation L_{in}over all directions (θ, ϕ) of the total hemisphere Ω (see Fig. 1), which leads to the following alternative form of the Rendering Equation:

$$L_{\text{out}}(\theta_r, \phi_r) = L_e(\theta_r, \phi_r) + \int_{\Omega} \rho(\theta_r, \phi_r, \theta, \phi)\, L_{\text{in}}(\theta, \phi) \cos\theta_{\text{in}}\, d\omega_{\text{in}} \qquad (1)$$

where $L_e(\theta_r, \phi_r)$ is the emitted radiance in direction (θ_r, ϕ_r) and $\rho(\theta_r, \phi_r, \theta, \phi)$ is the bidirectional reflectance of the surface. The L_{in} again are emitted radiances of other surfaces in the environment which have to be solved for all surfaces simultaneously.

Since it is not possible to find a closed form analytical solution there exist two general methods for approximating this complex multidimensional integral equation: finite– element methods and Monte Carlo methods. The former approach yields *radiosity* algortithms and the latter approach yields *stochastic ray-tracing* algorithms where light paths are followed from the eye all the way back to the light sources. These light paths correspond to Markov Chains [Kaj86]. Thus for every visible object point the hemisphere of incident radiation is sampled by a finite set of randomly selected rays (paths) [Lan91].

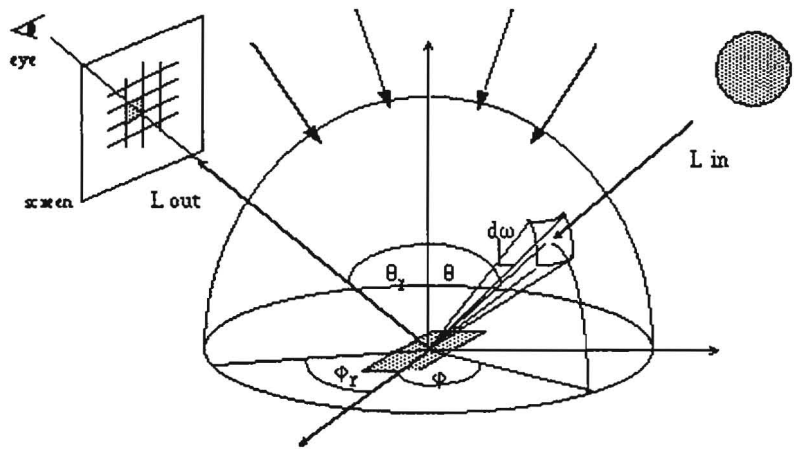


Fig. 1. Geometry for radiant light transfer calculations.

The advantages of Monte Carlo methods are its elegance and generality. They are easy to implement and handle arbitrary surface geometries and reflectance functions in a clean uniform way. Monte Carlo ray-tracing can simulate a lot more lighting effects all in a physical and mathematical way correct.

Unfortunately the basic Monte Carlo technique is extremely slow to converge and has an inherent limitation: sampling noise. Several techniques for variance

reduction try to alleviate this problem, but rendering typical global illumination effects such as indirect diffuse reflections still requires a lot of computational effort. By analyzing common variance reduction techniques for global illumination we will derive the requirements for a new optimization technique which further reduces the variance in order to achieve the best approximation.

2.1 Variance Reduction

Although it is possible to approximate the Rendering Equation using uniform stochastic sampling and sample–mean Monte Carlo integration [Rub81], the convergence under most conditions is so slow, that such a solution is impractical.

The key to optimal convergence lies in variance reduction. This is achieved by altering the probability density in such a way, that the information gained from each sample is maximized. There have been made several attempts to increase the efficiency of the Monte Carlo solution through variance reduction techniques like *importance sampling* and *stratification* ([Dre91], [Kir91], [Lan91], [Shi91], [War92]). A specific cause of noise is highly non-uniform irradiance distribution. The problem is that the emitted radiance L_{out} is essentially the product of the incident radiation L_{in} with a reflection term ρ. Generally we can obtain accurate information about the reflection term but not about L_{in}. For this reason importance sampling distributes the random variables according to the surfaces reflection properties and samples where the reflection term ρ is large, but usually does not consider the irradiance distribution. However if L_{in} is highly non uniform (for example 99light comes from only 1predictor of the important sampling directions, leading to high variance [Vea94].

Stratification is effective if the domain of integration can be partitioned into strata within which the variance is smaller than the difference between their means. In the context of rendering stratification is preformed explicitly by estimating the integrals of direct and indirect irradiance independently. This may become inefficient if there are too many light sources in the scene, because in this case it is very difficult to generate a set of appropriate sample rays. Although these variance reduction techniques reduce the number of samples, they have to rely on a priori assumptions about irradiance distribution. In order to reduce the variance of the estimated irradiance to tolerable levels for arbitrary surfaces, it still requires tracing thousands of rays, even if a combination of importance sampling and stratification is used. This is due to the fact, that the irradiance usually is a multimodal function, thus it is hard to determine a good primary estimator without any previous knowledge.

Another fundamental problem of classical variance reduction techniques is, that the information gained during the simulation process is not sufficiently exploited. The samples are distributed only once according to a predefined probability density function. Furthermore the irradiance information gained from each sample ray decreases with respect to the total number of sample rays. Therefore an adaptive sampling technique is needed which is superior to Monte Carlo sampling, in that it efficiently exploits irradiance information gained du-

ring the sampling process itself, thus giving the stochastic process a direction and improving the simulation process towards optimal convergence.

In the past Evolutionary Algorithms have proven to be powerful methods for global optimization problems in different fields of research, not using any predefined internal model of the objective function. They are robust even in the case of multimodal objective functions and provide mechanisms for self–adaptation. Due to these properties they are well suited to optimize the simulation of global illumination.

3 Evolutionary Algorithms

In nature, evolution, which is the process of adaptation of living organisms to their environment, can be regarded as a very powerful optimization method. Thus developing nature analogous problem solving strategies seems to be promising.

Evolutionary Algorithms are directed search techniques with a great versatility, that mimic the effects of evolution and natural selection. The most important ones being *Evolution Strategies* ([Rech73], [Schw75], [Schw77]) and *Genetic Algorithms* ([Gol89], [Hol75]), they share common concepts but differ in their implementation. Each of these approaches is based upon a collective learning process within a population of individuals representing points in the search space of potential solutions to a problem given by the objective function. The individuals of an arbitrarily initialized start population adapt to their environment, by evolving towards better and better regions of the search space (in terms of the objective function) by means of probabilistic *selection* and *genetic operators* (*mutation* and *recombination*) in such a way, that the average quality of the individuals increases. Each individual is assigned a quality value which usually depends on the objective function. Selection is an operator of the evolutionary process that favors individuals of higher fitness to reproduce more often than those of lower fitness, thus guaranteeing survival of the fittest, and giving the process a direction. The recombination operator allows the exchange of genetic information, whereas the mutation operator accounts for genetic innovation.

Evolutionary Algorithms have shown to be useful methods for the exploration of large search spaces using simulated systems of variation and selection. They achieve much of their breadth by ignoring information except that concerning payoff and they can tolerate extremely noisy function evaluation. Furthermore they find near optimal results quickly after searching only small portions of the search space. Due to these properties they seem to be well suited for the optimization of the sampling process in order to approximate the Rendering Equation.

If we want to apply Evolutionary Algorithms to the problem of calculating the total radiation incident to a given point, we first have to formulate the radiation calculation as an optimization problem. Next we have to find a suitable evolutionary model for the optimization problem above and define an Evolutionary Algorithm to solve it. The algorithm has to be designed in such a way, that

it produces successively better approximations to the integral equation by exploring the total hemisphere of incident radiation; searching for those regions or sample ray directions that contribute significantly to the irradiance. Thus increasing the average information gained from each sample and converging towards an optimal sample ray distribution.

In the following two different approaches to the global illumination problem by evolutionary algorithms will be presented. In contrast to the classical design of an EA, we have not chosen every individual to be a full representation of the solution. This would imply a population of ray distributions, which seems rather impractical, not only from the computational efficiency point of view, but also from the difficulties in defining an appropriate quality measure for individual ray distributions, as well as difficulties in designing suitable genetic operators.

4 Evolution of Sample Ray Distributions

As mentioned earlier, the approximation accuracy in the context of Monte Carlo integration strongly depends on the sample ray distribution. This means we are looking for an optimal distribution resembling the actual irradiance distribution in order to reduce the variance of the estimate.

In a figurative sense our first evolution model can be described as follows: the hemisphere of incident radiation to an object point represents the biosphere for a population of ray individuals. These ray individuals now have to adapt to their environment by finding optimal living conditions for themselves. Thus they have to search for attractive places to settle down. With increasing attractiveness of their actual residence on the hemisphere their willingness to move decreases. The attractiveness or quality of life at a certain place depends on the irradiance at that place as well as on the population density of its surroundings.

The goal of this settlement process is to find a suitable settlement structure, where the living conditions for all ray individuals are approximately equal. In contrast to classical Evolutionary Algorithms, where only one individual representing the optimal solution is searched for.

