Development of problem–specific Evolutionary Algorithms

Alexander Leonhardi[1], Wolfgang Reissenberger[1], Tim Schmelmer[2], Karsten Weicker[3], and Nicole Weicker[1]

[1] Universität Stuttgart, Fakultät Informatik, Germany,
email: {leonhaar,reissenb,weicker}@informatik.uni-stuttgart.de
[2] University of Kansas, Department of Electrical Engineering and Computer Science,
email: tim_schmelmer@hotmail.com
[3] Universität Tübingen, Wilhelm–Schickard–Institut für Informatik, Germany,
email: weicker@informatik.uni-tuebingen.de

Abstract. It is a broadly accepted fact that evolutionary algorithms (EA) have to be developed problem–specifically. Usually this is based on experience and experiments. Though, most EA environments are not suited for such an approach. Therefore, this paper proposes a few basic concepts which should be supplied by modern EA simulators in order to serve as a toolkit for the development of such algorithms.

1 Introduction

Theoretical work as well as practical experience demonstrate the importance to progress from fixed, rigid schemes of evolutionary algorithms (EA) towards a problem–specific processing of optimization problems. Since the "No Free Lunch" theorem [WM97] proves that there is no algorithm which performs better than any other algorithm for all kinds of possible problems, it is useless to judge an algorithm irrespectively of the optimization problem. Therefore, it is necessary to find a suitable algorithm for each problem. Experience has shown that the adaptation of a problem to standard evolutionary algorithms is frequently a difficult task since a fixed representation of the search space is required (e.g. bit strings for genetic algorithms or real valued tuples for evolution strategies). Very often it is not even possible to fit the problem in the structure of the standard algorithm's search space (e.g. structural optimization [Kaw96], optimization of facility layout [AAD97], or job scheduling [FRC93]). Coding complex data structures by simple lists of bits or real values leads to the problem that there is often no one–to–one relation between these lists and the problem instances. Hence, problem knowledge is either necessary in repair operators to deal with invalid solutions or in special operators tailored to the problem.

Most existing EA simulators (e.g. [Gre90,VBT91]) offer standard algorithms like genetic algorithms or evolution strategies with invariable representations. There are only few simulation environments for evolutionary algorithms with complex high–level data structures like matrices, trees, or other combined data

types (e.g. GAME [DKF93]). For instance, such data structures are used in [MHI96,BMK96].

In addition, there are numerous new flexible ideas: e.g. adaptation of evolutionary operators is presented in [SS94,Her96], the programming language RPL2 ([SR94]) uses models of structured populations, and several hybrid algorithms as combinations of standard operators from different EA are mentioned in [WGM94,IdGS94,DPT96]. Usually, those concepts are not supported by standard systems in general. Therefore, there is no possibility to test those algorithms for new applications in an uniform environment.

As a consequence, a new extensive conceptual framework for the design of evolutionary algorithms is proposed. It supports arbitrary problem representations, construction and testing of operators, and comparison of different user–defined algorithms. All well–known fields of evolutionary algorithms like genetic algorithms [Hol75,Gol89], evolution strategies [BHS91], genetic programming [Koz92], evolutionary programming [Fog92] as well as most other imaginable algorithms are possible in this framework.

The essential concepts for EA simulators are presented in Section 2. Section 3 discusses how these concepts are realized in a prototype GENOM. The ability and power of such a system is exemplified in Section 4. The paper concludes with remarks and aspects of future work in Section 5.

2 Concepts

A system for the development of problem–specific evolutionary algorithms should enable the user to find suitable algorithms for a concrete problem without restricting system constraints. Its purpose is to clarify which algorithm should be implemented efficiently.

2.1 Distinction Phenotype – Genotype

The first important concept is the strict separation of problem formulation and evolutionary method. If different methods should be tested and compared for a problem, it is indispensable to define the problem independently from the considered method in order to have a common platform for the evaluation of the methods.

Such separation results in two completely different views of an individual. On the one hand, there is the phenotypic representation for the problem in order to evaluate the individual by the fitness function. On the other hand, the algorithms only see an encoded individual – a genotype – to which the evolutionary operators are applied. Decoding – and if necessary encoding – functions switch between those views of an individual (compare Figure 1). Such a biologically oriented approach was already presented in [Ban94,TGF96].

