in: Third German Workshop on Artificial Life
C. Wilke and S. Altmeyer and T. Martinetz, Editors,
pp. 155-1703, Verlag Harri Deutsch, Frankfurt a. M, 1998

Mesoscopic Analysis of an Artificial Chemistry with Self-organizing Topology

Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf

University of Dortmund
Dept. of Computer Science, Systems Analysis
D-44221 Dortmund, Germany
dittrich—ziegler—banzhaf@LS11.cs.uni-dortmund.de
http://ls11-www.cs.uni-dortmund.de

The analysis of artificial life systems is often much harder than their construction, especially in population-based systems with a huge number of interacting individuals. Due to insufficient results of microscopic methods (e.g. monitoring the actions of single molecules) and macroscopic measurements (e.g. diversity, complexity) we developed a stepwise mesoscopic analysis method based on classification and dynamic clustering. Knowledge about the system is accumulated in an iterative process, in which classifiers (feature detectors) extract information, which is used in turn to create new classifiers. The mesoscopic analysis is applied to an artificial chemistry with a self-organizing topology. The topological structure is determined by the molecules themselves and is based on hashing which is a very efficient memory access method.

Keywords:

self-organization, computational chemistry, constructive dynamical systems, algorithmic chemistry, chemical computing, cluster analysis, binary string system, automata reaction, self-organizing topology

1 Motivation

The construction of artificial life systems with complex behaviors is surprisingly easy [1]. Analysis is often much harder, especially in population-based systems with a huge number of diverse and interacting individuals. In such cases the reduction of experimental data is necessary. Simply measuring macroscopic parameters (i.e. temperature, diversity or complexity) allows to see that something is happening, but this results do not provide insight into how or what. Microscopic tracing of an experiment and understanding every single step is possible in principal. But with growing computational resources, this becomes more and more difficult, probably impossible, because time complexity for the generation of n microscopic events (e.g., $O(n)$) is lower than time complexity needed for analysis (e.g., $O(n \log(n))$ for sorting the population). Therefore, methods for analysis in between macroscopic and microscopic methods are required.

In this paper we present a stepwise mesoscopic analysis method for artificial chemistries, which might be applied to other population-based systems as well. In this mesoscopic analysis the population is decomposed into groups. The decomposition is based on so called **classifiers**[1]. A classifier represents a property of an object. A first-order classifier is simply a function $C : S \longrightarrow [0, 1]$. The value $C(s) = p$ represents the strength of a property C of a string s and is used for a separation of the population into partial quantities with a cluster algorithm. Data produced by the classifiers as well as the collection of classifiers themselves represent knowledge about the system. This knowledge will grow by creating and collecting more and more classifiers. It is then possible to gain more information about the system by classifying the classifiers, i.e. by analyzing the basic properties of classifiers with, for example, the same clustering procedure that separated the population. This can be done in an iterative process (Fig. 1).

The mesoscopic analysis introduced here will be applied to a specific artificial chemistry with a self-organizing hash-topology.

[1] This classifiers are not be confused with Holland's classifier systems in [2].

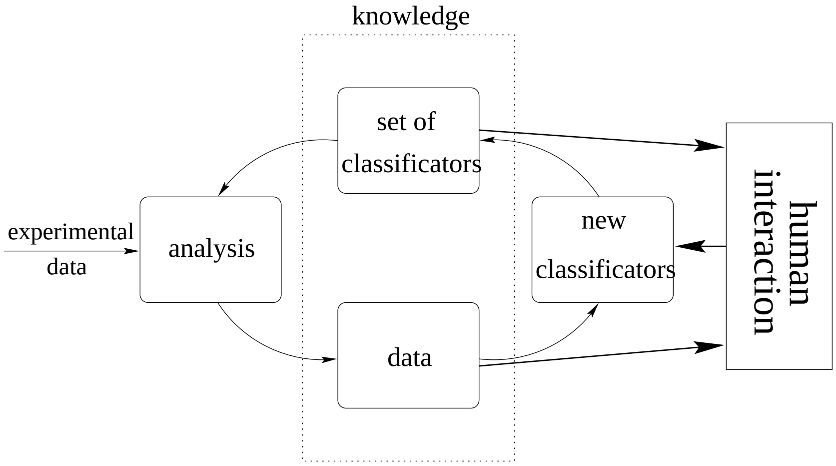

Figure 1: Cyclic increase of knowledge.

2 Mesoscopic Analysis

The mesoscopic analysis is a tool to find groups in data. The field of data analysis provides an overwhelming amount of methods, ranging from classical statistical methods (e.g., correlation analysis, c-means clustering) to modern adaptive methods from the fields of computational intelligence (e.g., neural networks, genetic programming). Furthermore, many more complex methods exist that are specialized for certain problem domains.

