Stochastic Search Genetic Algorithm Approximation of Input Signals in Native Neuronal Networks

Andrei Anisenia

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the

MSc degree in Computer Science

with specialization in Bioinformatics

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

The present work investigates the applicability of Genetic Algorithms (GA) to the problem of signal propagation in Native Neuronal Networks (NNNs). These networks are comprised of neurons, some of which receive input signals. The signals propagate though the network by transmission between neurons. The research focuses on the regeneration of the output signal of the network without knowing the original input signal. The computational complexity of the problem is prohibitive for the exact computation. We propose to use a heuristic approach called Genetic Algorithm. Three algorithms are developed, based on the GA technique. The developed algorithms are tested on two different networks with varying input signals. The results obtained from the testing indicate significantly better performance of the developed algorithms compared to the Uniform Random Search (URS) technique, which is used as a control group. The importance of the research is in the demonstration of the ability of GA-based algorithms to successfully solve the problem at hand.

Acknowledgements

The research thesis was done under the supervision of Prof. Dimitrios Makrakis in the Faculty of Engineering, University of Ottawa. I would like to thank all the people who have helped me along the way.

Firstly, I wish to express my heartfelt thanks to my supervisor Prof. Dimitrios Makrakis for his guidance and encouragement throughout my studies at the University of Ottawa. Without his knowledge, advice and mentoring the completion of this research would have been impossible. I am very grateful to him for his sincere sympathy and invaluable support.

I would like to acknowledge Prof. Franz Oppacher for his deep knowledge of genetic algorithms, which provided inspiration for the development presented in the current work. A special mention has to be made to Grigory Chudov, my friend, for his help with computational issues. I would like to thank the School of Electrical Engineering and Computer Science at the University of Ottawa for providing all the necessary assistance to complete my work.

I would like to express special gratitude to my wife, Tatiana Kolechkina, for her patience and unconditional support throughout my studies and also for invaluable help with Matlab.

Notations

Abbreviations:

NNN	Native Neuronal Network
ANN	Artificial Neural Network
GA	Genetic Algorithm
BFS	Breadth First Search
URS	Uniform Random Search
SGA	Simple GA
FIGA	Fixed Input GA
FGA	Fusion GA
CISGA	Complex Input Search GA

Terms:

Neuron	A type of biological cell, capable of transmitting electrical signals along its cell membrane.
Potential	The difference in electric charge between the two sides of a neuron's membrane. It changes in time and moves between the adjacent membrane areas, therefore is usually interpreted as "moving" potential.
Dendrite	Branching structure of a neuron cell, which can receive incoming electrical signals, transformed into potentials, moving towards the other parts of a neuron (soma, axons).

Soma (cell body)

Central part of a neuron, where potentials arriving from dendrites are summed.

Axon — Branching structure of a neuron cell, which transmits potentials away from the soma of a neuron.

Axonal terminal

The end of axonal branch where potentials arrive. This part of a neuron forms contacts or synapses with the neighbouring neuron's dendrites, allowing for transmission of electrical signals between the neurons.

Synapse — A point of contact between two neuron cells. It consists of an axonal terminal of the first neuron, short extracellular gap between the membranes of the two neurons and the dendrite's membrane of the second neuron.

Neurotransmitter

A molecule released from the axonal terminal of a neuron at the synapse that is able to bind a dendrite's membrane of another neuron, causing generation of a potential in the dendrite's membrane.

Synaptic strength

Value given to each synapse, indicating the power of the electrical signal that the synapse is able to generate.

Local potential

A potential initialized at a dendrite, propagating towards soma. These potentials decay as they propagate.

Spike (/ global potential / action potential)

Potential actively generated by a neuron inside the soma, propagating away from the soma along the axons. These potentials retain the same strength as they propagate until reaching axonal terminals.

Excitation threshold

Value of potential required to be reached or passed by the potential reaching the soma in order for a spike to be generated. Local potentials arriving from the dendrites contribute towards reaching this value.

Refractory period

A period of time during which no spike can be generated in a neuron. It immediately follows the previous spike, generated at the neuron.

Interference

Effect of signal propagation interruption in NNNs caused by the neurons, which enter refractory period after spiking, and therefore are unable to transmit other incoming signals.

Contents

List of tables

List of tables (cont.)

List of figures

List of figures (cont'd)

Chapter 1:

Introduction

1.1 Research motivation

The organization and structure of biological networks have become an important research direction in recent years due to the potentially huge impact on the public health and owing to the increased computational power and algorithmic advances. Prediction and utilization of biological networks' behaviour is now one of the crucial objectives in the current research related to computational biology.

Among the huge amount of different types of biological networks the best known and the most investigated are gene regulatory networks [1], protein interaction networks [2] and metabolic pathways [3]. The scientific knowledge accumulated in these areas allows fast and efficient investigation of arising research questions. However, this is not the complete list of biological networks. Among the other important types of biological networks that received much less attention by computer science researchers are native neuronal networks (NNNs). It can be contributed to the fact that in the past, the pace of technological development had not reached the level of allowing efficient analysis of such networks.

The interest in NNNs originates in the mystery of the animal brain and, particularly, of the human brain. These complex systems consist of relatively simple and well described cells called neurons, but when acting as a group, they are able to give rise to behaviour as complex as the human behaviour. The ability to learn and accommodate changes in the environmental conditions is only one item in the large list of functions the brain performs. Another important contribution to the interest in NNNs is the medical challenge of treating neurological disorders in humans. Interference in neurological processes, which regulate the functioning of the brain, is not possible without deep understanding and thorough analysis of the affected parts of the network under consideration.

The required analysis of NNNs may differ depending on the particular objective of treatment or research. The prediction of network's functioning under varying conditions is one of the required types of the analysis, since curing neurological diseases should be supported by the ability to foresee the changes in the behaviour of the network at hand [4], if costly or risky artificial adjustments, e.g., medications or direct interference, are to be employed. The present work introduces the tool and the underlying algorithms for the prediction of signal propagation in NNNs, the basic physiological process occurring in the brain networks.

1.2 Background on neuronal networks

Analysis of signal propagation in NNNs requires understanding of the biologically relevant details underlying this phenomenon. The signals, which are present in the human brain, are the electrical impulses transmitted between neurons. After being transmitted from one neuron to another, electric signals (from now on we will call them simply signals) travel through the network. We refer to this phenomenon as signal propagation. This process is governed by the physiological characteristics of single neurons, the basic constituents of NNNs, which are densely interconnected among each other.

1.2.1 Single neuron

A neuron is a biological cell capable of transmitting electrical signals along its membrane. Fig. 1.1 provides a schematic representation of a neuron. The propagation of a signal along the membrane is called potential. A neuron cell membrane has a number of input points, where signals are initiated, and a number of output points, where the signals terminate. The input points are situated in so-called dendrites of a neuron and the output points are situated in so-called axons of a neuron [5]. Connections between neurons are located in the regions called synapses, characterized by a close proximity of an output point of one neuron cell's membrane to an input point of a neighbouring neuron cell's membrane [6].

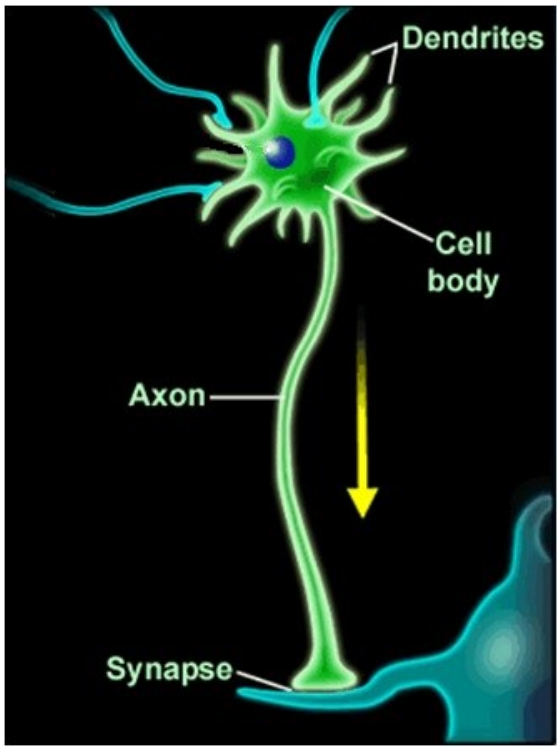


Figure 1.1 Schematic representation of a neuron. The yellow arrow shows the direction of electrical signal propagation (potential). Source: Morphonix LLC (http://morphonix.com/software/education/science/brain/game/specimens/neuron_parts.html).

A synapse acts as an environment for transferring a signal between two neighbouring neurons. The schematic view of the synapse is shown in Fig.1.2. Signals reaching an axonal terminal of the first neuron cause the release of molecules called neurotransmitters into the synapse, where they diffuse towards the dendrite's membrane of the neighbour neuron. The amount of neurotransmitters released at different synapses is different, even for the same neuron, therefore producing different effect on different neighbour neurons. This is called synaptic strength and it causes signals initialized at different synapses to vary in value within the same neuronal cell.

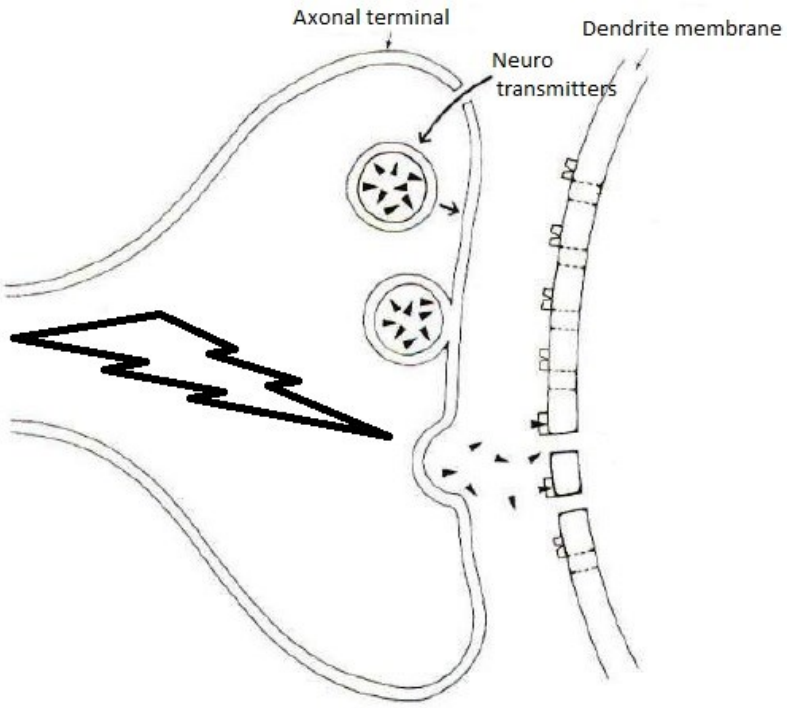


Figure 1.2 Schematic view of a synapse. The lightning symbol represents how arriving potential causes release of neurotransmitter molecules into the extracellular gap, where they diffuse towards another neuron dendrite's membrane to generate a new potential in that neuron. Source: CNS Clinic Human Neurophysiology (http://www.humanneurophysiology.com/synapse.htm)

1.2.2 Neuronal signal generation laws

There are several crucial laws governing the generation of signals in neuron cell membranes, which will be used further in the present work. Many biologically relevant details are omitted from the following description, since they will not be encoded in the algorithmic models implemented in the work. Those details include different types of molecular receptors at the surface of neurons' membranes, activated by different neurotransmitters and regulating ion flows through the membranes, which in turn create potentials. Types of receptors, ions and neurotransmitters are omitted from the present work, as well as their precise quantities, since each particular combination of these components impacts the strength of transmitted signals, which in our work is presented (on a higher abstraction level) by the synaptic strength discussed earlier.

Signal propagation through a neuron consists of three main stages. The process of spike generation and propagation is visualized in Fig.1.3. First, all input signals propagate from dendrites to the neuronal cell body, which is called soma. At this stage the signals are referred to as local potentials, since they propagate locally in the dendrites and the global potential has not been initiated yet. Second, at the soma, all the arriving local potentials are summed, yielding a particular summation value at each moment in time. The resulting sum should exceed a constant threshold, called excitation threshold, at some moment in time in order to generate a global potential or a spike. A spike is the signal, which is generated at the soma of a neuron and propagates away from the soma along the axons (branching structures rooted at the soma) until reaching axonal terminals and causing the release of neurotransmitters into synapses. A process of summation of local potentials in the soma is called spatial summation. This name reflects the fact that the summation depends on the spatial configuration of input synapses relatively to the soma and on the topology of the cell surface.

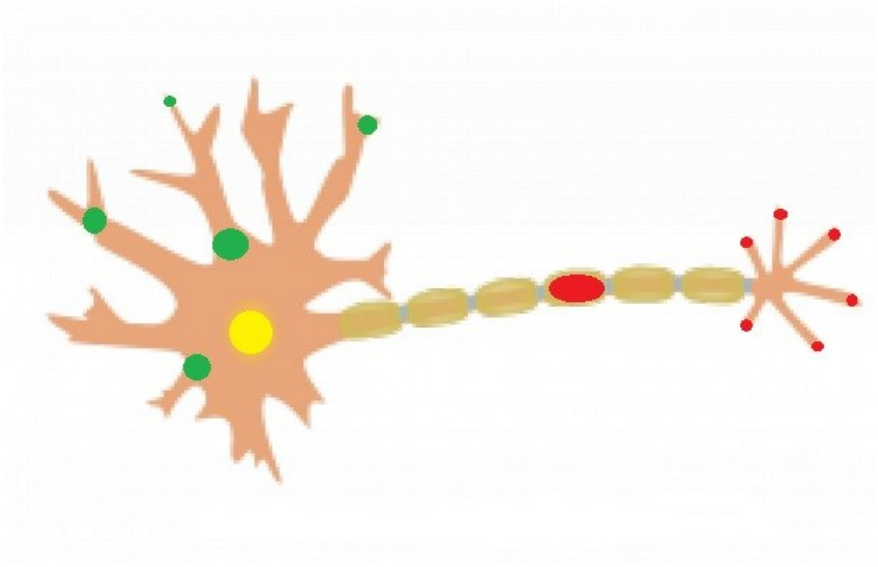


Figure 1.3 Visualization of spike generation and propagation phases. Local potentials (green circles) propagate from the dendrites towards soma, where the summation of potentials is performed (in the yellow circle). If the sum reaches excitation threshold, a spike is generated and it propagates away from the soma along the axon (red oval). At the end, the spike reaches all axonal terminals (small red circles) and causes release of neurotransmitters into corresponding synapses.

It is important to mention that neuronal spikes are binary in nature, since they are either not generated at all or generated with the same constant value. The value itself is irrelevant. The importance of a spike is in the activation of the output synapses of a neuron, if a spike is generated. The signals, supplied to a neuron as an input over a period of time, and the signals, generated by a neuron as an output over the same period, can be encoded as sequences of binary events, provided that appropriate time discrimination into discrete moments is defined.

Local potentials differ from spikes not only by the part of the neuronal cell where the propagation occurs, but also by the dynamics of their strength. While local potentials decay as they propagate along the dendrites, spikes retain the same strength during their propagation along the entire axon. That is why local potentials, if their sum is not sufficient to initiate a spike at any moment, just decay in the soma or the axons and, as a result, cannot reach axonal terminals.

Another important property of spike generation is the value that local potentials may contribute to the summation towards generating a spike. Local potentials may have either positive or negative values, corresponding to excitatory or inhibitory input synapses, respectively. As a result, activation of excitatory input synapses will increase the ability of a neuron to generate a spike by adding a positive value to the summation result, while activation of inhibitory input synapses will decrease the ability of a neuron to generate a spike by adding a negative value to the summation result.

After the termination of a spike at the axonal terminals there is a time period, when the neuron is unable to generate new local potentials even if there are neurotransmitters bound to its membrane at the dendrites' side of the cell. This period is called a refractory period. Refractory period is responsible for creating the so-called interference effects in the network, when several excitation signals reaching the same neuron can be blocked by the signal reaching it first, if this signal is sufficient to generate a spike, since it will put the neuron into the refractory period.

So-called temporal summation of local potentials also plays an important role. It takes time for a local potential to decay. So, if several local potentials from different

input points are generated in a neuron at different moments and none of those is by itself adequate to produce a spike, their sum may eventually exceed the excitation threshold dependent on their sufficient time synchronization at the soma. Another example of temporal summation is very frequent activation of a single input of a neuron. Frequent activation produces superimposed and hence accumulated local potentials until they exceed excitation threshold and generate a spike in the neuron. In fact, temporal summation and spatial summation are coupled together and cannot be used separately, since both space and time play a critical role in the arrival of local potentials at the soma.

It may seem that the neuronal networks presented here closely resemble artificial neural networks (ANNs). Although it is true and as a matter of fact NNN models are a particular type of ANNs, we try to give the former a clear distinction from the latter by ensuring that all basic neuronal laws observed in nature are encoded into NNNs (with some simplifications). The word “native” in the name emphasizes the fact that they originate from the real-life neuronal networks, opposed to ANNs which are a computational tool for solving a broad range of problems that have no connection with real-life neuronal networks.

Due to this distinction between NNNs and ANNs, we do not use conventional terminology of ANNs in the present work, such as “hidden nodes”, for example. Any node in NNNs can be an input or/and an output node and performs exactly the same computation as the rest of nodes (so all the nodes are “hidden”), that is why in the context of this work we found it confusing to use this terminology. Therefore, it is not used.

1.3 Definition of the research problem

1.3.1 Computation in neuronal networks

Although the basic laws presented above are not very complex, NNNs possess considerably higher inherent complexity than just a single neuron, since they operate in the scale of millions and even billions of neurons, if we consider the whole brain. A neuron can have thousands of synapses connecting it to other

neurons. Therefore, the main characteristic of NNNs is the great number of connections between neurons resulting in complex patterns of signal propagation within the network.

Currently it is computationally unfeasible to analyse networks of such size. A possible avenue towards a solution to this problem is to come up with suitable approaches that lead to the reduction of the amounts of neurons and synapses in the model networks without altering (at least significantly) their behaviour/functionality. We believe that such modelling may still yield a reliable approximation to the operation of real life networks. NNNs having hundreds of neurons and thousands of connections are analyzed within the scope of the present work, as well as in several other works mentioned in the following chapters.

Besides the question of direct simulation of signal propagation in NNNs, our main goal is to perform output-to-input signal reverse computation, also referred as prediction in the present work. In this context a network is presented as a “box” receiving input signals and generating output signals. Prediction is aimed to infer the input signal by knowing only the network structure and the generated output signal. The difference between the direct simulation and the prediction is highlighted below, since it directly corresponds to the main technique used in the present work.

1.3.2 Direct simulation versus prediction

The direct simulation of signal propagation in the network involves generation and propagation of the signals starting from input neurons in all possible directions, eventually reaching output neurons. Input neurons and output neurons are two predefined subsets of neurons in the network, the first is to be activated by given input sequences and the second is to be measured as it generates output sequences. The direct simulation can be carried out by Breadth First Search (BFS)-based technique [7] and is therefore linear in time. In contrast, prediction allows to know only the output signals, that are generated by the network as a result of unknown input signals. These unknown input signals should be “guessed” or retroactively predicted. Knowing the output of the network, the straightforward solution would be to traverse all possible input signal combinations and perform the

direct simulation of signal propagation for each combination. The resulting output signal in each simulation should be compared to the known output and the input combination reproducing the known output is the solution to the problem. However, this simple methodology is computationally unfeasible due to the exponential amount of input signal combinations, even in the scale of a single neuron.

1.4 Motivation for Considering Use of Genetic Algorithm and Related Background

Algorithms performing precise computations are unlikely to be able to solve the prediction problem in NNNs in feasible time due to the combinatorial explosion of possible input signal combinations. As a viable alternative, stochastic search algorithms are considered as a candidate group of algorithms, which are able to search for solutions in the exponential search space using randomization. Several families of stochastic search techniques are available, such as simulated annealing, stochastic hill climbing or evolutionary algorithms. We decided to use the latter, since there is evidence that evolutionary algorithms are less prone to getting trapped in local optima than the other two techniques [8].

At this point we have to relax the requirement of finding the exact input, which causes generation of the given known output, by turning the requirement from knowledge of the exact input that generates the output to approximating the input. Approximated input will generate approximate output, which should be as similar as possible to the known output, but is allowed to be different.

The present work explores the application of Genetic Algorithm (GA) to the defined prediction problem. GA is one of the best known evolutionary algorithms successfully applied in many fields [8]. Among the examples of the famous Computer Science problems, which GA was applied and yielded near-optimal results [9], [10], are the Knapsack problem [11] and Travelling Salesman problem [12], both known to be NP-hard.

GA is inspired by the natural evolution and inheritance processes and uses their principles. Random variation of solutions and the exchange of parts of viable

solutions are directed by the process of natural selection, which probabilistically selects better fit solutions for reproduction and survival, while worse solutions are discarded [13]. Reproduction requires selecting pairs of solutions from the pool of available solutions, when each pair of solutions (called parents) generates one or more new solutions (called children). The number of selected pairs is usually equal to half of the overall amount of solutions. The better fitness a solution has, the greater the probability of this solution to be selected as a parent becomes. Since after each reproduction event the number of solutions grows, selection for survival is performed after sufficient amount of children is produced (this number should be set according to each particular case) in order to maintain the same number of solutions. As mentioned earlier, worse fit solutions are discarded during the survival process.

The random variation of solutions is denoted as mutations. The exchange of solutions' parts is denoted as crossovers. Since both processes change the affected solutions in a way similar to the natural genetic inheritance, they are called genetic operators [14].

A GA starts with the initial set of randomly generated solutions for a given problem and tries to improve those until reaching a time limit or fulfilling some pre-set criteria. Improvement process is performed iteratively and is generally referred to as evolution. The entire set of participating solutions is called a population. The population of a specific iteration step may be referred as the generation of this iteration. Usually, population size is kept constant during the entire simulation. Passage between two successive generations is implemented using genetic operators (mutations, crossovers) and selection.

Mutation is a slight random modification of a solution rendering a different solution, which may be better or worse than the original one. Crossover is mutual exchange of randomly selected corresponding parts between a pair of solutions rendering one or more new solutions, which may be better or worse than the parent solutions. Finally, selection is the process that traverses all solutions and chooses probabilistically the best solutions to be transferred to the next generation. It should

be mentioned that usually crossovers impose some linear structure on the encoding of solutions in particular domains, because crossovers require corresponding parts of different solutions to be aligned against each other. However, more sophisticated encodings are also possible [9], [10].

The quality of solutions is measured by their fitness against some known value. According to the elitism principle [15], the best fit solution always survives in the population and is transferred to the next generations, until a better solution is generated. This way, in each iteration, the algorithm guarantees to produce a result that is at least as good as the result of the previous iteration. However, it does not guarantee to produce a better result. At the termination of the algorithm, the solution with the best fitness is returned as the algorithm's output.

1.5 GA encoding for the prediction problem in NNNs

In the context of the prediction of signals propagation in NNNs, a solution is the input signal combination supplied to a network, causing it to generate output signals over time. The encoding of input signal combinations into GA solutions consists of two sub-encodings: (i) which input neurons of the network are activated by this input combination and (ii) what their input signal sequences are. Both sub-encodings are represented as binary strings appended together to form a single linear encoding, which is presented in Fig. 1.4.

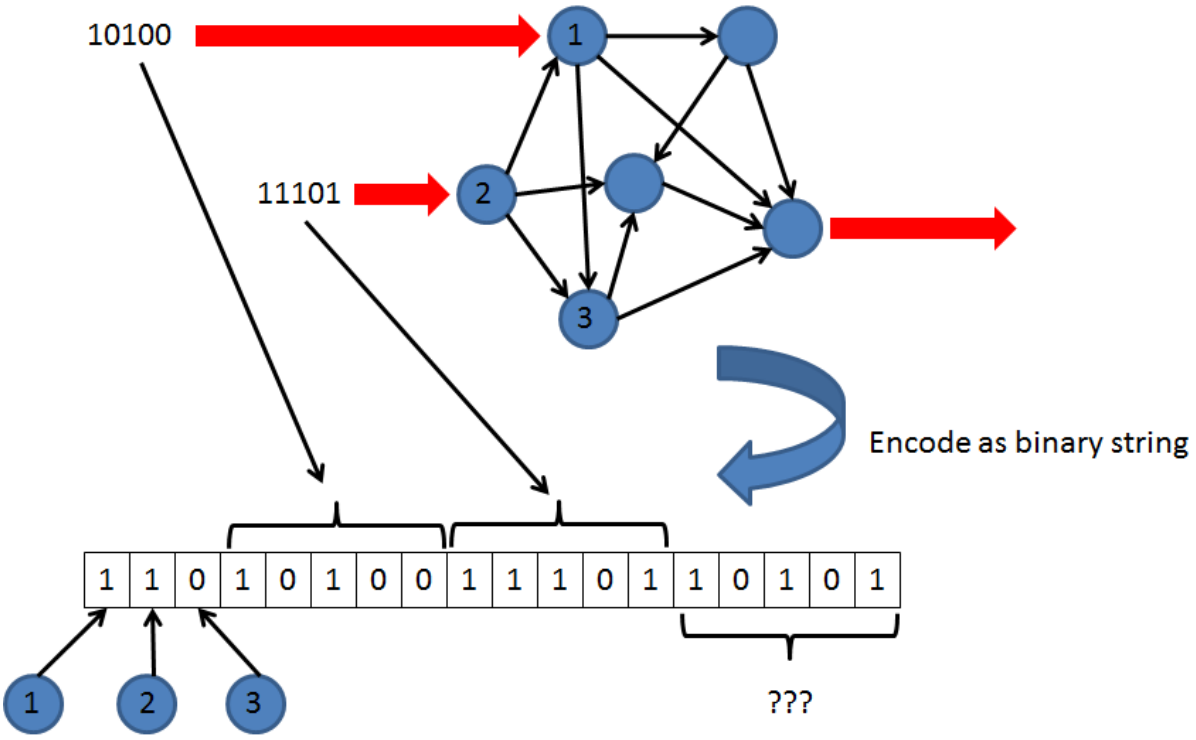


Figure 1.4 Example of input signal encoding for GA: input neurons are designated 1, 2, 3; 1 and 2 are activated by incoming input signals (two first bits 1 in the encoding sequence), 3 is not participating (third bit, equal 0). Output node is marked by the outgoing red arrow. The input sequence of neuron 3 is marked with "???", since the input sequence is not relevant for non-participating input node.

The fitness of a solution is defined by the similarity of the output signal, generated by the solution to the known output signal. Since output signals, as well as input signals, are represented by binary sequences, simple bitwise comparison is performed in order to calculate the similarity score between two output signals.

1.6 Thesis focus

The focus of this research is to develop a GA-based algorithm for finding approximate input signal sequences assigned to the specified input nodes of a given neural network, producing desired output sequences at the specified output nodes of the network. "Approximate" means that the computed input sequences should approximately produce the desired output sequences in the given network. The applicability and the efficiency of the algorithm will be tested on varying input conditions (different networks, different choice of input nodes, different input sequences).

The algorithms developed in the thesis include the framework for propagating signals in NNNs, built on the basic principles of signal propagation, and the GA-based algorithms using the framework to assess the fitness of solutions. The algorithms are implemented in the form of software program, which is used to run testing and evaluation simulations. The software is developed using the C++ programming language.

Therefore, the main steps involved in this work are:

1. Development of the framework for computing output sequences given input sequences in NNNs.
2. Development of several versions of the GA algorithm for finding approximate input sequences that produce desired output sequences, which make use of the framework defined above, but remain independent of the framework's implementation. This way the internal signal propagation laws defined in the framework may be adjusted or refined for different models without requiring any change in the prediction GA algorithms.
3. Implementation of the proposed framework and the GA algorithms in the form of a software program.
4. Testing of the developed algorithms' performance on two artificially generated networks, the first of which models the network located in the human brain and the second is the modification of the first with added feedback loops, with different topologies and varying parameters:
    - Different choice of input neurons
    - Different input sequences
5. Comparison of different versions of the GA algorithm to the URS, which samples random solutions by tossing a uniform random coin for each bit in a solution and returns the best solution found in the allocated time period.