Several advantages result from this two–sided view: First, it is possible to adapt the phenotypic representation to the problem independently from the EA, e.g. permutations may be used for a travelling salesperson problem although an

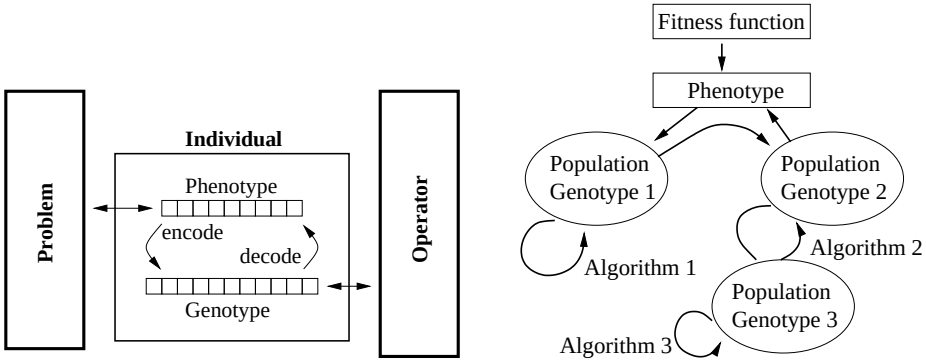

Fig. 1. Separation of problem and method (left) and interplay of populations with different genotypes in one experiment (right)

evolution strategy may be used to search for the optimum. In order to take advantage of this approach, arbitrary, universal data types for the phenotypic representation are demanded. They result in straight and simple fitness functions not restricted by method–depending representations. Second, the encapsulation of the fitness function means the fitness function needs only to be implemented once, independent of the number of methods to be tested.

On the method side, the genotypic representation should be as powerful as the phenotypic representation – they might be identical. Standard algorithms are possible with a decoding function from the standard genotypic representation to the phenotype. Finally, very sophisticated codings may be used, where e.g. the decoding function uses problem knowledge for genetic repair. An encoding function is only necessary if either existing good solutions from the problem side should be used for initialization in the method or several methods with different genotypes should be combined.

2.2 Modular operators

Another concept is the separation of operators from the representation in order to support a high re–usability and universal use of the operators. This may be reached by parameterization of the operators. Predefined standard operators should be supported as well as user–defined operators which are adjusted to specific user–defined genotypes (see Section 2.1). A modular composition allows to reuse previously defined operators. A high level of abstraction should enable the user to concentrate on the algorithms and operators and protect him from implementation details.

In addition to the toolbox character of the operators, complete evolutionary algorithms may be interpreted as operators too and used accordingly. In particular, this takes effect together with the experiment concept discussed in the next section.

2.3 Experiments

In order to combine different methods, a new concept is necessary in which the method does not represent the top instance. For that purpose, *experiments* are defined, uniting several populations with different methods and different genotypes for a problem. This enables migration between populations with identical genotype, e.g. in structural populations ([SR94]) and all parallel models of evolutionary algorithms. In addition, individuals may be exchanged among populations with different genotypes using the encoding and decoding functions. That is how methods, e.g. genetic algorithm and evolution strategy, may be combined which are incomparable otherwise (compare Figure 1). This experiment concept enables new adapted evolutionary concepts, e.g. with several operators, hybrid methods, or multiply nested loops. A system as open as possible is desired.

3 Genom

The proposed concepts have been implemented in a first prototype of a system GENOM (GENOM is an ENvironment for Optimization Methods). Altogether, GENOM is a system, which allows gradually getting into the development and design of problem–specific evolutionary algorithms. From the adaption of standard algorithms loaded from libraries to the definition of new genotypes and operators all gradations of algorithm development are possible. From that point of view, it is an open system in which great importance was attached to the fact not to exclude any conceivable adaption. In this section, the realization of the concepts in GENOM is explained.