We do not present a new classification or clustering method here, but we suggest a method how to apply available data analysis methods on artificial chemistries or on other population-based evolutionary artificial life systems. Our method should allow the integration of different available data analysis tools in an uncomplicated way.

3 First-Order Classifiers

A **first-order classifier** C relates an object s to a property by indicating to what extend s has the property.

A first-order classifier is defined as a map $C : S \longrightarrow [0, 1]$. Without loss of generality and for simplification the output of a classifier is normalized. If the output of $C(s) = p$ is either 0 (s does not have the property) or 1 we call the classifier **binary**, otherwise the classifier C might be called **fuzzy**.

There are different ways of constructing a classifier: (1) **manually** by using apriori knowledge or by estimating the classifying property by an apriori analysis of the data, and (2) **automatically** by learning the mapping C. Typical examples are artificial neural networks, evolutionary algorithms or any other supervised or unsupervised algorithm with the ability to learn a function $S \longrightarrow [0,1]$.

A typical example for a manually defined classifier is a detector C_s for a certain subsequence s. For the binary case $C_s(s')$ is equal to 1 if the subsequence s can be found in s' otherwise $C_s(s')$ is 0.

4 Higher-Order Classifier

A **second-order classifier** is defined as a mapping $C : S \times S \longrightarrow [0,1]$. If C is the spatial distance between two objects, it can be called a **distance measure** or shortly **distance**. As mentioned in the previous section we can define C manually or automatically. Using an evolutionary algorithm for the automatic generation of classifiers of any order is an appropriate method. Its fitness function then relies on the phenotypic/genotypic behavior of the objects or is given explicitly by user interaction [3]. It is also possible to construct a second-order classifier C_F based on first-order classifiers $C_1, \dots C_n$, e.g., by using the Euclidean distance $C_F(s,s') = \sqrt{\sum_i (C_i(s) - C_i(s'))^2}$ [4].

5 Application and Integration of Classifiers

In this section we shall describe and demonstrate how classifiers may be integrated into a mesoscopic analysis of an evolutionary population-based system. The development of the population $P = (s_1, \dots, s_M)$ in time requires the analysis of $P(t)$ (population at time t) for every single time step. The results of consecutive analyses have to be compared and correlated.

5.1 Clustering

The task of a cluster analysis procedure is to classify a set $P = \{s_1, \ldots, s_n\}$ into c partial quantities [5, 6, 7]. The degree of affiliation u_{ik} of object s_i to cluster c_k is computed and quantifies the probability of s_i to belong to c_k. In the case of fuzzy clustering, u_{ik} is $\in [0, 1]$. The partition of P should obey the following rules: (1) similar data should be partitioned into similar clusters: **homogeneity**, and (2) different data should be separated into different clusters: **heterogeneity**. A step in a mesoscopic analysis which analyses groups in huge data sets thus uses a clustering based on previously defined or selected classifiers. The creation of this classifiers is explained in Sec. 8.1 and requires some apriori knowledge of the observed system. So the properties of s_i, described by its vector of classifier values $\vec{c}(s_i)$, and its affiliation to a cluster reduces the amount of information and gives clues for further inverstigations.

5.2 Tracing of Cluster Development

The development of a population is traced by analyzing its state in consecutive generations (in our experiments each generation has about $2 \cdot 10^3$ reactions) and by clustering the population. The cluster c can now be described with the common properties of every s_i in c. Clusters at t_i and t_{i+1} are related if they have the same "property". This series of clusters can then be analysed to get insight into the systems behavior at a more coarse level.

An analysis of the development of properties of an artificial chemistry using the above explained techniques is the main object of the following sections.

6 Artificial Chemistry

Our mesoscopic analysis method will be applied to a static and dynamic analysis of an artificial chemistry (see [8],[9]). Usually, an artificial chemistry consists of three parts: (1) a set of objects (molecules, substances) S, (2) a set of collision rules R, and (3) an algorithm, which models the reaction vessel and is therefore called **reactor algorithm**. For a detailed introduction to the field of algorithmic and artificial chemistry see the description of the workshop in this proceedings.

In our case the objects are binary strings of fixed length $S = \{0,1\}^{32}$ [10, 11]). The collision rules used in this example are defined by the **automata reaction**, which is based on a finite state automaton. This automaton is a mixture of a Turing-machine and a register machine. It has also been inspired by Hofstadter's [12, 13] Typogenetics.

The automata reaction instantiates a deterministic reaction $s_1 + s_2 \Longrightarrow s_3$, with $s_1, s_2, s_3 \in \{0,1\}^{32}$. In order to calculate the product s_3, string s_1 is "folded" into an automaton A_{s_1}, which gets s_2 as an input. The construction of A_{s_1} ensures that the automaton will halt after a finite number of steps. Because A_{s_1} is a deterministic finite automaton, the automata reaction defines a function $r_{a2} : \{0,1\}^{32} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$. Due to space limitations, the automaton is not shown here. It is explained in detail in [14, 15]. The collision rules are all second-order catalytic reactions of the form $s_1 + s_2 + X \longrightarrow s_1 + s_2 + s_3$, shortly $s_1 + s_2 \Longrightarrow s_3$. All collisions of two objects s_1, s_2 will have a unique outcome s_3.

