GLEAM - an Evolutionary Algorithm for Planning and Control Based on Evolution Strategy

Christian Blume[1], Wilfried Jakob[2]

[1] University of Applied Sciences, Cologne, Campus Gummersbach,
Am Sandberg 1, 51643 Gummersbach, Germany
blume@gm.fh-koeln.de
[2] Forschungszentrum Karlsruhe, Institute for Applied Computer Science,
P.O. Box 3640, 76021 Karlsruhe, Germany
jakob@iai.fzk.de

Abstract. An evolutionary algorithm based on Evolution Strategy (ES) is presented, which includes time-related command execution and the generation of process control elements. Its concept is enlarged by problem-oriented type definitions for parameters, this has allowed a flexible implementation for different applications. The GLEAM algorithm includes new features which distinguish it from ES and GAs, among them especially a new and flexible kind of coding allowing a natural problem representation. The different kind of code interpretation is tailored, but not limited to finding solutions for time-dependent processes like the control of (industrial) robots or autonomous vehicles.

1 Introduction

The GLEAM (General Learning Evolutionary Algorithm and Method) algorithm was designed to generate optimized plans, i.e. sequences of commands, to control dynamic processes. First example of such a problem solved by GLEAM was the control of industrial robot movements. The algorithm should generate a command sequence consisting of robot control primitives, i.e. a robot language program, and the execution of these statements should result in an optimized robot move trajectory avoiding collision.

The genetic code of biological beings describes the attributes of the beings, e.g. color of the skin or their height. But additionally it controls dynamic processes of beings like growing up. This leads to the concept of action of GLEAM. A GLEAM code element represented by a more or less complex data type describes an attribute as well as a control command. The evaluation of control commands requires the execution of the command sequence (by simulation) to calculate the fitness value. Another property of the action concept is the not fixed number of genes resp. actions of an individual.

From the beginning the GLEAM concept formulated by Blume in [1] includes these new features compared to already existing concepts of ES [2] and GA [3]. It was implemented by the two authors using a robot path planning task for testing and enhanced by the well-established concepts of ranking-based selection and structured populations [4] as described later. Even there are implementations of GLEAM to different applications, up to now a more detailed theoretical description of the method was missing. The field of applications was enlarged by production planning [5], resource optimization [6], traverse path minimization [7], and design optimization [8]. The latter area of application is the area of interest of the Research Center, while most industrial applications and especially the further developments of collision-free robot path planning [9] were accomplished at the University of Applied Sciences of Cologne. The references above should document the many-sidedness and the easy transferability to new optimization problems of GLEAM.

2 Concept of GLEAM

The first applications of Evolution Strategies were optimization problems. Such a problem is described by a set of parameters defining how to build up a solution, e.g. for a production plan or the construction of a nozzle. This is a static point of view, which corresponds to the calculation of the fitness value by a fitness function, a simulation run or experiments in the real world. In addition to this, GLEAM focuses on dynamic interpretation and time dependant simulation or execution of a sequence of actions.

An important basis of the GLEAM concept are the changeable code elements, which represent so-called actions instead of pure parameters. The evolutionary code of a GA is a bit string and the representation of the ES is a vector of real numbers and strategy parameters. In GLEAM, the code is a sequence of actions which have to be performed for the calculation of the fitness value. As, in general, it is not known in advance how many actions are needed to solve a given problem, e.g. a complex robot movement, the length of the action sequence (also called action chain) is not fixed, but changed by evolution. In the simplest case, an action consists of the action identifier (an integer) only. In other cases, actions may have a number of parameters. An action is something like a control command, because all actions have to be performed during simulation (to calculate the fitness value) and later in real world application, they have to be executed by a control device, e.g. a robot control unit. The simulation has to model as exactly as possible the behavior of the control, including the time-dependent process and the environment, if applicable. Therefore, GLEAM includes a more dynamic point of view of the meaning of the evolutionary code. Some of the actions can be the reading of sensor values and a conditional reaction to them. This provides a response to environmental changes of the generated action sequence.