1.7 Thesis contribution

In the present research we explore the effect of GA application on the neuronal network model that corresponds to the real NNN found in the human brain. The impact of the expected result is the possibility to continue the research

concentrating more on specific application domains (particular neurological diseases and known types of neurological injuries, which affect parts of human brain networks) and adjusting the simulation tool accordingly. In addition, we know that the technology is still not developed enough for experimental testing of the model proposed here, i.e., nanodevices are not widely used in the medicine yet. However, it is a fast developing area that has huge potential in terms of advancing medicine [16]. We explain in the following paragraph how the algorithm may be used in the future in the area of neuroscience applications.

The materialization of compensatory signalling in the damaged areas of neuronal networks in the brain is one of possible applications. If the healthy output signal of the network is known, with the proposed algorithm it would be possible to approximate the input signal that will reproduce the healthy output signal. Applying the approximated input signal to neurons in the damaged network will reproduce the approximate healthy output signal. The described process will require extraction of the network structure, identification of candidate input neurons and knowledge of output neurons and their target output sequences. Running the algorithm on the obtained network model and applying nanodevices, which should transmit the signals produced by the algorithm to the real network, are the key steps of the algorithm application.

1.8 Thesis outline

The thesis outline is as follows. Chapter 2 presents a review of the existing methods for simulation of signal propagation in NNNs and the existing GA applications in the domain. Precise definition of signal propagation model and its parameters, as well as the assumptions and simplifications applied, are presented in Chapter 3. Different versions of GA-based algorithms for solving the presented problem are given in Chapter 4. Description of two artificial networks, used for evaluation of the proposed algorithms, and the ways of generating the input sequences for approximation, are presented in Chapter 5. Numerical results of these case studies are shown and discussed in Chapter 6. Finally, concluding remarks and directions for future research are presented in Chapter 7.

Chapter 2:

Literature review

2.1 Fundamental works

The fundamental works in the field of simulating signal propagation in NNNs include the models of a single spiking neuron and neuronal populations' dynamics, proposed by Hodgkin and Huxley [17] and Wilson and Cowan [18], respectively.

Hodgkin and Huxley [17] concentrated on the low-level biophysical modelling of the electrical properties of neuronal membranes. The low-level processes, such as ionic currents through the neuronal cell membrane, were modelled as a function of electrical potential difference between the two sides of the membrane and the permeability coefficients of the membrane for different types of ions. Virtually, the electrical model of the neuronal membrane is represented as an electrical circuit with parallel currents for different types of ions and their corresponding resistances, which mock the permeability of the membrane for those ions (Fig. 2.1).

The derived nonlinear ordinary differential equations (ODEs) for the total membrane current as a function of time and voltage, presented below, allowed to approximate initiation and propagation of spikes in a squid giant axon:

$$I = C_M \cdot \frac{dE}{dt} + \bar{g}_K \cdot n^4 \cdot (E - E_K) + \bar{g}_{Na} \cdot m^3 \cdot h \cdot (E - E_{Na}) + \bar{g}_l \cdot (E - E_l) \qquad (1)$$

The four summands in the equation correspond to the membrane capacity current, the potassium current, the sodium current and the leakage current (chloride and other ions), in the order of their appearance in the equation. These four components are connected in parallel and therefore add up to yield the total current through the membrane. *E* is the difference in the membrane potential, which changes in time. C_M, $\bar{g}_K$, E_K, $\bar{g}_{Na}$, E_{Na}, $\bar{g}_l$, E_l are constants. The variables *n, m* and *h* change in time as well, the change is defined by three additional subsidiary ODE equations (not presented here).

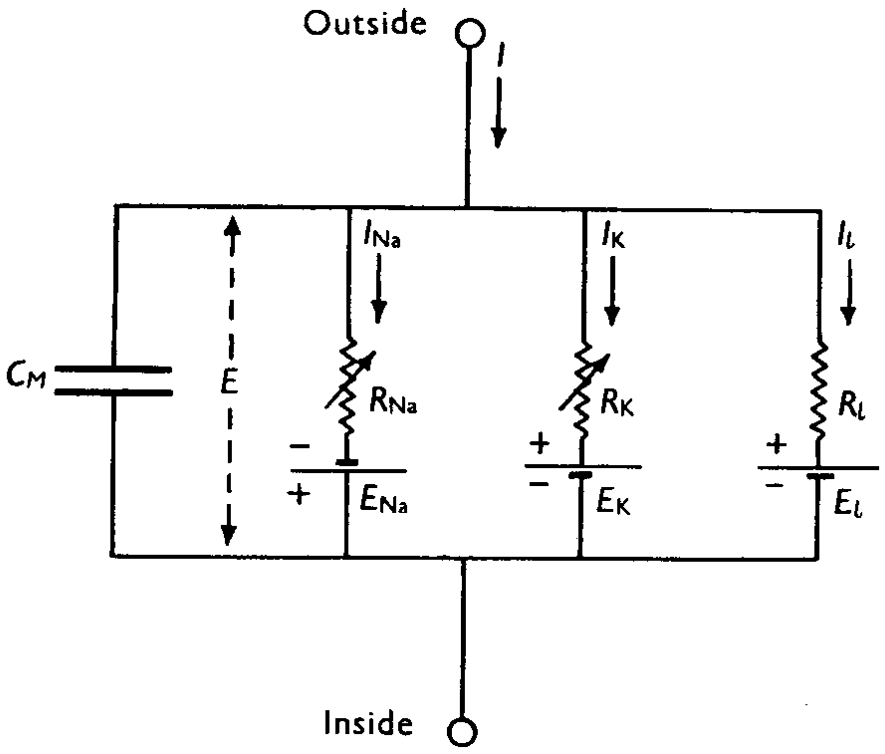


Figure 2.1 [16: Fig.1] Neuronal membrane as electrical circuit with 3 types of ions: sodium (Na), potassium (K) and leakage ions consisting of chloride and other ions (I). The corresponding ionic currents are designated INa , IK and Il , respectively. The resistances are designated RNa , RK and Rl , respectively. The equilibrium potentials for the ions are designated ENa , EK and El , respectively. E is the difference in the membrane potential between the outside and the inside and CM is the membrane capacity per unit area (a constant).

Although the level of complexity of equation (1) is feasible for computations in a scale of a single neuron, it does not allow for the computations in connected sets of neurons involving hundreds of neurons.

The necessary passage from a single neuron to neuronal populations was made by Wilson and Cowan [18], justified by the observation that higher neurological functions, such as sensory information processing and pattern recognition, are performed by involving large amounts of neurons simultaneously. Neuronal populations' spiking dynamics were modeled as the mean values of the underlying statistical processes. Stating that it is not possible to simulate spiking activity of neuron populations in the scale of single neurons due to the huge amounts of the

latter, it was suggested to treat the populations as connected sets of neuron aggregates with random dense connections, characterized by spatial localization and identical responses of the constituent neurons, as it is elaborated in the model presented below.

The model involves two neuronal subpopulations: excitatory neurons, designated *E*, and inhibitory neurons, designated *I*. The former contribute to the propagation of spikes in the population and the latter stop the propagation, respectively. For each of these subpopulations a non-linear differential equation, which expresses the proportions of neurons spiking at time *t*, is derived:

$$E(t) = \left[1 - \int_{t-r}^{t} E(t')dt'\right] \cdot S_e\left\{\int_{-\infty}^{t} \alpha(t-t') \cdot [c_1 E(t') - c_2 I(t') + P(t')]dt'\right\} \qquad (2)$$

$$I(t) = \left[1 - \int_{t-r}^{t} I(t')dt'\right] \cdot S_i\left\{\int_{-\infty}^{t} \alpha(t-t') \cdot [c_3 E(t') - c_4 I(t') + Q(t')]dt'\right\} \qquad (3)$$

E(t) and I(t) stand for the proportion of excitatory and inhibitory neurons spiking in the population at time *t*, respectively. Refractory period is denoted as *r*. The two main multipliers in each equation, written as (i) [...] and (ii) $S_{e/i}${...}, correspond to: (i) the proportion of cells, which are not in refractory period at time *t*, and (ii) expected proportion of the subpopulation receiving at least excitation threshold signal as a function of average levels of excitation within the subpopulation. Part (i) is derived as a complement of the proportion of cells, which are in refractory period at time *t.* Those are the neurons that were spiking between the moments *t-r* and *t.* Part (ii) consists mainly of the term defining the average levels of excitation within each subpopulation. *α* denotes the function of potential decay after a signal is received at the synapse. Connectivity coefficients c_1, c_2 represent the average number of excitatory and inhibitory synapses per neuron in the excitatory subpopulation, respectively, while c_3, c_4 represent the average number of excitatory and inhibitory synapses per cell in the inhibitory subpopulation, respectively. Finally, *P(t)* and *Q(t)* denote the external inputs to the excitatory and inhibitory subpopulations, respectively. These extremely complex equations, which involve temporal integrals, were further simplified by Wilson and Cowan in order to make them mathematically

interpretable, however, the model is not suited for application with any specific network structure.

Neither of the two approaches (Hodgkin and Huxley [17] or Wilson and Cowan [18]) is appropriate for the research question at hand, due to the computational unfeasibility of the former and the continuous modelling technique used in the latter. Discrete time simulations require different approach, which takes into consideration discrete signal transmissions between the neurons in the network according to its structure.

2.2 Structural analysis of NNNs

Knowledge of the NNNs' structure under investigation is crucial in order to be able to simulate signal propagation in those networks. The models of living neural networks or neuronal populations started to emerge recently, accompanied by attempts to uncover the structural and functional principles underlying those natural networks' organization [19], [20], [21]. This type of research became possible with the advance in neural spiking recording techniques and neuroimaging studies in humans. However, the large amount of neurons in the NNNs of the brain remains prohibitive for large-scale computations. There are several key simplifications described further, which make these networks possible to analyze.

Generally, there is a lack of evidence in the literature if brain networks' topologies have some common structure(s), which can be uniformly related to most of the brain. Since different parts of the brain evolved to perform particular tasks, it is unlikely to encounter the exact same topologies. Nevertheless, the parts of brain processing sensory information usually display well defined layered topology [22], [23], [24], [25], [26] which can be considered as a common motif structure for these compartments of the brain. We concentrate on this kind of topology in the present work, since we found it available in the literature. Presented below is the work, which provides a detailed model of such a network.

Modeling only a small part of the brain is one possible approach to deal with the natural complexity of NNNs. Based on this the primary auditory cortex was modeled by Chrostowski et al. [19] by using only 469 neuron nodes and about 10000 links

(synapses), the model is presented in Fig. 2.2. Structural properties of primary auditory cortex are translated into the topology of the designed network preserving essentials correlations among different kinds of neurons and their connections. The modeled network is much smaller than the real cortex network, but the simplification is essential in order to make it computationally analyzable. In addition, the work presents a way to model homeostatic plasticity of neuron connections – the ability of neurons to adjust the strength of their synapses in response to changed network activity passing through them. Homeostatic plasticity operates on a larger time scale (days) and is not covered in the present work.

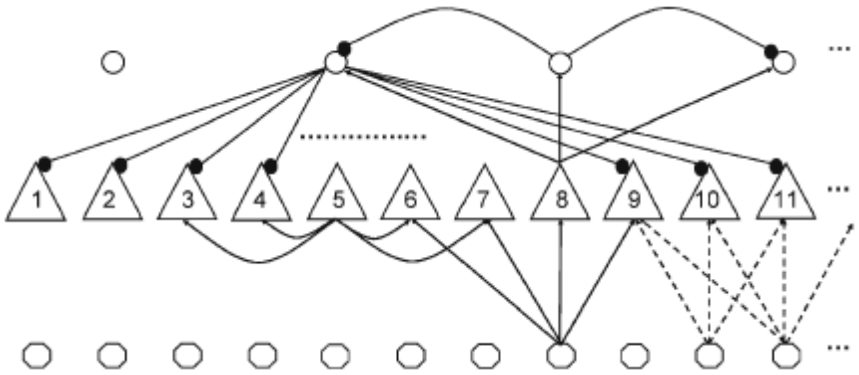


Figure 2.2 [18: Fig.1] The schematic layered structure of the primary auditory cortex. It consists of 3 layers of neurons: 201 thalamic neurons (regular octagons), 201 pyramidal neurons (triangles, numbered with their spatial coordinates) and 67 inhibitory interneurons (open circles). Arrowed lines are excitatory synapses, lines ending with full black circles denote inhibitory synapses.

The structure of auditory cortex model network introduced by Chrostowski et al. is employed in the present work as the network model corresponding to the existing real-life NNN, since it is elaborated with the sufficient level of detail in order to be used as an input to the software program.

Identifying sub-networks of neurons according to their particular function is another way of decreasing the amount of analysed sets of neurons. A way to achieve this is presented by Berger et al. [20], when the spiking activities of a large number of neurons are recorded simultaneously. In order to distinguish specific neuron

assemblies that are spiking correlatively within the massively recorded spiking set of neurons, the authors introduce two test statistics representing the patterns of relative spike complexities and spike frequencies of the neurons. These statistics are the main criteria according to which the neurons are identified as having a correlation in their spiking activity. Reducing data set of neurons to the relevant ones (neurons that tend to spike together, in this case) helps in lowering computational demand for further analysis of such assemblies.

Another aspect, hierarchical modular organization of human neural networks, is illustrated by the approach of Meunier et al. [21]. Generally, partition of a given graph (representing a network) into modules is a separation of the graph's set of vertices into subsets, which are referred to as modules. The approach is based on finding the best partition of a network into modules, where "the best" means the partition that gives maximum modularity (the term defined by Newman and Girvan [27]). Modularity evaluates how well a given partition concentrates the edges within the modules according to the following formula:

$$Q = \frac{1}{2m} \sum_{C \in P} \sum_{i,j \in C} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \qquad (4)$$

The number of edges in the graph is designated by m. $P = \{C_l\}_l$ is a partition, or a set of disjoint subsets C_l of the graph's vertices (C_l are also called modules), when the union of all modules C_l renders the whole set of the graph's vertices. i and j are indices traversing all the possible pairs of nodes within the modules. A_{ij} is the number of edges between nodes i and j (which can equal either 0 or 1). k_i and k_j are the number of edges touching nodes i and j, respectively. Therefore, formula (4) compares the number of edges between all pairs of nodes, which belong to the same module, to the expected number of such edges in an equivalent random graph. The former number is expressed by A_{ij} and the latter is expressed by the term $\frac{k_i k_j}{2m}$. In equivalent random graph all nodes have the same number of edges touching them (in this case k_i and k_j), but the connections between them are picked randomly. So, the possibility that any edge outgoing from node i will end up in the

node *j* equals $\frac{k_j}{m}$ (divided by *m*, because it is randomly picked). The overall number of expected edges between them should be then multiplied by k_i. However, this way each edge is counted twice, therefore, the result should be divided by 2, yielding $\frac{k_i k_j}{2m}$.

Here is a simple example of two different partitions P_1 and P_2 of the same graph, having different modularities (see Fig.2.3).

$P_1 = \{C_1, C_2\}$ $P_2 = \{C_3, C_4\}$

$$Q_1 = \frac{1}{2 \cdot 1}\left(0 - \frac{0 \cdot 1}{2 \cdot 1}\right) = 0 \qquad Q_2 = \frac{1}{2 \cdot 1}\left(1 - \frac{1 \cdot 1}{2 \cdot 1}\right) = \frac{1}{4}$$

Figure 2.3 The same graph with two different partitions, followed by the calculation of their modularities. Three vertices of the graph are represented by yellow full circles, a single edge is represented by a line, modules of the partitions are depicted by transparent ovals.

The first partition leaves the edge “outside” the modules, in contrast to the second partition, which contains the edge within module C_3. That is why the modularity for the second partition is higher.

Modules may be recursively decomposed into sub-modules providing hierarchical view of the network. Sub-modules may be presented as meta-nodes in meta-network of reduced complexity compared to the original one. The presented method was able to analyze the brain functional network with 1808 nodes and 8000 links, representing the functional magnetic resonance imaging [28] snapshot of the whole human brain.

It is important to mention, that the approach presented by Meunier et al. for shrinking given networks into the networks of higher organization levels and, therefore, smaller size, can be used for generating input networks for the algorithms developed in the present work. However, if such adjustment is performed, the basic signal propagation laws may be affected, since groups of neurons clustered together shall have behaviour different from the behaviour of single neurons. Let us consider an example of how basic neuronal laws may be affected by clustering two neurons, which are connected in the way shown in Fig 2.4.

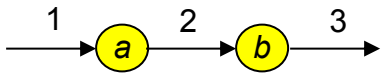


Figure 2.4 A cluster (or module) of two sequentially connected neurons a and b. Edge designated 1 represents a synapse incoming from outside into the module, edge 2 represents an internal synapse between the neurons and edge 3 represents a synapse connecting the module to some outside neuron.

Let us assume that synapses 1 and 3 are strong enough to cause a spike in a neuron, even if only one signal is transmitted through them. Synapse 2, however, is extremely weak and it requires a series of repetitive spikes to be passed from neuron *a* to neuron *b* in order to cause a spike in neuron *b* (according to the principle of temporal summation). In this situation, the module demonstrates behaviour, which is impossible to reproduce in a single neuron. In a single neuron it is always possible to cause a spike by having just one incoming signal, if the signal is strong enough. Here, in contrast, no matter how strong is the signal supplied through synapse 1, synapse 2 will not allow a single spike to be transmitted immediately to neuron *b*. Only a frequent series of spikes supplied through synapse 1, which will be reflected in a frequent series of spikes through synapse 2, can cause accumulation of local potentials in neuron *b* and, finally, cause a spike.

All the techniques described above demonstrate that it is possible to analyze existing native neural networks by obtaining neuroimaging / spike recordings and then reducing the size of computational models developed for the recordings. The obtained structures are to be analysed in the context of the basic neuronal laws encoded in the signal propagation framework. We should keep in mind that the framework is a tool, which is used by the GA algorithms developed in the present work, and is not the main purpose of the present work.

The existing approaches of utilizing signal propagation in NNNs are elaborated in the next section.

2.3 Utilization of signal propagation in NNNs

The possibility of using living neural networks as the mean of molecular communication was introduced by Balasumbramaniam et al. [29]. The authors focused on 2 important aspects of molecular communication passing through neuronal networks: (i) the design of interface between nanodevices and the neurons that can initiate signalling, and (ii) the design of transmission scheduling to ensure that signal initiated by multiple devices will successfully reach the receiver with minimum interference. Interference is the effect of signal propagation interruption in NNNs, caused by the neurons, which enter refractory period after spiking, and therefore are unable to transmit other incoming signals. This way a signal, which comes earlier, can interfere with the propagation of another signal, which comes later, by putting neurons in the propagation path of the second signal into refractory period. Part (i) of the work is done through wet lab experiments and part (ii) is developed using GA.

The described possibility to use nanomachines to activate specific neurons in a given network is important for the future application of the present work in real life networks. The idea of using GA search heuristic for better signal initiation is employed in the present work as well. However, the objective of the present work is different from the objective of the work by Balasumbramaniam et al. In the latter the authors try to maximize the number of input signals, which are able to reach the output neurons without being interrupted by the interference effects, and to minimize

the time difference between the spikes of initiating devices. The present work focuses on the maximization of the resemblance between the given known output signal sequences and the output sequences generated in the network by the spikes of initiating devices. Therefore, the core difference is in the fitness function of GA: Balasumbramaniam et al. maximize the amount of signals reaching output points of a network, while in the present work we maximize the resemblance between the given signals and the generated signals reaching output points of a network.

Communication between biological nanodevices using backbone neuronal network was proposed by Walsh et al. [30]. The paper explores how neuronal cell characteristics affect the performance of a network and proposes four different topologies for the backbone network (see Fig.2.5 and Fig. 2.6). Although the authors present the idea of the artificial creation of the networks serving communication purposes and they are not analysing naturally occurring networks, the work is still relevant in the context of the present work.

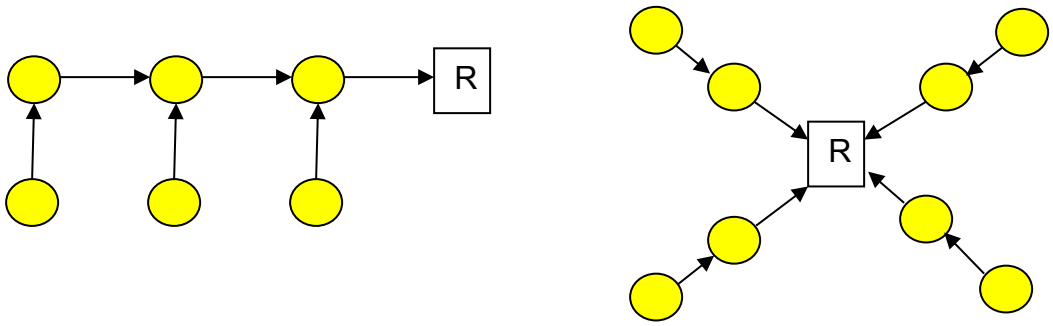


Figure 2.5 Bus and star backbone network topologies (from left to right). R designates a receiver nanodevice, which collects the incoming signals.

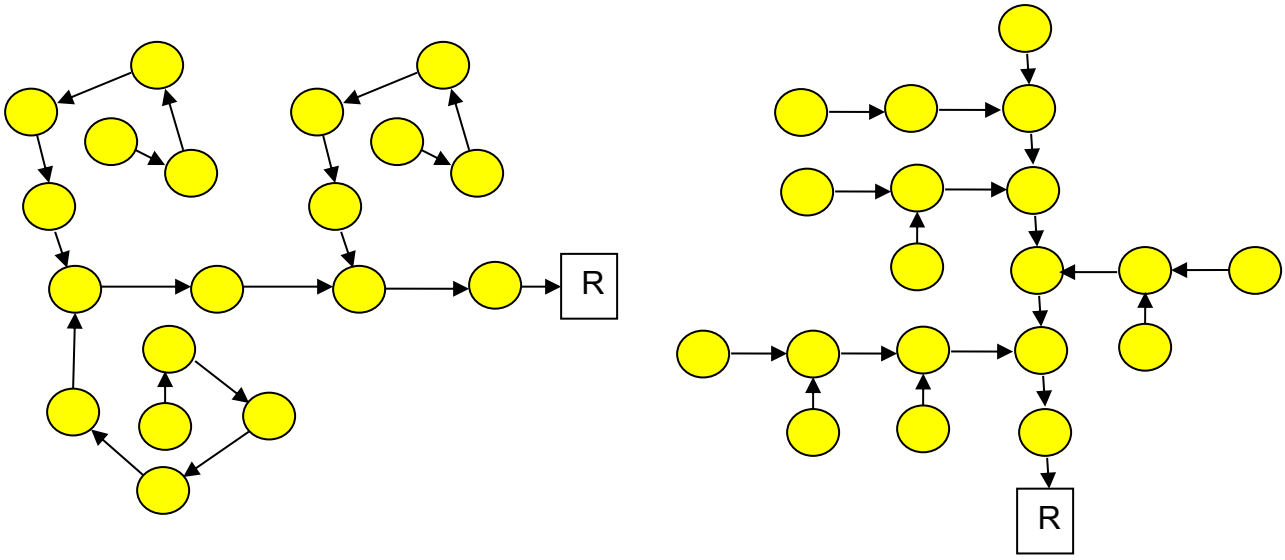


Figure 2.6 Spiral shape and tree backbone network topologies (from left to right). R designates a receiver nanodevice, which collects the incoming signals.

The topologies proposed by the authors can be easily plugged into the signal propagation framework and analysed by the prediction algorithms, developed in the present work. The authors define development of communication network consisting of neurons as the main objective of the work, as well as the influence of different topologies on the performance of the network (i.e., the amount of information that can be transmitted to the receiver per second). However, if there is an uncertainty in the information received at the receiver, meaning that its decoding is ambiguous, input sequences provided to the network should be reproduced as precisely as possible. In this case the prediction technique, developed in the present work, might be used, fed with the network topology and the output sequence recorded at the receiver.

2.4 Application of GA techniques in NNN domain

Most of GA applications operate on Artificial Neural Networks (ANNs), which will be considered in this work as the type of networks most similar to NNNs, and fall into one of the following main groups (each group is exemplified by chosen works):

1. GAs for training the weights on the edges of ANNs with given topologies. One of the influential works in the area presented by Montana and Davis [31] explores the performance advantages of using GA technique in order to obtain the near-optimum set of weights for the given networks instead of using the traditional backpropagation methodology [32]. The latter technique is not suited for the problems with high complexity (due to increased dimensionality of the problem or greater complexity of the training data), which can cause it to get trapped in the local optima. In contrast, a GA avoids getting trapped in local optima by intensive use of mutations and crossovers.
2. GAs used to evolve the ANN properties, in addition to evolving weights on the edges. Tulai and Oppacher [33] introduced competitive-cooperative co-evolutionary GA for evolving both edge weights and neuronal properties, such as characteristics of neuronal activation, of cascade neural networks (CNNs) used to solve the so-called two-spiral problem [34], which we do not present here due to this problem's irrelevance in the context of the current discussion. This advanced GA technique involves cooperation and competition between multiple populations of solutions evolving in parallel, the processes by which the survival of solutions is determined. Each solution in this case is a CNN solving the two-spiral problem. Importantly, by evolving neuronal properties the authors were able to develop more compact CNN architectures for solving the problem, than the architectures developed by evolving the weights on the edges only.
3. GAs constructing new ANN topologies for the problem at hand. Dasgupta and McGregor introduced the structured GA for the construction of application-specific ANNs [35]. Structured GA is a GA, which treats solution encodings as structured strings having several parts, connected by some context-dependent relations, and evolves the encodings by using different mutation and crossover rates for different parts [36]. In this work automatic generation of network structures and their weights was performed without using any learning techniques or problem domain assumptions.

As far as we are aware, there are only a few works focusing on the GA application for evolving input signals in NNNs/ANNs, one of which was discussed in the work of Balasumbramaniam et al [29]. Another interesting work was presented by Guo and Uhrig [37], in which ANNs were trained for fault diagnosis of nuclear power plants. GA technique was used in that study to guide the search for optimal combination of inputs for the neural networks to reach the criteria of fewer inputs and faster training.

Overall, most of the related works in the area concentrate on the achieving evolution of ANN structures by the means of GA. In contrast, the current work explores a new way of applying GA technology by evolving input signals for a better approximation of the output.

The next chapter presents the model of signal propagation in NNNs, which comprises the framework used by the GA-based algorithms, developed in the present work.

Chapter 3:

Model of signal propagation in NNNs

The model presented in this chapter is a simplified interpretation of the basic neuronal laws and does not reflect all the complexity of NNNs. However, we believe it still possesses the sufficient level of detailing in order to test the applicability of GA to the problem at hand.

3.1 Neuronal network model

A neuronal network is modelled as a directed weighted graph consisting of nodes, which represent neurons, and directed edges between the nodes, which represent synapses. The weights on the edges represent synaptic strengths. Each node stores the information about its ongoing state: spiking, in refractory period or resting (which means ready to receive local potentials from the neighbour nodes). Each edge stores a list of the local potentials initialized at the corresponding synapse. As it was described earlier, local potentials propagate from synapses towards the soma, reach the soma, pass it and decay in axons until their complete termination (if no spike is generated). A local potential can be visualized as a wave observed from a particular point in space, the soma in our case, which gradually increases the potential of the membrane at the soma, brings it to a maximal value and then gradually decreases the potential back to the resting state [37: 7-43] (see Fig. 3.1). Therefore, each local potential stored in the list has its ongoing value as it is measured at the soma at each moment in time (determination of the numerical values of local potentials is elaborated in section 3.2).

As the simulation proceeds in time through a series of time steps, the values of a local potential change from one step to the next, starting from zero, reaching a plateau and returning back to zero, once the local potential terminates. A local potential is stored in the local potentials' list of the edge until termination. After the termination local potentials are deleted from the list of their edge.