7 Self-organizing Topology Based on Hashing

A **topology** restricts the possible interactions in a population based system by defining a neighbourhood for each individual. A topology is called a **self-organizing topology** if the topology is changeable and if this change is directed by the elements of the topological system (the individuals) themselves.

7.1 Hashing

Hashing is an efficient and widely used technique in computer science for implementing dictionaries or sets. By careful design, storage management by hashing requires only a constant time per lookup. The aim of hashing is to find a certain object $s \in S$ from the set S specified by a **key** $k = key(s), k \in K$ in a large memory array.

The essential idea now is that the (possibly infinite) set S of objects is partitioned into a number n_B of "classes" with finite content by the **hash function**

$$hash : K \to \{0, 1, \ldots, n_B - 1\}. \tag{1}$$

The classes are called **buckets** and each object s belongs to bucket $hash(key(s))$. In order to look up an object of the memory array, its corresponding bucket number $hash(key(s))$ is calculated and the (fixed-sized) bucket is searched for the object.

7.2 The Hash-reactor Algorithm

The following general algorithm is used to run the binary string system with hash-topology (see [16] for more details). M iterations of the algorithm are called a **generation**. This notion is used to generate a time scale.

```
b1 ← chooseBucket1()
s1 ← getObj(b1)
b2 ← chooseBucket2(s1, b1)
s2 ← getObj(b2)
products ← collide(s1, s2)
for each s3 ∈ products do
   b3 ← chooseBucket3(s1, s2, b1, b2)
   storeObj(s3, b3)
done
goto 1 (iterate)
```

s_1, s_2, s_3 are binary strings and $b_1, b_2, b_3 \in \{0, 1, \ldots, n_B - 1\}$ are bucket numbers (indices). The function $chooseBucket_1$ selects a bucket number randomly. For $chooseBucket_2$ and $chooseBucket_3$ a hash function can be used to calculate b_2 and b_3, respectively. The function $getObj(b)$ returns a randomly selected string from bucket b. The procedure $collide(s_1, s_2)$ computes the outcome of a collision, in our case, by using the automata reaction. The procedure $storeObj(s, b)$ replaces a string in bucket b by s. The strategy for the replacement is first-in-last-out.

8 Application

For the experiments the following settings are used:

$chooseBucket_1()$	$= randomBucket()$
$chooseBucket_2(s_1, b_1)$	$= randomNeighbor(b_1)$
$chooseBucket_3(s_1, s_2, b_1, b_2)$	$= randomNeighbor(b_1)$
$randomNeighbor(b)$	$= hash(key(getObj(b)))$
$key(s)$	$= \sum_{i=0}^{15} 2^i s^{(i+8)}$, $s = (s^{(31)}, \ldots, s^{(0)}) \in \{0,1\}^{32}$
$hash(k)$	$= k \mod n_B$
n_B	$= 200$
n_{bs}	$= 10$
M	$= 2000$
$collide(s_1, s_2)$	$= \begin{cases} \{\ \}, & \text{if } r_{a2}(s_1, s_2) \in \{s_1, s_2\} \\ r_{a2}(s_1, s_2), & \text{else.} \end{cases}$
r_{a2}	= automata reaction a2

Figure 2 shows three macroscopic observables of an experiment. The **productivity** indicates the percentage of non-elastic collisions per generation, **Diversity** is the number of different strings divided by the population size. For a detailed explanation of the **distance distribution complexity** see [17]. The development of the macroscopic observables indicates an interesting developmental phase, beginning at $t \approx 200$, but does not elucidate the process. To get a better understanding of the process we begin the mesoscopic analysis by defining a set of specific classifiers.

8.1 Special Classifiers

For the mesoscopic analysis of the topological structure (e.g. the organization of the buckets) **bucket classifiers** are needed. A bucket classifier is a function $c : \{0, \ldots, n_B\} \rightarrow [0, 1]$ mapping a bucket number to the interval $[0, 1]$. We can construct a bucket classifier c_{Bucket} by using a string classifier $C_S : S \rightarrow [0, 1]$ with the formula:

$$c_{Bucket}(b) = \frac{1}{n_{bs}} \sum_{i=0}^{n_{bs}-1} C_S(soup(b, i)) \tag{2}$$