3.1 Phenotype and Genotype

In order to obtain a maximum of flexibility, phenotype and genotype can be arbitrary data types. Both types are constructed in the same manner. They are constructed from types like real values, bits, integers, or permutations, called *atoms*, which are arranged into arbitrary structures like e.g. lists, matrices, tuples, or trees, called *cells*. On the top level, these cells are arranged into lists of arbitrary but fixed length. This enables standardization of coding and decoding. Figure 2 shows an example of an individual containing a pair cell, a matrix cell and a tree cell.

The transformation between phenotype and genotype is defined by a pair of functions: the *coding* and *decoding* function. The fixed structure of pheno- and genotype as a list of cells makes it feasible to construct the coding and decoding function according to a fixed scheme. Those schemes are based on so–called *elementary coding schemes*, which are pairs of functions describing the bidirectional transformation between tuples of cells, e.g. between a pair of real values and a list of bits. A combination of such elementary coding schemes together with a mapping of the genotype and phenotype cells to the input and output positions form the *coding schemes* (see also Figure 2). By such a mapping, the coding and decoding functions are defined simultaneously through the functions of the elementary coding schemes.

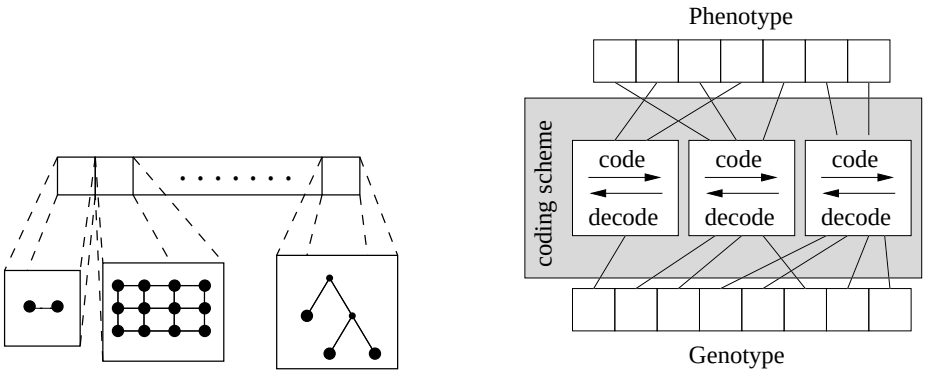

Fig. 2. Example of an individual (left) and structure of coding schemes (right)

3.2 Operators and experiments

Experimental frameworks like GENOM are intended to experiment with algorithms in order to find appropriate data structures and evolutionary algorithms to solve a given problem. Hence *experiments* contain a description of the problem (phenotype and fitness function), a collection of populations, and a top–level operator. Each population contains the genotype of its individuals and a coding scheme describing the conversion between the genotype and phenotype. Using these coding schemes, individuals can be properly migrated between populations.

Upon these populations, arbitrary methods can be defined. The methods are classified according to their scope. On the lower level, operators transform cells into cells or individuals into individuals (e.g. mutation, crossover). On the higher level, algorithms and methods work on a single population (e.g. a genetic algorithm or an evolution strategy) or even on multiple populations like a migration operator. All these methods can be parameterized and stored in libraries to achieve a maximum of flexibility and good re–usability.

Experimenting with evolutionary algorithms requires a tailored language for their definition. LEA$^+$ (Language for Evolutionary Algorithms), the definition language used in GENOM, is such a language, where experiments, individuals, coding schemes, and operators can be defined.

4 Examples

The following examples are intended to clarify how the different concepts interlock in reality. Moreover, the potential of such a system is demonstrated. The examples were realized with GENOM.

4.1 Example 1: Open–Shop Scheduling problem (OSSP)

The first example is an OSSP, which deals with the problem of generating an efficient production schedule. A number of jobs and a number of machines to process the jobs are given. Each job is composed of a set of tasks which must each be executed on a different machine for a task–specific amount of time. Neither the order of jobs, nor the order of tasks within each job are fixed and can therefore vary between two valid schedules. The phenotype is defined as a list of cells, where each cell contains the timetable for one machine.

