Fuzzy Evolutionary Algorithms

Hans-Michael Voigt*

TR-92-038

June 1992

Abstract

Evolutionary algorithms (EA) combine different approaches for solving complex problems based on principles, models, and mechanisms of natural evolution. Typical representatives of such algorithms are Genetic Algorithms(GA) and Evolution Strategies (ES), which are closely related in principle but show different emphasis on the representational and operational level. The basic ideas and concepts for GAs and ESs dates back to the early sixties. Central concepts of these approaches include the replication, recombination, mutation, selection, isolation–migration, and diffusion of individuals within or between populations or subpopulations, respectively. These algorithms do not take into account the development of an individual or organism from the gene level to the mature phenotype level. This development is a multistage decision process influenced by the environment and by interspecific as well as intraspecific competition and cooperation such that usually no inferences can be drawn from phenotype to genotype. The goal of this paper is to introduce a fuzzy representation and fuzzy operations to model the developmental process based on fuzzy decisions. Some first conclusions with respect to optimization will be stated.

The appendices include an up–to–date software survey for Evolutionary Algorithms and the description of "The Evolution Machine".

[1]International Computer Science Institute (ICSI),1947 Center Street, Suite 600, Berkeley, CA 94704, e–mail: voigt@icsi.berkeley.edu. On leave from the Technical University Berlin, Bionics and Evolution Techniques Laboratory, Bio – and Neuroinformatics Research Group, Ackerstrasse 71 – 76 (ACK1), D – 1000 Berlin 65, e-mail: voigt@fb10.tu–berlin.de and the Center for Applied Computer Science (GFaI), Rudower Chaussee 5, D – 1199 Berlin

Contents

1 Introduction

Natural evolution is a highly complex and very adaptive process. There seems to be always an answer by this process to an ever changing environment as long as these changes are slow and moderate. This process is mainly based on information proliferation and mixing which determines to a certain extend the physical realization corresponding to the information. The basic laws of natural evolution and genetics were discovered by the fundamental work of **Charles Darwin** [1] and **Gregor Mendel** [6].

The work of Darwin led to highly controversial philosophical discussions whereas the work of Mendel was unknown until the beginnings of the 20th century.

From then on the modelling of genetic and ecological processes for a deeper understanding were emphasized more and more and it culminated in the "Golden age of theoretical ecology" during the twenties of our century.

This development is especially due to such scientists as Fisher, Wright and Haldane with respect to genetics and population genetics and with Volterra and Lotka concerning the modelling of ecological processes. The famous Lotka–Volterra–Model stands for this achievements.

As it is the evolution of a population including genetic laws can be perhaps sketched by Figure 1. This figure reflects the generation of new individuals as replicating units of a population by genetic operators like recombination, crossing–over, inversion, deletion including erroneous replication due to mutations. By these operators one gets the genotype of an individual. Even these operators are subject to influences from the environment, especially with respect to mutations by e.g. radiation. The genotype will then be expressed via several stages also influenced by the environment, e.g. by metabolic processes, to a structural phenotype. This intermediate stage goes through an individual adaptation process to be a mature phenotype. All these different levels of development from a genotype to a mature phenotype may be determined as a developmental process. Once more it should be emphasized that the necessary transformations are all subject to environmental influences. This phenotype represents a number of attributes which are aggregated to a fitness value. Here fitness is also determined by the environment which on the other hand generates selection processes over the fitness values. These selection schemes include e.g. intra– and inter–specific cooperation and competition, mating success, figure of merit for offspring production etc.

As nature shows evolution processes based on the sketched outline are highly adaptive.

This was the basis to use models gleaned from natural evolution to adaption, especially optimization, problems.

The first ideas and experiments originate from the early sixties. They were independently developed in Germany by Ingo Rechenberg [7] and in the USA by John Holland [4].

These approaches were extended and further refined (Hans–Paul Schwefel [8], David Goldberg [2]).

The algorithms proofed to be very useful for optimization though there are only a few theoretical foundations up to now. As we will see later non of these approaches take an individual development into account.

Since the beginnings of the eighties there is a strong intensification of research in this field which is highlighted by the **International Conferences on Genetic**

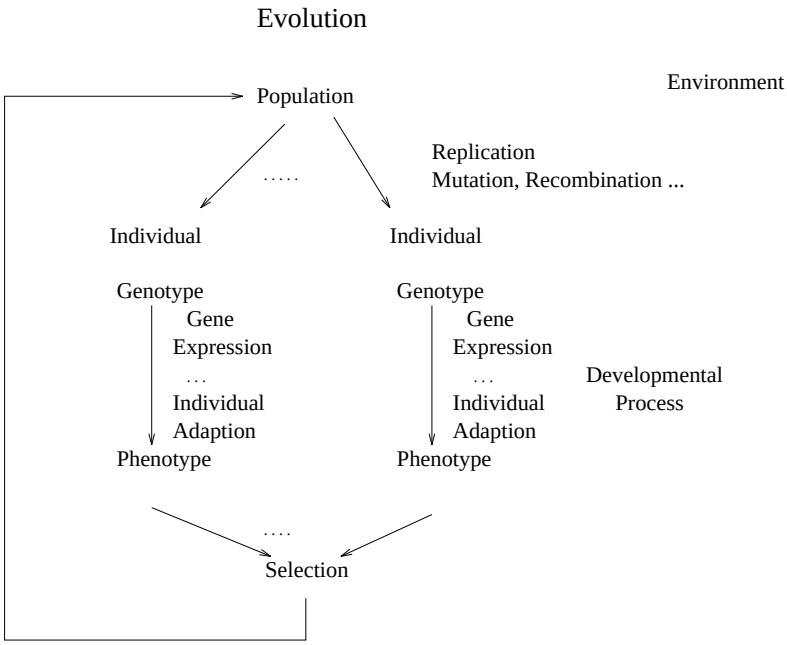


Figure 1: General Evolution Schema

Algorithms (1985, 1987, 1991) in the USA and the **International Conferences on Parallel Problem Solving from Nature** (1990, 1992) in Europe.

2 Evolutionary Algorithms

A very careful comparison of the present state of the art for Genetic Algorithms and Evolution Strategies is given by Hoffmeister & Bäck [3]. Therefore, we want to recall only the basic notions concerning the genetic representation and the genetic operators.

Common to both approaches are the following algorithmic steps which reflect the origin from an evolution point of view.

Step 1 Choose a genetic representation of an individual which is appropriate for the problem to be solved

Step 2 Generate at random an initial population of individuals and compute their fitness

Step 3 Generate a new population by genetic operators (mutation, recombination, crossing–over), compute the fitness values, and apply selection based on the fitness values. Repeat **Step 3** until specified conditions are met.

This very rough description may serve as a basis for every GA and ES. As it is clear, this algorithmic sketch raises a number of questions:

- What is an appropriate genetic representation?
- What are the corresponding genetic operators?
- What is the mapping from genotype to phenotype since selection is based on the phenotype?
- Does this mapping catch some aspects of a development?
- What kind of selection will be applied to the population?

We will touch the answers to these questions for GAs and ESs as prerequisites for the introduction of Fuzzy EAs.

This will be done in analyzing optimization problems of the form

$$min\{f(p)|p \in X\}, X \subseteq R^n, f : R^n \rightarrow R.$$

2.1 Genetic Algorithms

The basic genetic algorithm [2] can be described in a fairly simple way:

Let us assume that the genotype of an individual g_i, $i = 1, ..., \mu$, of a population P, $g_i \in P$, is encoded by a string of binary values $\{0, 1\}$, e.g.

$$g_i = \left(\begin{array}{cccccc} 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right).$$

By decoding this genotype by e.g. a fixed point integer decoding scheme, were the first three bits decode the first integer and the last three bits decode a second integer, the intermediate phenotype h_i for this genotype will be given by

$$h_i = \left(\begin{array}{cc} 2 & 1 \end{array} \right)$$

The number of bits used for encoding a fixed point integer depends on the desired accuracy.

This intermediate phenotype will be mapped to the feasible region G to form the phenotype, e.g. $G = (5, 10) \times (3, 6)$. Then the phenotype p_i is given by

$$p_i = \left(\begin{array}{cc} 6.43 & 3.43 \end{array} \right).$$

The fitness depends on the phenotype p_i by the given function f

$$f_i = f(p_i).$$

Having defined the genetic representation, the basic GA can be described by the following steps:

Step 1 Randomly generate an initial population $P(0)$

Step 2 Compute and save the fitness $f(p)$ for each individual p in the current population $P(t)$

Step 3 Define selection probabilities $p_{select}(p)$ for each individual p in $P(t)$ so that $p_{select}(p)$ is proportional to $f(p)$

Step 4 Generate $P(t+1)$ by probabilistically selecting individuals from $P(t)$ to produce offspring via genetic operators and go to **Step 2**

The used genetic operators crossing–over and mutation are introduced in the following way:

2.1.1 1–point–crossing–over

We consider two bitstrings as the genotypes of mother and father, which are broken up at one point, e.g. between the second and third bit position, and interchanged, thus forming two possible new bitstrings. One of the new bitstrings can be chosen as the genotype of a child, i.e.

$$g_{mother} = \left(\begin{array}{cc|cccc} 0 & 1 & | & 0 & 0 & 0 & 1 \end{array} \right)$$

$$g_{father} = \left(\begin{array}{cc|cccc} 1 & 0 & | & 1 & 1 & 0 & 1 \end{array} \right)$$

such that

$$g_{child} = \left(\begin{array}{cccccc} 0 & 1 & 1 & 1 & 0 & 1 \end{array} \right)$$

or

$$g_{child} = \left(\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right).$$

This crossing over is done with a certain uniformly distributed probability $P_{cross-over}$. All possible children with respect to the above given parents are shown in Figure2.

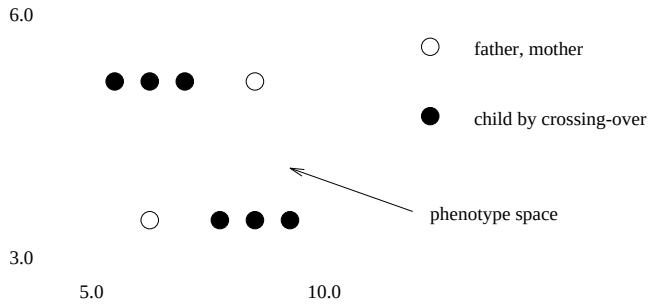


Figure 2: Possible children in phenotypic space

2.1.2 Mutation

Mutation can be introduced by flipping a bit in a child's genotype bitstring

$$g_{child} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

with a certain uniformly distributed probability $p_{mutation}$, such that

$$g_{child} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Here, the first bit was flipped. This is shown in Figure 3.

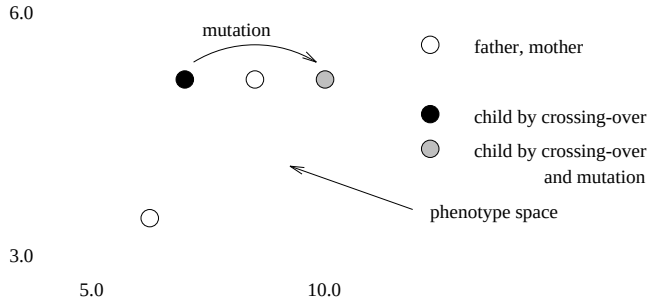


Figure 3: Mutation example in phenotypic space

Now we can answer the stated questions. Obviously, the genetic representation by a bit–string corresponds via the transformation

$$g_i \rightarrow h_i \rightarrow p_i$$

to a one–to–one mapping of the genotype to the phenotype and vice versa. There is no inference problem. Therefore, there cannot exist a developmental process of individuals neither in the form of a static nor dynamic mapping. The selection is done proportional to the fitness values.

It should be emphasized, that the algorithm is principally based on binary decisions because of the binary representation of the genotype.

2.2 Evolution Strategies

The multi–membered Evolution Strategy [8] can be also described in a fairly simple way:

Let us assume that the genotype of an individual g_i, $i = 1, ..., \mu$, of a population P, $g_i \in P$, is encoded by an array of floating–point values, e.g.

$$g_i = \begin{pmatrix} 6.43 & 3.43 \end{pmatrix}.$$

If the genotype values do not match the feasible region G they will be mapped by a certain procedure to this region.

The phenotype is exactly the genotype, i.e.

$$p_i = g_i$$

such that the fitness is given via the function

$$f_i = f(p_i) = f(g_i).$$

Having defined the genetic representation, the multi–membered ES can be described by the following steps:

Step 1 Randomly generate an initial parent population $P_{parents}(0)$ of μ parents

Step 2 Compute and save the fitness $f(p)$ for each individual p in the current parent population $P_{parents}(t)$

Step 3 Generate an offspring population $P_{offspring}(t+1)$ of λ children, $\lambda \geq \mu$, by applying genetic operators to uniformly random chosen parent pairs

Step 4 Generate a new parent population $P_{parents}(t+1)$ by truncation (extinctive) selection of the best individuals from $P_{parents}(t) \cup P_{offspring}(t+1)$ or from $P_{offspring}(t+1)$ only and go to **Step 2**

The used genetic operators recombination and mutation are introduced in the following way:

2.2.1 Recombination

We consider two floating–point arrays as the genotypes of mother and father, e.g.

$$g_{mother} = \left(\begin{array}{cc} 6.43 & 3.43 \end{array} \right)$$

$$g_{father} = \left(\begin{array}{cc} 8.57 & 5.14 \end{array} \right)$$

as in the example for the basic GA.

By discrete recombination the child's genotype will be formed by randomly choosing the values from either the mother or the father array, e.g.

$$g_{child} = \left(\begin{array}{cc} 8.57 & 3.43 \end{array} \right)$$

or

$$g_{child} = \left(\begin{array}{cc} 6.43 & 5.14 \end{array} \right).$$

Intermediate recombination can be done by taking the arithmetic mean of the gene values as the child's gene values, i.e.

$$g_{child} = \left(\begin{array}{cc} 7.50 & 4.29 \end{array} \right)$$

The possible children with respect to recombination are shown in Figure 4.

Recombination is done with a certain uniformly distributed probability $p_{recombination}$.

