



Project no. FP6-NEST-2003-1-12789  
ESIGNET  
Evolving Cell Signalling Networks in Silico

Specific Targeted Research Project  
Sixth Framework Programme Priority

Deliverable number 5.1  
Fast Evolutionary Algorithms – Report

Due date of deliverable: May 2006  
Actual submission date: May 2006

Start date of project: 2005-09-01

Duration: 36 months

Friedrich Schiller University Jena

Revision: final

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

## **Abstract**

The overall goal of the ESIGNET project is to study the computational properties of cell signalling networks (CSN) by evolving them using methods from evolutionary computation, and to re-apply this understanding in developing new ways to model and predict real CSNs.

As a research area, the field of Evolutionary Algorithms has been under consideration for decades, and a variety of different branches and flavours has been developed. To lay the foundation for the artificial evolution of CSNs, workpackage 5 compares different approaches and identifies several combinations suitable for the project. In this report, a short introduction into the field of Evolutionary Algorithms is given, followed by an outline of an Evolutionary Algorithm capable of evolving artificial CSNs. To improve and speed up the methods, parallel implementations are discussed, followed by a brief section with conclusions for future work.

## **Contents**

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Genetic Algorithms</b>	<b>3</b>
<b>3</b>	<b>Evolution Strategies</b>	<b>3</b>
<b>4</b>	<b>Genetic Programming</b>	<b>4</b>
<b>5</b>	<b>Choice of Methods</b>	<b>4</b>
<b>6</b>	<b>Parallel Implementations</b>	<b>6</b>
<b>7</b>	<b>Conclusion</b>	<b>7</b>

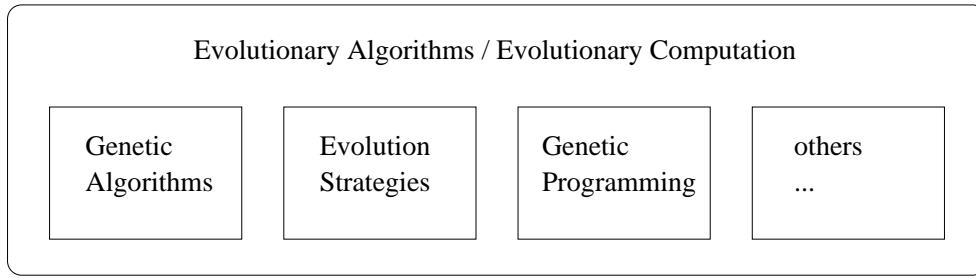


Figure 1: Broad structuring of the field of Evolutionary Algorithms.

## 1 Introduction

The ESIGNET workpackage (WP) 5 aims at the development of efficient Evolutionary Algorithms capable of evolving CSNs that perform prespecified computations. In fulfilment of this, WP5 is thought to focus on following steps:

- Compare existing Evolutionary Algorithms in terms of their suitability to evolve CSNs.
- Develop (choose) fast algorithms to evolve CSNs.
- Produce software library that can be used to calculate the fitness of a candidate CSN given target computational properties.

Objectives within WP5 are specified by three deliverables:

**D5.1.** A report describing the choice of methods.

**D5.2.** A piece of software that efficiently specifies the fitness of candidate solutions given a suitable fitness-function.

**D5.3.** Publication in peer-reviewed journal and/or conference.

Difficult optimisation problems for which globally optimal solutions can usually not be found abound in any technical and scientific discipline. When good approximate solutions are sought, Evolutionary Algorithms (EAs) are proven to be a powerful tool for finding such solutions in a complex and high-dimensional search space. These employ a population of candidate solutions (called individuals) which can be mutated, recombined, reproduced or selected against. Originally inspired by the process of biological evolution, this field has developed a dynamic of its own, growing into a diversity of subtopics such as Genetic Algorithms, Evolution Strategies, and Genetic Programming. In the ESIGNET project, we want to make use of this knowledge, take it back to biology, and gain new insights into the evolution and function of CSNs.

This report is organised as follows: After an introduction into workpackage 5, a short review on Genetic Algorithms, Evolution Strategies and Genetic Programming - the main branches of EA research (fig. 1) - is given, loosely based on [8]. For each, the typical representations of solution candidates as well as the main evolutionary operators are described. In section 5, ideas on the combination of standard methods are developed and evaluated with respect to the envisioned evolution of CSNs. Due to the inherent parallelism in population-based methods, EAs lend themselves well to implementation on parallel computing cluster, which is briefly discussed in section 6. The report finishes with a few concluding remarks.