The process of finding an optimal settlement structure corresponds to the ability of the EA to exploit the information gained by individuals during the settlement process. In order to have no loss of irradiance information accumulative cartographic irradiance maps are produced during the settlement process. The process terminates if the irradiance map of the hemisphere has a certain state of accuracy.

In order to find all attractive regions of the hemisphere quickly and at the same time achieve an overall representative settlement structure, the Evolutionary Algorithm has to be implemented in such a way that there is always a balance between exploration of the hemisphere and exploitation of the information gained by it.

4.1 Implementation

In the evolutionary model outlined above, a ray individual is defined by its ray direction (θ, ϕ) related to the local sphere coordinate system with cone angle θ and circumferential angle ϕ. This representation is appropriate, since it allows a problem specific design of the genetic operators. Furthermore individuals that are close to each other in the representation space are also close in the problem space [Mich92].

An initial population P_0 of μ ray individuals is represented by a set of random ray directions equally or stochastically distributed over the total hemisphere of a visible surface point within the scene:

$$P_0 = \{(\theta_1, \phi_1), (\theta_2, \phi_2), \ldots, (\theta_\mu, \phi_\mu)\}, \tag{2}$$

where

$$\theta_i \in \left[0 \ldots \frac{\pi}{2}\right], \ \phi_i \in [0 \ldots 2\pi] \ (\forall i \in \{1, \ldots, \mu\}).$$

The evaluation of the initial population P_0 is performed in four steps, one for the calculation of the irradiance incident from each individual ray direction and three to calculate a fitness value for each ray individual.

The irradiance $L_{\text{in}} : \mathbb{R}^3 \times I \rightarrow C$ (where $C = \mathbb{R}^3$ is the color space of RGB triplets) associated with each ray individual corresponds to the objective function of the general EA.

The fitness $f_{\text{fit}} : C \times I \rightarrow [0 \ldots 1]$ of a single ray individual is determined by the difference between the individual's irradiance and an assumed background radiance for that direction. Since the overall contribution of the background radiance to the total irradiance can be calculated in advance, the main effort of the evolutionary search procedure can be spend on those regions of the hemisphere, where the irradiance differs significantly from the background radiance, i.e. regions, where the gain in information for the evaluation of the reflected radiance is high. In combination with the selection procedure, f_{fit} is responsible for the exploitation of information gained during the evolution process, thus conducting a local search for regions considered to be important for the total reflected radiance.

The fitness value f_{fit} only depends on the irradiance value measured by ray tracing. Therefore it does not describe the living conditions of each ray individual completely according to the settlement model defined above. In order to satisfy this model, also the population density distribution over the hemisphere has to be taken into account. The sharing function $f_{\text{share}} : [0 \ldots 1]^\mu \rightarrow [0 \ldots 1]^\mu$ considers the distance between all individuals within the population and reduces the fitness value calculated for each ray individual by f_{fit} according to the population density of its neighborhood. Therefore it also reduces the fitness difference within the population, and in combination with the selection procedure it leads towards a global search process exploring the total hemisphere.

In order to achieve an appropriate balance between exploration and exploitation, i. e. between global and local search, the fitness values given by f_{fit} are scaled by the scaling function $f_{\text{scale}} : [0 \ldots 1]^\mu \rightarrow [0 \ldots 1]^\mu$ before f_{share} is applied.

f_{scale} is constructed in such a way that the minimum fitness value $f_{\text{fit}_{\min}}$ within the population remains unchanged whereas all other fitness values are scaled linearly so that the maximum scaled fitness value $f_{\text{scale}_{\max}}$ does not exceed $f_{\text{fit}_{\min}}$ times a constant factor m_{scale}.

The functionality described so far allows to initialize and rate the population of rays which then will evolve until a termination condition is reached. If for a number of successive generations there is no significant change in the radiance value estimated, then is assumed that no more gain in information can be achieved and the process terminates.

A new population is derived from the old one by first generating an intermediate generation of $\mu + \lambda_o$ individuals, where λ_o is the number of offsprings. Then from this intermediate generation the μ best individuals are selected in terms of their scaled and shared fitness values.

The offsprings are produced by randomly selecting a subpopulation of λ_o parents and applying a genetic mutation operator $m : I^{\lambda_o} \rightarrow I^{\lambda_o}$.

The mutation operator is the most important operator of the process. It is designed in such a way that it models a directed search for attractive regions of the hemisphere, thus allowing the individuals to move around with the goal to find a place with higher quality of life. It produces new individuals through altering the ray directions of the parents in dependence of their scaled and shared fitness values.

As a result we achieve an implicit stratification of the hemisphere into different regions, which are sampled according to the irradiance information they provide. Thus an optimal sample distribution is evolved, which becomes noticeable in a significant reduction of overall noise.

Based on the observation that the indirect irradiance tends to change slowly over a surface [War92], the evolved ray distribution can serve as a start population for neighboring object points. Thus information is exploited not only during the sampling process but also within the image space.

Figure 2 shows a typical sample ray distribution produced by Monte Carlo simulation with importance sampling and the corresponding sample distribution optimized by the Evolutionary Algorithm (for the geometry of the corresponding scene refer to Fig. 4). Comparing both reveals that the Monte Carlo algorithm spends a lot of its effort in regions of the hemisphere, where the irradiance information is low. In contrast, the Evolutionary Algorithm directs the sampling process towards those regions with large payoff in irradiance information, which particularly becomes evident by the concentration of rays in the range of the white light source.

One problem of this algorithm, however, lies in the final evaluation of the integral. Since there exists no analytical description of the probability density function the evolved ray individuals have to be mapped to a hemispherical grid, which leads to an undesirable increase of computational cost and may even result in approximation errors. In order to avoid the explicit mapping of rays to solid angles, we have developed an alternative evolutionary algorithm, where solid angles are implicitly accounted for.

5 Evolutionary Stratification

The concept here is to maximize the confidence in our estimate of the integrand by an evolutionary stratification of the hemisphere. The integration domain is subdivided adaptively by means of probabilistic selection and mutation in such a way that the estimated value of the integral in each stratum is equal. If the size of the strata does not change within certain limits, the optimal approximation result is reached.

5.1 Implementation

In this evolution model the hemisphere represents a population of irradiated solid angles. For computational efficiency each individual is defined by a spherical triangle with an associated mean irradiance value obtained by the sample rays at its vertices (see Fig. 3).

Initially the hemisphere is subdivided into four spherical triangles of equal size. Each individual is evaluated by determining its local and regional fitness value. The local fitness is obtained by calculating the mean irradiance value. The regional fitness of an individual is determined by the maximum deviation (in RGB–primaries) from the mean irradiance values of its direct neighbors weighted by its area size. Thus the regional fitness is a measure for confidence or quality of life.

The goal now is to reduce the variance of the estimate by achieving a stratification of the hemisphere into solid angles (shperical triangles) of equal confidence. In each generation the integral is evaluated and the size of the solid angles is adjusted. Therefore the mutation operator models a population growth by cell–splitting, i.e. subdividing the individuals into new triangles. Parents are selected by Roulette wheel, where the chance for being selected for subdivision is proportional to the triangles regional fitness. The process terminates if all individuals have an almost equal quality of life (see Fig. 3).

The advantages of this method are its fast convergence and computational efficiency, because the evolution process and the evaluation of the integral can be effectively combined by using the same data structures. It produces successively better approximations to the integral equation by exploring the hemisphere, searching for those solid angles that contribute significantly to the total irradiance. Thus increasing the average information value of the samples and converging towards the best approximation, which in turn results in an improvement of image quality (see Fig. 4).

6 Conclusions

The main objective approximating the Rendering Equation is to minimize the variance of the estimate. This implies an optimization of the sample ray distribution used. Since the irradiance distribution over the hemisphere is unknown in advance, Monte Carlo sampling techniques, which use predefined probability

density functions, have to rely on a priori assumptions and are mostly inefficient. Adaptive sampling techniques are superior, because they use information gained during the sampling process itself to generate a subset of new samples in each adaptation step. We have shown that in order to exploit this information nature analogous techniques like Evolutionary Algorithms can be successfully applied. In contrast to classical Monte Carlo methods the first Evolutionary Algorithm presented above achieves a self–adaptation of the sample ray distribution to a particular lighting situation. In order to avoid computational effort for the final evaluation of the integral, an alternative evolutionary approach has been investigated. In this approach not sample directions but solid angles are subject to evolution. The confidence in the estimate is maximized by evolutionary subdividing the integration domain according to local and regional irradiance information, which in turn results in a fast convergence towards the optimum.

References

[Dre91] Drettakis, G.; Fiume, E.: *Structure–Directed Sampling, Reconstruction and Data Representation for Global Illumination.* Proceedings of the Second Eurographics Workshop on Rendering, 1991.

[Gol89] Goldberg, David E.: *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, Massachusetts: Addison- -Wesley, 1989.

[Hol75] Holland, John H.: *Adaptation in natural and artificial Systems.* Ann Arbor, Michigan: The University of Michigan Press, 1975.