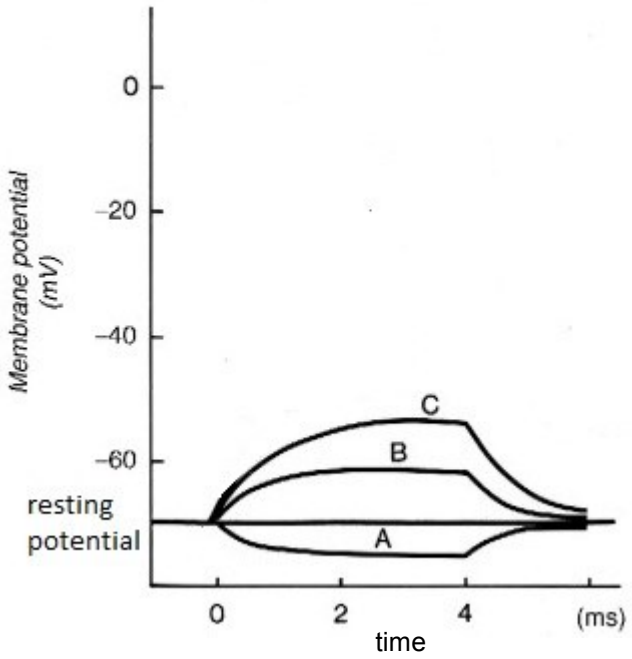


Figure 3.1 Local potential example curves (A, B and C), as they are measured at the soma separately. The local potentials are initialized at synapses at time = 0. A is initialized at an inhibitory synapse hence increasing the potential difference of the membrane, while B and C are initialized at excitatory synapses hence decreasing the potential difference of the membrane.

Nodes of the graph, which are defined as the input nodes, are fed with pre-specified incoming external binary sequences consisting of {0,1} bits for the entire runtime of the simulation, where 0 stands for not generating an input spike and 1 stands for generating an input spike. These sequences are the input signals which initialize the signal propagation through the network. The rest of the nodes are assumed not to perform any activity unless they are activated by the incoming edges.

Nodes of the graph, which are defined as the output nodes, are aimed to store their spiking activity at each simulation step or a time unit, as it is discussed later. The binary sequences generated this way are referred as output sequences. It is possible to define a node to function as both input and output node.

It is not always essential to use all the graph nodes and edges in order to reach output nodes. In fact, we are interested only in the neurons, which participate in the

generation of network's output signals. In this context we define a neuron as *affected* by an input neuron, if the former is located on a directed path starting at the input neuron. In addition, we define a neuron as *affecting* an output neuron, if the latter is located on a directed path starting at the former. These two definitions are aimed to formally refer to the neurons, which are potentially able to transmit signals from input to output nodes due to their location on the paths leading from input to output nodes. The objective of the following algorithm is to reduce the size of the initially given directed graph *G*, representing the whole network, to a more compact graph *R* comprised only of neurons, which are affected by input neurons and simultaneously affect output neurons. All the rest of neurons are dropped.

G can be reduced to *R* as follows:

1. Find all nodes that are accessible from the input nodes. That means, for each of those nodes there should be at least one directed path from one input node. It is done by running the BFS algorithm [7] from each input node and then performing union of the resulting node sets. Let's designate this group as *A*.
2. Find all nodes that have directed path to at least one of the output nodes. It is done by running BFS from each output node, when edge directions are reverted, and performing union of the resulting node sets. Let's designate this group as *B*.
3. Perform the intersection of *A* and *B*, let's designate it as *C*, and define *R* as the induced graph of *G* on the node set *C*, namely, a graph containing the intersection of *C* with the node set of *G* and all the edges connecting them in *G*. This is the desired reduced network structure. Notice that *C* should include at least one input node and one output node, otherwise the solution consists of 0 bit sequences for all output nodes. Any output node which is not in *C* will automatically get 0 bit output sequence as well.

The process is exemplified in Fig. 3.2.

Set *A* represents the nodes that can be affected by the input nodes of the network. Set *B* represents the nodes that can affect output nodes. If a node is not in the

intersection of *A* and *B*, then it either cannot be affected by input nodes or cannot affect output nodes. In any case, the node is irrelevant for the current problem of finding output sequences generated by given input sequences. That is why it can be eliminated from the graph with all its incoming and outgoing edges, which are not included in the induced graph *R*.

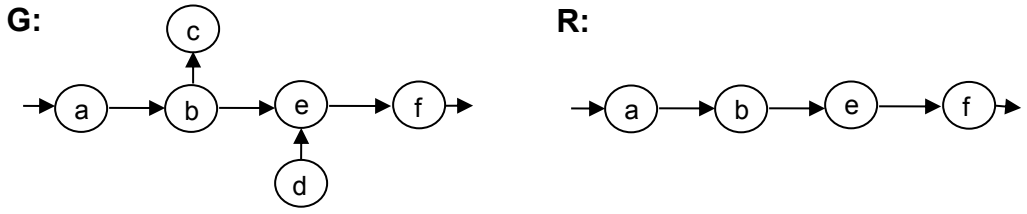


Figure 3.2 Reducing original graph G to induced graph R of participating neurons only. G includes nodes a,b,c,d,e and f. Node a is the only input node, node f is the only output node. Set A is formed by running BFS starting from node a, taking union over the following reachable nodes sets: {a}, {b}, {c,e}, {f}. Hence A = {a,b,c,e,f}. Set B is formed by running BFS in reverse starting from node f, taking union over the reachable node sets: {f}, {e}, {b,d}, {a}. Hence B = {f,e,b,d,a}. The intersection of A and B yields C = {a,b,e,f}. R is the induced graph of G on C, therefore, nodes c and d, as well as edges (b,c) and (d,e), are dropped. Notice that node c cannot affect output node f, while node d cannot be affected by input node a.

3.2 Model parameters

The important model parameters that should be set are the following: refractory period, spike propagation time (through a single neuron), local potential values, excitation threshold and synaptic weights. These parameters are divided into 2 groups: (i) parameters hard-coded within the model and (ii) parameters provided by the user. The first group includes refractory period and spike propagation time. The second group includes excitation threshold, local potential values and synaptic weights. The values assigned to the parameters of both groups are discussed below.

According to Betts et al. [38], neurons have spike velocity of roughly 10 m/s (meters per second) and refractory period between 1ms (millisecond) and 2ms. In order to calculate the time of spike propagation through a single neuron, we need to know

the average length of a neuron. Human brain neurons are in the length range between 0.0015m (meter) and 0.015m [39], assuming that the length of a neuron is measured as the summation value of the longest dendrite, the soma and the longest axon. This results in spike propagation time through a single neuron falling in the range between 0.15ms and 1.5ms. We set the spike propagation time to be equal to 1ms for all the neurons in a network. This simplification is required, since unequal spike propagation times for different neurons would lead to asynchronous spikes in the network, making it impossible to know which neurons are going to spike in the next simulation step without traversing and checking all of them. The implications of such a complication are explained below.

The search for the neurons, which are going to spike at each step of the simulation, is the crucial part of the simulation algorithm. In the current version of the algorithm we are able to perform this search in O(1) operations ("the order of 1" operations, which means the number of operations is constant and known) due to the global synchronization of spikes in the network (see the details on the synchronized algorithm further in the chapter). However, if we substitute this synchronization with asynchronous spikes, instead of O(1) operations the search will take O(*n*) operations ("the order of *n* operations", which means that the number of operations is a multiple of *n* by a known constant), where *n* is the total number of neurons in the network. Roughly, this complexity increase can cause the overall simulation run time to be multiplied by the number of neurons in the tested networks, which is about 10,000. Therefore, according to Table 6.1, which presents the overall simulation times for different test sets in the present work, the minimal among the simulation times will come up to 4,322,416,000 seconds (on a single 2.4GHz Intel processor), which is approximately 137 years. Each test set contains 100 independent tests, which are therefore easily parallelizable. However, each single test will still require more than a year to run. The search has to be performed at each step of the simulation and the steps are not parallelizable due to the dependency on the results of the previous steps.

As a result, due to the unfeasible run time following from setting unequal spike propagation times for different neurons, we found it necessary to make the

simplification of equal spike propagation times for all neurons. This simplification impacts the reliability of the results significantly for the networks consisting of differently sized neurons. However, the results for the networks consisting of neurons having the same size are expected to be reliable.

The excitation threshold is the electric potential of the membrane, which causes spike initialization, and it is measured in mV (millivolts). The threshold falls in a range between -70mV and -50mV for different types of neurons, when for the majority of the neuron types it is equal to -59mV [40]. We assign the same threshold value to all the neurons in simulated networks. The value is provided by the user as a parameter to the framework. It has to be in the valid range between -70mV and -50mV. In most cases we expect the user to set the value of the threshold to -59mV, unless he/she tries to simulate particular neuronal populations with different known excitation thresholds. However, this restriction of setting the same excitation threshold for all neurons can be easily dropped if needed, since the framework software design supports different excitation thresholds for different neurons.

Local potentials change their value from the initialization up to the termination. The value of a local potential is the function of voltage against time, which has passed since the initialization of the potential. We do not know the precise values of the function through the simulation time steps. That is why we rely on a user to provide local potential values: they are provided as a list of positive electric potentials from the initialization up to the termination. There are various sources of these values, both experimental and modelled [41]. It should be mentioned that different types of neurons may have different curves reflecting their local potential dynamics, therefore, the available experimental or computational data employed by the user should fit the type of investigated neuronal networks.

A local potential value is added to the resting potential of a neuron, which is the electric potential of the neuron when in the resting state. The value of the resting potential set in the framework is -70mV [40], however, it may be adjusted by the user. The connection between the resting potential, local potentials, the excitation threshold and the spike is summarized in Figure 3.3.

In addition, local potentials have different values at different synapses. The way to introduce this difference in the values is by using synaptic weights. Synaptic weights are the coefficients that multiply local potential values in order to adjust them to each particular synapse. Synaptic weights are implemented as the edge weights in the directed weighted graph representing a neuronal network structure. They are provided by a user together with nodes and edges of the graph. The values of synaptic weights may vary, just as the synaptic strengths of neurons found in the real-life neuronal networks. These values can be acquired from different sources of experimental data, when they become available for different types of neuronal networks, e.g., the data for the auditory thalamocortical systems, used in the present work [42]. This requires the user to be familiar with the ongoing experimental research in the field of neurological data, since no unified source of synaptic data is known.

Local potential values, which are provided by the user, are scaled automatically to the value corresponding to each particular synapse by using its synaptic weight. A synaptic weight with value 1 means that all local potentials initialized at the synapse will have the values equal to the ones provided by a user. Value 0.5 means that all local potentials initialized at that synapse will have the values equal to the half of the ones provided by a user, making it harder for the neuron to reach the excitation threshold. Value 0 means that the synapse is not active at all. Negative values of synaptic weight mimic inhibitory effect: they increase the gap between the ongoing electric potential of a neuron and the excitation threshold by adding negative local potential values to the resting potential. Each edge in the graph multiplies the sum of local potentials propagating through it by the edge's weight.

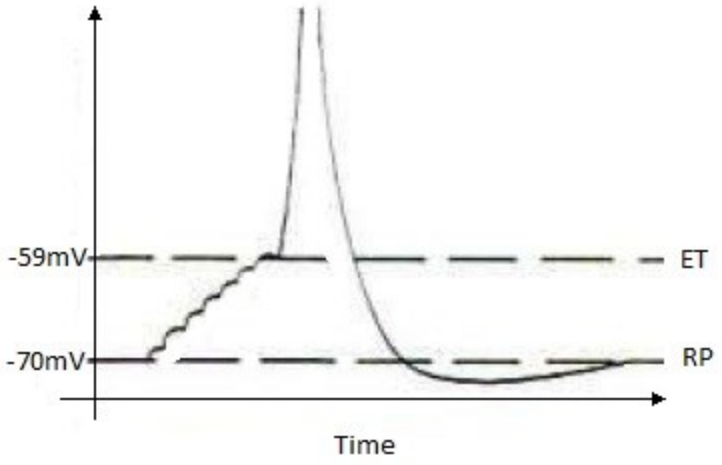


Figure 3.3 Membrane potential against time during spike generation at the soma. ET denotes excitation threshold, RP denotes resting potential. The stair-like potential shape at the beginning shows the accumulation of local potentials in the soma. Once the membrane potential reaches the excitation threshold, the spike is generated. The part of the curve below RP shows the potential during refractory period.

3.3 Simulation algorithm

Given the topology of a particular neuronal network, input neurons in the network, which are assigned particular sequences of binary signals (spike / no spike) and output neurons in the network, the binary signal sequences generated by the output neurons are recorded. Initialization of signal propagation is performed at the input neurons. Afterwards, signals propagate in the network taking all possible paths, defined by the synapses, in all possible directions. Signals reach the output neurons at different times and may come from different neighbours, but once spikes are generated by spatial and temporal summation at the output neurons, they are recorded in the output signal sequences as a bit with value 1. At each simulation step, when a spike is not generated in an output node, a bit with value 0 is recorded in the output signal sequence of the node.

The simulation proceeds in discrete time steps. The time period between two consecutive time steps is called time unit and it is assumed to be constant. The user supplies the number of time steps for the simulation to run, also referred to as the duration of the simulation. Both input and output sequences have exactly the same length in bits, which is equal to the duration of the simulation.

At each time step the simulation performs two traversals of all the nodes: (i) nodes, which spiked at the previous time step, generate new local potentials in all their outgoing edges and enter refractory period for one time unit; (ii) resting nodes perform summation of all local potentials stored at their incoming edges and spike if the sum reaches the excitation threshold (see Figs. 3.4 and 3.5). The reason for setting the refractory period to be equal to one time unit is in order to prevent asynchronous neuronal firings, since – as it was explained above - this would dramatically increase the run time of the simulation; approximately 10000-fold.. See elaboration on this issue in section 3.4.

There are two important points that should be mentioned. First, the input nodes have two potential triggers for generating spikes: (i) external input signal sequences; (ii) synapses incoming from their neighbour neurons in the network. The input nodes spike anyway if their external input sequences indicate bit “1” for the current time step, independent from the result of the summation of signals at the incoming synapses. Second, once a node spikes, all local potentials at its incoming edges are erased, since a spike causes complete re-initialization of the membrane’s electrical state.

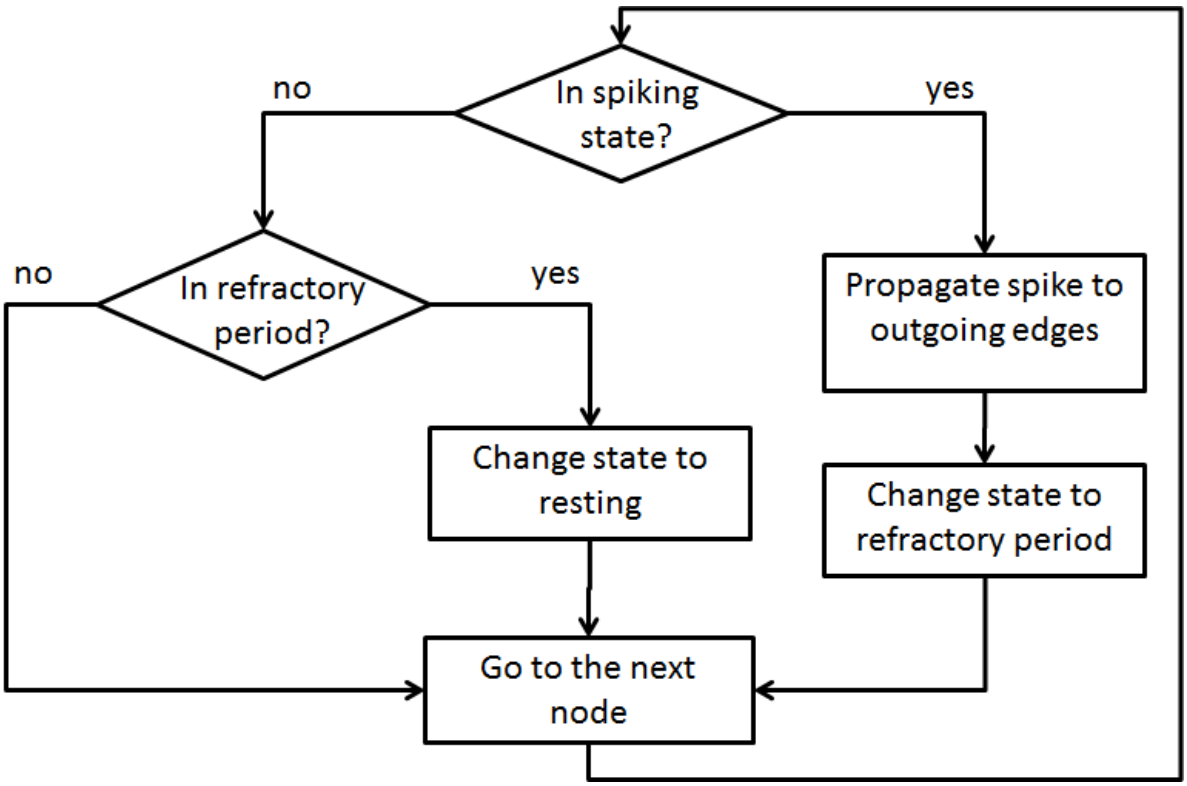


Figure 3.4 Signal propagation algorithm flow: 1st traversal of the nodes.

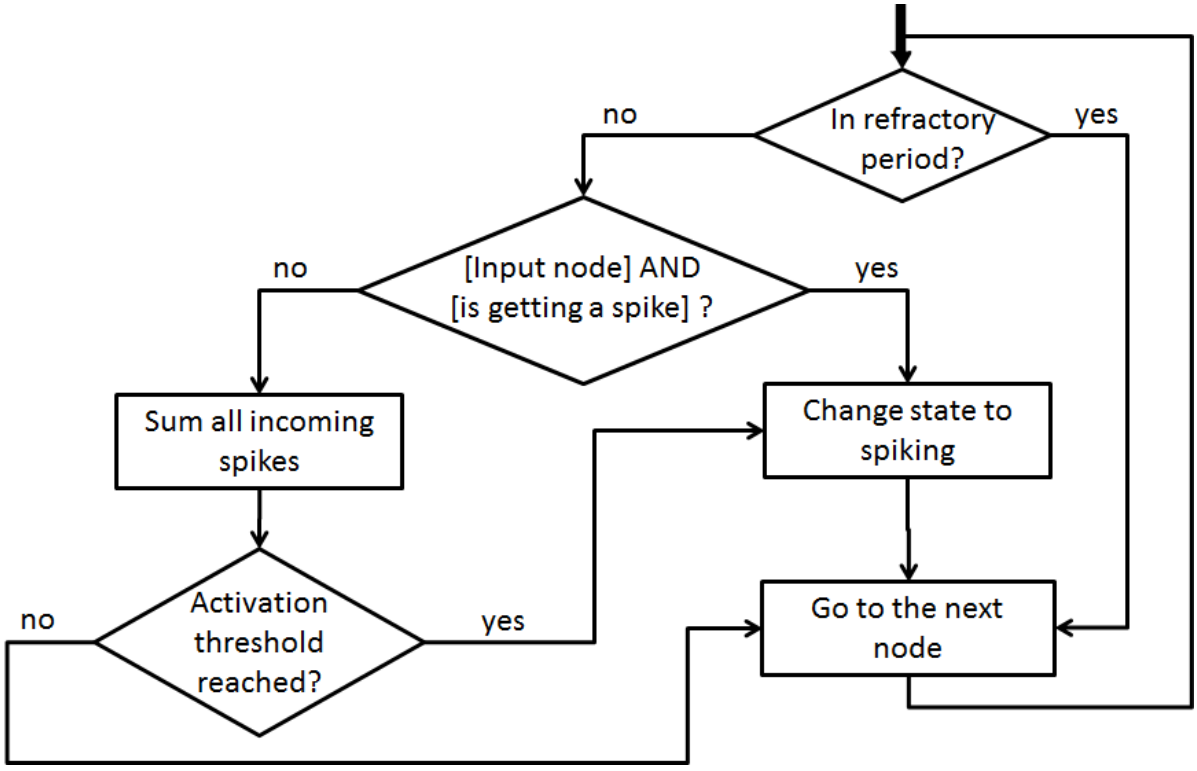


Figure 3.5 Signal propagation algorithm flow: 2nd traversal of the nodes.

During each one of the two above traversals the nodes may be iterated in an arbitrary order, since the propagation of signals in the edges (the 1st traversal) is decoupled in time from the initiation of spikes (the 2nd traversal). Virtually, the 1st traversal treats signal propagation in the edges (synapses), while the 2nd traversal treats signal propagation and spikes in the nodes (neurons). This order of the two traversals (first checking the edges, then the nodes) is justified by the fact that an edge represents not only the synapse, but also the dendrite where it passes the signal to, while a node represents the soma and all the axons of a neuron (see Fig.3.6). Since the propagation of signals occurs from dendrites to soma and axons, edges should be traversed first.

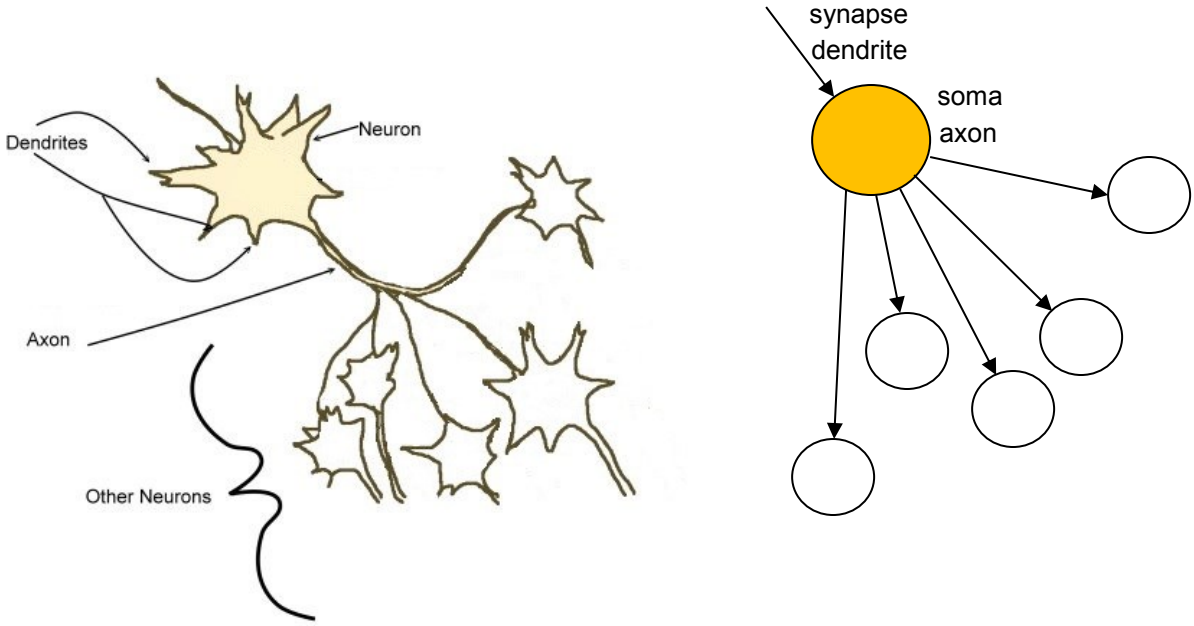


Figure 3.6 Visualization of artificial elements in modelled NNNs. Edges (on the right) represent synapses and the dendrites they are connected to (on the left), node (on the right) represents a soma and its axon (on the left). Branching structures at the end of the axon are considered as the pre-synaptic parts and are therefore included in the edges.

3.4 Simplifications and assumptions of the model

The proposed simulation algorithm makes two simplifications: (i) the time that it takes for a spike to propagate through a neuron is constant for all neurons and equals one time unit; (ii) the refractory period is also equal to one time unit, and therefore, is equal to the time of the spike propagation. Though the refractory period

is constant in real life neuronal networks, it is not necessarily equal to the time of the spike propagation. The spike propagation time differs among neurons, due to their different shapes and lengths. However, it is possible to apply the first simplification, since we are interested in the populations of similar neurons having similar shape and length. Setting the refractory period equal to the time of spike propagation is possible, since their ranges overlap. Although this is not always the case, we find it necessary to make this (second) assumption for the same reason we set the spike propagation times to be equal for all neurons: the global synchronization of neuronal spikes during the simulation. Both spike propagation time and refractory period affect the computation of the time when a neuron spikes, enters refractory period and returns to the resting state. If these two parameters are even slightly different for different neurons in the network, finding the next spiking neuron at each step becomes much more computationally intensive task. As it was mentioned earlier in the chapter, the search for the next spiking neuron would involve traversing all neurons in the network and checking which neuron is going to spike / enter refractory period / return to the resting state earlier than the rest. Since we deal with the networks having approximately 10,000 neurons, the lack of this simplification would lead to the unfeasibility of the simulation.

The fact that we make these necessary assumptions puts in doubt the reliability of the developed model. The main question that arises is if there are any NNNs in the human brain that have or approximately have this property of synchronization of neuron spikes. As the research shows [43][44][45], cortical networks of human brain should be able to sustain synchronous neuronal spiking. However, this is still a computationally analysed hypothesis, which is yet to be supported by real physiological data.

3.5 Complexity of the computation

Let us designate the graph representing a network of interest by $G = (V, E)$, where V is the set of nodes and E is the set of edges. In addition, let the maximal number of local potential values stored at any edge be designated as L. Then the complexity of a single simulation time step is $O(L * (|V|+|E|))$ operations, where $|X|$ notation stands

for the size of set X (absolute value) and the function O(Y) is the order of the number of operations defined by Y, which equals to a multiplication of $|Y|$ by some known constant. Operation in this context may be a retrieval of a single local potential from the edge's list of potentials and summing it with the previously calculated result, as well as traversing from one local potential to another, from one edge to another, from one node to another. Each of these operations requires constant and known number of steps/calculations and therefore contributes O(*1*) to the overall complexity. There are several important points about the single step complexity that are elaborated below.

Traversal of the graph requires traversal of the nodes and the edges, while treating each node potentially requires the summation of all the local potentials in the node's incoming edges. There is no outgoing edge that is traversed more than once in the 1st traversal, since each edge is directed from exactly one node, and each node is traversed only once. There is no incoming edge that is traversed more than once in the 2nd traversal, since each edge is directed into exactly one node, and each node is traversed only once. L is limited by the amount of local potential values supplied by the user, which generally should not exceed the 15, if we consider the time unit of 1ms and physical properties of neuronal cells, according to which a local potential can last for a maximum of 15ms [41]. Therefore, the maximum of $15*(|V|+|E|)$ operations are performed during the traversals, where treating a single edge can lead to the summation of up to $L=15$ values. Although L has a known maximal constant value (15), we consider it as a variable in the complexity calculation, since it multiplies the number of operations, which are performed during the traversal of nodes and edges, and can be significantly higher than 1.

Since the simulation consists of the predefined number of time steps, let us designate it T, then the overall complexity of the simulation is calculated as the multiplication of the single step complexity O($L*(|V|+|E|)$) with the number of the steps T, rendering O($T*L*(|V|+|E|)$).

There are several minor improvements implemented in the model, which do not lead to the theoretical complexity decrease, however, they affect the actual number of

nodes and edges the algorithm has to traverse at each step of the simulation. These improvements are presented in the next section.

3.6 Minor simulation algorithm improvements

The 1st traversal should be performed on previously spiked nodes, and the 2nd traversal – on the nodes, which are not in refractory period. These two sets are complementary and can be treated separately, since previously spiked nodes enter refractory period in the 1st traversal and therefore are skipped in the 2nd traversal. The two sets are implemented as two disjoint lists “spiked” and “not spiked”, interchanging between consecutive time steps. At each step all the nodes from “spiked” are entering refractory period, so they will not spike this time, therefore all of them are moved to “not spiked” list of the next iteration. Some of the nodes from “not spiked” list are spiking, so they are moved to the “spiked” list of the next iteration. The rest of the nodes from the “not spiked” list are added to the “not spiked” list of the next iteration.