## 2 Genetic Algorithms

The most widely known form of Evolutionary Algorithms are Genetic Algorithms (GAs), pioneered by Holland [3, 4]. Primarily conceived to deal with bitstring-coded solution candidates, they can be distinguished from other EA approaches by their probabilistic selection of parents for the next generation. Heavy emphasis is placed on recombination as the main evolutionary operator, while mutation is used at a low rate to introduce local genetic variations. For bitstrings, mutation can be implemented as the inversion of one bit, while a  $k$ -point crossover is created by aligning the parents strings, breaking them at  $k$  points, choosing one from each pair and recombining them into a new string.

Most theoretical work on GAs is based on the *schema theorem* [4], which states that in bitstring-coded GAs, short substrings that are highly fit will quickly spread through the population. Therefore, the optimisation does not have to search through all possible bitstrings, but rather through combinations of schemes. However, the consequences of this theorem as well as its application to problems with non-bitstring solution candidates are still a heavily debated topic.

Over time, GA implementations for solution representations different from bitstrings, e.g. for vectors of real numbers, have been developed. While mutation on real numbers can be canonically designed as the addition of a random value, devising a real-valued crossover operator is not straightforward. The choice for a specific operator depends on the problem one wants to solve, but mostly used are arithmetic recombinations between corresponding values in the vectors.

## 3 Evolution Strategies

This flavour of Evolutionary Algorithms (see [2] for a comprehensive review) usually deals with real-valued vectors  $x = (x_1, \dots, x_k)$  as solution candidates, taken from some interval  $G = [lb_1, ub_1] \times \dots \times [lb_k, ub_k] \subset \mathbb{R}^k$ . Therefore, Evolution Strategies (ES) are a special form of numeric optimisation that utilises ideas gained from biological evolution. The major difference to the procedure in GAs is that ES use a deterministic selection called elitist, in which only the best individuals of a population survive the selection step and are selected as parents for the next generation. The usual terminology is that from  $\mu$  parents  $\lambda$  offspring are produced. In  $(\mu, \lambda, \rho)$ -ES, the new generation (of size  $\mu$ ) is selected solely from the  $\lambda$  offspring, whereas in  $(\mu + \lambda, \rho)$ -ES, the new generation is selected from both parents and offspring ( $\rho$  denotes the number of offspring generated via recombination, the rest is generated using mutation).

Evolution Strategies employ mutation and recombination as evolutionary operators. Since the solution candidates in ES consist of real-valued vectors, mutation is realised by the addition of a random value  $u_i$  to the component  $x_i$ , usually taken from a Gaussian distribution with density function

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2\sigma^2}x^2}.$$

Theoretical work in the field of ES has led to various adaptation strategies for the mutation strength ( $\sigma$  for Gaussian mutations), the most popular of which is the 1/5 success rule. This rule states that on average, 1/5 of all mutations should have an advantageous effect on the fitness. If the success rate is higher than 1/5, it is assumed that the population is still far from the optimal solution and the mutation strength can be increased. When the success rate drops below 1/5, the mutation strength is decreased to allow local search around the optimum. This rule works well on unimodal optimisation problems, but its effect on multimodal fitness landscapes are rather mixed.

Self-adaptation is a more flexible adaptation approach, in which each solution candidate has a set of mutation strength parameters  $\sigma_i$ , one for each element in the vector  $x$ . These strategy parameters are themselves subject to evolution, so that solution candidates with well-fitted mutation strength will progress faster towards the optimum and thus spread through the population. An example of a recent ES implementation using self-adaptation is given in [7]. The success of ES on real-valued problems relies mainly on its adaptation strategies. However, this implies that ES cannot be used for optimisation in cases in which the mutation strength cannot easily be controlled.

## 4 Genetic Programming

As the name implies, Genetic Programming (GP, for a detailed account see [6, 1]) deals with the automated design of computer programs in an evolutionary process. In contrast to GA and ES approaches, the size of the solution candidates is not fixed in advance, but rather determined through mutations and selection. In its most widely used form, the computer programs under evolution are represented as syntax trees, in which leaves contain values and internal nodes contain commands or mathematical functions. Besides these, a variety of different program representations has been considered, from assembler languages to graph representations. The latter is especially suited to biochemical networks, since these can be understood as graphs with two types of nodes, molecular species and reactions (or as hypergraphs, in which multiple species are connected via reactions).