[Kaj86] Kajiya, James T.: *The Rendering Equation.* In: Computer Graphics (SIGGRAPH '86 Proceedings) 20(4), August 1986, S. 143–150.

[Kir91] Kirk, David; Arvo, James: *Unbiased Variance Reduction for Global Illumination.* Proceedings of the Second Eurographics Workshop on Rendering, 1991.

[Lan91] Lange, Brigitta: *The Simulation of Radiant Light Transfer with Stochastic Ray–Tracing.* Proceedings of the Second Eurographics Workshop on Rendering, 1991.

[Mich92] Michalewicz, Zbigniew: *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin; Heidelberg: Springer, 1992.

[Rech73] Rechenberg, Ingo: *Evolutionsstrategie.* Stuttgart: Frommann–Holzboog, 1973.

[Rub81] Rubinstein, Reuven Y.: *Simulation and the Monte Carlo Method.* New York: Wiley & Sons, 1981.

[Schw75] Schwefel, Hans–P.: *Evolutionsstrategie und numerische Optimierung.* Technische Universität Berlin, Fachbereich Verfahrenstechnik, Dissertation, 1975.

[Schw77] Schwefel, Hans–P.: *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie.* Basel, Birkhäuser, 1977.

[Shi91] Shirley, Peter S.: *Physically Based Lighting Calculations for Computer Graphics.* Urbana, University of Illinois, PhD Thesis, 1991.

[Vea94] Veach, Eric; Guibas, Leonidas: *Bidirectional Estimators for Light Transport.* Proceedings of the Fifth Eurographics Workshop on Rendering, 1994.

[War92] Ward, Gregory J.: *The RADIANCE Lighting Simulation System.* Global Illumination, ACM SIGGRAPH'92; Course Notes of the 19th Annual Conference & Exhibition on Computer Graphics and Interactive Techniques, July 1992.

Construction of Conflict-Free Routes for Aircraft in Case of Free-Routing with Genetic Algorithms

Ingrid Gerdes
German Aerospace Research Establishment
Institute for Flight Guidance, Traffic Systems Analysis
PO 3267, D-38022 Braunschweig, Germany
E-Mail gerdes@ibr.cs.tu-bs.de, Tel. (+49)(531) 295-2279, Fax (+49)(531) 295-2899

Abstract

This paper describes a special genetic algorithm for the creation of flight routes for aircraft in the airspace. A detailed description of the problem and the implemented algorithm is presented together with a test of two mutation types for a special gene. A short theoretical discussion about the building block hypothesis will be given. Furthermore the results of several experiments with a randomly generated flight scenario and a real-traffic scenario are lined out. Altogether the paper shows the initial stages in finding a solution to decrease the delay in the airspace and to use the airspace more efficiently.

1 Introduction

At the moment the air traffic control systems in many countries have to cope with increasing traffic congestions and therefore with an increasing delay. These congestions are caused by the growing air traffic over the last years together with the airspace structure with its overloaded sectors and prescribed standard routes. This does not mean that the airspace itself is overloaded, too. Improvements are necessary for a better use of the airspace around the prescribed routes. A possible new strategy could be 'Free-Routing', meaning that there would not be a system of prescribed routes and all aircraft may use the direct link between start and destination airport if no conflict would occur with other aircraft. But if all aircraft will use their own routes, this will lead to a very complex system and it will be impossible for the controllers to locate impending conflicts. A proposal how to find the best routes without conflicts is presented by the tool **ROGENA** (free **RO**uting with **GEN**etic **A**lgorithms) [4]. This paper will give a short description about the present state of ROGENA.

2 Description of the Problem

Over the last years a constant increase in air traffic could be recognized. As mentioned before, for facing the increasing traffic it is necessary to change the airspace structure and the operational procedures. The capacity problem does not only depend on the high number of aircraft but also on the strategies for the use of the airspace.

A big problem for an increase in airspace capacity is caused by the airspace sectors and standard flight routes. Each sector has its own limited number of aircraft which can be handled and every aircraft normally uses a standard route (normally not the direct link) between start and destination airport (figure 1). In case of aircraft on different routes this reduces the number of points where conflicts can occur to those ones at crossing points of routes. Therefore, talking about congested airspace means talking about congested standard routes. The limited number of aircraft for each sector or on each route is responsible for the increasing delay. Such a situation is very problematic for those aircraft which have missed the sheduled time slot for departure or are not planned in advance.

A possible solution for increasing the airspace capacity could be found in giving up these standard routes. This strategy is called 'Free-Routing', which includes the permission to fly in any direction in any level at any time if the controller allocate such a route. This would lead to the following advantages:

- The whole airspace would be usable for air traffic.
- It would be easier to find new slots or a new route if the time of departure has changed.
- The delay would decrease.
- Noise and pollution would be uniformly distributed.

But the introduction of free routing will lead to a major disadvantage. It would be nearly impossible for airspace controllers to detect conflicts between aircraft in advance. Because there are no longer fixed points where conflicts can occur, a system is needed which is able to develop routes between start and destination points in such a way that conflicts with other aircraft are avoided.

3 Formalization of the Problem

When applying genetic algorithms to free routing, one has to be aware of certain facts. Firstly, free-routing is more complicated as e.g. a TSP [2] because there is no predefined number of way points which should be used by the aircraft. Furthermore there are forbidden areas in airspace (e.g. military airspace / thunderstorms) and it

is necessary to avoid conflicts with other moving aircraft. Finally, conflict probability depends on the time when an aircraft arrives at a specific point in the airspace. This leads to a very complicated fitness landscape.

3.1 Formalization

For the formalization of free routing with genetic Algorithms an area of 200 x 200 Nautical Miles (NM) was covered by a grid of 20 x 20 squares of size 10 x 10 NM and 441 numbered grid nodes. A route is defined as a sequence of nodes (way points). Flying a route means moving on the links between the nodes which follow successively in the sequence. Therefore the actual position of the aircraft is not necessarily a grid node. The principle idea of the algorithm is based on the modGA of Michalewicz [6]. However, several modifications have to be included for a significant improvement in performance. The size of the population is 60 chromosomes (or routes).

The information is coded as a chromosome of length 11 with numbers of way points in the first 10 genes which could be used for the definition of a route. The information in the last gene is the number of actually used points which ranges from 1 to 10. Theoretically it is possible to get a solution with the use of all ten genes but several runs with the GA without the last gene have shown very bad results. Because it is necessary to arrange ten points straigth on a line instead of one or two it has taken more than 600 runs for getting just the diagonal between the upper right and the lower left corner of the grid. Like in biological chromosomes the unused information is not lost but stored for the moment it would be needed.

For representing the information in a gene integer numbers were chosen. In this way it is easier to code the node numbers and handle the genetic operators. Applying a crossover operator to a chromosome then means an exchange of way points in a very simple way. In case of mutation it is possible to change the content of a gene direct to a new way point. No repair algorithm is necessary. Furthermore it must be mentioned that the coding of a route is not unique. It is possible to code the same route e.g. with 2 or with 3 points where in the latter case e.g. the second point could lie on the diagonal between the others.

3.2 Evaluation

The following formula describes the evaluation function which is to be minimized:

eval(chromosome) = (length of route) * (1 + 0.2 * (number of conflicts with other aircraft))

For a better handling of local optima this type of definition includes the number of conflicts. Routes with conflicts will not be automatically removed from the population and therefore short routes with a small number of conflicts then have the chance to change under crossover or mutation to good routes without conflicts, e.g. if there is a local optimum surrounded by routes with conflicts, there is now a good possibility to jump over such a barrier of conflicts out of the local optimum.

3.3 Selection

As mentioned before the implemented GA is based on the modGA. Therefore it is necessary to select chromosomes for 3 different groups. Most of the selections are made dependent on the evaluation value (stochastic sampling). The first group includes 30 different chromosomes which remain unchanged. The best 3 chromosomes for this group are chosen by the ELITIST model [5], the remaining 27 with stochastic sampling **without** replacement. The 16 chromosomes of the second group will undergo crossover and the remaining 14 chromosomes in the third group will be treated with the mutation operator. Both are selected with stochastic sampling **with** replacement.

3.4 Crossover

Depending on the 'number of used points' two different types of crossover are applied. If the pair of chromosomes affected by crossover uses more than one way point for at least one chromosome two point crossover is applied. In the other case the operator must be one point crossover because it is not possible to use the normally better two point crossover [8] for a length under two. The crossover points were selected randomly between first and 10th position.

3.5 Mutation

As before for crossover two different types of mutation are in use, each with a probability of 50 %. Again the gene for the mutation is selected with the use of a random generator. We use

- Non-Bounded-Mutation (normal mutation) with the same probability of 1/441 to be selected as the new node for each node of the grid.
- One-Bounded-Mutation with selection of one of the eight points next to the old one.

The second type simulates a limited hill climbing in the surrounding of a given solution.