3 The Coding

In comparison to the ES and GA, the GLEAM genetic code structure is enlarged by the structure of so-called segments. The genetic code for an individual of GLEAM consists of a number of segments (minimum one). Each segment consists of a number of actions. An action consists of an action identifier, optionally followed by a number of parameters of type integer or real (or character, which are represented as integer, too, that is only a matter of interpretation). A population M of μ individuals is defined as follows:

$M = \{ c_1, c_2, \dots ,c_\mu \}$

$c_i = (H, S_1, S_2, \dots ,S_r)$ with $H = (l_1, l_2, \dots ,l_r)$ $l_i \in \mathbb{N}$

$S_i = (a_{i1}, a_{i2}, \dots ,a_{il})$ with $i \in \mathbb{N}$ and $l \in H$

$a_{ik} = (ak)$ or

$a_{ik} = (ak , p_1, p_2, \dots ,p_z)$ with $ak, z \in \mathbb{N}$ and $p_i \in \mathbb{N} \cup \Re$

The meaning of the structure elements is:

c_i:	action chain, an individual
H:	housekeeping information of an individual, e.g. segment length
S_i:	segment consisting of an action sequence
a_{ik}:	action
ak:	action identifier (integer)
p_i:	parameter of an action (integer or real)

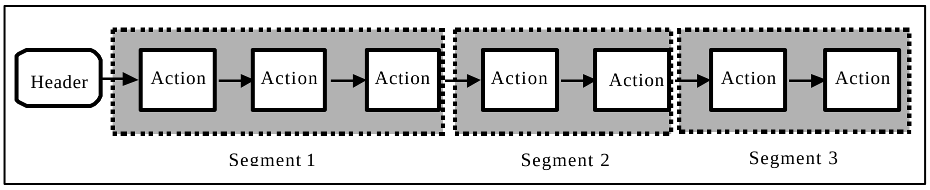


Fig. 1. Action chain (an individual) consisting of segments of actions

A data file exists, which includes a description of each action type. There it is defined, how many parameters of which type belong to an action. This information is used by the evolution machine, as described later. It allows an adaptation to new applications without new programming the changed gene representation.

3.1 Genetic Code Structure in Nature and in GLEAM

The genetic code of all biological beings consists of sequences of an alphabet of four bases only: U (Uracil), C (Cytosin), A (Adenin), and G (Guanin). Three of these basic elements form so-called triplets which are part of the genes. Without going into further details a hierarchical structure exists in nature for representing the genetic information, which has not yet been completely analyzed. But it is known that e.g. there is a start triplet AUG of an information string called gene. Other structures correspond to the folding of the string and other features. A mutation of a basic element of a gene can change the type of the gene, for example:

C-C-U-G-A-G-G-A-G	normal gene type, no illness
C-C-U-G-**U**-G-G-A-G	genetic defect, illness (haemolytic anaemia)

The (rough) model of biological coding was transferred to GLEAM. The basic elements of the gene types in GLEAM are the actions as described above. The higher structure level is formed by the segments similar to the gene structures.

The genetic operators like translocation or deletion of an element are applied to actions as well as to segments. As the crossover is performed by an exchange of segments, good substructures represented by segments (i.e. a sequence of actions) of one parent may probably be combined with good substructures of the other parent to speed up the evolution.

3.2 Action Concept for Planning Dynamic Processes and Control

The result of an action of GLEAM is defined by the implementation of the interpretation and simulation procedure performing this action. The simulation of an action chain representing an individual is necessary to calculate the fitness value for this individual. The simulator can be treated as an interpreter of the action code. If GLEAM is applied to a task to optimize plans, e.g. the production plan for scheduling or the sequence of cities for the TSP, the "action" is performed by changing the sequence of plan elements (i.e. actions) or parameter values.