In a syntax-tree based GP setup, the recombination operator can be easily implemented by exchanging subtrees of two parents to obtain two offspring individuals. Mutation can be achieved by two operations: the replacement of a subtree by a randomly generated new tree, or the replacement of a single node by a new node. Recombination within one individual, i.e. the exchange of two subtrees, can be seen as another form of mutation. Analog to subprograms and functions in manual software engineering, the technique of automatically defined functions (ADF) evolves a set of separate trees simultaneously to the main tree. These are represented by special nodes and can be plugged into the main tree of the program.

In contrast to the binary problems in GA and the numeric optimisations in ES, most mutations and recombinations on programs in GP will have a deleterious effect. To counter this, a small part of the population is copied to the next generation without modification, so that good solutions are not lost again. Additionally, the average population sizes used in GP are much larger, which is also due to the belief that an initially diverse population is needed for a successful GP run. Typical population sizes used in GP are between 500 and 10000 individuals. The major selection mechanism used in GP research is tournament selection, in which  $q$  individuals are randomly selected from the population, and the winner is chosen for the next generation, following recombination or mutation.

## 5 Choice of Methods

The last three sections gave a brief introduction into some of the main categories of Evolutionary Algorithms. This list is by no means exhaustive, but is meant to highlight a few of the main ideas in the field. As in any discipline, standard methods can only be applied for certain categories of problems and need to be modified for others. When new problems have to be tackled, new variants of EAs are developed, which perform better than those mentioned above for the specific

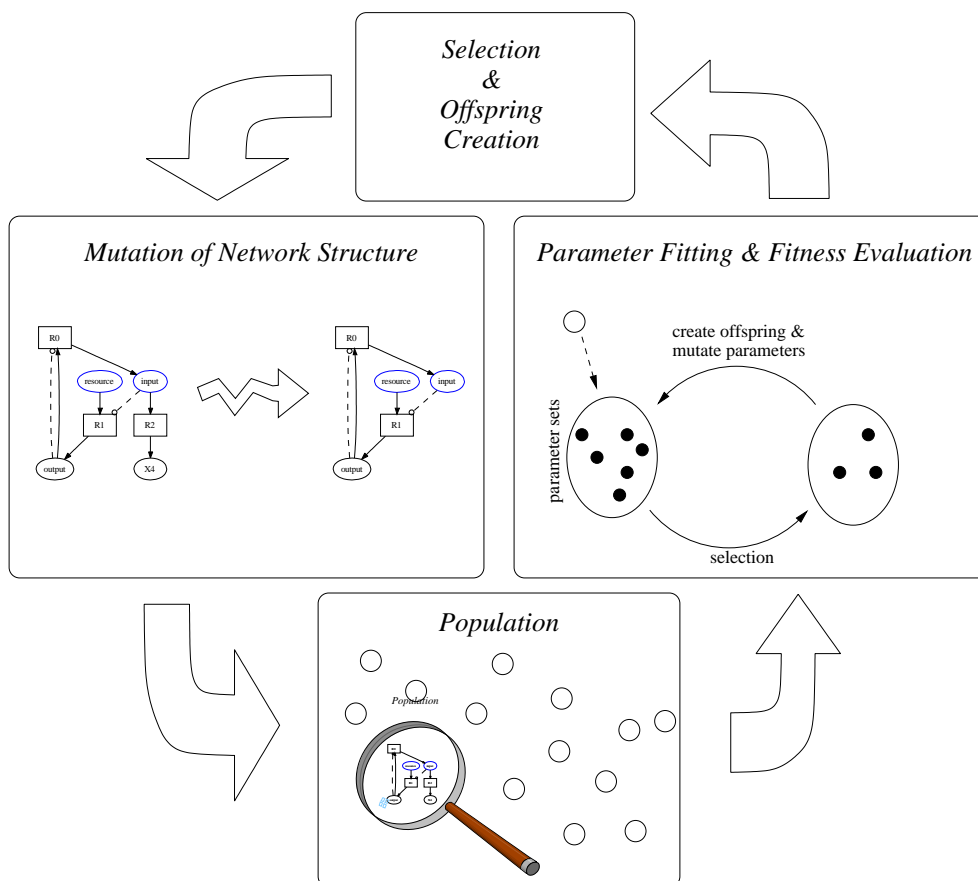


Figure 2: Illustration of the dual-layer Evolutionary Algorithm. The outer loop optimises the network structure, while the inner loop (inside the fitness evaluation) fits the network parameters to the desired behaviour.

problem under study. However, it can be observed that all EAs follow a similar scheme: they consist of a population, to which variation operators (mutation, recombination) and selection operators are applied. Following this scheme, we will now briefly sketch our algorithm of choice, which is a combination of different ideas from EA research.

