Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2012, Article ID 703878, 20 pages
doi:10.1155/2012/703878

Research Article

Modelling Biological Systems with Competitive Coherence

Vic Norris, Maurice Engel, and Maurice Demarty

Theoretical Biology Unit, EA 3829, Department of Biology, University of Rouen, 76821 Mont-Saint-Aignan, France

Correspondence should be addressed to Vic Norris, victor.norris@univ-rouen.fr

Received 8 February 2012; Revised 16 April 2012; Accepted 24 April 2012

Academic Editor: Olivier Bastien

Many living systems, from cells to brains to governments, are controlled by the activity of a small subset of their constituents. It has been argued that coherence is of evolutionary advantage and that this active subset of constituents results from competition between two processes, a Next process that brings about coherence over time, and a Now process that brings about coherence between the interior and the exterior of the system at a particular time. This competition has been termed competitive coherence and has been implemented in a toy-learning program in order to clarify the concept and to generate—and ultimately test—new hypotheses covering subjects as diverse as complexity, emergence, DNA replication, global mutations, dreaming, bioputing (computing using either the parts of biological system or the entire biological system), and equilibrium and nonequilibrium structures. Here, we show that a program using competitive coherence, Coco, can learn to respond to a simple input sequence 1, 2, 3, 2, 3, with responses to inputs that differ according to the position of the input in the sequence and hence require competition between both Next and Now processes.

1. Introduction

The quest for universal laws in biology and other sciences has led to the development—and sometimes the acceptance—of concepts such as tensegrity [1], edge of chaos [2, 3], small worlds [4], and self-organised criticality [5]. This quest has also led to the pioneering (N, k) model developed by Kauffman in which N is the number of nodes in an arbitrarily defined Boolean network and k is the fixed degree of connectivity between them [6]. The actual use of the (N, k) model to the microbiologist, for example, is that it might help explain how a bacterium negotiates the enormity of phenotype space so as to generate a limited number of reproducible phenotypes on which natural selection can act. Although the (N, k) model successfully generates a small number of short state cycles from an inexplorable vast number of combinations—which might be equated to generating a few phenotypes from the vast number apparently available to the cell—the model has its limitations for the microbiologist as, for example, it has a fixed connectivity, it does not evolve, and it does not actually do anything.

In a different attempt to find a universal law in biology, one of us began working on the idea of network coherence in the seventies. This idea is related to neural networks (though the idea was developed with no knowledge of them) which have indeed been proposed as important in generating phenotypes [7]. The network coherence idea turned out to be scale-free and to address one of the most important problems that confronts bacteria, eukaryotic cells, collections of cells (including brains), and even social organisations. This problem is how a system can behave in (1) a coherent way over time so as to maintain historical continuity and (2) a coherent way at a particular time that makes sense in terms of both internal and environmental conditions. A possible solution would be for these systems to operate using the principle of *competitive coherence* [8, 9]. Competitive coherence can be used to describe the way that a key subset of constituents—the Active Set—are chosen to determine the behaviour of an organisation at a particular level. This choice results from a competition for inclusion in the Active Set between elements with connections that confer coherence over time (i.e., continuity) and connections that confer internal and external coherence at the present time. Given that living systems are complex or rather *hypercomplex* systems, characterised by emergent properties, we have speculated that competitive coherence has parameters useful for clarifying emergent properties and, perhaps, for classifying and quantifying types of complexity

[10]. We have further argued that competitive coherence might even be considered as the hallmark of life itself.

One of the objections to universal laws such as competitive coherence is that without testing they are mere handwaving. To try to overcome this objection and to make the central idea and related ideas clear, we have implemented competitive coherence into a toy-learning program in which the competitive coherence part, Coco, learns or fails to learn when playing against an environment that rewards or punishes Coco by changing connections between elements. In what follows, we describe the structure of the program, results from this toy version, and optional extras that could be added to or manipulated within the program. Some of these optional extras are wildly speculative but they are included to show that Coco might be useful as a generator, editor, and test-bed for new and/or woolly concepts including those that might find a home in biology-based computing [11].

2. Principle of the Program

The central idea is that an *active* subset of elements determines the behaviour of a system: the majority of elements are inactive. The members of this active subset, the size of which is fixed, are chosen by a competition between two processes, *Next* and *Now* (Figure 1). The active subset corresponds to those elements that have their addresses in the current line of the Activity Register. The subset of elements that are active at the same time constitutes the state of the system at that time and each line in the Activity Register says what state is at a particular time; consecutive lines correspond to consecutive states of the system. How is this state obtained? An overview of the program is given in Figure 2 to show the order of the essential subroutines, which are explained in detail in the following subsections. COMPUTE is the cyclical core of the system. INITIALISE sets up the initial conditions by filling at random the Now and Next fields (two separate sets of weightings) of each element with the addresses of other elements, and by filling at random the first line (which corresponds to the state of the system) of the Activity Register with addresses. INPUT provides one input at a time (by loading the address into the Activity Register of a specific element); these inputs are taken in order from a defined sequence. DOWNTIME prevents an element whose address has been loaded into the Activity Register from having its address loaded again within a brief time. MUTATE changes at random the contents of the Now and Next fields (in a sense, the weightings). LOAD is responsible for extracting and ranking the scores of the elements' Now and Next fields and then comparing these scores so as to decide which address to load to the Activity Register. LOAD is helped by REVERSE DOWNTIME, which stops an element likely to be useful later from being used too soon, and by COMPATIBILITY-EMERGENCE, which increases the probability that some groups of elements have their addresses loaded at the same time (corresponding to these elements being active together). REWARD-PUNISH routines detect and evaluate outputs and reward or punish the elements involved by strengthening or weakening the Now/Next links between them. The results are displayed every loop of the COMPUTE routine, which corresponds to a line in the Activity Register being filled (see the following).

2.1. The Biological Elements. An individual gene or neurone or another biological building block is represented in the program as an element. In the version presented here, there are a thousand elements. Only ten elements (again in the version presented here) can be active—that is, determine the state of the system—at any one time. The composition of this subset of active elements determines whether this active subset is successful or not.

Each element contains, in the version presented here, two fields: Now and Next (Figure 3). Each element has an address. Each field contains the addresses of other elements with which the element has been associated successfully during learning (see the following). The Now field contains the addresses of those elements that have been successful in the same time step in which the element itself was successful whilst the Next field contains the addresses of those elements that have been successful in the time step following the step in which the element was active. A time step corresponds to a single line in the Activity Register, hence the active subset of elements corresponds to those elements that have their addresses in the current line of the Activity Register. In other words, each line says what the state of the system is in terms of *activity* in that particular time step.

2.2. The Activity Register. Each line in the Activity Register contains the addresses of the small subset of elements that are active at a particular time (Figure 4). Each line is filled according to a set of rules (see the following), and when the register is full, the first line is filled again (so the register operates cyclically). A line is set to zero before it is filled.

2.3. Coherence via the Now Process. Biological systems are characterised by coherence. At the level of human society, the composition of a successful football team reflects the importance of coherence in the manager's choice of players: the team must contain players who can take on the roles of goalkeeper, defenders, midfield players and attackers. Choosing the players is a progressive process: the choice of the goalkeeper influences the choice of the defenders which then influences the choice of the defenders. At the level of a bacterium, the composition of a bacterium reflects its growth strategy: if *Escherichia coli* is to grow rapidly, it needs to express the genes that encode ribosomes, tRNA synthetases, RNA polymerases, and the proteins that drive the cell cycle whereas if it is to survive in harsh conditions and in the absence of nutrients, it needs to express the genes that, for example, encode proteins that compact and protect its genome such as Dps [12]. Biological systems from sports' teams and brains to bacteria that fail to achieve this *internal* coherence risk elimination in a world in which natural selection operates. The coherence that a biological system must achieve also has an *external* component. The football manager may choose his team as a function of the

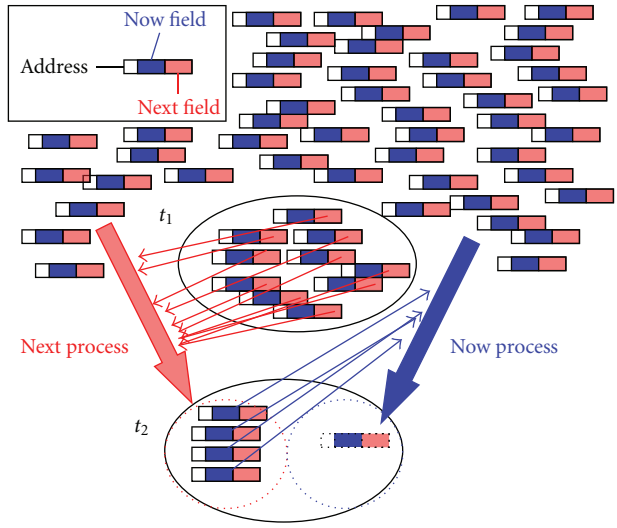


Figure 1: The principle of Now-Next competition. The elements active at time t_1 (inside black ellipse) are used to select the elements to be active at time t_2. This selection uses the Next fields of the elements active at time t_1 (red arrows). As the new elements are activated at time t_2 (inside the red dotted circle), the Now fields of these new elements are also used to select and activate more elements (blue arrows) such as the element inside the dotted blue circle. The elements to be activated are selected from a large set of inactive elements by a competition between the Now and Next processes. The inset shows an element which has an address and two fields.

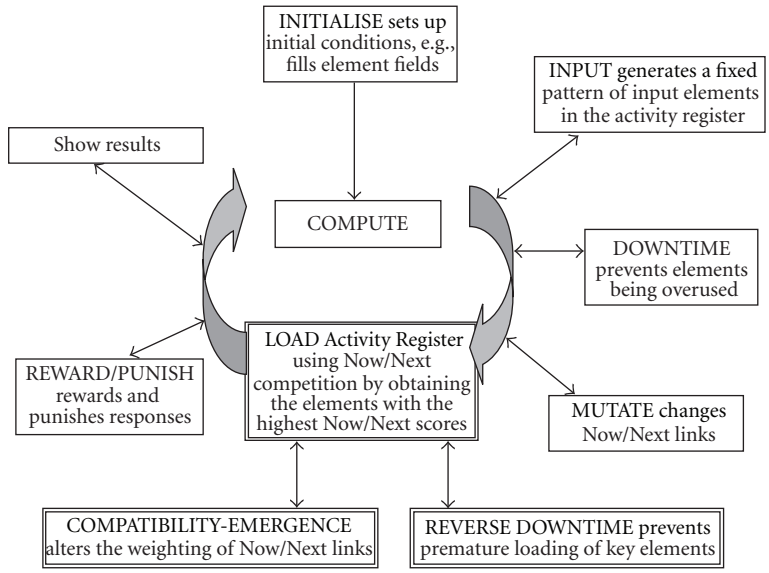


Figure 2: Overview of essential modules. A single cycle of the COMPUTE routine results in the addresses of a subset of the elements being loaded into a line of the Activity Register; this corresponds to *activating* these elements. Certain elements are inputs and others are outputs. The INPUT routine creates an input by loading the address of an input element into the Activity Register. The learning part of the program, Coco, eventually responds to an input by loading the address of an output element into the Activity Register; this corresponds to an output. The actual loading of addresses results from a competition between Now and Next links; the scores obtained from counting these links may be modified by an EMERGENCE routine. The DOWNTIME and REVERSE DOWNTIME routines may make certain elements ineligible for loading, depending on the history of these elements. Outputs are detected, evaluated, and rewarded or punished by REWARD/PUNISH routines. Each time a line of the Activity Register has been filled, the results are displayed.