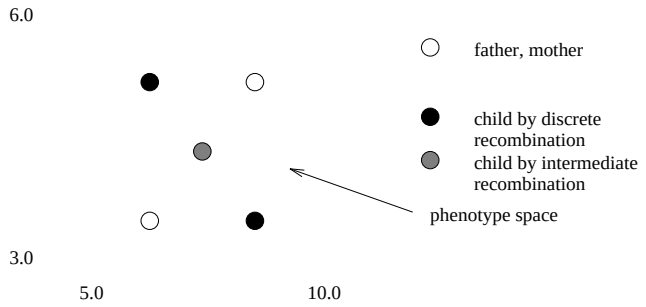


Figure 4: Possible children in phenotypic space

2.2.2 Mutation

The mutation operator is more sophisticated. Mutations are done corresponding to the distribution

$$exp(\mathcal{N}(\delta_{step}) + \mathcal{N}(\delta_{step_i}))$$

where the stepsizes δ_{step_i} are variances of a normal distribution. It should be noted that the stepsizes are treated as gene values such that they are also subject to the genetic operators and selection (mutator genes).

This is shown in Figure 5.

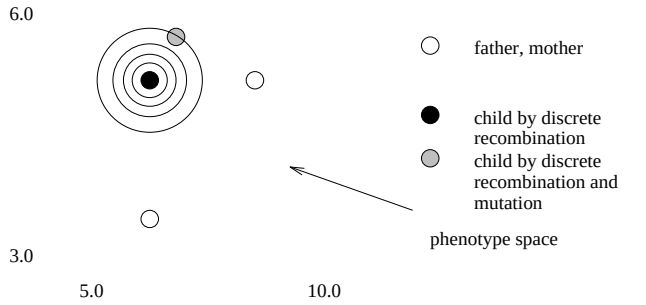


Figure 5: Mutation example in phenotypic space

For more details refer to Appendix B.

Now we can answer the stated questions also for ESs. Here, the genetic representation is given by an array of floating–point values and this representation is at the same time the phenotypic representation, i.e.

$$g_i \equiv p_i.$$

Therefore, we can conclude that there is no individual developmental process. Selection is done by truncation, and extinction, respectively, of a certain percentage of the population. This algorithm is much more in correspondence with local optimization procedures. Decisions are done in the space of phenotypes on the base of continuous variables in the range of the machine accuracy.

3 Fuzzy Evolutionary Algorithms

For the description and simulation of developmental processes a number of models have been created and analyzed as e.g. Lindenmayer–Systems, or L–Systems for short.

Here we want to model such a developmental process as a fuzzy decision process [9], [5] to emphasize the complexity of development from genotype to a mature phenotype. In this sense it is quite natural to introduce a genotypic representation in the following form:

Let us a assume that the genotype of an individual g_i, $i = 1, ..., \mu$, of a population P, $g_i \in P$ is encoded by a string of fuzzy decision variables $\phi \in (0, 1)$, e.g.

$$g_i = \left(\begin{array}{cccccc} \phi_0 & \phi_1 & \phi_2 & \phi_3 & \phi_4 & \phi_5 \end{array} \right).$$

This means, every gene value represents a decision variable whose value may have a range between zero and one.

By an example we want to show how to use such a fuzzy representation for our optimization example.

Let us assume that an individual is encoded by the array of fuzzy gene values

$$g_i = \left(\begin{array}{cccccc} 0.22 & 0.86 & 0.14 & 0.03 & 0.18 & 0.92 \end{array} \right).$$

Now we can apply as in case of the GA a fuzzy fixed point integer decoding scheme where the first three fuzzy decision variables decode the first fuzzy integer and the last three fuzzy decision variables decode the second fuzzy integer. The intermediate phenotype h_i for this genotype is then given by

$$h_i = \left(\begin{array}{cc} 0.22 \cdot 2^2 + 0.86 \cdot 2^1 + 0.14 \cdot 2^0 & 0.03 \cdot 2^2 + 0.18 \cdot 2^1 + 0.92 \cdot 2^0 \end{array} \right)$$

or

$$h_i = \left(\begin{array}{cc} 2.74 & 1.40 \end{array} \right).$$

This intermediate phenotype will be mapped to the feasible region G to form the phenotype, e.g. $G = (5, 10) \times (3, 6)$ as for the GA–example. The phenotype p_i is then given by

$$p_i = \left(\begin{array}{cc} 6.96 & 3.60 \end{array} \right).$$

Just as for GAs and ESs the fitness depends on the phenotype p_i by the given function f

$$f_i = f(p_i).$$

Then we can describe the Fuzzy EA by the following steps:

Step 1 Randomly generate an initial parent population $P_{parents}(0)$ of μ parents

Step 2 Compute and save the fitness $f(p)$ for each individual p in the current parent population $P_{parents}(t)$

Step 3 Generate an offspring population $P_{offspring}(t+1)$ of λ children, $\lambda \geq \mu$, by applying genetic operators to uniformly random chosen parent pairs

Step 4 Generate a new parent population $P_{parents}(t+1)$ by truncation (extinctive) selection of the best individuals from $P_{parents}(t) \cup P_{offspring}(t+1)$ or from $P_{offspring}(t+1)$ only and go to **Step 2**

The overall scheme of the Fuzzy EA is quite similar as for ESs since it has been shown that proportional selection as for the basic GA is not the best choice and may lead to a premature end of the evolution process.

Extinctive selection is much more dedicated to the introduction of limited resources or to domestic selection (breeding).

The question is how to introduce fuzzy genetic operators!

3.1 Recombination

This can be done using the union and intersection operators of fuzzy sets for defining all possible children by recombination.

We consider the two arrays of fuzzy decision variables as the genotypes of mother and father, e.g.

$$g_{mother} = \left(\begin{array}{cccccc} 0.22 & 0.86 & 0.14 & 0.03 & 0.18 & 0.92 \end{array} \right)$$

$$g_{father} = \left(\begin{array}{cccccc} 0.86 & 0.01 & 0.92 & 0.79 & 0.01 & 0.88 \end{array} \right)$$

and apply the min– and max–rule for computing the appropriate fuzzy subsets

$$g_{child_{min}} = \left(\begin{array}{cccccc} 0.22 & 0.01 & 0.14 & 0.03 & 0.01 & 0.88 \end{array} \right)$$

$$g_{child_{max}} = \left(\begin{array}{cccccc} 0.86 & 0.86 & 0.92 & 0.79 & 0.18 & 0.92 \end{array} \right).$$

This represents the set of all fuzzy subsets as children. The operation is shown in Figure 6.

Finally, the child will be defined at random by using a uniform probability distribution applied to the range of all fuzzy min and max values, e.g. for the first fuzzy decision variable we choose with a uniform probability a value from the range $(0.22, 0.86)$.

3.2 Mutation

Mutation are introduced by a uniform mutation probability $p_{mutation}$. The mutation probability will be mapped by an inverse encoding scheme onto the fuzzy decision variables, e.g. having given an array of fuzzy decision variables by

$$\left(\begin{array}{cccccc} \phi_0 & \phi_1 & \phi_2 & \phi_3 & \ldots & \phi_n \end{array} \right)$$

where the fuzzy integer is given by

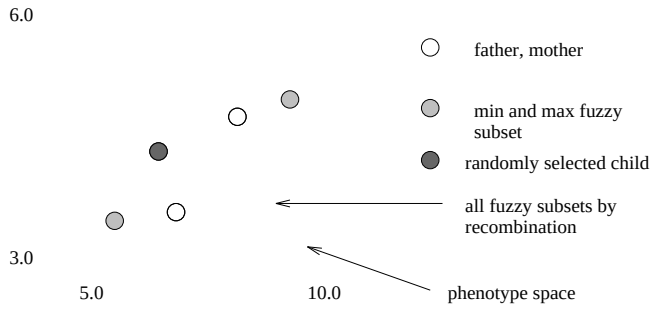


Figure 6: Possible children in phenotypic space

$$\sum_{i=0}^{n} \phi_i 2^i.$$

Then the probability for mutation will be determined by the inverse encoding, i.e.

$$p_{mutation}(\phi_i) = p_{mutation} \cdot \frac{2^{n-i}}{2^{n+1}-1}.$$

The mutated decision variables are drawn at random from the interval $(0, 1)$. This is shown in Figure 7.

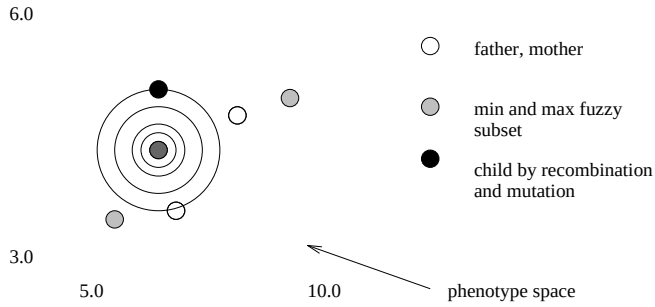


Figure 7: Mutation example in phenotypic space

Obviously, in case of a fuzzy representation there is no one–to–one mapping from genotype to phenotype and vice versa. Different genotypes may lead to the same phenotype. Recombination can be interpreted as a weak contraction operator. Weak means here that a child need not necessarily be located in the cube spanned by the parents fuzzy variables. The weakness depends on the number of fuzzy variables used for encoding a fuzzy integer. Contrary to GAs, fuzzy integers allow a continuous encoding of the feasible region. The mutation probabilities are dependent on the position of the fuzzy decision variable within the encoding scheme to avoid large jumps.

First simulation results show a rather good problem solving behavior but has to be confirmed by a careful simulation study.

References

[1] Ch. Darwin. On the Origin of Species by Means of Natural Selection. London: John Murray 1859

[2] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison–Wesley 1989

[3] F. Hoffmeister & T. Bäck. Genetic Algorithms and Evolution Strategies – Similarities and Differences. Papers on Economics and Evolution #9103, European Study Group for Evolutionary Economics ESGEE 1992

[4] J.H. Holland. Adaption in Natural and Artificial Systems. Ann Arbor: University of Michigan Press 1975

[5] B. Kosko. Neural Networks and Fuzzy Systems. Englewood Cliffs: Prentice Hall 1992

[6] G. Mendel. Versuche über Pflanzenhybride. Verhandl. Naturf. Vereins. Brünn, Bd. 4, Abhandlungen, Bünn 1866 & 1870

[7] I. Rechenberg. Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart: Frommann–Holzboog 1973

[8] H.–P. Schwefel. Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie. Basel und Stuttgart: Birkhäuser 1977

[9] L.A. Zadeh. Fuzzy Sets. Information and Control, vol. 8, 338–353, 1965

A Evolutionary Algorithms Software

This section contains a rather comprehensive survey of available software packages for Evolutionary Algorithms (EA). It was collected by Nici Schraudolph [24] from the Computer Science Department of the University of California at San Diego as part of the workshop on Software Support and Test Functions at the 4th International Conference on Genetic Algorithms. You can get it via anonymous ftp from ftp.aic.nrl.navy.mil from the file /pub/galist/information/ga-software-survey.txt. It corresponds to the updated version of February 25, 1992.

I have added the Evolution Strategy Library evo which is written by Karsten Trint and Uwe Utecht from Technical University at Berlin. This library contains also subpopulation models.

It should be mentioned that the Evolution Machine is the only package which integrates Evolution Strategies and Genetic Algorithms in a common programming environment. Appendix A contains the manual for its use. At present The Evolution Machine will be updated and transferred to SUN workstations.

This list is included in the report to give you the opportunity to make your own experiences with Evolutionary Algorithms since most of the packages are free of charge and available via ftp.

The legend for the descrition of the software packages is as follows:

- **NAME** Software name
- **TYPE**
  - **GE** General Genetic Algorithm
  - **SS** Steady–state Genetic Algorithm
  - **ES** Evolutionary Strategy
  - **OO** Object–oriented
  - **AL** Artificial Life (implicit fitness)
- **OS** Operating System (X11 implies UNIX)
- **LANG** Programming Language (source code not included)
- **PRICE**
  - **(1)** free to governement contractors, other pay nominal fee
  - **(2)** available on license basis
- **AC** Author or Contact, given as Internet e–mail address if possible

A.1 List of Evolutionary Algorithms Software

NAME	TYPE	OS	LANG	PRICE	AC
GA Workbench	GE	DOS	(C++)	free	mrh@camcon.co.uk (Mark Hughes)
Splicer	GE	Mac X11	C	(1)	bayer@galileo.jsc.nasa.gov (Steve Bayer)
Evolution Machine	GE ES	DOS	(C) (C++)	free	voigt@fb10.tu-berlin.de (Hans–Michael Voigt)
OOGA GENESIS	OO GE	DOS	Lisp C	$60 both	(Lawrence Davis) gref@aic.nrl.navy.mil (John Grefenstette)
GAGA	GE	UNIX	C	free	jon@cs.ucl.ac.uk (Jon Crowcroft)
ESCaPaDE	ES	UNIX	C	free	iwan@gorbi.informatik.uni–dortmund.de (Frank Hoffmeister)
evo	ES	DOS	C		uwe@biene.fb10.tu–berlin.de (Uwe Utecht, Karsten Trint)
GAucsd	GE	UNIX	C	free	nici@cs.ucsd.edu (Nici Schraudolph)
Genitor	SS	UNIX	C	free	whitley@cs.colostate.edu (Darrell Whitley)
GAC GAL	GE	UNIX	C Lisp	free	spears@aic.nrl.navy.mil (Bill Spears)
mGA1.0 SGA–C	GE	UNIX	Lisp C	free	@ualix.ua.edu:rob@galab2.mh.ua.edu (Robert Elliott Smith)

Table 1: General Evolutionary Algorithms Software

NAME	TYPE	OS	LANG	PRICE	AC
SGA–Cube	GE	nCube	C	free	@ualix.ua.edu: rob@galab2.mh.ua.edu (Robert Elliott Smith)
Genetic Genocop Gafoc	GE	UNIX	(C)	free	zbyszek@unccvax.uncc.edu (Zbigniew Michalewicz)
WOLF	SS	Mac UNIX	C	$20 free	drogers@riacs.edu (David Rogers)
Tierra	AL	DOS UNIX	C	$70 free	ray@brahms.udel.edu (Thomas Ray)
Evolver	GE	DOS Mac	(C, Pascal)	$345	(Phil Rybeck, Axcelis Inc.)