However, for the planning of dynamic processes like the control of an industrial robot, this is not sufficient. Therefore, the action concept includes a relation of the action chain to a time scale. Based on the cycle time Δt of the control unit(s) for the process, the relation defines: If action a_i starts at t_i, the action a_{i+1} starts at $t_i + \Delta t$. In a simple case, the mapping of actions to the time scale is

action a_1	$\longrightarrow$	t_0
action a_2	$\longrightarrow$	$t_0 + \Delta t$
……		……
action a_i	$\longrightarrow$	$t_0 + (i-1)*\Delta t$

There are two enlargements for a differentiated control of this simple time sequence:

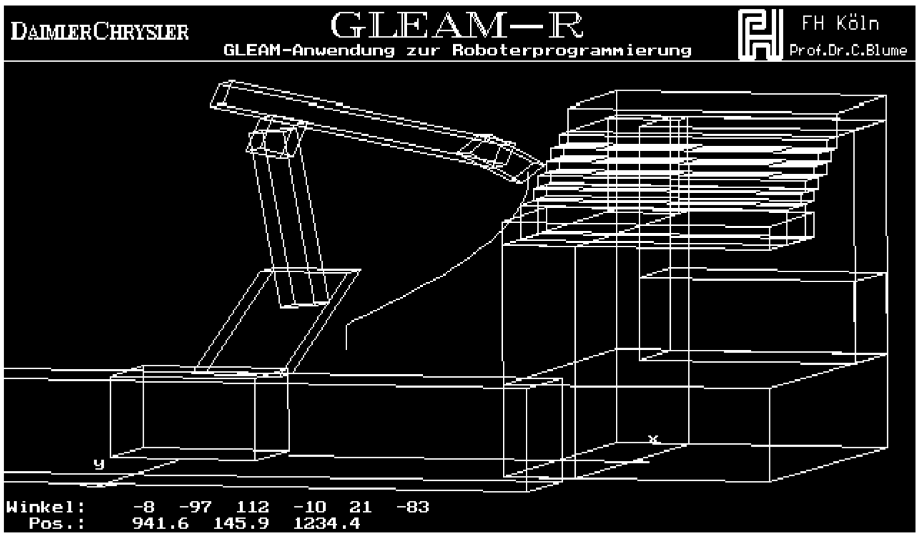
1. There are two special actions called BLOCK_BEGIN and BLOCK_END. All m actions between these two actions will start at the same time t_n:

action a_i: BLOCK_BEGIN	$\longrightarrow$	t_n
action a_{i+1}	$\longrightarrow$	t_n
action a_{i+2}	$\longrightarrow$	t_n
……	……	
action a_{i+m}	$\longrightarrow$	t_n
action a_{i+m+1}: BLOCK_END	$\longrightarrow$	t_n
action a_{i+m+2}:	$\longrightarrow$	$t_n + \Delta t$

2. There is an action called UNCHANGED with a parameter n of type integer. This action is interpreted as follows: The next action in the action chain after this action will start after n time slices:

action a_i	$\longrightarrow$	t_i
action a_{i+1}: UNCHANGED n	$\longrightarrow$	ti + Δt
action a_{i+2}:	$\longrightarrow$	$t_i + n*\Delta t$

The effect of the start of an action can last much longer than the associated time slice. For example, the action "start the movement of a robot axis" will move the robot axis until an action "stop robot axis" is executed or until the move limit of the robot axis is reached. The simulation should be performed by modeling the time relation as exactly as possible.



```
Winkel:    -8  -97  112  -10  21  -83
  Pos.:    941.6  145.9  1234.4
```

Fig. 2. Simulated execution of robot move statements generated by GLEAM in an industrial environment, avoiding collisions with obstacles and workpieces [9].

The example of Fig. 2 illustrates the effect of time-dependent interpretation of the action chains. GLEAM was applied successfully to generate and optimize collision-free robot trajectories. Other solutions have already been found for this problem. But instead of optimizing many (statically defined) robot configurations (see [10, 11]), GLEAM generates and optimizes the control commands, which can be loaded on the robot control unit. This gives a more realistic result and a useful output for industrial applications.