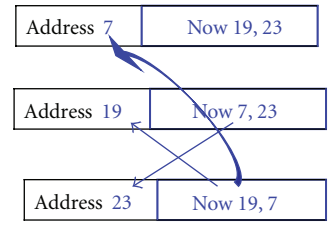
pitch, the weather, their position in a league table, and the opposing team (which may contain players known for their "physical" approach). The bacterium should not express the full complement of genes needed for fast growth if it is in conditions in which there are few nutrients and in which physical conditions such as temperature and humidity require stress responses. The Now process is intended to represent the way biological systems achieve both internal and external coherence. For example, suppose that at high temperatures, phospholipids with long, saturated fatty acids

Address	Now field					Next field				
1	878	63	50	533	119	43	227	136	19	256
2	13	218	516	923	121	724	447	225	134	15
3	611	577	488	818	63	70	337	722	297	98
4	437	316	218	726	413	124	126	117	747	299
...										
999	165	534	183	612	626	667	879	355	649	220
1000	388	431	566	435	924	654	255	818	921	33

Figure 3: The Elements table. The Now and Next fields of an element contain the addresses of the elements with which the element in question has been successfully active (for convenience of representation each field here contains only 5 addresses).

Present state t	10	5	18	11	6	4	23
Developing state $t+1$	7	22	*16*				

Figure 4: The Activity Register. Two lines are shown (corresponding to two successive active states), one full and the other (italics) in the process of being filled (for convenience, only 7 entries and only two lines are shown).



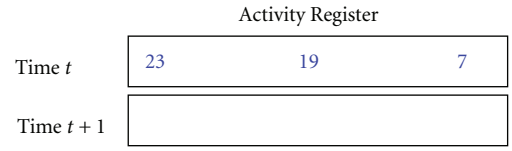


Figure 5: The Now Process. Out of a large set of elements (genes), a few, well-connected ones are chosen (positioned) to be active (expressed) at a particular time. For example, if gene 23 encodes a protein with the same lipid preferences as the proteins encoded by genes 19 and 7 the expression of these genes may be synergistic (for convenience, only two addresses are shown in the Now field).

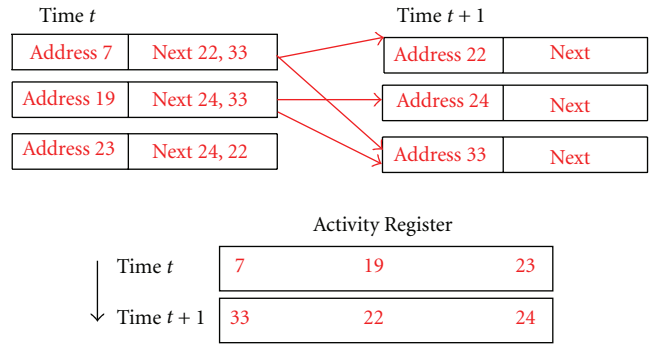


Figure 6: The Next Process. The elements (genes) active in time step t determine the elements that will be active in time step $t+1$.

are needed to confer stability to the membrane; suppose that genes 7, 19, and 23 encode membrane proteins with an affinity for such phospholipids (Figure 5); the expression of one of these genes (e.g., 23) may help to create a domain on the membrane enriched in both the protein encoded by 23 and the long chain phospholipid; the existence of this domain then helps the expression of 19 which has a protein product that also contributes to the size and stability of the domain; this in turn helps the expression of 7 (for the possible biological significance see [13]).

2.4. Coherence via the Next Process. The Now process on its own is not enough to allow a biological system to adapt effectively to its environment. This is because these environments often require different responses to the same stimulus because the historical contexts are different. One may not respond in the same way to an invitation from Human Resources following a successful sales campaign as following the news that the company is about to go out of business. A Next process is required to take account of the dependency of a correct response on the history of the system. This is made easier by the fact that environments are often unchanging for long periods and, when they do change, change in predictable ways; rules in sport, for example, do not change very often. And even when different environments exist at the same time, it would not be advisable for a sports' team that wants to compete at a good level for it to play football one week, switch to hockey the next, then rugby, and so forth. It would not be a winning strategy for a bacterium to switch phenotypes either (we ignore here events at the level of the population of bacteria which needs to explore phenotypic heterogeneity). For example, three genes that are active in one time step, 7, 19, and 23, may have connections via their Next fields to the same subset of genes, 22, 24, and 33 (Figure 6) with 7 encoding a transcriptional activator being produced in one time step and 22 and 23 being the genes under its control and hence expressed in the following time step.

2.5. Competition between the Now and Next Processes. Modelling competitive coherence in the program consists of the competition between the Now and Next processes for choosing the subset of elements that are to be active (i.e., determine the state of the system). This activation takes the form of loading the addresses of elements into the new line of the Activity Register. The competition is on the basis of the scores of the elements. To load a new line, first, the Next fields of the elements present in the current line are consulted and the number of occurrences of the addresses in these fields is counted to give Next Scores (Figure 7). These Next Scores are then ranked in HighestNext to give, in Figure 7, element 7 with the highest score (of 8).

Once the address of this first element has been loaded into the new line, the following highest score in the

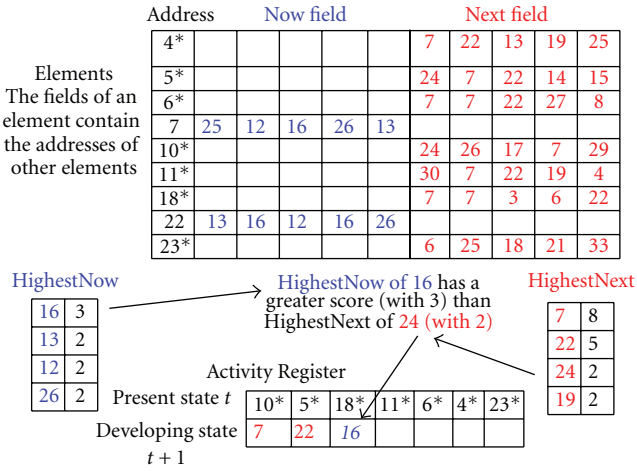


FIGURE 7: Competition between Now and Next Processes. The diagram shows the operation of filling the third position in the new line of the Activity Register (corresponding to the developing state of the system). The elements in the current line of the Activity Register are labelled with an asterisk and only their Next fields are shown. The two elements loaded into the new line at the start of the competition for third position have their Now fields in bold; their Next fields are not shown. The Now and Next scores are in the second column of HighestNow and HighestNext, respectively. See text for full explanation.

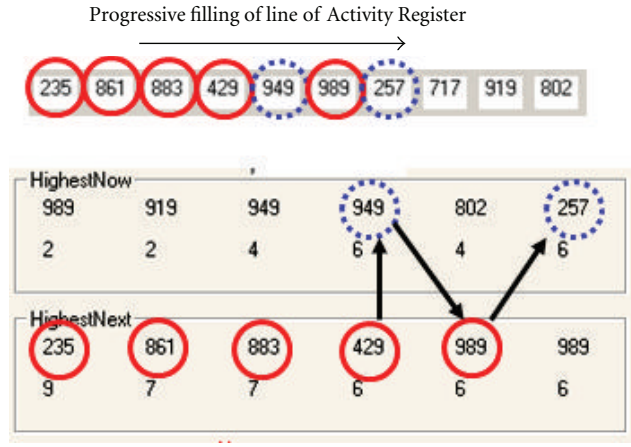


FIGURE 8: Differential Contributions of the Next and Now processes. A line of the Activity Register that has just been filled is shown along with part of the contents of the HighestNext and HighestNow counters at this time. The addresses in these counters are on the top and the corresponding scores on the bottom. The red circles show the addresses of elements contributed by the Next process and the dotted blue circles show those contributed by the Now process. The arrows show where there is a clear change in the contributions of the processes to filling the Activity Register.

HighestNext list is consulted: this is 22 with a score of 5. It is not, however, loaded straightaway because 7 has been loaded and 7 has a Now field. The addresses of the elements in 7's Now field are counted and ranked in HighestNow; in this example, the ranking is at random because the scores of these addresses are all the same (here, 1). The highest scores in HighestNext and HighestNow are then compared and the address of the element with the higher score is then loaded into the Activity Register (if they have the same score, the Now element is preferred).

There are now two addresses in the Activity Register, 7 and 22. The addresses in the Now fields of both 7 and 22 are counted and ranked to give at the top of HighestNow 16 with a score of 3. The address at the top of HighestNext is 24 with a score of 2. Comparison of the two scores shows that the address of element 16 has the greater score and this address is therefore loaded to the Activity Register. Once an address has been loaded, its score is set to zero to prevent it from being loaded a second time.

This competition between Next and Now processes continues until the line of the Activity Register has been filled. Note that, during the determination of the new state of the system, the relative contribution of the two processes changes since, first, the number of Now fields increases as addresses are progressively loaded into the Activity Register (i.e., as more and more elements become active), and, second, each time an element is chosen from the HighestNext, the following one has a lower score. In other words, the Next process dominates at the beginning and the Now process at the end (Figure 8).

2.6. Inputs and Outputs. Both inputs and outputs correspond to specific elements each of which has a Now and a Next field. In this version, there are three inputs, 1, 2, and 3, and three outputs, 998, 999, and 1000. An input is generated when the INPUT subroutine inserts one (and only one) of the three inputs into the new line of the Activity Register. An output is generated when Coco loads an output element into the new line of the Activity Register. The lines between the input line and the output line constitute an *input-output module*.

A new input is generated in the line immediately following an output. A new output must occur in the four lines following an input; if an output is not generated in these time steps, an arbitrarily chosen output is inserted into the fourth line. A maximum number of lines before forcing an output is needed if the program is to run rapidly when the proportion of output elements to the total number of elements becomes small (since the initial probability of generating an output is similarly low). The choice of four lines as a maximum is also arbitrary.

Inputs are not related to outputs: it does not matter whether the output is right or wrong. The sequence of inputs is fixed: $(1, 2, 3, 2, 3)_n$. Elements in lines containing outputs that correspond to inputs are rewarded—or punished if the outputs are wrong (see the following). Coco is considered to have succeeded when it has learnt to generate $(1000, 999, 999, 1000, 998)_n$ in response to the input sequence (Figure 9). This simple input-output relationship was chosen because neither the Now nor the Next process alone is sufficient to lead to Coco learning. The Now process fails because two different outputs—999 and 1000—are required when 2 is the input (and also because two different outputs—999 and 998—are required when 3 is the input).