Table 2: Specialized Evolutionary Algorithms Software

NAME	TYPE	OS	LANG	PRICE	AC
GENEsYs	GE	UNIX	C	free	baeck@gorbi.informatik.uni–dortmund.de (Thomas Baeck)
GA Framework	OO	Mac	C++	(2)	America Online: Wilson SD (Steve Wilson)
XYpe	SS	Mac	(C)	$275	(Ed Swartz, Virtual Image Inc.)
GAME	OO	X11	C++		J.RibeiroFilho@cs.ucl.ac.uk (Jose Ribeiro Filho)

Table 3: Incomplete Evolutionary Algorithms Software

A.2 Description of Evolutionary Algorithms Software

- **GA Workbench**

  A mouse-driven interactive GA demonstration program aimed at people wishing to show GAs in action on simple function optimizations and to help newcomers understand how GAs operate. Features: problem functions drawn on screen using mouse, run-time plots of GA population distribution, peak and average fitness. Useful population statistics displayed numerically, GA configura- tion (population size, generation gap etc.) performed interactively with mouse. Requirements: MS-DOS PC, mouse, EGA/VGA display. Available on 5.25" disk by request from:

  Mark Hughes
  Cambridge Consultants Ltd.
  The Science Park
  Milton Road
  Cambridge CB4 4DW
  United Kingdom

- **Splicer**

  Splicer is a genetic algorithm tool that can be used to solve search and optimization problems, created by the Software Technology Branch (STB) of the Information Systems Directorate at NASA/Johnson Space Center with support from the MITRE Corporation. Splicer was written in C on an Apple Macintosh, then ported to Unix workstations running X11; it has a modular architecture with well-defined interfaces between a GA kernel, represen- tation libraries, fitness modules, and user interface libraries.

  The representation libraries contain functions for defining, creating, and decoding genetic strings, as well as multiple crossover and mutation operators. Libraries supporting binary strings and permutations are provided, others can be created by the user.

  Fitness modules are typically written by the user, although some sample applications are provided. The modules may contain a fitness function, initial values for various control parameters, and a function which graphically displays the best solutions.

  Splicer provides event-driven graphic user interface libraries for the Macintosh and the X11 window system (using the HP widget set); a menu- driven ASCII

interface is also available though not fully supported. The extensive documentation includes a reference manual and a user's manual; an architecture manual and the advanced programmer's manual are currently being written.

An electronic bulletin board (300/1200/2400 baud, 8N1) with information regarding Splicer can be reached at (713) 280-3896 or (713) 280-3892. Splicer is available free to NASA and its contractors for use on government projects by calling the STB Help Desk weekdays 9am-4pm CST at (713) 280-2233. Government contractors should have their contract monitor call the STB Help Desk; others may purchase Splicer at a nominal fee from:

COSMIC
382 E. Broad St.
Athens, GA 30602
U.S.A.
Phone: (404) 542-3265

- **The Evolution Machine**

  The "Evolution Machine" presents a collection of evolutionary algorithms (Genetic Algorithms and Evolution Strategies) in a common framework. It runs on PCs with MS-DOS and includes extensive menu techniques. A more detailed description of the "Evolution Machine" is given by the manual, available in PostScript form via anonymous ftp from ftp.wtza-berlin.de (141.16.244.4), file em-man.ps.Z.

  In this manual an introduction is given, the handling is fully described and the included algorithms are compared with regard to their performance. Interested parties can order the code of the "Evolution Machine" free of charge by request from one of the authors:

  Hans-Michael Voigt
  voigt@mike.fb10.tu-berlin.de
  Joachim Born
  born@max.fb10.tu-berlin.de

  Technische Universitaet Berlin
  Fachgebiet Bionik und Evolutionstechnik
  Forschungsgruppe Bio- und Neuroinformatik, Sekr. ACK1
  Ackerstrasse 71-76
  1000 Berlin 65
  Germany
  Phone: +49-30-314-72-677

- **OOGA, GENESIS**

  OOGA (Object-Oriented GA) is a genetic algorithm designed for industrial use. It includes examples accompanying the tutorial in the companion "Handbook of Genetic Algorithms". OOGA is designed such that each of the techniques employed by a GA is an object that may be modified, displayed or replaced in object-oriented fashion. OOGA is especially well-suited for individuals wishing to modify the basic GA techniques or tailor them to new domains.

The buyer of OOGA also receives GENESIS, a generational GA system written by John Grefenstette. As the first widely available GA program GENESIS has been very influential in stimulating the use of GAs, and several other GA packages are based on it. This release sports an improved user interface.

OOGA and GENESIS are available together on 3.5" or 5.25" disk for $60 ($52.50 inside North America) by order from:

T.S.P.
P.O. Box 991
Melrose, MA 02176
U.S.A.

- **GAGA**

  GAGA (GA for General Application) is a self-contained, re-entrant procedure which is suitable for the minimisation of many "difficult" cost functions. Originally written in Pascal by Ian Poole, it was rewritten in C by Jon Crowcroft. GAGA can be obtained by request from the author; given suffi- cient interest it will be made available via anonymous ftp.

- **ESCaPaDE**

  ESCaPaDE is a sophisticated software environment to run experiments with Evolutionary Algorithms, such as e.g. an Evolution Strategy. Future versions of the software will provide a well-defined interface to any kind of Evolutionary Algorithm, for instance Genetic Algorithms. The main support for experimental work is provided by two internal tables:

  – a table of objective functions and
  – a table of so-called data monitors,

  which allow easy implementation of functions for monitoring all types of information inside the Evolutionary Algorithm under experiment.

  ESCaPaDE 1.2 comes with the KORR implementation of the Evolution Strategy by H.-P. Schwefel which offers simple and correlated mutations. KORR is provided as a FORTRAN 77 subroutine, and its cross-compiled C version is used internally by ESCaPaDE.

  ESCaPaDE 1.2 will be available by e-mail request in order to track the spread of the software as this is its first public release. An extended version of the package was used for several investigations so far and has proven to be very reliable. The software and its documentation is fully copyrighted although it may be freely used for scientific work; it requires 5-6 MB of disk space.

  In order to obtain ESCaPaDE via mail request, please send a message to

  iwan@ls11.informatik.uni-dortmund.de

  The SUBJECT line should contain the request 'help' or 'get ESCaPaDE'. (If the subject line does not match a predefined set of mail requests the mail handler will NOT recognize your request!)

- **GAucsd**

GAucsd is a GENESIS-based GA package incorporating numerous bug fixes and user interface improvements. Major additions include a wrapper that simplifies the writing of evaluation functions, a facility to distribute experiments over networks of machines, and Dynamic Parameter Encoding, a technique that improves GA performance in continuous search spaces by adaptively refining the genomic representation of real-valued parameters.

GAucsd was written in C for Unix systems, but the central GA engine is easily ported to other platforms. The entire package can be ported to systems where implementations of the Unix utilities "make", "awk" and "sh" are available.

GAucsd can be obtained via anonymous ftp from cs.ucsd.edu (132.239.51.3), file pub/GAucsd/GAucsd12.sh.Z, or via mail server - send an empty message with the subject line containing "send GAucsd source" to nici@cs.ucsd.edu. Requests to be added to a mailing list for dissemination of GAucsd bug reports, patches and updates should be directed to the same address.

- **Genitor**

  Genitor is a modular GA package containing examples for floating-point, integer, and binary representations. Its features include many sequencing operators as well as subpopulation modelling.

- **mGA1.0, SGA-C, SGA-Cube**

  mGA1.0 is a Common Lisp implementation of a messy GA as described in TCGA report No. 90004. Messy GAs overcome the linkage problem of simple genetic algorithms by combining variable-length strings, gene expression, messy operators, and a nonhomogeneous phasing of evolutionary processing. Results on a number of difficult deceptive test functions have been en- couraging with the messy GA always finding global optima in a polynomial number of function evaluations.

  See TCGA reports 89003, 90005, 90006, and 91004 for more information on messy GAs; they can be obtained from the address below. Please note that 91004 is a dissertation and requires a pre-payment of $9.00 US ($12.00 US to ship overseas) to offset the cost of copying, binding and shipping.

  SGA-C is a C-language translation and extension of the original Pascal SGA code presented in Goldberg's book "Genetic Algorithms in Search, Optimization & Machine Learning" (Addison Wesley 1989). It has some additional features, but its operation is essentially the same as that of the Pascal version. SGA-C is described in TCGA report No. 91002, which is included in the distribution as a PostScript file.

  SGA-Cube is a C-language translation of Goldberg's SGA code with modifications to allow execution on the nCUBE 2 Hypercube Parallel Computer. When run on the nCUBE 2, SGA-Cube can take advantage of the hypercube architecture, and is scalable to any hypercube dimension. The hypercube implementation is modular, so that the algorithm for exploiting parallel processors can be easily modified.

  In addition to its parallel capabilities, SGA-Cube can be compiled on various serial computers via compile-time options. In fact, when com- piled on a serial computer, SGA-Cube is essentially identical to SGA-C. SGA-Cube has been nominally tested on a Sun 4/70 workstation, a VAX Ul- trix system, a Cray

X-MP/24 running UNICOS 5.1, and the nCUBE 2. It is described in TCGA report No. 91005, which is included in the distribution as a PostScript file.

Each of these programs is distributed in form of a Unix shar file, available via e-mail or on various formatted media by request from:

Robert Elliott Smith
Department of Engineering of Mechanics
The University of Alabama
Box 870278
Tuscaloosa, Alabama 35487

email: @ua1ix.ua.edu:rob@galab2.mh.ua.edu
phone: (205) 348-1618
fax: (205) 348-6419

SGA-C and SGA-Cube are also available in compressed tar form via anonymous ftp from the GA-List archive server ftp.aic.nrl.navy.mil (192.26.18.56) in the pub/galist/source-code/ga-source directory.

- **GAC, GAL**

  For those of you interested in obtaining some free GA software, I'm providing the packages I've been using for a few years. GAC is a GA written in C. GAL is my Common Lisp version. They are similar in spirit to John Grefenstette's Genesis, but they don't have all the nice bells and whistles. Both versions currently run on Sun workstations. If you have something else, you might need to do a little modification. [Alan Schultz informs me that GAL is easily ported to the Mac - although his version is no longer available.]

  In the spirit of "freeware", I am willing to e-mail either version (or both) to anyone who wants it. All I ask is that I be credited when it is appropriate. Also, I would appreciate hearing about improvements! This software is the property of the Department of the Navy.

  The code will be in a "shar" format that will be easy to install. This code is "as is", however. There is a README and some documentation in the code. There is NO user's guide, though (nor am I planning on writing one at this time). I am interested in hearing about bugs, but I may not get around to fixing them for a while. Also, I will be unable to answer many questions about the code, or about GAs in general. This is not due to a lack of interest, but due to a lack of free time!

  Bill Spears

- **Genetic, Genocop, Gafoc**

  These are four bare-bones GA programs (no documentation, no comments, no interfaces) written by Zbigniew Michalewicz for his own research. Genetic optimizes a transportation problem, either linear (Genetic-2L) or nonlinear (Genetic-2N). Genocop is a system to optimize any function with any set of linear constraints, while Gafoc solves discrete optimal control problems.

- **WOLF**

This is a simulator for the G/SPLINES (genetic spline models) algorithm which builds spline-based functional models of experimental data, using crossover and mutation to evolve a population towards a better fit. It is derived from Friedman's MARS models. The original work was presented at ICGA-4, and further results including additional basis function types such as B-splines have been presented at the NIPS-91 meeting.

Available at no cost via anonymous FTP by contacting the author; runs on SUN (and possibly any SYSV) UNIX box. Macintosh version available on floppy disk for a $20 fee. Both versions can be redistributed for noncommercial use. Simulator includes executable and C source code; a technical report (RIACS tech report 91.10) is also available.

David Rogers
drogers@riacs.edu
MS Ellis, NASA Ames Research Center
Moffett Field, CA 94035

- **Tierra**

  The Tierra software emulates a MIMD computer on which self-replicating, evolving pieces of machine code compete for CPU time and memory. The architecture has been designed such that randomly mutated or recombined code fragments remain functional often enough for selection to be able to improve the code over time. Tierra provides memory management and timesharing services for the evolving creatures, control over a variety of factors that affect the course of evolution, and a very elaborate observation system.

  Digital communities evolved with Tierra have been used to computationally study ecological and evolutionary processes such as competitive exclusion and coexistence, symbiosis, host/parasite density dependent population regulation, and the effect of parasites in enhancing community diversity.

  Tierra was written by Thomas Ray but includes significant contributions from Tom Uffner, Dan Pirone and Marc Cygnus. The software remains copy- righted ("all rights reserved"), and is not being placed in the public domain. However, it will be made available free of charge and may be freely distributed. The intent is that it not be used for profit making activities unless some royalty arrangement is entered into with the authors.

  The version of the software currently being distributed is considered to be a research grade implementation: it emphasizes modifiability and modularity over speed of execution, and is not free of bugs. The code is under rapid development, so if you use it, be prepared to pick up new versions from the ftp site frequently.

  The complete source code for the Tierra simulator is available via anonymous ftp from the /tierra directory at tierra.slhs.udel.edu (128.175.41.34) and life.slhs.udel.edu (128.175.41.33). The file "announce" there contains all the information necessary to get started. Although the source code on the ftp site will compile with Turbo C and run under DOS, a DOS version of the Tierra software is also for sale for $70. Make checks payable to Virtual Life, and mail your order to:

  Virtual Life
  P.O. Box 625
  Newark, DE 19715

- **Evolver**

  Evolver is a spreadsheet add-in which incorporates the first commercially available genetic algorithm to search for solutions. Evolver can be customized through the macro language, and is available for $345 on 3.5" or 5.25" floppies for the Excel, WingZ and Resolve spreadsheets on the Mac and PC computers. For further information, contact:

  Axcelis, Inc.
  4668 Eastern Avenue North
  Seattle, WA 98103-6932
  Phone: (206) 632-0885

  To order Evolver, contact:

  Spreadware Distributors
  P.O. Box 4552
  Palm Desert, CA 92261
  Phone: (619) 347-2365
  FAX: (619) 347-6045

- **GENEsYs**

  GENEsYs is a GENESIS-based GA implementation which includes extensions and new features for experimental purposes, such as selection schemes like linear ranking, Boltzmann, (mu, lambda)-selection, and general extinctive selection variants, crossover operators like n-point and uniform crossover as well as discrete and intermediate recombination. Self-adaptation of mutation rates is also possible.