Three experiments are based on this problem. The first experiment uses permutations to represent the schedule, the second experiment applies a genetic algorithm with a standard binary representation, while the last experiment represents the schedule by a matrix. All three experiments use the standard structure of evolutionary algorithms (recombine, mutate, select). Each experiment defines its own operators `Recombine` and `Mutate`, while all experiments use the same `Select` operator (Best Select).

This example demonstrates that the fitness function needs only to be defined once independently from the three different genotypes and algorithms. As a consequence, the results can be compared very easily since they all use the same phenotype and the same fitness function.

1. **Permutations**
   The outline for this experiment is described in [FRC93]. The genotype is a permutation of pairs (i, j), where i is the job's number and j the number of the task. It is implemented as a single cell containing the permutation. The `Recombine` operator takes two permutations, selects some random positions from the first one, and fills in the empty positions with the missing elements in consideration of the order in the second permutation. The `Mutate` operator is the 2-opt operator (e.g. [LK73]), which takes a subsequence of the permutation and reverses it. In addition, the decoding function optimizes an individual by moving elements of the permutation to the earliest possible slot. This is an example for the very powerful mechanism of coding schemes.
2. **Binary Coding**
   The coding of this experiment is based on the coding to a permutation as used in the experiment described above. Afterwards the permutation is coded into a binary string, which allows standard operators used in genetic algorithms. Hence, the genotype is a list of cells, where each cell contains one single bit.
3. **Matrix Representation**
   In this experiment, the genotype is a single cell containing a matrix. The first column of this matrix contains the job's number. The rest of each row forms a permutation, which defines the order in which the tasks for the respective job have to be executed, e.g. in the left individual of Table 1, job 4 executes its tasks in the order $1, 2, 3, 4, 5$. The order of the rows defines a priority, i.e. if there exists a conflict, which team should be granted execution first, the team from the upper row is preferred.

individual 1					
4	1	2	3	4	5
2	2	3	4	5	1
1	3	4	5	1	2
5	4	5	1	2	3
3	5	1	2	3	4

individual 2					
4	5	4	3	2	1
3	1	5	4	3	2
5	2	1	5	4	3
1	3	2	1	5	4
2	4	3	2	1	5

selection scheme				
*		*	*	
	*	*		*
*	*			*
		*		

result					
4	1	5	3	4	2
2	2	3	4	5	1
1	3	2	1	5	4
5	4	5	2	1	3
3	1	5	2	4	3

Table 1. Recombination of two matrix individuals

The recombination operator `Recombine` works similarly as the respective operator for the permutation coding. Table 1 shows a selection matrix and the result, when the operator is applied to the two individuals. The `Mutate` may switch two rows or use the 2-opt operator for permutations working on single rows, leaving the first column unmodified. This is an example for evolutionary operators tailored to the problem structure.

This example shows how the same problem can be coded into different genotypes although the same standard algorithm is used. Only the operators used in the algorithm have to be specific to the genotype.

4.2 Example 2: Travelling Salesperson Problem (TSP)

This example shows the flexibility of the experiment concept and the ability to combine different codings. Two optimization algorithms, a genetic and a threshold algorithm, are combined to form a new evolutionary algorithm. The problem to be optimized is a TSP problem and is represented by a permutation of the cities' indices. The length of the round trip through these cities is the fitness function to be minimized. Three experiments are applied to this problem.

1. **Threshold Algorithm (TA)**
   The threshold algorithm works directly on the permutation and uses the identity coding. The mutation of the individuals is done with the 2-opt operator.
2. **Genetic Algorithm (GA)**
   The second experiment is a classical genetic algorithm. Hence, the permutation has to be coded into a list of bits which is done in such a way that decoding always results in a valid permutation. A three–point–crossover is used for the recombination of individuals and a random inversion of bits for mutation. The individuals of the next generation are selected using an elitist proportional selection operator.
3. **Combined Algorithm (GA + TA)**
   This experiment contains a genetic algorithm interacting with a threshold algorithm. After every 20 generations, a few of its individuals are optimized with the threshold algorithm. Their results are migrated back and the 20

best individuals are selected for the next generation of the genetic algorithm. Here, two conceptually completely different algorithms are combined.