3.6 Building Block Hypothesis

The building block hypothesis says that schemata with short defining length, low order, and high fitness have the better chance to survive crossover and mutation than other schemata. Applying the building block hypothesis to the algorithm of ROGENA gives the impression that we have to cope with a high defining length for each schema. This is caused by the number of used points which is necessary for each schema and occurs in a last position. Another problem is the time dependence of each solution. E.g. a route with 6 points where the last 3 points form a very short route without conflicts could change to a bad route with conflicts at the last three positions with a mutation on the first three genes. Because of this it is more important to have a good beginning of a route instead of a good ending since each gene will be influenced by the preceding genes.

Altogether this means that good schemata would have a high defining length. But if we look closer at this

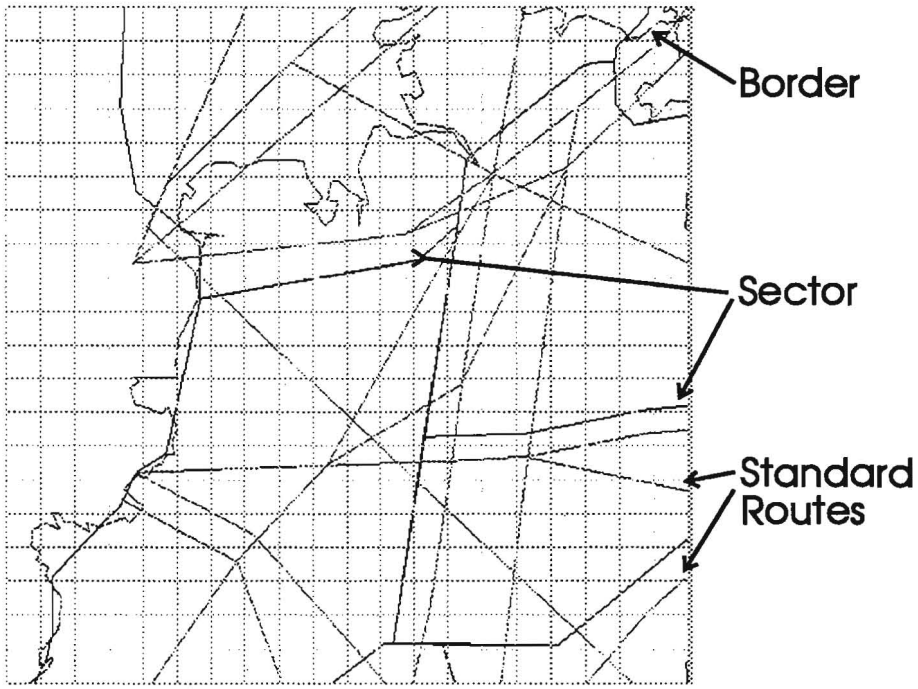


Figure 1: Air space structure over Northern Germany with sectors (black lines), standard routes (grey lines) and the German border.

problem it can be recognized that according to the two point crossover it does not make any difference whether the number of used genes are in the last or in the first position if we treat the chromosomes like acting on a circle [8]. If the last gene is moved to the first position we have a small defining length.

Analyzing the structure of good solutions more carefully shows that a good schema normally has defined first positions and 'don't-care-symbols' at the end. Good schemata with a high defining length are schemata with high order, too.

4 Mutation of the 'Number of Used Points'

At the beginning of the evolution process the routes with high numbers of way points are very long because they were generated by random. In this case the evaluation function leads to bad values. Therefore higher numbers in the 11th positions (number of used genes) of the chromosomes will get lost within the first simulation runs. To prevent this loss a mutation at this gene must be carried out. Again two types of mutation have been applied to this gene:

- random mutation, and
- increase by one ('+1').

As mentioned before, the second type will not lead to a loss of good solutions. As test case a scenario with 20 flights was chosen and 6 runs of ROGENA were conducted for each type. The results of these tests can be found in table 1. The table contains the results for all flight, where the direct route leads to conflicts with other aircraft. The 'Number of A/C' means the number of this flight in the sequence of arriving aircraft within the scenario. For each aircraft 200 runs of ROGENA were made. It can be observed easily that '+1' leads to better results for each flight within this limited number of runs than the random mutation does. But another very important point is the number of nodes used for the found route. As mentioned before increasing the number of used points often leads to bad routes and this makes it difficulty to get routes with higher number in an appropriate time. For type '+1' this increase occured earlier than for the random type (figure 2).

Figure 1 shows the progress of fitness of the best solution for type '+1' and type 'Random'. For both types the graph with the best final result was chosen. It can be easily recognized, that type '+1' shows better results at an earlier state than in the case of type random. Because of this result the type '+1' mutation was chosen as the normal mutation for the gene which contains the number of used points.

5 The Algorithm of ROGENA

ROGENA is a program which is driven by a flight schedule. This schedule contains a number of flights with the following information:

1. Time when the aircraft enters the system.
2. x- and y-co-ordinate of the start-point.
3. x- and y-co-ordinate of the destination-point.
4. Speed.
5. Priority type of flight (number between 1 and 6).

Number of A/C	Type '+1' Ø Nodes	Type '+1' Ø Length	Type 'Random' Ø Nodes	Type 'Random' Ø Length	Direct
7	2	242.74	2	242.74	242.44
9	1	210.44	1	210.44	210.16
10	1.83	120.42	1.16	120.70	119.34
12	1.16	211.57	1	211.57	211.56
15	1.83	188.20	1.5	188.21	188.17
16	1.66	208.83	1	208.93	207.00
20	2.33	191.11	2	194.50	186.37
Ø	1.68	196.18	1.38	196.72	

Table 1: Comparison between mutation types '+1' and 'Random'.

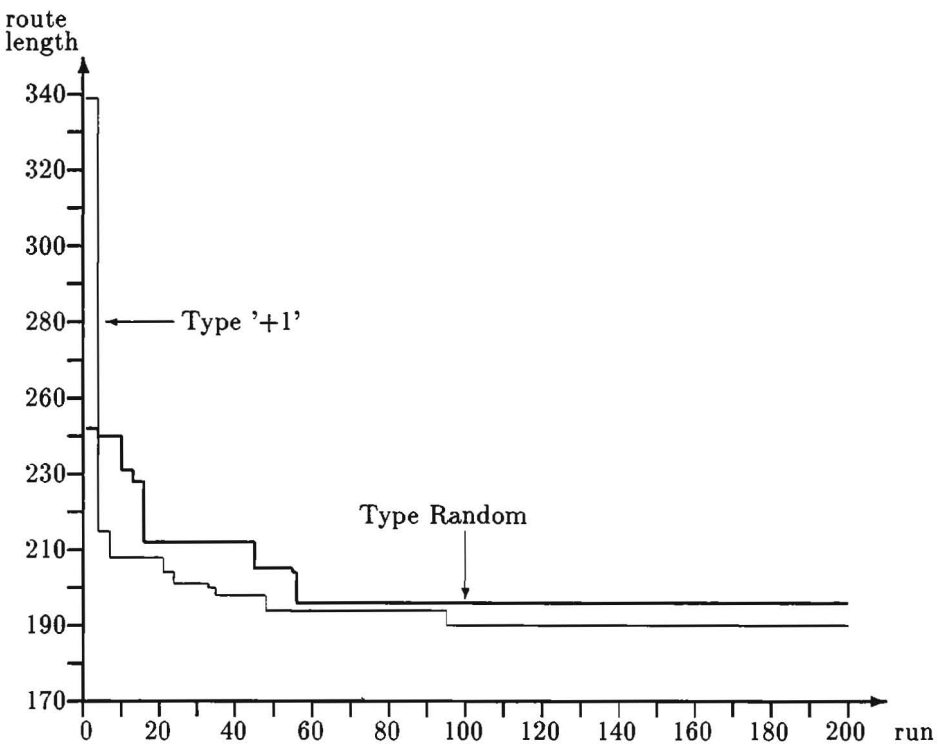


Figure 2: Best solutions of type '+1' and type Random over the time in dependence of the number of runs.

The lowest number could stand e.g. for an aircraft type like B747 with a high speed and a high number of passengers. An aircraft with a high priority type like 6 are e.g. small general aviation aircraft.

An update of the position of the aircraft which has already entered the system is made every 10 seconds of flight time. Distances between aircraft are calculated on the basis of the actual positions.

If a new aircraft enters the system the following process is executed:

Step 1: Checking the direct link between start and destination point for conflicts. If there are no conflicts assign the direct link as flight route for the new aircraft and resume normal simulation. If there are conflicts continue with step 2.

Step 2: Start of the genetic algorithm of ROGENA. Create a first generation of 60 random generated routes between the start and destination point. Evaluate these routes and select 30 routes which will survive this run without changes. Select 16 routes to which crossover will be applied and 14 routes for mutation. Evaluate the new generation and repeat this process until an appropriate conflict-free route is found. If a route is found resume normal simulation with the new route for the new aircraft. If no short route with a good evaluation value is found after a certain number of runs of the genetic algorithm continue with step 3.

