NTNU
Norwegian University of
Science and Technology

Bio-inspired Reverse Engineering of Regulatory Networks

A Revised Approach

Pedro Leon Pozo

Master of Science in Computer Science
Submission date: September 2011
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

NTNU – Trondheim
Norwegian University of
Science and Technology

Bio-inspired Reverse Engineering of Regulatory Networks
A Revised Approach

Pedro León Pozo
leonpozo@stud.ntnu.no

June 20, 2011

Contents

List of Figures

List of Algorithms

Chapter 1

Abstract

This work appears to complement an existing project, *"Bio-inpired reverse engineering of regulatory networks"*[STH09], proposes a new algorithm inspired in the artificial development technique performing reverse engineering over regulatory networks. The present project studies that article addressing possible weaknesses and scalability issues. Nevertheless, during the investigation some updates have been performed over the algorithm, improving the previous results in some scenarios. Moreover DARBNs have been used as representation, looking for an alternative update schema and its possible improvements over results. Lastly, software reusable tools have been implemented and documented to allow additional experiments.

Chapter 2

Introduction

2.1 Background

Genes are the most essential part of the live, the most basic representation of all living forms over the earth. The complexity of the entire world can be represented by them and however they are relatively simple. Only four bases are enough to represent all the life of the world. Obviously it is not so simple. Thinking about all the complexity of the living beings; in the different cellular types that compose them and also in the gene-encoded information needed to allow the life, is easy imaging that only the gene explicit representation is not enough. For example, the brain of a newborn baby has millions and millions of neurons interconnected that allow the basic vital functions. This information comes directly from the genetic information (is not learned) and it determines the interconnection of a huge number of neurons making possible the life. It can not be encoded in a direct way into the (around of) 26,000 genes of the human genome. In the last years the science has showed that the complexity of the living beings is not the direct result of the number of genes; it is product of complex mechanisms of genetic regulation.

During years, single genes have been studied looking for the relation between them and phenotypic characteristics; several important discoveries have been done using this approach (some genetic diseases and some very concrete characteristics have been mapped), but finally the regulatory networks has been introduced to explain this huge complexity. Using this approach the phenotypic characteristics can be understood studying the topology and the dynamics of the regulatory network. This network has much more information than the information encoded in the gene sequences.

Regulatory networks are the base concept used to perform the work described by this report. Regulatory networks are the abstraction that represents the interactions and regulatory relations between DNA and the substances present in the cell. The cell contains proteins, that can be structural (to for example build the cell membrane), enzymes (to activate chemical processes) and transcription factors (to regulate the expression of the genes, e.i. the phenotype).

2.2 Modeling regulatory networks

As a simplified model, the regulatory networks can be represented as a graph where the genes are nodes and transcription factors are edges. The genes can be switched on or switched off by other genes using transcription factors that connect both genes. When a node is connected with others, the expression it depends of the states of the related nodes. The combination of active and inactive genes into the network determines the resultant phenotype.

To modeling this kind of networks, Dr. Stuart A. Kauffman proposed in 1969 random boolean networks (RBN) as representation of regulatory net-

works; before the Kauffman's proposal, differential equations were the only available model to represent regulatory networks. These equations were proposed by Francois Jacob and Jacques Monod. Unfortunately the proposed model was focused on the concentration of proteins more than in the activation of genes.

Kauffman hypothesized that the attractors of the RBNs correspond with the cellular fates or kinds of cells; after years of this theory, it has been ratified trough experiments by other researchers. Finally, the Kauffman's model captures the essential aspects about the gene interactions with a relative simple model.

To perform this project RBN will be used as representation, following the Kauffman's model.

2.3 Motivation

This project continues an ongoing project developed by researchers of Crablab; the project uses a simple developmental technique to achieve real biological results. The goal of this work is refining the technique and further investigate the scalability limitations of it. "Bio-inspired Reverse Engineering of Regulatory Networks" [STH09] is the article published that investigates the search of regulatory networks models using incomplete topology of a regulatory network as data, RBNs as representation and the algorithm proposed in the previous project. Yeast cell regulatory network is the model used to perform the experiments.

The cited technologies have been studied using several specialized articles and simulation tools have been built to achieve previous results and refine conclusions. Special emphasis has been done over algorithm's design, model's representation and optimizations performed. All these points will be detailed since an important part of this work is study scalability and produce reusable tools.

Chapter 3

Concepts

3.1 Regulatory networks

Regulatory networks have been introduced in the previous section, in this section all the important details related with them will be described in all the important concepts related with the experimentation. Topology, reference model, dynamics and attractors are fundamental concepts to understand this work.

Yeast cell is the reference cellular type used to perform this project. It has been chosen paying attention to several points:

- The previous project uses this network to evaluate the proposed approach, because that results to compare are available.
- It is a well known model and particularly simple case of study.
- It is appropriate to this project's scope.
- The existence of several previous articles studying it, helps to refine conclusions and find others insights.

3.1.1 Topology

The network's topology is an abstraction of the cell's chemical dynamics, describing which substance affects all the others to which it is connected. Topology can be abstracted, represented and implemented by a digraph. This abstraction simplifies the cellular structure which is expressed as a simple and schematic system composed by nodes and directed edges.

According to this abstraction, the **nodes represent genes** and **edges represent transcription factors**. Transcription factors are proteins that bind to the start of the DNA sequences (known as promoter region), thereby they can activate o deactivate the expression of the bound gene up with them. This abstraction can be seen in the figure 3.1. The genes of the yeast cell are represented by circles, transcription factors are directed edges of different colors. The red color means negative regulation or deactivation. The green color means positive regulation o activation. The yellow arrows represent negative self regulation. [1]

The cell's chemical dynamics are much more complex than the used representation; in this sense [LLL+04] says: *"In Prince, the arrows in the network have very different time scales of action, and a dynamic model would involve various binding constants and rates"*. It means that in the real world the bound transcription factors up to the gene have different thresholds and more complex behaviors that regulates the gene expression, however in this work a simplified model is applied and only one boolean function is used to update the gene status. This function is called **transition rule** and it is defined mathematically at (3.1), this simplified transition rule has

[1] An erratum was found in [STH09] (*"Experiments and results"*) about self regulation inputs, the author says: *"The model also has 'self-degradation' (yellow loops) on those nodes that are not negatively regulated by others."*, this affirmation is not true to the node *Swi5*. It can be checked in [LLL+04].

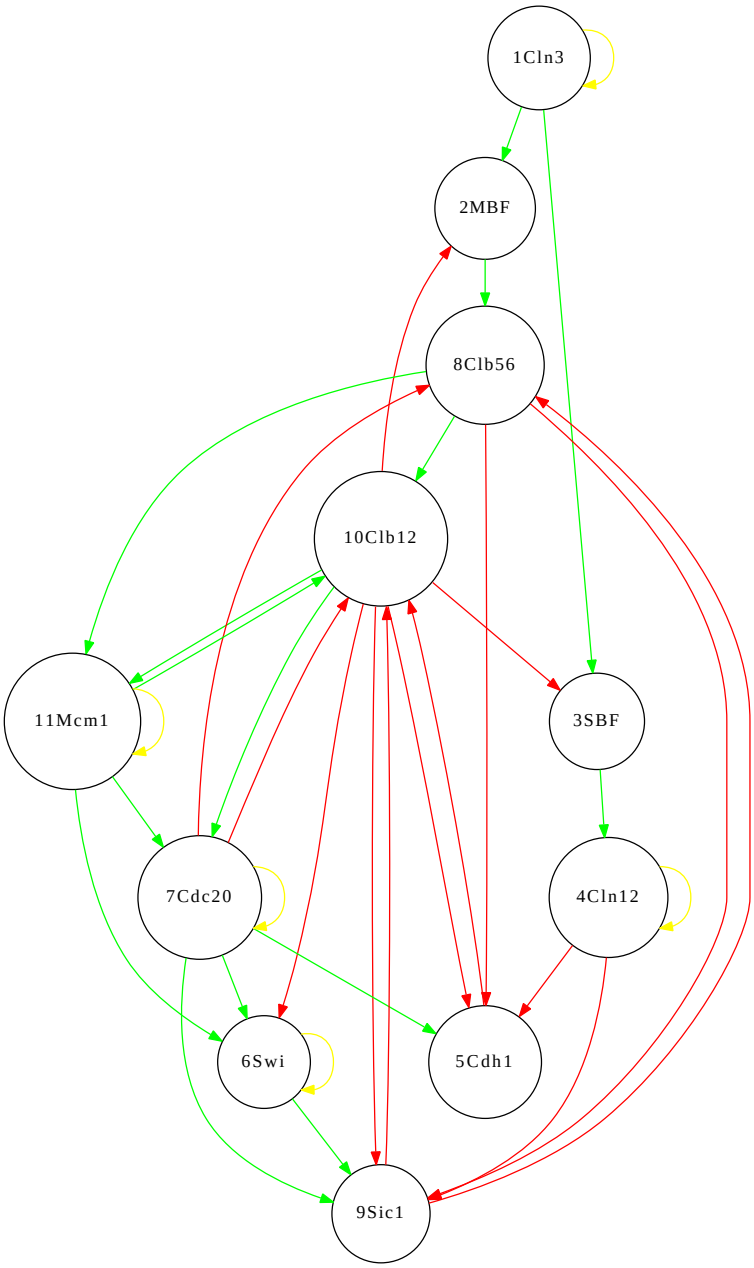


Figure 3.1: Yeast cell's regulatory network. Digraph representation.

been extracted from [LLL+04].

$$S_i(t+1) = \begin{cases} 1, & \sum_j a_{ij} S_j(t) > 0 \\ 0, & \sum_j a_{ij} S_j(t) < 0 \\ S_i(t), & \sum_j a_{ij} S_j(t) = 0 \end{cases} \quad (3.1)$$

Each node's state i will be determined at time $t+1$ as $S_i = 1$ or $S_i = 0$ (active or inactive respectively state of the gene), depending of the $\sum_j$ result at time t where j is the set of regulatory inputs of the gene i.

According with checked articles as [Wue98a] and [LLL+04], the DNA sequences content by genes and the regulation relations between them represent the **genotype** of the cell. This information contains all the possible manifestations of the final individual.

3.1.2 Dynamics

The dynamics of the network describe how the gene's expression occurs over time according to the network's structure, kind of transcription factors (positive or negative regulation) and the signals received by the cell. It is important to study this aspect to see which states are the most stable; typically the network's dynamics becomes cyclically stable over several states of the all possibles. The graphical representation of the changes from a network state to others is represented in figure 3.3, it shows network's states transitioning to the next states and creating attractor's basins.

A network state is defined as the set of genes active in the regulatory network. For example, paying attention to a simple network, represented in the figure 3.2 a potential network state could be defined as $S(0) = [A = 0, B = 1, C = 1]$ (or directly as 011 according to the ordered nodes).

The activation or deactivation of the genes determines the phenotypic characteristics of the cell. For example, in human cells, the expression of the genes can determinate the cellular destiny (duplication, apoptosis, etc.) and the cellular kind (liver, kidney, etc.). Is possible to see the cell in a determinate state paying attention to expression of each gene, each combination of turned on/off genes will portray a determinate **phenotype**.