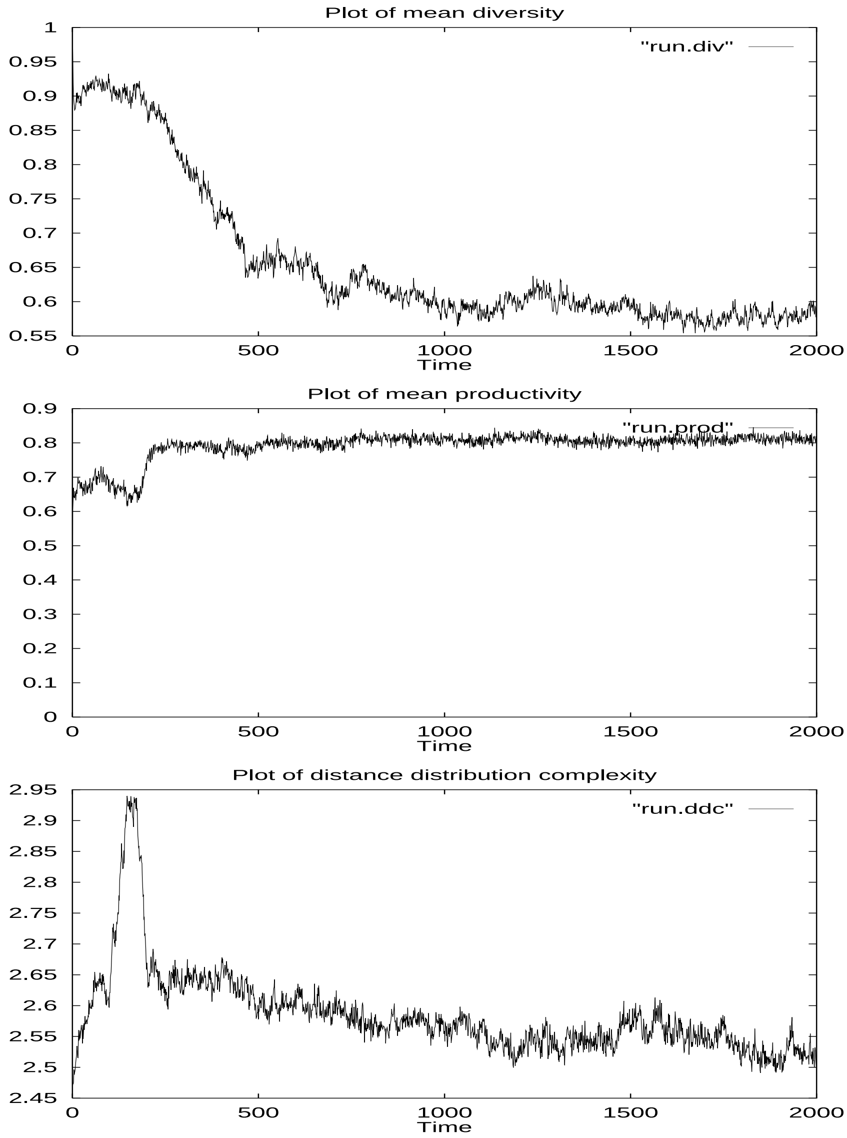

Figure 2: Macroscopic parameters of a typical experiment. The x-axis shows the time t in generations. For further explanations see text.

where $soup(b, i)$ denotes the i-th sequence in bucket b.

In our experiments we are particularly interested in the spatial organization of buckets, so our first approach is the creation of classifiers measuring topological features. The classifier

$$c_{out}(b) = \frac{N(b)}{n_{bs}} \tag{3}$$

with

$$N(b) = |\{hash(key(soup(b, 0))), \ldots, hash(key(soup(b, n_{bs} - 1)))\}|$$

the number of different neighbors of bucket b, indicates the diversity of the neighborhood of bucket b by counting the different addresses the objects in b point to. We additionally built the classifier $c_{in}(b)$

$$c_{in}(b) = \frac{1}{M} \sum_{i=0}^{n_B-1} \sum_{j=0}^{n_{bs}-1} \begin{cases} 1, & \text{if} \quad hash(key(soup(i,j))) = b, \\ 0 & \text{otherwise.} \end{cases}, \tag{4}$$

which calculates the number of $s_i \in S$ pointing to b.

$C_1(s)$ (5), and $C_{s_m}(s)$ (6) were used to build bucket classifiers with (2). The $C_1(s)$ classifier estimates the self-replicating ability by testing s against the strings s_0 =**0000000** and s_f =**fffffff**. String s is the operator processing both, s_0 and s_f.

$$C_1(s) = 1 - \frac{d_{Ham}((s \oplus s_0), s) + d_{Ham}((s \oplus s_f), s)}{64} \tag{5}$$

The number of bit flips, either from 0 to 1 ($d_{Ham}((s \oplus s_0), s)$) or from 1 to 0 ($d_{Ham}((s \oplus s_f), s)$) caused in s_0 and s_f indicate the replicating ability of s. $C_1(s)$ is at maxixmum if s causes as many bit flips in s_0 (and s_f) as necessary to reproduce its bit pattern.

