GPCORE – A GENERIC FRAMEWORK FOR GENETIC PROGRAMMING

Torben Pullmann, Martina Schnellenbach-Held, Peer Lubasch
Institute of Structural Concrete Essen, Essen, Germany

ABSTRACT: Complex engineering tasks are rarely unique in their solution. The question is not to determine a single solution – it is more to retrieve an optimal solution. Automated, software-based optimization technologies turned out to be a very promising and applicable solution for the class of non-deterministic optimization problems.

In this paper a generic framework for the application of genetic programming will be introduced. The framework is based on the Backus-Naur representation, which can be seen as a meta-programming language. According to specific demands on engineering problems, various detail solutions have been evolved in this context. Finally, the application of the developed framework in load identification will be presented. An example of an evolutionary optimization within the analysis of measured data will be given. By means of two encapsulated optimization routines structural responses recorded at instrumented bridges are analyzed to determine gross weights, velocities, axle loads as well as axle spacings of passing vehicles. Most significant aspects of the developed framework will be covered within the example: Problem decomposition, problem space definition based on the meta-language, genotype generation, examples of crucial crossover and mutation operations, fitness evaluation and entire operation of the optimization process.

KEYWORDS: genetic programming, framework, genetic algorithms, evolutionary algorithms, evolutionary optimization, evolutionary computation.

1 INTRODUCTION

Evolutionary algorithms evolved to very powerful optimization strategies for finding solutions to non-deterministic optimization problems. Actually many different EA-types are co-existing (Spears et al. 2003, Whitley 2001), while genetic algorithms (GA) and evolution strategies (ES) represent the most common variants. In EA the individual solutions are denoted as "phenotype", their coded counterpart as "genotype" (Geyer-Schulz 1995). The genotype is normally coded in form of a long binary or string representation. The main objective of any EA is to produce "better" individuals based on a set of given or randomly chosen solutions. To measure the quality or "fitness" of an individual, a problem specific fitness function is applied. During the evolutionary process, new generations of individuals are produced. The forming of a generation follows Darwin's principal "survival of the fittest", which is obtained by fitness-addicted selection of parent individuals to form a new generation. Furthermore, genetic operators are applied to the offspring, like crossover and mutation operators.

A demanding task during design and implementation of any optimization algorithm for a specific problem is a suitable problem space representation. Due to the complexity of engineering problems, individuals which are represented in form of their corresponding genotype usually contain a variety of different information types at a varying size. The higher the homogeneity of this information, the more likely simple linear representation types, like binary, real or string coded representation used for genetic algorithms, become inefficient through uncoordinated optimization operations and mostly fixed genotype lengths.

A groundbreaking variation of genetic algorithms is Genetic Programming, fundamentally introduced in Koza (1992). In this innovative approach genotypes are represented by tree-based computer programs instead of a linear representation. Originally designed to create programs for the language LISP, Koza's approach is generally adoptable to every optimization problem which can be formulated using a sufficient tree-based structure. The tree structure offers several advantages. Type safe nodes allow goal oriented and semantically correct crossover and mutation operations, which finally leads to dramatically better performance. Furthermore, a node-specific and very fine granular adjustment of the optimization parameters is possible, while the problem of fixed genotype size becomes completely obsolete. Besides tree representation, there are other representation paradigms in genetic programming, for instance linear sequential representation, which are not further addressed in this paper.

2 THE "GP CORE"

Actual research projects conducted by the authors use genetic programming in different scopes. It turned out that the realizing of genetic programming is a fairly demanding task during design and implementation phase of

scientific software applications. While the fitness landscapes of the optimization problems differ widely, the general genetic programming part remains similar but consumes an important amount of the development time and testing effort. The outcome of this was the design and development of a universal and robust framework, the "GPCore", which is very flexible and adoptable to a variety of different optimization problems.

The GPCore basically consists of a set of C++ classes which cover definitions and operations to apply genetic programming methodologies to real world problems. For definition and maintenance of templates and for performing test runs, the application "CPCore Manager" has been developed by the authors. A second application, the "GPCore Runtime Module", is the intristic runtime environment for genetic optimization tasks using the CPCore classes. In this context, a "program" as genotype produced by the GPCore may be a real software program following the rules of any known language syntax, or for most cases it may define a problem specific description of an individual. Mathematical functions can also be interpreted as a program, so these are used in the following examples. Dependent on the desired problem space, the interpretation and transformation of the genotypic "program" into phenotypes and the verification of feasibility as well as the fitness determinations are expected to be delivered by an external application. For this reason the genotype syntax is not further limited by the architecture of the GPCore.

