Eating Data is Good for Your Immune System: An Artificial Metabolism for Data Clustering using Systemic Computation

Erwan Le Martelot[1,3], Peter J. Bentley[2], and R. Beau Lotto[3]

[1] Engineering Department, University College London, London, UK,
e.le_martelot@ucl.ac.uk,
[2] Computer Science Department, University College London, London, UK,
[3] Institute of Ophthalmology, University College London, London, UK

Abstract. Previous work suggests that innate immunity and representations of tissue can be useful when combined with artificial immune systems. Here we provide a new implementation of tissue for AIS using systemic computation, a new model of computation and corresponding computer architecture based on a systemics world-view and supplemented by the incorporation of natural characteristics. We show using systemic computation how to create an artificial organism, a program with metabolism that eats data, expels waste, clusters cells based on the nature of its food and emits danger signals suitable for an artificial immune system. The implementation is tested by application to a standard machine learning set and shows excellent abilities to recognise anomalies in its diet.

1 Introduction

An increasingly popular view in the field of Artificial Immune Systems (AIS) holds that innate immunity (as enabled by non-adaptive cells such as dendritic cells) can play a significant role in maintaining immunity in computer systems [1]. Notions such as the Danger Theory suggest that normal self cells may provide signals when damaged, thus helping to encourage the response of immune cells in the right areas of the tissue of an organism at the right time [2]. Previous work has investigated the development of an artificial tissue to serve this function, providing an interface between data and AIS, and performing preliminary data processing and clustering [3].

In this work we extend the previous work on tissue for AIS, and investigate a different implementation based on the recent paradigm and computer architecture, systemic computation (SC) [4] designed to support any bio-inspired system by enabling natural characteristics found in biology. In contrast to previous implementations of tissue, which largely ignore the relationships between real organisms and their environments, here we present a model of organism, implemented as a systemic computation program with its own metabolism that eats data, expels waste, clusters its cells depending on the nature of its food and

can emit danger signals for an AIS. The implementation is tested by application to a standard machine learning set (Breast Cancer data [5]) and shows excellent abilities to recognise anomalies in its diet.

2   Background

Although not commonly modelled, the notion of tissue is fundamental to immunity. The immune system within an organism defends the tissue of that organism. The concept of artificial tissue has been used for instance in the POEtic project, aiming at creating a hardware platform organised with a similar hierarchy as found in biological systems [6], and using reconfigurable circuits to simulate tissue growth [7]. It has also been used in work that implemented an AIS in a sensor network, the sensor nodes taking on the role of tissue cells [8].

In biology, tissue is a crucial part of the immune system and its importance was particularly highlighted by Polly Matzinger when introducing the Danger Model [2]. This view rejected the notion that the immune system differentiates self from non-self and suggested that it instead responds to cellular damage. It thus suggests that cells that die abnormally release signals which encourage immune cells to converge on that location and become more active.

This theory was adopted in [3] to propose two ways of growing tissues where damaged cells would release danger signals exploitable by an AIS. Tissue was defined as the interface between a problem to solve and the AIS. Here we follow a similar view, but attempt to improve the tissue model and its potential advantages by implementing a tissue-growing program designed for AIS using systemic computation - a parallel computer architecture designed to support natural computation.

Systemic computation is not the only model of computation to emerge from studies of biology. The potential of biology had been discussed in the late 1940s by Von Neumann who dedicated some of his final work to automata and self-replicating machines [9]. Cellular automata have proven themselves to be a valuable approach to emergent, distributed computation [10]. Generalisations such as constrained generating procedures and collision-based computing provide new ways to design and analyse emergent computational phenomena [11] [12]. Bio-inspired grammars and algorithms introduced notions of homeostasis (for example in artificial immune systems), fault-tolerance (as seen in embryonic hardware) and parallel stochastic learning, (for example in swarm intelligence and genetic algorithms) [4].

New architectures are also popular, whether distributed computing (or multiprocessing), computer clustering and grid computing and even ubiquitous computing and speckled computing [13]. Thus, computation is increasingly becoming more parallel, decentralised and distributed. However, while hugely complex computational systems will be soon feasible, their organisation and management is still the subject of research. Ubiquitous computing may enable computation anywhere, and bio-inspired models may enable improved capabilities such as reliability and fault-tolerance, but there has been no coherent architecture that

combines both technologies. Indeed, these technologies appear incompatible - the computational overhead of most bio-inspired methods is prohibitive for the limited capabilities of ubiquitous devices.