  A set of objective functions is provided, including De Jong's functions, complicated continuous functions, a TSP-problem, binary functions, and a fractal function. There are also additional data-monitoring facilities such as recording average, variance and skew of object variables and mu- tation rates, or creating bitmap-dumps of the population.

  GENEsYs is expected to become available in June 1992.

- **GA Framework**

  This object-oriented framework for doing GAs includes classes of objects that are easily subclassable to add any special features. There is also a library of objects designed to integrate neural networks with GAs. Originally written in Object Pascal, GA Framework is currently being rewritten in C++; Steve Wilson is also looking for collaborators for later improvements.

  Not commercially available yet, but interested users should contact:

  Steve Wilson
  Emergent Behavior
  635 Wellsbury Way
  Palo Alto, CA 94306
  U.S.A.

- **XYpe**

  XYpe (The GA Engine) is a commercial GA application and development package for the Apple Macintosh. Its standard user interface allows you to design chromosomes, set attributes of the genetic engine and graphically display its progress. The development package provides a set of Think C libraries and include files for the design of new GA applications. XYpe supports adaptive operator weights and mixtures of alpha, binary, gray, ordering and real number codings.

  The price of $725 (in Massachusetts add 5% sales tax) plus $15 shipping and handling includes technical support and three documentation manuals. XYpe requires a Macintosh SE or newer with 2MB RAM running OS V6.0.4 or greater, and Think C if using the development package.

  Currently the GA engine is working; the user interface will be completed on demand. Interested parties should contact:

  Ed Swartz
  Virtual Image, Inc.
  75 Sandy Pond Road #11
  Ayer, MA 01432
  U.S.A.
  Phone: (508) 772-4225

- **GAME**

  GAME (GA Manipulation Environment) aims to demonstrate GA applications and build a suitable programming environment. Currently in the early development stage, the programming environment will comprise a graphic interface (using X-Windows), a library of parameterized algorithms and applications, a specialized high level language based on C++, and com- pilers to various workstations and parallel machines.

The Evolution Machine *

Manual

Version 2. 1

Hans–Michael Voigt
Joachim Born
Jens Treptow

Technical University Berlin
Ackerstrasse 71–76 (ACK1)
D–1000 Berlin 65
e-mail: voigt@fb10.tu–berlin.de

* This project is part of activities of the joint research group 'Evolution Strategies and Evolution Techniques' of the Technical University of Berlin, the Institute for Informatics and Computing Techniques of Berlin, and the Humboldt University at Berlin.

B.1 Introduction

Complex systems in nature are the result of evolutionary processes. Accepting this fundamental thesis, the idea is manifested that simulation of the main principles will supply a procedure for finding well–adapted or optimal versions in man–made systems too.
Basic ideas in simulation of evolution processes for optimization of complex computer models can be traced back to the early 70's. First algorithms with respect to optimizing computer models have been suggested by Rechenberg [20], Holland [14] and Schwefel [25].
Till today, the number of presented evolution algorithms – such as Evolution Strategies and Genetic Algorithms – is still growing. Due to their inherent parallelism, increased attention has been paid to evolution algorithms with the appearance of massively parallel computers.

In the Evolution Machine (*EM*), we have coded (in Turbo C, resp. Turbo C_{++} for PC's with MS–DOS) fundamental algorithms, and added some of our approaches to evolutionary search.

We intended to present a representative collection of evolutionary algorithms with the aims:

> Presentation of evolutionary algorithms, different in its practical convergence behavior,
>
> Extensive menu techniques with:
>
> > Default parameter setting for unexperienced users.
> >
> > Well-defined entries for *EM*– control by 'evolution freaks', who want to leave the standard process control.
> >
> > Data processing for repeated runs (with or without change of the strategy parameters).
> >
> > Graphical presentation of results: online presentation of the evolution progress, one–, two– and three–dimensional graphs to analyse the fitness function and the evolution process.
> >
> > Integration of MS–DOS utilities (Turbo C).

The *EM* is universally applicable to optimization problems which can be expressed in the following way:

$$\min \quad \{f(x) \quad | \quad x \in X\} \quad , \qquad X \subseteq R^n, f : R^n \longrightarrow R.$$

The feasible set X may be any subset, the objective function may be given by the values $f(x)$ (the fitness values) of parameter vectors x (the genes) only, i. e. no suppositions such as convexity, differentiability and unimodality are required.

The user can train the handling of the *EM* by means of the coded test functions. If he has a real problem, then he has only to code the problem with an interface to the *EM*, and the *EM* will try to solve the problem. Theoretically, the *EM* is able to find the solution. Allowing infinite time (generations), an evolution algorithm converges to a global optimum with probability one [4] !

B.2 The Handling

B.2.1 The Menus

General Procedure To start the *EM*, the initial program *EMM* has to be called. The *EM* will be displayed on the screen with a copyright cover at first. After pressing the key $< Esc >$, the first menu table is available.
Most of the menus are explained by help facilities. Entering the key $< F1 >$ within a menu table, a short help message is screened.

A menu can be entered by placing the cursor (with cursor keys) on a displayed menu line, or by pressing the parenthesized character key. In both cases, the enter action has to be closed by pressing the key $< Enter >$.
To pass on a submenu and come back to the higher menu, the user has to press the key $< Esc >$.
At the end of the hierarchy of menus, the user will be asked to specify numerical values (or characters). Mostly, there will be displayed a default value. The user can accept this proposal by pressing the key $< Esc >$ immediately. For entering another value, the user has to delete the default value by pressing the key $<\leftarrow>$ (resp. $< Backspace >$).
Then, he can type a new value and enter the value by pressing the key $< Enter >$.

A Guided Tour The first menu table makes the following menu entries available:

First Menu

E*dit Fitness Model*
I*nclude Fitness Model*
Start Evolution **M***achine*
Evolution **G***raph*
P*rint Evolution Process*
Print **F***inal Population*
T*ype Evolution Process*
S*etup*
Q*uit*

This menu table gives the possibilities to activate an editor. Selecting the menu

Edit Fitness Model

the table

T*urbo C*
W*ordStar*

will be additionally displayed in a window of the screen .
This menu is provided for defining or correcting the fitness function by the user, if he

wants it.
To prepare the *EM* with the fitness model and an evolution algorithm, the user has to activate the menu

Include Fitness Model

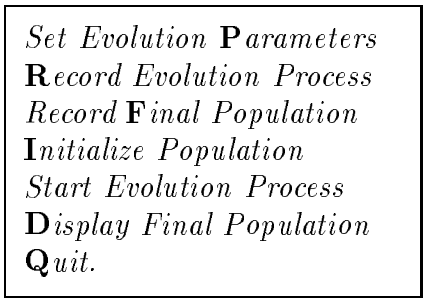
This menu initiates the display of the next menu table:

Evolution Strategy by Rechenberg
Evolution Strategy by Rechenberg,Schwefel
Evolution Strategy by Born
Genetic Algorithm by Goldberg
Genetic Algorithm by Voigt,Born
Turbo C Model
Other Model

This table holds the evolution algorithms (Evolution Strategies and Genetic Algorithms), and presents the menu for entering the fitness model (*Turbo C Model*; a next menu will be displayed the available models). Having entered an algorithm and a fitness model, the *EM* will automatically call a linker to produce an executable file. Using menu *Other Model*, an already prepared executable program can be entered.

After successful completion of this process, the menu

has to be selected. A new menu table will be displayed on the screen:



The menu *Set Evolution Parameters* allows to create a parameter file for the chosen algorithm, or to correct an existing one. After selecting this menu the table

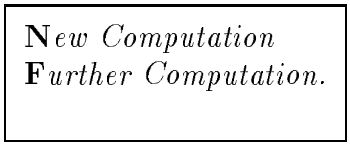
is additionally displayed .

Using the menu *Par File*, the user can load an existing parameter file, or he can save a new parameter file. New parameter files can be produced selecting the menu *Keyboard*. With the help of somefollowing menu tables, all parameters for the chosen algorithm can be set. Most of the parameters will be offered with default values.

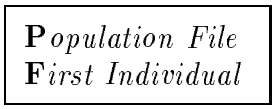
Now we will skip these menu tables. Every algorithm has its own strategy parameters. The different tables are explained in the next chapter.

With the help of the menus *Record Evolution Process* and *Record Final Population*, the user can define data files to save the data of the evolution process, and at the end of the run.

Always, the user has to enter the menu *Initialize Population*. The next submenu is



By typing the first menu line the menu



appears. Within the menu *Population File (Save Population)*, the user can set an option to store the starting population. Within the menu *First Individual*, the user can set a starting individual.

One individual is to determine by its components. By components are meant here the genes and the step sizes. The menu *Keyboard* allows to enter every component individually, and the menu *Random* allows to generate the initial individual randomly (in upper and lower bounds, uniformly distributed). In this case, the whole population will be generated randomly from the initial individual.
With the menu *Further Computation* one can choose a starting file for continuing a former computation. There are two entries:

Submenu *From Population File* can be choosen, if one wants to start from a previously stored file. Such save file exists if in a previous run the evolution process (with *Record Evolution Process*) or the final population (with *Record Final Population*) was stored.

For starting from the very last run, if the population size is to change, *From Recent*

Computation is provided. With the submenu

Random Parents
Random Mutated Parents
Best Parents

the start population can be generated in different manner. *Random Parents* initializes the start population newly, by random selection of parents of the old population. *Random Mutated Parents* selects parents randomly, and mutates the parents additionally. *Best Parents* selects parents by its fittness order. (In case of generation a larger population it is done repeatedly).

The menu *Start Evolution Process* initializes the run of the evolution algorithm. The progress will be displayed in a window. It is possible to regularly interrupt the process by pressing any key at any time. The process will be terminated before the generation of the next population.

The submenu to *Start Evolution Process*:

N*umerical Display*
*Graphical Display, Li**n***
*Graphical Display, Lo**g***
U*ser Graphic*
S*tart*

offers different online presentations of the evolution process: numerical, graphic in linear scale, graphic in logarithmic scale, and the possibility to use one's own presentation graph.

Display Final Population offers to show final results on the screen, and with *Quit* the first menu table is attainable.

The further menus in the first menu table:

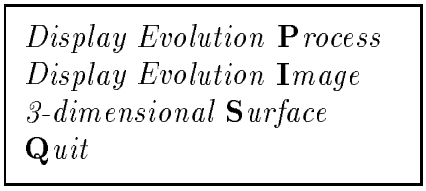
Evolution Graph, *Print Evolution Process*, *Print Final Population*, *Type Evolution Process* are assigned for presentation of results.

Selecting menu

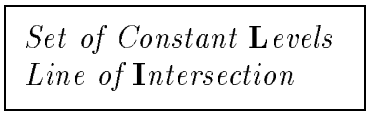
Evolution Graph

it is possible to choose a result file (with default extension .bin), and the first

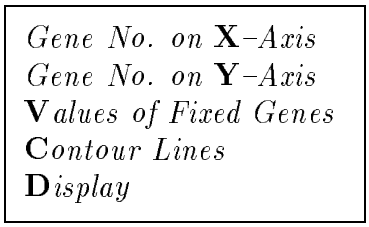
graphic menu appears:



By pressing the first point one immediately gets a graph. This shows the progress of the evolution process by displaying fitness values versus generations.
With the second menu point, one can have a two dimensional level set presentation (using an intersection parallel to principal axis) displayed, or a presentation along a line intersection which can be chosen in the whole feasible set:



For a two dimensional presentation, the menu *Set of Constant Levels* offers the submenu:



The default values are zero for the x–axis and one for the y–axis. In these menu points one can also choose lower and upper boundaries for the gene values:

Defaults for the boundaries are: The initial gene values for the computation as *Startpoint*, and Startpoint + one as *Endpoint*, respectively. All genes, not chosen as axis, are set to fixed fixed values. These genes can be locked up in the menu point *Values of Fixed Genes*. Defaults are also the gene values, set in the initial step of the evolution process.
In the menu point *Contour Lines*, one can select either exact or smooth (by means of Bezier functions) representation, and choose the number of constant levels to be drawn.

After pressing *Display* one can read some information of the file for control, nominally the name, recording date, no. of parents, genes, steps, and angles. Into the graph are projected individuals of the population, saved during the evolution process. Depending on the computer and the graphic card, one can see black and white or colored contour lines, the graph on the whole screen, or with a level scale including the maximum number of colors available on the right-hand side of the screen.

By entering the menu point *Line of Intersection*, one can see a one–dimensional intersection through the fitness function mountain. For the limits, the same rules hold as mentioned above.

The third submenu of the graphic menu table *3-dimensional Surface* generates a 3-D graph of the fitness model. Using the submenu

X–*Gene No.*
Y–*Gene No.*
H*oriz. Steps*
V*ert. Steps*
X–*Low*
X–*High*
Y–*Low*
Y–*High*
X–*Rotation*
Y–*Rotation*
S*how Surface*

one can generate the surface in different views. *Horiz. Steps* and *Vert. Steps* determine the resolution in x– and y– direction, *X–Rotation* and *Y–Rotation* determine the tilt. For instance: X = 0.0 and Y = 0.0 views a surface from above. X = 0.5 or Y = 0.5 views a surface, tilted over 45^o.

With *Quit* the first menu table is attainable.

By means of the menu in the first table

Print

one can send the save file of the recent evolution process to the printer. If you are not sure which file to print or how large a file is (economize paper!), you can look into the file by pressing *Type Evolution Process*. You have the possibility to interrupt at any time.

The menu

Setup

stands for the option to change the menu interface to adapt the *EM* to different hardware configurations, and to select some design options in menu–presentation.

B.2.2 Interface for User Tasks

To apply the *EM* to a user-defined model, the fitness model has to be code by the user observing some interface requirements.