2.1 *Template representation*

The template representation of the GPCore is based on the Backus-Naur Form (BNF) proposed in Geyer-Schulz (1995), which generally represents a metasyntax and is used in terms of a meta program language. Meta program languages define the grammar which finally describes a complete program language or an equivalent description grammar. As we consider an individual of the optimization process being a kind of a "program", the Backus-Naur representation allows defining the possibilities and requirements of these problem specific programs.

A genotype based on a BNF definition may be depicted in a tree, which is based on different kinds of nodes, also called "symbols". Symbols can be non-terminated or terminated (also called "terminals"). Non-terminated symbols refer to one or more child-symbols, which themselves can be either terminated or non-teminated. The root-node of a Backus-Naur tree is called start-symbol.

Numeric values are normally represented through a recursive combination of terminated and non-terminated symbols, which leads to different disadvantages during optimization. For this reason the Backus-Naur representation as introduced by Geyer-Schulz (1995) has been extended by a symbol called "value range". A value range contains min. and max. values and an increment. As a very simple example, the representation of a trigonometric function in the Backus-Naur notation may be defined as shown in figures 1 and 2.

```
<S>:=<Func> "(" <N> "*[x])"
<Func>:="sin"|"cos"|"tan"
<N>:=[0;10;0.1]
```

Figure 1. Backus-Naur representation of a simple mathematical function.

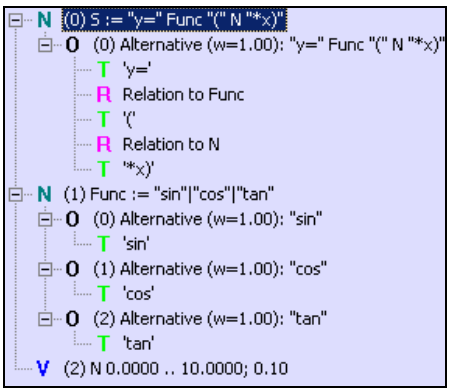

Figure 2. Backus-Naur representation in GPCore manager.

In this definition, "S" is the start symbol followed by an enumeration of terminated symbols and references to other symbols. "Func" is a non terminated symbol with three alternatives, and "N" is a value range between 0 and 10 with an increment of 0.1.

Generation of individuals is being performed by processing the BNF structure hierarchically. The diversity of individuals is obtained by applying the various decisions within the structure, in our example selection of a trigonometric function and extraction of a numeric value for N. On randomly defined individuals these decisions are performed by random values, which may be influenced by certain goal oriented weights. For instance it is possible to assign a higher weight for "sin" and "cos" than for "tan", which leads to higher selection probability for these alternatives. In the same manner, numeric values can be polarized to aspire a pre-known suitable value and its surrounding with a higher probability than values which are more different to the pre-known value.

This is realized through Gaussian shaping of the corresponding probability distribution. Possible individuals based on the program definition are shown in figure 3.

```
y=sin(3.1*x)
y=cos(9.7*x)
y=tan(2.1*x)
```

Figure 3. Individuals based on the definition in figure 1.

2.2 *Fitness evaluation*

Due to the generic architecture of the GPCore, fitness evaluation is up to an independent external routine which may be called directly or initiated by a hot folder data exchange mechanism. For simplification of the data exchange, the generated individuals are notated in pure string or xml representation. Dependent on the problem domain, single or multiple criterion fitness evaluation can be performed.

2.3 *Selection*

Based on the fitness or the pareto rank in case of multi criteria, children for a new generation are selected using either fitness proportionate selection or tournament selection. A defined elitism count ensures the preservation of the n best individuals, resp. the ones representing a suitable distribution within the first pareto rank.