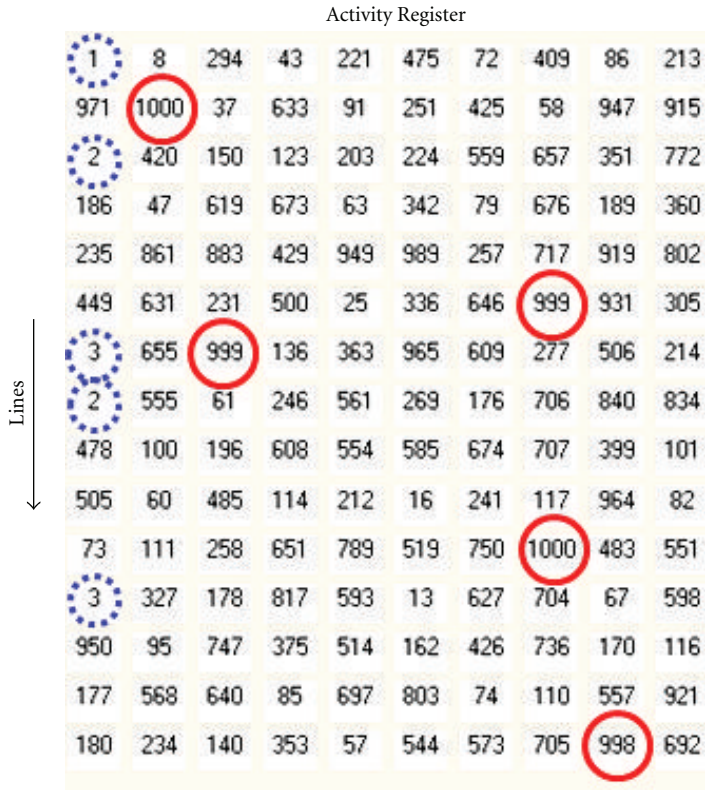


Figure 9: Contents of Activity Register after learning. The screen print shows consecutive lines in the Activity Register. Inputs are circled with dotted blue lines and outputs with continuous red lines.

The Next process fails because two different outputs—999 and 998—are required when 3 follows 2.

2.7. Downtime. An element that has been active (i.e., its address has been loaded into a line of the Activity Register) cannot be activated again for ten more timesteps. Inputs that are generated by the environment subroutine are not affected by downtimes (artefactual inputs generated by the dynamics of Coco do have downtimes). Outputs do not have downtimes. Downtimes are taken into account when the Now and Next scores are worked out. Downtime can be problem when the size of the Activity Register is increased since Coco can run out of elements to load; for example, if there are only 1000 elements, if the Activity Register has a line containing 100 elements, and if Downtime is set to 10, there are not enough elements to load. There is an echo here of the *E. coli* cell cycle in which, after initiation of chromosome replication, the constituents of the initiation hyperstructure are inactivated or sequestered to prevent a second initiation event [14], whilst after cell division, the constituents of the division hyperstructure are presumably also disabled to prevent repeated divisions in the bacterial poles [15].

2.8. Rewarding and Punishing. Rewarding entails strengthening the connections between successful states (and series of successful states). Briefly, this is achieved by (1) taking an element with its address in a line of the Activity Register between and including the input and output lines, which we term a module (see the previous part) and (2) writing this address into the Now or Next fields of other elements in the same or in the preceding lines of the Activity Register (Figure 10). First, the environmental part of the program detects whether there is an output in the new line and, if so, decides whether there is more than one output (more than one output is punished, see the following). Then, if the output is correct, two addresses are chosen from the same line in the successful module and the second address is written into the Now field of the first element. This is done for every element in the entire module (i.e., each line is treated). These connections are not made when it would entail connecting an element to itself (i.e., self-referral) or overwriting the address of another element that is actually already in the same line. The Next connections are rewarded in much the same way except that elements are taken from one line of the module and the addresses that are written into their Next fields are taken from the following line. Self-referral and overwriting of successful elements is avoided as in the case of rewarding via the Now connections. It should be noted that a connection is made between the previous module and the rewarded module insofar as the Next fields of the last line of the previous module are connected to the first line of the rewarded module (note too that the Next rewarding has to stop at the penultimate line of the rewarded module because the future line is not

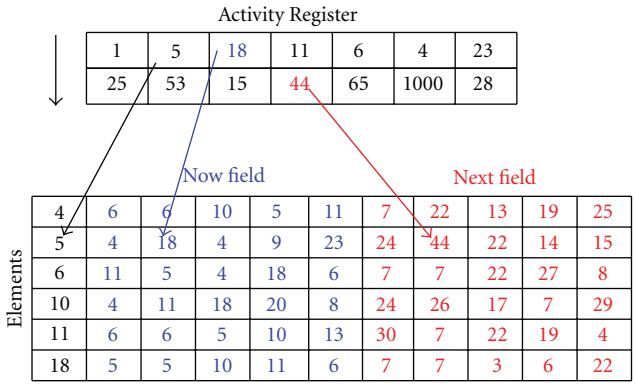


Figure 10: Rewarding by overwriting. The connections of element 5, whose address is in a line of a successful module in the Activity Register, are strengthened (black arrow). A randomly chosen address in the Now field of 5 is overwritten with 18, which is another address present in the same line as 5 (blue arrow), whilst a randomly chosen address in the Next field of 5 is overwritten with 44, which is an address present in the following line (red arrow).

known!). A small section of the elements, along with a couple of lines of the Activity Register and the HighestNow and HighestNext registers (Figure 11) shows the pattern of connectivity resulting from learning to couple an input with the appropriate output for Coco with a 1000 elements, Anumber of 10 (size of Active Set) and Knumber of 7 (size of Now and Next fields).

Punishing entails taking a *single* Activity Register line at random from the failed input-output module or taking the last line of the previous module. The Now fields of the elements whose addresses are in this line are then consulted. A randomly chosen address is then used to overwrite one of these Now addresses. The Next fields of these elements (which helped determine the following line) are altered in a similar way. There is a mutation aspect to punishment (Section 4.3).

2.9. Synchrony. There is a synchrony in the program that results from it being based on successive lines in the Activity Register, each of which is examined in a time step. Reward and punish decisions are then made in this time step to change the connections between the elements by altering the contents of their Now and Next fields; these alterations are made together in a synchronous fashion. The actual loading of addresses into a new line is a mixture of synchronous and asynchronous events: the Next scores are obtained together at the end of one time step (and the process exhibits synchrony) whilst the Now scores are obtained progressively during the loading of the Activity Register (and the process exhibits asynchrony).

3. Results

3.1. With Next Alone. The relative contributions of Now and Next scores to the loading of addresses into the Activity Register can be altered by a constant factor, NowNextWeighting. If this factor is set very low or very high, it switches the program so that it operates with either just Nexts or just Nows. Generally, this factor equals 1. By setting NowNextWeighting to 1/100, the Next scores dominate. The graphs show the fraction (total outputs-correct outputs)/total outputs, so a line towards the top of the figure corresponds to a failure to learn effectively. When the Nexts alone determine the loading of the Activity Register (red circles), learning does not occur (Figure 12). Note that only a truncated part of HighestNext and HighestNow is shown and that the scores shown in the bottom row are before scaling by NowNextWeighting.

3.2. With Now Alone. Setting NowNextWeighting to 100 allows the Now scores to dominate. The graphs in Figure 13 show the proportion of incorrect outputs (failures to learn) to total outputs, so a series of successful outputs corresponds to a negative slope. In one case, learning has not occurred after 20000 timesteps (Figure 13(a)). In another case, learning has actually occurred at a late stage (Figure 13(b)). The explanation is that the Next scores are still operating even with this NowNextWeighting because the first address chosen for the NewLine of the Activity Register is chosen from the Next element with the HighestNext score (unless an input is loaded). This initial choice does not depend on the NowNextWeighting. The Activity Register corresponding to this surprising learning confirms that the Nows have contributed the addresses (compare contents of last line of Activity Register and with contents of HighestNow and HighestNext in Figure 13(c)).

3.3. Without Downtime. The address of an element that has loaded into the Activity Register cannot be loaded again within the next ten timesteps. In the absence of DOWNTIME, Coco does not learn (Figure 14(a)). One evident reason for this is that false inputs can be loaded into the Activity Register (Figure 14(b)).

3.4. With Now, Next, and Downtime. Three independent runs of the program show that Coco learns the task albeit sometimes with difficulty (Figure 15). Inspection of the Activity Register and the HighestNext and HighestNow confirms that the initial contribution to the last line in the register comes first from the Next (red circles) and then from the Now (blue circles). Note that only a truncated part of HighestNext and HighestNow is shown. Note too the limited size of the Activity Register (Section 4.14). What might DOWNTIME correspond to in a biological system? In a bacterium, it might correspond to the state of a gene x in an operon which requires an activator but which also encodes an unstable repressor Y of this operon; activating the gene would then lead to both production of X and of Y; Y would then switch off the operon for a Downtime until it was degraded. A more interesting possible example is that of the sequestration of newly replicated, hemimethylated DNA in *E. coli* (which occurs when the SeqA protein recognises that a GATC sequence in the old strand is methylated whilst its complement in the new strand has yet to be replicated); since a gene may have a greater chance of being expressed when

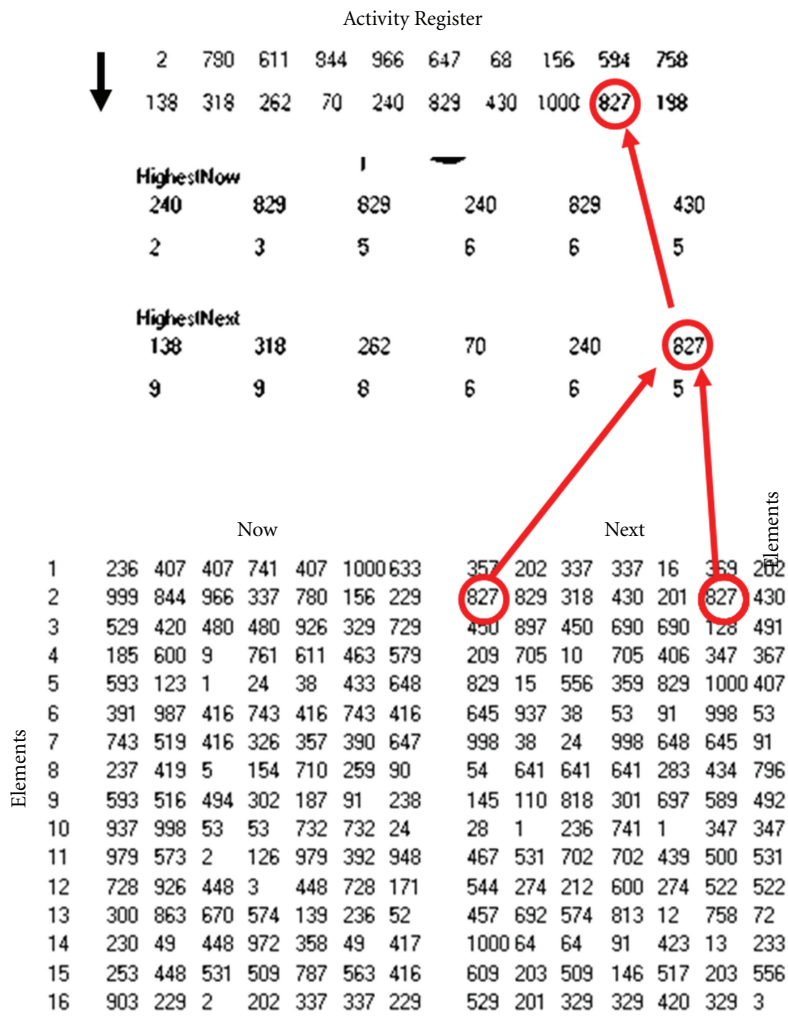


Figure 11: The connectivity after successful learning. An input line and the following, correct, output line in the Activity Register are shown. The total scores of some of the elements whose addresses are in the Next fields of the elements in the top line of the Activity Register can be seen in the bottom row of HighestNext. If the Next field of element 2 is inspected, two references to element 827 are found; these two references are part of the total of five references to element 827; this score of five is sufficient for the address of element 827 to be loaded into the new line of the Activity Register (so activating element 827). Note the presence of the output itself, 1000.

the region within which it lies is being replicating (perhaps due to greater accessibility to RNA polymerase), some genes with GATC sequences may be both switched on and switched off by the act of replication. This latter possibility might be modelled specifically by confining DOWNTIME to those elements that are activated by the CYCLE subroutine, as mentioned in Section 4.2. What then might the inputs and outputs correspond to in a biological system? As shown here, Coco readily learns to give the same output to different inputs. It also learns to give a different output to the same input depending on the history of the inputs; this could correspond to a low concentration of nutrients having a different meaning for a bacterium if this concentration follows a period of starvation or a period of plenty; in the former case it means that conditions are improving and in the latter case it means that they are getting worse, and the appropriate response of the bacterium would be to grow or to sporulate, respectively.