The first-order classifier $C_{s_m}(s)$ (6) checks for replicating activity at certain bit positions specified by the substring s_m:

$$\begin{aligned} C_{s_m}(s) = 1 \;&-\; \frac{d_{Ham}(((s \oplus s_0) \wedge s_m), s \wedge s_m)}{2\#ones(s)} \\ &-\; \frac{d_{Ham}((s \oplus s_f) \vee \neg s_m, s \vee \neg s_m)}{2\#ones(s)} \end{aligned} \tag{6}$$

with $s_m \in S$ and $\#ones(s)$ denoting the number of ones in s.

8.2 Application of Classifiers

The application of the bucket-classifiers reduces the information about the population to bucket-properties (a bucket-property is a result of a bucket-classifier) and by clustering the buckets using these results, we can now investigate the development of the experiment in Fig. 2 more accurately.

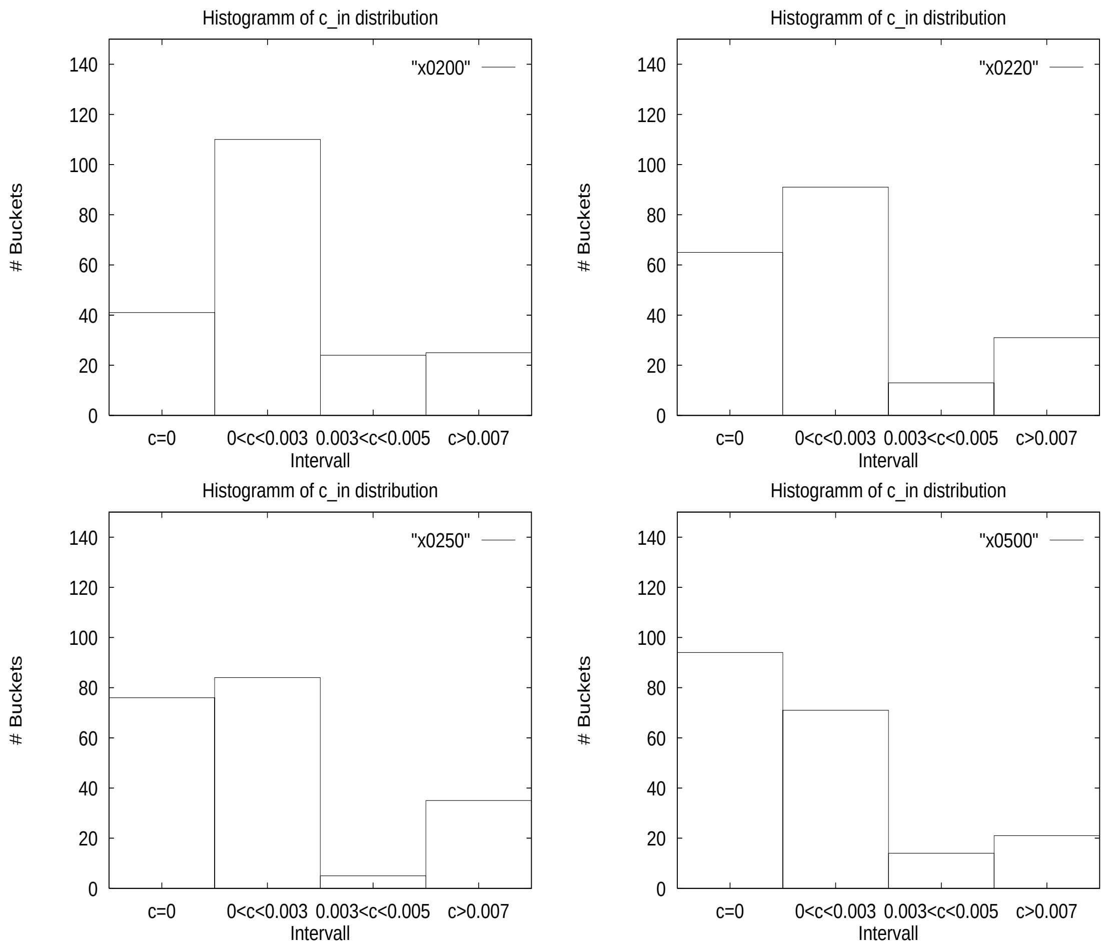

Figure 3: Histogramm of the distribution of c_{in}-values.

Looking at the information obtained from clustering the buckets with (4), the $c_{in}(b)$ classifier, reveals that buckets in earlier generations have a distribution of c_{in} values as in Fig. 3. The development of this histogramm shows that the spatial organization is changing. At the beginning nearly every bucket has a c_{in}-value of non-zero and few have a remarkable higher c_{in}-value. At about $t = 250$ the situation changed: there are now much more buckets with low or zero c_{in}-values. But there still are some "hot spots".