3.3 Type Concept of Actions and Action Chains

The concept of action chains was implemented using the concept of abstract data types from computer science. The coding is based on problem-configurable typed objects called *actions*. The content of an action is defined by its type definition, which constitutes its set of real, integer or Boolean parameters together with their ranges of values. The plainest action type consists of no parameters at all for pure sequencing problems or exactly one real parameter, which is one suitable form for parameter optimization. An *action chain* consists of objects of previously defined action types. The rules for constructing such a chain are defined according to the application, based on three classes of chains at present:

1. Fixed-length action chains with irrelevant action order
   A chain consists of exactly one object of every action type.
2. Fixed-length action chains with relevant action order
   A chain consists of exactly one object of every action type in an arbitrary sequence.
3. Variable-length action chains with relevant action order
   A chain consists of none, one, or more objects of every action type in an arbitrary sequence. Therefore, probabilities of appearance are assigned to every action type.

The *segmentation* of a newly generated chain is done by arbitrary insertion of segment boundaries according to given limitations of the segment length. The type definitions, the rule descriptions for chain construction, and the segment limits form the so called *model of activities*, i.e. a model of what the evolution can affect. This concept provides the user with a flexible mechanism of naturally mapping his problem to the gene structure (action chains, actions and parameters), often resulting in genotypes from which phenotypic properties can be derived easily. For setting up an activity model no programming is required. All the user has to do is writing his definitions of activities and the chain construction parameters into a file. Action chains are implemented as linear lists as shown in Fig.1, because this is very useful for many of the genetic operators described in the next chapter.

4 The Genetic Operators

The genetic operators are controlled by the model of activities in terms of what can be evolved with respect to limitations and by the configuration of the *evolutionary operations*. An evolutionary operation consists of one or more genetic operators like mutation or crossover and produces one or two offsprings. Both, the evolutionary operations and the genetic operators are assigned a probability of execution. Thus, the exact number of offsprings of one mating and the operators involved in their production are a matter of chance.

The concept of the type definitions of genes allows for mutation operators that take explicit restrictions into account. Thus, a set of general genetic operators can be implemented, which obtain their application-specific information from the definitions of the activity model used for the initialization of GLEAM. Another advantage of this approach is that problem-specific genetic operators can be added easily to the set of general ones, if desired.

4.1 Mutation of Actions

The set of applicable mutations of actions depends on the chain class they are applied to. In general, there are parameter mutations, positional mutations, and mutations to delete, double or generate actions.

The relative parameter mutation is inspired by the ES: At first, it is randomly chosen whether to increase or decrease the actual value and then, the range of possible alteration is calculated. This range is divided into ten equidistant classes, of which one is chosen by chance to define the actual range of mutation, from which the value of modification is selected randomly. The resulting probability distribution is shown in Fig. 3. This is faster than calculating the normal distribution, as done with ES, but has a similar effect: Greater changes are less likely than smaller ones. In contrast to ES, however, there are no evolved strategy parameters for mutation. The absolute mutation just generates a new random value for a parameter within its limitations. Table 1 summarizes the standard mutation operators for actions.

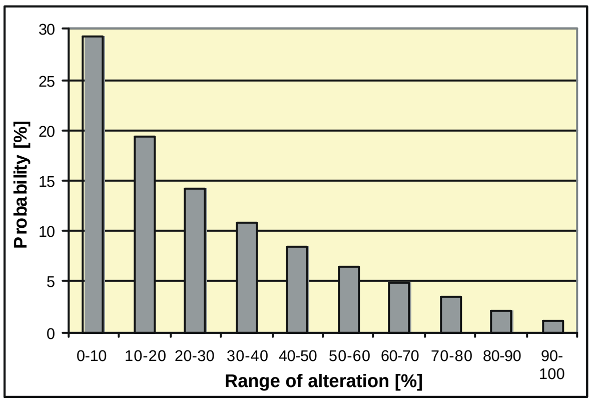


Fig. 3. Probability distribution for parameter changes

Table 1. Mutations for actions and their applicability to the different chain classes.