3.5. An Oscillatory Input Gives an Oscillatory Output. The environment gives the inputs (1, 2, 3) in a cycle $(1, 2, 3, 2, 3)_n$ and immediately Coco responds with an output the environment gives the next input. Hence, Coco learns to respond to an oscillating input pattern with an oscillating output pattern. This can be confirmed after learning has occurred by removing inputs and following the pattern active elements (i.e., the addresses in the Activity Register). It does indeed maintain the output pattern in the absence of inputs and absence of changes to connectivity normally caused by rewarding, punishing, and noise (data not shown). This shows the extent to which the state of the system reflects the interdependency of the elements loaded to the Activity Register. Note though that if the Activity register is small (low Anumber) and the connections are weak (low Knumber), inputs must be continued (along with rewarding) to maintain the oscillating output pattern. We discuss further the significance of

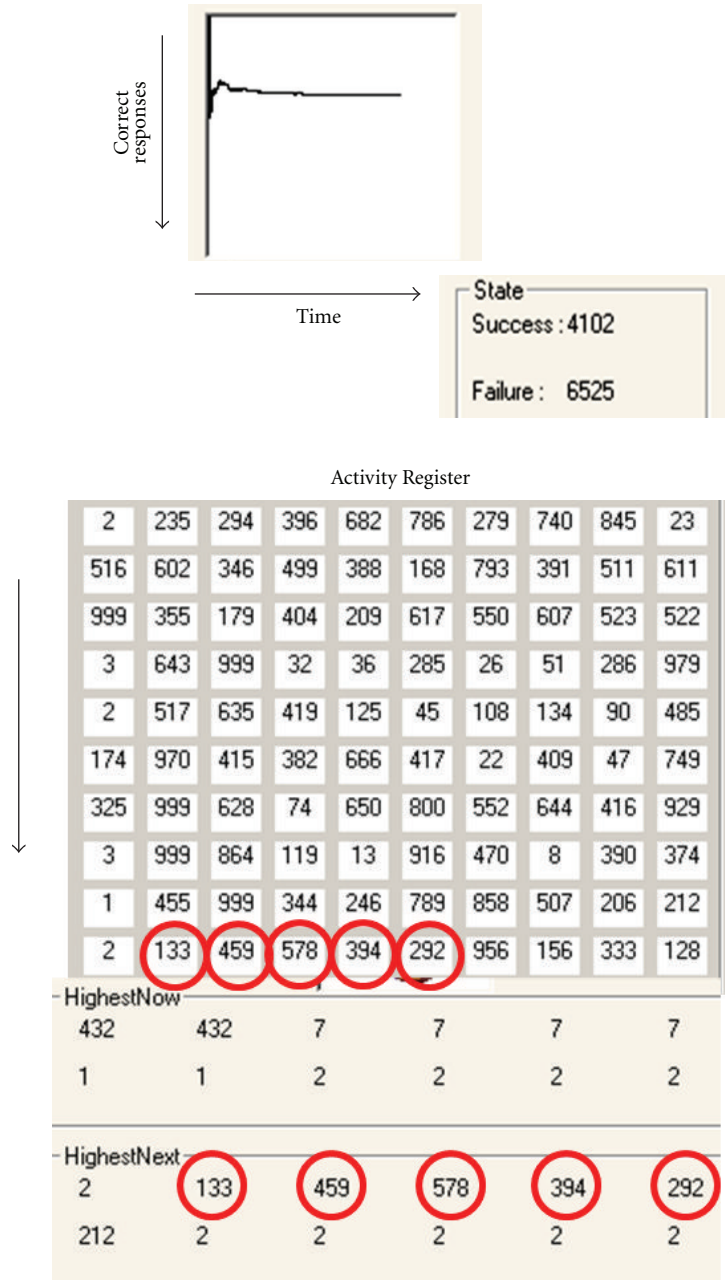


Figure 12: Running the program depending on the Nexts. No learning has occurred after 20000 time steps (the time taken to fill, analyse, and respond to a single line in the Activity Register); the contents of the Activity Register and the HighestNow and HighestNext confirm the Next contribution (red circles). Correct responses is (total number of responses–correct responses)/total number of responses. Note that the total number of successes and failures (10627) would only equal the number of timesteps or lines (here 20000) if the program were constrained to give an output in the same line as it had received an input (rather than up to four lines later).

a noncyclical input-output pattern—and how to achieve this—in Section 4.18.

4. Optional Extras

4.1. Positive and Negative Links. Biological systems generally have both activators and repressors. Simulation suggests that the ratio between them is a major influence on the dynamics [16]. In the program, elements can be connected positively and negatively. When the Now and Next scores are calculated, each address present in a field with a positive link counts as +1 whilst an address with a negative link counts as −1. The positive or negative nature of a link is defined at random at the start of the program and is not modified afterwards. In the present version, 10% of the signs are negative (though it learns when none of them are negative).

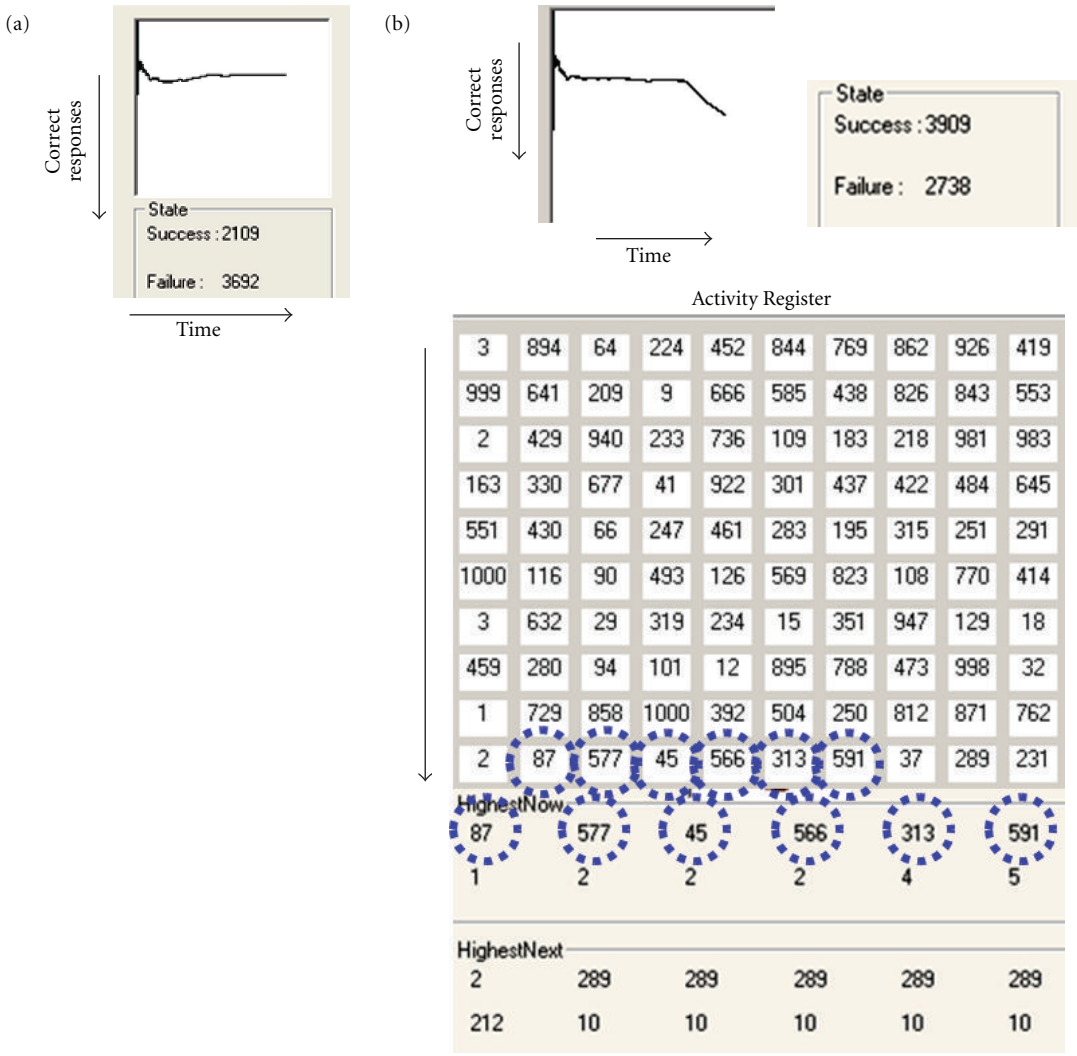


Figure 13: Running the program depending on the Nows. (a) No learning has occurred after 20000 timesteps, and (b) Learning has been delayed in this case after 20000 timesteps and the contents of the Activity Register and the HighestNow and HighestNext confirm the Now contribution (dotted blue circles).

4.2. DNA Replication. Growing bacteria replicate their DNA. It has been proposed that the replication of a gene affects the probability that the gene and its physical neighbours may have an altered probability of transcription [17, 18]. One consequence of this could be to enable the bacterium to avoid getting trapped in a very limited state cycle of phenotypes. In other words, DNA replication itself might constitute a way of exploring phenotype space. Such exploration could even constitute a coherent exploration of phenotype space if the position of the gene on the chromosome were close to those of other genes with related functions (and far from those with opposed functions). To introduce this parameter into the program, a CYCLE subroutine allows an element to be loaded into the Activity Register irrespective of the Now and Next connections. This element is chosen in order from the elements (e.g., first 17 is inserted, then 18, then 19, etc.). It is not inserted into the Activity Register if Coco has just given the correct output (which corresponds in this version of the program to CyclePermission = 0).

4.3. Mutations and Noise. Mutations occur as part of the PUNISH subroutines. In fact, the overwriting of the Now and Next fields (of the elements with addresses in the line to be punished) is a mutation process insofar as the new addresses that are written into these fields are chosen at random. This overwriting is done at a frequency determined by the MutationThreshold which, in the version presented here, is set so that overwriting occurs on one out of ten occasions.