Further investigation of the strings inside these four intervals will show if there is a correlation between topological structure and functional behavior.

In earlier experiments we used an automatically generated classifier to separate strings in different generations [4]. A Genetic Programming system created a classifier testing bit 14 in the strings to make this separation. Although this experiment was very different from that explained here, application of c_{14} proved to be useful (Fig. 4). Thus, recycling of "old" classifiers is possible and reduces the effort for analysis.

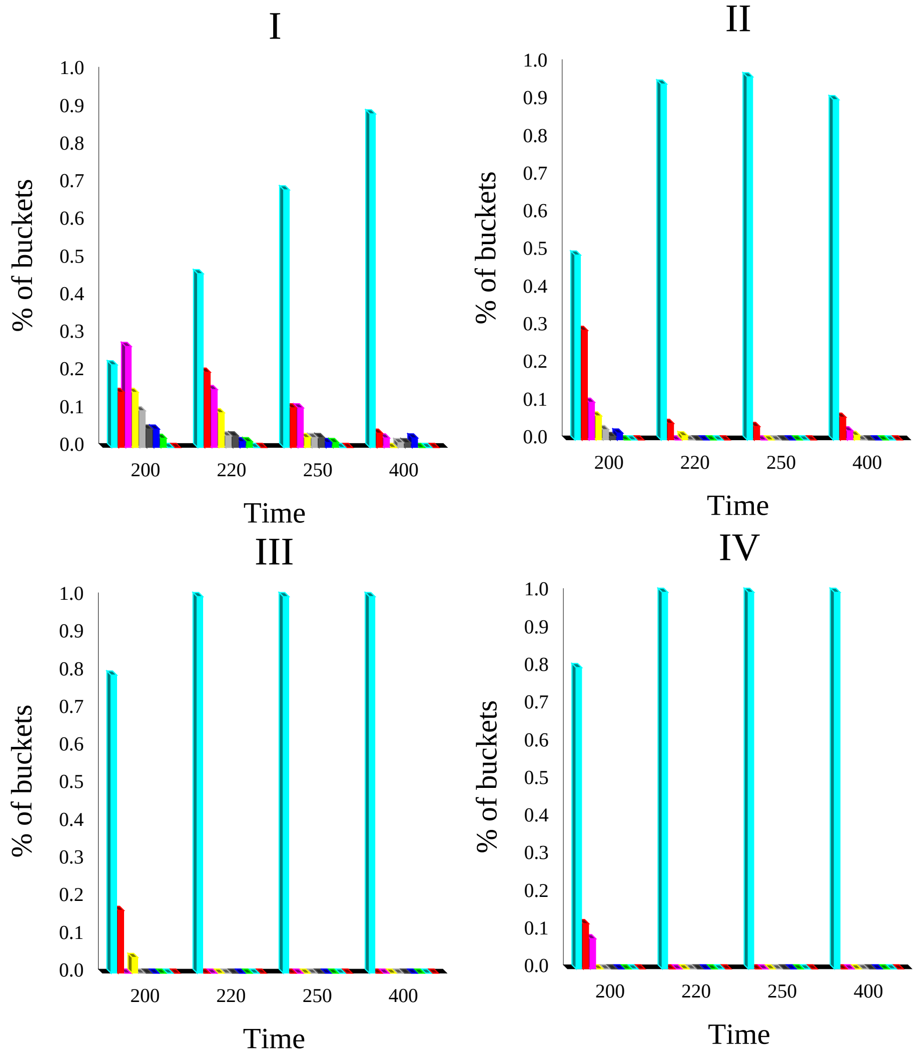

Figure 4: Histogramm of bit-14-distibution at $t = 200, 220, 250, 400$ in the intervals I, II, III, IV.

At the beginning of the second phase of development (at about $t = 200$) the strings inside of interval III ($c_{in} \in [0.003, 0.007]$) and IV ($c_{in} > 0.007$) show no significant difference from strings in interval I ($c_{in} = 0$) and II ($c_{in} \in [0, 0.003]$). But with increasing time the strings become more divers and at $t \approx 250$ it is possible to separate strings in III and IV (100 % of them have bit $14 = 0$) from those in I and II (bit $14 = 1$ in some cases).

It is now clear that the strings have a different structure, but is there a functional difference, too? The first clustering resulted in four groups of different buckets. Functional difference of strings must be tested with a string classifier and therefore we used (6) with $s_m = 0000ffff$ (replicating ability on the right side) to analyse the functional properties of strings in the four different clusters. The development between $t = 200$ to $t = 400$ is shown in Fig. 5. There is a decrease of $C_{0000ffff}$-values in interval IV, and an increase in interval I. This is another evidence that the strings in different intervals have both different functional and structural properties.