In the evolution of cell signalling networks, different levels have to be considered. The highest is the network topology, containing the participating proteins and their interactions. On an intermediate level, the reaction kinetics contain information about the type of interaction, while the lowest level consists of the exact numeric parameters. In our approach, we have chosen to apply different strategies at each level, in the hope of making maximal use of the advantages of each of the aforementioned concepts. Due to the short period of investigation so far, we do not yet have concluding results on the effect of that decision.

Our approach understands the evolution of the network topology (i.e. species and their connections via reactions) as a form of Genetic Programming, since CSNs are representable as programs that perform computations depending on their initial values. Different kinetic laws of reactions constitute different interactions, thus integrating the second level with the first. In contrast to classical GP programs, CSNs cannot easily be represented as trees, as feedback and circles are integral components in many of them. Therefore, graph-based GP is the current algorithm of choice for the evolution of a CSN topology. Apart from its topology, a CSN description also

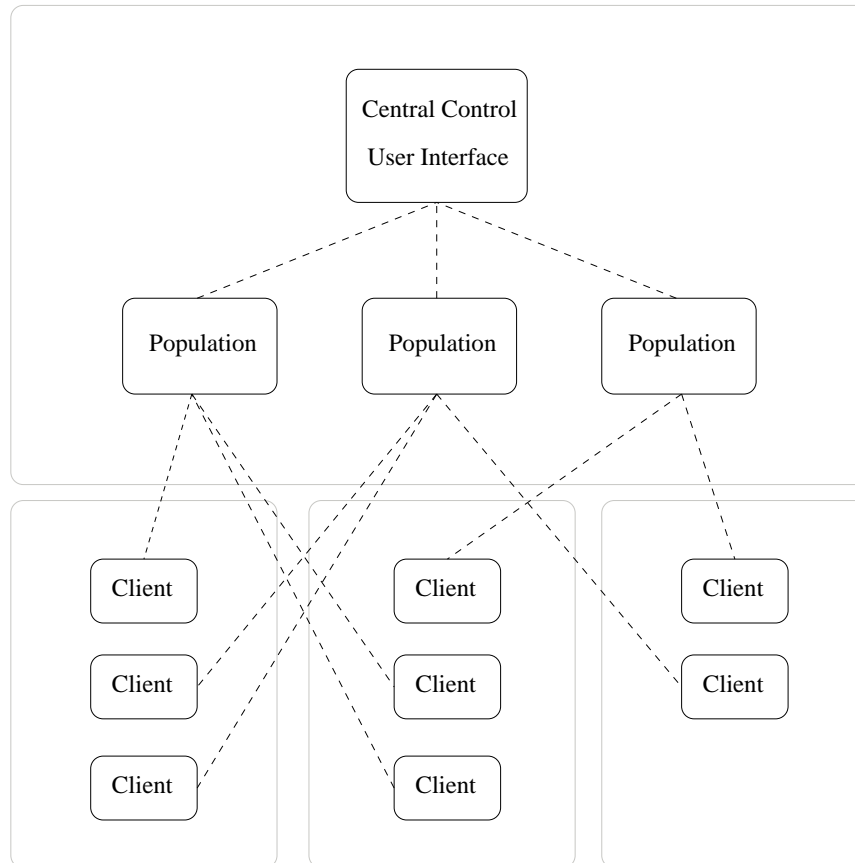


Figure 3: Network architecture of proposed parallel implementation. A central server maintains direct control (and thus analysis capabilities) over the whole computation, while the fitness-evaluations are delegated to different clients in potentially separated grid organisations.

involves a variety of kinetic parameters (the third level), which have to be tuned to the given problem. This is essentially a numeric optimisation problem in a highly nonlinear setting, for which ES approaches are good candidates. Therefore, we couple an ES algorithm with the graph-based GP, optimising the network parameters for each evolved topology. However, constraints on the computational time complexity may make it necessary to use less computing-intensive methods. Ideas under study are to use local search algorithms such as Simulated Annealing [5] or to combine numeric mutations directly with the main GP run.