This swapping of the two lists requires only a few additional operations and memory objects, or formally O(1) additional time and space complexity. This improvement decreases the number of visited nodes to |V| in a single time step, as opposed to 2|V| visited nodes in the initial version of the algorithm, where each of the two traversals requires visiting |V| nodes.

In order to exclude the possibility of initialization of local potentials in the nodes, which are entering refractory period, we will use consecutive step counter for spiking nodes. At the beginning the global counter’s value is 0. All the nodes spiking at the same moment are put into the “spiked” list and are given the current counter value plus 1. When the simulation proceeds to the next time step, the global counter is increased by 1. When the “spiked” nodes are traversed, they should not send signals to each other, since they are entering the refractory period. This is achieved by “looking” into the counter value of a neighbour node: if it is less than the counter value of the currently treated node, then the node may send signals to the neighbour, otherwise (if their counter values are equal) it cannot. This is because

equal counter values mean that they spiked together, so they should have refractory period at the same time and both cannot accept signals.

An additional improvement comes from storing the list of the edges having active propagating local potentials in each node. The reason this reduces processing complexity follows. Let us designate the list as *AL[i]* (Active List of neuron *i*). Initially the list *AL[i]* is empty for all neurons in the NNN. When an edge receives a new local potential and it has no other active local potentials, it is added to *AL[i]*. When all the local potentials of an edge terminate or are reset due to the spike occurring in neuron *i*, the edge is removed from *AL[i]*. Hence, each time a node *i* is checked for local spikes, only the traversal of input edges with active local potentials *AL[i]* is required (not all the input edges of the node). This reduces the overall number of visited edges in single time step from |E| to the number of edges actually spiking at that step.

Chapter 4:

GA-based algorithms

As it was mentioned earlier, the framework presented in the previous chapter is used by GA in order to evolve input sequences to those which produce a better approximation of the given output. There are three algorithms (or 3 versions of GA based algorithm) developed in the scope of the present work: Simple GA (SGA), Fixed Input GA (FIGA) and Fusion GA (FGA). The three algorithms employ the same Template GA flow, while providing it with different components to be used. The components that each algorithm should provide to Template GA are the GA encoding for possible solutions, fitness function, mutation and crossover operators.

4.1 Requirements

Since, to the best of our knowledge, no algorithms exist in the literature for solving the defined problem, the results of the three algorithms will be compared to the results generated by the URS algorithm [46]. The performance comparison between the algorithms will be based on the same number of solutions evaluated during each algorithm's run. It means that each algorithm is allowed to generate and evaluate the same number of solutions during the run. We have chosen to restrict the search of each algorithm to 100,000 solutions solely on the basis of the total runtimes observed (see Table 6.1). The maximal runtime among the input data sets, presented in the table, is approximately two weeks for the data set called *Feedback network with CISGA generated input* on 2.4GHz Intel processor, when using 100,000 solutions. We were able to run different data sets in parallel, hence, the runtime bottleneck is imposed by the abovementioned data set. We consider any runtime longer than two weeks unfeasible for the purposes of the current research, therefore, the maximal number of evaluated solutions allowing for the feasible runtime is 100,000.

The criterion used to access performance is the degree of similarity between the output sequences of the best found solution and the target output sequences. Target output sequences are the output sequences that should be approximated by the algorithms. Generation of target output sequences will be presented in detail in the next chapter. However, it is necessary to understand that the target output sequences are not picked randomly. They are also generated from some input (consisting of participating input nodes and their input sequences) by invoking signal propagation in the given NNN using that input. This approach (of generating target output sequences) provides us with the corresponding perfect “target input”; the one, which ideally should be reproduced by the presented algorithms.

The implemented algorithms are described in detail in the following subsections.

4.2 Uniform Random Search

This simple technique relies on the variation paradigm only. In contrast to the GA, this algorithm does not rely on the selection of previously found good solutions. URS is based on the idea of guessing each bit in a solution.

Recall from the Fig. 1.1 that each solution is represented as a binary sequence of fixed length. For each bit, URS tosses a ransom coin, which could turn “0” or “1” with equal probability $p=0.5$, therefore the “uniform” term in the name of the algorithm, and this value is assigned to the bit. This way, the participating input nodes and their corresponding input sequences are uniquely determined in the generated binary sequence.

In the next step URS invokes the signal propagation framework on the generated input in order to record the output sequences, appearing at the output nodes of the network. After the framework invocation the resulting output binary sequences are compared in a bitwise fashion with the target output sequences.

URS works in a simple loop, which guesses and evaluates one solution per iteration. The best solution is kept through the run by comparing the presently held as best solution with the new newly generated solution, selects the one between them that gives better fitness and places it as the updated best solution. The

algorithm continues running until the number of iteration steps has reached a pre-specified threshold, which is 100,000 in our case. After termination it returns what is kept as best solution.

URS is the only algorithm appearing in this work that is not based on GA principles, except for the principle of variation. Recall that GA implements the principles of mating, variation, exchange and survival by using selection for mating, mutation, crossover and selection for survival operators, respectively. Out of those four principles and their implementing operators, URS employs only the principle of variation, implemented by random mutations. The algorithms presented next form the GA core of the thesis and introduce a way to apply GA methodology to the investigated problem.

4.3 Template GA

This section describes the basic template flow of the GA, regardless of the nature of the constituent components. Template GA controls several important parameters and template components, which are used in all the algorithms: population size, number of generations, selection for mating of solutions and selection for survival of solutions. Notice that these components will be elaborated in the following sections, since they are defined differently for different algorithms. This section presents only the common GA flow, which is shared by all the developed algorithms. For clarity, in this section the components are marked using **bold** font.

We direct the reader's attention to the following: the **population size** multiplied by the **number of generations** equals to the overall number of solutions, which are generated and evaluated during the algorithm's run. In order to prove this, let us designate the **population size** by Π and the **number of generations** by G. If we assume that we start with the initial population of size Π, which corresponds to the first generation, and at each next generation the number of newly generated solutions is Π, then the overall number of generated solutions is $O = \Pi * G$. This number (O) is restricted to 100,000 solutions.

First, let us go through the overview of the algorithm:

1. *Randomly generate the initial population of solutions with the size equal to the given **population size** parameter. (The process of random generation of solutions is explained in the next section.)*
2. *Assess the fitness of all the solutions in the population using the fitness function.*
3. *Perform the following loop with the number of iterations equal to the given parameter of the **number of generations**, keeping in each iteration the reference to the best solution generated:*
    a. *Select the pairs of solutions from the population for mating using the **selection for mating operator**. The number of the selected pairs should be equal to $\left\lfloor \frac{populationSize}{2} \right\rfloor$, where $\lfloor X \rfloor$ represents the result of rounding down X to the first integer. The selection is performed with repetitions, meaning that the same solution can be selected multiple times, while some other solutions may not be selected at all. All the unselected solutions remain in the population, as well as the selected ones, but the former do not produce children solutions.*
    b. *Each pair of mating solutions undergoes a crossover and mutations by invoking crossover and mutation operators. Template GA expects that two child solutions are created out of each mating.*
    c. *Assess the fitness of the newly created children solutions.*
    d. *Each newly created child solution undergoes **survival selection** against the worst solution in the existing population. This is done by picking the worst solution in the population and checking whether the child solution has a better fitness: if yes then the old solution is substituted by the child solution, if no then the child solution is discarded.*
4. *Return the best fit solution.*

The **population size** and the **number of generations** play crucial role in the computation. Setting the **population size** too small makes the GA search much more locally focused and dependant on the initial population, therefore increasing

the possibility of converging to one of the local optima, which are still far from the global optimum. On the opposite, setting it too large makes the GA search similar to the random search, hence losing the focus necessary for the gradual substantial improvement. The **number of generations** and the **population size** are interdependent (recall the formula for the overall number of generated solutions, *O = Π * G)*. Generally, too small **number of generations** may result in the premature termination of the algorithm, before the local optimum is reached. Both parameters are highly dependent on the problem solved by GA and no precise guidelines are available to find the optimal ones. The simplest approach is to try varying values of the parameters and check which values yield better results on average. Each version of the algorithm (SGA, FIGA, FGA) sets its own parameters' values for the **population size** and the **number of generations**. It is discussed in the corresponding sections, following this section.

The two important template components used in the Template GA flow (presented on the previous page) are the **selection for mating** and **selection for survival**.

As it is presented in step *3.d* of the template algorithm, selection for survival puts very high selection pressure on the unfit solutions. The worst solutions in the population do not have any chance to survive once better fit child solutions are generated. This high selection pressure should be balanced by a much less "selective" operator for mating selection. The most common types of selection operators are fitness proportional selection, ranking selection and tournament selection [8][47]. We considered the properties of each of these operators before selecting the operator used in the present work, as it is elaborated below.

In fitness proportional selection the probability of selecting a solution is in direct proportion to the fitness of the solution. If we designate the fitness of solution *i* as *f(i)*, then, we need to compute first the sum of all fitness values of the population: $S = \sum_i f(i)$. Second, the probability of selecting each solution *i* is computed as $p(i) = \frac{f(i)}{S}$ and the cumulative probability of selecting solution *i* is computed as

$q(i) = \sum_{j=1}^{i} p(j)$. The order of solutions in the population is not important, however, it should be kept constant during the process of selection for mating, since *q(i)* depends on it. Finally, when the selection for mating is performed, a random number *r* (with uniform distribution) in [0,1] diapason is generated and compared against the sequence of *q(i)* probabilities: if $r \le q(1)$ then select the first solution in the population, otherwise select solution *k* such that $q(k-1) < r \le q(k)$. The drawbacks of this technique are resulting from the extreme dependency of selection on the absolute fitness differences among solutions: (i) at the early stages of GA initially well fit solutions get excess advantage in mating, blocking other solutions from producing children, therefore causing more locally concentrated exploration of the solution space and premature convergence; (ii) later, when the differences in fitness among solutions become less evident, the selection loses power of distinction between slightly better and slightly worse solutions and turns into a random selection. Since the other two types of selection operators lack these critical deficiencies, we decided not to use fitness proportional selection.

Ranking selection requires ranking of the solutions in decreasing order of fitness, from 1 to **population size** (each solution *i* in the population gets a unique rank *rank(i)*, when the lower rank value means better fitness). Then, for solution *i* the probability of being selected is calculated as $p(i) = P^{rank(i)}$, where *P* is a fixed probability value, the parameter of ranking selection. Since the sum $S = \sum_i p(i) = \sum_i P^{rank(i)}$ is not equal to 1, *p(i)* values should be scaled by the factor of $\frac{1}{S}$ in order to sum up to 1. Then, identically to fitness proportional selection, the cumulative probabilities *q(i)* are calculated based on *p(i)*, for each solution *i* (from 1 to **population size**). A random number (uniform distribution) *r* in [0,1] is generated and solution *k* is selected, such that *r* falls within the [*q(k-1), q(k)*] interval. The crucial difference between this selection operator and fitness proportional selection is in the consideration of ranks instead of absolute fitness values. However, introduction of ranks implies sorting all the solutions in the population at each

generation in order to compute the ranks. In addition, both fitness proportional selection and ranking selection require searching for an interval corresponding to a random number in the sequence of *q(i)* terms per each solution ,selected for mating. These two extra operations, if implemented by standard sorting and searching algorithms [48], will slow down the selection step by the factor of $O(\log_2(\Pi))$, where Π is the **population size**. For example, for **population size** of 1,000 this factor is approximately equal to 10.

In tournament selection *n* solutions are picked randomly from the population and the best solution out of the *n* is selected with probability *q*. Each one of the rest of the picked solutions is selected with probability $\frac{1-q}{n-1}$. The two parameters *n* and *q* enable the adjustment of selection pressure according to the context. This technique, similarly to ranking selection, does not rely on the absolute fitness values of solutions. In addition, it has two important properties that make it a better choice than ranking selection. First, the selection pressure is easily adjustable: *q* and *n* values are in direct linear correlation with selection pressure. In ranking selection it is harder to adjust selection pressure by tuning a single parameter *P*, which is not in direct linear correlation with the selection pressure. Second, and most importantly, tournament selection requires neither sorting all the solutions in the population nor searching among them, since at each step only *n* solutions are picked and considered, when $2 \le n \le 10$. Therefore, tournament selection has much better runtime performance than ranking selection. Based on the above observations, we decided to use tournament selection as the mating selection operator for the current work.

The next step is to set the parameter values of tournament selection, namely, *q* and *n.* In our Template GA we have chosen to use tournament selection for mating with parameters set to $n=4$ and $q=0.4$. In this case the best fit solution out of the four picked solutions has the chance to be selected twice more likely than any of the three others. These parameter values were chosen in order to impose lower selection pressure for balancing the “greedy” survival selection of the Template GA. Tournament selection is often used with parameters $n=4$ and $q=0.7$. However, it

highly depends on the particular context of the problem and, like in our case, q needs to be set to the lower value in order to avoid premature convergence in the suboptimal local peaks.

The following sections present how Template GA is instantiated by the three main algorithms. The domain specific components supplied by the algorithms make the actual connection between the signal propagation in NNNs and the evolutionary process encoded in the Template GA.

4.4 Simple GA

This version of GA comprises the simplest instance of Template GA. It defines the main parameters and components of the GA technique in the context of the prediction problem investigated in the present work. The parameters are the population size and the number of generations. The components include GA encoding for possible solutions, fitness function, mutation and crossover operators. SGA performs a single Template GA invocation with the above mentioned parameters and components, which are discussed in detail in the following subsections.

4.4.1 Population size and Number of generations

Since the multiplication result of the population size and the number of generations should be equal to 100,000, two options were considered for SGA: (i) 100 solutions in the population and 1000 generations, versus (ii) 1000 solutions in the population and 100 generations. Option (ii) was chosen, since population size of 100 is more prone to the premature convergence in suboptimal local peaks.

4.4.2 GA encoding of solutions

The binary encoding of GA solutions for the problem has been already explained in section 1.5 (Fig. 1.4). Recall, the encoding is a binary string consisting of two consecutive substrings: the first substring encodes for participating input nodes (bit 0 denotes non-participation, bit 1 denotes participation), the second substring encodes for binary input sequences fed into each of the input nodes. The order of

these sequences within the second substring is the same as the order of the nodes in the first substring. Let us term the two encoding substrings as sub-encodings.

If the first sub-encoding contains only bits with value 0, then it encodes for no participating input nodes at all. We consider the solutions, which encode for zero participating input nodes, to be invalid, since they are not able to generate any signal in the network. When creating new solutions during the invocation of genetic operators (mutations and crossovers), this rule is applied and it enforces the generation of valid solutions only. The procedure for this enforcement will be described in section 4.4.4 **Crossovers and Mutations**.

4.4.3 Fitness function

The GA fitness function assesses the fitness of solutions according to the criteria specified in the context of the problem. In order to perform the assessment of a solution for the prediction of signal propagation in NNNs, the fitness function executes the following steps:

1. Generate output sequences by running signal propagation in the network using the input, encoded by the assessed solution.
2. Perform bitwise comparison of the output sequences, generated in the previous step, with the target output sequences. In order to be able to execute this step, the fitness function holds a reference to the target output sequences, which are supplied before the simulation. In addition, the result of the bitwise comparison is cached, so both steps 1 and 2 are executed only once per solution regardless of how many times its fitness is evaluated during the GA run.
3. Return the number of bit mismatches as the fitness of the solution. We have also considered use of weighted fitness function, which considers the locations of the mismatches and applies different weights to different locations. However, in the context of signal propagation in NNNs, we were not able to identify sources the user could acquire knowledge of the values of such weights, turning the idea of weighted fitness function to not being meaningful, at least not for the time being. Therefore, our fitness function

does not apply any weights, or equivalently, it gives equal weight to all mismatches. This definition of the fitness function implies that the discussed problem is a minimization problem, where a lower number of mismatches corresponds to higher fitness.

In order to better reward solutions for having less mismatches, the number of mismatches is calculated differently from what it would be intuitively expected. Each mismatch (1 against 0 or 0 against 1) adds 1 to the number of mismatches, which is expected. However, each match of 1 against 1 subtracts 1 from the number of mismatches, which is somehow counterintuitive, since it provides double effect for guessing the right bit. When a mismatch at some position changes to 1-to-1 bit match, not only the mismatch is not counted any more in the overall number of mismatches, decreasing the number by 1, but also the match itself subtracts 1 from the number, further decreasing it.

The match of 0 against 0 does not subtract 1 from the number of mismatches in the contrast to 1-to-1 match and therefore does not have the double effect described above. This difference in treating 0-to-0 matches versus 1-to-1 matches follows from the observation that for GA populations it was easier to generate 0 bits in the output by supplying 0 bit input sequences, since the solutions were not rewarded enough for replacing bits 0 with bits 1. Therefore, most of the generated solutions contained large number of 0's and were not useful to solve the problem at hand. In order to overcome this deficiency, we switched to the uneven mismatch strategy described above. As a result, the reward that the fitness function gives a solution for matching 0 against 0 is twice smaller than the reward for matching 1 against 1 (-1 mismatch and -2 mismatches, respectively).

However, we could not identify any differences in performance between strength>2 and strength=2 (here, the term *strength* refers to the degree of inequality of reward the fitness functions gives to 1-to-1 match as compared to 0-to-0 match) , thus we kept strength=2.

We suggest there is a particular threshold of correlation of 0's and 1's in the target output that may require adjustment of strength, like in the above case. However, the

identification of the threshold is analytically impossible and requires running simulations with SGA. If the algorithm converges to solutions with most of bits equal to zero (as opposed to the target output), then the strength should be increased.

4.4.4 Crossovers and Mutations

The versions of crossovers and mutations are different for different versions of GA. Here we will present crossovers and mutations used for the SGA. The other versions will be described in the sections corresponding to the related algorithms.

Crossovers and mutations are grouped together, since their application to the prediction problem is interdependent. Crossovers enable the exchange of the corresponding encoding parts of the two parent solutions, potentially creating new well fit encoding combinations in the children solutions. Mutations enable exploration of small search subspaces near the newly created solutions by introducing small random changes in their encodings. Once the crossover operator combines corresponding parts of parents to generate children, the mutation operator is immediately used to make small random changes in the children solution encodings in order to introduce innovation into the new generation.

Both crossovers and mutations have their associated probabilities. Crossover probability is the probability of having a crossover in a mating pair of solutions. This allows sometimes not having a crossover between parents when generating children. As a result, the created children are just the copies of their parents before the mutation operator is invoked. Mutation probability refers to the probability of having a single bit mutation and it applies to every single bit of the children solution encoding. Usually crossover probability is kept very high (close to 100%), while the mutation probability is kept very low (close to 1% or lower) [8]. It is worth mentioning that in nature, for example in humans, crossover occurs always when a child is conceived, while mutations are extremely rare, occurring once in approximately 2.5×10^8 nucleotides, or positions [49]. In contrast to naturally occurring mutations, the GA mutations are kept at much higher rates (or probabilities) in order to artificially increase the speed of evolution. This enhancement is supported by two considerations: (i) the user, who runs the software, does not wish to wait millions of

years to get some noticeable changes; (ii) the length of human genes' encoding is much higher than the length of GA encodings (order of trillions versus order of thousands, respectively).

The GA encoding of solutions consists of two sub-encodings, as it was mentioned earlier: the first encodes for the participating input nodes and the second encodes for their corresponding input sequences. The functionalities performing crossovers and mutations in the first sub-encoding of solutions are different from those used in the second sub-encoding. Let us examine the type of crossovers and mutations occurring in the first sub-encoding.

Since changing any bit in the first sub-encoding of a solution potentially causes great change to the generated output due to the different choice of participating input nodes, the crossover probability is chosen to be 70% in the first sub-encoding, therefore being lower than usually used values (90%-100%).. This reduction of crossover probability for the first sub-encoding prevents introducing too many changes of participating input nodes in the offspring, making the choice of participating input nodes more stable than the choice of their corresponding input sequences. It is important to emphasize that the decreased crossover probability for the first sub-encoding is still considerably higher than 50%, causing more crossovers than non-crossovers.

Given a pair of parent solutions, their first sub-encodings will undergo a crossover with probability 70%. In other words, out of 10 pairs of parent solutions, on average 7 pairs will have a crossover between their first sub-encodings before inheriting the sub-encodings to the offspring, while 3 pairs will inherit their first sub-encodings intact to the offspring.

Once a pair of first sub-encodings undergoes a crossover, the crossover type used is the uniform crossover, exemplified in Fig. 4.1, with the first sub-encodings having four bits. In this type of crossover a decision should be made whether of the two children will get the copy of each parent's bit. All the bits, located at the same positions of the two parent encodings, undergo the same decision process. The decision for each bit is made based on the result of tossing a coin randomly, each

side having probability *p=0.5* to come up*:* if head comes up, then copy a bit from Parent 1 into Child 1 and the corresponding bit of Parent 2 into Child 2; if tail comes up, then copy a bit from Parent 1 into Child 2 and the corresponding bit of Parent 2 into Child 1. Please note that this is unrelated to the crossover probability (which is 70%), since this is the internal tossing process of the uniform crossover. This crossover allows for the greatest degree of flexibility in generating new combinations, since any subset of the encoding sequences can be picked up for being copied into each child.

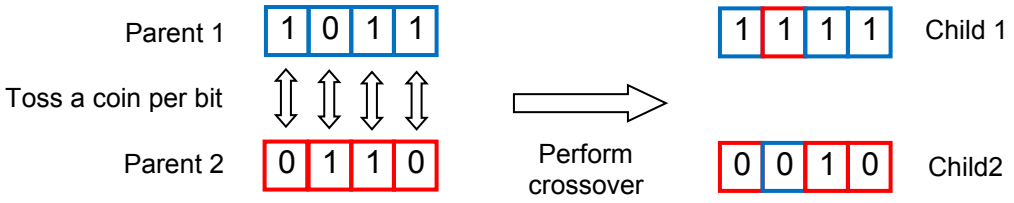


Figure 4.1. Uniform crossover in the first sub-encoding: red colour represents bits of Parent 1 and blue colour represents bits of Parent 2. Each toss determines which parent's bit is copied into each child. The sequence of tosses that led to the observed result is: [Head, Tail, Head, Head].

Mutation operator is applied to the first sub-encodings immediately after those are generated by the uniform crossover. Mutation probability for the first part is kept at the rate of 1%. This mutation probability corresponds to the chance of having a single bit inverted in 100 bits on average, because each bit in the first sub-encoding is inverted with 1% probability. In addition, the mutation operator is responsible for the correction of invalid children solutions generated with zero participating input nodes, which means that all the bits are zero in the first sub-encoding coming out of the mutation process. If such a solution is generated, it is identified after a crossover and mutations are performed. Then, an additional compulsory mutation is introduced in the first sub-encoding, rendering single random participating input node (see Fig. 4.2).

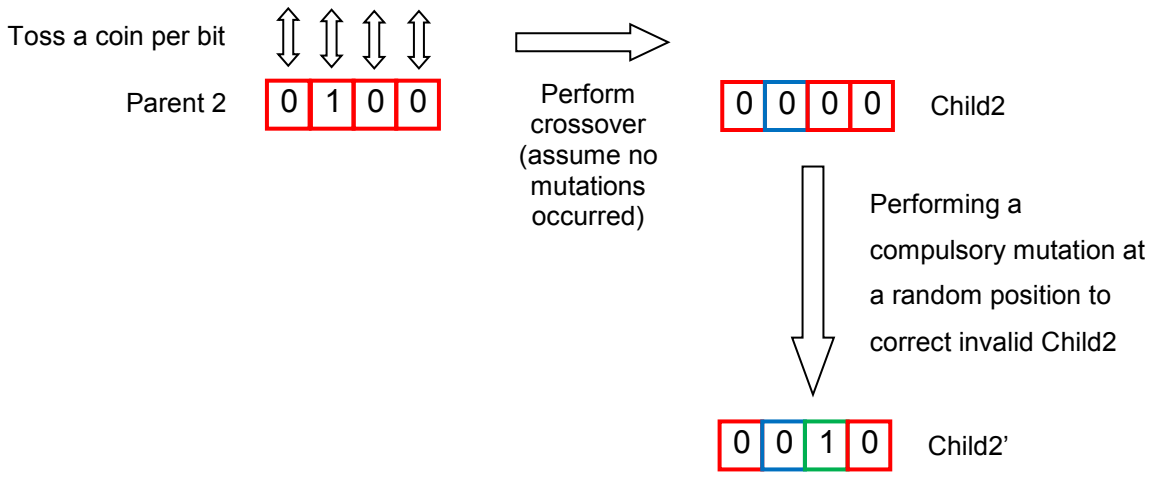


Figure 4.2. Performing a correction in the generated first sub-encoding with all bits zero (Child 2). A random compulsory mutation is performed at the third position (shown with green).

After the participating input nodes are determined by the first sub-encoding, the second sub-encoding of solutions should provide binary sequences, acting as input signals to the input nodes. All these binary input sequences have the same length and are treated similarly for different input nodes. In addition, binary input sequences, fed into different input nodes, are treated independently from each other. Since there are typically more than one input node, there are more than one binary input sequence. The second sub-encoding contains all the binary input sequences as sub-strings, placed in the same order as their respective input nodes in the first sub-encoding (see Fig. 1.4).

The selected crossover operator used in the second sub-encoding is the 2-point crossover [50] with 90% crossover probability. The 2-point crossover selects 2 random points on the equally probable basis in the recombined sequences rather than treating each bit separately, as it is done in the uniform crossover. L-point crossover for L > 2 was not used in the present work, since it did not show noticeable improvement in comparison to 2-point crossover. In addition, 1-point crossover (L = 1) performed worse than 2-point crossover, hence it was not used.

The two points of the 2-point crossover are selected without repetitions, so they cannot overlap. Let us assume input sequences consist of *M* bits (the length of the second sub-encoding). For each parents' pair, two numbers are selected randomly without repetitions, n_1 and n_2. For the sake of explaining the technique easier, let us also assume that $n_1 < n_2$., $n_2 \leq M$. Their selection is done in accordance to a uniform random process. We split each of the two parent sequences in 3 sub-sequences: $Subseq_1^j=[bit_1^j,\ldots,bit_{n1}^j]$; $Subseq_2^j = [bit_{n1+1}^j,\ldots, bit_{n2}^j]$; $Subseq_3^j = [bit_{n2+1}^j,\ldots,bit_M^j]$ where $j \epsilon \{1, 2\}$ identifies the parent sequence. The two children sequences are generated as follows: *Child 1*=[$Subseq_1^1$, $Subseq_2^2$, $Subseq_3^1$]; *Child 2*=[$Subseq_1^2$, $Subseq_2^1$, $Subseq_3^2$]. An example corresponding to $M = 4$ is shown in Fig. 4.3.

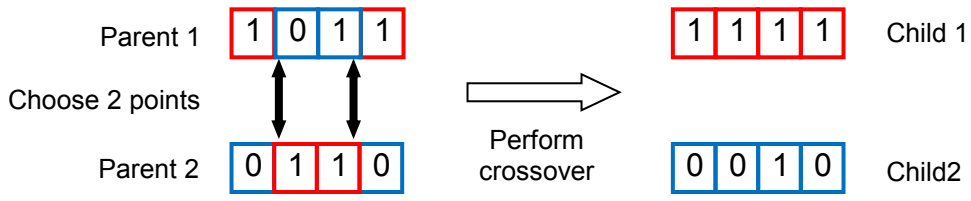


Figure 4.3. 2-point crossover in the input node sequence in the second part of GA encoding.

The mutation operator is applied immediately after the crossover generates a binary input sequence for every input node. It is the same operator used at the first sub-encoding. The same value of 1% is used for the mutation probability.

The algorithms to be presented next incorporate some additional heuristics in order to make more efficient use of the Template GA.

4.5 Fixed Input GA

If we denote the length of the first sub-encoding by *N* and the length of the second sub-encoding by *M*, then the Simple GA performs a search in the search space of $2^{N+M} - 2^M$ possible solutions (-2^M stands for the invalid encoding combinations). In simple words, the algorithm performs a double search: for both participating input nodes and their input sequences. We introduce Fixed Input GA in attempt to reduce the search space to 2^M. The algorithm separates the search for participating input nodes from the search for the input node sequences.