2.4 *Crossover*

One of the most important and effective operations on genetic programming is the crossover operation. The decision about the utilization of a crossover operation to a newly selected individual is being performed on a random basis controlled by the global crossover rate. A crossover operation is being performed by the following steps:

Step 1: Selection of a second parent individual

Step 2: Selection of a randomly chosen node N1 within the first parent

Step 3: Selection of a randomly chosen node N2 of the same type like N1 within the second parent. Equity of types in this context means both nodes originate from the very same element within the BNF template. If there is no corresponding node of the same type, step 2 is repeated

Step 4: Exchange of the complete subtrees beginning at the selected nodes

Due to the type-aware subtree exchange, the consistency and integrity of a child after a crossover operation is guaranteed, as long both parents have been conform to the underlying BNF template (Koza 1994). Additionally each node within the BNF template allows the definition of an individual crossover variance, which enables a fine granular adjustment of the node selection.

2.5 *Mutation*

In contrast to most genetic programming approaches, the GPCore distinguishes between two completely different kinds of mutation, “Cumulative Mutation” and “Value Ordered Mutation”.

Cumulative Mutation

The “cumulative mutation” can be applied to any kind of symbol within the individual representation. A cumulative mutation performed on any node results in deletion and new random initialization of the node and (for non-terminals) the entire attached sub-tree. The initialization is ensured to be consistent with the BNF template. Dependent on the depth of the mutating node within the tree, this kind of mutation may result in a small up to an overall manipulation of the individual.

Value ordered Mutation

The representation of engineering problems often involves a vast variety of numeric parameters. Goal oriented adjustment of these parameters during optimization is a powerful mechanism to dramatically improve overall optimization performance. Instead of replacing numerical values with completely new ones during mutation, the value ordered mutation incorporates the original value and modifies it gently to a larger or smaller value. The magnitude of this modification is controlled by three factors:

- A randomly chosen direction (increase /decrease value)
- A randomly chosen α-cut value between 0 and 1
- A Gaussian function, dependent on optimization parameters

Figure 4 shows an example of mutating the numerical value X=6.0. The parameter μ of the Gaussian distribution function adopts the original numerical value. The parameter σ depends on the product of a global optimization parameter σ for valued mutation and a corresponding BNF-node-specific variance factor σ. The randomly chosen mutation variables in this example are “increase” and “α-cut = 0.63”, which results in a mutated value of X=6.7.

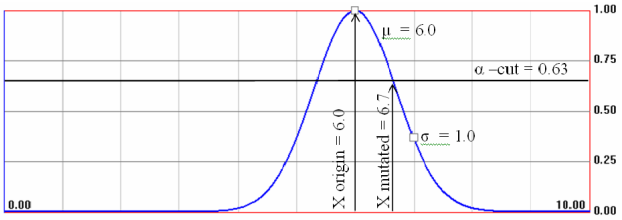

Figure 4. Value ordered mutation of numeric values, based on a Gaussian function.

Besides application to numerical values, value ordered mutation can also be applied to non-terminated symbols, if declared as being “ordered”. In this case a chosen alternative of a non terminated symbol will change to a probably adjacent one, while replacing a possibly assigned subtree by a randomly defined one. For example the symbol “NTSymbol2” in figure 5, which originally has a value of “B”, may mutate to “A” or “C”, but less likely to “D” or the least to “E”. The first two symbols in figure 5 would be mutated with equal results.

```
<NumericSymbol>          := [1;5;1]
<NTSymbol1> (ordered)    :=
"1"|"2"|"3"|"4"|"5"
<NTSymbol2> (ordered)    :=
"A"|"B"|"C"|"D"|"E"
```

Figure 5. value-range and ordered non–terminal symbols.

2.6 *Optimization parameters*

Dependent on the utilization of the GPCore, the described framework may be used in terms of genetic algorithms or more in the scope of evolutionary strategies, while the classification is fuzzy and mainly dependent on the choice of suitable optimization parameters. The most important parameters supported by the GPCore are

- Population size
- Fitness threshold (stop criterion)
- Generation count (alternative stop criterion)
- Selection (tournament or fitness proportionate selection)
- Selection power (impact of individual’s fitness to selection decision)
- Crossover rate (probability of crossover of a new breed individual)
- Cumulative mutation rate (probability of cumulative mutation)

- Value ordered mutation rate (probability of value ordered mutation)
- Value ordered power (global factor σ)

Most of these parameters can be dynamically adjusted during the optimization process, dependent on the actual optimization performance or based on pre-defined generation limits and transitions. This is especially efficient for optimization problems with random initialization. Dependent on the problem specific strategy, in many cases it is useful to adjust the parameters during the optimization process. In case of the described function approximation problem, the first phase uses a high crossover rate with medium mutation. With a growing number of generations a low crossover and a higher, but less aggressive mutation for "fine-tuning" has been chosen. In this case the optimization was performed much more effective with the generation-dependent parameter adjustment than with any other fixed parameter configuration which has been evaluated.