The names of the routines and variables can be found in the listing of our Template Fitness Model (coded in the file template. c). In any case, the function

$$fitness\_value,$$

the include file

$$"function.h",$$

and the parameters

$$no\_genes, no\_constraints$$

have to be provided.

We have foreseen the option to use the *EM* with a non-standard, user-defined control. For instance, the user can code his own on-line data processing (graphic menu), or other actions. The user can come into action before the first call, after the last call of the function $fitness\_value$, and after each generation. All global control parameters are picked up in the include file

$$"strat\_al.h".$$

(See the listing in Appendix 1.)

B.3 The Algorithms

B.3.1 The Basics

The *EM* simulates natural evolution principles in order to obtain efficient optimization procedures for computer models. Simulation of nature, strong in detail and exactness, was not intended. Evolutionary methods – included in the *EM* – were chosen in order to provide algorithms with different numerical characteristics. Here, numerical characteristics are the speed of rapid (local) improvements (if the principle of strong causality is valid), and the chance of global convergence (in case of a multimodal objective function).

The *EM* operates on a set of a fixed number of individuals(respectively parameter vectors): the population. The individuals in the population will be improved, i. e. the individuals have to vary with the goal to minimize its fitness values (respectively, the parameter vectors have to vary with the goal to minimize the objective function) by means of the following principles:

> reproduction and variation (random choise of parents, generation of offspring using genetic operators such as mutation and crossover),
>
> selection (the best offspring form the next generation of the population).

This basic procedure is differently completed by the algorithms in the *EM* by:

Operating with a strategy (a deterministic rule or extension of the evolution principles to the strategy parameters themselves). Strategy parameters can be variances for (normally distributed) mutations, probabilities of mutations, crossover and the selection pressure.

Simulating more evolution principles than in the basic procedure (e. g. simulation of a genetic load).

Version 2. 1 of the *EM* contains the algorithms:

Evolution Strategy by Rechenberg,
Evolution Strategy by Rechenberg and Schwefel,
Evolution Strategy by Born,
Genetic Algorithm by Goldberg,
Genetic Algorithm by Voigt and Born.

In the following, a short description of the algorithm is given. Chapter 3. 2. contains more details of the implemented versions.

Test results with different strategy variants and different optimization problems are discussed in chapter 4.

B.3.2 The Evolution Strategy by Rechenberg

Rechenberg's [20] two membered evolution scheme (in the notation of Rechenberg the (1+1)–ES) is the simplest algorithm in the *EM* (but not in every case the most inefficient one). The population consists of two individuals, one parent and one offspring. The population size remains constant. The parent of a generation produces an offspring, only by means of point mutations to the individual genes randomly and independently of each other. Only the parent or the offspring can be the parent in a next generation, namely the one which represents the higher fitness. In this procedure, an individual in principle has an infinite life and capacity for producing offspring.
Mutations on the genes are generated using the normal distribution with individual variances on the gene locations. In this strategy, the variances are adapted in the evolution process by a deterministic rule, the so–called 1/5 success rule. Rechenberg discovered analytically for two unimodal fitness functions (sphere model and corridor model) that the most probable rate of convergence corresponds to a particular valuefor the probability of a success, i. e. an improvement in the fitness function value. He was thus led to formulate the following rule for controlling:
From time to time during the evolution process check the ratio of the number of successes to the total number of trials (mutations). If the ratio is greater than 1/5, increase the variance, if it is less than 1/5, decrease the variance.
Our experiences confirm that in many real problems this scheme proves to be extremely efficient in the rate of progress towards a (local) optimum.

B.3.3 The Evolution Strategy by Rechenberg and Schwefel

In this strategy, we have combined the Multimembered Evolution Strategy (see Rechenberg [20], Schwefel [25, 26] with two procedures to adapt the strategy parameters.
The Multimembered Evolution Strategy (in the notation of Rechenberg the $(\mu/\rho \stackrel{+}{,} \lambda)$–ES) realizes the genetic operators:

reproduction,

mutation,

recombination (crossover),

correlated mutations (search directions are not fixed to the coordinate axis, a flexible ellipsoid is used),

selection.

The population consists of μ individuals. An individual is defined by its genes and, additionally, by step sizes to mutate the genes. (Each gene position of an individual has its own step size). The population size remains constant.
The parents of a generation produce (reproduction) jointly λ offspring. Each parent has the same probability for reproduction: $1/\mu$. The reproduction is connected with recombination and mutations. Recombination (crossover) can be realized with pairs of parents ($\rho = 2$) or intermediate with all parents ($\rho = \mu$).
Mutations on an individual are realized differently for genes and step sizes.
Step sizes can be varied individually using a log–normal distribution with fixed variances, or a common mean step size which is varied by a fixed factor in each generation. (For details see chapter 3. 2. 3).
Genes are mutated normally distributed, using the step sizes as variances.
A special feature of this strategy is the possibility to rotate the mutation ellipsoid. In addition to the length of steps on the principal axes (step sizes), the positions of the axes in space are strategy parameters (part of the components of an individual) which are adaptable within the population.
For selection, it can be chosen that only the μ best of the λ offspring become parents of the next generation (the (. , .)– strategy), or that the parents are additionally included into the selection (the (. + .)–strategy).

B.3.4 The Evolution Strategy by Born

In this strategy, the major principles of the EGL (Evolution Strategy with Genetic Load) –method [4][6] are realized.The EGL–method uses the following genetic operators:

reproduction,

mutation,

inversion,

crossover,

genetic load,

selection.

Using Rechenberg's notation , we can describe the EGL- method as a $(\mu \dagger \delta/2 + 1)$-ES, wereby $\dagger\delta$ denotes the number of elements in the genetic load.

In the *EM*, the following principles of the EGL–method are missing at present:

operating with strategy parameters for probabilities of mutations, inversion and crossover,

'breeding' procedure: successful variations are reiterated (by means of an approximative line search).

Instead of a strategy population, as in the EGL– method, we have introduced a learning parameter 'learning rate mutations'. This parameter controls the number of simultaneously mutated genes by means of a simple success rule.

The attention should be focussed on the principle genetic load. The motivation to introduce a genetic load, i. e. the population contains individuals which are selectively neutral, is founded by a biological phenomenon [12] and extensive simulation.
Numerical tests with the $(\mu, +\lambda)$–ES of Rechenberg and Schwefel demonstrate that there is a certain chance for global convergence in the case of multimodal problems. This chance depends on the initial population and the number of parents. During the evolution process, it can be observed that the variability in the population becomes quickly small. After some generations, the individuals are nearly identical and there is practically no chance to overcome a local minimum. On the other side, this scheme approximates a minimum very fast.
These numerical properties are determined by the use of log- normally distributed variances and the selection procedure. The combination of both – a relative high variability in the variances, and a strongly success-oriented selection procedure – entails the tendency to rapidly reduce of the step sizes.
Operation with a genetic load obeys a certain variability over all generations. The variability can be controlled by the ratio of the number of selectively active parents to the number of all parents, and by mechanisms to generate individuals for the genetic load.
At present, we have coded two basic procedures. Firstly, it can be chosen a fixed genetic load. All individuals in the genetic load are initialized at the beginning, and are never subjected to the selection. Secondly, the genetic load is generated at the beginning, and will be newly generated after some generations, regardless of the fitness values.
Other genetic operators are used in the same manner as in the Evolution Strategy of Rechenberg and Schwefel. Crossover is done between random chosen segments of the parents. As inversion we used the random choice of a segment of an individual (more exactly, a segment of the gene components and a segment of the step size components) in which the sequence of components are changed inversely (regardless to a possible signification of positions).

B.3.5 The Genetic Algorithm by Goldberg

Genetic Algorithms are different from Evolution Strategies in one fundamental way. They require the 'natural' (e. g. continuous) parameter set of the optimization problem to be coded as afinite–length string over some finite alphabet [10]. In our point of view, there is no other principal difference between Evolution Strategies and Genetic Algorithms. In the *EM*, we have implemented a basic Genetic Algorithm – the SGA (Simple Genetic Algorithm) with a multiparameter, mapped, fixed– point coding.
This algorithm is composed of three genetic operators:

reproduction,

crossover,

mutation.

In Rechenberg's notation we can describe the SGA with a $(\mu/2, \mu)$–scheme, i. e. the population consists of μ parents, the parents produce μ offspring, recombination (crossover) is done with pairs of parents. An individual consists of coded genes. Coded genes means a coding of the continuous parameters (genes) as fixed–point variables concatenated to a bit string – the chromosome. (In the SGA as 'bit string' is used a boolean array. The length of this array determines the precision of the parameters).
The operator reproduction includes a selection procedure. The algorithm chooses (selects) pairs of parents to produce two offspring in the reproduction step randomly, but not equally distributed over the population as it is done in Evolution Strategies. In the random selection, parents have different probabilities for reproduction, depending on the fitness values (i. e. a parent with a better fitness has a greater probability to select).
With the operator crossover, the two selected parents produce two offspring, with a certain probability and using a randomly determined segment for crossover. (A crossover segment starts in the first bit of the individual and ends in a random determined position (equally distributed over all positions)). Mutation of the two offspring is carried out 'bit by bit',i. e. each bit changes its value with a certain probability, regardless of its position. The mutation probability is the same evermore (and should be chosen very small).

B.3.6 The Genetic Algorithm by Voigt and Born

With this algorithm, we intend the objective to combine the principal operators of an Evolution Strategy with basic mechanisms of creating mutations of Genetic Algorithms, namely the transformation to a discrete (0,1) problem.
Differently to the approach of well–known Genetic Algorithms, e. g. the Genetic Algorithm by Goldberg, we do not use a mapping of the continuous variables as fixed–point variables to bit strings resp. inversely, but we operate immediately with the bits in the floating point format of the variables. This approach avoids coding problems for transformation, but entails difficulties in programming of the bit-operations.
Our Genetic Algorithm can be described with the following genetic operators:

reproduction,

recombination (within genes on bit level),

inversion (within genes on bit level and on individual level),

mutation (within genes on bit level),

genetic load,

selection.

For all operators which are not on bit level, we use the procedures as in the Evolution Strategy of Born.
One individual is defined by its genes and additionally by step sizes. Here, we define as step sizes the probabilities of mutations, inversion, recombination within genes on bit level and rate of mutations on individual level (i. e. probability that within a gene a variation can take place).
These probabilities are not constant for every bit position. They are reduced internally with higher positions (especially bits in the exponent format). For details see chapt. 3. 2. 3).

B.3.7 Specialities of the Algorithms

We will present for every algorithm all control parameters in a box analogously to submenus of *Keyboard*:

The Evolution Strategy by Rechenberg

Parameter	*Default*	Usage
MUTATION		
Absolute Lower Bound	*1.00000E–16*	Lower bound to vary the step sizes, absolute
Individual Variance	*1.00000E–16*	Lower bound to vary the step sizes relative to values of genes
Step Size Factor	*0.85*	Factor for step size adjustment
Factor Correction Mutations	*1*	Quantity used in step size management
LIMITS		
Time Limit (sec)	*1000*	
Generation Limit	*5000*	
CONVERGENCE		
Check After Generations	*100*	
Abs. Fitness Difference	*1.00000E–30*	
Rel. Fitness Difference	*1.00000E–20.*	

The strategy parameters 'steps' will be used as variances for normally distributed mutations on genes. The variances themselves will be controlled by the 1/5 success rule. In realizing this, the parameters *Factor Correction Mutations lr* (default value 1) and *Step Size Factor sn* (default value 0. 85) are used. The variances are adjusted so that on average one success is obtained in $5 * lr$ iterations (fitness function calls). This is computed on the last $10 * no\_genes * lr$ iterations. The success rate indicated by lr is used to adjust the variances by the common factor sn (if the ratio is less than 1/5) or $1.\ 0/sn$ (if the ratio is greater than 1/5). The variances can be kept constant during the iterations by setting $sn = 1.\ 0$.

The Evolution Strategy by Rechenberg and Schwefel

Parameter	*Default*	Usage
POPULATION		
No. Parents	*10*	Number of parents
No. Offspring	*100*	Number of offspring
MUTATION		
Mutation Type	*2*	1: Individual changes 2: Mean changes
Global Variance	*7.07107E–01*	Upper bound to vary the step sizes
Absolute Lower Bound	*1.00000E–06*	Lower bound to vary the step sizes
Individual Variance	*8.40896E–01*	Individual variance to vary the step sizes
Mutation Factor	*1.5*	Factor to vary the step sizes
Correlated Mutations	*YES*	YES: Ellipsoid can extend and rotate NO: Ellipsoid cannot rotate
Correlation Factor	*8.72666E–02*	
No. Correlation Angles	*1*	
RECOMBINATION		
Recombination Type	*4*	1: No recombination 2: Discrete recombination of pairs of parents 3: Intermediate recombination of pairs of parents 4: Discrete recombination of all parents 5: Intermediate recombination of all parents in pairs
SELECTION		
Selection Type	*1*	1: Selection only to offspring 2: Selection to offspring and Parents
LIMITS		
Time Limit (sec)	*1000*	
Generation Limit	*5000*	
CONVERGENCE		
Check After Generations	*5*	
Abs. Fitness Difference	*1.00000E–30*	
Rel. Fitness Difference	*1.00000E–20.*	

The strategy parameters 'steps' will be used as variances for normally distributed mutations on genes.
Two possibilities are implemented to mutate step sizes:
– Individual mutations ('mutation type 1') – a log–normal distribution with fixed variances is used. Two parameters are used: the common parameter $\triangle_{step}$ ('global variance') and an individual variance $\triangle_{step_i}$ ('individual variance').

A current step size will be generated by

$$\exp(\quad \mathcal{N}(\triangle_{step}) + \mathcal{N}(\triangle_{step_i}) \quad) \quad .$$

As default values we have coded

$$\triangle_{step} = \frac{c}{\sqrt{no\_genes}}, \quad c = 1 \qquad and \qquad \triangle_{step_i} = \sqrt{\frac{\sqrt{no\_steps}}{no\_genes}} \quad .$$

These rules are suggested to provide rapid convergence for sphere models and a (10,100) – strategy (comp. [26]).