To unify notions of biological computation and electronic computation, [4] introduced systemic computation as a suggestion of necessary features for a computer architecture compatible with current processors, yet designed to provide native support for common characteristics of biological processes.

In this paper we use an approach similar to [3] and deepen the biological analogy by modelling an artificial organism as a program with metabolism. The program does not only mimic some tissue features but also mimics many fundamental properties of living organisms: eating data as food and expelling waste, while growing tissue, and releasing danger signal when its cells die in an abnormal way.

To implement such program SC provides a suitable alternative approach to traditional computation. Indeed with SC, organisms and software programs now share a common definition of computation. The work illustrates how organisms and programs can behave similarly, sharing the notion of metabolism, using SC.

3   Overview of Systemic Computation

SC [4] is a new model of computation and corresponding computer architecture based on a systemics world-view and supplemented by the incorporation of natural characteristics (previously listed). This approach stresses the importance of structure and interaction, supplementing traditional reductionist analysis with the recognition that circular causality, embodiment in environments and emergence of hierarchical organisations all play vital roles in natural systems. Systemic computation makes the following assertions:
- Everything is a system.
- Systems can be transformed but never destroyed.
- Systems may comprise or share other nested systems.
- Systems interact, and interaction between systems may cause transformation of those systems, where the nature of that transformation is determined by a contextual system.
- All systems can potentially act as context and affect the interactions of other systems, and all systems can potentially interact in some context.
- The transformation of systems is constrained by the scope of systems, and systems may have partial membership within the scope of a system.
- Computation is transformation.
In systemic computation, everything is a system, and computations arise from interactions between systems. Two systems can interact in the context of a third system. All systems can potentially act as contexts to determine the effect of interacting systems. A system is divided into three parts: two schemata and one kernel. These three parts can be used to hold anything (data, typing, etc.) in binary as shown in Figure 1(a). The kernel defines the result of two systems interacting in its context (and may also optionally hold data if it is interacting with

another system). The two schemata define which subject systems may interact in this context as shown in Figures 1(b) and 1(c). A system can also contain or be contained by other systems. This enables the notion of scope. Interactions can only occur between systems within the same scope. An SC program therefore comprises systems that are instantiated and positioned within a hierarchy (some inside each other). It thus defines an initial state from which the systems can then randomly interact, transforming each other through those interactions and following an emergent process rather than a deterministic algorithm. For full details see [4] and [14].

(a) Data system   (b) Context system   (c) Interaction
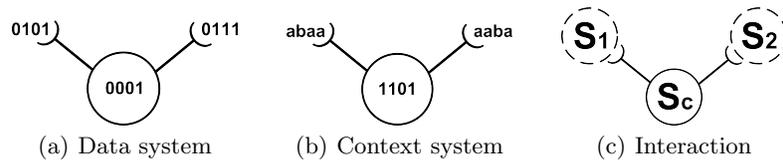
Fig. 1. 1(a): A system used primarily for data storage. The kernel (in the circle) and the two schemata (at the end of the two arms) hold data. 1(b): A system acting as a context. Its kernel defines the result of the interaction while its schemata define allowable interacting systems. 1(c): An interacting context. The contextual system S_c matches two appropriate systems S_1 and S_2 with its schemata and specifies the transformation resulting from their interaction as defined in its kernel.

Systemic Computation has been used to model genetic algorithms, neural networks, and has demonstrated properties of flexibility, fault tolerance, and self-repair [14] [15] [16].

4 An SC Program with Metabolism

4.1 Systemic Analysis

When programming with SC it is necessary to perform a systemic analysis in order to identify and interpret appropriate systems and their organisation [14]. The first stage is to identify the low-level systems (i.e. determine the level of abstraction to be used).

In most artificial immune systems, the level of abstraction is the cell: few approaches require modelling of the internal organelles or genome of cells, and few require modelling of populations of organisms. Here we intend to model the growth of tissue cells, the consumption of "food" (data items), the expulsion of waste and the emission of danger signals. Thus an abstraction at the cellular level is appropriate, with systems being used to explicitly model each element.

The identification of appropriate low-level systems is aided by an analysis of interactions. The organism should be able to eat food from its environment, use this food to grow organs (clusters of cells) by creating new cells and expel waste into the environment.