2.7 *Evaluation environment*

For evaluation of all GPCore functionalities, the GPCore Manager provides a simple but flexible built-in problem space covering mathematical function approximation. Main advantage of this is the flexible degree of complexity when setting up a BNF template representation for evaluation purposes. The test environment includes a powerful function parser to deliver a fitness value of an individual by testing against a given target function within a certain value range. The summarized mean squared error of the two functions is provided as fitness value. Figure 6 shows a slightly more complex function definition, which has been used for evaluation of the GPCore.

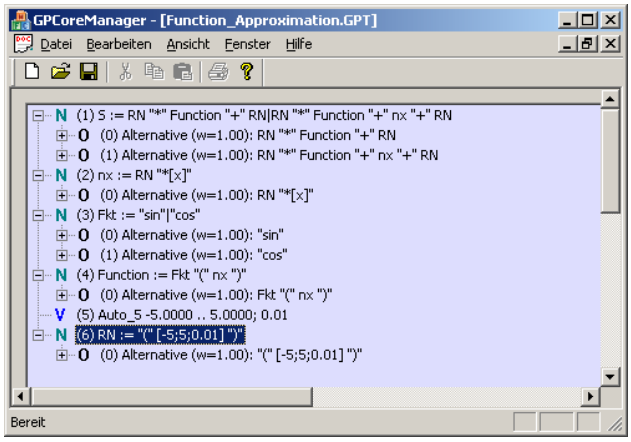

Figure 6. GPCore Manager - Template representation for evaluation.

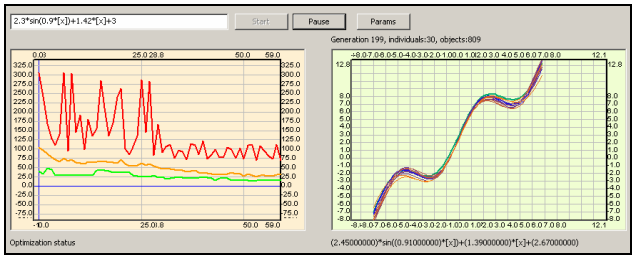

Figure 7. GPCore Manager – Progress of optimization during evaluation.

Figure 7 shows a running optimization task based on the template shown in figure 6. The diagram on the left shows the best, average and worst species in each generation, while the diagram on the right visualizes the target function (in green, defined on top) together with the 10 best individuals within the current generation. Below to the diagram on the right, the best so far solution is shown, which in this case is already very close to the desired solution. During the running task, the global parameters can be directly modified to verify their impact on the optimization progress.

3 EXAMPLE OF APPLICATION: VEHICLE IDENTIFICATION FROM BRIDGE MONITORING DATA

Load identification is the inverse problem of response prediction: Instead of the determination of a system's response due to a known loading situation the system's response is known from measurements and the load represents the unknown component. A common approach to solve the problem is the adaptation of an analytical model's load component by comparison of a prediction with a measured observation. The model's system component is assumed to be known a priori.

Since measurements can hardly be accomplished in all degrees of freedom, the adaptation is conducted on the basis of incomplete and limited information. Generally, a unique solution cannot be determined. The problems are said to be ill-posed. Traditional optimization procedures may end in local extrema and are little suitable to solve the problem.

The GPCore was integrated in an algorithm to perform load identification from bridge monitoring data. This algorithm – the Soft-WIM algorithm – was developed in order to obtain detailed information about vehicles that pass the monitored superstructure of a bridge. Soft-WIM is based on two encapsulated GP routines to analyze the recordings of two essential sensors. The sensors are installed in one cross section of the bridge's superstructure. Gross weights, velocities, axle loads as well as axle spacings of passing vehicles are determined. Single vehicles and corresponding attributes are identified from data recorded during the presence of one or multiple vehicles on the bridge at a given point of time.