There is more to the mutation story than this though. After the program has run for 200 timesteps, a RunningScore is kept of how many of the last ten outputs have been correct (this involves a sliding window, RunningScoreWindow, set to ten). Depending on this RunningScore, mutations are either made at different frequencies or not made at all. A mutation is made by taking an element with an address that is rarely found in the fields of the other elements (i.e., a lonely element) and writing it into a field of any one of the other elements chosen at random. This is done ten

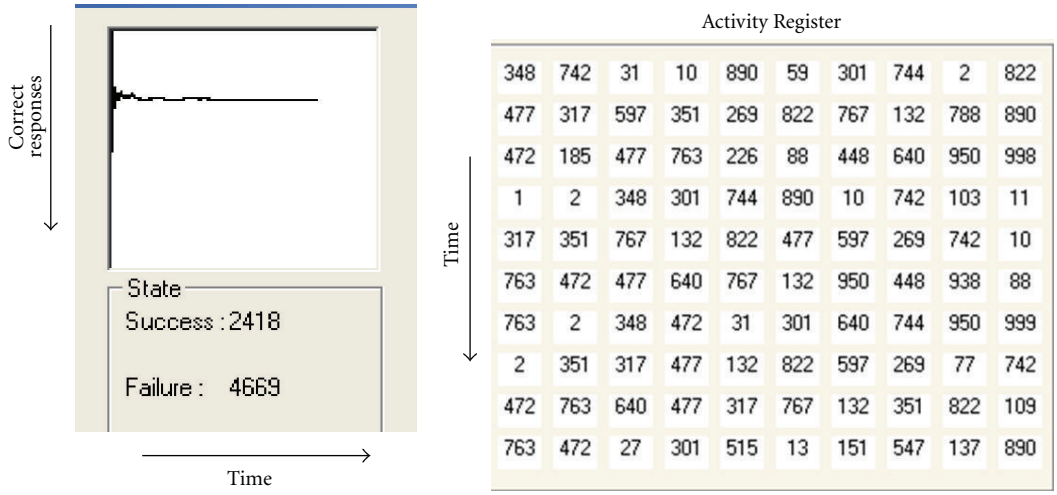


Figure 14: No learning without DOWNTIME. (a) After 20000 timesteps, Coco shows no sign of learning. (b) The Activity Register shows that inputs are being generated inappropriately.

times per output if the RunningScore is low. If the last ten outputs were all correct, no mutations are made. Clearly, RunningScoreWindow is a parameter that the program itself could modify during running but, in the version presented here, it is held constant at ten.

Noise is not present in the basic version of the program presented here but is easy to introduce. For example, in Figure 16 the NoiseLevel has been set to insert a random address for every twenty or so (on average) addresses loaded into the Activity Register (i.e., around every three lines). The results show that the learning displayed by Coco is fairly robust and, even when it is forced to *forget*, it can relearn rapidly.

Finally, a SCRAMBLE subroutine may be a source of *mutations*. This routine operates in a democratic way to ensure that when elements have the same score, they are chosen at random to be ranked in HighestNow and HighestNext. This may result in an address being loaded into a position in the Activity Register that it had not occupied previously and bring to an end a winning streak, at least, temporarily. The fact that learning is often stable despite scrambling is again indicative of the robustness of this learning.

4.4. Reverse Downtime. Input-output modules can readily become compressed so that, for example, an input in line n of the Activity Register followed by an output in line $n + 4$ can be shortened such that the output occurs in line $n + 2$ or indeed in line n itself (Figure 17). The idea behind REVERSE DOWNTIME is to avoid this compression by preventing the addresses of elements in line $t + 1$ from being loaded into the previous line t. This entails using the REVERSE DOWNTIME subroutine to examine progressively the elements of addresses loaded into the NewLine and ensuring that the addresses in their Nexts, which may correspond to elements often active in the following line, are not loaded via the Now process. Preventing such addresses from being loaded is done via the NOW EXTRACTION subroutine which sets their scores to zero (in this version, it is not done by the NEXT EXTRACTION subroutine too—but could be). Although it is not clear to us that the REVERSE DOWNTIME subroutine has an equivalent in cells, it might be argued that REVERSE DOWNTIME resembles checkpoints, which prevent late cell cycle events from occurring until earlier ones have been completed [19].

4.5. Uptime. It would be easy to introduce an uptime in which an element that had already participated successfully would have a greater chance of being loaded again. This would be a variant of the Matthew effect in which the rich get richer (and the poor, poorer), which has been explored in connectivity studies [20].

4.6. Emergence. One of the characteristics of an emergent property is that it resists attempts to predict or deduce it [21]. An emergent property could be, for example, the affinity of certain membrane proteins for the phospholipid, cardiolipin, so that they assemble into a membrane domain enriched in cardiolipin where these proteins then function together. In the framework of competitive coherence, emergence is related to the formation of the new state, the subset of elements that are active together because their addresses are loaded together into the Activity Register [10]. Suppose that a subset of the elements (e.g., 27, 37, 47, 57, 67, and 77) correspond to proteins with a strong affinity for cardiolipin. Similarly, another subset of elements (e.g., 23, 33, 43, 53, 63, and 73) might correspond to proteins with an affinity for another phospholipid, phosphatidylethanolamine. This could be done for a hundred different phospholipids. In the program, these affinities could correspond to a hardwiring done in the INITIALISE subroutine such that the probability that 27 and 37 and so forth are loaded into the Activity

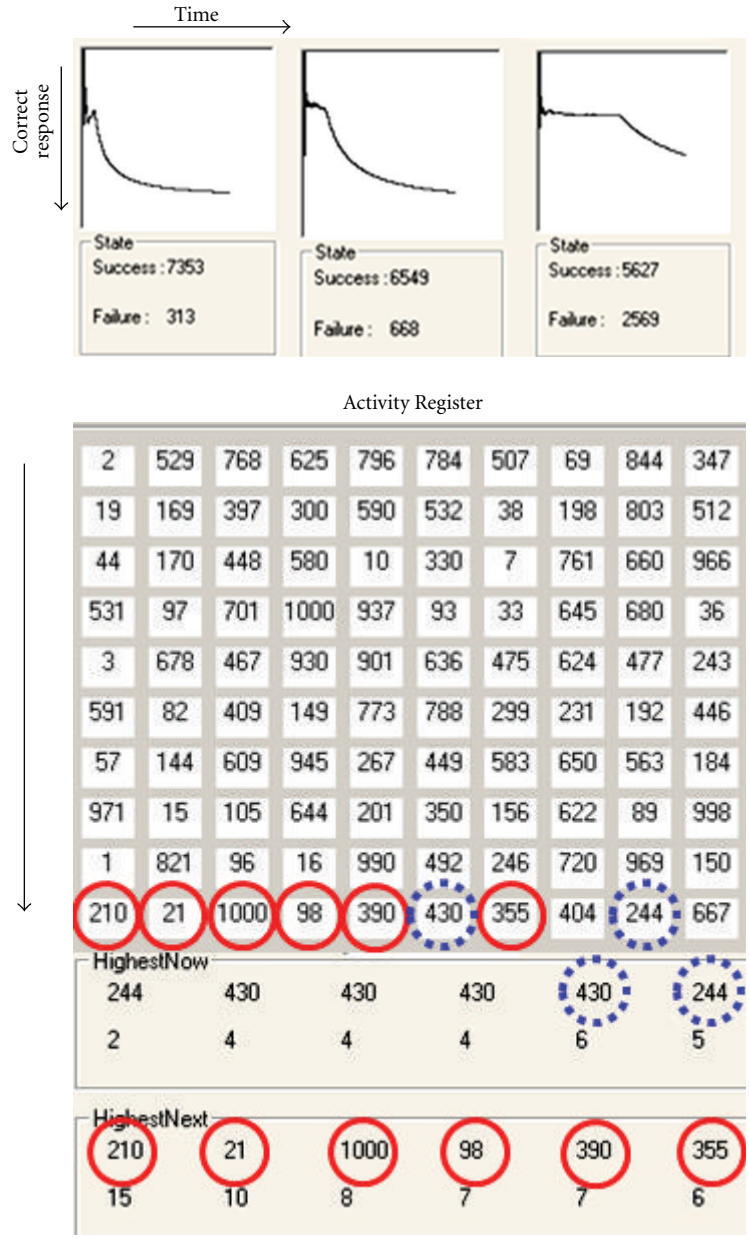


Figure 15: Successful learning with Now, Next, and Downtime processes working. Three, consecutive, independent examples are shown after 20000 timesteps. The contents of the Activity Register, HighestNext, and HighestNow of one of them are shown (now addresses are circled with a dotted blue line and Next addresses with a red line).

Register is greater if one of them is already present. If this combination of elements turns out to be a successful one, this might be considered as the emergence of the property of an affinity for cardiolipin.

The aforementioned approach to emergence can be modelled to some extent in the program via a Compatibility Table, which is a 2D matrix of the elements in which the Compatibility of Element(i) $\times$ Element(j) is a factor. During the loading of the Activity Register, this factor is used to multiply the Now and Next scores of the elements so as to help determine which is the highest. The factor in (i, j) is determined once and for all at the start of the program (i.e., it is hardwired). It would be possible to have a large number of sets of factors (e.g., a hundred sets of factors each linking ten elements) and then explore the effect—for example, it might significantly reduce the combinatorial space. In the present version, all the factors in the Compatibility Table are set to 1 (so they have no effect) with the exception of the inputs with one another which are set to zero (e.g., Compatibility(2,3) = 0).

4.7. Global Changes via Yin-Yang. A bacterium like *E. coli* can be exposed suddenly to an environmental change that

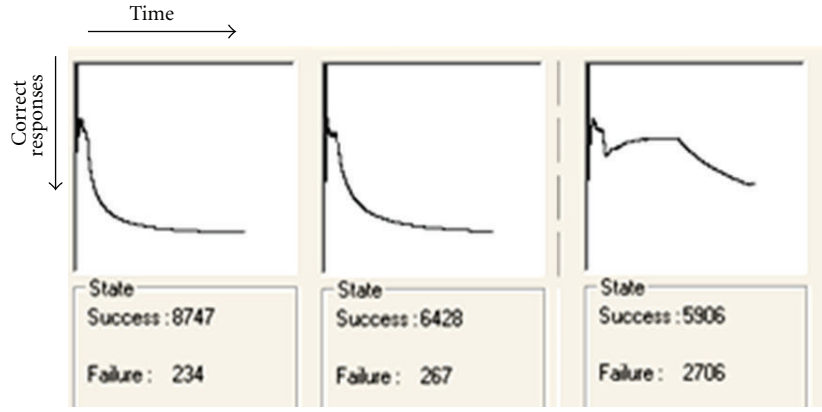


FIGURE 16: Robustness to noise. Three separate runs of the program, each for 20000 timesteps, show how allowing noise has in the third run perturbed Coco twice and caused it to forget a previously learnt sequence.