The algorithm heuristic is based on the assumption that input nodes, which originally participated in the generation of target output sequences, should have a greater impact in recreating the target output sequences than the rest of input nodes, if assessed separately. This assumption is not always realistic, since it implies that NNN should in a sense follow a close to linear behaviour. The auditory cortex model network, one of the two networks tested in the present work, does have linear structure , i.e. does not contain feedback loops, and it demonstrates much less complex behaviour than a network containing feedback loops (see section 6.2 **Input Linearity Levels**). Since the auditory cortex model network is based on naturally occurring NNN in the human brain, we consider the discussed assumption to be valid for at least some of NNNs. For the rest of NNNs, namely, the NNNs with non-linear complex behaviour, this algorithm will most likely produce poor results (as we can see in section 6.6 **Modified feedback network and CISGA generated input**). In this case, the SGA and FGA algorithms are preferable.

Separate assessment of each input node means that only one input node participates in each assessment. One of the ways to assess the impact of an input node is to run GA with only that input node participating in all the solutions of the population, while all the rest of the input nodes are not allowed to participate (their corresponding input bits in the first part of the encoding are always set to zero). The best found solution of each separate GA assessment is the indicator of the corresponding input node's impact.

Assume *K* out of the *N* input nodes originally participated in the generation of the target output sequences. Then, when using the approach presented above, running the GA for each of the *K* originally participating nodes separately will yield better fit solutions than running the GA for each of the remaining (*N – K)* nodes. Therefore, sorting the resulting fitness of the best fit solutions (one best found solution per one input node) in decreasing order should yield a list, in which the originally participating nodes are located at the beginning of the list. Then, starting from the beginning of the list, we try to include each input node in the set of participating input nodes: if adding a next input node improves the GA result, which we can know

by running the GA with this node added, then the node stays in the set, otherwise the node is dropped.

The formal algorithm flow is the following:

1. *Traverse all input nodes and for each node i:*
    a. *Run Template GA instance with node i as the single participating input node (all the rest of the input nodes are not allowed to participate).*
    b. *Record the fitness of the resulting best fit solution for node i.*
2. *Sort the list of the input nodes in decreasing order of the corresponding fitness, determined in the previous step.*
3. *Create the set P of participating input nodes. Initially P = { }.*
4. *Set the fitness of P to be +∞. (*)*
5. *Traverse all input nodes in the list from the beginning to the end and for each node i:*
    a. *Add i to P: P' = P ∪ {i}.*
    b. *Run Template GA instance with the set P' of participating input nodes (all the rest of the input nodes are not allowed to participate).*
    c. *Record the best fit solution of this GA run. Denote it as Best(P').*
    d. *Check if the fitness value of Best(P') is lower than the fitness value of P:*
        - *If yes, then assign P ←P', set the fitness of P to be equal to the fitness of Best(P') and record Best(P') as the global best fit solution.*
        - *If no, then leave the previous set P (without node i).*
6. *Return the global best fit solution.*

(*) Here we introduce the fitness of a set of input nodes P, as the fitness of the best solution found by the GA, when P determines the participating input nodes. This measure is initially set to be infinitively high, or in other words bad, and will be decreasing, or improving, as input nodes are added to it. If adding an input node does not improve the fitness of P, the node is not added).

The presented algorithm employs the number of Template GA instances, which should be supplied with the appropriate parameters and components in order to be used. The important point to notice is that the first sub-encodings, which encode for participating input nodes, are identical for all the solutions generated within any single Template GA instance (step *1.a* and step *5.b* in the above FIGA flow). In step *1.a* only input node *i* participates at a time, while in step *5.b* only input nodes in *P'* participate, rendering the exact same first sub-encodings for all the solutions within a single Template GA instance. This directly impacts the crossover and mutation operators defined for FIGA: neither crossovers nor mutations are allowed for the first sub-encodings of solutions. The implementation of this restriction is provided in section 4.5.2 entitled **GA components implementation**.

All the used Template GA instances require the same type of components, which are discussed further.

4.5.1 Population size and Number of generations

The main difference between this algorithm and SGA is that the complexity of this algorithm depends on the number of input nodes. While the complexity of SGA computation does not depend on the number of input nodes, FIGA is linearly dependent on the number of input nodes: in the previous section (4.5), step 1 and step 5 of FIGA are both performed in O(N * {complexity of single Template GA}), where N is the number of input nodes, also equal to the length of the first sub-encoding. Since the number of input nodes in the tested NNNs is 10 (see the next chapter) and the maximal number of evaluated solutions is 100,000, the following sizes of population and numbers of generations are used for each Template GA instance in the algorithm:

- Each Template GA instance in step *1.a* of FIGA (section 4.5) has population size equal to 100 and the number of generations equal to 50, therefore for 10 input nodes 50,000 solutions are evaluated overall in this step

- Each Template GA instance in step *5.b* of FIGA (section 4.5) has population size equal to 100 and number of generations equal to 50, therefore for 10 input nodes 50,000 solutions are evaluated overall in this step

As we can see, the presented algorithm has fewer solutions per population and fewer generations per GA instance compared to Simple GA, which runs a single Template GA instance with population size of 1000 and number of generations 100. However, the search space in each Template GA invocation is reduced to 2^M (where M is the length of the second sub-encoding) in each GA run, confirming the reduction in the population size and in the number of generations. The preprocessing steps *1* and *2* of the algorithm (section 4.5) enable considerably more focused search, but also make the algorithm overly dependent on the separate nodes' assessment, while the sorted list of assessed nodes may not always reflect correctly the real impact of input nodes. In fact, the results presented in Chapter 5 will highlight this limitation of the algorithm for specific (non-linear) inputs.

4.5.2 GA components implementation

The algorithm reuses most of the domain specific components defined in SGA, such as solution encoding and fitness function. However, random initialization of the Template GA populations differs from SGA, since the participating input nodes are pre-set in steps *1.a* and *5.b* of FIGA (section 4.5) and therefore cannot be initialized randomly. Only the second sub-encodings of solutions are initialized randomly.

Crossover and mutation operators are also slightly different from SGA, since none of the operators is allowed to modify the first sub-encodings due to the immutable set of participating input nodes per Template GA instance. The name of the algorithm reflects the immutability of the participating input nodes.

The crossover operator has always crossover probability equal to 0 for the first sub-encoding. In fact, it does not matter if the crossover is invoked on the first sub-encoding or not, since encoding of the two same sets of participating input nodes is identical and therefore no change can be introduced by recombining two identical encodings using uniform crossover or 2-point crossover. In order to avoid meaningless crossover operations the probability is set to 0. The same setting of

zero probability is applied to the mutation operator for the first sub-encoding. In addition, no check is needed to verify the validity of the generated solutions, since each solution always has the same valid set of participating input nodes, pre-set by the main algorithm.

The crossover and mutation operators for the second sub-encoding have exactly the same implementation as in SGA. The corresponding probabilities are kept the same – 90% and 1%, respectively.

4.5.3 Scalability issue

The algorithm does not scale well in regard to the number of input nodes. The required number of Template GA instances to be invoked is twice greater than the number of input nodes in the network. The amount of solutions each Template GA instance evaluates is inversely proportional to the number of input nodes, because the entire algorithm is not allowed to evaluate more than the constant total number of solutions, specified by the user (in our case, 100,000). This limitation puts a restriction on the number of input nodes in the network to be analyzed by FIGA.

However, we believe that 10 input nodes is a sufficient number of input nodes to be tested in the scope of the present work due to the processing time limitation and increasing approximation error (the number of output bit mismatches). The reason for deteriorating performance is the increasing size of search space $2^{N+M} - 2^{M}$ (where N is the number of input nodes and M is the length of input sequences) with increasing N. As we can see, the increase is exponential in N, if M is kept constant. Using a supercomputer can improve the scalability, but only to a certain extent. Suppose we use one of the latest supercomputers, Titan, introduced in 2012 (see the official website for details: http://www.olcf.ornl.gov/titan/). Its performance is about 20 Peta-Flops per second, while the performance of the computer used in the current research is about 10 Giga-Flops per second. Rough performance comparison of these two computers yields the ratio of 2,000,000. Therefore, if we want to increase the number of input nodes by using Titan, while staying within the same processing time, we can add about 21 input nodes (since 2^{21} = 2,097,152 ≈ 2,000,000).

When using our computer, the amount of input nodes greater than 10 does not seem to be feasible for handling by any version of GA. In Chapter 6 sections 6.3, 6.4, 6.5, 6.6 (see the upper plots of Fig. 6.3, Fig. 6.4, Fig. 6.5, Fig. 6.6) it is shown how increasing the number of participating input nodes causes deterioration of the algorithms' performance in terms of the number of mismatches between target output sequences and generated output sequences.

4.6 Fusion GA

This version of GA was developed to fix the main weakness of FIGA, namely, its dependence on the assessment of separate input nodes, which does not take into account possible interactions among signals coming simultaneously from different input nodes. At the same time, Fusion GA (FGA) preserves some degree of search space reduction, achieved in FIGA by the separate assessment of the input nodes. The name of the algorithm reflects the fact that it combines the main features of both SGA and FIGA, which are the flexibility (the search is not restricted to particular combinations of participating input nodes) and the reduction of the search space respectively.

The algorithm has two phases, just as FIGA does: the initial assessment (the first phase) and the main search (the second phase). Recall that we know only the set of potential input nodes, but we do not know which of them are supposed to receive input signals – we are interested in identifying the best sub-set of input nodes to be fed with input signals, and we refer to those input nodes as participating input nodes. In the phase of the initial assessment, rather than assessing separate input nodes, FGA dissects the search space into several sub-spaces, defined by the number of participating input nodes in a solution (equal to the number of bits 1 in the first sub-encoding of a solution). That is, a sub-space containing solutions with 1 participating input node only, a sub-space containing solutions with 2 participating input nodes only, ..., a sub-space containing solutions with maximal possible amount of participating input nodes only (this amount is equal to the overall number of input nodes in a network; the two networks tested in the present work contain 10 input nodes).

Let us examine the algorithm's flow:

1. *First phase: Iterate number **n** from 1 to **N**, where **N** is the overall number of input nodes in the network (in the present work **N**=10, therefore **n** runs from 1 to 10):*
    a. *Run Template GA instance with the number of participating input nodes set to **n** (all solutions are generated with exactly **n** participating input nodes). Notice that we only set the number of participating input nodes, while the set of participating input nodes is formed by Template GA.*
    b. *Record the fitness of the resulting best fit solution and the set of its participating input nodes. Designate it **BestSolution(n)**.*
2. *Second phase: Pick up a solution having the best fitness among **N** best fit solutions $\{\textbf{BestSolution(n)}\}_{n=1,2,..,N}$ recorded in the first phase. Designate it **BestSolution**.*
3. *Third phase: Run Template GA instance with **BestSolution**'s set of participating input nodes (all solutions are generated with these participating input nodes only).*
4. *Fourth phase: Return the best fit solution found in the previous step.*

This approach allows exploring considerably more input nodes' combinations as compared to FIGA, while still restricting the number of participating input nodes in the search performed in the first phase (by setting it to ***n***). The set of participating input nodes ***P*** that achieves a better output fitness for all possible values of ***n*** (***BestSolution***'s set of participating input nodes) is passed to the second phase, where the search concentrates only on solutions with the set of participating input nodes identical to ***P***, just as in FIGA.

The effect of preserving the number of participating input nodes is achieved by using a new method of applying crossovers and mutations: we need to keep the amount of participating input nodes unchanged, while allowing for variation in the set of participating input nodes. None of the known general crossover / mutation techniques could guarantee preserving the amount of participating input nodes.

These crossovers and mutations keep the number, but not the set of participating input nodes. In addition, the main GA parameters, population size and number of generations, differ among different Template GA instances, which are invoked by FGA.

4.6.1 Population size and Number of generations

The limitation of 100,000 solutions is divided among the Template GA instances in the following way:

- Each Template GA instance invoked in step *1.a* has population size equal to 100 and number of generations equal to 50.
- Template GA instance invoked in step *3* has population size of 100 and number of generations of 500.

The search space that the algorithm explores in steps *1.a* and *3* is considerably higher than the search space explored by FIGA in any of its Template GA instances. Recall that for FIGA it is 2^M (where *M* is the length of the second part of the encoding). In the case of FGA, for each *n* participating input nodes the search space is equal to $\binom{N}{n} \cdot 2^M$, where *N* is the overall number of input nodes and $\binom{N}{n}$ stands for the number of combinations of *n* out of *N* input nodes (equal to $\frac{N!}{n!(N-n)!}$, where *n!* stands for "*n*" factorial). The latter term reflects the difference in the search space between FIGA and FGA for each *n*. The cumulative complexity of the search space explored in step *1* for all *n* is nevertheless equal to $2^M \cdot \sum_{n=1}^{N} \binom{N}{n} = 2^M \cdot \left(2^N - 1\right)$, the same as in Simple GA. However, the dissection of the search space into *N* subspaces provides a much more focused search per each Template GA instance in step *1*.

In addition, step *3* of FGA concentrates on exactly one subspace. This allows investing more effort in that particular subspace by having single Template GA instance run for 500 generations. Although, FGA depends on the initial assessment,

the combination of participating input nodes is determined with a much more flexible process. This is the main difference between FGA and FIGA.

4.6.2 GA components implementation

In order to enable the genetic operations required by FGA, new genetic operators are introduced for the treatment of the first part of the encoding. All the operators for the second part of the encoding are kept unmodified as they are defined in Simple GA.

First, random generation of the initial population in each Template GA instance requires generation of solutions with the same fixed amount of participating input nodes. This is implemented by randomly picking up the required amount of input nodes without repetitions out of *N* input nodes.

Second, the crossover operator is required to generate children with the same amount of participating input nodes as their parents. The operator attempts to preserve the shared participating input nodes of the parents in both of the children, while interchangeably copying not shared participating input nodes into different children. The idea is illustrated in Fig. 4.4 with $N = 5$ and $n = 3$. The crossover is given the name **Shared Interchange Crossover**.

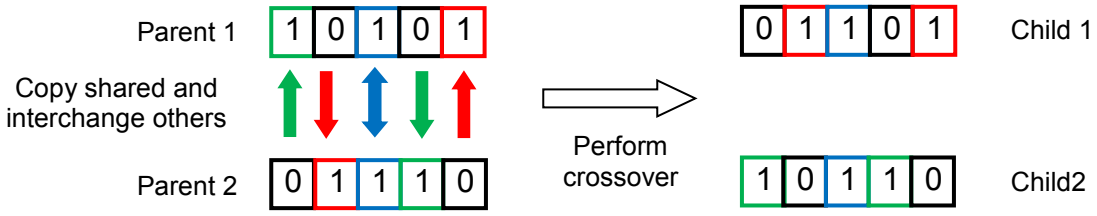


Figure 4.4. Shared Interchange Crossover: green arrows pointing to the green cells show the bits 1 copied to Child 2, red arrows pointing to the red cells show bits 1 copied to Child 1. Blue bidirectional arrow pointing to a blue cell show the shared bit 1 copied to both children. Unidirectional green and red arrows consecutively change their colour, meaning that each next unshared bit 1 is copied to a different child. Black cells contain bits 0, which are not copied. Initially children contain only bits 0, which are overwritten by bits 1 from parents.

Copying shared bits 1 to both children and interchanging unshared bits 1 between children ensures that at the end of the crossover both children have the same number of bits 1 and the number is equal to the original number of bits 1 in the

parents. If there are *n* participating input nodes, *k* of which are shared between the parents, then the number of bits 1 copied to each child is equal to $k + \frac{2 \cdot (n-k)}{2} = n$, where *2* in the numerator reflects the presence of 2 parents, while *2* in the denominator takes into consideration the interchangeable fashion of the crossover operation on the unshared bits 1.

Crossover probability is kept at 70% for the first part of the encoding and 90% for the second part of the encoding.

Finally, the mutation operator for the first part of the encoding is also required to preserve the number of participating input nodes in the solution. We use Swap Mutation [8], which swaps pairs of unequal bits in the same sequence, therefore keeping the number of bits 1 unchanged. Swap Mutation was selected over a simple mutation, which flips a random bit in a sequence, since the latter cannot preserve the number of bits 1. The probability is set to 1%, as in all the rest of the algorithms, but here it is interpreted slightly differently. Each bit in the first part of the encoding has 1% chance of being swapped with any other unequal randomly picked bit in the sequence. In the case when all the bits are equal (all the bits are 1), no mutations are performed. It is impossible to get solutions with all bits 0 before applying the mutation, since Shared Interchange Crossover is guaranteed to produce only solutions with *m* participating input nodes, where *m* runs from 1 to 10 and cannot be 0.

4.6.3 Scalability issue

FGA has the same scalability drawback as FIGA. If the number of input nodes is high, then each Template GA instance will be allowed to process smaller amount of solutions than might be sufficient for the efficient search. Since we do not test more than 10 input nodes in the present work, the algorithm still scales well for our purposes.

Chapter 5:

Generation of inputs for testing

In order to test the applicability of the proposed algorithms and to conduct their performance comparison, we need to define the input, to be used for the tests. The input consists of the NNN structure and its related signal propagation parameters. In addition, we need to define the input sequences generating the target output sequences in the given NNN. It is important to mention that the same NNN may yield completely different patterns of signal propagation, and therefore different target output sequences, and is depending on the input sequences provided. As a result, ways of generating input sequences for the tests play a crucial role in the scope of the present work.

5.1 Overview

In section 5.2 of the chapter we present the auditory cortex model network, the structure of which is taken from Chrostowski et al. [19], and the settings of the related signal propagation parameters. Section 5.3 elaborates on a generating so-called pulse shaped input sequences, which are used for the generation of target output sequences in the auditory cortex model network. In section 5.4 we present a measure for assessing the linearity level of inputs and their generated target outputs. The presented linearity level reflects the difficulty in approximating the generated target output sequences by the algorithms developed in the thesis. Based on this linearity level measure, we developed a GA-based technique for generating non-linear input sequences and their corresponding target output sequences, which are harder to approximate. The technique is presented in section 5.5 of the chapter and it is applied to the auditory cortex model network. Finally, we further increase the non-linearity of the generated input by adding excitatory and inhibitory feedback loops into the auditory cortex model network in section 5.6. The new model NNN is

used together with the developed non-linear input generation technique in order to generate target output sequences, which are extremely hard to be analysed.

5.2 Auditory cortex model network

The scheme of the network is presented in Fig. 5.1. There are 3 layers of neurons, which are depicted by yellow circles, blue triangles and green diamonds. The overall amount of neurons is 469 (not all neurons are included in the figure). Excitatory synapses are depicted by yellow, blue and green arrows. Each neuron of the first layer is connected to the 19 closest neurons of the second layer (yellow arrows). Each neuron of the second layer is connected to its 8 closest neurons of the same layer (blue arrows). Also each neuron of the second layer is connected to its 13 closest neurons of the third layer (green arrows). Inhibitory synapses are depicted by purple and black lines. Each neuron of the third layer is connected to its 6 closest neurons, located at the same layer, with inhibitory synapses (purple lines). In addition, each neuron of the third layer is connected back to its closest 39 neurons of the second layer with inhibitory synapses (black lines). The network has about 10,000 synapses in total. Synaptic weights are in inverse proportion to the spatial distance between the pairs of neurons. Input sequences are applied to the neurons of the first layer (red incoming arrows at the bottom). Output sequences are recorded for all the neurons located at the third layer (red outgoing arrows), one output sequence per neuron.

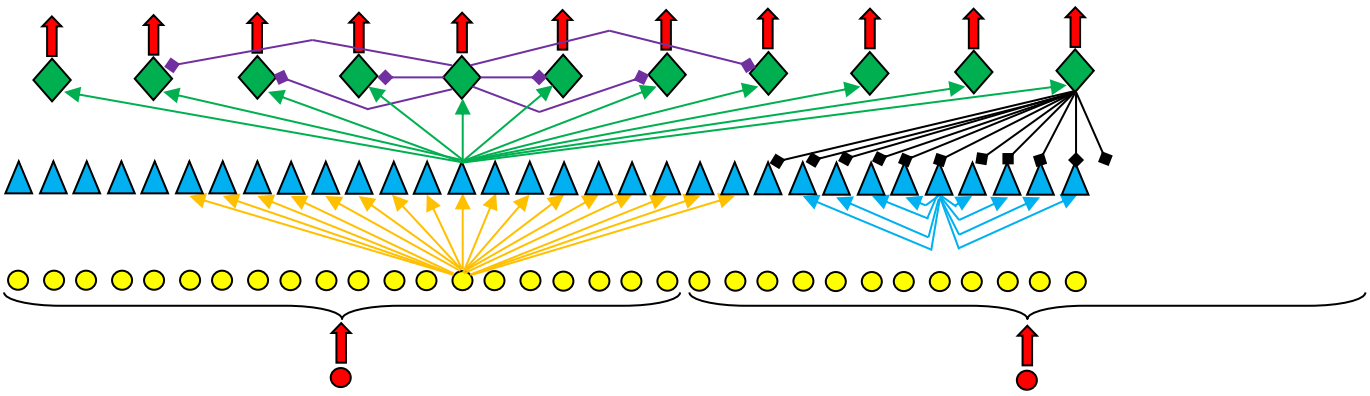


Figure 5.1. Auditory cortex model network (layer 1 – yellow circles, layer 2 – blue triangles, layer 3 – green diamonds).

Since there are about 200 input neurons in the first layer of the original network, we simplified search complexity by introducing only 10 candidate input neurons into the network (layer 0, red circles in Fig. 5.1). Each candidate input neuron is connected to 20 consecutive neurons of the 1st layer with 20 edges, each having synaptic strength of 1. By doing so, the added candidate input neurons span the entire range of the neurons at the 1st layer and are able to activate the entire network.

Input/output sequences were chosen to be of size 100 bits. Therefore, the length of the first sub-encoding of GA equals 10 and the length of the second sub-encoding equals 1000 (10 nodes multiplied by the size of a single sequence, which is 100). The complexity of the search space for the defined parameters is 2^{1010}, where 1010 stands for the overall length of the encoding. However, there are invalid combinations out of these 2^{1010}, corresponding to input sequences with no participating input nodes or, in other words, with all bits 0 in the first sub-encoding. The number of these invalid combinations is 2^{1000}, so the adjusted complexity of the search space equals $2^{1010} - 2^{1000} = 2^{1000}(2^{10}-1) = 1023 * 2^{1000}$.. The number of recorded output sequences is equal to the number of the output nodes. There are 67 output nodes overall in the 3rd layer, yielding 67 sequences, each having length of 100 bits.

5.3 Pulse shaped input sequences

In general, each test is generated in two steps. First, a number indicating how many input nodes will be participating, is randomly picked. Per each selected number of participating input nodes (running from 1 to 10) there is a corresponding test that is randomly selecting as many input nodes as the number indicates, with uniform distribution. Second, the input sequences are generated for all the selected input nodes in order to generate target output sequences. In the current section we present the way of generating simple input sequences having pulse shaped form.

An input sequence with pulse shape is a binary sequence, in which all its bits are 0s except of those –(bits in the sequence) having the same constant distance in bits between each other. The distance between 2 consecutive bits having a value of 1 is referred to as the pulse frequency. All the pulse frequencies are constant and equal

for all the participating input nodes. Therefore, each input sequence has the exact same shape with bit 1 occurring with the required pulse frequency. See the example of pulse shaped input with frequency 3 in Fig. 5.2.

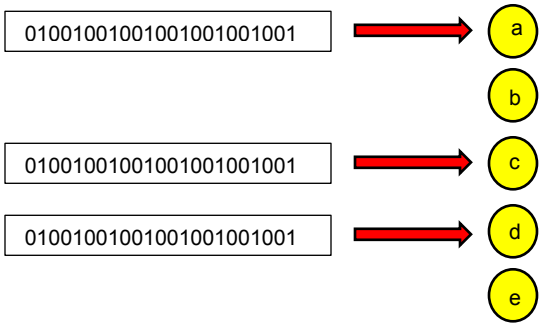


Figure 5.2. Pulse shaped input sequences with frequency 3 applied to participating input nodes a, c and d. Input nodes b and e are not participating, therefore not fed with any input sequences.

We are interested to find out what is the effect of different pulse frequencies on the performance of the algorithms. To achieve this, we generated three test sets. The first set uses the pulse frequency $f = 3$, the second set uses $f = 6$ and the third uses $f = 12$. The reason for choosing these three particular frequencies lies within the topology of the auditory cortex network: it has three layers. It takes 3 time units for an input signal to reach output nodes. Frequency 3 is aimed to check the ability of the developed algorithms to approximate input signals with "distinguishable" pulse in the context of the model network. Higher frequency (for example, frequency 2) could create unnecessary interference of signals in the network. Frequencies 6 and 12 are aimed to check the ability of the algorithms to correctly approximate the location of bits 1 within "sparse" sequences (having mostly bits 0).

Each test set includes 100 tests. For each number of participating input nodes that is selected to be tested, we run 10 tests (i.e., 10 tests for each i participating input nodes, where i runs from 1 to 10), therefore yielding 100 tests. It is important to mention that all three sets (corresponding to f = 3, 6, 12) are run using the auditory cortex model network as the underlying NNN. Unfortunately, it was not possible to

run more than 10 tests for each test group, because the execution time of the program for a single testing set would take more than a week (for all three algorithms). More precise runtime measures are presented in the next chapter.

5.4 Input linearity level

Generating simple inputs is not sufficient for the thorough testing of the proposed algorithms. We would like to generate inputs having higher degree of non-linearity, i.e., where signals applied to different input nodes are harder to reconstruct once they reach output nodes, if compared to pulse shaped inputs. In order to be able to measure the linearity level of the generated inputs, we introduce a metric, which reflects how difficult it would be to reproduce a given input with regard to its generated output. We believe that the proposed metric reflects the level of linearity of signal propagation interactions within the network. By signal propagation interactions we mean either the interference in the network due to refractory periods or the accumulation of local potentials from different signals contributing to initialization of new spikes in the nodes of the network.

The basic idea behind the metric is that each participating input node contributes its part to the generated output sequences. If the participating node's signals do not interact with other nodes' signals during their propagation through the network, then the node's corresponding part of the output signal is easily distinguishable in the generated output sequences.

The extreme example of this situation is when a network has several unconnected components, each component containing one input node and one output node. In this case the signal propagation from each participating input node is completely independent from the other input nodes' signals, since they propagate in different unconnected components. Each output node has its output sequence generated from a single input node located in the same component. It is sufficient to treat each input node separately in order to identify which part of the output signal is generated by the node. In this case, the behaviour of the NN is linear, meaning that the output generated when applying several individual input signals jointly, is the superposition of those outputs which become generated when each one of the individual input

signal is applied on its own. Therefore, it is extremely easy to identify which input nodes participate and which do not participate in the generation of the output sequences. We will refer to this case as the linear case.

On the other hand, if there are numerous interactions among the signals whose generation is initialized by different input nodes, no part of the output sequences can be unambiguously identified as originated from any separate input node (the NN does not follow linear system behaviour). Trying to regenerate the overall output signal as a superposition of separate output signals, generated from single input nodes, is expected to fail in this case. This observation is the core of the proposed metric.

The metric was originally derived from the FIGA approach, which makes its initial assessment using separate input nodes. In order to “deceive” FIGA, input signals generated at different input nodes should have a high level of interaction with each other while propagating towards output nodes. However, the performance of all the proposed algorithms, and not only FIGA, will be affected by incorporating the metric in input’s generation procedure, as we will see in the following chapter.