Figure 3 shows the fitness of the best individual displayed over the number of evaluated individuals for a typical example of the experiments described above. To show the effects more clearly, only the threshold algorithm with the best result is considered in the curve of the combined algorithm. It corresponds to the peaks in the diagram after each twentieth generation of the genetic algorithm. Although this paper is not concerned with comparing the algorithms or finding optimal parameter values, the experiments indicate that a combination of methods can do better than each of them alone.

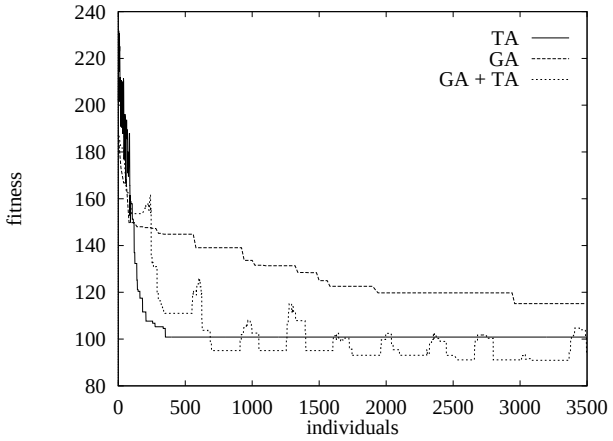

Fig. 3. The fitness of the best individual for the threshold algorithm, the genetic algorithm, and for a combination of both.

5 Remarks and Future Work

Summarizing, the proposed concepts define a very powerful tool for the design of evolutionary algorithms. All standard representations and methods (compare Chapter C1 and C3 in [BFM97]) are supported by an unifying foundation. The separation of problem formulation, coding, and algorithms has proved to be very useful. Once a problem, coding, or algorithm has been defined it can be used in various combinations and experiments (e.g. the genetic algorithm in the experiments above). Further flexibility in adapting an algorithm to a problem is facilitated by the powerful data structures for geno- and phenotype.

Nevertheless, there is still conceptual work necessary in order to integrate more general arbitrary methods to evaluate and compare experiments with different algorithms. A special topic for future work will be a general concept for the

handling of constraints. So far, only repair functions are possible, e.g. as part of the decoding function ([VM97]). Finally, polymorphic and object–oriented models should be integrated into the coding concept. Furthermore, novel concepts and developments are to be expected for a general standardized EA simulation environment. This work is intended as a first step in that direction.

The purpose of GENOM is to provide a standard platform for development and evaluation of suitable and adapted representations and algorithms. For practical use, those algorithms have to be re–implemented in the application environment. A standard library of operators, codings, and algorithms is on the way to be developed. Additionally, the integration of an existing prototype for defining and running evolutionary algorithms in a distributed environment with communication channels is planned.

The presented concepts and the prototype of GENOM have been developed in student's team–projects and diploma theses ([AJK+95,JW95,GLS97,Leo97]). For more documentation and information on the availability of GENOM see http://www.informatik.uni-stuttgart.de/ifi/fk/genom/ .

References

[AAD97] K. Alicke, D. Arnold, and V. Dörrsam. (R)evolution in layout–planning – a new hybrid genetic algorithm to generate optimal aisles in a ficility layout. In H.-J. Zimmermann, editor, *Eufit 97 – 5th European Congress on Intelligent Techniques and Soft Computing*, pages 788–793, Aachen, 1997. Verlag Mainz, Wissenschaftsverlag, Aachen.

[AJK+95] F. Amos, K. Jung, B. Kawetzki, W. Kuhn, O. Pertler, R. Reißing, and M. Schaal. Endbericht der Projektgruppe Genetische Algorithmen. Technical Report FK95/1, University of Stuttgart, Institute of Computer Science, Dept. Formal Concepts, 1995. german.

[Ban94] W. Banzhaf. Genotype-phenotype-mapping and neutral variation — a case study in genetic programming. In Davidor et al. [DSM94].

[BB91] R.K. Belew and L.B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms – ICGA V*, San Mateo, California, 1991. Morgan Kaufmann.