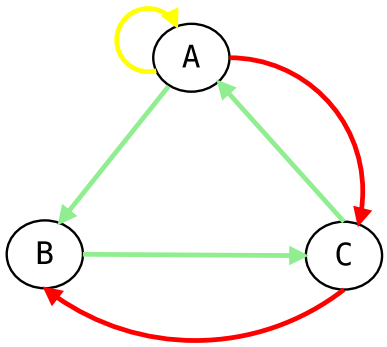


Figure 3.2: Simple regulatory network.

The dynamics make possible to express much more information than the information directly encoded by the DNA sequences contained by the genes. Fox example, according to Kauffman's model, the representation of the yeast cell have 2^n possible phenotypic states (it is because each node expression is a boolean value with two possible values, $S = 0$ (off) or $S = 1$ (on), where n corresponds with the number of nodes present in the network (for the reference cell only eleven nodes). It means 2048 possible states are possible from a small set of genes.

Dynamics are deterministic; it means, every execution will produce the same space state with the same pathways, same attractors and same "garden-of-Eden" states (no previous states). This is a key point on this work. Asynchronous random boolean networks are non deterministic and this characteristic makes it not valid for this experimentation.

3.1.3 Stable network states

During the dynamics execution the network changes the inner state combining the different genes turned on/off, since the expression of a gene influences, directly or indirectly, all the rest in the network genes the state is updating over the time, but gradually several states are repeated again and again. These network states, more stable than the rest are **attractors** of the regulatory network. Sometimes, when a particular attractor is set, it keeps the network state

fixed not changing along the time or it changes but as part of a cycle (several states in a loop), changing sequentially between an small set of states; both situations are different kind of attractors. It can be seen in figure 3.3, gray nodes represent the attractor states.

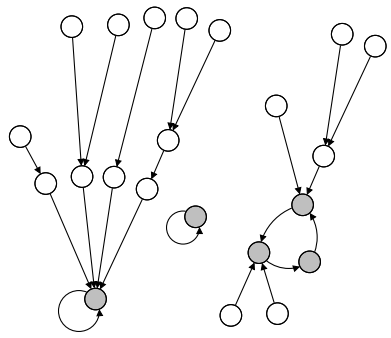


Figure 3.3: Potential subset of network dynamics graph representation. Gray nodes represents attractors.

The stable states correspond with regular phenotypes of the cell, the design of regulatory network's dynamics does it more robust and stable the genetic expression and prevents the occurrence of mutations into the individual respect a small perturbations of the network. This property can be seen in [LLL+04], article with the robustness of yeast cell-cycle as main investigation topic.

3.1.4 Update schema

It exists a debate about to whether the synchronous update schema is appropriate to represent regulatory networks behavior. Kauffman did it in his work (1993) and was criticized by other authors that believe in the asynchronous approach. The nature's events do not occur sequentially, but several authors, in contrast, think that a synchronous is an appropriate approach. [STH09] and [LLL+04] uses this approach to perform their experiments, but [Ger02] says: *"We are agree with the critic that the synchronicity was an assumption without a base, but we do not believe that genetic regulatory networks are random. They should be most probably modeled better with DARBNs, if we can model the updating periods for each gene."*.

DARBNs (deterministic asynchronous random boolean network) [**?**], are RBNs that updates asynchronously the network according to fixed parameters (transitions and periods), but the dynamics **are not random** (is deterministic). This approach is studied in this work.

The main problem of the asynchronous update schema is that the resultant dynamics schema is non deterministic, getting random results in each execution. This is not a valid approach to this work.

Chapter 4

Implementation

One of the goals of this project is to refine and address the problems of the developmental inspired algorithm written by Cristina Costa in [STH09]. To perform this target, previous work has been analyzed and the same principles has been implemented, in some cases, optimizing them in terms of scalability and possible hardware implementation.

The main points to accomplish such goal are:

- Understand how the regulatory networks work, their representation and fundamental biology.
- Implement RBN simulation to study network's dynamics (also implementing DARBN).
- Implement artificial developmental algorithm described by [STH09].

Some implementation details are not fully described in the previous article and the aspects like *data structures, fixed point's calculation algorithm or phenotype distance calculation.* These implementation details have been open to interpretation, others like *individual representation and RBN implementation* has been rewritten seeking simplicity. All the code has been written from scratch.

During the software's design several aspects have taken priority over the rest: **The lower level of abstraction and highest simplicity of used data structures** and **algorithm's temporal and spatial complexity optimization** to make a potential implementation over hardware architecture easier and **potential reuse** of the code as well.

During the design and implementation stages, the system has been splited in several modules making it easier to build a complete experimentation platform which are:

- **RBN Tools**
  Over the rest of the components have been build these tools. Contains RBN, DARBN and ARBN implementation, attractor calculation, utilities and interfaces to extend the basic functionality.

- **Developmental algorithm**
  It is the engine of the experimentation process; is implemented following the [STH09] notes. Also it contains fitness functions, individual implementation and interfaces to extend the module.

- **Simulators (RBN Simulator and Developmental Algorithm simulator)**
  These tools have been created to allow the easy execution by the user without programming skills, using execution parameters providing a fast experimentation.

These components can be checked in figure 4.1. This figure shows the system's architecture and how the different modules are related each others.

In the next points, the details over the listed components will be described using graphics for a better compression and highlighting any important design or optimization's decision.

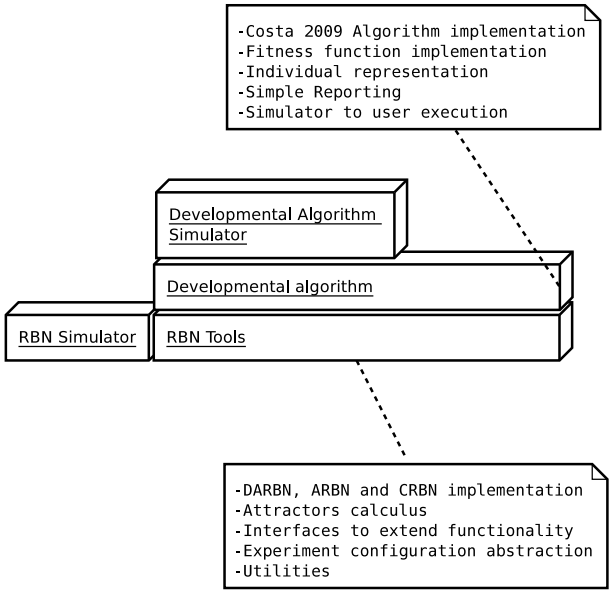


Figure 4.1: System's modules.

4.1 RBN Tools

4.1.1 Random boolean networks

Designing an implementation of a random boolean network is a well known challenge, there are many papers published about it with examples and tools that solve this challenge. Here, several implementations have been checked and tested attempting to find the most suitable implementation according to the previously described criteria.

There are several approaches to represent the network's structure through data structures: Object oriented (using different object instances to represent it, linking the instances to objects references), the representations used in the reference project (using a table that describes each node as a column and each input as a signed integer number) or through boolean arrays that represent all the possible connections of the network (chosen approach).

According to the chosen approach each RBN of the system uses the next structures to represent it:

- **Inputs matrix**
  Represents the existence of an input relation between nodes. if inputs[toNode][fromNode] = true the input relation exists. This structure is implemented by a bidimensional boolean array with size n^2, where n is the number of nodes in the network.

- **Regulation matrix**
  Represents the regulation nature of an input relation between nodes. This matrix's values are not representative if the same input is not expressed in the inputs matrix. If regulation[toNode][fromNode] = true express a positive regulation, regulation[toNode][fromNode] = false represents a negative regulation. This structure is implemented by a bidimensional boolean array with size n^2.

- **Periods matrix (p)**
  According to [Ger04] each node of a DARBN has an update period that determines its update behavior. Only for DARBN. This structure is implemented by a unidimensional integer array with size n.

- **Translations matrix (q)**
  According to [Ger04] each node of a DARBN has a translation that determines its update behavior. Only for DARBN. This structure is implemented by a unidimensional integer array with size n.

The figure 4.2 represents these structures and shows how each matrix represent a different property of the regulation relation. This representation, using boolean values can be implemented over hardware architectures directly with bits.

- **Simplicity**
  Arrays of booleans are simple representation. It could be represented using bits. ([Had08] points the easy implementation of these structures directly over hardware and the similarities between them).

- **Fast evaluation**
  Performing boolean operations is really fast. Depending on the architecture it takes only one clock cycle to be evaluated; in more advanced

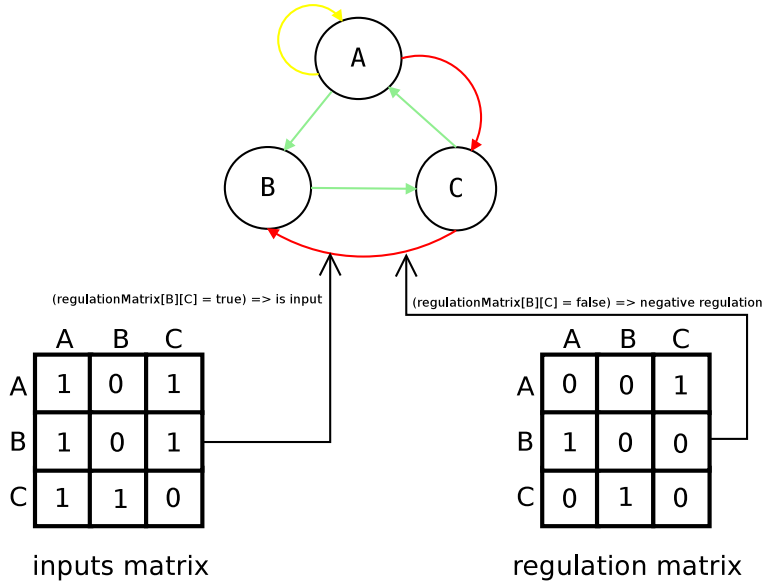


Figure 4.2: RBN representation using boolean matrices.

architectures (like GPGPU) it could be done in the same time over all the matrix.

- **Flexibility**
  Allows the representation of all possible connections with a fixed size, meaning that is not needed to control the structure's size in the algorithms.

Unfortunately this representation has the problem of the **complexity**. Algorithmic complexity to calculate a network's state is always n^2 (see 4.3); with other implementations like the used in [HJS] the complexity is n^k where $k \leq n$; only in a full connected network $k = n$. However, this approach has been chosen for simplicity and potential parallelism facilities since it is suitable to check different values of k for each node and is suitable to perform very optimized bit representation over hardware architectures.

This implementation is **deterministic** and **synchronous** and is named **Clasical random boolean networks** (CRBN).

4.1.2 Deterministic asynchronous random boolean networks

DARBN is a new kind of RBN proposed by Carlos Gershenson in [Ger02]. This implementation is more realistic than the classic one (used by Kauffman) because the update scheme is not synchronous. Several authors do not agree with the classical approach, they think that the regulation process in the nature does not occurs in a synchronous way. However asynchronous schemas are not deterministic and the authors do agree that regulatory networks in the nature are deterministic. DARBNs have both properties. About these ideas the author says ([Ger04]): *"I agree with the criticisms to the synchronous assumption: genes do not march in step. But I do not believe that they are random. Having this in mind, I proposed Deterministic Asynchronous RBNs (DARBNs)"*. In the same way Li says in the article "The yeast cell-cycle network is robustly designed" [LLL+04]: *"Making the time constants of all arrows the same could have disastrous consequences in network dynamics. However, we are saved for this particular network because of its intrinsic sequential nature. We have tested the*

dynamics with varied time scales of action (phosphorylation and transcriptional activation) for different arrows and obtained similar results.”. **Both affirmations could indicate potential applying the previous work's algorithm to calculate phenotypical distances**.