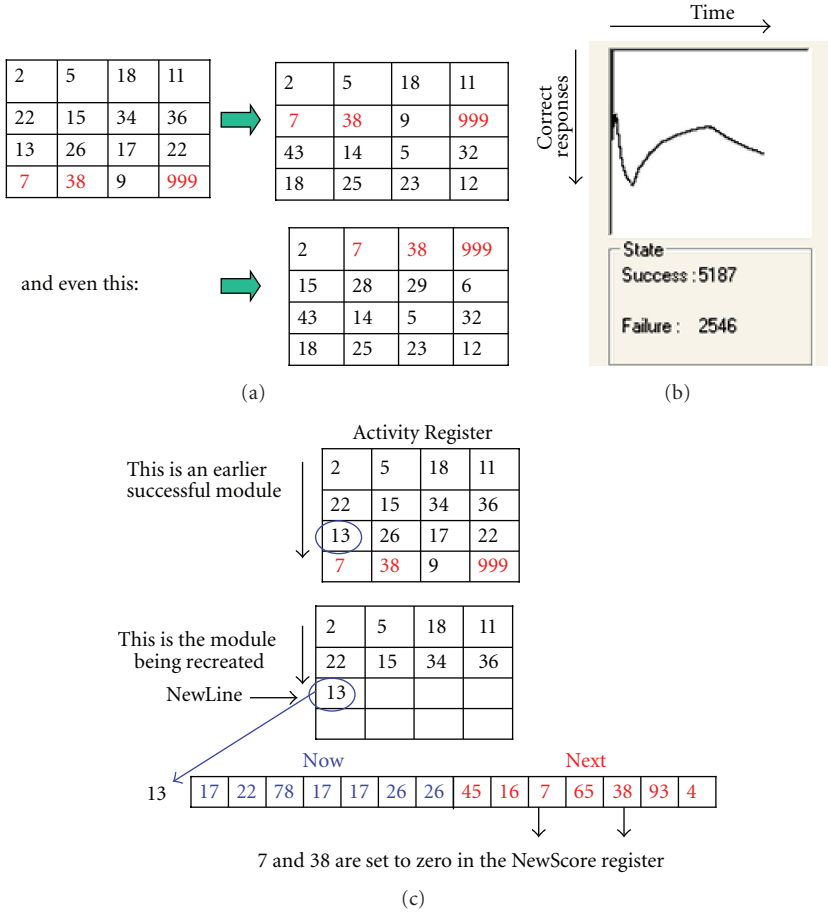


FIGURE 17: Principle of REVERSE DOWNTIME. Without this rule to reduce the compression of input-output modules, addresses can move from one line to another line closer to the input line. (a) The Activity Register is shown here with only four elements (for simplicity) and initially has addresses of certain elements (red) in a successful output line; these addresses can become displaced towards the input line. (b) The results of 20000 timesteps without REVERSE DOWNTIME. (c) Implementing REVERSE DOWNTIME during the loading of a NewLine requires the Next field of 13 to be consulted; this shows that 7 and 38 often follow 13, so, to avoid compression by 7 and 38 being loaded into the NewLine, the NowScore register for 7 and 38 is set to zero.

affects a great many of its systems. Typically, such global changes might include those in temperature or calcium concentration. Global changes can also result from internal changes such as those resulting from alterations in DNA supercoiling, mediated by topoisomerases like DNA gyrase, which then affect the expression of many genes [22]. Such changes distort the entire phenotype but in a way that is related to the original phenotype. It seems reasonable to imagine that mutations that affect the activity of enzymes like gyrase have a different role in evolution from those that affect enzymes involved in the metabolism of lactose. One way to explore this in the program is to choose two elements (e.g., 6 and 7) to represent two different global conditions (such as a high calcium level and a low calcium level alias Yin and Yang). When a 6 is loaded into the Activity Register, the Compatibility Table is modified temporarily such that different addresses may be loaded into the Activity Register under these Yin conditions. The effect of loading a 6 can be carried over several time steps before the Compatibility Table is reset. When a 7 is loaded into the Activity Register, the Compatibility Table is modified in a different way to take into account the Yang conditions. In the present version, no such changes are allowed to the Compatibility Table.

4.8. Equilibrium and Nonequilibrium Structures. It has been proposed that bacteria and other cells are confronted with the task of reconciling surviving harsh conditions, which requires quasi-equilibrium structures (thick, cross-linked walls, and liquid crystalline DNA), with growing in favourable conditions, which requires nonequilibrium structures (such as those formed by the dynamic, ATP/GTP consuming, dynamically coupled processes of transcription and translation) [23–26]. To put it picturesquely, cells are confronted with life on the scales of equilibria and, conceivably, use the cell cycle in their balancing act [23, 27]. It might therefore be interesting to explore what happens when the equivalent of the energy currencies of the cell, ATP, GTP, and polyphosphate, is introduced into the program. This might be achieved by attributing the role of ATP to an element and giving this element special properties. For example, this element (e.g., element 77) might have to be present in the Activity Register for a subset of other nonequilibrium elements to be loaded (which could be done via the Compatibility Table); if 77 were absent from the Activity Register, this subset could not be loaded although another subset of equilibrium structures could be; the probability with which 77 could be loaded might depend on a combination in a line of the Activity Register of an input element (corresponding to glucose) and other elements (corresponding to glycolytic enzymes).

4.9. Several Activity Registers. The Yin-Yang approach (Section 4.7) could be adapted to study some of the under-appreciated implications of DNA being double stranded [28, 29]. This might be done by employing two Activity Registers running in parallel, one using the odd numbers and the other the even numbers. Loading one Activity Register (corresponding to creating a nonequilibrium hyperstructure) would require ATP whilst loading the other Activity Register (corresponding to creating an equilibrium hyperstructure) would require the absence of ATP (Section 4.8). This might allow two phenotypes to be selected simultaneously so as to balance the scales of equilibria (Section 4.8). More interesting still would be to create a hierarchy of Activity Registers or to allow Coco itself to create them during learning.

4.10. Free-Running, Dreaming, and Looking Ahead. If Coco's environment were partially disconnected, Coco can continue running—more exactly, free-running—in the absence of inputs. Such free-running would occur if the environment were not to respond immediately to an output. Periods of free-running could be used to play in a sandbox or to dream. A sandbox is a concept that refers to a software environment where potentially dangerous operations can be tested in isolation, thus reducing the chances of damaging the primary program. Using a sandbox allows risk-free exploratory behaviour and, in a sense, corresponds to Coco operating in a look-ahead mode. In this mode, Coco might be disconnected from the environment and run through different combinations of stored input and output modules (where modules are sequential lines of the Activity Register). These modules might be associated with special elements 12 and 13 to represent pleasure and pain, respectively, depending on whether they have been rewarded or punished. Starting from the present state, Coco might load the Activity Register with different addresses for different runs (e.g., 20 timesteps) and compare the pleasure index (e.g., sum of $12/(12 + 13)$) for these runs (in which inputs from the environment are replaced by those stored in the modules). The initial loading of the Activity Register corresponding to the most successful run would then be adopted and environmental inputs once again were allowed.

Hypotheses about the function of dreaming might be explored via the creation of a parallel set of objects copied from the normal ones, namely, a DreamActivity Register running in dream time with DreamElements, DreamNow, and DreamNext. Perhaps this could be used to discover and remedy pathogenic connectivities that lead the system to get stuck in deep basins of attraction and so forth. Such action might be based on a characterisation of the states in the Activity Register. For example, for each line in the Activity Register, the ten addresses have elements where each contains seven Now addresses and seven Next addresses, hence a total for the line of 70 Now addresses and 70 Next addresses. Each line, alias the state of the system at that time, can therefore be characterised by a pair of coordinates (different Now addresses, different Next addresses). Intuitively, a successful state should tend towards (length of line, length of line)—here (10, 10)—whilst an unsuccessful state should tend towards (length of line $\times$ size of Now field, length of line $\times$ size of Next field)—here (70, 70). (This is not strictly speaking correct, but it gives the flavour.) The sequence of these coordinates then constitutes a function that might be recognised and used during dreaming.

A different approach would be to use dreaming to make a landscape of the connection space by loading the Activity Register with different elements and then letting it run so as to determine state cycles and basins of attraction; when the program has done, it might be possible to modify the connections so as to maximise the use of the landscape and connect basins. One attractive possibility is that Coco acts during free-running to minimise conflict between the Next and Now processes. This would take the form of changing the connections so that in loading addresses into a line of the Activity Register the same elements are scored highly by both processes. It amounts to making the Next and Now processes coherent with one another. Indeed, insofar as incoherence results in unhappiness, it could even be argued that this would create a state of happiness in systems as different as men, bacteria, and machines!

How far away is all this from real biology? Is the dreaming envisaged here only relevant to higher organisms or does it extend, for example, to bacteria? If that were the case, related questions include how we would know that a bacterium was dreaming and what it would mean for the bacterium. Showing that dreaming had a role in the learning of Coco might cast some light on its potential evolutionary value for all organisms.

4.11. Varying the Size of the Now and Next Fields. In the version presented here, the Now and Next fields are of constant size (each contains 7 addresses). This is far from biological reality where networks generally contain nodes with very different connectivities, including hubs and "driver" nodes [30–32]. These connectivities can take the form of protein activators and repressors of gene expression [33], small molecules acting on functioning-dependent structures [34], ions travelling along charged filaments such as microtubules or DNA [35], ions and molecules moving along and through pili and nanotubes [36, 37], convergence on common frequencies of oscillation [38, 39], joining a hyperstructure [25], and so forth. Coco would be much closer to modelling reality if the size of the fields were to vary as a function of learning (one reason for this is that increasing the size of a field permits element X to have stronger connections to element Y because X's field can hold more copies of Y's address). This may prove relatively easy to implement: for example, rewarding and punishing might entail increasing and decreasing the fields, respectively (as well as overwriting).

The relative strengths of the Now and Next connections can also be modified via the NowNextWeighting. As shown in Sections 3.1 and 3.2, setting NowNextWeighting very low or very high makes Coco run with either just Nexts or just Nows. Insofar as Next connections can be equated with local connections and Now connections with global connections, changing the value of NowNextWeighting can result in a phase transition in connectivity with similarity perhaps to the great deluge algorithm [40] or to Dual-Phase Evolution [41] or even, in the world of microbiology, to maintaining the right ratio of nonequilibrium to equilibrium hyperstructures within cells [23].

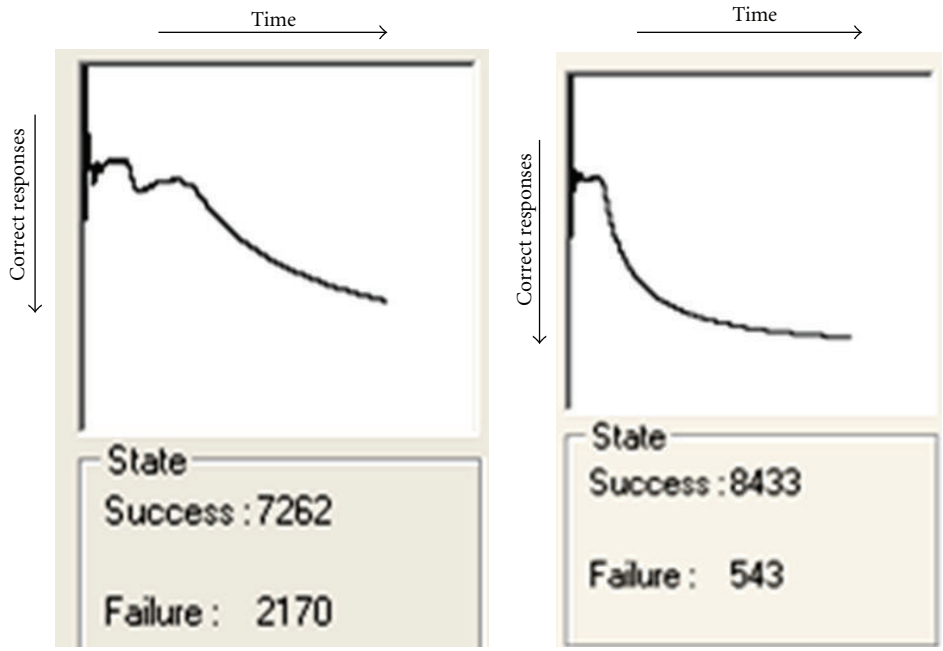
4.12. An Interactive Environment. An output is immediately followed by an environmental input that does not depend on the nature of the output. There is no possibility therefore for the present version of the program to influence the environment. This excludes the richness of connections that may emerge from dialogues between a learning system such as Coco and its environment. Two-way connections between a biological system such as a bacterium and its environment are fundamental and trying to understand them using Coco might take the form of Coco learning to play a simple game such as Noughts and Crosses (Tic-Tac-Toe).