To prevent being overloaded with systems, the waste can be recycled into new food (a simple ecosystem model). Food and waste could therefore be seen as different states of the same system (in SC systems can be transformed, but never created from nothing or destroyed). Also, the food is what the organism takes from its environment to be able to grow. Therefore cells and all the necessary matter for the growth should also derive from the food systems.

We can thus visualise the ecosystem between the organism and the environment as shown in Figure 2.
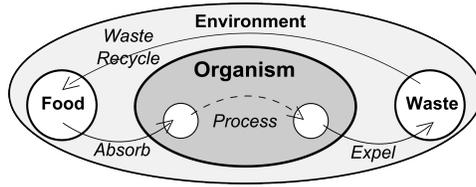
Fig. 2. 'Food to waste' cycle for an organism within its environment: Food is absorbed by the organism, processed as energy to grow tissues before being expelled when the organism cannot make use of it any more.

Looking within the organism, it takes food as input and this food must be sufficient to grow tissue. One simple way to model this is by using the approximation that the food is transformed into cells when absorbed by the organism. However, to enable cells to adhere to each other (rather than float free), cells need some sticky adhesion molecules. Here we do not need to explicitly model all these molecules but an "adhesion surface" is at least required to bind two or more cells together. As SC forbids the creation of systems from nothing, the adhesion surfaces must be obtained either from incoming food or from the cells themselves. In a biological organism each cell has a limited lifespan and thus dies at some point. It may then be consumed by macrophages or dendritic cells and its energy is partially recycled. In the model dead cells can thus be recycled to make adhesion surfaces. A growth process can now attach cells to each other by using adhesion surfaces to create tissue. To regulate this growth and introduce the notion of time, a decay process simulates the aging of cells. When cells die, a split process splits them from the adhesion surfaces they are bound to.

So the organism eats new data, converts each data item into a new cell, and attempts to bind that cell to itself, with cells made from similar data items binding to each other. Thus, a cell unable to bind to any group of cells reveals itself to be significantly different from them - more like the result of an invading pathogen then part of the organism. If this abnormal cell dies unbound, it can therefore be spotted as a potential anomaly. In that case, the death of the cell can entail that cell releasing a Danger signal (i.e. the cell can be converted into a signal). This signal can then be used by an AIS algorithm which can be implemented through the addition of systems corresponding to immune cells. (Here we focus on the organism.)

The organism can also make use of a hunger parameter defining a maximum amount of alive cells it can contain at a time. This parameter can be stored in the organism system and the absorption context then only allows food absorption if the organism is "hungry". This parameter can be useful to avoid having the organism growing too big and using too much memory/data at a time. A bad usage of memory could indeed to some extend slow down the computation process significantly.

The organism food to waste chain is therefore as shown in Figure 3.
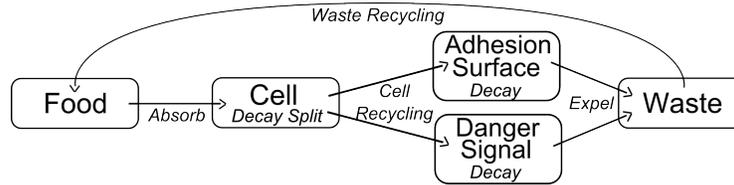
Fig. 3. 'Food to Waste' cycle within the organism: Food is absorbed, transformed into cells. When dying cells can be recycled into adhesion surfaces if they were part of a tissue or turned into a danger signal if they were single. Cells, adhesion surfaces and danger signals have a limited lifespan and decay over time (i.e. when they reach a certain age they die). When dying, cells also need a split process to detach them from the tissue they were part of.

From this defined cycle, the interactions and systems in the model can be written as follows (also see Figure 4):

```
organism }-absorb-{ food              → organism(cell)
cell }-growth-{ adhesion_surface      → cell(adhesion_surface)
cell(adhesion_surface) }}-split       → (cell adhesion_surface)
organism(cell) }}-cell_recycling      → organism(adhesion_surface or danger_signal)
X[age](time) }}-decay                 → X[age+1](time),X=cell or adhesion_surface or danger_signal
organism(X) }}-expel                  → (organism waste), X=adhesion_surface or danger_signal
universe(waste) }}-waste_recycling → universe(food[data])
```

The absorb system models endocytosis (e.g. via cell receptors), the growth system models the organism's genome, the decay models the aging (progression along the axis of time), the split system models a chemical breakdown between adhesion molecules and cell wall, the cell recycling models the phagocytes, the expel system models exocytosis, waste recycling systems model the ecosystem, the universe models the environment, the organism system models the boundary between tissue and environment, food systems model nutrients, cells model tissue cells, adhesion surfaces model adhesion molecules, danger signal systems model Matzinger's danger signals, waste systems model cell waste (unused or unusable compounds), and the time system models the dimension of time.