Let us define the precise procedure for computing the input linearity level metric for any given input:

1. *Initialize the superimposed output sequence (designated SOUT) with bits 0 for all the output sequences (overall, a sequence having 6700 bits 0, in our case).*
2. *Traverse all the participating input nodes in the given input and for each participating input node i:*
    a. *Propagate the input signal sequence applied to node i only, in order to generate output sequence i (designated OUT[i]).*
    b. *Perform the superposition of OUT[i] and SOUT, which is the bitwise OR operation between the two sequences (at each position: if both sequences contain bit 0, then write 0, otherwise write 1). The resulting bits overwrite the original SOUT bits.*

3. *Propagate the input signal defined by the given input (this time using the participating input nodes altogether) in order to generate original output sequence (designated OOUT).*
4. *Perform bitwise comparison of SOUT and OOUT and save the number of bit mismatches (designated as mS).*
5. *Return mS divided by the length of the output sequence (6700, in our case).*

There are several key points to be emphasized regarding the above computation:

- The calculated metric highlights the level of similarity between the superimposed output sequences (each generated by a applying a single – individual – input to the NN) and the overall output sequence: it is high for high similarity and low for low similarity.
- Technically speaking, the input linearity level is inversely proportional to the metric: the lower the value of the metric, the higher the non-linearity and vice versa.
- The superposition of separate outputs is employed in the computation of the metric, since spikes generated at output nodes at the same time behave in binary OR fashion. It does not matter how many spikes can be potentially generated at the same time moment in an output node, maximum one spike is recorded.
- Interactions between nodes of the network can change signals coming from separate input nodes, resulting in the change of sequences generated at the output nodes in a non-linear manner. High level of input linearity corresponds to high similarity between the superimposed output sequences of used multiple inputs and the generated output sequences. Although it is theoretically possible that numerous interactions within the network may cancel each other out and result in low level of input linearity, it is much more likely that they will cause significant change to the output signal.

By using the metric we can measure the linearity level of generated inputs and their corresponding target outputs. The target outputs and their corresponding inputs in the three test sets, defined in the previous subsections, will be assessed using this

metric in order to get the indication of how much linear the constant frequency pulse shaped input is. The metric is also employed to generate highly non-linear inputs, as it is explained below.

The next section presents a GA-based algorithm called Complex Input Search GA (CISGA), which is designed exactly for this purpose. We will introduce the algorithm having the opposite goal from the one pursued by the earlier proposed algorithms. While those algorithms (SGA, FIGA and FGA) are aimed to solve the prediction problem, a new algorithm (CISGA) is aimed to make the prediction problem as hard to solve as possible, namely, its goal is to maximize the non-linearity metric value of the generated inputs

5.5 Complex Input Search GA

The proposed algorithm employs the evolutionary genetic approach in order to evolve input signals having high level of non-linearity. The metric defined in the previous subsection is used as a fitness criterion for the generated solutions. The goal of the algorithm is to minimize the linearity metric value of the generated solutions. Once the best fit input signal is generated by CISGA, its corresponding output signal is used as the target output for all the rest of the algorithms (URS, SGA, FIGA and FGA). These algorithms in their turn try to generate the input, which reproduces the target output, generated by the best fit input of CISGA, with the best possible precision.

On the implementation level CISGA invokes a single instance of the same Template GA used by the rest of the algorithms. We will follow the same description pattern, which was used in the previous chapter, to present the parameters and components passed by CISGA to Template GA by CISGA.

5.5.1 Population size and Number of generations

Since the algorithm is not compared to any other algorithm, there is no limit on the number of solutions evaluated. However, in order to make its runtime reasonably low, we decided to use 50 generations and population size of 100 solutions as its parameter values. This brings the overall number of evaluated solutions to 5000,

which is 20 times less than the number of solutions evaluated by any of URS / SGA / FIGA / FGA. The reason for using fewer evaluated solutions in CISGA is in order to keep the problem solvable and the run time comparatively low. Additional rational is that it is generally much harder to solve the problem than to generate it. Since population sizes of 100 or 1000 are widely used, we decided to set population size at 100, while keeping the number of generations at 50, because using fewer generations is usually not recommended. For example, population size of 5 with number of generations of 1000 would lead to evaluation of the same amount of solutions (5000), however, this setup is highly likely to be much less efficient due to insufficient population size. Insufficient population size does not allow much “space” for random variation [8].

5.5.2 GA components implementation

Before running the algorithm, the number of participating input nodes, required by a particular test, is randomly chosen from the input nodes. Therefore, when the algorithm is run, the set of participating input nodes is known and is not allowed to change during the run. This puts specific restrictions on the used GA components.

Random generation of new solutions is performed using the exact same set of participating input nodes during the entire run. As a result, all the solutions are generated with the same participating input nodes. Definition of solutions is identical to the one provided in section 1.5. They are sequences of binary format and consist of two sub-encodings: the first sub-encoding is used for the selection of participating input nodes, while the second sub-encoding acts as input sequences to the input nodes.

The fitness function is much more complex than any of the previously defined fitness functions, since it has to compute the linearity metric for each generated solution. The function invokes the algorithm presented in the previous subsection per each solution, which should be evaluated, and caches the result in order to avoid costly re-computations. Pay attention to the fact that each time the input linearity level is calculated, the number of times the signal propagation framework is invoked equals to the number of participating input nodes plus one. First, each

participating input node is used separately. Second, all the participating input nodes are used altogether.

The crossover operator of FIGA is re-used here. Let us recall: it defines 0% crossover probability for the first part of the encoding and uses 2-point crossover with 90% crossover probability for the second part of the encoding. Similarly, the mutation operator of FIGA is reused here. It defines 0% mutation probability for the first part of the encoding and single bit mutation with the 1% mutation probability for the second part of the encoding.

5.6 Test set generated by CISGA

The forth test set used in the present work includes 100 tests with inputs (and, as a result, target outputs) generated by CISGA. Each subset of 10 tests has *i* participating input nodes (*i* runs from 1 to 10). As it was mentioned earlier, the inputs generated by CISGA are pipelined into each of the algorithms URS / SGA / FIGA / FGA, which use the generated target outputs, but not the inputs used to generate them.

It is important to mention that up until now we used the same auditory cortex network for all the four test sets. The drawback of the network is that it does not have excitatory feedback synapses between the layers of neurons (recall, there are three neuron levels), which may contribute to the generation of more complex inputs. In the next section we introduce a modification (extension) of the auditory cortex network with additional excitatory and inhibitory feedback synapses, which are designed to allow for more intense interactions between different neuron layers in the network. We will refer to this network as the modified feedback network.

5.7 Modified Feedback Network

The modification of the auditory cortex model network is comprised of two stages. First, we will start by deleting some of the originally present inhibitory feedback synapses. By doing this we aim to increase the number of spikes that reach the output nodes (layer 3). Second, we will introduce new excitatory and inhibitory synapses from output nodes (layer 3) back to the input nodes (layer 0).

The result of the first modification stage is presented in Fig. 5.3. The inhibitory synapses going from layer 3 back to layer 2, shown in Fig. 5.1 with black lines, are eliminated. The rest of the graph remains unchanged. It is important to mention that the inhibitory synapses within layer 3 (purple lines) are preserved in order to maintain some degree of the local containment of signal propagation, once signals reach layer 3.

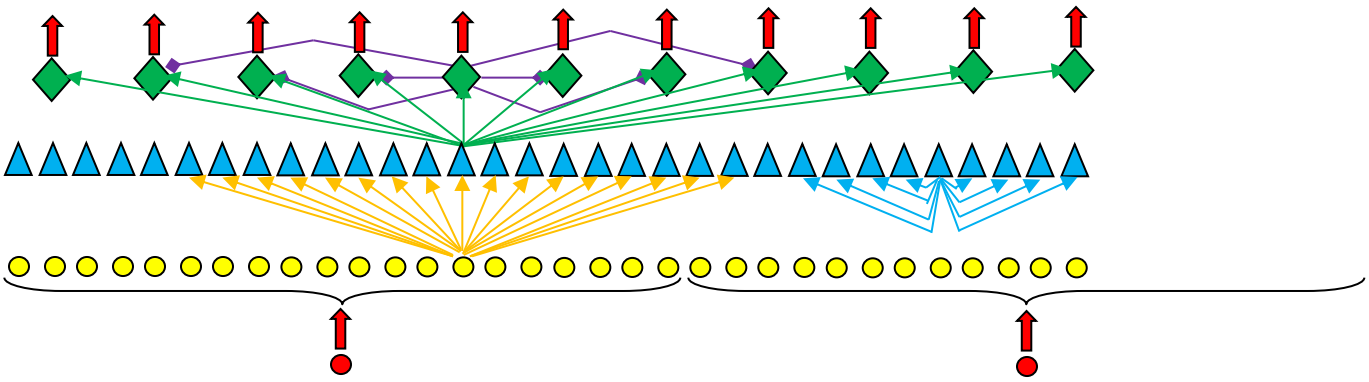


Figure 5.3. The first phase of the modification: no inhibitory synapses between layers 3 and 2.

The goal of the second stage of the modification is to introduce the possibility of competition among different groups of the output nodes. There are 67 output nodes in layer 3 and they are divided into 10 groups: 9 groups, each containing 7 consecutive output nodes, and one more group, which contains (the last remaining) 4 consecutive output nodes (visually, it is located on the right side of the graph). The output nodes in each group are located sequentially and the groups themselves are located sequentially as well. Visually, the sequence is read from the left to the right.

Each group of layer 3 corresponds to an input node of layer 0. Recall, there are 10 input nodes in layer 0, which is equal to the number of output groups. The order of the corresponding output groups and input nodes is the same with 1-to-1 correlation between them. The purpose of each output group is to activate its corresponding input node and deactivate the rest of the input nodes. Hence, each output node is connected by an excitatory synapse to the input node, which corresponds to the group of the output node. The synaptic weight of this synapse is set to 1, which reflects transmitting unchanged signals. In addition, each output node is connected

by inhibitory synapses to the rest of the input nodes. The synaptic weight of the synapses is set to -1, which means that each of them is aimed to neutralize a single incoming excitatory signal.

It is hard to draw a picture containing all the additional synapses with complexity of 10 input nodes and 10 output groups. For this reason, Fig. 5.4 depicts a network of lower complexity, specifically the one containing 3 input nodes and 3 output groups, each group containing 3 output nodes. The network component presented in Fig. 5.3 is omitted in Fig. 5.4 in order to make the plot readable. However, the intention is that the entire network is present in the figure. The notation remains the same: red circles designate input nodes and green diamonds designate output nodes.

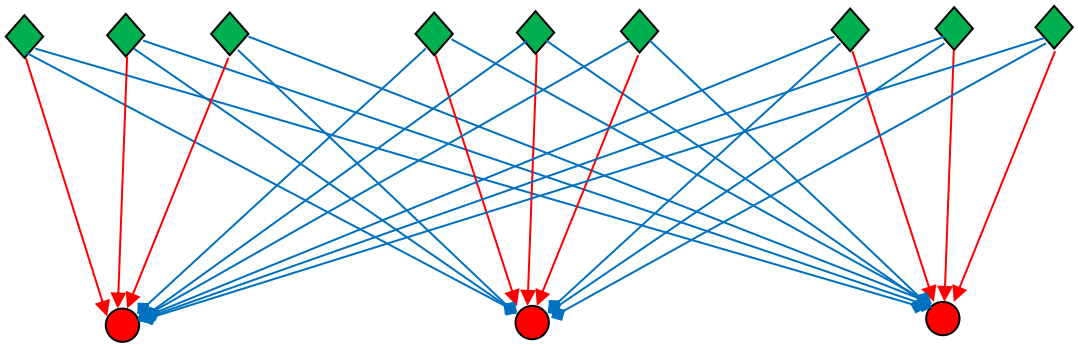


Figure 5.4. The second phase of the modification: additional feedback excitatory synapses are presented by red arrows and additional feedback inhibitory synapses are presented by blue arrows.

The competition among the output groups, which was mentioned earlier, is possible in the modified feedback network, since domination of the signal coming from a particular output group will necessarily suppress signals generated by all the input nodes, except for the input node corresponding to the dominating output group. This input node's signal will propagate through the network and further reinforce the dominating output group signal. More formally, the majority of all possible directed paths, starting from input node *i* and having three directed edges, will end up in the output nodes belonging to the output group, which corresponds to the input node *i*.

This topology creates a huge potential for the interactions among different input signals. In the context of the fitness function, defined for CISGA, the output signal

domination of a certain output group can give a great advantage in fitness. Recall, the fitness is defined as input linearity level, which in turn reflects the level of dissimilarity between the superposition of the separate output signals and the overall output signal produced by the input (see first 6 paragraphs in section 5.4). Domination of a certain output group and its corresponding input node will interfere with the signals generated by the rest of the input nodes and, therefore, change considerably their separate output signals. As a result, the superposition of the separate output signals will not be similar to the generated output signal.

Application of the evolutionary technique implemented in CISGA to the modified feedback network makes it possible to evolve considerably less linear inputs than the pulse shaped inputs described previously. These non-linear inputs comprise the fifth test set used for the comparison of the developed algorithms. The comparison of the input linearity levels for different test sets, as well as the performance of the developed algorithms, is presented in the next chapter.

Chapter 6:

Simulation results

In the previous chapters we introduced the four algorithms, which are compared in the present chapter. The algorithms include URS as the reference group and the GA-based algorithms: SGA, FIGA and FGA. There are five test sets used for the comparison: four test sets are testing the auditory cortex model network and the fifth test set is testing the modified feedback network. The first three test sets are generated with pulse shaped input sequences having constant frequencies equal to 3, 6 and 12. To generate the forth test set we invoke the CISGA algorithm on the auditory cortex model network. For the fifth test set we invoke the CISGA algorithm on the modified feedback network in order to generate the inputs.

6.1 Collected statistics overview

As it was mentioned earlier, each test set consists of 100 tests grouped by 10 tests per each number of participating input nodes, which are used to generate the target output signals. The statistics to be calculated are based on 100 generated results per each tested algorithm (out of the four algorithms) per each test set (out of the five test sets). There are two types of statistics collected in each test: (i) the number of bitwise mismatches between the target output sequences and the output sequences, generated by the best fit solution of an algorithm; (ii) the number of participating input node mismatches between the input, which corresponds to the target output, and the input of the best fit solution of an algorithm. The latter statistic shows the precision of finding the originally used input nodes.

Each of the two statistics yields the mean and standard deviation value per each group of 10 tests, corresponding to a particular number of participating input nodes in the original inputs of the test group. Unfortunately, we could not perform more tests in order to collect more data to increase the reliability of the statistics due to the extremely long runtime of the entire test sets (see Table 6.1).

Test Set	Total runtime of the entire set (sec)
Auditory cortex with pulse input, freq.= 12	432,241.6 (=120 hours or 5 days)
Auditory cortex with pulse input, freq.= 6	536,860.3 (=150 hours or 6.2 days)
Auditory cortex with pulse input, freq.= 3	562,450.1 (=156 hours or 6.5 days)
Auditory cortex with CISGA generated input	528,523.5 (=147 hours or 6.1 days)
Feedback network with CISGA generated input	1,293,799 (=359 hours or 15 days)

Table 6.1 Runtimes of the test sets

An additional statistic that was collected and analysed for each test is the input linearity level of the original input of the test. Following the same approach used with the first two statistics, the mean and standard deviation values were calculated for the input linearity levels of each 10 grouped tests (for the definition of input linearity level see section 5.4). All the five test sets were analysed using this statistic in order to show the effect of introducing CISGA and the modified feedback network on the linearity levels of generated inputs.

The following subsections present the resulting plots and the data for each of the test sets. We will start from the comparison of input linearity levels among the test sets.

6.2 Input linearity levels

Before presenting the results of the algorithms' comparison, we would like to show the input linearity levels of the test sets generated for the comparison. The linearity levels are depicted in Fig. 6.1 with the exact values of mean and standard deviation presented in Tables 6.2 and 6.3.

The X axis of both plots in the figure represents the number of participating input nodes used to generate the target output, corresponding to the related row in one of the tables. Each integer point on X axis (between 1 and 10) aggregates the results of 10 tests for each of the test sets, with the results listed in the corresponding rows of the tables. It is important to recall that lower input linearity values correspond to higher input linearity levels and vice versa. The information presented in the figure is separated into two plots, (a) and (b), for easier visual distinction, since the input

linearity values of the fifth test set range is over a much greater scale compared to the scale of the rest four of the test sets.

The first plot (a) shows considerably higher input linearity for the auditory cortex with pulse input frequency 12 (named Original Freq=12), when compared to the rest of examined cases (lines named Original Freq=6, Original Freq=3 and Original Complex). In addition, using CISGA to generate input sequences (Original Complex) does not introduce any significant difference in the input linearity level compared to the inputs with frequency 6 (Original Freq=6). The overall non-linearity increase is extremely low for all the graphs (less than 2.25% between 1 and 10 participating input nodes). This result is expected, since the original auditory cortex model network, used in these test sets, does not have positive feedback loops, which could reinforce or dramatically change the input signals. The only feedback is negative, therefore suppressive, and it is from layer 3 to layer 2 (see Fig.5.1 in section 5.2). We expected it to only slightly modify input signals, and it is also expected from the auditory system to filter (supress) some noise, but not to change the external signal significantly.

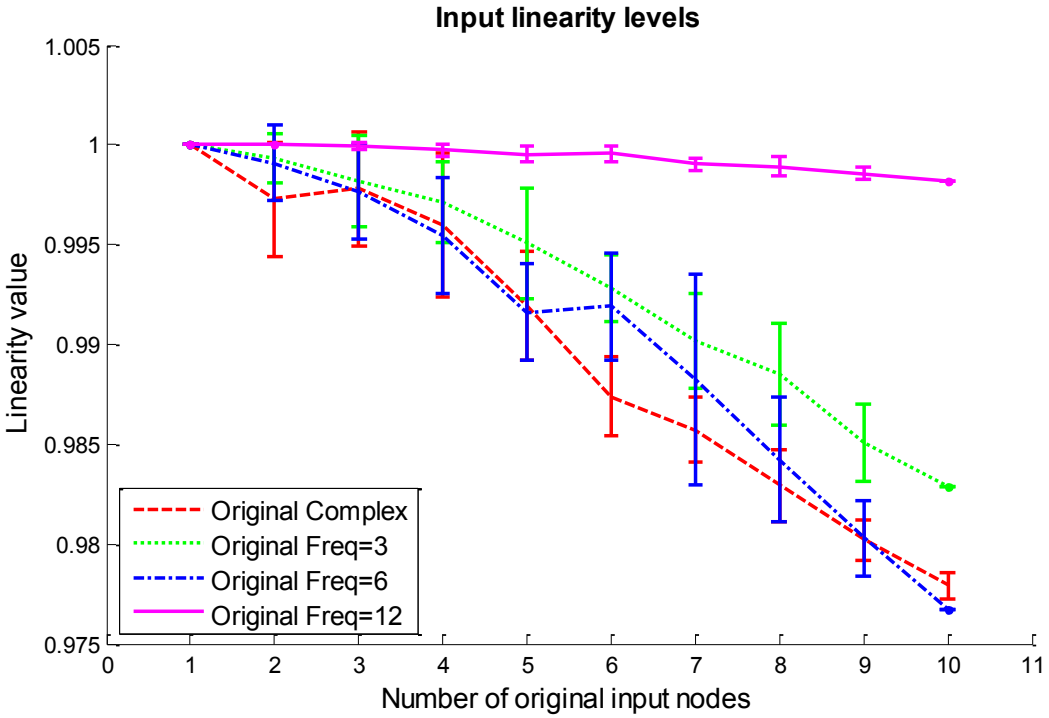
However, the second plot (b) presents a completely different picture. When CISGA is run on the modified feedback network, input linearity values increase at almost 50% between 1 and 10 participating input nodes. This behaviour was anticipated, since the modified feedback network was developed in order to enable changing input signals to a much greater extent, when compared to the original auditory cortex model network (see section 5.7). Introduction of multiple positive and negative feedbacks into the network, directed from layer 3 back to layer 0, was aimed at creating such a topology, where different groups of input neurons would be trying to supress all other groups of input neurons. This “competitiveness” of the network’s topology is the main reason for high non-linearity of inputs generated by the CISGA.

Number of original input nodes	Original Freq=12 input linearity values: mean ; standard deviation	Original Freq=6 input linearity values: mean ; standard deviation	Original Freq=3 input linearity values: mean ; standard deviation	Original Complex input linearity values: mean ; standard deviation
1	1.0000 ; 0.0000	1.0000 ; 0.0000	1.0000 ; 0.0000	1.0000 ; 0. 0000
2	1.0000 ; 0.0000	0.9991 ; 0.0019	0.9994 ; 0.0012	0.9973 ; 0. 0029
3	0.9999 ; 0.0002	0.9976 ; 0.0024	0.9982 ; 0.0023	0.9978 ; 0. 0028
4	0.9998 ; 0.0003	0.9954 ; 0.0029	0.9972 ; 0.0020	0.9960 ; 0. 0036
5	0.9995 ; 0.0004	0.9916 ; 0.0024	0.9951 ; 0.0028	0.9919 ; 0. 0027
6	0.9996 ; 0.0004	0.9919 ; 0.0027	0.9928 ; 0.0017	0.9874 ; 0. 0020
7	0.9990 ; 0.0003	0.9882 ; 0.0053	0.9902 ; 0.0023	0.9857 ; 0. 0016
8	0.9989 ; 0.0005	0.9842 ; 0.0031	0.9885 ; 0.0026	0.9829 ; 0. 0018
9	0.9986 ; 0.0003	0.9803 ; 0.0019	0.9851 ; 0.0020	0.9802 ; 0. 0010
10	0.9982 ; 1.17e-16	0.9767 ; 0.0000	0.9828 ; 0.0000	0.9779 ; 0. 0007

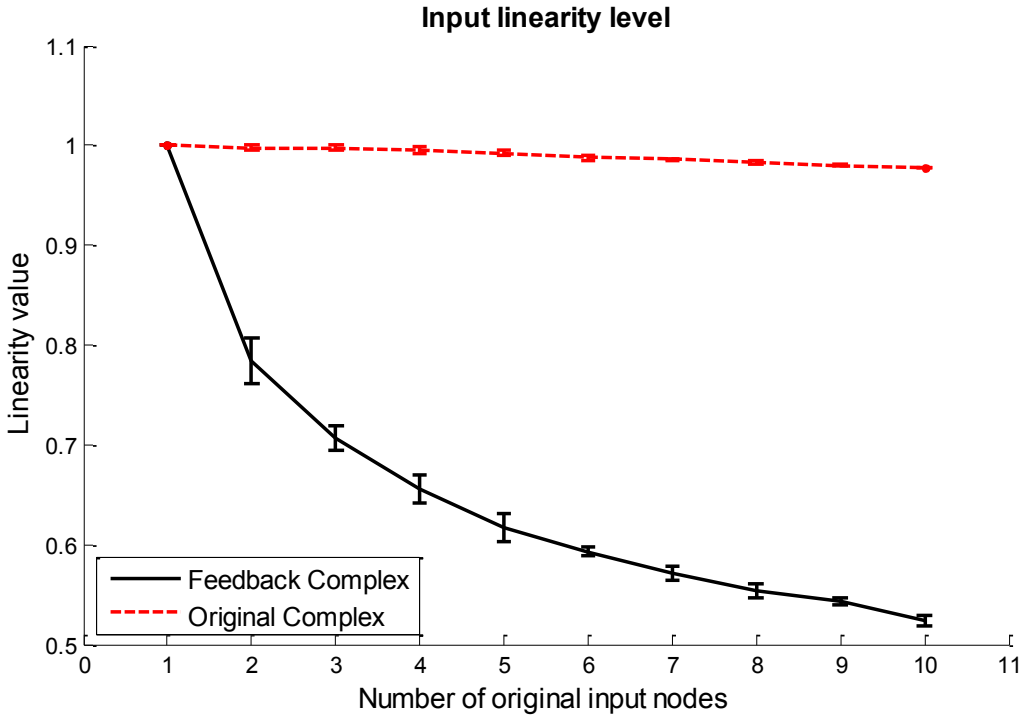
Table 6.2 Mean and Standard Deviation values for input linearity of theauditory cortex network for pulse shaped input with frequencies 12, 6, 3 and CISGA generated input (columns from left to right)

Number of original input nodes	Original Complex input linearity values: mean ; standard deviation	Feedback Complex input linearity values: mean ; standard deviation
1	1.0000 ; 0. 0000	1.0000 ; 0.0000
2	0.9973 ; 0. 0029	0.7844 ; 0.0222
3	0.9978 ; 0. 0028	0.7064 ; 0.0127
4	0.9960 ; 0. 0036	0.6555 ; 0.0146
5	0.9919 ; 0. 0027	0.6171 ; 0.0139
6	0.9874 ; 0. 0020	0.5929 ; 0.0045
7	0.9857 ; 0. 0016	0.5705 ; 0.0071
8	0.9829 ; 0. 0018	0.5538 ; 0.0065
9	0.9802 ; 0. 0010	0.5430 ; 0.0038
10	0.9779 ; 0. 0007	0.5242 ; 0.0049

Table 6.3 Mean and Standard Deviation values for input linearity levels of CISGA generated input in the auditory cortex and the modified feedback network (Original Complex and Feedback Complex, respectively)


Input linearity levels
Linearity value
Number of original input nodes
Original Complex
Original Freq=3
Original Freq=6
Original Freq=12
1.005
1
0.995
0.99
0.985
0.98
0.975
0
1
2
3
4
5
6
7
8
9
10
11

(a)


Input linearity level
Linearity value
Number of original input nodes
Feedback Complex
Original Complex
1.1
1
0.9
0.8
0.7
0.6
0.5
0
1
2
3
4
5
6
7
8
9
10
11

(b)

Figure 6.1.(a,b) Input linearity levels. The top plot depicts the input linearity level of the first four test sets (auditory cortex with the pulse input having frequencies 3,6,12, denoted as Original Freq=3, Original Freq=6, Original Freq=12, respectively, and auditory cortex with the input generated by CISGA, denoted as Original Complex). The plot at the bottom depicts the input linearity level of auditory cortex versus modified feedback network, both having the input generated by CISGA (denoted as Original Complex and Feedback Complex, respectively).

Please notice that that confidence intervals depicted in Fig.6.1 correspond to ± 1 standard deviation, which has a confidence level of 68% in normal distribution. All further plots in the present work correspond to this confidence level.

Almost constant input linearity with the values close to 1 for the pulse input with frequency 12 indicates a low level of network signal interactions for this frequency of the signal. The observation is supported by additional evidence in the next section. Another important observation is that inputs with frequency 3 are slightly more linear than inputs with frequency 6 (lines Original Freq=3 and Original Freq=6, respectively). This difference is not as noticeable as the difference between frequencies 3,6 and frequency 12 and can be explained by a higher degree of interactions of signals having frequency 6. Signals having frequency 3 are “firing” too often in order to create a distinguishable change as more input nodes are added, therefore, these signals are able to increase input non-linearity to a smaller extent.

According to the fact that input linearity levels are similar for inputs with frequencies 3, 6 and for CISGA generated inputs in the auditory cortex, we expect the performance of the developed algorithms to be similar for those three test sets. We expect the sharp deterioration of performance for the fifth test set is due to the higher input non-linearity achieved by CISGA in the modified feedback network.

6.3 Auditory cortex and pulse input with frequency equal to 12

This test set stands apart from all the rest of the test sets, since it presents the results, which are completely different from the rest. Let us examine the two plots depicted in Fig. 6.2 and the data in Tables 6.4 and 6.5.

The first statistic is the number of output bit mismatches. Recall that this is the number of bitwise mismatches between the externally applied input signal sequences and the input signal sequences predicted by the developed algorithms (URS, SGA, FIGA, FGA).The statistic does not show a big difference among the four compared algorithms, except for the slightly better performance of SGA, FIGA and FGA in comparison to URS. Almost constant gap between the values computed for URS and the values computed for the rest three algorithms is observed for any number of original input nodes.

The second statistic is the number of input node mismatches. Recall that this is the number of incorrectly predicted participating input nodes, or the size of the difference set between the originally selected participating input nodes and the participating input nodes predicted by the developed algorithms. The statistic shows the same performance for URS and SGA, and slightly better performance for FGA and FIGA. In fact, FIGA performs the best and this is the only result that could be intuitively expected for this simple input, since only FIGA uses separate input nodes' assessment.