Mutation	Action Chain Type (length, relevance of action order)		
	fixed, irrelevant	fixed, relevant	dynamic, relevant
Alteration of the value of a parameter	yes	yes	yes
Small parameter alteration (10% of the range)	yes	yes	yes
New value for one parameter of an action	yes	yes	yes
New value for all parameters of an action	yes	yes	yes
Translocation of an action		yes	yes
Substitution of an action by a new one			yes
Insertion of a new action			yes
Deletion of an action			yes
Doubling of an action			yes

4.2 Mutation of Segments

The set of segment mutations at first consists of the action mutations applied to the actions of a segment. E.g., the parameter mutation of a segment means to execute the basic operator with a randomly chosen number of actions of the particular segment. In the same manner, the position of a whole segment can be changed or a complete new segment inserted.

Additionally, there is an operation called *inversion*, which reverses the order of the actions within a segment.

Mutation of the segment boundaries may also occur as well as the merging and separation of segments. As these operations do not affect the phenotype of the action chain, they are only applied as background operators in conjunction with the other mutation operators. Table 2 gives an overview of the segment mutations.

4.3 Crossover

Three crossover operators are available: The standard 1-point and n-point crossover operators and a special operator, which exchanges just one segment between the two chains. Crossover is always performed on segment boundaries and yields two offsprings.

Table 2. Mutations for segments and their applicability to the different chain classes.

Mutation	Action Chain Type (length, relevance of action order)		
	fixed, irrelevant	fixed, relevant	dynamic, relevant
Parameter alteration of the actions of a segment	yes	yes	yes
Small parameter alteration of the actions of a segment (10% of the range)	yes	yes	yes
New value for one parameter of the actions of a segment	yes	yes	yes
Integration of adjacent segments	yes	yes	yes
Splitting of a segment	yes	yes	yes
Shifting of segment limits	yes	yes	yes
Inversion of a segment		yes	yes
Translocation of a segment		yes	yes
Integration of two non-adjacent segments		yes	yes
Substitution of a segment by a new one			yes
Insertion of a new segment			yes
Deletion of a segment			yes
Doubling of a segment			yes

5 The Evolutionary Engine

GLEAM uses a structured population based on the neighborhood model introduced by Gorges–Schleuter [4]. The idea is that mates are not selected from the entire population, as it is the case with panmictic populations, but from a neighborhood of the individual, the so-called deme. For this purpose, the individuals are placed on a geographical structure like e.g. a ring. As the acceptance of an individual is done within the deme of its parent too, only local interaction of the individuals takes place. Now, information can spread only via the areas of deme overlapping and, as a consequence, niches of comparable good content establish. This process is much more likely as with panmictic populations. After a certain amount of generations, the niches begin to merge, and the competition between the best approaches to the solution begins. The advantages of this concept do not only comprise favorable properties for parallelization, but also a better promotion of global diversity preventing premature convergence, and an adaptive balance between depth and width search. For multimodal problems, and most practical problems are of that type, this is the procedure of choice, because more different areas of the search space are investigated. [12].

GLEAM is a steady-state EA: It uses ranking-based mate selection and different acceptance rules for the best offspring of every mating. The child must either be better than the parent to be substituted or, with less selective pressure, better than the weakest individual of the deme. An elitist modification of the latter acceptance rule is available, and it was used for most applications of GLEAM. In ES terms, GLEAM can be characterized as follows:

$\mu/\rho+\lambda$ where μ is the population size
ρ is the number of parents, at present two
λ is the number of offsprings (typically between 3 and 10 times of μ)

Start populations can be generated by random initialization or by using already existing solutions (which may be from variants of the task as long as they are based on the same activity model) or a mixture of both. For reusing action chains, they can be stored into a file and loaded back again.

6 Conclusion: Positioning of GLEAM within the Range of Evolutionary Algorithms

Exact positioning of an evolutionary algorithm seems to be impossible, because there are too many new developments in the different communities, which are oriented to ES or GA concepts, but enlarge their algorithms by features coming from outside. The "nearest" connection is between GLEAM and ES, but there are also similarities between GLEAM and GA. In some way, there even is a likeness of GLEAM to GP, in so far as GLEAM may generate a control program. Table 3 tries to summarize the similarities and differences of GA, ES, and GLEAM.