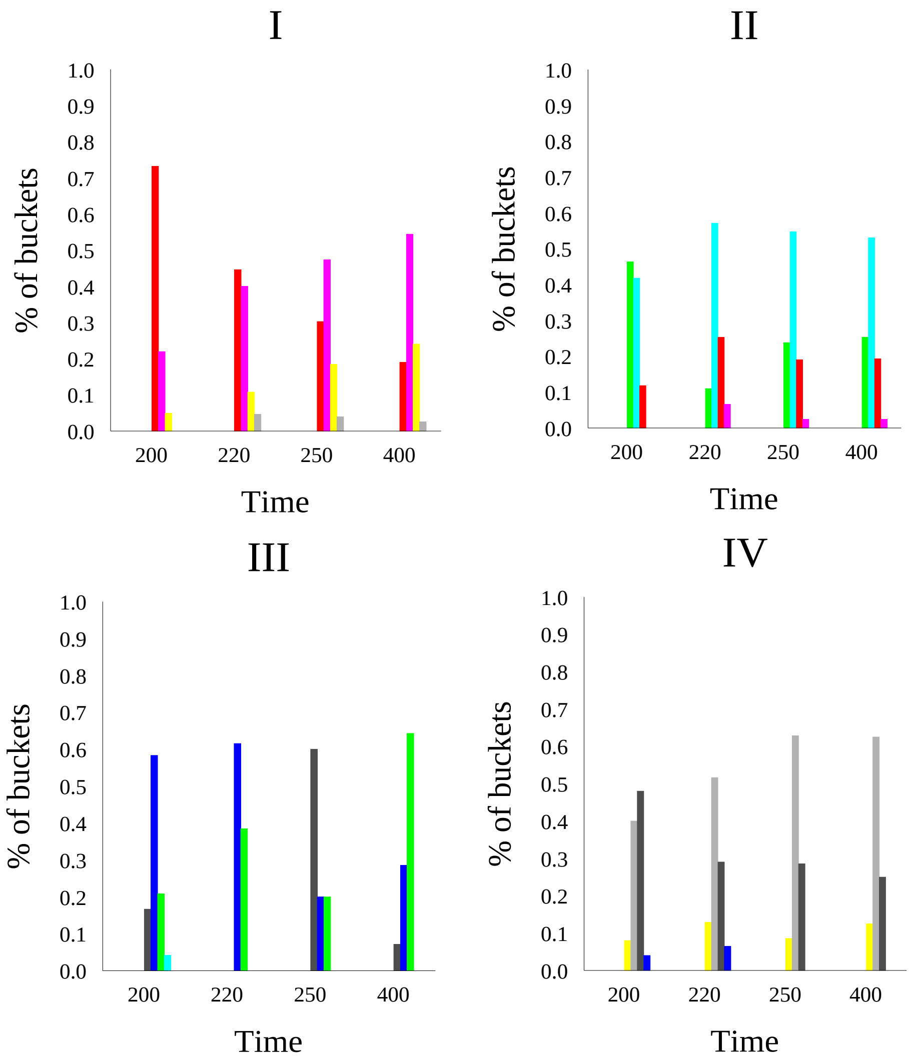

Figure 5: Histogramm of right-replicating ability of buckets at $t = 200, 220, 250, 400$ in the intervals I, II, III, IV.

9 Meta-Analysis of Classifiers

The analysis can be regarded as a dynamic process in which classifiers are generated and discarded. If we interpret our analysis as a population system with classifiers beeing the individuals, we can easily apply the method to itself. The advances of this iterative analysis is the accumulation of knowledge not only about the system beeing observed. Dynamic clustering of classifiers will in turn result in more specific knowledge about the development of new classifiers and will thus give new and more detailed insight into the properly observed system. Figure 6 shows an example of a classifier-clustering to demonstrate this approach. Clustering of classifiers creates groups of classifiers whith