Step 3: Check the system for those aircraft which have a conflict with the direct link for the new aircraft. If the new aircraft has a lower priority type than the other aircraft, choose the one with the highest priority type and the highest number of conflicts. Remove this aircraft from the system and construct a new route for the new aircraft according to step 2. After this continue with step 2 for the removed aircraft. If there is no aircraft with a higher priority type assign the best route found in step 2 to the new aircraft and resume simulation.

6 Results

In this section the results of two different experiments with ROGENA are described.

- The first type of experiment (scenario 1) was carried out with 3 randomly generated test-scenarios. Each has included 20 flights with an interval of 30 seconds between the starting times and random-generated speed between 300 and 600 knots (Nautical Miles per hour). For these scenarios a comparison between the length of the direct routes ignoring possible conflicts and the conflict-free routes generated by ROGENA was made.
- Scenario 2 is composed of real aircraft (A/C) trajectories which were extracted from radar data from the north of Germany (figure 1). Starting time for each aircraft was the actual time when the aircraft crossed the border of the grid, respectively. Start and destination points were the positions where the aircraft had entered and left the grid. Each of the three scenarios contained the data of a special flight level. The comparison was made between the measured length of the radar tracks and the routes generated by ROGENA.

For both types of scenarios there was no reassignment necessary, e.g. step 3 of the genetic algorithms did not apply.

A comparison between the length of the direct routes and the ROGENA routes shows that the requirement to avoid conflicts does not increase the length of the route dramatically in spite of the high number of moving aircraft (figure 2). The average loss per conflict is 1.24 NM but this value must be seen in connection with the sum of the route length. The average of the ROGENA routes in percent of direct routes is 100.16 %. It can be said that all aircraft are able to fly a route which is very near to the direct route. If this program would allow more aircraft to enter a sector than in the moment the delay for every aircraft caused by the avoidance of conflicts would be small in comparison to the increased number of handled aircraft.

The results in table 3 which represent scenario 2 with real traffic show just a small number of conflicts and shorter routes for ROGENA than for the standard routes. This confirms the assumption that free-routing works well and would not lead to a high number of conflicts. The gain is caused by using the direct routes. But we have the highest gain in scenarios with more conflicts. The sum of length for all routes are not as high as for the generated scenarios in table 2 because the routes of the traffic scenarios are divided into different levels of airspace and it was necessary to simulate these levels one after the other.

Not all scenarios were as good as the three ones shown above in table 3. For one scenario a bad value for a ROGENA-route was found. The main reason for this was that the route was very short and resolution for the grid points not high enough.

7 Conclusions

The forecasts for the future air traffic demand show a further increase of aircraft movements in order of 4.5 % per year. This will make it necessary to find new strategies for the usage of airspace and to develop new tools which are able to reduce the controller work load.

In order to get a tool which is applicable to the air traffic control system much more details have to be investigated including the possibility to climb and descend to other levels, the flight behaviour of aircraft (e.g. how they are flying curves), smaller squares for the grid and the connection between adjoining sector grids.

Since ROGENA has to rely on the navigation accuracy with which the aircraft are able to fly the assigned route the equipment of the aircraft is very important.

Finally additional theoretical analysis for the algorithm of ROGENA will be carried out including the schema theorem, influence of crossover and mutation operators and the relation between the number of conflicts per route in a population and the average number of

Number of A/C	Conf.	Route Length (Sum) DIRECT	ROGENA	ROGENA in % of DIRECT	Loss per Conflict
20	7	3734.47	3742.17	100.21	-1.10
20	3	3240.99	3244.05	100.09	-1.02
20	3	3152.96	3158.41	100.17	-1.82

Table 2: Scenario 1. Comparison between the length of direct links with conflicts and the length of routes created with ROGENA.

Number of A/C	Conf.	Route Length (Sum) Traffic	ROGENA	ROGENA in % of Traffic	Gain per Route
12	0	834.43	820.29	98.31	1.18
22	2	1647.70	1602.07	97.31	2.02
31	2	1747.24	1714.00	98.10	1.07

Table 3: Scenario 2. Comparison between the length of standard routes in a real traffic scenario and more direct routes generated by ROGENA.

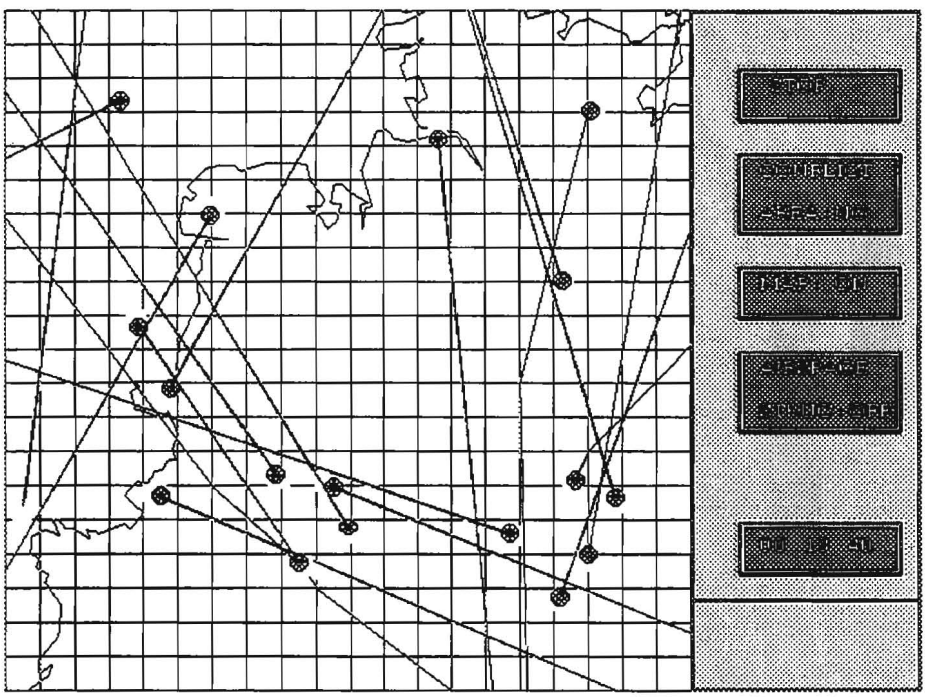


Figure 3: Example for the solution of a generated scenario with remaining route for each aircraft in black, aircraft as circles with radius half of the minimum separation between aircraft and the border in grey.

used nodes. A comparison between the genetic algorithm of ROGENA and other optimization algorithms like simulated annealing and hillclimbing will be carried out. Furthermore there will be a test of the applicability of genetic programming [7] for the creation of routes within ROGENA.

References

[1] Casadei, G. / Palareti, A. / Proli, G.: Classifier System in Traffic Management, Artificial Neural Nets and Genetic Algorithms, in : Proceedings of the International Conference in Innsbruck, Austria 1993.

[2] Fogel, D.B.: An Evolutionary Approach to the Travelling Salesman Problem, Biological Cybern., Vol. 60, pp. 104-109

[3] Fox, B.R. / Mc Mahon, M.B.: Genetic Operators for Sequencing Problems, in: Rawling, G.: Foundation of Genetic Algorithms, Morgan Kaufmann Publishing, Los Altos CA 1991.

[4] Gerdes, I.S.: Application of Genetic Algorithms to the Problem of Free-Routing of Aircraft, in : Proceedings of ICEC'94, Volume II, IEEE, Orlando 1994.

[5] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley Publishing Company, Reading/Massachusetts 1989.

[6] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag Berlin, 1992.

[7] Souček, B. and IRIS Group: Dynamic, Genetic, and Chaotic Programming, Wiley, New York, 1992.

[8] Whitley, D.: A Genetic Algorithm Tutorial, Colorado State University, Department of Computer Science, Technical Report CS-93-103, March 1993.

Genetic Algorithms at Siemens

Jochen Heistermann

Siemens AG München
Otto-Hahn-Ring 6
81730 Munich
ZFE ST SN 44
e-mail: Jochen.Heistermann@zfe.siemens.de

Abstract. The purpose of this contribution is twofold. First, a model will be presented, which allows the estimation of the runtime of Genetic Algorithms (GA). A mathematical model is presented with the purpose of seeing genetic algorithms (GA) from an abstract standpoint. The model sees GA as a process, which copies genes from one generation to the other by preferring good genes to inferior ones. The algorithm finally reaches a point, where there is only one allele left for each gene in the population. Second, an overview of practical GA applications at Siemens will be given. The applications are from different fields like nuclear fuel management, neural learning of problems from speech and vision, fuzzy control or filter development. Most of this paper is part of [Hei94].

1. The Model

1.1 Basic Assumptions

When designing a GA for a specific problem the developer of a GA has to be aware of how the genetic operators work. There are various good explanations with regard to standard operators for example - but there are relative few studies about the exact quantitative influence of the genetic operators during the whole evolutionary process. One purpose of the model presented here is to fill this gap.

From an abstract point of view the main task of GA is to copy genes from one generation to the next. The genes of the actual generation will be spread into the next generation according to their quality. In the beginning a GA has a large gene pool. The better alleles supercede the inferior ones by selection mutation works as a countereffect somehow). Eventually all individuals of the population share the same gene material. Reaching this state means degeneration of the population to a single point in the search space - the algorithm should stop there at the latest. For a designer of GA is of major interest how long this process takes in dependence to his genetic operators.