Figure 4 summarises the organism's organisation and shows the potential interactions.

Note that waste recycling, absorption, cell recycling and expel systems should have the same amount of instances. Indeed, on average if one food system can
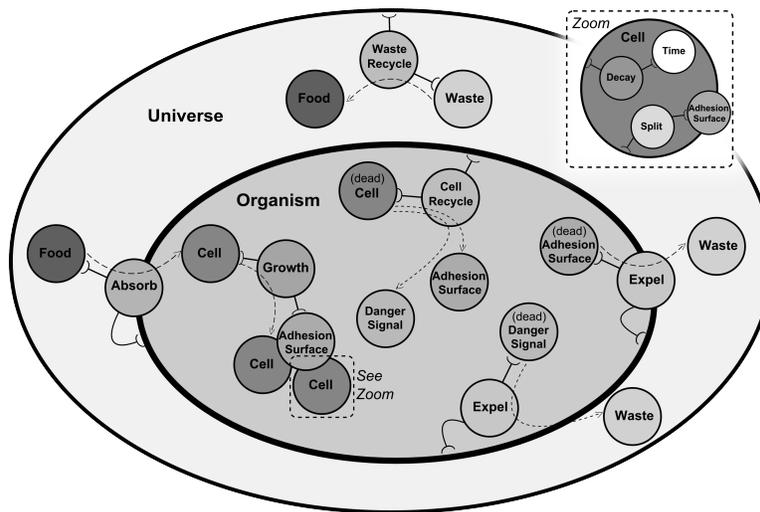
Fig. 4. Systemic organisation of the organism. The universe contains a waste recycling system, some waste and food, and an organism. The organism shares with the universe the absorption context. It contains cells, adhesion surfaces, danger signals, growth contexts, cells recycling contexts and expelling contexts. Finally cells (and thus all derived system states like adhesion surfaces and danger signals) contain the time system, a decay process and a split process. The schemata appear on context systems to show the allowable interactions between systems. The dashed arrows indicate the potential transformation of some systems during an interaction. For instance on the far left we can observe a food system interacting with an organism in an absorption context: the food is turned into a cell and injected into the organism.

be created at a time, then only one can be absorbed, then recycled and finally expelled at a time.

4.2 Data Clustering within the Artificial Organism

So far an organism has been modelled within SC. To use this organism for data clustering the data processing method has to be defined.

To incorporate data into the organism's metabolism, new data items are placed into food systems, where it is stored in the schemata. Data from an incoming stream can be introduced when recycling waste (i.e. new data are pushed into the resulting food systems). The amount of waste recycling and absorption systems gives the data introduction rate (the more food can be absorbed at a time, the more data are introduced). The data are then absorbed into the organism and transformed to cells. When a growth interaction occurs between a cell and an adhesion surface, the two are bound based on their data similarity. Algorithm 1 describes in pseudo-code the binding method. For binding a cell to an adhesion surface the adhesion surface is injected into the cell but remains also part of the organism so that more cells can bind to it.

Algorithm 1 Pseudo-code for the growth context binding method. τ is a given threshold. The distance function calculates the Euclidian distance of two vectors.

```
if adhesion surface not bound to anything then
    Bind cell and surface
    Surface data value ← Cell data value
else if distance(Cell data, Surface data) ≤ τ then
    Bind cell to surface
    Surface data value ← Average(Surface data, Cell data)
end if
```

The measure chosen in this implementation to compare data is the Euclidian distance (as was used in [3]). In the organism, cells cluster according to their values, and various clusters may emerge from this, thus reflecting the data distribution. If a cell is left single then it means it cannot bind anywhere and therefore holds data significantly different from the current most common data values contained within the organism. This cell is then turned into a danger signal holding the data that an AIS could use to develop antibodies.