DARBN introduces two variables more for each node, P_i and $Q_i (P_i, Q_i \in \mathbb{N}, P_i < Q_i)$, P_i is called *period* and (Q_i) is called *translation*. Both parameters remain fixed and determine when a node is updated. With $P_i = 1, Q_i = 0$ the DARBN will work like a CRBN.

In this project the search of regulatory networks using DARBN to calculate the individual's phenotype is experimented.

4.1.3 Space state

One of the most important questions of this work has been to choose a proper and optimal state space's representation. The spacial and algorithmic complexity of the space state calculation is a very important point and a potential bottle neck of the developmental algorithm.

In the first implementation of the state space it was represented trough an object's model, relating each state to others by references. This schema was the literal representation from the graph theory to OOP implementation; it was valid and elegant but needed a big amount of memory and the cost of calculate the distance between phenotypes was high. Since this operation is used intensively during the developmental algorithm's execution is needed to keep it simple following the same spirit of the RBN representation.

As a solution, other representation was found using only one array of integer positive numbers. This representation keeps the algorithmic complexity in $O(2^n)$ (see 4.3). *Complexity will grow exponentially with the number of nodes.* Fortunately the networks proposed to experiment are small networks $n = 11$, with a practical upper limit of about $n = 26$ ([Wue98a]). When $n > 26$ then **hardware implementation approach is the most viable option**.

According to the proposed representation, the **attractors calculation can be done at the same time that the space state calculation is performed without incrementing the temporal complexity's order**.

To understand this representation a key point should be defined: **It is possible to represent a network state as an integer number**. For example, paying attention to the figure 4.4, it illustrates a possible state of a very simple network. In this representation the green nodes are active and the the red one is inactive. This particular network state can be represented as $S = A = 1, B = 0, C = 1$, as a simplification is possible to express it as a list of ordered boolean values (or bit string) where each position corresponds to a determinate network's node: $S = 101$ and it can be converted to an integer number doing a base change: $S = 5$. Is very important to respect a fixed node's positions and the same most-significant bit position order. **In the implementation the bit strings are expressed as little endian**.

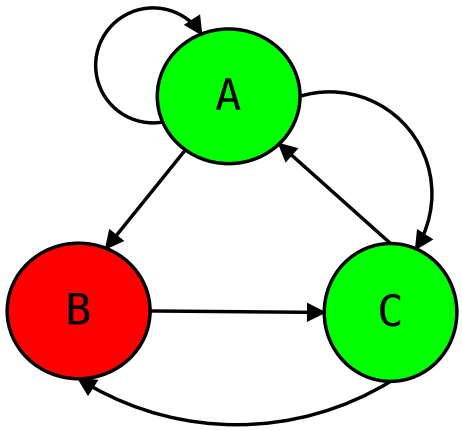


Figure 4.4: Simple network state representation.

To this representation each array's position (index) represents a state into the space state and the value at this position represents the state that will go after it (it works like a pointer to the next state). The

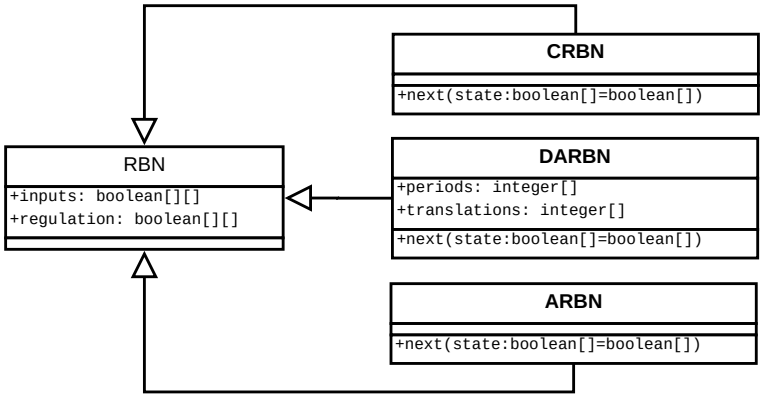


Figure 4.3: RBN Tools classes diagram.

figure 4.5 represents a possible instance of this structure, **each position's value has the reference of the destiny state from the state represented by the index**. It is possible to draw a parallel between the state space's representation with the graph showed in figure 4.5 and the space state represented as an array in the figure 3.3, both represent the same information.

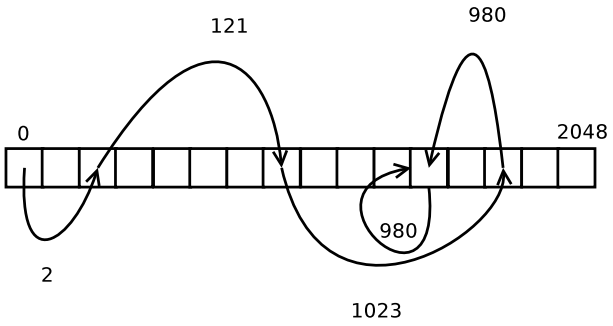


Figure 4.5: Integer array as space state representation.

This representation has several advantages:

- **Simple implementation**
  The structure is a low level structure, suitable to be implemented using hardware.
- **Very good to compare as phenotype**
  One of the best properties of this representation is the possibility of use it in developmental or genetic algorithms as phenotype. It is really simple and fast to compare phenotypical distances and has also a great accuracy (because it includes attractor's cycles, the fixed points and the garden-of-Eden states).

4.1.4 Basin size

The basin size is the summatory of all states (garden-of-Eden states and transitional states) in the attractor's pathways. This property is used in the previous work to compare the individual's phenotypes. In this work the phenotypical distance calculation has been re implemented because the representation to the space state used (see 4.1.5). However the basin size is still calculated to ensure the attractor's existence as feedback.

To calculate this property of the attractor nodes an extra step during the space state algorithm's execution. Two extra arrays of integers are needed to count the basin size and cache the final attractor of a given state (to avoid check all the states between actual node and its attractor). For each new calculated state, the new state's basin size (basin_size[state]) value will be updated adding the content of the previous state to the actual basin_size. It will happen

for each state. This change in the algorithm 2 adds several operations with complexity $O(1)$ so the algorithmic complexity of the space state does not change and the spatial complexity grows to $(3 * 2^n)$.

4.1.5 Phenotypical distance

Checking the phenotypical distance between individual is pretty easy since the space state representation is calculated. It is just the Hamming distance between two integer's arrays, which is pretty simple and efficient to perform. Is specially interesting that over hardware architectures it would be faster.

In the previous work, Cristina Costa, saiys: *"Then, the fitness of this network is determined by executing it (according to the transition rule shown in 1) to all and each of the given initial conditions and measuring the Hamming distance between the obtained final attractor and the yeast network's final attractor for that initial condition, which can only be one of the seven attractors"*([STH09]).

With this affirmation **the author does not clarify if it is representing all the possible phenotypes**, in the previous work the author only considers only a type of attractor (at least explicitly) in the phenotype: the **fixed point**. **It could be a mistake evaluating potential individuals and specially applying this schema to find other regulatory networks (the yeast cell only has fixed points as attractors).** During the experimentation phase, the different kinds of attractors have been found applying CRBN and DARBN approaches. In the articles [Wue98a] and [Wue98b], it can be seen the different types of states: Fixed points and cycles as stable states, and garden-of-Eden and transitional states.

In order to make it sure that this problem does not happen, this implementation checks the distance of the entire space state which contains all the states, not only fixed points. Since the basin size is also checked (to perform the fitness calculation) in the previous project and also in this, any attractor present should have exactly the same number of previous states (garden-of-Eden and transitional states, i.e. all the pathways involved states).

4.1.6 RBN Simulator

The RBN Simulator is only an entry point. The package RBN Tools can be used as library (as Developmental algorithm package does) and as simulator to run and analyze RBN, ARBN and DARBN networks.

The simulator tools accept parameters and can be extended through experiments implementation. (B.0.1 for usage and design information).

4.2 Developmental algorithm

4.2.1 Algorithm

Maybe the most important component of the system is the search algorithm, it comes directly from the reference project and such algorithm is a developmental inspired algorithm. The author says: *"Nature's way of handling complexity clearly points in the direction of a non one-to-one mapping from genotype to phenotype. Biological development is nature's way of coping with scaling. It provides a plan of the phenotype, describing how the organism is to be built rather than a blueprint containing explicit information about every detail of the proposed phenotype"*.

It performs a search over a generated random population of individuals; each individual has random edges into the network regulated by probability, depending on the probability value ($0.0 < P_i < 1.0$), the a particular node's input could be present in the genotype or not; if the existence of the regulation factor makes the individual better the probability of inclusion will grow. After several iterations the best regulation factor's probability tends to 1.0 (always included) or 0.0 (never included). Probabilities are adjusted along the iterations.

The algorithm is described by (1) with pseudocode, and a diagrammatic representation in figure 4.6.

Small improvements have been done: For example, avoid the occurrences variable in the probabilities adjustment process. It will be fixed non normalized values. Also in the original article the adjustment was

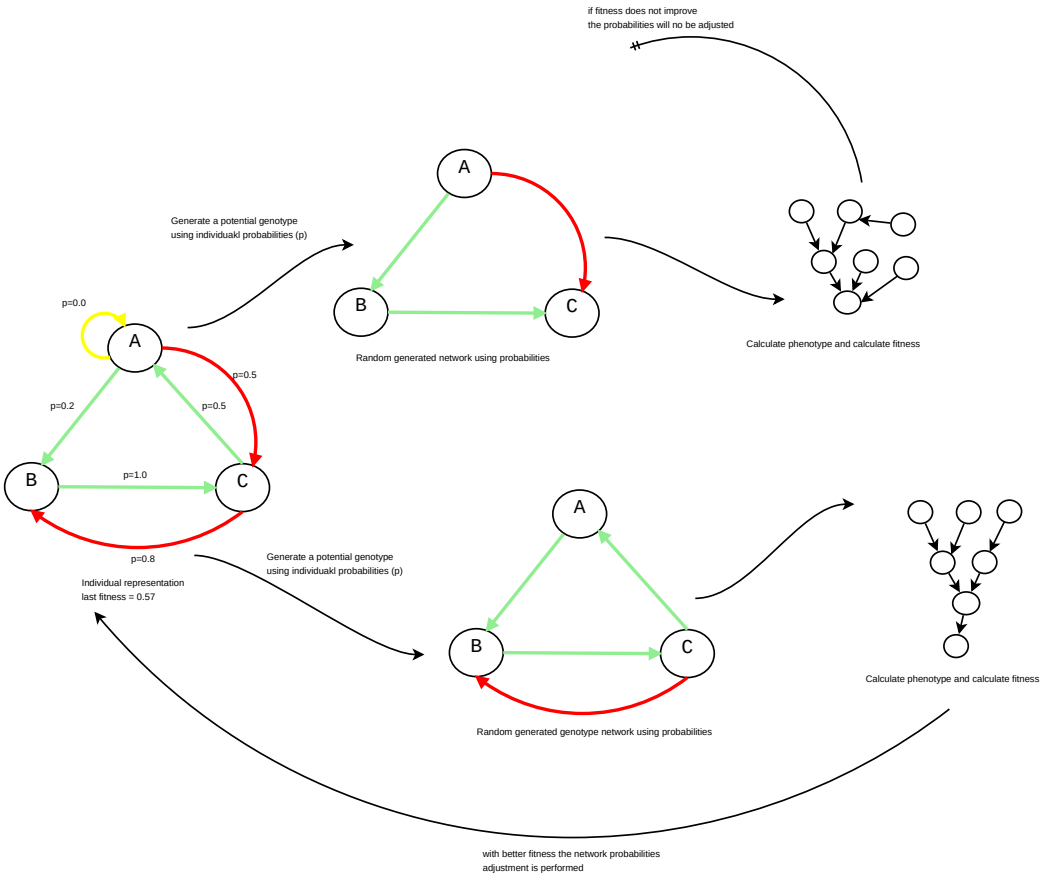


Figure 4.6: Algorithm graphical representation.