## 6 Parallel Implementations

A shared characteristic of all Evolutionary Algorithms is the utilisation of a population of solution candidates. For each of these, a fitness value has to be computed, which constitutes a computationally expensive process for complex systems such as CSNs. Therefore, it is worthwhile to use parallel computing power in order to speed up the evolution and to tackle problems of a size that is of biological interest. The ESIGNET project makes use of such techniques, and a brief outline of possible parallel implementations as well as current choices is given in this chapter.

A simple approach is a straightforward client-server architecture, in which a server manages the population while the clients calculate fitness values of any new candidate solutions. The server takes care of mutation, recombination, reproduction, and selection. Because of its simplicity, this

approach can easily be constructed from a non-parallel algorithm. The algorithm progresses in well-defined generations, which is helpful for the analysis as well as for theoretical investigations. The downside is that in an algorithm progressing in generations, all clients have to wait for the slowest one to finish before new work is distributed.

A significant speedup can be achieved by giving up the concept of strictly separated generations, using tournament selection to determine which candidates reproduce, while keeping the population size constant by removing randomly chosen individuals (this is called a steady-state strategy). This way, clients can constantly request new tasks whenever they are idle. We are currently following this approach, since it provides a good compromise between the speedup offered by a parallel system and the tractability of a client-server architecture.

Apart from these adaptations of sequential EAs, architectures optimised for parallel usage can be devised. Most of these use multiple populations, each either on a single computational unit or on a group of units. One example here is the island-approach, in which multiple populations are assumed to exist on islands which are separated by the water between them. Most of the time, evolution on each island is independent of the others, so that different solutions can arise on different island. Occasionally, the best individuals from one island migrate to another one, thus leading to a dripping exchange of information which can help populations to leave local minima.

Our current approach uses a client-server model to distribute the fitness-evaluation between several clients, while all handling of the populations are localised on the server. Even when multiple populations exist, they run as separate threads on one server, which is justifiable by the small amount of computing time needed to manage the population in comparison to the fitness-evaluation time. Solution candidates are transferred as strings via XML-RPC, which is a simple and yet robust protocol for transmitting data between different computing environments. This standard enables us to cross organisational borders and facilitates the grid-computing idea. An overview of the utilised network architecture is given in figure 3.

## 7 Conclusion

Finding the appropriate algorithm is essential to the automatic design of CSNs. Given the plethora of available approaches and the lack of experience and knowledge in the evolution of biological networks, it is not straightforward to create an efficient algorithmic scheme. The ESIGNET project is currently generating vital progress and experience in this field, and new insights will certainly be gained from the combination of well-proven approaches into a software capable of the artificial evolution of CSNs.

Considering the number of algorithmic combinations to choose from, it is of vital importance to design any approach in a modular way, so that individual parts of the algorithms can be exchanged and the results can be compared. Results from these comparisons will have scientific value of their own, as well as giving hints to the further development of CSN-evolution software. In the discussion above, we gave of short insight into the diversity of design principles that are currently used and outlined a possible combination of suitable methods for our project. As a promising example, we have chosen a graph-based Genetic Programming method, combined either with direct numeric mutations or an ES algorithm. First promising results will be reported on in other workpackages.

## References

- [1] W. Banzhaf, Peter Nordin, R.E. Keller, F.D. Francone. *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, 1998
- [2] H-G. Beyer, H-P. Schwefel. *Evolution strategies*. Natural Computing 1, pp 3-52, 2002
- [3] J.H. Holland. *Genetic algorithms and the optimal allocation of trials*. SIAM Journal on Computing 2(2), pp. 88-105, 1973
- [4] J.H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press, 1975
- [5] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. *Optimization by Simulated Annealing*. Science 220(4598), pp. 671-680, 1983
- [6] J.R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press, 1992
- [7] T.P. Runarson, X. Yao. *Stochastic Ranking for Constrained Evolutionary Optimization*. IEEE Transactions on Evolutionary Computation 4(3), 2000
- [8] K. Weicker. *Evolutionäre Algorithmen* (in German). Teubner, Stuttgart, 2002