5   Experiments and Results

To test the model and compare it with similar previous models [3], a series of experiments were performed using the standard "breast cancer" UCI machine learning data set [5], comprising 458 benign items (class 1) and 241 malignant items (class 2), each item being a vector of 9 real-valued numbers. The values were normalised to lie within the [0,1] interval.

5.1   Tuning the system

To tune the organism for this data set several settings were employed. Each experiment was repeated 20 times. Each run consisted of 3000 iterations with randomly picked data presented each iteration. Class 1 is treated as the "normal" class of data and class 2 is treated as "abnormal", from which one data item is introduced on average every 25 iterations (these values are taken from [3] to enable comparison), i.e. with a probability of 1/26.

All experiment settings involve a universe, an organism, a time system, an equal amount of waste recycling, absorption, cell recycling, expelling system varying in the experiments (see data introduction rate in Table 1), 250 data systems in experiments 1 to 12 and respectively 500, 750, 1000, 1250 and 1750 in experiments 13, 14, 15, 16 and 17. Each data system contains a decay and a split system. The organism initially contains 5 adhesion surfaces.

Table 1 shows the results of various tunings that were used. These results are computed by discarding the early computations (we discarded here the first 50000 computations) during which the organism grows to an adult (stable) state and stopping the experiments when the flow of data ends (thus not permitting organism's death from starvation).

Exp	Rate	τ	Lifespan	#Growth	Class 1		Class 2	
					mean	stddev	mean	stddev
1	1	0.2	15	100	22.70	1.27	99.87	0.39
2	1	0.3	15	100	11.54	0.76	99.63	0.69
3	1	0.4	15	100	7.56	0.51	98.72	1.01
4	1	0.5	15	100	5.39	0.67	96.19	1.69
5	1	0.4	5	100	9.16	1.11	99.20	1.17
6	1	0.4	10	100	7.85	0.48	99.06	1.14
7	1	0.4	20	100	7.23	0.41	98.73	1.06
8	1	0.4	15	5	14.40	0.81	99.91	0.42
9	1	0.4	15	10	8.98	0.57	99.61	0.62
10	1	0.4	15	25	7.85	0.59	99.53	0.87
11	1	0.4	15	50	7.62	0.40	98.95	0.90
12	1	0.4	15	150	7.62	0.50	98.44	1.00
13	2	0.4	15	100	6.99	0.59	98.56	1.36
14	4	0.4	15	100	7.01	0.68	97.88	1.34
15	8	0.4	15	100	7.48	0.94	98.88	1.24
16	16	0.4	15	100	6.85	0.81	98.33	2.39
17	24	0.4	15	100	7.24	0.90	98.29	2.21

Table 1. This table shows in percentage the average and standard deviation of data from each class creating a danger signal (false positive for class 1 and true positive for class 2) for various setups. Parameters are respectively in order: new data introduction rate (per cycle), data comparison threshold τ, cell's lifespan, and amount of growth systems. Experiments 1-4 investigate the effect of varying τ, experiments 5,6,3,7 investigate the effects of varying lifespan, experiments 8-11,3,12 investigate changing the amount of growth systems, experiments 3,13-17 investigate varying the data introduction rate.

From the first four experiments we can observe that the data similarity threshold has a significant impact on the performance of the program. The bigger the threshold, the less benign items are left unbound but the more malign cells can potentially bind somewhere. A threshold of 0.4 was then used for the next experiments.

Experiments 5, 6, 3 and 7 show that increasing the lifespan lowers the misclassification of class 1 data whilst slightly increasing the one of class 2. This parameter should thus be tuned depending on the priorities in the potential application (i.e. class to be most precisely detected).

Experiments 8 to 11, 3 and 12 show that, similarly to lifespan, increasing the amount of growth systems better classifies class 1 and lesser classifies class 2.

Experiments 3 and 13 to 17 show that varying the data introduction rate does not have a significant impact on the classification accuracy.

Comparing these results with the ones from [3] and looking at the best setups, we clearly outperform their results. It is interesting to notice that the best setups here use a threshold of 0.4 against 0.2 in [3]. It seems that for this study, having a low threshold in a deterministic program as in [3] is better whilst in a stochastic approach such as SC a larger threshold works better.

5.2 Looking into the organism

This section investigates what can be learnt from the organism's inner state over time. It is expected that observing the organism from within should reveal information about the data set. To be an effective and useful tissue algorithm, suitable for use with AIS algorithms, the organism should organise itself in a stable pattern reflecting the data distribution. If for instance the data stream contains three distinct sets we expect to observe three distinct clusters of cells.