The reason for such unsatisfactory result lies in the input signal, generated for the test set. A very small amount of original input signals actually reach the output nodes. The signals decay and do not reach the output nodes. The root cause of such decay is the local potential values defined for the simulation.

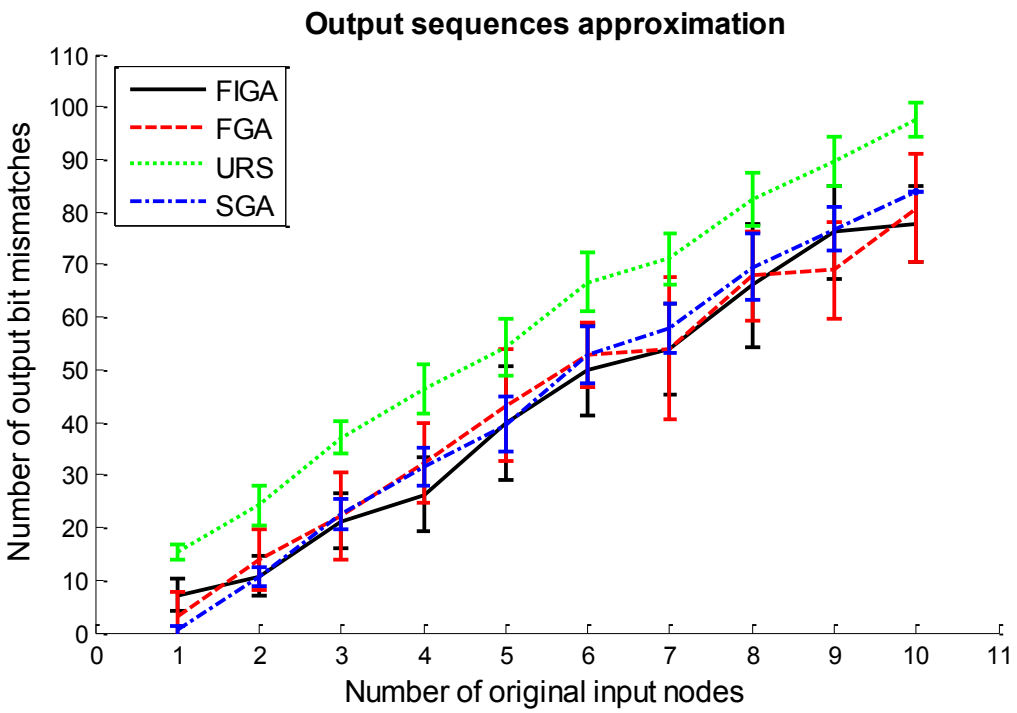
Number of original input nodes	URS output bit mismatches: mean ; standard deviation values	SGA output bit mismatches: mean ; standard deviation values	FIGA output bit mismatches: mean ; standard deviation values	FGA output bit mismatches: mean ; standard deviation values
1	15.30 ; 1.34	0.40 ; 0.84	7.10 ; 3.07	3.10 ; 4.58
2	24.20 ; 3.74	10.60 ; 1.90	10.80 ; 3.71	13.90 ; 5.63
3	37.10 ; 3.00	22.60 ; 2.99	21.20 ; 5.25	22.10 ; 8.33
4	46.40 ; 4.72	31.60 ; 3.50	26.30 ; 6.99	32.20 ; 7.48
5	54.30 ; 5.50	39.60 ; 5.15	39.90 ; 10.80	43.20 ; 10.72
6	66.70 ; 5.60	52.80 ; 5.27	50.00 ; 8.71	52.90 ; 6.24
7	71.10 ; 4.82	58.00 ; 4.71	53.80 ; 8.70	54.00 ; 13.47
8	82.40 ; 5.02	69.60 ; 6.31	66.10 ; 11.78	67.90 ; 8.45

9	89.70 ; 4.67	76.80 ; 4.13	76.10 ; 8.75	68.90 ; 9.23
10	97.50 ; 3.17	84.00 ; 0.00	77.70 ; 7.06	80.70 ; 10.30

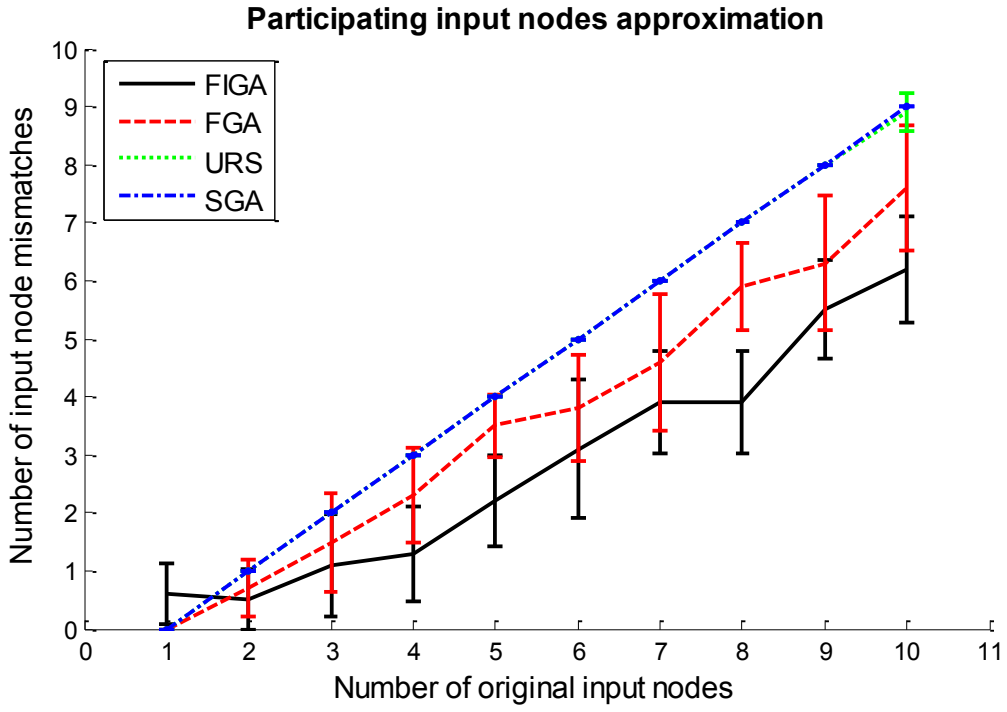
Table 6.4 Mean and Standard Deviation values for the number of output bit mismatches for the auditory cortex network with pulse shaped input having frequency 12

Number of original input nodes	URS input node mismatches: mean ; standard deviation values	SGA input node mismatches: mean ; standard deviation values	FIGA input node mismatches: mean ; standard deviation values	FGA input node mismatches: mean ; standard deviation values
1	0.0000 ; 0.0000	0.0000 ; 0.0000	0.6000 ; 0.5164	0.0000 ; 0.0000
2	1.0000 ; 0.0000	1.0000 ; 0.0000	0.5000 ; 0.5270	0.7000 ; 0.4830
3	2.0000 ; 0.0000	2.0000 ; 0.0000	1.1000 ; 0.8756	1.5000 ; 0.8498
4	3.0000 ; 0.0000	3.0000 ; 0.0000	1.3000 ; 0.8233	2.3000 ; 0.8233
5	4.0000 ; 0.0000	4.0000 ; 0.0000	2.2000 ; 0.7888	3.5000 ; 0.5270
6	5.0000 ; 0.0000	5.0000 ; 0.0000	3.1000 ; 1.1972	3.8000 ; 0.9189
7	6.0000 ; 0.0000	6.0000 ; 0.0000	3.9000 ; 0.8756	4.6000 ; 1.1738
8	7.0000 ; 0.0000	7.0000 ; 0.0000	3.9000 ; 0.8756	5.9000 ; 0.7379
9	8.0000 ; 0.0000	8.0000 ; 0.0000	5.5000 ; 0.8498	6.3000 ; 1.1595
10	8.9000 ; 0.3162	9.0000 ; 0.0000	6.2000 ; 0.9189	7.6000 ; 1.0750

Table 6.5 Mean and Standard Deviation values for the number of input node mismatches for the auditory cortex network with pulse shaped input having frequency 12



(a)



(b)

Figure 6.2.(a,b) Result for pulse inputs with frequency 12. The top plot depicts the number of bitwise mismatches in the output sequences of the

algorithms compared to the target output sequence (out of 6700 bits overall) and the plot at the bottom depicts the number of participating input node mismatches (out of 10 input nodes overall). X axis on the both plots represents the number of participating input nodes used to generate the target output. Each integer point on X axis (between 1 and 10) aggregates the results of 10 tests for each of the four algorithms.

Recall that local potentials have up to 15 values (1 value for each consecutive moment in time). We defined 10 values per each local potential. It means that if a local potential is generated in an edge, it will persist in it for 10 consecutive time units before decaying completely.

It turns out that a single local potential with 10 assigned values is not enough to generate spikes in the given network in most cases. In order to generate spikes it is required to have accumulating local potentials, allowing for efficient application of spatial and temporal summation. Pulse shaped input with frequency equal 12 does not allow for the accumulation of local potentials. For frequency 12, once a local potential is fired, the next local potential is fired only 12 time units later. Meanwhile, the first local potential decays in 10 time units because we defined only 10 time-phased values for it, so it disappears completely by the time the next local is fired. No accumulation of local potentials is possible in such a case.

However, there are still some signals reaching the output nodes. This can be concluded from the slightly better results of the three GA-based algorithms and, particularly, on the ability of FGA and FIGA to better identify the participating input nodes. In addition, the input linearity values presented in the previous subsection demonstrate that the linearity of this input is slightly different from 1 and, therefore, provides some weak interactions within the network.

The input signals, which do not reach output nodes, do not provide the essential information needed for GA-based algorithms to perform selection, since the algorithms can operate only on the output sequences. As a result of the observations made in this section, we expect pulse input sequences with frequencies less than 10 to have a higher impact on output sequences and hence provide better testing conditions for the presented algorithms.

The general trend, which is expected to repeat itself in the rest of the test sets, is the dependency of the algorithms' approximation precision on the number of the originally participating input nodes. The more input nodes participate in the generation of the target output sequences, the lower the precision of the algorithms is expected to be. However, it also highly depends on the non-linearity of the input.

6.4 Auditory cortex and pulse input with frequencies 6 and 3

We observe from Fig. 6.3 and Fig. 6.4, generated by using the data presented in Tables 6.6 to 6.9, respectively, that the performance of the GA-based algorithms sharply increases in comparison to URS, the performance of which sharply deteriorates, in contrast to the results provided in the previous section. This behaviour is justifiable from the fact that the non-linearity of the inputs increased due to a higher degree of interactions among the signals in the network and the higher amount of signals reaching the output nodes. This allows the GA-based algorithms to use the output signal information more efficiently. On the contrary, URS does not "know" how to use the information. Therefore, its results deteriorate considerably more.

In most cases, the approximation precision of all the algorithms deteriorates as the number of the participating input nodes increases. The worst performance among the GA-based algorithms in terms of the output bit mismatches is observed for SGA and the best for FGA. However, FIGA generally performs better in terms of input node mismatches.

It is important to notice that for Freq=6 SGA performs only slightly better than URS in the approximation of output bits, and it performs approximately the same in the prediction of participating input nodes. In the case of frequency equal 3 SGA yields higher precision and reaches the precision of FIGA for 10 participating input nodes, however, both of the algorithms are still far from the result accomplished by FGA. Indeed, we expected SGA to perform better in comparison to URS because we believe that even a simple GA is better than random search. In addition, we expected FIGA to perform better in comparison to SGA, when they are run on the

linear inputs (the original auditory cortex model network fed with pulse shaped input sequences). Due to its nature, FIGA makes a better use of separate input nodes when the network performs linearly. Finally, the results supported our expectation of a higher FGA precision in comparison to the rest of the algorithms.

Another important observation is that in terms of participating input nodes slightly higher approximation precision of FIGA against FGA does not give the former any advantage in approximating the output sequences. The observation may be explained by the insufficient number of generations spent by FIGA on the best configuration of participating input nodes. Recall that each combination of participating input nodes is assessed for only 50 generations in FIGA. In contrast, FGA spends 500 generations with the same population size (100 solutions) on the best fit combination of participating input nodes. This difference gives FGA a crucial advantage in analysing inputs with more than 5-6 input nodes (see the first plot in Fig. 6.3 and Fig.6.4).

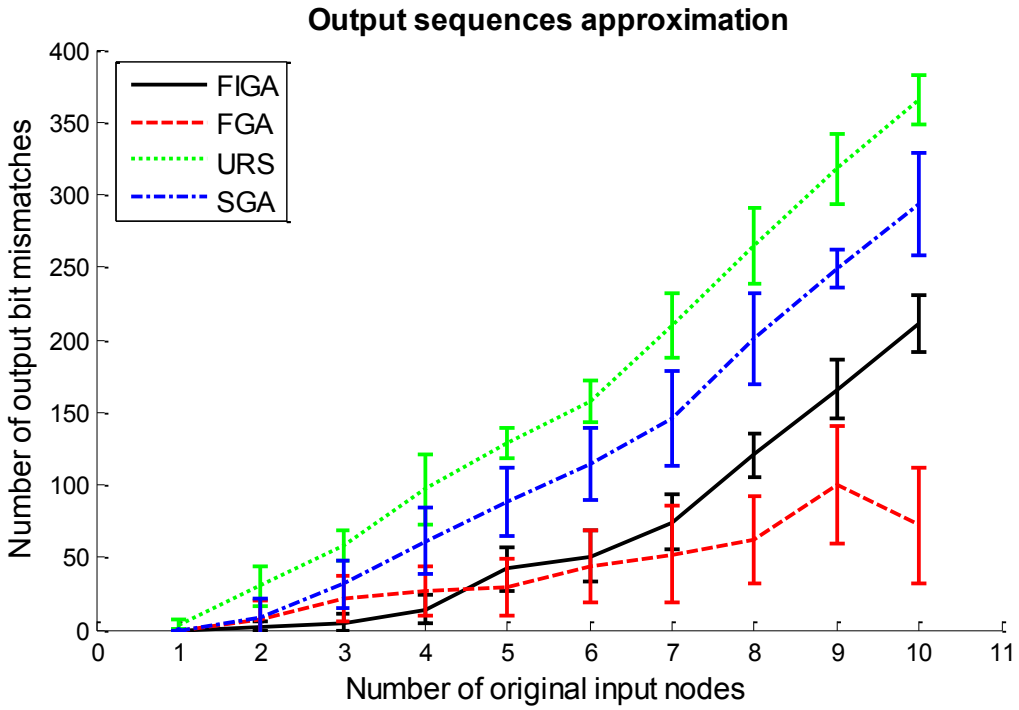
Number of original input nodes	URS output bit mismatches: mean ; standard deviation values	SGA output bit mismatches: mean ; standard deviation values	FIGA output bit mismatches: mean ; standard deviation values	FGA output bit mismatches: mean ; standard deviation values
1	3.80 ; 3.58	0.00 ; 0.00	0.00 ; 0.00	0.00 ; 0.00
2	30.20 ; 14.16	8.50 ; 12.69	2.50 ; 3.03	7.70 ; 12.94
3	58.00 ; 10.12	31.40 ; 16.80	4.90 ; 5.65	21.50 ; 16.23
4	97.20 ; 24.28	61.40 ; 22.94	14.20 ; 10.15	27.00 ; 16.86
5	129.30 ; 10.42	88.00 ; 23.52	42.40 ; 15.08	29.30 ; 19.21
6	157.60 ; 13.95	114.60 ; 25.26	50.80 ; 17.95	43.80 ; 24.69
7	210.00 ; 22.38	145.80 ; 32.09	74.00 ; 18.83	52.10 ; 32.91
8	264.50 ; 26.40	200.20 ; 31.48	120.50 ; 14.93	62.40 ; 29.73
9	318.00 ; 24.07	248.60 ; 12.87	166.00 ; 20.15	99.40 ; 40.49
10	364.90 ; 16.82	293.60 ; 35.46	210.80 ; 19.55	71.90 ; 39.61

Table 6.6 Mean and Standard Deviation values of the number of output bit mismatches for the auditory cortex network with pulse shaped input having Freq=6

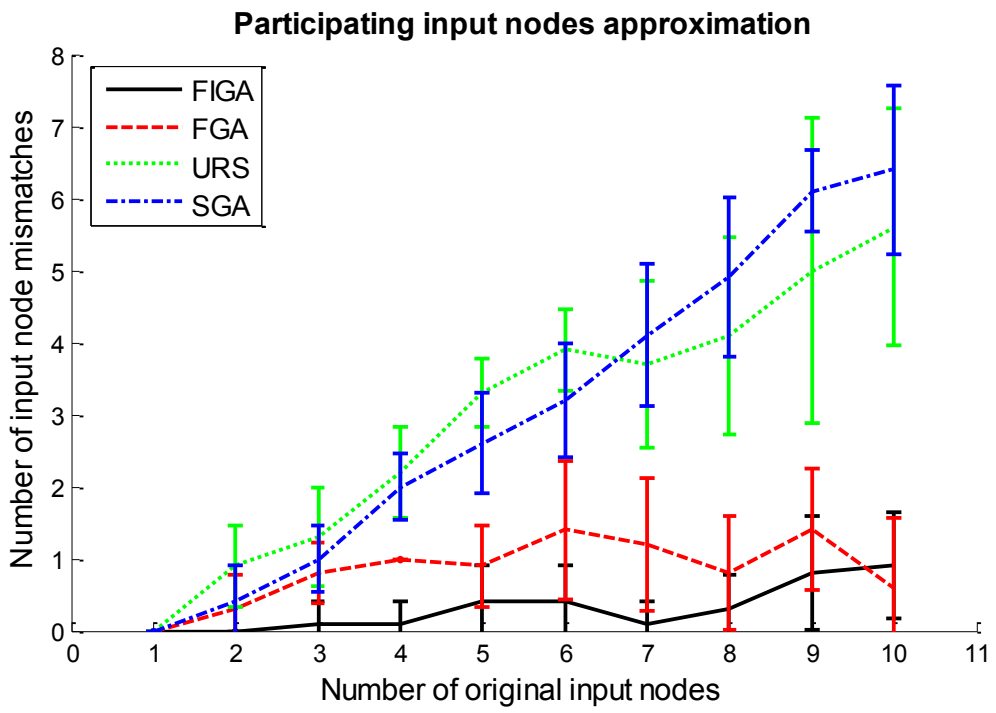
Number of original input	URS input node mismatches: mean ; standard deviation values	SGA input node mismatches: mean ;	FIGA input node mismatches: mean ; standard deviation values	FGA input node mismatches: mean ; standard deviation values

nodes		standard deviation values		
1	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000
2	0.9000 ; 0.5676	0.4000 ; 0.5164	0.0000 ; 0.0000	0.3000 ; 0.4830
3	1.3000 ; 0.6749	1.0000 ; 0.4714	0.1000 ; 0.3162	0.8000 ; 0.4216
4	2.2000 ; 0.6325	2.0000 ; 0.4714	0.1000 ; 0.3162	1.0000 ; 0.0000
5	3.3000 ; 0.4830	2.6000 ; 0.6992	0.4000 ; 0.5164	0.9000 ; 0.5676
6	3.9000 ; 0.5676	3.2000 ; 0.7888	0.4000 ; 0.5164	1.4000 ; 0.9661
7	3.7000 ; 1.1595	4.1000 ; 0.9944	0.1000 ; 0.3162	1.2000 ; 0.9189
8	4.1000 ; 1.3703	4.9000 ; 1.1005	0.3000 ; 0.4830	0.8000 ; 0.7888
9	5.0000 ; 2.1082	6.1000 ; 0.5676	0.8000 ; 0.7888	1.4000 ; 0.8433
10	5.6000 ; 1.6465	6.4000 ; 1.1738	0.9000 ; 0.7379	0.6000 ; 0.9661

Table 6.7 Mean and Standard Deviation values of the number of input node mismatches for the auditory cortex network with pulse shaped input having Freq=6



(a)



(b)

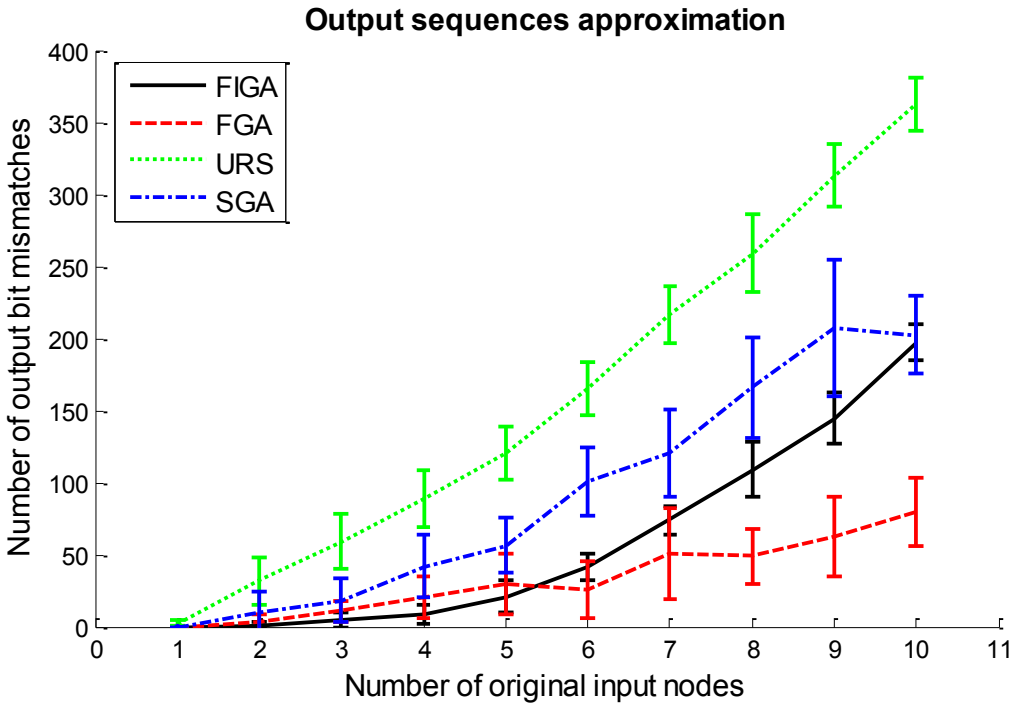
Figure 6.3(a,b). Results for pulse shaped input with frequency 6. Confidence level 68.3%

Number of original input nodes	URS output bit mismatches: mean ; standard deviation values	SGA output bit mismatches: mean ; standard deviation values	FIGA output bit mismatches: mean ; standard deviation values	FGA output bit mismatches: mean ; standard deviation values
1	2.60 ; 2.22	0.00 ; 0.00	0.00 ; 0.00	0.00 ; 0.00
2	31.60 ; 16.11	9.30 ; 15.42	1.00 ; 1.76	3.10 ; 4.82
3	58.80 ; 18.95	18.40 ; 15.12	4.40 ; 5.06	10.60 ; 7.26
4	88.50 ; 19.60	42.00 ; 21.96	8.40 ; 6.75	20.60 ; 14.36
5	120.70 ; 18.11	56.30 ; 19.10	21.00 ; 11.02	30.20 ; 21.02
6	165.70 ; 18.40	100.60 ; 23.25	41.90 ; 9.37	25.70 ; 19.40
7	216.40 ; 19.97	120.60 ; 30.17	73.70 ; 9.64	50.60 ; 31.75
8	259.00 ; 26.89	165.80 ; 35.01	108.80 ; 19.20	48.80 ; 18.72
9	313.10 ; 21.97	206.70 ; 47.42	144.70 ; 18.29	62.30 ; 27.78
10	362.60 ; 18.61	202.30 ; 26.85	197.00 ; 12.68	79.90 ; 24.00

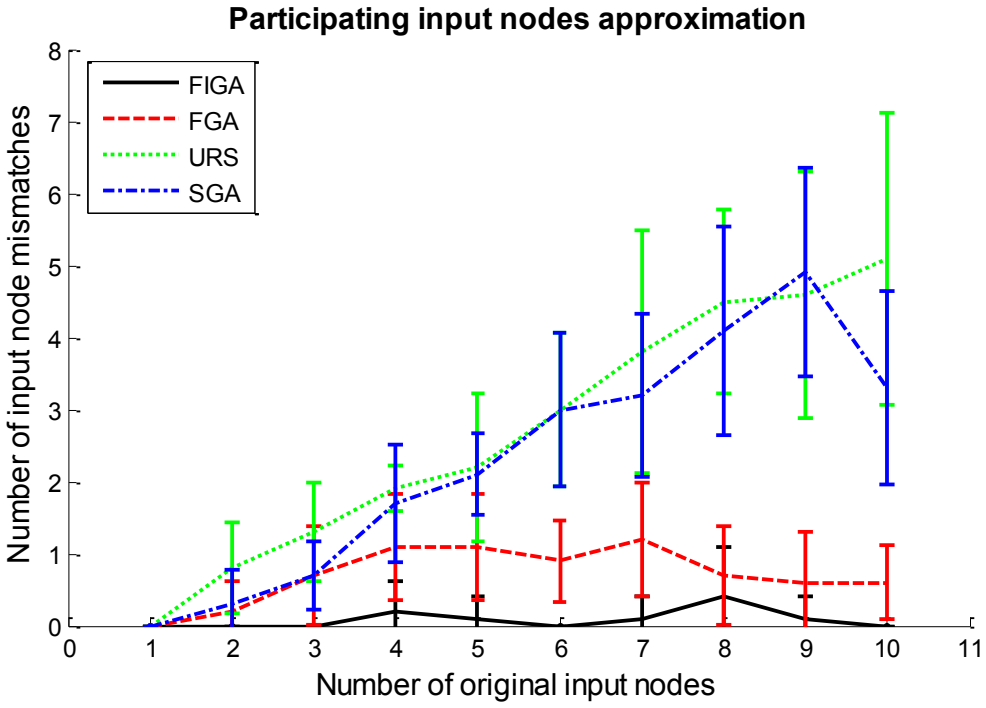
Table 6.8 Mean and Standard Deviation values of the number of output bit mismatches for the auditory cortex network with pulse shaped input having Freq=3

Number of original input nodes	URS input node mismatches: mean ; standard deviation values	SGA input node mismatches: mean ; standard deviation values	FIGA input node mismatches: mean ; standard deviation values	FGA input node mismatches: mean ; standard deviation values
1	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000
2	0.8000 ; 0.6325	0.3000 ; 0.4830	0.0000 ; 0.0000	0.2000 ; 0.4216
3	1.3000 ; 0.6749	0.7000 ; 0.4830	0.0000 ; 0.0000	0.7000 ; 0.6749
4	1.9000 ; 0.3162	1.7000 ; 0.8233	0.2000 ; 0.4216	1.1000 ; 0.7379
5	2.2000 ; 1.0328	2.1000 ; 0.5676	0.1000 ; 0.3162	1.1000 ; 0.7379
6	3.0000 ; 1.0541	3.0000 ; 1.0541	0.0000 ; 0.0000	0.9000 ; 0.5676
7	3.8000 ; 1.6865	3.2000 ; 1.1353	0.1000 ; 0.3162	1.2000 ; 0.7888
8	4.5000 ; 1.2693	4.1000 ; 1.4491	0.4000 ; 0.6992	0.7000 ; 0.6749
9	4.6000 ; 1.7127	4.9000 ; 1.4491	0.1000 ; 0.3162	0.6000 ; 0.6992
10	5.1000 ; 2.0248	3.3000 ; 1.3375	0.0000 ; 0.0000	0.6000 ; 0.5164

Table 6.9 Mean and Standard Deviation values of the number of input node mismatches for the auditory cortex network with pulse shaped input having Freq=3



(a)



(b)

Figure 6.4. The results for the auditory cortex with pulse shaped input, frequency Freq=3. Confidence level 68.3%

6.5 Auditory cortex and CISGA generated input

The results presented below, in Fig. 6.5 and Tables 6.10 and 6.11, for the inputs generated by CISGA are similar to those discussed in the previous subsection.