$updatedP_i = P_i * 0.8 + normalized\_occurrence_i * 0.2$. The implementation done in this project allows adjustment of the probability weight and the normalized_occurrence weight, in that way the adjustment can be more progressive or less. Experimentation shows that $updatedP_i = P_i * 0.95 + 0.05$ have a less number of local maximums but needs more iterations to find a correct genotype, it works better with difficult cases.

4.2.2 Individual representation

An individual in the context of this report means "generated network" by the system and "potential solution". Typically an individual will be part of a population and it will be adjusted or developed in each iteration.

The individual's representation is a very important implementation detail, since it conditions the complexity of the implemented algorithms. Individuals that compose the population are intensively checked time after time in each iteration, evaluating the fitness, calculating dynamics and adjusting probabilities. This is a heavy process that needs the best possible implementation.

To keep the abstraction level lower and reduce the algorithmic and procedural complexity, simple structures have been used and algorithms have been optimized.

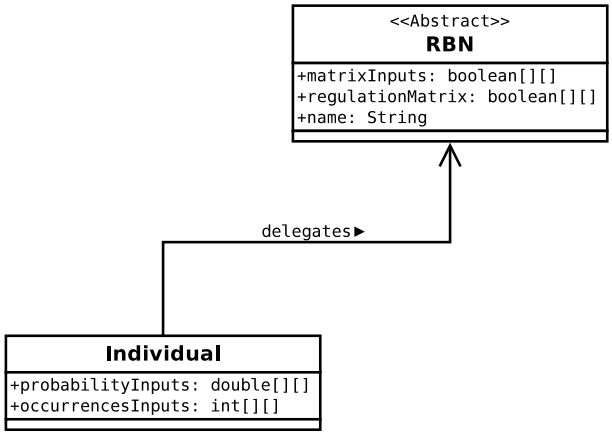


Figure 4.7: RBN - Individual classes diagram.

As can be seen in the figure 4.7 an individual decorates a RBN (could be any subclass like a DARBN) delegating the tasks related with states calculation. The individual only aggregates two matrices to keep the probabilities and occurrences used in the developmental algorithm.

4.2.3 Fitness function

After implementing the system the fitness function has been changing and been perfecting along the programming stage. The function, at the end, is pretty simple, based on the phenotypic distance between reference network and generated individual network. It is as follows:

$$fitness(i) = \frac{2^n - distance(i)}{2^n}$$

The best advantages of this fitness function implementation are that grows slowly with the phenotype distance minimization, it allows to the algorithm to get better individual along the time progressively without sharp changes. Especially for this algorithm this issue is very important because the probabilities adjustment is only done after getting a better fitness than before.

4.2.4 Developmental Algorithm Simulator

The Developmental Algorithm Simulator is the entry point to the user. It accepts parameters that will be specified in B.0.3 and dumps algorithm execution traces to study the experiment result. It can be extended through fitness function implementation.

4.3 Complexity

Unfortunately the inherent complexity to this problem is really high. As can be seen in [HJS], Hawick proposes an efficient binary representation very similar to the used in this implementation. The author presents the time complexity of calculate the true table (concept similar to space state but non valid to the current problem that needs a transition rule to

calculate the state's transitions) is 2^{2^k} and the spatial complexity is $n2^k$ (k is always de same for each node in the article's networks). Paying attention to the complexity of find attractors the authors Cheng and Hongsheng, in their article *"A Linear Representation of Dynamics of Boolean Networks"* ([CQ10]) affirm: *"finding fixed points and cycles of a Boolean network is an NP-complete problem"*.

To exemplify better the complexity of the problem, the implemented algorithms (1, 2 and 3) complexity will be studied:

- **Complexity of calculate next network's state from the actual state**
  It requires to check all the inputs to all the nodes if k is the number of inputs and n the number of nodes and in this implementation (due to the input's representation) $k = n$ always; the complexity is $O(n^2)$ (See algorithm pseudocode in 3).

- **Complexity of calculate space state**
  The space state has always 2^n possible states and is mandatory examine all the states to calculate the network dynamics; for each state the next state from the actual is calculated. The complexity to this operation is $2^n n^2$ that $O(2^n) < 2^n n^2 < O(n!)$ (See algorithm pseudocode in 2).

- **Complexity of run developmental algorithm**
  In the general algorithm the space state is calculated for each individual in the population (P) between 1 and maximun iterations (I) times. The complexity is $O(2^n) < ip2^n n^2 < O(n!)$ (See algorithm pseudocode in 1).

This complexity could be improved changing some details in the representation, like representing only the needed inputs instead all possible inputs, however this approach has been not done because the current fits better with a possible hardware representation and because is easier to parallelize.

Algorithm 1 Developmental algorithm's pseudocode

```
i = 0;
found = 0;
population = random_population(size);
while i ¡ max_iterations AND found ≠ false do
  for all individual in population do
    phenotype = dynamics(individual);
    found = fitness(phenotype);
    adjustment(individual);
  end for
  i = i+1
end while
```

Algorithm 2 Dynamics algorithm's pseudocode

```
space_state_size = 2^nodes;
dynamics = integer[nodes];
for all actual in space_state_length do
  dinamycs[actual] = individual.next(actual);
end for
return dinamycs
```

Algorithm 3 Calculate next state from actual algorithm's pseudocode

```
next = integer[nodes];
for all node in individual.nodes do
  active_inputs = individual.active(node, actual);
  next[node] = transition_rule(active_inputs);
end for
return next
```

4.4 Scalability

Undeniably the inherent complexity of this domain is a huge problem simulating large regulatory networks, the inner mechanism which allows the possibility of express a big amount of phenotypic information is in addition the most difficult point to compute. As has been noted along this report through references, several authors have attempted to optimize their implementations and algorithms to improve the process performance which it still need a big amount of time and memory to compute. Nevertheless there are some options to get results in less time; after studying the problem, paying attention to the improvement possibilities and implementing them using software, it has been concluded that the only option is the **parallelization of the implementation** (this conclusion is also present in [Had08]). As it can be seen in the previous sections each data structure and algorithm have been designed thinking in a possible hardware implementation, keeping every key point simple and fortunately several good characteristics have been found that make possible to implement this problem in a more scalable way. The main characteristics are:

- **The hardware implementation is plausible**
  The binary character of the automatons allows to implement several key points, like the calculation of state transitions or the full space state, directly using large arrays of bits with binary operators. This approach will give a big speedup to the execution not only because the higher time performance of a possible hardware implementation, but also because some operations potentially could decrease their complexity: The network state transition calculation could improve their complexity from $O(n^2)$ to $O(1)$ if the hardware architecture allows to process all the positions of big arrays in parallel. The real bottleneck using a classical programming approach is perform the process sequentially. The parallelization creates new implementation's possibilities that make it possible to achieve a much better processing time. Unfortunately some parts of the problem, more abstract (like the developmental process) are more difficult to implement using hardware but apparently they are still possible or in the worst case a potential mixed implementation (software and hardware) could be plausible.

- **The independency between each step of the algorithm**
  It means, for example, the space state calculation ($n^2 2^n$) could be splited in j subsets distributing them over several processors working on parallel. It would be specially good to perform the space state representation in a distributed way. Unfortunately the proposed algorithm to calculate the attractor's basin size is not possible without a shared memory approach, moreover it could be done in a separated step after the space state calculation with still good time profits.

The functional and data parallelism are good to implement RBNs operations, besides some parts of the developmental algorithm as evaluation of individuals and probabilities adjustment has good characteristics to do a parallel implementation. The transition rule and phenotypic distance could be potentially implemented using hardware because the simplicity of them.

4.5 Optimizations

In addition some optimizations could be done and have been implemented in some cases also in the experimentation software of this project. The most important optimization is to include **cache mechanism** for some calculations. For example, the implementation done needs to transform integer numbers to boolean arrays. This could be done directly in hardware but in the implementation of an extra algorithm is needed, obviously this calculation could be done once and save it an static resource getting a speedup.

Other small optimization is preventing unnecessary operations. For example, for some individuals the probabilities adjustment could be avoided when has

0 or 1 as probability, saving iterations along the process.

In networks with a fixed k a more optimal implementation could be done saving unnecessary inputs matrix and regulation matrix memory with the same advantages; however in the checked regulatory networks a fixed number of inputs for each node is not a regular scenario.

Chapter 5

Experimentation

After the stages of background reading and implementation the next step is the experimentation phase. Several tools have been buildt to perform this phase and during the experimentation several implementations improvements and approach adjustment have been done to get a better understanding of the global scenario. Only after experimentation it is possible to get own conclusions.[1]

5.1 Previous approach

The first step during this work's creation was to achieve the same results that the researches achieve in the previous project, it is, find regulatory networks through the developmental algorithm using classic random boolean networks. This step ensures the understanding of the problem and helps **to find possible weak points and errors** in the previous work (as has happened with some erratum and possible optimizations).

In the previous work, the author introduces a particular instance of individual to perform the experimentation over. The individual is described in the sections VI (*"Experiments and results"*) and subsection B (*" Applying the Artificial Development (AD) reverse engineering method"*). The author says: *"Figure 6a explicitly shows the genotype used in the experiments shown here: a list of 39 interactions and their probabilities of being in the network. The network used, the yeast cell-cycle Boolean network, is fully described by the items 1 to 29 in the list (when P =1). In order to perform a reverse engineering investigation using this network, we have to pretend we don't know some of its interactions, as we did in Section VI-A. In the experiment shown here, this is done by assigning different probabilities P to the 'correct' interactions (1 to 29), as if we were not sure they should be in the network that models the system, and by including other random interactions (30 to 39)".*[2]

Therefore a first phase of experimentation is performed with the previous approach and same parameters.[3].

All the experiments present in the experiments appendix where the name starts with *"Costa2009"* are those experiments based on the previous work approach, when the name starts with *"Leon2011"* are experiments based in the new approach.

5.1.1 Same scenario experimentation

The first experiments were oriented to reproduce the previous project's results. The experiment described in A.1 shows that the achieved implementation has the same capabilities to cover the previous experiment, such copies several random inputs from the

[1] Along this investigation several experiments with the same parameters has been performed, these experiments parameters, results and charts are attached to this report as appendix.

[2] There was found a mistake into this affirmation. Author affirms that the network is *"fully described by items 1 to 29 in the list"* but it is not correct. The described and used yeast cell regulatory network has a total of 34 inputs, and because that it needs 34 positions in the previous work's individual representation.

[3] All the experiments has been performed using the Developmental Algorithm Simulator with different parameter's values

reference network and introduces random inputs because, potentially, in a real scenario some known inputs and some potential inputs will be present in the individual's genotype.