– Mean mutations ('mutation type 2') – a (common) current step size will be generated by

$$\tilde{q} = \begin{cases} mutation\_factor & ; \text{ with probability } \frac{1}{2} \\ \frac{1}{mutation\_factor} & ; \text{ else} \end{cases}$$

$$step\_size\_current = \tilde{q} * \left(\frac{\sum_{j=1}^{no\_parents} step\_size\_old_j}{no\_parents} \right)$$

The 'Mutation Factor' ($mutation\_factor$) is fixed with the default value 1. 5.

The individual $current\_step\_size$ is a slightly disturbed $current\_step\_size$:

$$step\_size\_current_i = step\_size\_current * \mathcal{N}(\triangle_{step}) \quad , \qquad i = 1, \ldots, no\_steps$$

According to the Evolution Strategy KORR [26], we have implemented the possibility to rotate the mutation ellipsoid. Using the option *'Correlated Mutations' = YES*, the axis will be varied. In addition to the length of the principal axes (step sizes) the positions of the axes in spaces (with dimension no_steps) are strategy parameters which are adjustable within the population. A linear correlation of random change through prescribed angles is realized. These angles represent the additional strategy variables, which can take all values between $-\pi \ldots + \pi$. In the maximum, there are a total of

$no\_angles = \frac{no\_steps*(no\_steps-1)}{2}$.

The angles will be generated by an additive mutation process using the normal distribution with the fixed variances $\triangle_{angles}$ ('Correlation Factor') which is the same for all angles. $\triangle_{angles} = 5 * 0.01745$ is used (5 degrees) as default value in accordance to the use in KORR.

The recombination will be applied to all parameters, including steps and angles (again in accordance to KORR). The recombination types 1 and 2 may be clear immediately. Using type 3, the arithmetical mean of the recombined components is applied. In this type, a check must be made on the difference between the parental angles to establish suitable mean values. Type 4 includes all parents (not only two) into the recombination. Type 5 is the combination of type 3 and 4, i. e. : all parents are included, and the arithmetical mean is built by means of a special version for angles.

Parameter	*Default*	Usage
POPULATION		
No. Parents	*10*	Number of parents (without genetic load)
No. Indiv. Genetic Load	*5*	Number of individuals in the genetic load
INVERSION		
Probability Inversion	*0.1*	Probability of inversion
MUTATION		
Global Variance	*7.07107E–01*	Upper bound to vary the step sizes
Absolute Lower Bound	*1.00000E–06*	Lower bound to vary the step sizes
Individual Variance	*8.40896E–01*	Individual variance to vary the step sizes
Learning Rate Mutations	*YES*	Prob. of simultaneously mutated genes
LIMITS		
Time Limit (sec)	*1000*	
Generation Limit	*5000*	
CONVERGENCE		
Check After Generations	*1000*	
Abs. Fitness Difference	*1.00000E–30*	
Rel. Fitness Difference	*1.00000E–20*	
GENETIC LOAD		
Fixation Type Genes	*Random*	Fixed: No changes over all generations
Fixation Type Steps	*Fixed*	Random: Repeated, random variation after some generations

Actually, we realize the random variation of the genetic load ('Fixation Type: Random') in the following manner: The individuals are newly generated after *no_genload* (number of individuals in the genetic load) generations. An individual is newly generated by componentwise addition of random increments. The random increments are equally distributed in case of genes, and log-equally distributed in case of steps, within bounds. The bounds are determined from the initial genetic load, using the minimum and the maximum of the components of the individuals.

Parameter	*Default*	Usage
STRATEGY PARAMETERS		
Population Size	*30*	No. of parents, no. offspring
Chromosome Length	*320*	No. of bits in an individual
CODE PARAMETERS		
length_parm_all	*32*	Part of the (full) chromosome length, the same for every gene
max_parm_all	*1.0E+15*	Upper bound in coding of a gene (the same for all)
min_parm_all	*–1.0E+15*	Lower bound in coding of a gene (the same for all)
Crossover probability	*0.6*	
Mutation probability	*0.0333*	
LIMITS		
Time Limit (sec)	*1000*	
Generation Limit	*5000*	
CONVERGENCE		
Check After Generations	*1000*	
Abs. Fitness Difference	*1.00000E–30*	
Rel. Fitness Difference	*1.00000E–20.*	

According to the original SGA, we have implemented the use of standard binary coded individuals. Selection can be done by proportional selection [14], or using a ranking scheme [2]. The ranking scheme combined with elitist strategy (the best individual of a generation is guaranteed to survive to the next generation) is the standard variant. Actually, as selection procedure, we have implemented a ranking scheme where the section probabilities are correlated linearly with decreasing order of fitness values.

Parameter	*Default*	Usage
POPULATION		
No. Parents	*10*	Number of parents (without genetic load)
No. Indiv. Genetic Load	*5*	Number of individuals in the genetic load
INVERSION		
Probability Inversion	*0.1*	Probability of inversion (on individual level)
MUTATION		
Global Variance	*7.07107E–01*	Upper bound to vary the step sizes
Absolute Lower Bound	*1.00000E–06*	Lower bound to vary the step sizes
Individual Variance	*8.40896E–01*	Individual variance to vary the step sizes
LIMITS		
Time Limit (sec)	*1000*	
Generation Limit	*5000*	
CONVERGENCE		
Check After Generations	*1000*	
Abs. Fitness Difference	*1.00000E–30*	
Rel. Fitness Difference	*1.00000E–20.*	

The strategy parameter vector 'steps' is used as field of probabilities of mutations, recombinations, inversions within genes, and rate of mutations on individual level (i. e. probability that within a gene a variation can take place) – in this order.

B.4 Performance of the Algorithms

B.4.1 Test Problems

Performance evaluation of random search algorithms needs a representative suite of test functions, and at least one performance measure. We will compare the evolution algorithms by means of the measure 'average number of function evaluations to approximate the (global) optimum'. As test suite, we choose the Functions F1 and F6 - F8, often used in the random search community (see e.g.: [18]).

Function F1 is a unimodal function. Nevertheless, it is a standard problem by De Jong [15], and was used by Rechenberg [20] to investigate his Evolution Strategies. F6 - F8 are highly multimodal. F6 and F8 are contained in Toern and Zilinskas [28], F7 was proposed by Schwefel [25].

$$\text{(F1)} \qquad \min\{\textstyle\sum_{n=1}^{10} x_i^2 \quad | \quad -500.0 \leq x_i \leq 500.0\} \quad .$$

$$\text{(F6)} \qquad \min\{100 + \textstyle\sum_{n=1}^{20} x_i^2 - 10\cos(2\pi x_i) \quad | \quad -5.12 \le x_i \le 5.12\} \quad .$$

$$\text{(F7)} \qquad \min\{\textstyle\sum_{n=1}^{10} -x_i \sin(\sqrt{|x_i|}), \qquad | \quad -500.0 \le x_i \le 500.0\} \quad .$$

$$\text{(F8)} \qquad \min\{\textstyle\sum_{n=1}^{10} x_i^2/4000 - \prod_{n=1}^{10} \cos(x_i/\sqrt{i}) + 1 \quad | \quad -600 \le x_i \le 600\} \quad .$$

The test functions have a different complexity. In broad terms, the complexity of a continuous optimization problem depends on

the number of local minima
the distribution of the local minima
the domains of attraction for local minima

It is beyond the scope of this paper to discuss the open question: "How is to generate a representative suite of test problems in global optimization". We will only remark that all multimodal problems in our test suite are 'semi-causal' (Rechenberg [21]). Semi-causal means that the local minima fulfil an order relation. There is no 'neadle in the haystack' problem with a chaotic placement of the minima.

In the following, we use the notions 'minima-attractor' and 'monotonous minima-attractor'. Let be $x^i, \quad i = 1, \ldots$, local minima of a multimodal function over the n-dimensional Euclidian space.

A minima-attractor x^a is defined as follows:

In a neighbourhood of x^a,

$$f(x^a) \le f(x^i), \quad i = 1, \ldots$$

holds for all local minima x^i.

A minima-attractor x^a is monotonous if:

$$If \quad \|x^a - x^i\| \le \|x^a - x^j\| \quad then \quad f(x^i) \le f(x^j), \quad i, j = 1, \ldots$$

.

We will discuss the complexity of F6 - 8 in detail (Function F1 is not complex). Next we will present 3D pictures of the landscapes created by the test functions (fig. 1 - 4), to give an impression of their high multimodality.

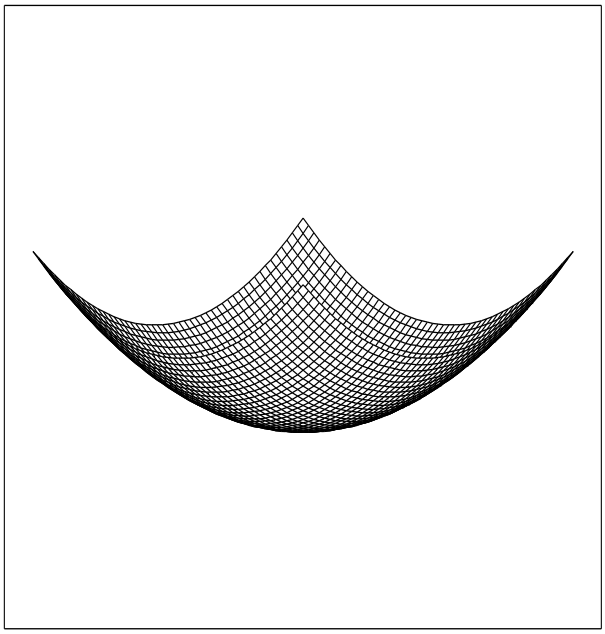


Figure 1: Fitness Function F1

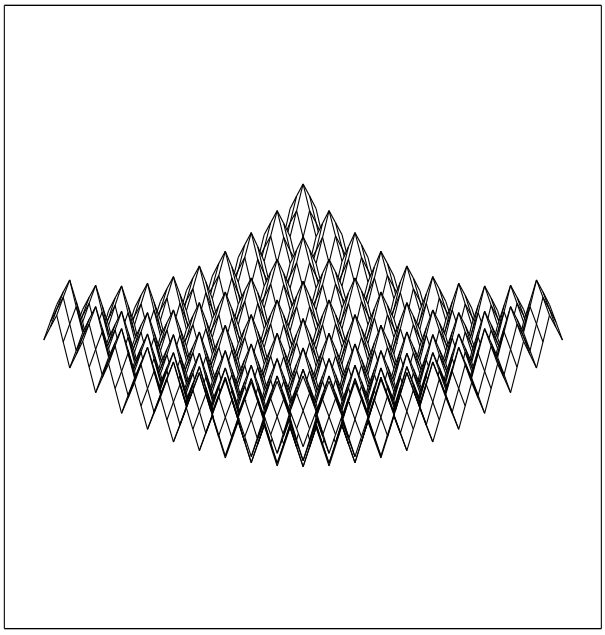


Figure 2: Fitness Function F6

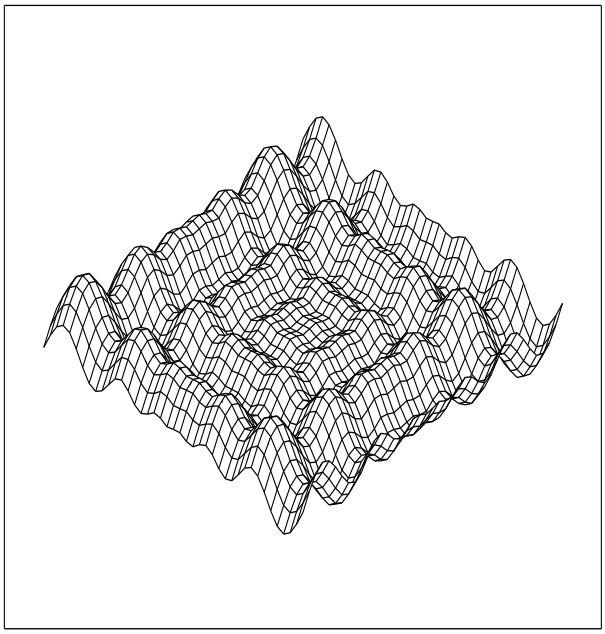


Figure 3: Fitness Function F7

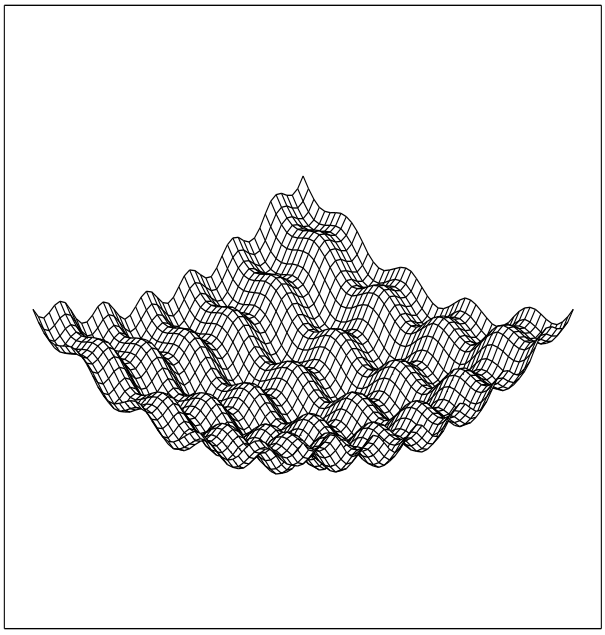


Figure 4: Fitness Function F8

All functions, exept F8, are separable and symmetric problems. For a test function $f(x)$ holds:

$$minf(x) \quad = \quad \sum_{i=1}^{n} minf_i(x_i) \quad = nf_0(x_0) \quad , f_i = f_0, x_i = x_0, \quad i = 1, ..., n.$$

For all problems, the minima are located on a rectangular grid.
For F6, the local minima are located on the rectangular grid with size 1. The global minimum is at $x_i^g = 0, i = 1, ..., n$, yielding $f(x^g) = 0$. All the grid points with $x_i = 0$ in all except one coordinate, where $x_i = 1$, yield $f = 1$, the second best minimum. This function has the only one minima attractor x^g, and it is monotonous.