By inspecting the evolutionary process in the population it becomes necessary to evaluate the mean value and variance of a variable A that counts the number of generations, until there is only one allele left for a single gene position.

How is a generation modelled? Consider a ballot containing n balls. The balls are all different by having individual numbers. The ballot corresponds to one gene position and each ball to one possible allele for that gene. One ball stands for an allele already contained in the population. Assume that all n initial individuals in the starting population have a different allele. The evolution process works in the following way: one ball is taken out of the ballot, copied, put back into the ballot and the new copy saved. Repeat this n times and finally a duplicate ballot is produced.

If a ball with number K_x ($x \in \{1,2,...,n\}$) is not copied it dies out. The question is, "how long will it take until the population is reduced to one ball?". More precisely:

1) What is the mean value of A (A counts the number of generations, which is equivocal of the number of ballots used) until there is only one sort of balls K_x remaining?

2) What is the standard deviation of A?

From intuition one always expects the same distribution of balls in the ballot, but by random drift all but one sort will eventually die out (Random drift is generally described using the elementary probability theory), for example [Fel68]). Convergence to a final state (final state meaning - only one ball with some K_x remaining) can be described by markov chains [Kem60]. A ballot with i balls of sort K_x ($i \in \{0,1,2,...,n\}$) corresponds to a system state with i zeroes and (n-i) ones. The state s_i ($i \in \{0,1,2,...,n\}$) shows the actual number i of a sort of balls K_x. Of significant interest is the probability distribution p_{ij}, which contains the probabilities that the system changes from state s_i to s_j. Each member s_i in the process chain of successive states depends only on its predecessor - the process requires no memory. The p_{ij} can be placed into a matrix, which contains all transitional probabilities in order to move from one state s_i to s_j ($i,j \in \{0,1,2,...,n\}$). For example: $s_i = 5$ and $s_j = 11$ - meaning that five balls of K_x are in the first ballot and eleven balls of K_x in the new one. The states of the markov chain are binomial distributed:

$$p_{ij} = \binom{n}{j} * \left(\frac{|K_x|}{n}\right)^{j} * \left(\frac{n-|K_x|}{n}\right)^{n-j},$$

with $|K_x| = i$ the number of balls of type K_x. It is then possible to construct a matrix $\mathbf{P}'$ containing all state transitions p_{ij} ($i,j \in \{0,1,2,...,n\}$). The states 1,2,...,n-1 are transient; only 0 and n are stationary. The matrix $\mathbf{P}'^{m}$ contains the mean values that s_i changes to s_j in exactly m ($m \in \mathbb{N}$) steps. To calculate how often a state j is visited within the whole process one formulates the equation:

$$sum(j) = \sum_{m=1}^{\infty} \sum_{i=0}^{n} p'^{m}_{ij}.$$

The matrix $\mathbf{P}$ is defined as the sum of the infinite chain of the infinite number of transition matrices. The sum has a finite value, because the system reaches a final state (0 or 1) with probability 1 [Fel68]. The most interesting fact is that the sum of each row contains the mean value of how long it takes to reach one of the final states.

The next step is to build a matrix $\mathbf{Q}$ by removing the rows and columns with number 0 and n from $\mathbf{P}$. $\mathbf{Q}$ then contains information exclusiviely about the transient states. This matrix will be discussed during the rest of the paper. The mean value of A (A is the number of generations before reaching a final state) will be calculated. The probability of $A=1$ ist p_{in} (with i usually 1 in the initial population). The probability for the direct transition from some state s_i to s_n is $(i/n)^n$. The equation required to arrive at the final state in exactly j steps is

$$P(A=j) = (n\,0\,...\,0)\,\mathbf{Q}^{j-1} * \left(\left(\frac{1}{n}\right)^{n}\left(\frac{2}{n}\right)^{n} ... \left(\frac{n-1}{n}\right)^{n}\right)^{T}.$$

We can now compute the mean value of A by summing up the $P(A=j)$ for $1 \le j \le \infty$, which is modelled as a finite sum in the form

$$E(A) = (n\,0\,...\,0) \sum_{j=1}^{\infty} j * \mathbf{Q}^{j-1} * \left(\left(\frac{1}{n}\right)^{n}\left(\frac{2}{n}\right)^{n} ... \left(\frac{n-1}{n}\right)^{n}\right)^{T}.$$

This formula can be brought to a closed form, what is technically complicated. The details are presented in [Hei93]. The final form, which shows a formula without infinite sums is given as

$$E(A) = (n\,0\,...\,0)\,(\mathbf{I}-\mathbf{Q})^{-2}$$

$$\left(\left(\frac{1}{n}\right)^n \left(\frac{2}{n}\right)^n \dots \left(\frac{n-1}{n}\right)^n \right)^T .$$

When computing the standard deviation σ one has also to evaluate $E(A^2)$:

$$E(A^2) = (n\,0 \dots 0) \sum_{j=1}^{\infty} j^2 * Q^{j-1} \left(\left(\frac{1}{n}\right)^n \left(\frac{2}{n}\right)^n \dots \left(\frac{n-1}{n}\right)^n \right)^T .$$

and the eventual result being (see again [Hei93] for the details)

$$E(A^2) = (n\,0 \dots 0)\ (I+Q)\ (I-Q)^{-3} \left(\left(\frac{1}{n}\right)^n \left(\frac{2}{n}\right)^n \dots \left(\frac{n-1}{n}\right)^n \right)^T .$$

This formula obtains the mean value and standard deviation of A.

Figure 1 shows the statistics for E(A) and σ(A). The x-axis contains the various population sizes. The y-axis shows the outcome for E(A) and σ(A). The medium curve models E(A) and the two remaining curves show the range of the standard deviation.

E(A) develops linearly - a result also achieved by Goldberg [Gol87]. The convergence shown in figure 1 needs more generations, because Goldberg looked exclusively at binary coding of genes. The deviation is very widespread leading to very different result in practise.

1.2 The Multi-Gene Model

Until now we have only considered one single gene position. For "real life examples" one would not wait until the last position had reached a homogenous state, but rather interrupt the GA when a percentage of genes are in their final state. It is possible to estimate this interruption point by using Tschebyscheff's inequation:

$$P(|X - E(X)| < a) \geq 1 - \frac{\sigma^2}{a^2}.$$

Generally the fact that a percentage z of the genes are in their final state is enough to stop the algorithm in practise. With a value of $\sigma^2/a^2 = 1 - z$ we can arrive at the further equation of:

$$a = \sigma \sqrt{\frac{1}{1-z}}.$$

The minimal percentage of genes, which should be in their final state, is thus assessable. The value of "a" can be computed from the formula above.

Figure 2 presents results, which are an expansion of figure 1. The x- and y-axis have the same meaning as in figure 1. The lowest curve shows the probability that 10% or more of the genes are in their final state. The remaining curves show the probabilities for the different percentages of genes reaching their final state. For example: a population size of 50 takes more than a 100 generations to assume that 75% of the genes have reached their final state.

1.3 Selection

To provide selection the model from chapter 2 has to be enlarged in a way that selection of the balls corresponds to their quality. Concerning the quality variation of the balls one uses differing matrices **P** and **Q**.

$$p_{ij} = \binom{n}{j} * \left(\frac{i * \mu(K_x)}{i * \mu(K_x) + (n-i) * \mu(\rho - K_x)} \right)^j * \left(\frac{(n-i) * \mu(\rho - K_x)}{i * \mu(K_x) + (n-i) * \mu(\rho - K_x)} \right)^{n-j},$$

This quality dependent approach leads to a new formula for the matrix elements p_{ij}. "i" denotes the number of K_x, (n-i) the number of the remaining balls in the ballot. ρ-K_x stands for all of the balls with the exception of K_x, and $\mu(K_x)$ being the quality of K_x. Each allele will get an own quality - we would like to demonstrate how quality dependent selection will shorten the convergence rate. Experimenting with varying selection scenarios leads to the following results:

Surprising results were attained when comparing the mean values of A in relation to the various selection advantages. A selection advantage of 70/30 leads to a cut in the expectation of A from 196 to 5 at a population size of 100. For larger populations the difference will be even greater. From the graph one can state that selection reduces the linear growth of A with the number of individuals to a logarithmic growth.

2. The Applications

2.1 Nuclear Fuel Management

The search for an optimal arrangement of fresh and burnt fuel and control material within the core of a pressurized water reactor represents a formidable optimization problem. In-core fuel management for pressurized water reactors entails identifying the arrangement of fresh and partially burnt fuel and burnable poisons within the core that optimizes the performance of the reactor over the next operating cycle, while ensuring that operational constraints are always satisfied. Typical objectives might be to maximize the cycle-length, to minimize the power peaking or the individual assembly and region averaged burnups. The optimization problems consists of shuffling the assemblies around in the core and to evaluate the consequences with a simulation program based on partial difference equations.