After performing the experiment with the parameters described in A.1 the conclusion is that the algorithm implementation works. In all the experiment instances a solution has been found with $fitness = 1.0$. Sometimes, when $population\_size \approx 100$, it happens before iterate the algorithm, i.e. randomly. This issue evidences that the chosen experimentation case is specially next to the phenotype.

Studying the charts, sharp changes in the fitness representation occurs during the execution in all the instances. It is related with the probabilities weight used to calculate the new genotypic probabilities (4.2.1). In this example the weight used is $P_i = 0.8 + 0.2$. **This behavior could be a potential problem in the application of the algorithm to more complicated networks; it favors the local maximums, and with smaller populations or more complex scenarios (more random inputs) is easy to not find a good individual**, it happens because the probabilities grows too fast. When the individual is easy to find (as the current) the iterations are always < 80. To check this aspect the experiment A.2 performs the same parameters excepts the weight adjustment to $P_i = 0.95 + 0.05$ getting a smoother behavior and finding $fitness = 1.0$ individual. In A.2 is possible also to see the pre-search match (good individual by random), it is related with the population size. Other important detail is that with this parameters the the iterations are always < 140.

The experiment A.3 studies the algorithm behavior with a more complex network. In this scenario the network has the same copied inputs but it also has more randomly generated inputs than in the previous experiments. It produces a bigger genotypic distance and, indeed, the difficulty of finding a suitable individual grows. According to the charts the genotype adjustment is smoother and the algorithm needs more steps (because the bigger genotypic distance) but it still finds a solution. The experiment A.3 has the same principles but with also more genotypic distance than previously. It needs more steps but always finds a solution too. To evidence the problem of the sharp adjustment other experiment A.5 with more genotypic distance (but less distance than A.3 and A.4) and adjustment $P_i = 0.8+0.2$, has been performed; these charts show the problem; the local maximum happens around iteration 120 and individuals with $fitness = 1.0$ never emerge.

5.1.2 Different scenario experimentation

In the previous project, the search of new individuals has always random copied inputs from the reference network. To check the potential capabilities of the proposed algorithm, an experiment A.9 with even more phenotypic distance than the previous experiments has been performed. This experiment doesn't copy any input from the reference network (assign $P_i = 1$ over an input involves the perpetuation of this genotypic characteristic). The full genotype is always included ($P_i > 0.0$ for each input) but it does not guarantee the expression of it; this is inherent of the previous project, and is justified because if an input has $P_i = 0.0$ then it will never be expressed, an then the individual can not be found. **The algorithm's results were promising, in all the runs a good individual was found**. It took about 600 iterations (much more than previous experiments) and the chart shows how the individuals are well adjusted, starting at iteration 0 with $fitness < 0.1$ and finding individuals with $fitness = 1.0$. The same experiment with $P_i = 0.8 + 0.2$ does not find a valid solution.

5.2 DARBN approach experimentation

One of the most important motivations to accomplish this project has been trying to find plausible regulatory networks using an asynchronous and deterministic approach. In the background reading phase several affirmations of researchers have been found supporting the bigger similarity of the asynchronous and deterministic approach, looking like a better option

to perform a more realistic simulation. This work has been build over these ideas, trying to find a better way to achieve the global goals. To do that tools have been implemented, documentation has been read and experiments has been performed. In this report only a few of the experiments has been collected because the results were always similar.

The experiments A.10, A.10 and A.6 have been performed using a DARBN network. The results are always similar. The fitness grows from $fitness < 0.1$ to $0.6 \geq fitness < 0.8$ but $fitness = 1.0$ is never found. These experiments have as parameters a small possible values to periods (P) and translations (Q), and together with the dynamics calculation process ensures to perform the state transition (evaluating each state several times to ensure the state transition) but still with these advantages the algorithm never finds a solution. Other experiment has been done to check possible better scenarios, the experiment A.10 copies almost all the reference genotype in each random individual. With the classic approach the $fitness = 1.0$ is found always in the initial random population, in the ARBN approach is never found after thousands of steps.

The proposed approach that uses ARBNs has genotypic representation and the proposed algorithm is not capable to find solutions. The inclusion of two more variables (periods and translations) involves a bigger space of solutions, and with the used technology and implementations no satisfactory solutions or promising signs have been found.

Chapter 6

Conclusions

After accomplishing the experimentation and studying the results several useful conclusions have been found. It was confirmed that the previous project's approach, using CRBNs, works with the proposed parameters but it does not work when the genotypic distance is incremented, however performing small improvements it works very well also in experiments that involves a big genotypic distance (experiment A.7), which makes it promising to use this methods over other more complex models. Unfortunately the new approach, using DARBNs instead CRBN, does not work very well, It seems that the inclusion of new variables to generate the network dynamics makes the space solution bigger and the method never finds a good solution. It is unclear if the problem is the use of DARBNs or the product between the algorithm and the DARBNs, but together are not a good method. Other approaches could be checked, modifying the algorithm allowing it to adjust all the variables (inputs regulation, p and q) but it would probably imply a bigger solution space.

Other important conclusion is about the huge complexity of the handled domain. The internal details of the regulatory networks show the incredible possibilities of these networks and how they can be represented with a very simple structures; the second point opens a big field of potential implementation approaches using hardware, distributed computation and, possibly, new emerging technologies (as, curiously, DNA computing that allows huge parallelism).

According with the experimentation, has been found several possible weak points about the previous project, these details are sometimes related with parametric values or with details that could be not considered (to check it would be necessary check the algorithm's code). The possible weaknesses found in the previous project are:

- **Possible problem of phenotypic expressions being ignored was pointed out**
  It was pointed the possible problem of phenotypic expressions ignored. The previous work only talks about fixed points, but there are more kinds of attractor. To yeast cell this is not relevant because does not have cycles but applying the same algorithm with other problems it could not work. In this project the cycles have been found in phenotypes of potential individuals and specially when the DARBN as genotype approach is applied.
- **Too high adjustment**
  As shows the section 5.1.1 and the experiment A.5, the adjustment weights chosen in the previous project were too high and in difficult searches it could derive in local maximums, non finding the target network.
- **Erratum found in the report**
  Some erratum have been detected in the previous project. These details have been collected as foot notes in this report.
- **Immutable regulation**
  The actual **algorithm does not allow the po-**

tential adjustment over the regulation nature of an input. When, randomly, an input is designated as positive or negative it will remain with the same regulation during all the execution. This issue conditions the final genotype, therefore the phenotype too, and this property can not be adjusted with the actual implementation. On the other hand this lack makes the space solution smaller.

- **Big initial random generation dependency** Since the algorithm does not include a mechanism like mutation, all the potential genotypes are conditioned by the first random generation. If an input has $Pi = 0$ will never be expressed, therefore when randomly (at the individual creation step) a correct input is not set as possible $0 < Pi < 1$ is pruning the potential effectivity of the algorithm. On the other hand, with the current implementation, it is possible to ensure that all the inputs could be potentially abject setting the parameter random inputs as n^2, but it makes the space of solutions bigger (and is inherently huge with less random inputs really).

Finally, after accomplishing this project some ideas about possible alternative ways to solve the problem have emerged. According to conclusions, solving some algorithm's existing limitations and improving the implementation to improve the speed of execution could be accomplished to achieve better results.

Potential algorithm improvements:

- **Parallelization**
- **Hardware implementation**
- **More flexible genotypic adjustment**

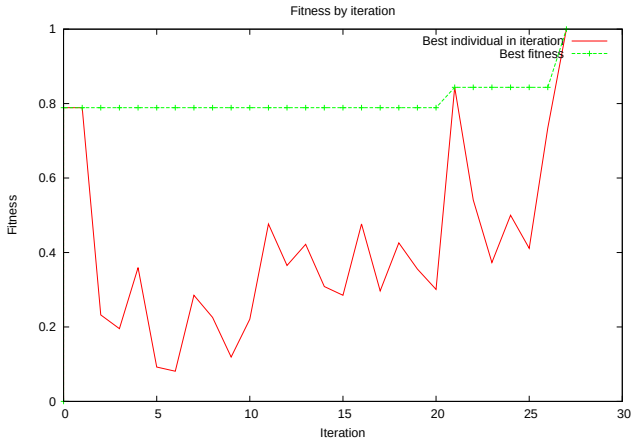
Additionally a small set of simulation tools has been build and could be used in other investigations. All the software artifacts are described in the section B, explaining how to use and extend them.

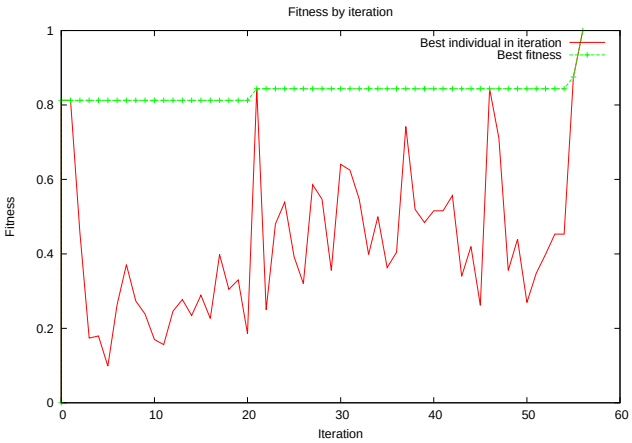
Appendix A

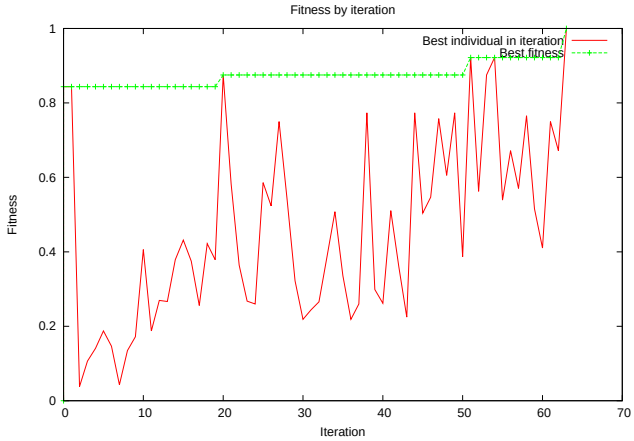
Experiments appendix

A.1 Experiment Costa2009

Max iterations	1000
Population size	50
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	6
Probability weight	0.8
Ocurrences weight	0.2
Maximun period (p)	1
Max transition (q)	0
Time per iteration	258 miliseconds
Best fitness found	1.0