The relevant values to observe along the computation are the amount of cells, clusters and danger signals. Similarly to a real case on-line program execution, Figure 5 shows the state over time of an organism during a run of 5000 samples using the configuration of experiment 3. Again, the computations corresponding to the organism's early life (here the first 100000 iterations) are discarded in order to focus on the mature aspect of the organism.

Results from Table 1 already provide insights regarding the inner shape of the organism (i.e. its inner organisation). Class 2 (malign) items are very well identified which means that class 2 data are not easily aggregated to other data (otherwise they could not be well detected). Therefore class 1 (benign) data only is actually clustering and it is thus expected to observe on average one main cluster all along the program execution.

Figure 5 shows that the organism has a constant amount of cells in spite of constant cellular death. The organism therefore shows homeostatic behaviour. Danger signals are regularly emitted and represent the detected supposedly malign cells that could then be used by an AIS algorithm. The curve at the bottom shows the amount of clusters over time (in a smaller scale along the Y-axis for clarity). We can observe that as expected the organism keeps settling down into one cluster. New clusters are constantly created with the appearance of new adhesion surfaces but quickly these new clusters bind to the main one.

As the organism is designed to grow to match the data rate, such a program is therefore able to cope with various (unexpected) parameter changes, self-(re)organising with the data flow, and providing information over time regarding detected potentially abnormal data items. When used in conjunction with an artificial immune algorithm, the good accuracy of detection (albeit with a high false positive rate), and the automatic organisation of similar data into clusters should enable excellent performance overall. While temporal information is currently lost because of the stochastic computation of SC, this could be added as additional features of data items, enabling the clustering according to similar timings in addition to data values.

One advantage of SC is the simplicity of modelling new stochastic systems, so an immune algorithm could be added to this model by simply adding two or three new types of system (e.g. B-cell, T-cell, antibody) that would then automatically interact with the existing tissue systems. Another valuable advantage of using SC in our approach is the fault-tolerance and self-repair ability an SC model can naturally have, as investigated in [16]. Having robust software can indeed be an important feature in network security to ensure the program can survive even severe damage provoked for instance by hacking.
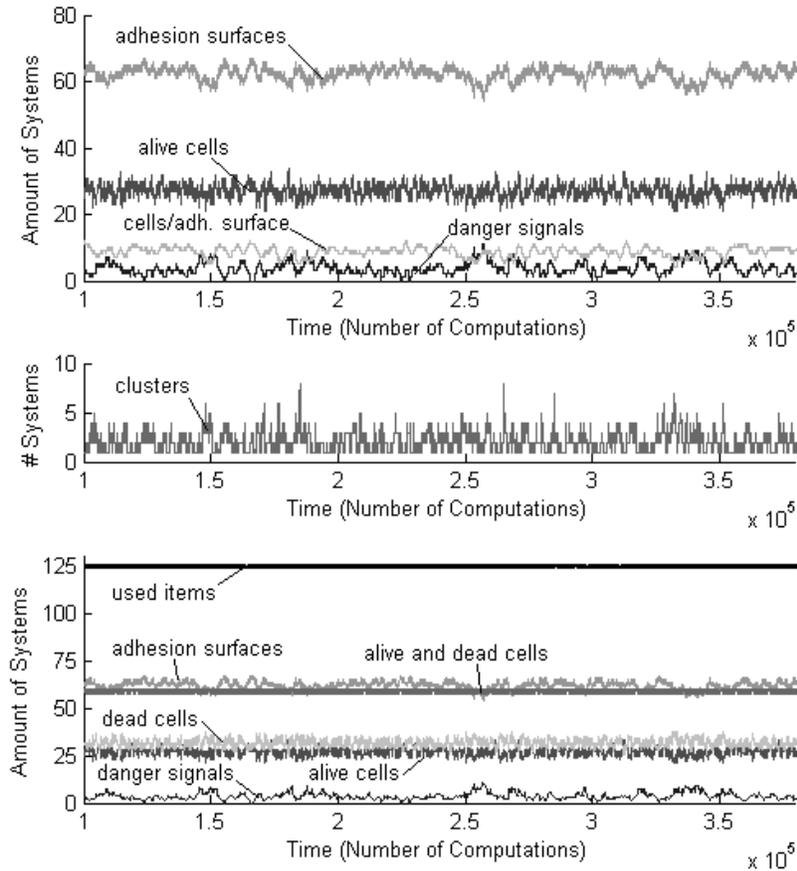
Fig. 5. Organism's inner organisation over a run of 5000 samples.