4.13. Three or More Fields: from, Now, and Next and So Forth. The addition of a From field might add a new dimension to Coco. A From field would record the addresses of elements which preceded successful states in which the element was active. This would allow Coco to run backwards during Dreamtime (Section 4.10), analogous perhaps to the way humans mull over the day's events. In this speculation, such running might then allow input-output modules with similar characteristics to be identified (Section 4.10) and eventually connected via yet another, higher-level field involving a higher-level Activity Register. This, we would like to think, might be the equivalent of generating concepts.

4.14. Size of the Active Set. The size of the Active Set is an important parameter that can be changed and, in particular, increased, to take into account biological systems in which many elements can be active at the same time. In the present version of the program, the maximum size of the Activity Register is limited by the number of elements and by DOWNTIME (Section 2.7). This limits the size of the Active Set to around 80. An example of results obtained with an Activity Register containing 60 addresses is shown in Figure 18. Increasing the number of elements to, for example, 4000, allows learning with an Activity Register containing 100 addresses (Figure 18). It may prove important under some circumstances for Coco itself to modify the size of the Active Set. This might be the case if an input were to arrive "out of the blue" when Coco is in free-running mode; if an input were to trigger a sudden change in the size of the Activity Register, this could have a major effect, perhaps similar to that reported for the effects of a stimulus on connectivity in the cortex (for references see [41]) and perhaps similar too to the greater receptivity of bacteria to their environment when conditions start to worsen [42].

4.15. Collaborative Coherence. It will not have escaped the attention of the reader that, instead of separating the scores of the elements in competition for inclusion in the Active Set into Next and Now scores, these scores could be added together or even be combined synergistically to yield a kind of collaborative coherence. Philosophically, it would be nice to escape competition but we have no preliminary evidence that collaborative coherence leads to learning.

4.16. Pain and Pleasure. The present version punishes incorrect responses by randomly overwriting connections.



(a) (b)

Figure 18: Examples of successful learning with (a) 1000 elements and an Activity Register containing 60 addresses. (b) 4000 elements and an Activity Register containing 100 addresses.

This squanders a lot of information. A potentially better approach would be to retain and reuse information about mistakes by, for example, making use of specific elements such as 12 and 13 to represent pleasure and pain, respectively, in combination with a LOOK AHEAD subroutine (Section 4.10). Rewarding successful states might then entail writing a 12 into the Now and Next fields of the elements whose addresses are in the rewarded line of the Activity Register and, reciprocally, punishing might entail writing a 13 into them. This information could then be exploited using the LOOK AHEAD subroutine sketched out before that would count and compare the total of the addresses of these two elements when looking into possible futures (Section 4.10). An interesting question here is what would end up in the Now and Next fields of elements 12 and 13 themselves. Presumably, these fields would reflect connections to common, strong sources of pleasure and pain; these connections might then be used to drive the system towards or away from these sources.

4.17. Long-Term Memory. One way to obtain a long-term memory would be via a connections' matrix (number of elements $\times$ number of elements) that would record Now connections between elements that had participated in the same successful state (there could be similar ones for Next connections and, perhaps, further two matrices for unsuccessful states). A SUCCESSFUL CONNECTIONS subroutine could then be used to prevent overwriting successful connections or, at least, to alter the probability of overwriting these connections.

4.18. Noncyclical Input Sequences. Cycles are of major importance in biology. A primary example is the cell cycle which still remains to be fully understood [43]. The learning task presented here is based on an input sequence that is presented cyclically. Each output is immediately followed by a new input; at no time does Coco "run on its own" or run freely (Section 4.10). In responding to an input that "comes out of nowhere," as in the case of the first input in a linear series of inputs, the size of the Activity Register may be important (Section 4.14). One might envisage that during free-running the Activity Register would be small but would increase greatly on receiving an input; such increase might enable an input to make a decisive contribution to the composition of the Activity Register, particularly in the case of a variable Knumber since in such conditions, and when learning has occurred, the Now and Next fields of inputs become large (Section 4.11, unpublished data). It would also be interesting to explore the effects (on responding to an unannounced input) of changes to connectivity made during free-running (Section 4.10), changes that might even include modification of the weights associated with the Now and Next fields of inputs; such temporary modifications could be made during looking ahead.

Free-running whilst waiting for an input would entail Coco filling the Activity Register and wandering through the enormity of the combinatorial space in a state cycle [6]. The nature and length of such cycles may prove an important parameter in learning to respond to inputs that are given at random intervals from one another and from the outputs. This is because it is of little value in learning to connect (via the Next process) an active state containing an input to the

active state that immediately precedes it if this preceding state is hardly ever repeated. A possible solution would be for Coco to enter a short state cycle whilst waiting for the next input such that each input in the sequence was accessible from its own preceding state cycle. It is conceivable that these cycles might again be generated by the changes in connectivity occurring during dreaming (Section 4.10).

5. Relationship to Existing Systems

Hopfield's network model [44] uses the same learning rule as used by Hebb in which learning occurs as a result of the strengthening of the weights of the links between nodes [45]; this resembles the strengthening of links in Coco by the writing of addresses into the Now and Next fields of elements. In the Hopfield model it is assumed that the individual units preserve their individual states until they are selected at random for a new update; in Coco, the elements also preserve their identity until they are selected, but this selection is confined to members of the Active Set and occurs during rewarding and punishing. In a Hopfield network, each unit is connected to all other units (except itself); in Coco, each element can only be connected to a few others (the Knumber) via the Now and Next fields. A Hopfield network is symmetric because the weight of the link between unit i and unit j equals that between unit j and unit i; the network in Coco is asymmetric. In a Hopfield network, all the nodes contribute to the change in the activation of any single node at any one time; in Coco, only the elements (nodes/units) in the previous Active Set and in the developing Active Set contribute to the activation of an element (node/unit). In a Hopfield network, there is one type of connection between the nodes; in Coco, there are two types of connection—Next and Now. In a Hopfield network, the units can be in a state of either 1 or 0; similarly, in Coco, the elements can be either active or inactive.

It might be argued that a Hopfield network is a type of the Coco program. For example, if an attempt were to be made to turn Coco into a Hopfield network, (1) the size of the Knumber would be set similar to the size of the Enumber (i.e., the total number of elements) to make it closer to a weighting factor that takes into account all elements, (2) the activity of an element would be determined by its absolute score (using a threshold) rather than by its score relative to a limited number of competing elements, (3) the Anumber (the size of the Active Set) would therefore become a variable whose size would vary with the number of elements deemed to be active, (4) the Next links would correspond to the links between nodes but the Now links would have no equivalent, (5) the asymmetrical Coco network would tend towards symmetry if changes in the links to element i in the Next field of element j were accompanied by reciprocal changes in the links to element j in the Next field of element i (of course, Coco would then no longer run in the same way), and (6) some of the biologically relevant developments of Coco would have to be implemented in a Hopfield network which would be hard since Coco lends itself to the study of types of links with different properties; see Sections 4.6 and 4.7. In this context, it should be stressed that the weights in Coco are discrete, transparent, and easy to study and to manipulate.

Boolean networks have been extensively used to model biological systems. Thomas and collaborators have developed logical analysis which they have used both to study specific systems [46] and to derive general principles [47]. From such analyses, predictions can be made for experimental biologists to test. Logical analysis is not, however, a learning system like Coco. Reciprocally, Coco is not designed at present to model specific biological systems. As mentioned in Section 1, the (N, k) Boolean network of Kauffman [6] has given insight into the dynamics of biological systems and, in particular, into the concept of cells as living on the "edge of chaos" [2, 3]. But again, it is not a learning system like Coco.

6. Discussion

Few would deny that living systems are rich, complicated, and (hyper)complex. Such systems are often, almost necessarily, modelled and simulated by invoking Occam's Razor and adopting a reductionist approach. Life may, however, have originated as a rich, complicated, and diverse system, as, in other words, a prebiotic ecology [48]. In attempting to capture some of the characteristics of living system in a program, we have therefore adopted the holist approach of putting everything in and seeing what, if anything, emerges. To try to create a test-bed for concepts and to ensure that these concepts have some substance, we have written a program with a learning part, Coco. Coco contains parameters that may have very loose equivalents in aspects of evolution via global (as opposed to local) mutation, DNA replication, emergence, life on the scales of equilibria, and even the generation of concepts and dreaming. Most importantly, Coco is based on coherence.

Coherence characterises living systems. Coherence mechanisms operating—or suspected by some to operate—at the level of cells include tensegrity, ion condensation, DNA supercoiling, and a variety of oscillations [25, 48–52] plus, of course, mechanisms based on the usual activators and repressors of transcription along with DNA packaging proteins. Competitive coherence is an attempt to describe how bacterial phenotypes are created by a competition between maintaining a consistent story over time and creating a response that is coherent with respect to both internal and external conditions. Previously, it has been proposed that the bacterium *E. coli* can be considered as passing through a series of states in which a distinct set of its constituent molecules or "elements" are active [8]. The activity of these elements is determined by a competition between two processes. One of these processes depends on the previous cell state whilst the other depends on the internal coherence of the developing state. The simultaneous operation of these two processes is competitive coherence. Competitive coherence is in fact a scale-free concept. It can be applied to a population of bacteria such as a colony in which each cell is an element with its own Now and Next

fields. In this case, a typical Now process might involve global connections via a sonic vibration created by the combined metabolic activity of all the growing cells in the colony (which constitute the Active Set) [38] whilst the Next process might involve essentially local connections via diffusible molecules, sex pili, and lipid nanotubes [37]. We have invoked competitive coherence at higher levels to explain, for example, how a football team is selected. It is perhaps no longer original in computer science since the idea of two competing processes staggered in time can be found elsewhere in Simple Recurrent Networks [53]. What may be new is the possibility that the implementation of competitive coherence into a learning program could give rise to parameters suitable for describing the rich form of complexity found in living systems which depends on the interaction between many types of connection and which we have termed hypercomplexity [10]. The preliminary results from the toy program presented here encourage us to think that this may be the case.

One of these preliminary results is on the maximum size of the Activity Register that can result in Coco learning (Section 4.14). In Section 1, we mentioned the problem of how cells manage to negotiate the enormity of phenotype space in a reproducible and selectable way [6]; if, for example, a phenotype were determined by a simple combination of on-off expression of genes, a bacterium like *E. coli* with over 4000 genes would have the difficult task of exploring 2^{4000} combinations. (It should be noted though that no one knows how many genes are expressed at one time in an individual cell, let alone how many of these expressed genes are actually determining the phenotype, that is, form part of the Active Set.) However, if the phenotype were determined not directly by genes but at a higher level by a hundred or so extended macromolecular assemblies or *hyperstructures* comprising many different macromolecules—for which there is good evidence [25, 54, 55]—the number of on-off combinations would fall to 2^{100}. As we show here, a system with 4000 elements, of which a hundred form an Active Set, can learn via competitive coherence.

Finally, if there is any substance to our claim that competitive coherence is a fundamental to life, perhaps even its defining characteristic [56], the concept should be of value in novel approaches to computing inspired by and reliant on the way real cells behave [11].

7. Conclusion

Competitive coherence is a concept used to describe how a subset of elements out of a large set is activated to determine behaviour. It has been proposed as operating at many levels in biology. The results of the toy version of a type of neural network, based on competitive coherence, are presented here and show that it can learn. This is consistent with competitive coherence playing a central role in living systems. The parameters responsible for the functioning of the competitive coherence part of the program, which, for example, are related to complexity and emergence, may be of interest to biologists and others.