[BFM97] T. Bäck, D.B. Fogel, and Z. Michalewiez, editors. *Handbook of Evolutionary Computation*. IOP Puplishing Ltd and Oxford University Press, 1997.

[BHS91] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In Belew and Booker [BB91].

[BMK96] C. Bierwirth, D. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In Voigt et al. [VERS96].

[DKF93] Laura Dekker, Jason Kingdon, and J. R. Filho. *GAME Version 2.01, User's Manual*. University College London, 1993.

[DPT96] D. Duvivier, Ph. Preux, and E.-G. Talbi. Climbing up NP–hard hills. In Voigt et al. [VERS96].

[DSM94] Y. Davidor, H.-P. Schwefel, and R. Maenner, editors. *Parallel Problem Solving from Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, Berlin, 1994. Springer-Verlag.

[Fog92] D.B. Fogel. An analysis of evolutionary programming. In D.B. Fogel and J. W. Atmar, editors, *Proceedings of the First annual Conference on Evolutionary Programming*, La Jolla, 1992. Evolutionary Programming Society.

[FRC93] H.-L. Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling and open-shop scheduling problems. In *Proceedings of the Fifth Int. Conf. on Genetic Algorithms*, pages 375–382. Morgan Kaufmann Publishers, 1993.

[GLS97] M. Großmann, A. Leonhardi, and T. Schmidt. Abschlußbericht der Projektgruppe Evolutionäre Algorithmen. Technical Report 2, University of Stuttgart, Institute of Computer Science, 1997. german.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, 1989.

[Gre90] J.J. Grefenstette. *A User's Guide to GENESIS, Version 5.0*, 1990.

[Her96] M. Herdy. Evolution strategies with subjective selection. In Voigt et al. [VERS96].

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[IdGS94] H. Iba, H. de Garis, and T. Sato. Genetic programming with local hill-climbing. In Davidor et al. [DSM94].

[JW95] K. Jung and N. Weicker. Funktionale Spezifikation des Software–Tools EAGLE. Technical Report FK95/2, University of Stuttgart, Institute of Computer Science, Dept. Formal Concepts, 1995. german.

[Kaw96] B. Kawetzki. Topologieoptimierung diskreter Tragwerke mittels Evolutionsstrategien am Beispiel ebener Fachwerke. Master's thesis, University of Stuttgart, 1996. german.

[Koz92] J.R. Koza. *Genetic Programming*. MIT Press, 1992.

[Leo97] A. Leonhardi. Eine Beschreibungssprache für Evolutionäre Algorithmen. Master's thesis, University of Stuttgart, Institute of Computer Science, 1997. german.

[LK73] S. Lin and B. Kernighan. An efficient heuristic procedure for the traveling salesman problem. *Operations Res.*, 21:498–516, 1973.

[MHI96] M. McIlhagga, P. Husbands, and R. Ives. A comparison of search techniques on a wing–box optimisation problem. In Voigt et al. [VERS96].

[SR94] P.D. Surry and N.J. Radcliffe. RPL2: A language and parallel framework for evolutionary computing. In Davidor et al. [DSM94].

[SS94] M. Sebag and M. Schoenauer. Controlling crossover through inductive learning. In Davidor et al. [DSM94].

[TGF96] S. Tsutsui, A. Ghosh, and Y. Fujimoto. A robust solution searching scheme in genetic search. In Voigt et al. [VERS96].

[VBT91] H.-M. Voigt, J. Born, and J. Treptow. *The Evolution Machine, Manual, Version 2.1*. Institute for Informatics and Computing Techniques, Berlin, 1991.

[VERS96] H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature – PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, Berlin, 1996. Springer-Verlag.

[VM97] D. Vigo and V. Maniezzo. A genetic/tabu thresholding hybrid algorithm for the process allocation problem. *Journal of Heuristics*, 3(2):91–110, 1997.

[WGM94] D. Whitley, V.S. Gordon, and K. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Davidor et al. [DSM94].

[WM97] D.H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions On Evolutionary Computation*, 1(1):67–82, April 1997.