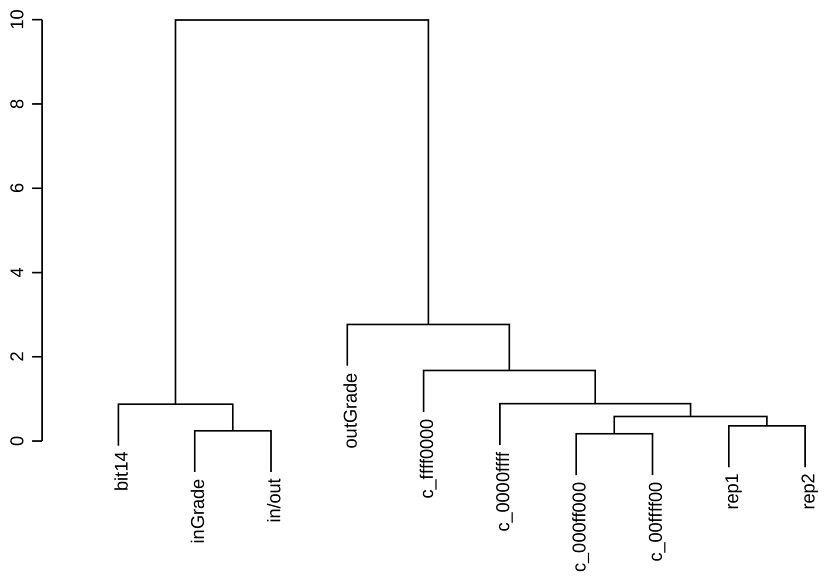

Figure 6: Hierachical clustering of the classifiers in generation 3000.

similar properties. If a classifier's subdivision of the population resembles another, they are likely to belong to the same cluster. As shown in Fig. 6 there is a strong correlation between different replicating properties. The classifiers using only topological information constitute another, separated cluster. The assignment of c_{out}, a spatial classifier, to the replicating cluster seems to be an error. But the calculation of $hash(key(s))$ depends on the structure of s and therefore c_{out} belongs to the "structure" cluster.

10 Summary

A mesoscopic step-wise analysis method for evolutive population-based systems has been suggested in which knowledge about the system is gained by decomposing it with the help of classifiers. The application of classificators generates information which can be used to generate new classificators which in turn can be applied to the system. Iterating the analysis cycle (Fig. 1) and using alternately classifiers and clustering now becomes a useful tool for analysing complex systems. But the crucial point is the analysis of the results of this method. Assumptions may lead to inappropriate classifiers which in turn may lead to wrong conclusions. So careful investigation and verifying of the results is necessary.

Acknowledgements

This project is supported by the DFG (Deutsche Forschungsgemeinschaft), under grant Ba 1042/2-2.

References

[1] Thomas S. Ray. An approach to the synthesis of life. In Christopher G. Langton et al., eds., *Proceedings of the Workshop on Artificial Life (ALIFE '90)*,1992.

[2] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40, 1989.

[3] W. Banzhaf. Interactive evolution. In Thomas Bäck et al., eds., *Handbook of Evolutionary Computation.* IOP Publishing, 1997.

[4] P. Dittrich, J. Ziegler, and W. Banzhaf. Mesoscopic analysis of self-evolution in an artificial chemistry. In Chris Adami et al., eds., *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*, 1998.

[5] J. Bacher. *Clusteranalyse.* Oldenbourg, München, Wien, 1996.

[6] S. Miyamoto and K. Nakayama. Similarity measures based on a fuzzy set model and application to hieracical clustering. *IEEE Trans., Syst., Man and Cybernetics*, 16(3):479–482, 1986.

[7] J. C. Dunn. Well separated clusters and optimal fuzzy-partitions. *Journal of Cybernetics*, 4:95 – 104, 1974.

[8] F.J. Varela. On being autonomous: The lessons of natural history for systems theory. In G.J. Klir, editor, *Applied General Systems Research*, pages 77–84, 1978.

[9] Walter Fontana and Leo W. Buss. "The arrival of the fittest": Toward a theory of biological organisation. *Bull. Math. Biol.*, 56:1–64, 1994.

[10] Wolfgang Banzhaf. Self-replicating sequences of binary numbers – Foundations I and II: General and strings of length n = 4. *Biological Cybernetics*, 69:269–281, 1993.

[11] W. Banzhaf. Self-organizing algorithms derived from RNA interaction. In W. Banzhaf and F. H. Eeckman, editors, *Evolution and Biocomputation: Computational Models of Evolution*,1995.

[12] Douglas R. Hofstadter. *Gödel, Escher, Bach:*. Klett-Cotta, Originalausgabe: 1979, Basic Books, New York, 6. edition, 1985.

[13] Harold C. Morris. Typogenetics: A logic for artificial life. In Christopher G. Langton, editor, *Artificial Life*, 1989.

[14] P. Dittrich and W. Banzhaf. Self-evolution in a constructive binary string system. To appear in: *Artificial Life*, 4(2), 1998.

[15] P. Dittrich. Source code of the automata reaction. FTP server at University of Dortmund: *ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/src/autoreac-1.0.tar.gz*

[16] P. Dittrich and W. Banzhaf. A Topological Structure Based on Hashing - Emergence of a "Spatial" Oranization. *Fourth European Conference on Artificial Life (ECAL97)*, 1997.

[17] Jan T. Kim. Using distance distribution to measure complexity of populations. In D.J. Stein and L. Nadel, editors, *1993 Lectures in Complex Systems*. Addison-Wesley, Redwood City, CA, 1995.