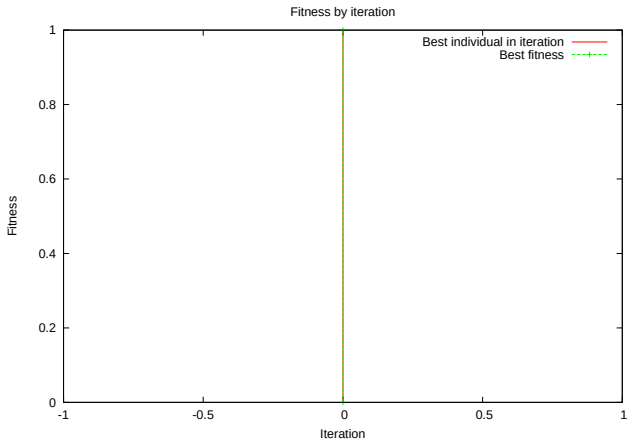


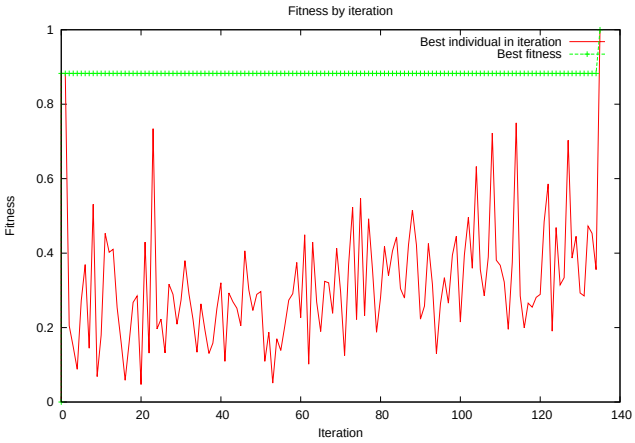


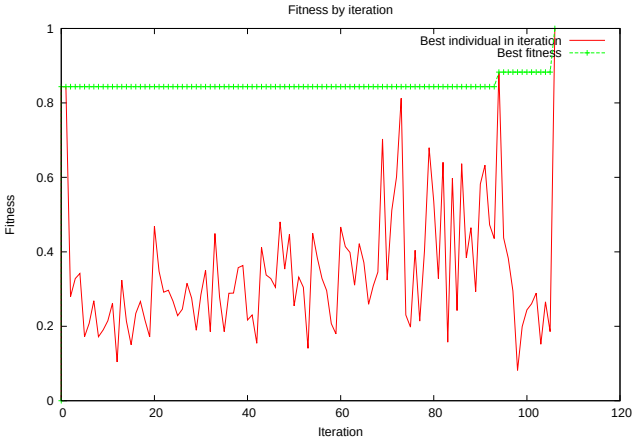


A.2 Experiment Costa2009r1

Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	6
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	1
Max transition (q)	0
Time per iteration	596 miliseconds
Best fitness found	1.0



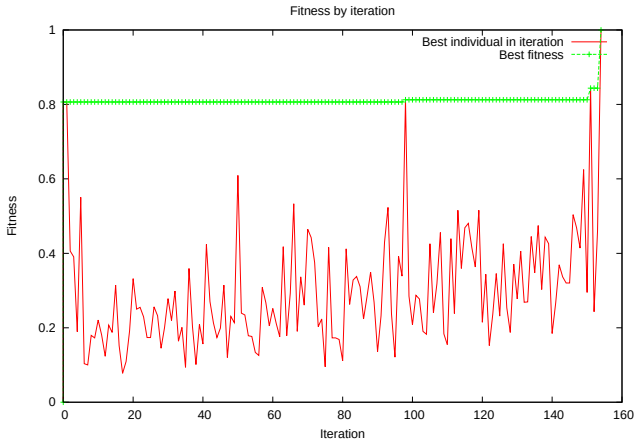




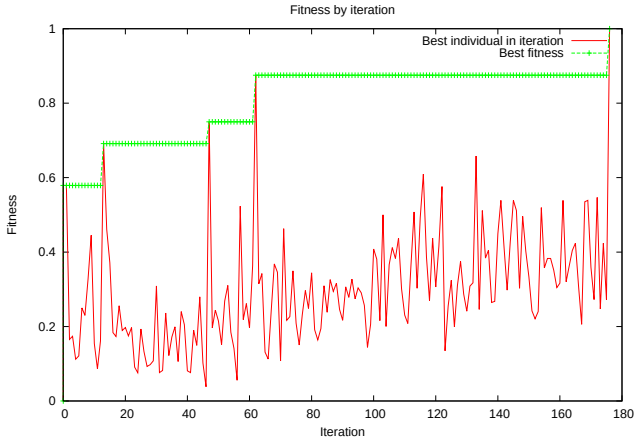
A.3 Experiment Costa2009r2

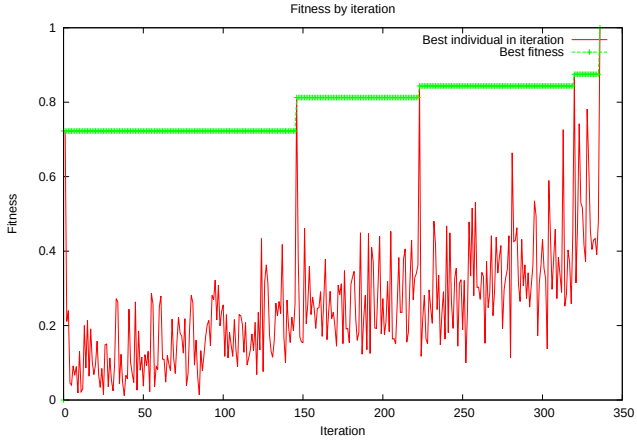
Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	10
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	1
Max transition (q)	0
Time per iteration	521 miliseconds
Best fitness found	1.0

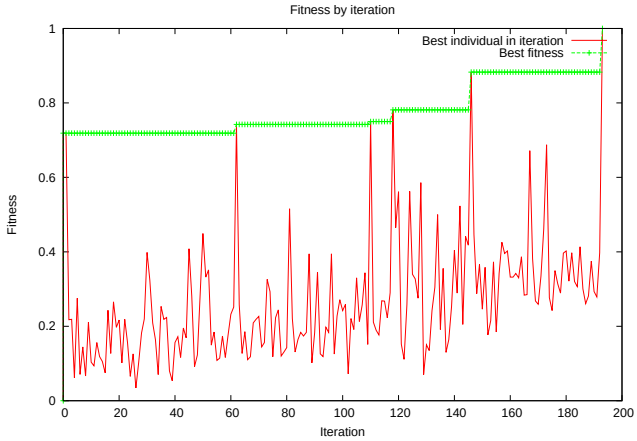
A.4 Experiment Costa2009r3

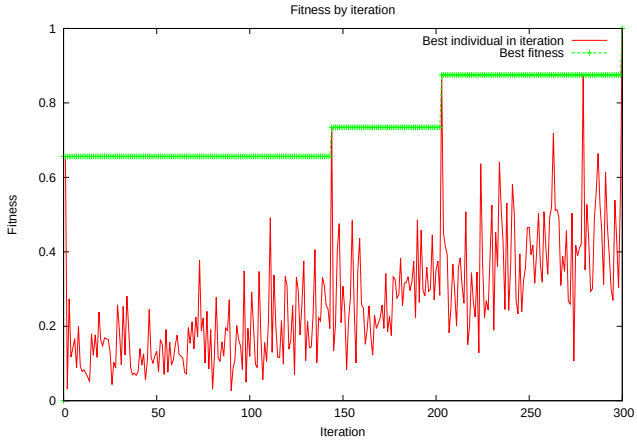


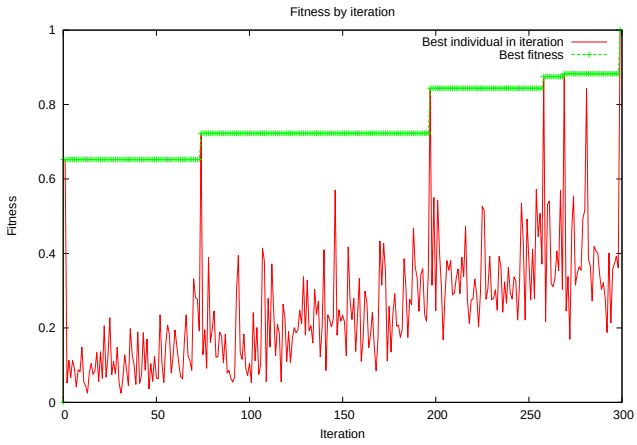
Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	10
Random inputs	10
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	1
Max transition (q)	0
Time per iteration	521 miliseconds
Best fitness found	1.0

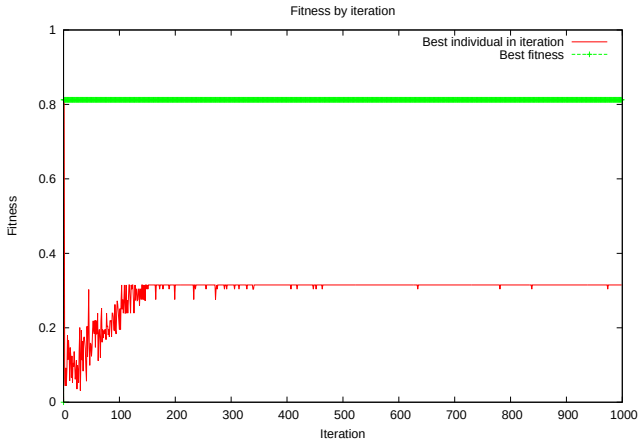






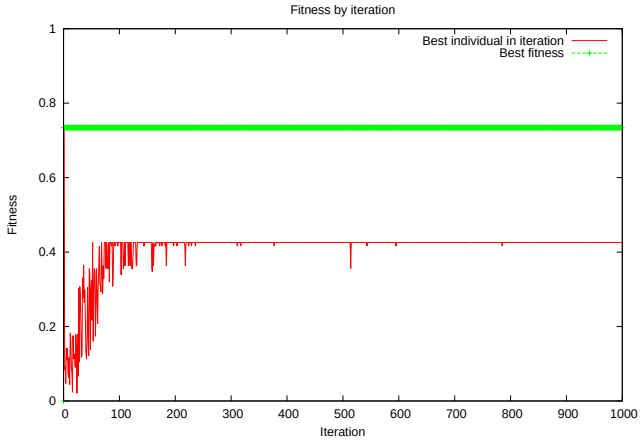


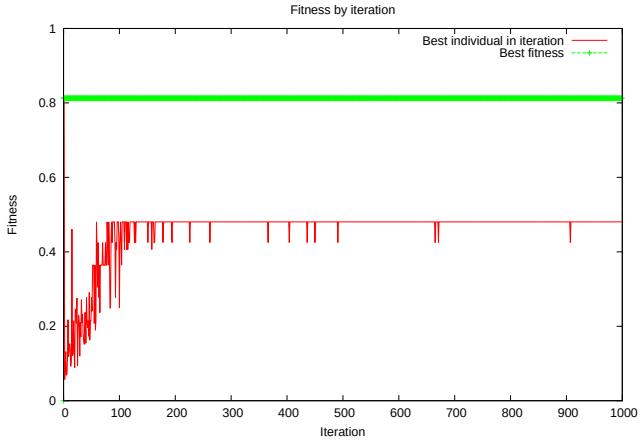




A.5 Experiment Costa2009r4

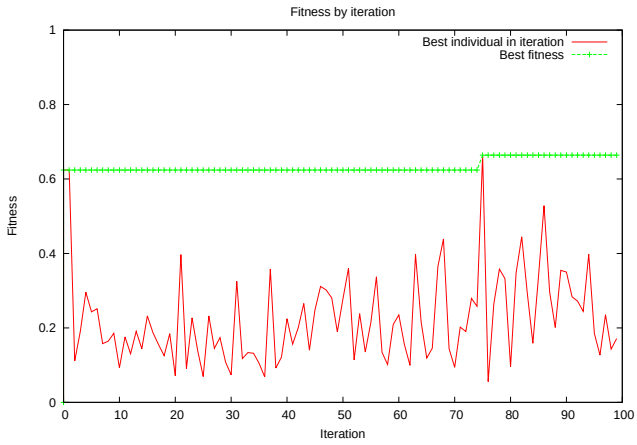
Max iterations	1000
Population size	50
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	10
Random inputs	6
Probability weight	0.8
Ocurrences weight	0.2
Maximun period (p)	1
Max transition (q)	0
Time per iteration	267 miliseconds
Best fitness found	0.8125