Table 3. Similarities and differences of GA, ES, and GLEAM

Features	GA	ES	GLEAM
representation	bit string	real vector	vector of data structures of integer /real
basic elements of the solution	parameters	parameters	actions with parameters
assignment of genotype to phenotype	mapping of bits to the entities of the solution	reals are parameters of the solution	problem-specific action types
natural problem representation	no	yes	yes
genetic operators	mutation and recombination	mutation and recombination	mutation and recombination on different levels
adaptive behavior	coding of control parameters	adaptive deviations	segments for sub-solutions
kind of problem solutions	static description	static description	static description and dynamic control

The GLEAM concept includes the following new features:

- Action concept: Problem-oriented type definitions, parameters of different data types
- Meta elements: Segments which represent sub-solutions to speed up evolution
- Genetic operators for basic and meta elements
- Code interpretation: Action execution instead of parameter interpretation
- Description of dynamic processes: Time-related action execution and generation of process control elements

With these features, GLEAM can be regarded as an EA positioned between ES and GA, which is tailored, but not limited to evolving dynamic control processes.

This paper was written using the terms and definitions of VDI/VDE 3550 [13].

References

1. Blume, C.: GLEAM - A System for Intuitive Learning. In: Schwefel, H.P., Männer, R. (eds.): Proc. of PPSN I, LNCS 496, Springer, Berlin (1990) 48-54
2. Rechenberg, I.: Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart-Bad Cannstatt (1973)
3. Holland, H.J.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
4. Gorges-Schleuter, M.: Genetic Algorithms and Population Structures - A Massively Parallel Algorithm. Doctoral thesis, University of Dortmund, Germany (1990)
5. Blume, C.: Planning and Optimization of Scheduling in Industrial Production by Genetic Algorithms and Evolution Strategy. In: Proc. of 2nd Biennial European Joint Conf. on Engineering, Systems, Design, and Analysis (ESDA). (1994) 285-292
6. Blume, C., Jakob, W.: Cutting Down Production Costs by a New Optimization Method. In: Proc. of Japan-USA Symposium on Flexible Automation. ASME (1994)
7. Blume, C.: Optimization in Concrete Precasting Plants by Evolutionary Computation. In: Whitley, D. et al. (eds.): Proc. GECCO 2000, Vol. Late Papers, M. Kaufmann, (2000) 43-50
8. Jakob, W., Quinte, A., Scherer, K.-P., Eggert, H.: Optimization of a Micro Fluidic Component Using a Parallel Evolutionary Algorithm and Simulation Based on Discrete Element Methods. In: Hernandez, S., et al.: Computer Aided Design of Structures VII, Proc. of OPTI'01, WIT Press, Southampton (2001) 337-346
9. Blume, C.: Programming of Industrial Robots by the GLEAM Evolutionary Algorithm. In: Callaos, N. et al. (eds.): Proc. SCI'2001, Vol. III, IIIS, Orlando, (2001) 151-156
10. Ortmann, Matthias, Weber, Wolfgang: Multi-Criterion Optimization of Robot Trajectories with Evolutionary Strategies, Proc. GECCO 2001, Spector, Lee et. al., San Francisco (2001) 310 (Late breaking papers)
11. Solteiro Pires, E.J., Tenreiro Machado, J.A.: Trajectory Optimization for Redundant Robots Using Genetic Algorithms, Proc. GECCO 2000, M. Kaufmann, San Francisco (2000) 967
12. Gorges-Schleuter, M.: A Comparative Study of Global and Local Selection in Evolution Strategies. In: A. Eiben et al. (eds.): Proc. PPSN V, LNCS 1498, Springer, (1998) 367-377
13. Beyer, H.-G., et al.: Evolutionary Algorithms – Terms and Definitions. VDI/VDE-Richtlinie-3550, Blatt 3, Gründruck (in German with English glossary; English version to be published in 2002). VDI, Düsseldorf (2001)