6 Conclusion

In this paper we introduced the notion of artificial metabolism using systemic computation to create an organism for clustering data that is suitable for an artificial immune system. This work is inspired by Matzinger's Danger Theory and uses the notion of danger signals. Starting from scratch and working on-line our organism is able to cluster data according to its similarities and can provide danger signals when cells die in an abnormal way for the current organism. Our organism proved to be able to detect anomalous UCI Breast Cancer data with better accuracy than in previous work [3]. The study of the evolution over time of our organism showed that its inner organisation reflects the data distribution of the current flow. Also, previous results have shown that SC programming is very robust, with programs easily showing fault-tolerance and self-repair abilities even when undergoing severe damage [16].

References

1. Aickelin, U., Greensmith, J.: Sensing Danger: Innate Immunology for Intrusion Detection. Elsevier Information Security Technical Report, pp 218–227 (2007)
2. Matzinger, P.: Tolerance, Danger and the Extended Family. Annual Reviews in Immunology, vol. 12, pp. 991–1045 (1994)
3. Bentley, P. J., Greensmith, J., and Ujjin, S.: Two Ways to Grow Tissue for Artificial Immune Systems. In Proc. of the Fourth Intl. Conf. on Artificial Immune Systems (ICARIS 2005), pp. 139–152. Springer LNCS 3627 (2005)
4. Bentley, P. J.: Systemic computation: A Model of Interacting Systems with Natural Characteristics. Int.J. Parallel, Emergent and Distributed Systems, vol. 22:2, pp. 103–121 (2007)
5. Breast Cancer Wisconsin (Diagnostic) Data Set, Creator: Wolberg, W. H., Donor: Mangasarian, O., UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/] (1992)
6. Tempesti, G., Roggen, D., Sanchez, E., and Thoma, Y.: A POEtic Architecture for Bio-Inspired Hardware. In Proc. of the 8th Intl. Conf. on the Simulation and Synthesis of Living Systems (Artificial Life VIII), pp. 111–115. MIT Press, Cambridge (2002)
7. Thoma, Y., Tempesti, G., Sanchez, E., Moreno Arostegui, J-M: POEtic: an electronic tissue for bio-inspired cellular applications. BioSystems, vol. 76, pp. 191–200 (2004)
8. Wallenta, C., Kim, J., Bentley, P. J., and Hailes, S.: Detecting Interest Cache Poisoning in Sensor Networks using an Artificial Immune Algorithm. To appear in Journal of Applied Intelligence, Springer (2008)
9. von Neumann, J.: The theory of self-reproducing automata. Univ. of Illinois Press, Champaign (1966)
10. Wolfram, S.: A New Kind of Science. Wolfram Media, Inc., Champaign (2002)
11. Holland, J. H.: Emergence, From Chaos to Order. Oxford University Press, Oxford (1998)
12. Adamatzky, A.: Computing in Nonlinear Media and Automata Collectives. Institute of Physics Publishing, Bristol (2001)
13. Arvind, D.K., Wong, K.J.: Speckled Computing: Disruptive Technology for Networked Information Appliances. In Proc. of the IEEE Intl. Symposium on Consumer Electronics (ISCE'04), pp. 219–223 (2004)
14. Le Martelot, E., Bentley, P. J., and Lotto, R. B.: A Systemic Computation Platform for the Modelling and Analysis of Processes with Natural Characteristics. In Proc of 9th Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 2809–2816. ACM Press, New York (2007)
15. Le Martelot, E., Bentley, P. J., and Lotto, R. B.: Exploiting Natural Asynchrony and Local Knowledge within Systemic Computation to Enable Generic Neural Structures. In Proc of 2nd International Workshop on Natural Computing (IWNC 2007), (2007)
16. Le Martelot, E., Bentley, P. J., and Lotto, R. B.: Crash-Proof Systemic Computing: A Demonstration of Native Fault-Tolerance and Self-Maintenance. In Proc of 4th IASTED International Conference on Advances in Computer Science and Technology (ACST 2008), ACTA press (2008)