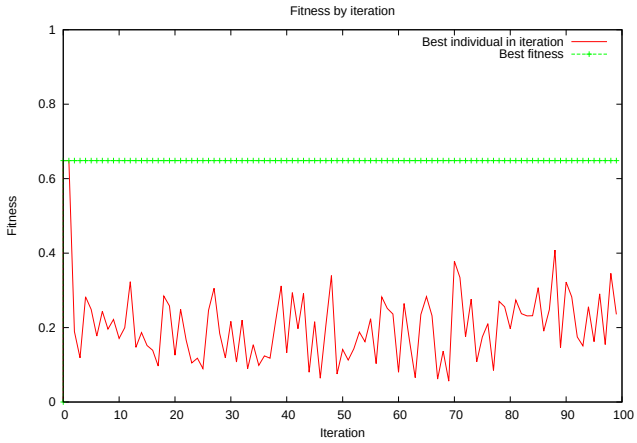


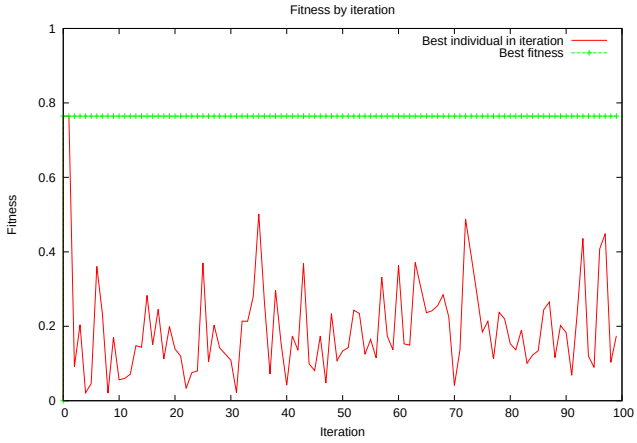


A.6 Experiment Leon2011r1

Max iterations	100
Population size	1000
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	5
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	2
Max transition (q)	1
Time per iteration	4818 miliseconds
Best fitness found	0.65625





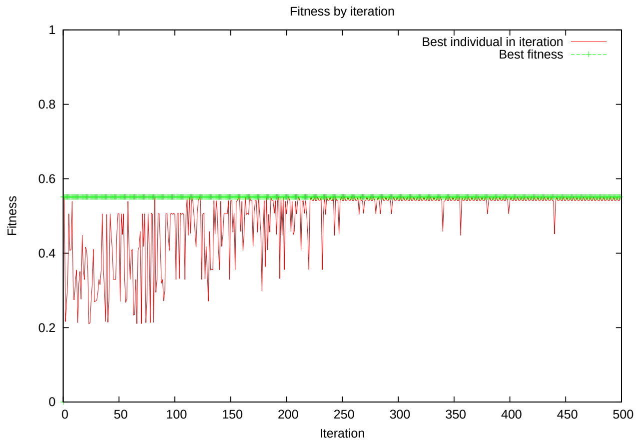


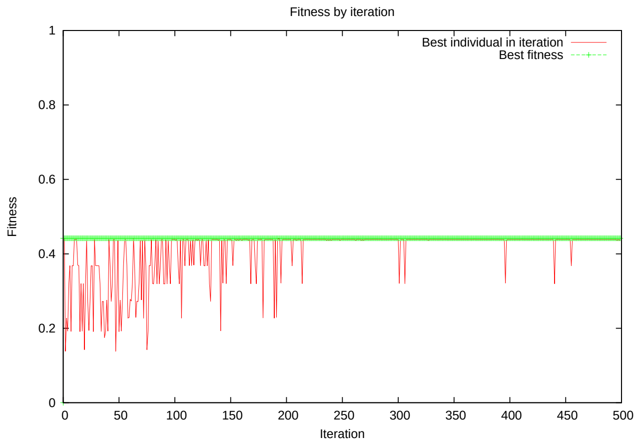
A.7 Experiment Leon2011r2

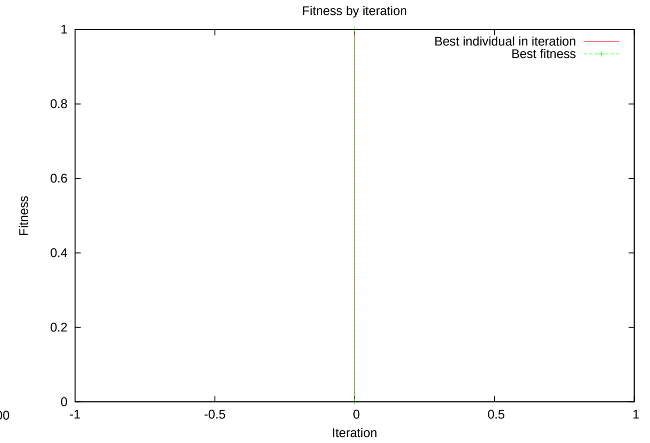
Max iterations	500
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	30
Random inputs	0
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	3
Max transition (q)	2
Time per iteration	468 miliseconds
Best fitness found	0.52197

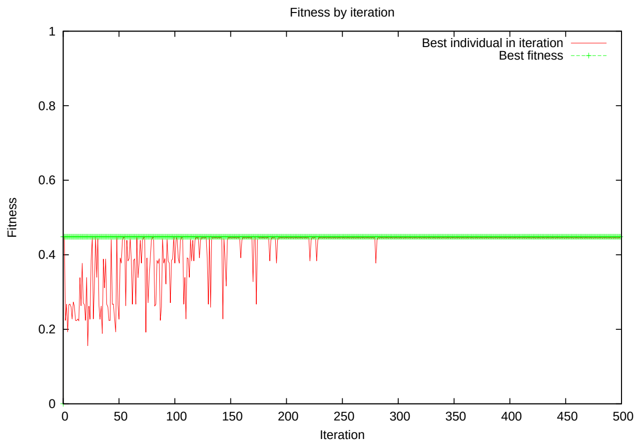
A.8 Experiment Leon2011r3

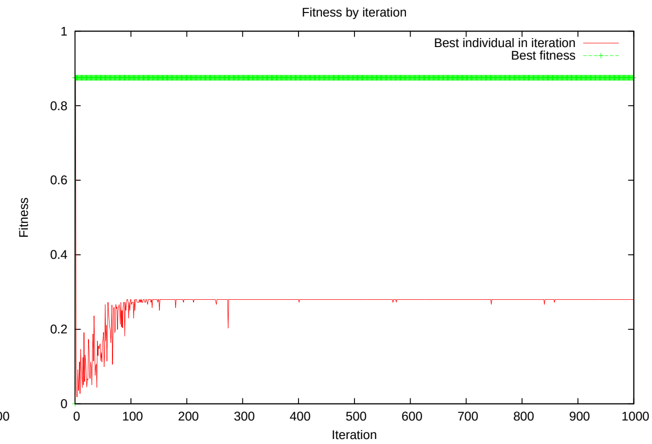
Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	0
Random inputs	5
Probability weight	0.8
Ocurrences weight	0.2
Maximun period (p)	1
Max transition (q)	0
Time per iteration	513 miliseconds
Best fitness found	0.84375

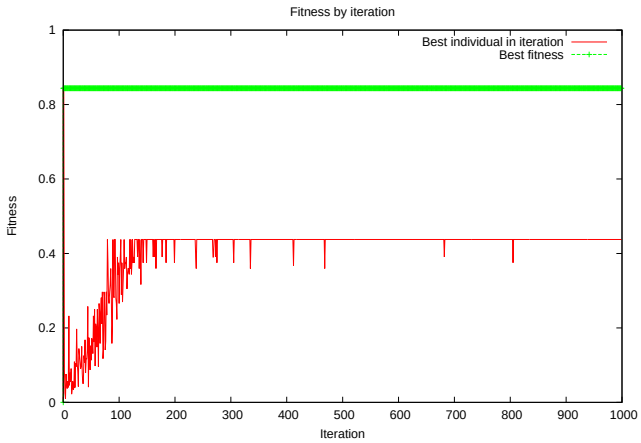


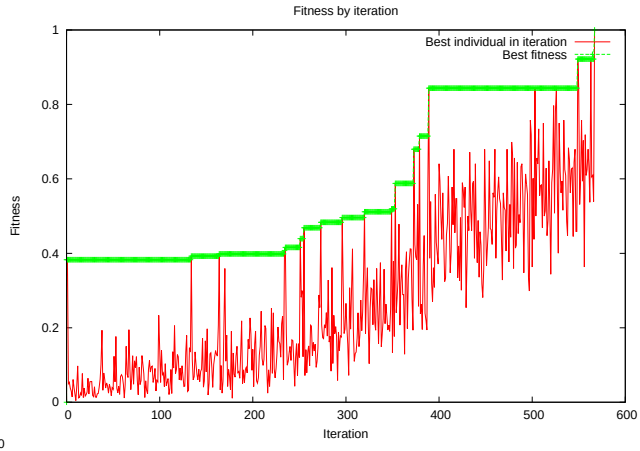






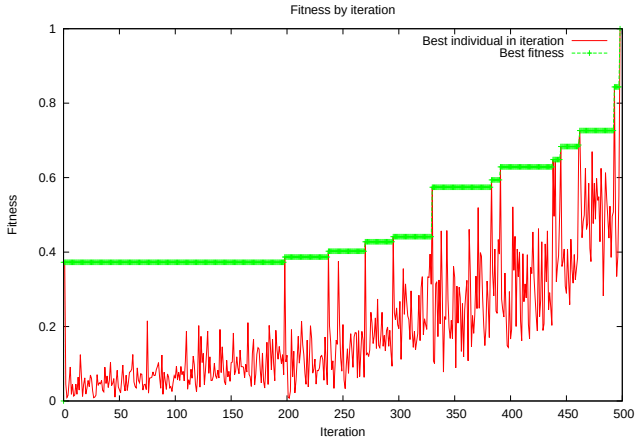


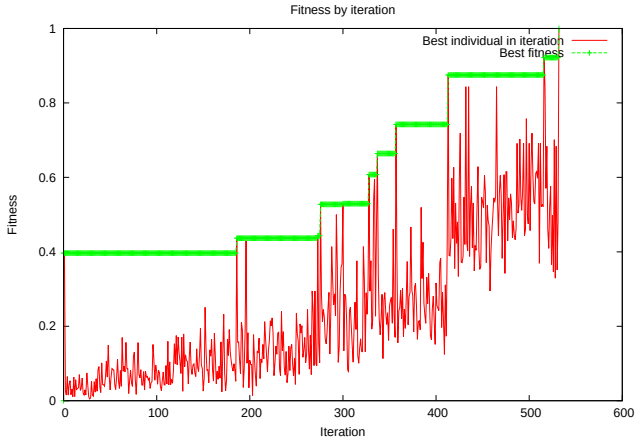




A.9 Experiment Leon2011r4

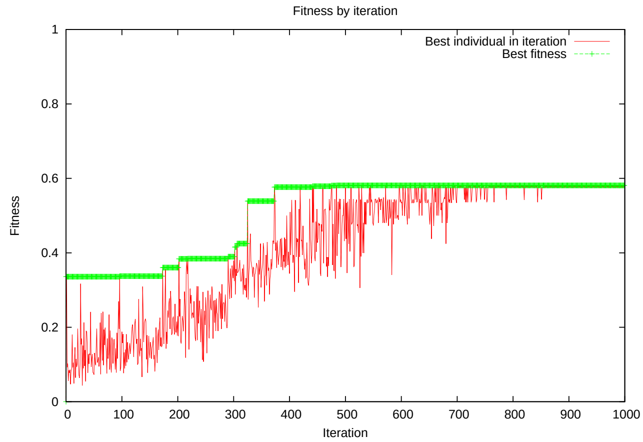
Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	0
Random inputs	20
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	1
Max transition (q)	0
Time per iteration	520 miliseconds
Best fitness found	1.0