3.1 *Fundamentals*

The signals of two sensors, which are installed in one cross section, are considered within two optimization kernels (Lubasch et al. 2005, Schnellenbach-Held et al. 2006). The analysis is based on numerically computed strains and deformations which are compared to measured values. The goal of optimization is described by the minimization of the mean squared error between computed strains resp. deformations and the corresponding measured data. A differentiation according to the kind of recorded structural response is drawn: One optimization kernel serves for the analysis of measured global structural responses, whereas the other analyses local responses. Global reactions of a bridge are recorded from sensors, which are located in the cross section to record

significant values while a vehicle crosses the bridge. Reactions of the bridge superstructure due to vehicle loads are global by this definition. Local reactions are obtained from sensors being placed close to acting forces and in consequence their recording is of short duration. The response of the bridge deck due to single wheel loads is considered as local.

3.2 *Analysis of global responses*

In comparison to local responses, the global responses of a bridge structure are of longer duration. The reaction of the superstructure due to a loading situation may be the result of one or multiple vehicles on the bridge. Measured data may represent one vehicle or the combination of several vehicles.

Figure 8 shows sample data of two articulated vehicles following each other in short distance. The data was obtained during the monitoring of a post-tensioned concrete bridge. Recorded strains as well as the results from the automated analysis are demonstrated.

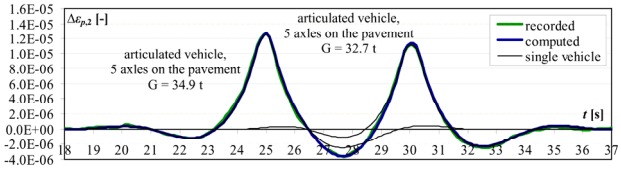

Figure 8. Analysis of global responses: Identification of vehicle properties.

To comply with the characteristics of global measurements, the data analysis is performed in time steps by considering data of a corresponding time interval. For every time interval a minimization of the mean squared error of computed and measured strains is performed within an evolutionary optimization routine. The population of individuals is described by vehicle combinations. After the analysis of a time interval a time step is performed. For the initialization of the new population the vehicles that were identified during the data analysis of the preceding time interval are incorporated. A vehicle that was object of optimization for a predefined number of time steps is assumed to describe the actual vehicle and gets transferred to the database containing the identified single vehicles.

3.3 *Analysis of local responses*

The local responses are of short duration. Accordingly, the data analysis is performed per vehicle. In contrast to the analysis of measured global responses, a proceeding in time steps and the consideration of time intervals is not necessary.

Figure 9 shows measured and computed strains for the second 5 axle articulated vehicle, which is shown in figure 8.

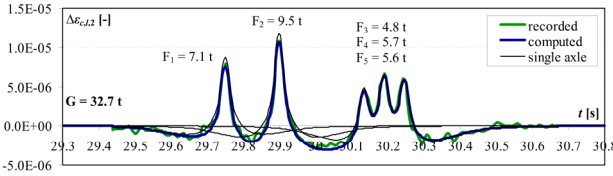

Figure 9. Analysis of local responses: Identification of axle properties.

The data analysis starts with the application of a neural network (NN) to the measured data. The NN was trained on the identification of a single vehicle's axles' times of occurrence. In an extensive evaluation with wide test data the NN has proven to identify axles with high reliability. The NN has shown very good results for the analysis of smeared sensor signals from light axles (e.g. triple axles of unloaded trailers).

At the beginning of the evolutionary optimization a phenotype is generated based on the results from the NN. The evaluation of the NN's performance has shown that the NN either identifies the exact number of axles or more axles than actually existent. A number of axles less than actually present has not been specified by the NN within the evaluation. Thus, in conjunction with estimated axle forces the generated phenotype based on the NN results already describes a quite good solution. The initialization of the first population of individuals is performed on basis of this phenotype. In order to generate the first population, copies of the corresponding genotype are manipulated slightly and assigned to the population. For the purpose of slight manipulation the mutation operator is used. According to common understanding (Eiben et al. 1998), for the present optimization task exploitation is emphasized over exploration. In this sense, the optimization process shows similarities to the classical ES approach: For the required exploitation, the mutation operation is underlined. It shall be noted that traditional local search techniques are inappropriate to solve the problem. Global search is still required to obtain the real number of vehicle axles.