Function F7 has its global minimum at $x_i^g = 420.9687, i = 1, ..., n$. The local minima are located at a rectangular grid with increasing size $\approx (\pi(0.5 + k))^2, k = 0, 2, 4, ...$ for positive directions of the coordinate axis and $\approx (\pi(0.5 + k))^2, k = 1, 3, 5, ...$ for negative directions. There are more than 60 000 local minima within the feasible set [4]. Points with $x_i = 420.9687 \quad , i = 1, ..., n, i \neq j, \quad x_j = -302.5232$, represent the second best minimum - but are located far away from the global minimum. Therefore, the search may be trapped in the wrong region. This function has many minima attractors. All of them are monotonous. (See figure 3.)

Function F8 is the most difficult problem in the test suite. It has been suggested by Griewangk [11]. The function has its global minimum $f = 0$ at $x_i^g = 0, i = 1, ..., n$. The local minima are located on a rectangular grid with increasing size $\approx k\pi\sqrt{i}, \quad k = 1, 2, 3, ...$ The point x^g is the only one minima attractor, and it is a monotonous one. There are four points with the suboptimal minimum $f(x) \approx 0.0074$ at $x \approx (\pm\pi, \pm\pi\sqrt{2}, 0, ..., 0)$. But F8 is an unseparable and unsymmetric function to the principal axis. (See figure 3.)

The highly multimodal functions F7 - F8 are semi-strong causal. The local minima are located on rectangular grids (see the following figures 5 - 7).

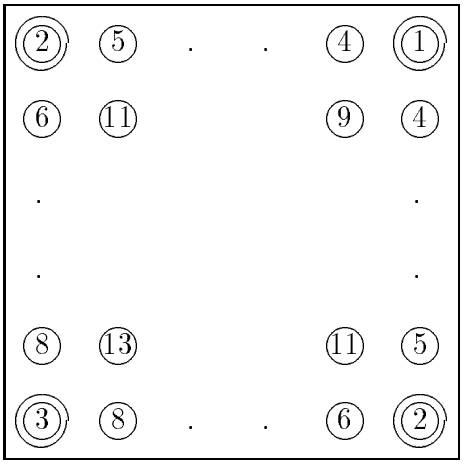


Figure 5: Distribution of minima for F7

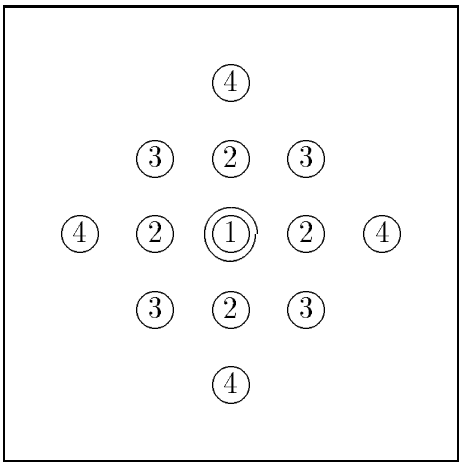


Figure 6: Distribution of minima for F6

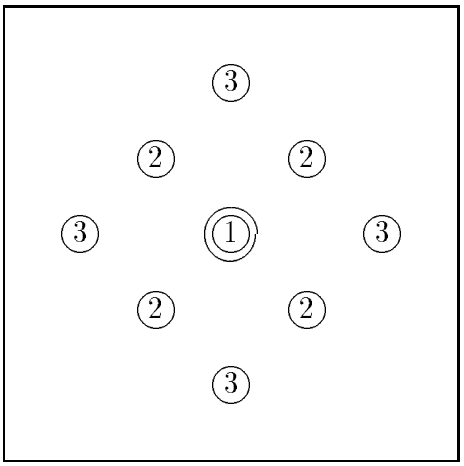


Figure 7: Distribution of minima for F8

B.4.2 Results

A difficult part in examining evolution algorithms is to explain why and when they will work. We will compare the evolution algorithms, included in the *EM* and explain the more tricky parts of the algorithms.

This test series was constructed to simulate the approach of a user without sophisticated knowledge in evolution algorithms. This point of view is reflected in following recommendations, which we have tried to realize:

> Use the algorithms with their default values of strategy parameters.
> Generate genes of the initial population randomly, equally distributed in the feasible set.
> Generate steps for the initial population so that in principle the evolution process starts with a covering of the feasible set.
> Initialize genetic loads in the same manner as it is proposed for the initial population.

The following results are based on 50 runs for every version. The runs were terminated if the region of attraction of the global optimum was reached, or were stopped after 25.000 function evaluations. We present the number of function evaluations needed to find the global minimum (its region of attraction), or the best fitness value after 25.000 function evaluations, if all runs had been terminated in local optima.

The regions of attraction of the global minimum are found if the fitness values are lower then the following limits:

Function	fitness value
F1	1.0e-03
F6	9.0e-01
F7	-4.18e+03
F8	1.0e-03

In the following tables we use the notations:

> Algorithm - the evolution algorithm following the Rechenberg notation; the control parameters are presented completely in appendix 2.
>
> Failure - the number of runs without approximating of the minimum within the required precision after 25.000 function evaluations.
>
> Best, Worst and Average - the number of function evaluations resp. the fitness value after 25.000 function evaluations.

Algorithm	Failure	Best	Worst	Average
RB-(1+1)	0	708	814	763
RS-(10,100)	0	4300	9300	4900
BO-(15†5+1)	0	9462	23728	17429
GO-(50,50)e	50	1.87e+00	3.01e+00	2.06e+00
VB-(15†5+1)	0	3908	7392	5484

Table1: Results for F1

Algorithm	Failure	Best	Worst	Average
RB-(1+1)	50	5.37e+01	7.86e+01	7.16e+01
RS-(10,100)	50	7.95e+00	5.96e+01	2.35e+01
BO-(15†5+1)	50	1.46e+00	6.28e+01	3.77e+01
GO-(50,50)e	50	1.52e+02	1.67e+02	1.58e+02
VB-(15†5+1)	0	1059	18940	6766

Table2: Results for F6

Algorithm	Failure	Best	Worst	Average
RB-(1+1)	50	-2.05e+03	-3.12e+03	-2.31e+03
RS-(10,100)	49	20000	20000	20000
BO-(15†5+1)	46	2019	23217	15816
GO-(50,50)e	47	23950	24850	24550
VB-(15†5+1)	3	2382	20846	10148

Table3: Results for F7

Algorithm	Failure	Best	Worst	Average
RB-(1+1)	50	5.41e-02	2.53e-01	7.31e-02
RS-(10,100)	50	9.86e-03	1.05e-01	4.94e-02
BO-(15†5+1)	50	6.24e-03	4.59e+00	5.95e-01
GO-(50,50)e	50	1.42e-01	3.60e-01	2.15e-01
VB-(15†5+1)	50	2.39e-01	8.52e-01	4.81e-01
BO-(30†20,200)l	0	13000	19800	13800

Table4: Results for F8

B.4.3 Comparison of the Algorithms

Evolutionary algorithms are random search procedures having an 'intelligent' set of operators. Rechenberg formulated that the operator 'mutation' is the 'power' of the evolution. This is right in case of Evolution Strategies.

Evolution Strategies

The most dominating factor for numerical properties as global convergence and efficiency, is the suited choice of the step sizes. The step sizes control the normally distributed mutations. They have to be controlled adaptively. In the evolution process, the step sizes have to converge to zero.

A strong local oriented step size control, as it is happened with the 1/5 - success rule for the (1+1)–ES with its inner spheric model, entails rapidly decreasing step sizes. It creates the best results in case of F1 with RB–(1+1), and leads to the worst results in case of the multimodal functions F6-8. Depending from the start values, a next local minimum will be approximated rapidly.

The multi-membered (μ, λ)–ES, with its natural step size control by applying the evolution operators to the strategy parameters themselves, owns a greater variability in step size generation. In tendency, the step sizes converge to zero too, but the rate of decrease can be controlled by the size of the population, preferably by the number of offspring λ.

The standard version, RS–(10,100), shows sligtly better results than RB–(1+1) in case of the multimodal test suite F6-8. But only one run (F7) terminated in the global minimum. Observation of the numerical values showed: The individuals in the population became nearly identical after just 10 generations. The step sizes were reduced quickly, and became too small for jumping out of a local minimum, in the latest after 40 generations.

The version BO–(15†5+1) of the EGL-method does not demonstrate significantly better results in our test series than the strategy RS–(10,100). In case of the unimodal problem F1, it is demonstrated that the genetic load is really a load. The large number of function evaluations is produced by the high rate of big jumps. This rate is totally unfit in this case.

Both, the version RS–(10,100), as well the version BO–(15†5+1) have too small complexities in comparison with the complexity of the multimodal test problems F6-8. Furthermore, tests with increasing size of μ, λ and δ show significantly better results in case of the multimodal test problems F6-8. The best results, we found with a 'mixed' version BO–(μ†δ $\overset{+}{,}$ λ)l. We find in testing problem F8 the following orders of performance (the operator $<$ means 'is better than'):

$$BO-(30\dagger 20,200)l \quad < \quad BO-(30\dagger 0,200) \quad < \quad BO-(15\dagger 5+1)$$

In table 4, the version BO–(30†20 , 200)l demonstrates its power. It outperforms all other algorithms.

We have regarded the operator 'mutation' and the factor 'size of population' now.

In the Evolution Strategy by Born, the operator 'inversion' is available. Fairly, we have ommited the use of this operator in our tests. Applying this operator, an extremely high speed of global convergence would be achieved for our test suite. For all problems, it holds that the global minimal point has identical components. In this case, the 'inversion' in combination with 'crossover' duplicates components of the global minimum point within an individual very quickly.

This is an extrem case of speed-up effects with evolutionary operators.

Figures 5-6 illustrate that the operator 'crossover', in case of separable problems with monotonous minima attractors, combines individuals, located in local minima, to individuals in better local minima. This is what the Genetic Algorithm community denotes as 'building block feature' (see [10]).

Genetic Algorithms

In our tests the 'classic' Genetic Algorithm in version GO–(50,50)e demonstrates a global search behavior which is able to compete with the results from Evolution

Strategies version RS–(10,100) and BO–(15†5 + 1). The use of binary coded individuals and a 'mutation' operator, which is used independently from the bit–position, guarantees a high degree of blind search in the whole feasible set. The principle of strong causality is invalid for this algorithm. This could have positive effects in case of multimodal functions. But the test results show that there is no effect. The reason is, that the 'crossover' does not work 'intelligently'. 'Crossover' is done without regard of bit–blocks in the chromosome determining a gene (variable), and by this the 'building block feature' of F6-7 is not exploited fully. This is also the cause for the very bad results in case of F1. The efficiency is too low than that the minimum could approximated within the required precision.

The Genetic Algorithm by Voigt and Born, version VB–(15†5 + 1), demonstrates the best results in case of F6-7. Nearly all runs approximated the global minima. The rapid progress is caused by different variation of bits in the exponential part and the part for the mantissa using the genetic operators on bit level. Variations in the exponential part have a logarithmic scale, and effect big jumps within the evolution process. Variations are controlled by a small probability for mutations in the exponential field, reduced internally with higher positions. This leads to a high chance of global convergence, without loss of efficiency - in case of F6-7. Problem F8 could not be solved by any run with version VB–(15†5 + 1). This demonstrates the higher complexity of this problem, as outlined in chapter 4.1.

Now, the question may appear: "What is the best evolutionary algorithm ?". From the test results and the remarks above should be evident that there is no easy answer to this question. Every real-world problem should be solved by experiments using the full power of the *EM* !