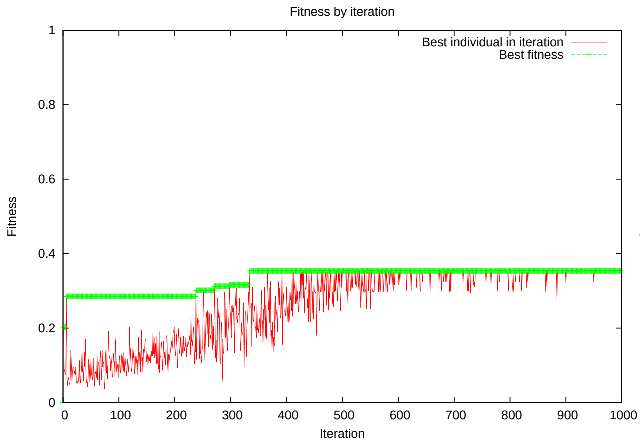




A.10 Experiment Leon2011r5

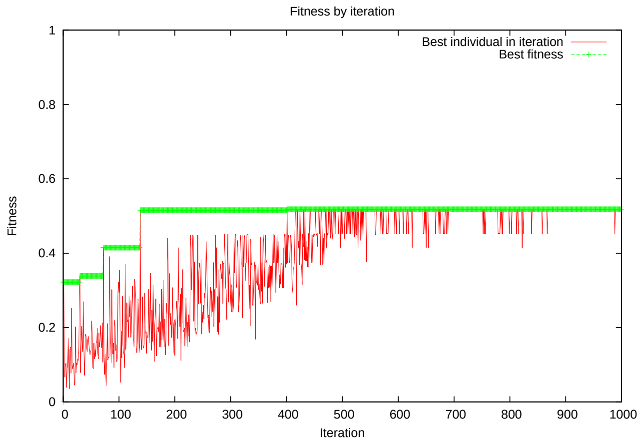
Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	6
Probability weight	0.95
Ocurrences weight	0.05
Maximun period (p)	3
Max transition (q)	2
Time per iteration	469 miliseconds
Best fitness found	0.57861

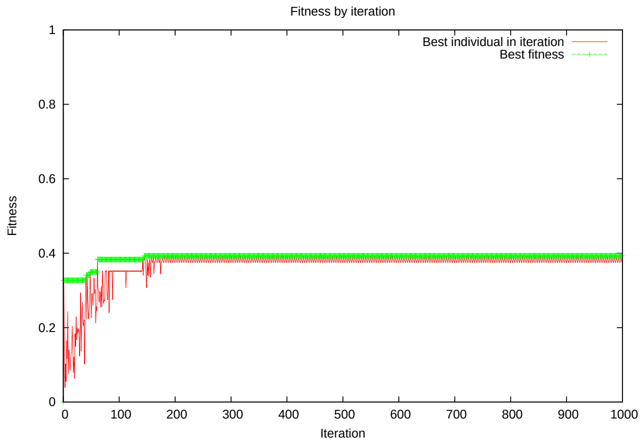


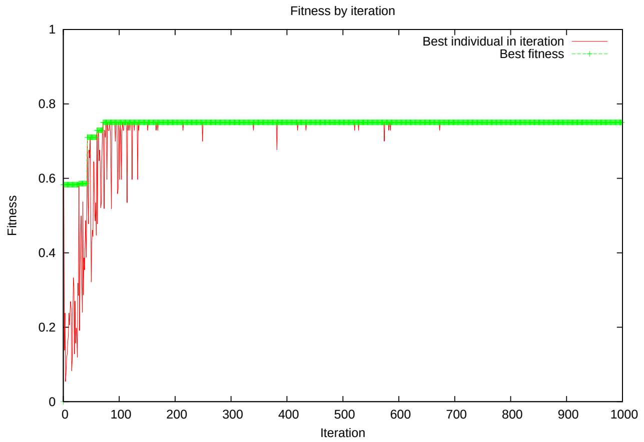


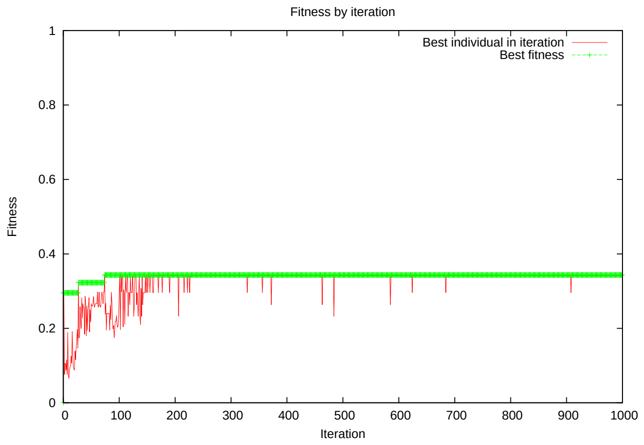
A.11 Experiment Leon2011r6

Max iterations	1000
Population size	100
Transition rule	YeastTR [STH09]
Fitness function	SPDistance (4.2.3)
Reference RBN type	CRBN
Copied inputs	19
Random inputs	6
Probability weight	0.8
Ocurrences weight	0.2
Maximun period (p)	3
Max transition (q)	2
Time per iteration	465 miliseconds
Best fitness found	0.34473




Fitness by iteration
Best individual in iteration
Best fitness
Fitness
Iteration
0
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700
800
900
1000


Fitness by iteration
Best individual in iteration
Best fitness
Fitness
Iteration
0
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700
800
900
1000


Fitness by iteration
Best individual in iteration
Best fitness
Fitness
Iteration
0
0.2
0.4
0.6
0.8
1
100
200
300
400
500
600
700
800
900
1000

Appendix B

Implemented tools

The implemented tools have been written using Java. It means that to execute the provided tools mandatory use the java technology. Both tools are packaged as jar executables, and to run them is used the command *"java jar tool.jar"*.

The tools uses the console as communication channel with the researcher by lines of text. To use the tools the is needed to specify the execution parameters. For help inline execute it with the flag "-h".

B.0.1 RBN Tools

Usage

This tool was designed to works as library but also runs as executable. The goal is allow the researcher to study a concrete network's dynamics. Is possible to see information about the network's attractors and several steps from an initial network state. Also is possible to run a concrete topology with different kinds of RBN.

B.0.2 Parameters

B.0.3 Extending tools

The system has been designed to be easily extensible. Both simulator implements interfaces to extend the functionality and perform new experiments.

RBN tools implement different kinds of RBN (CRBN, DARBN and ARBN). It is possible to extend with a new kind of RBN inheriting from RBN class and implementing a simple method nextStep (that depends of the RBN update schema and characteristics).

Additionally, RBN Tools allow the **creation of new experiments**. To do that is necessary to implement the interface **Experiment** and the corresponding **TransitionRule**. An experiment instance will configure the RBN's topology and contains the transition rule implementation. As an example can be checked the implemented experiment class **YeastCellCRBN**. To use a new implementation is valid to compile the class, include it in the classpath and run the simulator with the parameter **-e set with the full described new class name and namespace**.

Developmental-Algorithm-Tools

Usage

This tools allows to the researcher to use the described developmental algorithm. The algorithm is fixed but all the implied parameters can be set using running options. This tools runs the algorithm and shows for each iteration the progress. Works creating a trace files and directories to save them. It allows the study of the performed experiments. Is specially useful to run experiments as a batch process. The directory name to save it is provided as parameter, and if it already exists only the fitness trace will be dump. The idea is perform the same experiment several times and observe de behavior. This implementation always uses an DARBN to run dynamics (with maxp=1 and maxq=0 a CRBN behavior will be performed).

B.0.4 Parameters

This tool allows the user to set several parameters that are important into the algorithm. These parameters can be checked inline using the parameter *-help*.

- **-copiedInputsFromInputNetwork** Number of copied inputs from the reference genotype.
- **-experiment** Name of the experiment (mandatory, a directory will created with it).
- **-fitness** Fitness function to use during the execution.
- **-help** Prints helps.
- **-inputNetwork** Experiment (from RBN-Tools) to use as genotype.
- **-iterations** Number of max iterations.
- **-maxP** Set maxP parameter to indivuduals.
- **-maxQ** Set maxQ parameter to indivuduals.
- **-ocurrencesWeight** Adjustment ocurrence weight.
- **-populationSize** Number of individual during the execution.
- **-probabilityWeight**Adjustment probabilities weight.
- **-randomInputs** Number of total inputs generated randomly in the network.
- **-rbnKind** Type of RBN to simulate the experiment.
- **-verbose**

B.0.5 Extending tools

Developmental-Algorithm-Tools can be extended easily through the implementation of available interfaces present on the package. The only implementable interface is the **FitnessFunction** interface. It allows to create new ways to evaluate the individuals. To use it is enough to include the compiled class into the classpath and specify the parameter **fitness** to the full qualified name of the implemented class. To perform more interesting extensions it can be combined with the RBN-Tool interfaces, changing the network topology, transition rule, etc.

B.0.6 Code

Comments about source code.

- The code is full documented using javadoc system, it can be found package with the source code.
- The code is compiled and packaged using Maven.
- To run the system is needed Java 1.6 interpreter.
- The system executable is packaged as .jar.

Bibliography

[CQ10] Daizhan Cheng and Hongsheng Qi. A linear representation of dynamics of boolean networks. *Automatic Control, IEEE Transactions on*, 55(10):2251 –2258, oct. 2010.

[Ger02] Carlos Gershenson. Classification of random boolean networks. *Artificial Life*, Li(2002):1–8, 2002.

[Ger04] Carlos Gershenson. Introduction to random boolean networks. *Arxiv preprint nlin/0408006*, 2004.

[Had08] Pauline Haddow. Evolvable hardware: a tool for reverse engineering if biological systems. *ICES*, 2008.

[HJS] K. A. Hawick, H. A. James, and C. J. Scogings. ABSTRACT Simulating Large Random Boolean Networks.

[LLL+04] Fangting Li, Tao Long, Ying Lu, Qi Ouyang, and Chao Tang. The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences of the United States of America*, 101(14):4781–4786, 2004.

[STH09] Cristina Costa Santini, Gunnar Tufte, and Pauline Haddow. Bio-inspired reverse engineering of regulatory networks. pages 2716–2723, 2009.

[Wue98a] A Wuensche. Genomic regulation modeled as a network with basins of attraction. *Pacific Symposium On Biocomputing*, 3:89–102, 1998.

[Wue98b] Andrew Wuensche. Classifying cellular automata automatically; finding gliders, filtering, and relating space-time patterns, attractor basins, and the z parameter. *Complexity*, 4:47–66, 1998.