Figure 10 shows the BNF definition for the analysis of the presented measured data (figure 9). Figure 11 demonstrates the corresponding phenotype represented in a derivation tree. Individuals are described by the vehicle's transversal distance q and the vehicle's axles. The optimization parameter q characterizes the distance of the right row of wheels to the sensor whose signal is analyzed. The consideration of this parameter within the optimization process is required since local responses are very sensitive to the exact location of acting loads.

```
S                    := <Q><AxlesReg>
<Q>                  := [-1.0;0.9;0.1]
<F>                  := [3.6;11.5;0.1]
<D>                  := [-0.20;0.20;0.01]
<AxlesReg>           := <Ax-
leReg1><AxleReg2><AxleReg3>
<AxleReg1>           := <F><Axles1>
<AxleReg2>           := <F><Axles2>
<AxleReg3>           := <F><Axles3>
<Axles1>             := <A1>
<Axles2>             := <A2>
<Axles3>             :=
<A3>|<A3><A3>|<A3><A3><A3>|<A3><A3><A3><A3>
<A1>                 := <D><T1>
<A2>                 := <D><T2>
```

```
<A3>                    := <D><T3>
<T1>                    := [29.724;29.784;0.002]
<T2>                    := [29.872;29.932;0.002]
<T3>                    := [30.108;30.276;0.002]
```

Figure 10. Analysis of local responses: Sample BNF definition for the GP-analysis.

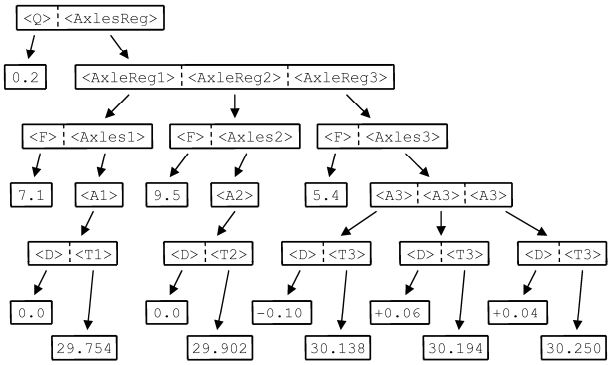

Figure 11. Analysis of local responses: Sample derivation tree.

4 CONCLUSIONS

A generic framework for genetic programming has been presented and introduced in this paper. The differences between genetic algorithms and genetic programming in terms of representation space, genetic operators and overall usability have been addressed. Simple but effective examples demonstrated the use and capabilities of the system to mathematical problems, which can be easily transformed to a vast variety of real world optimization environments.

Furthermore, a practical approach to determine operational loads of instrumented bridges was presented. The developed and implemented Soft-WIM algorithm serves for the identification of single vehicles that pass an instrumented bridge. The bases of the algorithm are two encapsulated evolutionary optimization kernels based on the previously described framework to analyze recorded global and local structural responses. Gross vehicle weights, vehicle velocities, axle loads and axle configurations are obtained. The presented approach and the results demonstrate that structural health monitoring in conjunction with adequate mechanisms can successfully support the acquisition of additional information. By means of appropriate analyses, measured data, which may be obtained during structural health monitoring, can be explored for valuable information beyond the monitored structure.

REFERENCES

Eiben, A.E., Schippers, C.A. (1998). On evolutionary exploration and exploitation. Fundamenta Informaticae, 35, pp. 35-50.

Geyer-Schulz, A. (1995). Fuzzy Rule-Based Expert-Systems and Genetic Machine Learning. Physica-Verlag.

Holland, J.H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: The University of Michigan Press.

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

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

Lubasch P., Freischlad M., Schnellenbach-Held M., Buschmeyer W. (2006). Knowledge Discovery in Bridge Monitoring Data: A Soft Computing Approach. In I.F.C. Smith (Ed.): Intelligent Computing In Engineering and Architecture, Lecture Notes in Artificial Intelligence, Springer Verlag.

Schnellenbach-Held, M., Lubasch, P., Buschmeyer, W. (2006). Erfassung der Verkehrsbelastung – die Brücke, die zählt. Festkolloquium zum 50-jährigen Bestehen des Instituts für Massivbau, Leibniz Universität Hannover (to be published).

Spears, W.M., De Jong, K.A., Bäck, T., Fogel, D.B., De Garis, H. (1993). An Overview of Evolutionary Computation. Proceedings of the 6th European Conference on Machine Learning (ECML '93), pp. 442-459, Springer.

Whitley, D. (2001).An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls. Information and Software Technology, 43(14), pp. 817-831.