Acknowledgments

For helpful discussions, the authors thank Abdallah Zemirline, Alex Grossman, Francois Kepes, Jacques Ninio, and Michel Thellier. They also thank the anonymous referees for many valuable comments. For support they thank the Epigenomics Project, Evry, and the University of Brest. This paper is dedicated to the memory of Maurice Demarty.

References

[1] D. E. Ingber, "The origin of cellular life," *Bioessays*, vol. 22, no. 12, pp. 1160–1170, 2000.

[2] J. P. Crutchfield and K. Young, "Computation at the edge of chaos," in *Complexity, Entropy and the Physics of Information: SFI Studies in the Sciences of Complexity*, W. H. Zurek, Ed., pp. 223–269, Addison-Wesley, Reading, Mass, USA, 1990.

[3] C. G. Langton, "Computation at the edge of chaos: phase transitions and emergent computation," *Physica D*, vol. 42, no. 1–3, pp. 12–37, 1990.

[4] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world9 networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[5] P. Bak, *How Nature Works: The Science of Self-Organized Criticality*, Copernicus, New York, NY, USA, 1996.

[6] S. Kauffman, *At Home in the Universe, the Search for the Laws of Complexity*, Penguin, London, UK, 1996.

[7] D. Bray, "Intracellular signalling as a parallel distributed process," *Journal of Theoretical Biology*, vol. 143, no. 2, pp. 215–231, 1990.

[8] V. Norris, "Modelling *Escherichia coli* The concept of competitive coherence," *Comptes Rendus de l'Academie des Sciences*, vol. 321, no. 9, pp. 777–787, 1998.

[9] V. Norris, "Competitive coherence," in *Encyclopedia of Sciences and Religions*, N. P. Azari, A. Runehov, and L. Oviedo, Eds., Springer, New York, NY, USA, 2012.

[10] V. Norris, A. Cabin, and A. Zemirline, "Hypercomplexity," *Acta Biotheoretica*, vol. 53, no. 4, pp. 313–330, 2005.

[11] V. Norris, A. Zemirline, P. Amar et al., "Computing with bacterial constituents, cells and populations: from bioputing to bactoputing," *Theory in Biosciences*, pp. 1–18, 2011.

[12] S. G. Wolf, D. Frenkiel, T. Arad, S. E. Finkeil, R. Kolter, and A. Minsky, "DNA protection by stress-induced biocrystallization," *Nature*, vol. 400, no. 6739, pp. 83–85, 1999.

[13] V. Norris and M. S. Madsen, "Autocatalytic gene expression occurs via transertion and membrane domain formation and underlies differentiation in bacteria: a model," *Journal of Molecular Biology*, vol. 253, no. 5, pp. 739–748, 1995.

[14] T. Katayama, S. Ozaki, K. Keyamura, and K. Fujimitsu, "Regulation of the replication cycle: conserved and diverse regulatory systems for DnaA and oriC," *Nature Reviews Microbiology*, vol. 8, no. 3, pp. 163–170, 2010.

[15] V. Norris, C. Woldringh, and E. Mileykovskaya, "A hypothesis to explain division site selection in *Escherichia coli* by combining nucleoid occlusion and Min," *FEBS Letters*, vol. 561, no. 1–3, pp. 3–10, 2004.

[16] Y. Grondin, D. J. Raine, and V. Norris, "The correlation between architecture and mRNA abundance in the genetic regulatory network of *Escherichia coli*," *BMC Systems Biology*, vol. 1, p. 30, 2007.

[17] C. L. Woldringh and N. Nanninga, "Structure of the nucleoid and cytoplasm in the intact cell," in *Molecular Cytology of*

Escherichia coli, N. Nanninga, Ed., pp. 161–197, Academic Press, London, UK, 1985.

[18] V. Norris, L. Janniere, and P. Amar, "Hypothesis: variations in the rate of DNA replication determine the phenotype of daughter cells," in *Modelling Complex Biological Systems in the Context of Genomics*, EDP Sciences, Evry, France, 2007.

[19] L. H. Hartwell and T. A. Weinert, "Checkpoints: controls that ensure the order of cell cycle events," *Science*, vol. 246, no. 4930, pp. 629–634, 1989.

[20] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[21] M. H. V. Van Regenmortel, "Emergence in Biology," in *Modelling and Simulation of Biological Processes in the Context of Genomics*, P. Amar, V. Norris, G. Bernot et al., Eds., pp. 123–132, Genopole, Evry, France, 2004.

[22] K. S. Jeong, Y. Xie, H. Hiasa, and A. B. Khodursky, "Analysis of pleiotropic transcriptional profiles: a case study of DNA gyrase inhibition.," *PLoS Genetics*, vol. 2, no. 9, p. e152, 2006.

[23] V. Norris, M. Demarty, D. Raine, A. Cabin-Flaman, and L. Le Sceller, "Hypothesis: hyperstructures regulate initiation in *Escherichia coli* and other bacteria," *Biochimie*, vol. 84, no. 4, pp. 341–347, 2002.

[24] A. Minsky, E. Shimoni, and D. Frenkiel-Krispin, "Stress, order and survival," *Nature Reviews Molecular Cell Biology*, vol. 3, no. 1, pp. 50–60, 2002.

[25] V. Norris, T. Den Blaauwen, R. H. Doi et al., "Toward a hyperstructure taxonomy," *Annual Review of Microbiology*, vol. 61, pp. 309–329, 2007.

[26] V. Norris, "Speculations on the initiation of chromosome replication in *Escherichia coli*: the dualism hypothesis," *Medical Hypotheses*, vol. 76, no. 5, pp. 706–716, 2011.

[27] V. Norris and P. Amar, "Life on the scales: initiation of replication in *Escherichia coli*," in *Modelling Complex Biological Systems in the Context of Genomics*, EDF Sciences, Evry, France, 2012.

[28] E. P. C. Rocha, J. Fralick, G. Vediyappan, A. Danchin, and V. Norris, "A strand-specific model for chromosome segregation in bacteria," *Molecular Microbiology*, vol. 49, no. 4, pp. 895–903, 2003.

[29] M. A. White, J. K. Eykelenboom, M. A. Lopez-Vernaza, E. Wilson, and D. R. F. Leach, "Non-random segregation of sister chromosomes in *Escherichia coli*," *Nature*, vol. 455, no. 7217, pp. 1248–1250, 2008.

[30] D. J. Raine and V. Norris, *Metabolic cycles and self-organised criticality*. Interjournal of complex systems, Paper 361, http://www.interjournal.org/, 2000.

[31] A. L. Barabási and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nature Reviews Genetics*, vol. 5, no. 2, pp. 101–113, 2004.

[32] Y. Y. Liu, J. J. Slotine, and A. L. Barabási, "Controllability of complex networks," *Nature*, vol. 473, no. 7346, pp. 167–173, 2011.

[33] N. Guelzim, S. Bottani, P. Bourgine, and F. Képès, "Topological and causal structure of the yeast transcriptional regulatory network," *Nature Genetics*, vol. 31, no. 1, pp. 60–63, 2002.

[34] M. Thellier, G. Legent, P. Amar, V. Norris, and C. Ripoll, "Steady-state kinetic behaviour of functioning-dependent structures," *FEBS Journal*, vol. 273, no. 18, pp. 4287–4299, 2006.

[35] C. Ripoll, V. Norris, and M. Thellier, "Ion condensation and signal transduction," *BioEssays*, vol. 26, no. 5, pp. 549–557, 2004.

[36] G. Reguera, K. D. McCarthy, T. Mehta, J. S. Nicoll, M. T. Tuominen, and D. R. Lovley, "Extracellular electron transfer via microbial nanowires," *Nature*, vol. 435, no. 7045, pp. 1098–1101, 2005.

[37] G. P. Dubey and S. Ben-Yehuda, "Intercellular nanotubes mediate bacterial communication," *Cell*, vol. 144, no. 4, pp. 590–600, 2011.

[38] M. Matsuhashi, A. N. Pankrushina, K. Endoh et al., "Bacillus carboniphilus cells respond to growth-promoting physical signals from cells of homologous and heterologous bacteria," *Journal of General and Applied Microbiology*, vol. 42, no. 4, pp. 315–323, 1996.

[39] G. Reguera, "When microbial conversations get physical," *Trends in Microbiology*, vol. 19, no. 3, pp. 105–113, 2011.

[40] G. Dueck, "New optimization heuristics; The great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, vol. 104, no. 1, pp. 86–92, 1993.

[41] G. Paperin, D. G. Green, and S. Sadedin, "Dual-phase evolution in complex adaptive systems," *Journal of the Royal Society Interface*, vol. 8, no. 58, pp. 609–629, 2011.

[42] J. Vohradský and J. J. Ramsden, "Genome resource utilization during prokaryotic development," *The FASEB Journal*, vol. 15, no. 11, pp. 2054–2056, 2001.

[43] X. Wang, C. Lesterlin, R. Reyes-Lamothe, G. Ball, and D. J. Sherratt, "Replication and segregation of an *Escherichia coli* chromosome with two replication origins," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, no. 26, pp. E243–E250, 2011.

[44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.

[45] D. O. Hebb, *The Organization of Behavior*, Wiley and Sons, New York, NY, USA, 1949.

[46] M. Kaufman, J. Urbain, and R. Thomas, "Towards a logical analysis of the immune response," *Journal of Theoretical Biology*, vol. 114, no. 4, pp. 527–561, 1985.

[47] R. Thomas, D. Thieffry, and M. Kaufman, "Dynamical behaviour of biological regulatory networks I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state," *Bulletin of Mathematical Biology*, vol. 57, no. 2, pp. 247–276, 1995.

[48] A. Hunding, F. Kepes, D. Lancet et al., "Compositional complementarity and prebiotic ecology in the origin of life," *BioEssays*, vol. 28, no. 4, pp. 399–412, 2006.

[49] H. Fröhlich, "Long range coherence and energy storage in biological systems," *International Journal of Quantum Chemistry*, vol. 42, no. 5, pp. 641–649, 1968.

[50] V. Norris and G. J. Hyland, "Do bacteria sing? Sonic intercellular communication between bacteria may reflect electromagnetic intracellular communication involving coherent collective vibrational modes that could integrate enzyme activities and gene expression," *Molecular Microbiology*, vol. 24, no. 4, pp. 879–880, 1997.

[51] D. E. Ingber, "The architecture of life," *Scientific American*, vol. 278, no. 1, pp. 48–57, 1998.

[52] A. Travers and G. Muskhelishvili, "DNA supercoiling—a global transcriptional regulator for enterobacterial growth?" *Nature Reviews Microbiology*, vol. 3, no. 2, pp. 157–169, 2005.

[53] J. L. Elman, "An alternative view of the mental lexicon," *Trends in Cognitive Sciences*, vol. 8, no. 7, pp. 301–306, 2004.

[54] R. Narayanaswamy, M. Levy, M. Tsechansky et al., "Widespread reorganization of metabolic enzymes into reversible assemblies upon nutrient starvation," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, no. 25, pp. 10147–10152, 2009.

[55] P. M. Llopis, A. F. Jackson, O. Sliusarenko et al., "Spatial organization of the flow of genetic information in bacteria," *Nature*, vol. 466, no. 7302, pp. 77–81, 2010.

[56] V. Norris, P. Amar, G. Bernot et al., "Questions for cell cyclists," *Journal of Biological Physics and Chemistry*, vol. 4, pp. 124–130, 2004.