Appendix 1: Listing of the Template Fitness Model

```
/********************** begin template.c  ******************************
***                                                                  ***
***                   Template Fitness Model                         ***
***                                                                  ***
**********************************************************************/

/**********************************************************************
***                                                                  ***
***     The following include files has to be provided in any case:  ***
***                                                                  ***
***     "function.h" defines the structure of an individual          ***
***                                                                  ***
***     <stdlib.h>, <math.h> are necessary for use of the mathematics ***
***                  functions of Turbo C                            ***
***                                                                  ***
***                                                                  ***
***     The other include file is optional:                          ***
***                                                                  ***
***     "strat_al.h" pickes up  common control parameters            ***
***                  of evolution algorithms included in the EM      ***
***                  The user has the possibility to code his own    ***
***                  control and actions by means of these parameters ***
***                  and the interface routines                      ***
***                                                                  ***
**********************************************************************/

#include "function.h"
#include <stdlib.h>
#include <math.h>
#include "strat_al.h"



/**********************************************************************
***                                                                  ***
***             The optional interface routines                      ***
***                                                                  ***
**********************************************************************/

void first_call_interface (void);
void last_call_interface  (void);
void generation_interface (void);

/**********************************************************************
***                                                                  ***
***     The following parameters has to be provided in any case       ***
***                                                                  ***
***     no_genes         this is the number of independent variables  ***
***     no_constraints   this is the number of constraints           ***
***                      (if an unconstraint problem has to be solved ***
***                      you have to set no_constraints = 0)         ***
***                                                                  ***
**********************************************************************/

int no_genes       = ... ;
int no_constraints = ... ;
```

```
/*************************************************************************
***                                                                    ***
***                The fitness function model                          ***
***                                                                    ***
***     The function fitness_value computes for a set of               ***
***     independent variables  x[i], i=0,...,no_genes-1,               ***
***     a correponding value y which has to be minimized               ***
***                                                                    ***
***     This function has to be provided by the user                   ***
***                                                                    ***
*************************************************************************/

float fitness_value (struct individual *current)
{
  float   y;
  float  *x;

  x = current->gene;

  y = ..... ;

  return(y);
}
/*************************************************************************
***                                                                    ***
***       The constraints for the fitness function model               ***
***                                                                    ***
***     The function constraint_value computes for a set of            ***
***     independent variables x[i], i=0,...,no_genes-1, the            ***
***     values of the different  constraint functions which            ***
***     has to be given in the form                                    ***
***                                                                    ***
***              g (x[0],...,x[no_genes-1] >= 0                        ***
***               j                                                    ***
***                                                                    ***
*************************************************************************/


float constraint_value(int active_constraint,
                       struct individual *current)
{
  float *x;

  x = current-> gene;

  switch (active_constraint) {

    case 0               : return(...);

/*************************************************************************
***          The return value has to be set >= 0                       ***
***          if the active_constraint is fulfilled and                 ***
***          has to be set < 0 if it is violated         ***
*************************************************************************/

    ..................................

    case no_constraints-1: return(...);

  }
}
```

```
/*****************************************************************************
***                                                                        ***
***                      Actions for the                                   ***
***                        first call                                      ***
***             of the fitness function model                              ***
***                                                                        ***
*****************************************************************************/

void first_call_interface (void)
{
}

/*****************************************************************************
***                                                                        ***
***                       Actions for the                                  ***
***                          last call                                     ***
***                of the fitness function model                           ***
***                                                                        ***
*****************************************************************************/

void last_call_interface (void)
{
}

/*****************************************************************************
***                                                                        ***
***                  Actions after each generation                         ***
***                                                                        ***
*****************************************************************************/

void generation_interface (void)
{
}

/**********************  end  template.c  ******************************/
```

```
/********************** begin strat_al.h  ******************************/
/****                                                                ****/
/****      Header file to pick up  common control parameters         ****/
/****      of evolution algorithms included in the Evolution         ****/
/****      Machine.                                                  ****/
/****      The user has the possibility to code his own control      ****/
/****      and actions by means of these parameters and the in-      ****/
/****      terface routines (see our Template Fitness Model)         ****/
/****                                                                ****/
/************************************************************************/

extern long generation;                     /* Counter of generations  */
extern long generation_limit;               /* max generation          */
extern long time_limit;                     /* max cpu-time in seconds */
extern int convergence_mode;
       /* =1: The difference in the fitness values between the */
       /*     best and worst parents at the start of each      */
       /*     generation is used to determine wether to        */
       /*     terminate the search before the time linit is     */
       /*     reached. It is assumed that no_parents > 1.      */
       /* >1: (best >= 2*no_genes). The change in the mean     */
       /*     of all parental fitness function values in       */
       /*     convergence_mode generations is used as the      */
       /*     search termination objective.                    */
       /* In both cases epsilon(3) serves as the absolute and  */
       /* epsilon(4) as the relative bound for the decision to */
       /* terminate the search.                                */

extern float epsilon[5];
       /* epsilon[0],epsilon[1],epsilon[2] have different in-  */
       /*              ternal use and should be changed by the */
       /*              authors, only                           */
       /* epsilon[3] : Limit of absolute value of fitness      */
       /*              function differences for convergence    */
       /*              test                                    */
       /* epsilon[4] : as epsilon[3] but relative              */

       /* The following variables are valid in a population    */
       /* according the generation counter:                    */
extern float best_fitness;               /* Best fitness in the pop.  */
extern float worst_fitness;              /* Worst fitness in the pop. */
extern float new_fitness_sum;
       /* Sum of all fitness values in the population          */
extern float old_fitness_sum;
       /* Sum of all fitness values in the population at       */
       /* convergence_mode  generations before                 */

extern int no_parents;
extern int no_offsprings;
extern int no_genload;
extern int no_steps;
extern int no_angles;

/********************** end  strat_al.h  ******************************/
```

Appendix 2: Listings of all Control Parameters, used in the Performance Tests

Algorithm RB-(1+1):

```
************************************************************
***           EVOLUTION STRATEGY BY RECHENBERG           ***
************************************************************

MUTATION        Absolute Lower Bound    = 1.00000E-15
                Individual Variance     = 1.00000E-15
                Step Size Factor        = 8.50000E-01
                Factor Corr. Mutations  = 1
                No. Distinct Variances  = 10
                Correlated Mutations    = No
                Correlation Factor      = 8.72666E-02
                No. Correlation Angles  = 1
LIMITS          No. Constraints         = 10
                Time Limit (sec)        = 1000
                Generation Limit        = 25000
CONVERGENCE     Check After Generations = 25000
                Abs. Fitness Difference = 1.00000E-30
                Rel. Fitness Difference = 1.00000E-20
INITIAL VECTOR
                Genes                   = Random(xl ... xu)
                Steps                   = Random(xu-1 ... xu)
```

Algorithm RS-(10,100):

```
************************************************************
***     EVOLUTION STRATEGY BY RECHENBERG & SCHWEFEL     ***
************************************************************

POPULATION      No. Parents             = 10
                No. Offspring           = 100
                No. Genes               = 10
RECOMBINATION   Recombination Type      = Random Genes from Gene Pool
SELECTION       Selection Type          = Offspring Only
MUTATION        Global Variance         = 3.16228E-01
                Absolute Lower Bound    = 1.00000E-06
                Individual Variance     = 5.62341E-01
                No. Distinct Variances  = 10
                Correlated Mutations    = No
                Correlation Factor      = 8.72666E-02
                No. Correlation Angles  = 1
                Mutation Type           = Indiv. Changes
                Mutation Factor         = 1.50000E+00
LIMITS          No. Constraints         = 10
                Time Limit (sec)        = 1000
                Generation Limit        = 250
CONVERGENCE     Check After Generations = 250
                Abs. Fitness Difference = 1.00000E-30
                Rel. Fitness Difference = 1.00000E-20
INITIAL VECTOR
                Genes                   = Random(xl ... xu)
                Steps                   = Random(xu-1 ... xu)
```

Algorithms BO-(15†5+1) and BO-(30†20,200)l (in brackets):

```
************************************************************
***    EVOLUTION STRATEGY WITH A GENETIC LOAD BY BORN    ***
************************************************************

POPULATION     No. Parents             = 15  (30)
               No. Offspring           = 1  (200)
               No. Indiv. Genetic Load = 5   (20)
               No. Genes               = 10
RECOMBINATION  Recombination Type      = Random Genes from Parent Pairs
INVERSION      Probability Inversion   = 0.00000E+00
SELECTION      Selection Type          = Whole Population
                                         (Offspring Only)
MUTATION       Global Variance         = 3.16228E-01
               Absolute Lower Bound    = 1.00000E-06
               Individual Variance     = 5.62341E-01
               No. Distinct Variances  = 10
               Learning Rate Mutations = No (Yes)
               Correlated Mutations    = No
               Correlation Factor      = 8.72666E-02
               No. Correlation Angles  = 1
               Mutation Type           = Indiv. Changes
               Mutation Factor         = 1.50000E+00
LIMITS         No. Constraints         = 10
               Time Limit (sec)        = 1000
               Generation Limit        = 25000 (125)
CONVERGENCE    Check After Generations = 25000 (125)
               Abs. Fitness Difference = 1.00000E-30
               Rel. Fitness Difference = 1.00000E-20
GENETIC LOAD   Fixation Type Genes     = Random Genes
               Fixation Type Steps     = Random Steps
INITIAL VECTOR
               Genes                   = Random(xl ... xu)
               Steps                   = Random(1.0E-4 ... xu)
INITIAL VECTOR GENETIC LOAD
               Genes                   = Random(xl ... xu)
               Steps                   = Random(1.0E-4 ... xu)
```

Algorithm GO-(50,50)e:

```
*****************************************************************
***               GENETIC ALGOTITHM BY GOLDBERG               ***
*****************************************************************


POPULATION      No. Parents/Offspring   = 50
                No. Genes               = 10
PARAMETERS      Cromosome Length        = 320
                Crossover Probability   = 5.97656E-01
                Mutation  Probability   = 3.32031E-02
LIMITS          No. Constraints         = 10
                Time Limit (sec)        = 1000
                Generation Limit        = 500
CONVERGENCE     Check After Generations = 500
                Abs. Fitness Difference = 1.00000E-30
                Rel. Fitness Difference = 1.00000E-20
INITIAL VECTOR
                Genes                   = Random(xl ... xu)
                Lenght_Parm_All         = 32
                Min_Parm_All            = xl
                Max_Parm_All            = xu
```

Algorithm VB-(15†5+1):

```
*****************************************************************
***             GENETIC ALGORITHM  BY VOIGT & BORN            ***
*****************************************************************

POPULATION     No. Parents             = 15
               No. Offspring           = 1
               No. Indiv. Genetic Load = 5
               No. Genes               = 10
RECOMBINATION  Recombination Type      = Random Genes from Parent Pairs
INVERSION      Probability Inversion   = 0.00000E+00
SELECTION      Selection Type          = Whole Population
MUTATION       Global Variance         = 3.16228E-01
               Absolute Lower Bound    = 1.00000E-06
               Individual Variance     = 5.62341E-01
               No. Distinct Variances  = 3
               Correlated Mutations    = No
               Correlation Factor      = 8.72666E-02
               No. Correlation Angles  = 1
               Mutation Type           = Indiv. Changes
               Mutation Factor         = 1.50000E+00
LIMITS         No. Constraints         = 10
               Time Limit (sec)        = 1000
               Generation Limit        = 25000
CONVERGENCE    Check After Generations = 25000
               Abs. Fitness Difference = 1.00000E-30
               Rel. Fitness Difference = 1.00000E-20
GENETIC LOAD   Fixation Type Genes     = Random Genes
               Fixation Type Steps     = Random Steps
INITIAL VECTOR
               Genes                   = Random(xl ... xu)
               Steps                   = Random(0.01 ... 0.1)
INITIAL VECTOR GENETIC LOAD
               Genes                   = Random(xl ... xu)
               Steps                   = Random(0.01 ... 0.1)
```

References

[1] F. Archetti and M. Cugiani. *Numerical Techniques for Stochastic Systems.* North–Holland, Amsterdam, New York, Oxford, 1980.

[2] J.E. Baker. Adaptive selection methods for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 101–111, New Jersey, 1985. Hillsdale.

[3] K. Bellmann and J. Born. Numerical solution of adaptation problems by means of an evolution strategy. In *Modelling and Optimization of Complex Systems. Proceedings of the IFIP-TC7 Working Conference, Novosibirsk, Lecture Notes in Control and Information Science Vol. 18*, pages 157–167, Berlin,Heidelberg,New York, 1979. Springer.

[4] J. Born. *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben.* PhD thesis, Humboldt Universität zu Berlin, 1978.

[5] J. Born. Adaptively controlled random search – a variance function approach. *Systems Analysis, Modelling, Simulation*, 2:109–112, 1985. Akademie–Verlag, Berlin.

[6] J. Born and K. Bellmann. Numerical adaptation of parameters in simulation models using evolution strategies. In K. Bellmann, editor, *Molecular Genetic Information Systems: Modelling and Simulation*, pages 291–320. Akademie–Verlag, Berlin, 1983.

[7] H.J. Bremermann. Numerical optimization procedures derived from biological evolution processes. *Cybernetic Problems in Bionics*, 1968. Gordon and Breach, New York.

[8] H.J. Bremermann. A method of unconstrained global optimization. *Mathematical Bioscience*, 9:1–15, 1970.

[9] L.C.W. Dixon and G.P. Szegoe. *Towards Global Optimization 1 and 2.* North–Holland, Amsterdam,New York,Oxford, 1975 and 1978.

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

[11] A.O. Griewangk. Generalized Decent for Global Optimization. *JOTA*, 34:11–39, 1981.

[12] J.B.S. Haldane. The cost of natural selection. *J. Genet.*, 55:511–524, 1957.

[13] F. Hoffmeister and Th. Bäck. Genetic Algorithms and Evolution Strategies: Similarities and Differences. In University of Dortmund, editor, *PPSN - First International Workshop on Parallel Problem Solving from Nature. October 1-3, 1990. Dortmund, FRG. Preprints*, pages A–VII, 1–14, 1990.

[14] J.H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[15] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michgan, Dissertation Abstracts International 36(10), 5140B. (University Microfilms No. 76-9381), 1975.

[16] K. Marti. Random search in optimization problems as a stochastic decision process (adaptive random search). *Methods of Operations Research*, 36:223–234, 1980.

[17] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[18] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 1991. to be published.

[19] U.G. Oppel. Evolution und Optimierung, Teil 1 (Evolution ohne Rauschen). Vorlesung "Auf der Zufallssuche basierende Optimierungsverfahren und Evolutionsprozesse", 1978. L–M–Universität, München.

[20] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog, Stuttgart, 1973.

[21] I. Rechenberg. Höhere ES-Optimierung, 1991. Script April 1991, Erg Chebbi (Sahara), Lecture in the Seminar "Ausgewählte Kapitel zur Bionik und Evolutionsstrategie", TU Berlin.

[22] J.D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms and their Applications.* Morgan Kaufmann, 1989.

[23] B. Schneider and U. Ranft. *Simulationsmethoden in der Medizin und Biologie. Workshop Hannover 1977.* Springer, Berlin, Heidelberg, New York, 1978.

[24] N. Schraudolf. Genetic algorithm software survey, 1992. Anonymous ftp from ftp.aic.nrl.navy.mil, file /pub/galist/information/ga-software-survey.txt.

[25] H.P. Schwefel. Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie. *Interdisciplinary Systems Research; Birkhäuser, Basel and Stuttgart*, 26, 1977.

[26] H.P. Schwefel. *Numerical Optimization of Computer Models.* J. Wiley, Chichester, New York, Brisbane, Toronto, 1981.

[27] J.A. Snyman and L.P. Fatti. A Multi-start global minimization algorithm with dynamic search trajectories. *JOTA*, 54:121–141, 1987.

[28] A. Törn and A. Zilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Sciences.* Springer, Berlin, 1989.

[29] H.-M. Voigt. *Evolution and Optimization.* Akademie Verlag, Berlin, 1989.

[30] H.-M. Voigt and J. Born. A structured distributed genetic algorithm for function optimization, 1991. submitted to the Fourth Conference on Genetic Algorithms, San Diego, July 1991.

[31] H.-M. Voigt, J. Born, and I. Santibanez-Koref. Modelling and simulation of distributed evolutionary search processes. Preprints of the First International Workshop on Parallel Problem Solving from Nature (PPSN), October 1–3, 1990, Dortmund, A XXII., 1990.

[32] H.-M. Voigt, H. Mühlenbein, and H.P. Schwefel. *Evolution and Optimization '89. Selected Papers on Evolution Theory, Combinatorial Optimization and Related Topics.* Akademie Verlag, Berlin, 1990.

[33] D. Whitley. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best . In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, CA, 1989. Morgan Kaufmann.