A core contains 193 fuel assemblies with quarter core symmetrie. After each cycle around 25% - 35% of the fuel elements must be replaced. The old fuel has to be rearranged to yield optimal core performance. The optimization problem is devided into several (dependent) steps.

1. Select a number of fresh fuel elements.
2. Select k out of n elements to fill the core and store the others for one more cycle.
3. Place the elements selected under 1) and 2) into the core.
4. Give each element an orientation (each element has four orientations).

A GA for possible solving the problem:
1. Build an initial population
Choose a number how many new elements will be taken for each individual. Take randomly 193-n elements from the rest. Give those elements a random orientation.

2. Genetic operator crossing-over
Make a cut through the core and take one half of the cut from one random element of the population and the other half from another random choosen individual. Combine the core elements with respect to replace fuel elements, which may be contained twice in the new individual by random fuel elements.

3. Genetic operator mutation
Mutate an individual by
- changing the orientation of single elements.
- switch two elements in the core randomly.
- replace core elements with elements from the reserve objects.

A different genetic approach to that problem is discussed in [Poo93]. Our work is now concentrated on building models to evaluate the goal function.

2.2 Neural learning of problems from speech and vision

Two real world examples were choosen from speech and image processing for evaluating different learning approaches. The data were used for practical industrial applications at Siemens. Many experiments were

performed and their results were carefully analyzed.

A phoneme is the smallest significant language unit. The data sets were coded from colloquial language. Each 10 ms a discrete fourier transformation about the last 20 ms of fluently speech was performed to code the short term language signal. The signals were transformed to cepstral coefficients, which were represented by a vector of 16 real numbers. The second application dealt with hand written numbers. The numbers were stored as binary pictures with different size and orientation. To avoid the use of too many neurons the pictures were rastered with 16x16 pixels. The NN's had to identify all those pictures correctly.

We implemented several algorithms - two of them genetic to compare their performance for problems of varying difficulty.

The two genetic based algorithms :
GA + Grad(GA)
GA have its strength in exploring the database in the beginning of the optimization process. They can find promising regions of the search space with high probability. Gradient search algorithms (Grad) are strong dependent on their starting position (in case of a non-convex search space) being strong in fast convergence. It is obvious that an algorithm, starting with GA to find a good starting point and finishing the training session with Grad, will be a promising approach. The strength of both algorithms could thereby combined.

Grad(GA)
The optimal step size varies through gradient search from step to step. Optimization theory offers a variety of line search methods. Those methods required a lot of objective function computations - in case of NN they might be very expansive. To solve this problem, a simple GA could help. After having worked out the gradient the objective function was computed using the actual step size. The step size was then mutated to a larger and a shorter value and the two corresponding objective function values were computed. The best of the three values was taken. This method offered a simple, fast and very smooth adaptation to the step size. Practical experiments showed that the optimal step size changed slowly within short time intervals, but dramatically for longer terms. The other four algorithms are based on gradient descent methods combined with methods like conjugate gradient search. The six different optimization algorithms were compared with respect to the two examples. Two examples based on data from practical applications were studied for different problem sizes. Larger problems were solved best by a hybrid algorithm, where a GA first found a good starting point for gradient descent. The step size of the gradient search was also controlled by a simple genetic algorithm. If the problems were medium sized, a gradient method with genetic controlled step size outperformed all other algorithms. Simple problems were solved easily by all the proposed algorithms. Error backpropaga-tion was the best approach, because it did not calculate the objective function.

2.3 Fuzzy Control

Fuzzy systems have some parameters like membership functions or rules, which have influence on the behavior of the system. If the fuzzy system can be evaluated by a goal function the development of the fuzzy controller is a complex optimization problem itself. Genetic optimization is used on two different problem areas in the fuzzy system: first, the exact position of the membership function and second, one bit for each rule, which switches rules on and off.

At Siemens there is some work in progress to evaluate the usefulness of GA for fuzzy systems [Tau93], [Wol93]. Basically, Tautz differs between two types of fuzzy systems - driven by expert knowledge or adaptive to an objective function. The second type of fuzzy system is a candidate for optimization with GA.

The first problem is then coding of the fuzzy system. Tautz codes the membership functions by five genes. Assuming that the membership functions are trapezoid-shaped

its shape can be fixed by the four corner points. In addition to that one more gene switches the membership functions on or off. Each rule of the fuzzy system is controlled by two more genes. The first gene switches the rule on/off and the second gene chooses a membership function. if the condition of the corresponding rule is fulfilled.

As an example to evaluate his system Tautz choosed a fuzzy-PI-controller . With the GA the controller was able to adapt himself to global optimum for different tasks by using between 2000 and 5000 individuals.

2.4 Filter development

Acustic filters are important components in the fast growing filter market. There is a linear connection between the geometrical structure of the filter and its function. To optimize the structure of the filter a genetic algorithm is used as part of the automatic filter design tool [Hei93].

The filters have the task to strenghten the incoming signal in a special frequence area and to surpress the signal otherwise. The function of the filter can easily be computed from the filter design by using Fast Fourier Transformations.

The structure of the filter was modeled by a very specialized GA, which was optimized in performance to construct filters in less than two minutes. The algorithm is now part of the filter development design tool.

References:

[Akt90] Aktas, A et al.; Classification of Coarse Phonetic Categories in Continuous Speech: Statistical Classifiers vs. Temporal Flow Connectionist Network; Intern. Conf. on Acoustics, Speech, and Signal Processing; New Mexico 1990.

[Fel68] Feller,W.: An introduction to probability theory and its applications: Wiley New York 1968.

[Gol87] Goldberg,D.E.: Finite Markov Chain Analysis of Genetic Algorithms: in Grefenstette,J.J.: 2nd International Conference on Genetic Algorithms; Lawrence Earlbaum Assoc., Hillsdale 1987.

[Gol89] Goldberg,D.E.: Genetic algorithms; Addison-Wesley 1989.

[Hei91] Heistermann, J.: A Parallel Hybrid Learning Approach to Artificial Neural Nets; Proceedings of the third IEEE Symposium on Parallel and Distributed Processing; Dallas 1991.

[Hei92] Heistermann,J.: Genetische Algorithmen und ihre Anwendung als Lernverfahren für neuronale Netze; Dissertation an der J.W.v.Goethe -Universität; Frankfur t am Main 1993.

[Hei93] Heistermann, J.: Schropp, I.; Ruppel, C.: Optimierung von Oberflächenwellenfiltern mit genetischen Algorithmen; NTIV1 (Siemens-interne Tagung); München 1993.

[Hei94] Heistermann.J.: Genetische Algorithmen - Theorie und Praxis evolutionärer Optimierung; Leipzig 1994.

[Hol75] Holland,J.H.: Adaptation in Natural and Artificial Systems; Ann Arbor The University of Michigan Press 1975.

[Kem60] Kemeny,J.G.; Snell,J.L.: Finite Markov Chains; Van Nostrand Princeton 1960.

[Poo93] Poon, P.W.; Parks, G.T.: Application of Genetic Algorithms To In-Core Nuclear Fuel Management Optimization; Joint International Conference on Mathematical Methods and Supercomputing In Nuclear Applications; Karlsruhe 1993.

[Pre88] Press, W.H.; Flannery, B.P.; Teukolsky, S.A.; Vetterling, W.T.: Numerical Recipes in C; Cambridge University Press New York 1988.

[Rec73] Rechenberg,I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution; Friedrich Frommann Stuttgart-Bad Cannstatt 1973.

[Rum86] Rumelhart, D.E.; Hinton, G.E.; Williams, R.J.: Learning Internal Representations by Error Propagation; in Rumelhart. D.E.; McCLelland. J.L. (Eds.): Parallel Distributed Processing Vol.1; MIT Press Cambridge Massachusetts 1986.

[Sch77] Schwefel,H.-P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie; Birkhäuser Basel und Stuttgart 1977.

[Tau94] Tautz, W.: Genetische Algorithmen zum Entwurf von Fuzzy-Systemen; NTIV2 (Siemens-interne Tagung); Nürnberg 1994.
[Tro91] Troll, A.: Optimierungsverfahren für die Lernphase bei Neuronalen Netzen; Diplomarbeit an der LMU München, Fachbereich Mathmatik; München 1991.
[Whi89a] Whitley, D., Bogart, C.: The evolution of connectivity: Pruning neural networks using genetic algorithms; Proc. of the 3rd Intern. Joint Conf. on Neural Networks 1989.
[Whi89b] Whitley, D., Hanson, T.: Optimizing neural networks using faster, more accurate genetic search; Proc. of the 3rd Intern. Conf. on Genetic Algorithms 1989.
[Wol94] Wolf, T.: Optimierung von Fuzzy-Systemen mit Neuronalen Netzen und genetischen Algorithmen; NTIV2 (Siemens-interne Tagung); Nürnberg 1994.

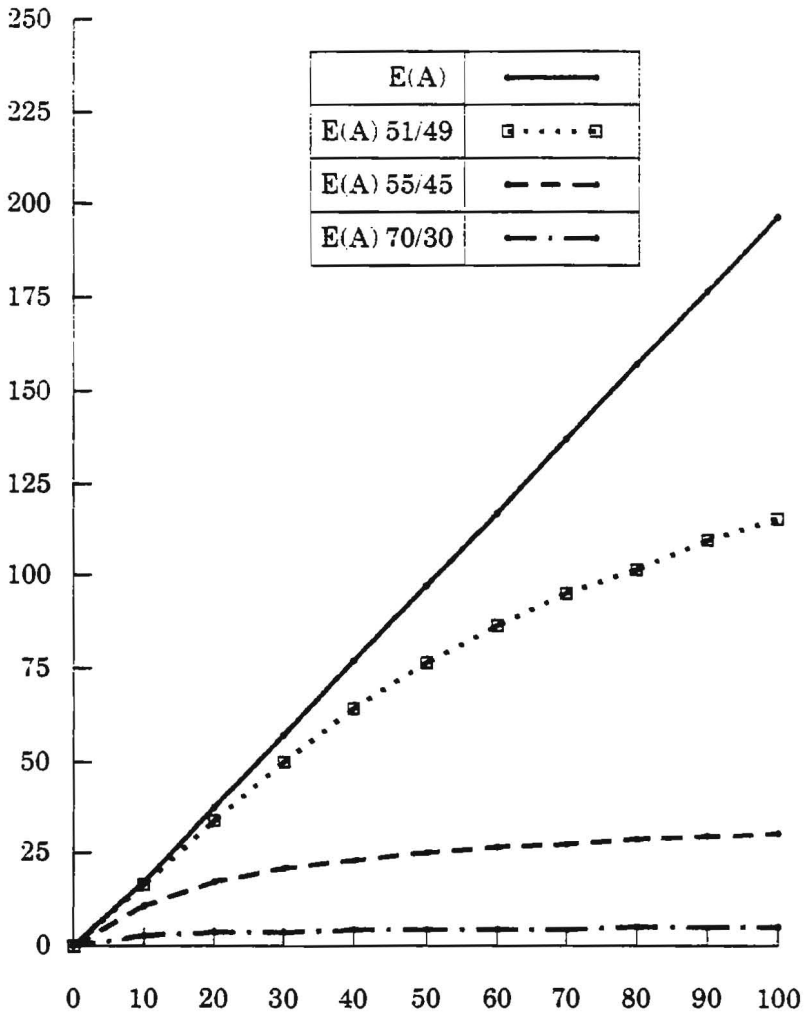


Figure 1: Mean value and standard deviation of A

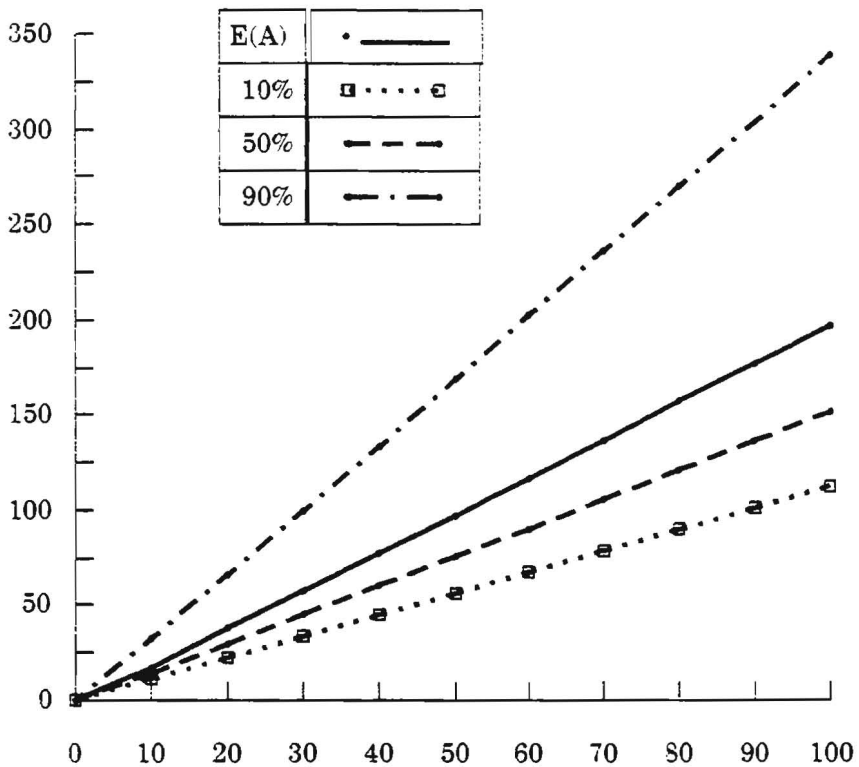


Figure 2: E(A) for multiple genes

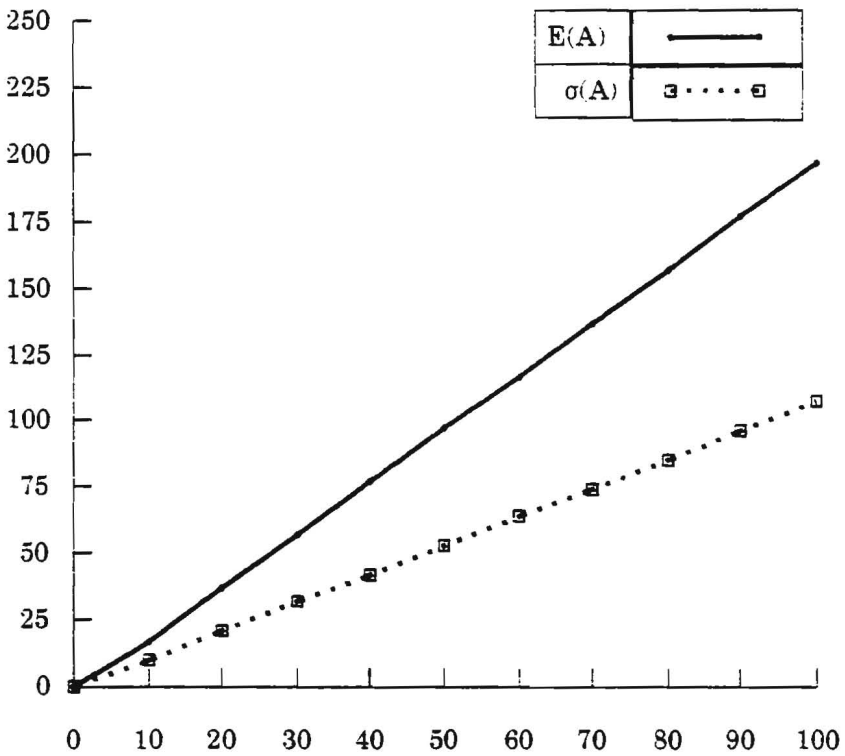


Figure 3: Convergence to a final state under selection

Authors Index

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory pub/papers/reports. If you have any questions concerning ftp access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `kraemer@mpi-sb.mpg.de`

MPI-I-94-241	J. Hopf	Genetic Algorithms within the Framework of Evolutionary Computation: Proceedings of the KI-94 Workshop
MPI-I-94-239	P. Madden, I. Green	A General Technique for Automatically Optimizing Programs Through the Use of Proof Plans
MPI-I-94-238	P. Madden	Formal Methods for Automated Program Improvement
MPI-I-94-235	D. A. Plaisted	Ordered Semantic Hyper-Linking
MPI-I-94-234	S. Matthews, A. K. Simpson	Reflection using the derivability conditions
MPI-I-94-233	D. A. Plaisted	The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison
MPI-I-94-232	D. A. Plaisted	An Abstract Program Generation Logic
MPI-I-94-230	H. J. Ohlbach	Temporal Logic: Proceedings of the ICTL Workshop
MPI-I-94-228	H. J. Ohlbach	Computer Support for the Development and Investigation of Logics
MPI-I-94-226	H. J. Ohlbach, D. Gabbay, D. Plaisted	Killer Transformations
MPI-I-94-225	H. J. Ohlbach	Synthesizing Semantics for Extensions of Propositional Logic
MPI-I-94-224	H. Aït-Kaci, M. Hanus, J. J. M. Navarro	Integration of Declarative Paradigms: Proceedings of the ICLP'94 Post-Conference Workshop Santa Margherita Ligure, Italy
MPI-I-94-223	D. M. Gabbay	LDS - Labelled Deductive Systems: Volume 1 — Foundations
MPI-I-94-218	D. A. Basin	Logic Frameworks for Logic Programs
MPI-I-94-216	P. Barth	Linear 0-1 Inequalities and Extended Clauses
MPI-I-94-209	D. A. Basin, T. Walsh	Termination Orderings for Rippling
MPI-I-94-208	M. Jaeger	A probabilistic extension of terminological logics
MPI-I-94-207	A. Bockmayr	Cutting planes in constraint logic programming
MPI-I-94-201	M. Hanus	The Integration of Functions into Logic Programming: A Survey
MPI-I-93-267	L. Bachmair, H. Ganzinger	Associative–Commutative Superposition