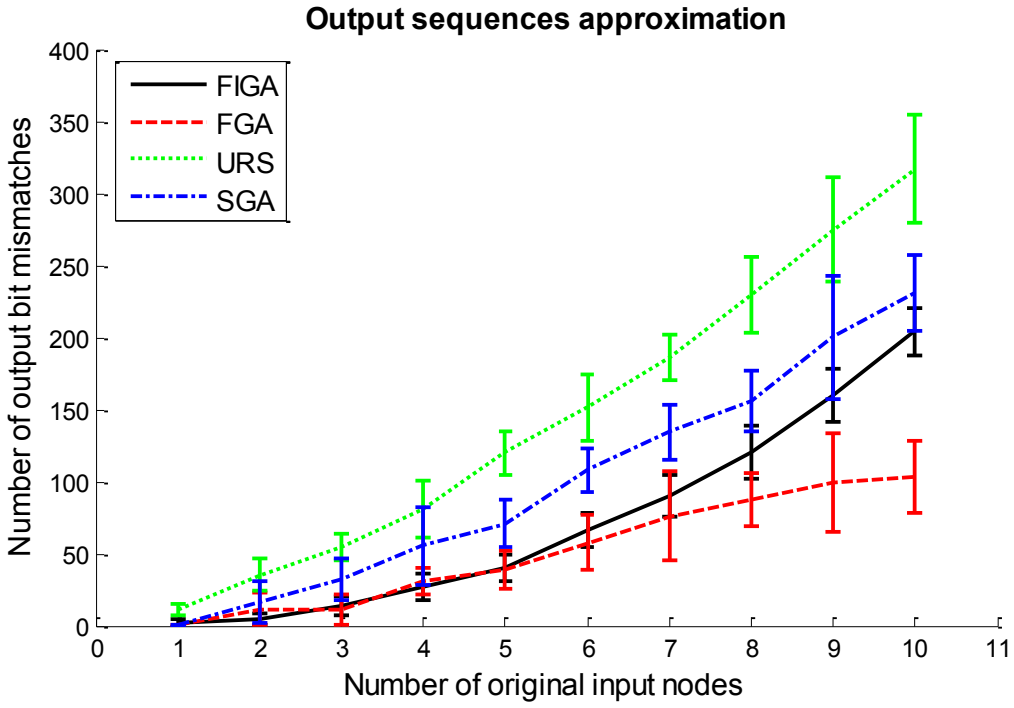
Number of original input nodes	URS output bit mismatches: mean ; standard deviation values	SGA output bit mismatches: mean ; standard deviation values	FIGA output bit mismatches: mean ; standard deviation values	FGA output bit mismatches: mean ; standard deviation values
1	10.70 ; 3.95	0.30 ; 0.95	1.70 ; 2.63	0.20 ; 0.63
2	35.40 ; 10.91	16.00 ; 14.52	4.30 ; 4.00	11.60 ; 10.82
3	54.20 ; 9.04	32.00 ; 14.82	13.40 ; 6.74	11.30 ; 10.12
4	81.50 ; 19.72	55.50 ; 27.06	26.70 ; 8.92	31.30 ; 9.37
5	119.80 ; 14.97	70.90 ; 16.49	40.10 ; 9.47	38.90 ; 13.48
6	151.60 ; 22.81	108.30 ; 15.10	65.80 ; 11.84	57.80 ; 18.99
7	186.20 ; 16.07	134.60 ; 19.26	90.20 ; 14.50	76.30 ; 30.66

8	229.00 ; 26.27	155.90 ; 20.55	120.50 ; 18.49	87.40 ; 18.09
9	274.60 ; 36.09	200.10 ; 42.96	159.80 ; 18.10	98.90 ; 34.16
10	317.10 ; 37.70	231.20 ; 26.15	204.00 ; 16.35	102.80 ; 24.85

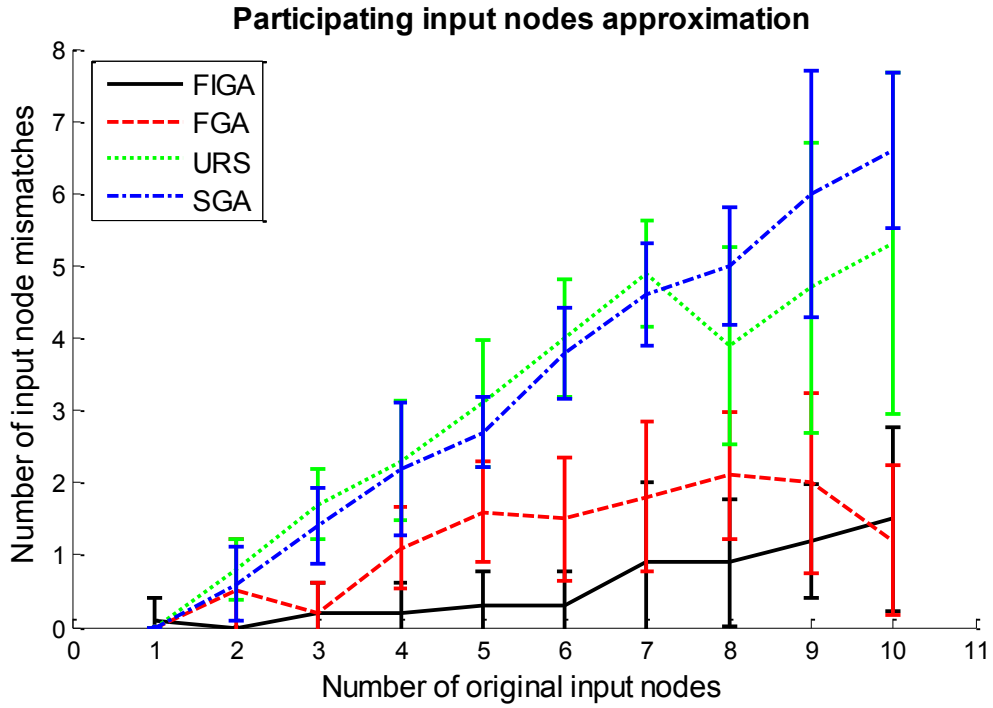
Table 6.10 Mean and Standard Deviation values of the number of output bit mismatches for the auditory cortex network with CISGA generated input.

Number of original input nodes	URS input node mismatches: mean ; standard deviation values	SGA input node mismatches: mean ; standard deviation values	FIGA input node mismatches: mean ; standard deviation values	FGA input node mismatches: mean ; standard deviation values
1	0.0000 ; 0.0000	0.0000 ; 0.0000	0.1000 ; 0.3162	0.0000 ; 0.0000
2	0.8000 ; 0.4216	0.6000 ; 0.5164	0.0000 ; 0.0000	0.5000 ; 0.7071
3	1.7000 ; 0.4830	1.4000 ; 0.5164	0.2000 ; 0.4216	0.2000 ; 0.4216
4	2.3000 ; 0.8233	2.2000 ; 0.9189	0.2000 ; 0.4216	1.1000 ; 0.5676
5	3.1000 ; 0.8756	2.7000 ; 0.4830	0.3000 ; 0.4830	1.6000 ; 0.6992
6	4.0000 ; 0.8165	3.8000 ; 0.6325	0.3000 ; 0.4830	1.5000 ; 0.8498
7	4.9000 ; 0.7379	4.6000 ; 0.6992	0.9000 ; 1.1005	1.8000 ; 1.0328
8	3.9000 ; 1.3703	5.0000 ; 0.8165	0.9000 ; 0.8756	2.1000 ; 0.8756
9	4.7000 ; 2.0028	6.0000 ; 1.6997	1.2000 ; 0.7888	2.0000 ; 1.2472
10	5.3000 ; 2.3594	6.6000 ; 1.0750	1.5000 ; 1.2693	1.2000 ; 1.0328

Table 6.11 Mean and Standard Deviation values of the number of input node mismatches for the auditory cortex network with CISGA generated input



(a)



(b)

Figure 6.5 (a,b). The results for the auditory cortex with CISGA generated inputs. Confidence level 68.3%

6.6 Modified feedback network and CISGA generated input

There are several critical changes in prediction performance of the algorithms that appear in this test set, depicted in Fig. 6.6 and presented in Tables 6.12 and 6.13, in comparison to the sets presented previously.

First, the number of bitwise mismatches grew approximately an order of magnitude for all four algorithms, compared to those reported in the previously presented results. This can be explained by the double effect of the feedback topology of the network and the non-linear input generated by CISGA using this topology. Since increasing input non-linearity was the reason for introducing both CISGA and the modified feedback network, the observed result is expected.

Second, SGA and FGA show similar performance in terms of the number of bitwise mismatches with small differences. Moreover, FIGA performs worse than SGA in this test set, the opposite behaviour from the one observed in all cases of the previously presented results. This can be explained by the excessive dependence of FIGA on the initial assessment of participating input nodes and, as it was mentioned in the previous subsection, insufficient number of generations spent by FIGA on the chosen set of participating input nodes. FGA is not prone to this pitfall, since its initial assessment takes into consideration varying subsets of participating input nodes and not only separate input nodes.

Third, in terms of the number of mismatches of participating input nodes, it is impossible to distinguish which algorithm performs better, since the results are mixed and cannot serve as a basis to any conclusion. Additionally, FIGA is no longer the best regarding this statistic. The strategy of CISGA can be seen as competing with the strategy of FIGA. That is why in the context of the modified feedback network we can observe such a change. In the context of the auditory cortex CISGA could not reach more than 2.5% increase in input non-linearity due to the topological properties, including lack of positive feedback loops between the layers of the network. URS is not distinguishable from the rest three algorithms in the number of mismatches occurring to participating input nodes. Overlapping error

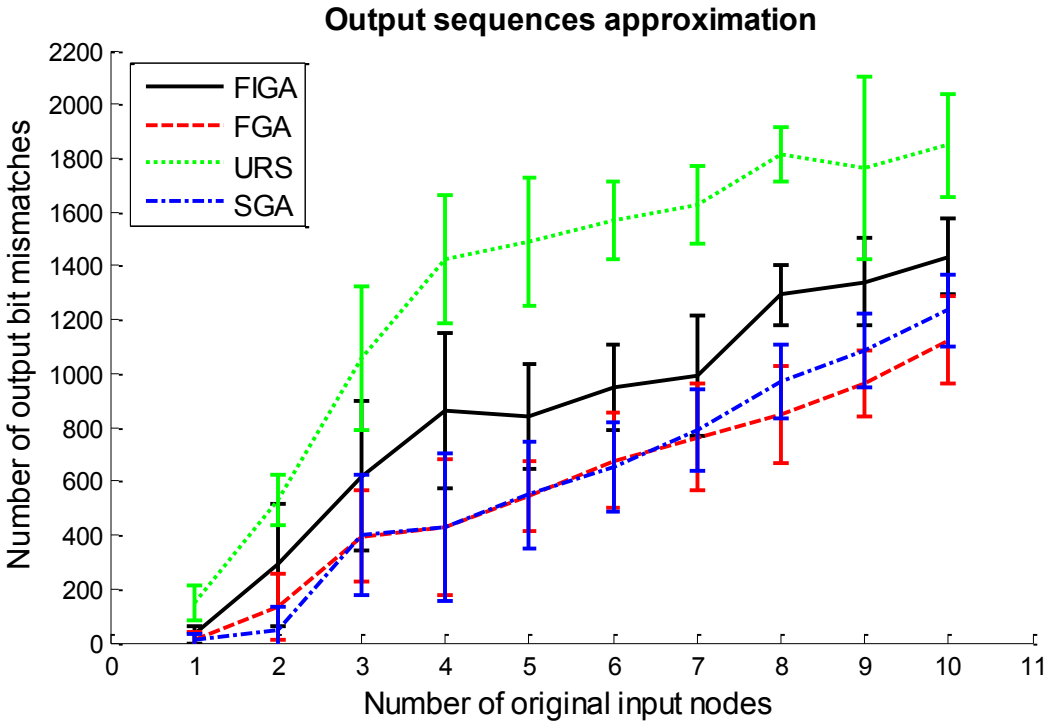
bars indicate mostly the increased difficulty of the algorithms in finding the originally participating input nodes.

Number of original input nodes	URS output bit mismatches: mean ; standard deviation values	SGA output bit mismatches: mean ; standard deviation values	FIGA output bit mismatches: mean ; standard deviation values	FGA output bit mismatches: mean ; standard deviation values
1	149.40 ; 65.65	8.20 ; 25.93	29.70 ; 31.39	13.70 ; 29.57
2	528.80 ; 94.11	47.60 ; 87.95	290.10 ; 227.13	134.30 ; 122.93
3	1054.60 ; 266.67	402.80 ; 224.58	620.30 ; 277.92	395.80 ; 169.75
4	1425.30 ; 236.18	430.60 ; 275.10	861.80 ; 290.11	430.40 ; 253.70
5	1488.90 ; 236.54	549.80 ; 196.47	842.20 ; 193.27	545.60 ; 128.53
6	1567.80 ; 143.38	654.50 ; 164.53	945.70 ; 158.51	676.40 ; 178.69
7	1627.70 ; 142.12	787.80 ; 149.91	990.40 ; 224.76	761.50 ; 197.93
8	1815.80 ; 101.33	967.10 ; 136.74	1291.90 ; 111.91	848.10 ; 183.13
9	1766.10 ; 338.07	1083.50 ; 137.53	1340.40 ; 160.19	963.20 ; 120.30
10	1847.70 ; 192.57	1233.70 ; 136.06	1434.60 ; 143.33	1124.90 ; 160.00

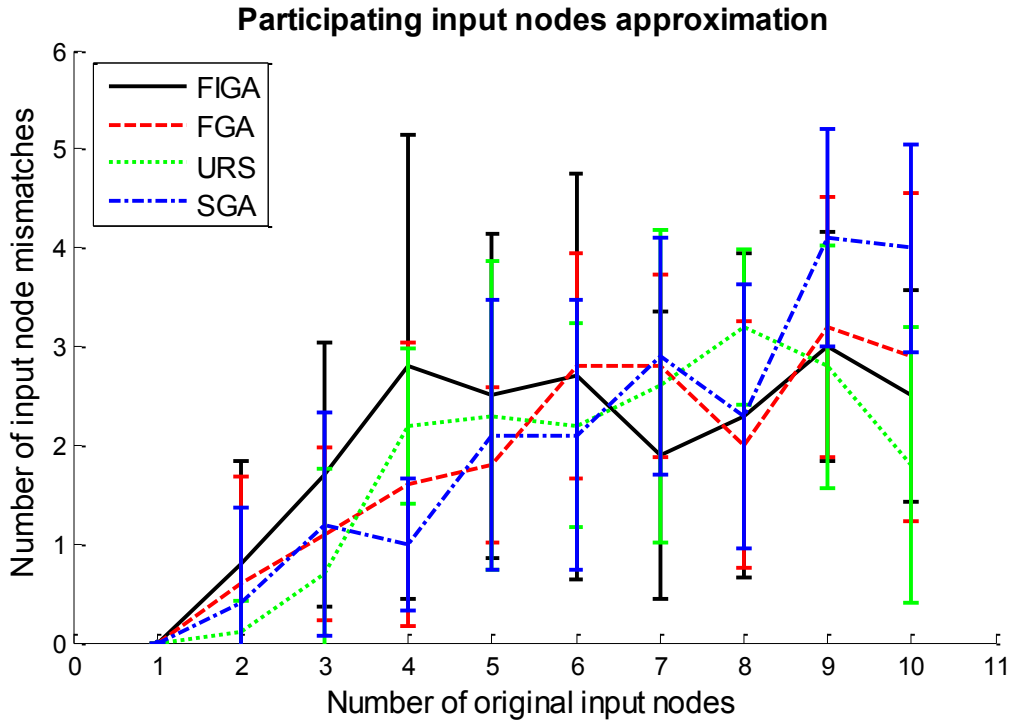
Table 6.12 Mean and Standard Deviation values for the number of output bit mismatches for the modified feedback network with CISGA generated input

Number of original input nodes	URS input node mismatches: mean ; standard deviation values	SGA input node mismatches: mean ; standard deviation values	FIGA input node mismatches: mean ; standard deviation values	FGA input node mismatches: mean ; standard deviation values
1	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000	0.0000 ; 0.0000
2	0.1000 ; 0.3162	0.4000 ; 0.9661	0.8000 ; 1.0328	0.6000 ; 1.0750
3	0.7000 ; 1.0593	1.2000 ; 1.1353	1.7000 ; 1.3375	1.1000 ; 0.8756
4	2.2000 ; 0.7888	1.0000 ; 0.6667	2.8000 ; 2.3476	1.6000 ; 1.4298
5	2.3000 ; 1.5670	2.1000 ; 1.3703	2.5000 ; 1.6499	1.8000 ; 0.7888
6	2.2000 ; 1.0328	2.1000 ; 1.3703	2.7000 ; 2.0575	2.8000 ; 1.1353
7	2.6000 ; 1.5776	2.9000 ; 1.1972	1.9000 ; 1.4491	2.8000 ; 0.9189
8	3.2000 ; 0.7888	2.3000 ; 1.3375	2.3000 ; 1.6364	2.0000 ; 1.2472
9	2.8000 ; 1.2293	4.1000 ; 1.1005	3.0000 ; 1.1547	3.2000 ; 1.3166
10	1.8000 ; 1.3984	4.0000 ; 1.0541	2.5000 ; 1.0801	2.9000 ; 1.6633

Table 6.13 Mean and Standard Deviation values for the number of input node mismatches for the modified feedback network with CISGA generated input



(a)



(b)

Figure 6.6. The results of the modified feedback network with CISGA generated inputs. Confidence level 68.3%

6.7 Outperformance of GA-based algorithms over URS

We would like to emphasize the outperformance of GA-based algorithms over URS, based on the output sequences' approximation results of the test sets. This is done in order to highlight the higher precision, achieved by employing GA techniques. Here we do not consider participating input nodes approximation, because the main objective of the present work is the approximation of output bit sequences. In addition, we omit from the analysis the results for the auditory cortex with pulse shaped inputs having Freq=12, since this test set did not provide proper assessment conditions for the developed algorithms.

Since the distribution of results (namely, the number of mismatches between algorithms' outputs and the target output) is unknown, we decided to use non-parametric rank test. Each of the four used test sets is analysed separately. Per each test set we check how many times more precise the best result of the three GA-based algorithms is in comparison to the result, achieved by URS. In other words, how many times less mismatches there are in the best results of SGA / FIGA / FGA versus URS. We will refer to this multiplication factor as outperformance factor (of each out of SGA / FIGA / FGA over URS).

The outperformance factors of each algorithm are sorted (in other words, ranked) across the performed tests. Since there are 100 tests in each test set, each test set yields a sorted list of 100 outperformance factors (one per test) for each of the algorithms. A number of tests with lowest outperformance factors are discarded, depending on desired confidence level. The minimal outperformance factor out of the remaining tests is reported as the one corresponding to the confidence level.

The findings, summarized in Table 6.14, correspond to the confidence level of 0.95. This means that in 95% of cases (tests) outperformance factor of the algorithms is greater or equal to the number indicated in the table. Namely, in each of the four analysed test sets only 5 out of 100 test results are allowed to have a lower outperformance factor.

The table clearly indicates that the "usefulness" of the algorithms varies in different situations. SGA is slightly better than URS in its performance, when the auditory

cortex network is fed with pulse inputs having frequency 6 and with CISGA generated inputs. Surprisingly, SGA becomes considerably more useful in the modified feedback network. We can conclude from this observation that SGA is not able to deal efficiently with high linearity of the auditory cortex network, the reason for which remains unclear to us. As a matter of fact, dealing with linear network is expected to be easier than dealing with non-linear one.

FIGA shows higher outperformance over URS in the auditory cortex network. However, its performance abruptly decreases in the modified feedback network, and for the confidence level of 0.95 it can be no longer stated that FIGA outperforms URS. This behaviour is expected in accordance with the design of the FIGA: it was designed to deal with linear inputs, in which inputs are modified only slightly as a result of their propagation through the linear auditory cortex network. Modified feedback network fed with the CISGA generated inputs was aimed at failing the FIGA, and this result was achieved.

FGA displays the highest outperformance values among the algorithms with evident deterioration of its outperformance, as the non-linearity of the inputs increases. We expected this result as well. The FGA was developed as a fix to the dependence of the FIGA on the linearity of the network, while preserving the flexibility of the SGA. However, it is impossible to keep the performance of any algorithm at the same levels, when the complexity of the problem (i.e., non-linearity of the network and of its inputs) increases exponentially.

Test Set	SGA outperformance factor over URS	FIGA outperformance factor over URS	FGA outperformance factor over URS
Auditory cortex with pulse input, Freq.= 3	1.24	1.84	2.20
Auditory cortex with pulse input, Freq.= 6	1.09	1.66	1.84
Auditory cortex with CISGA generated input	1.07	1.48	1.58
Feedback network with CISGA generated input	1.38	0.97	1.43

Table 6.14 Outperformance values for SGA / FIGA / FGA in the four test sets, corresponding to confidence level of 0.95

We conclude this chapter with the observation that FGA consistently outperforms URS and the two other versions of GA-based algorithms. Based on the evidence presented in this chapter, we believe that FGA is the most successful version among the ones developed in the present work. However, since we have not tested all the possible input patterns and different topologies, which is obviously not feasible with the computing tools we had available, we would not recommend to completely neglect SGA and FIGA when dealing with new networks and new inputs. In addition, due to the stochastic nature of the developed algorithms, there is no guarantee that FGA will perform better in each single case.

In the next chapter we present concluding remarks and future possible directions of the current research.

Chapter 7:

Summary and future work

7.1 Research contributions

The present thesis addressed the problem of regenerating target output sequences in NNNs as precisely as possible by manipulating input sequences. The generation of output sequences involves propagation of signals among the neurons of the network obeying to the basic neuronal laws.

The amount of possible input combinations, as well as the amount of neuronal interactions, leading to a particular output is computationally prohibitive, especially in the networks containing hundreds of neurons and thousands of synapses. In order to overcome the complexity of the required computation, we proposed to use GA based approach. Using this approach we performed a selection of gradually improving input sequences in respect to the approximation precision of their generated output. The evolution of input sequences was implemented by mutations and crossovers.

We developed the algorithm for simulation of signal propagation in NNNs, referred to as the signal propagation framework, and three GA-based algorithms for regenerating target output sequences: SGA, FIGA and FGA. The additional algorithm implementing simple guessing procedure without using evolutionary selection mechanisms, called URS, was introduced as a control for assessing the performance of SGA, FIGA and FGA. Each of the four algorithms uses the signal propagation framework in order to generate outputs.

The performance of the four algorithms was compared in five test sets. The sets differ in the way the input sequences are generated (pulse shaped with constant frequency versus CISGA generated input sequences) and in the underlying NNNs (auditory cortex model network versus modified feedback network). We believe that the combination of CISGA with the modified feedback network, presented in the fifth

test set, provides the most complex target outputs to be handled by the developed algorithms.

Results showed the superiority of FGA over SGA and FIGA in most of the cases, with a few exceptions, where either FGA or SGA achieved slightly better performance. FGA achieved up to 100 out of 6700 bit mismatches in the first four test cases, corresponding to 98.5% precision, and up to about 1000 out of 6700 bit mismatches in the fifth test set, corresponding to 85% precision. The importance of the approximation precision is to be discussed in each particular application domain separately. Generally, in order to avoid finding a single local optimum it is recommended to invoke GA-based algorithms several times on the same problem instance. We recommend running each of the algorithms (SGA, FIGA and FGA) for at least three times per problem instance and then picking the best solution out of the nine solutions found.

Weaknesses of the methodology include dependence of the developed algorithms on the discrete model of signal propagation and low scalability of FIGA and FGA in respect to the number of input nodes. In addition, there is no guarantee on the performance of the algorithms for another network / other propagation parameters. Generally, GA-based algorithms do not provide any theoretical limits on the approximation precision. It is also known that these algorithms are highly sensitive to the parameters supplied to GA computation (mutation and crossover probabilities, population size and number of generations). Therefore, each specific network requires fine tuning of the parameters, which can drastically affect the achieved results.

Another problematic point is possible over fitting of the developed algorithms to their input. Since there are many parameters that should be tuned during the simulations (population size, number of generations, recombination and mutation rates), it is impossible to set them without considering a particular input to be analyzed. Once the parameters are tuned for a particular input, the model/algorithm can no longer be considered general and it is not suited for other inputs. In particular, the results presented in the previous chapter should not be extrapolated to any different type /

structure of NNNs. On the contrary, we do not think that inputs that were generated in this work can in any way over fit the developed algorithms. The generation of inputs is a separate and a completely independent process from the parameters' tuning of the algorithms.

7.2 Future research directions

The research presented in this thesis can be extended in the future in the following directions:

- Improving the model of signal propagation. Incorporation of non-constant time periods between firings of different neurons and asynchronous signal propagation.
- Involving new GA techniques in the developed algorithms for their better performance: using different crossover, mutation and selection operators.
- Testing the algorithms on more real-life NNNs as they emerge in the ongoing research in neuroscience.
- Parallelization of testing in order to collect more measures for computing more precise statistics.
- Parallelization of Template GA step, in which the selection of pairs for mating and the generation of children is performed. It is particularly applicable to the tournament selection for mating, since each pair of parent solutions is selected independently from the rest of the parent pairs and children are generated independently as well.
- Parallelization of the initial assessment phases of both FIGA and FGA. In the case of FIGA it will require a separate thread for each assessed input node. In the case of FGA it will require a separate thread for each amount of input nodes. The scalability problem of FIGA and FGA can be solved, dependent on the availability of the amount of cores equal to the number of input nodes in a network.
- The second phase of FIGA can be parallelized further by running a separate thread per each subset of participating input nodes in the order of the list, generated in the first phase of FIGA.

- Developing nanodevices technology in order to synchronously activate specific neurons in a network with the provided input spike sequences. Porting the output of the proposed algorithms to the nanodevices in a physical level is another direction, which requires thorough investigation.

References

1. Jeff Hasty, David McMillen, Farren Isaacs, James J. Collins (2001). Computational studies of gene regulatory networks: in numero molecular biology. Nature Reviews Genetics 2, 268-279, doi:10.1038/35066056

2. Albertha J.M. Walhout (2006). Unraveling transcription regulatory networks by protein–DNA and protein–protein interaction mapping. Genome Research, 16: 1445-1454, doi:10.1101/gr.5321506

3. Schilling C.H., Schuster S., Palsson B.O., Heinrich R. (1999). Metabolic pathway analysis: basic concepts and scientific applications in the post-genomic era. Biotechnol. Prog. 15, 296−303.

4. He H, Sui J, Yu Q, Turner JA, Ho B-C, et al. (2012) Altered Small-World Brain Networks in Schizophrenia Patients during Working Memory Performance. PLoS ONE 7(6): e38195. doi:10.1371/journal.pone.0038195

5. Cajal, R. (1995). Histology of nervous system of man and vertebrates. Oxford University Press, Oxford.

6. Stuart, G., Schiller, J., Sakmann, B.(1997). Action potential initiation and propagation in rat neocortical pyramidal neurons. Journal of Physiology, 505.3, pp.617-632

7. Knuth, D.E. (1997). The art of computer programming.Vol.1., 3rd ed., Boston: Addison-Wesley, ISBN 0-201-89683-4.

8. Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Boston.

9. Cotta, C., and Troya, J.M. (1998) A hybrid genetic algorithm for the 0-1 multiple knapsack problem. Artificial Neural Nets and Genetic Algorithms 3: 250-254.

10. Freisleben, B., and Merz, P. (1996) New genetic local search operators for the travelling salesman problem. Parallel Problem Solving from Nature – PPSN IV: 890-899.

11. Martello, S., Toth, P. (1990). Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York.

12. Lawler, E. L., Lenstra, J. K., Rinnooy, A. H. G., Shwoys, D. B. (1985). Traveling salesman problem: a guided tour of combinatorial optimization. John Wiley & Sons, Inc., New York.

13. Back, T. (1996). Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Inc., New York.

14. Holland, J. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan.

15. Reeves, R.C., Rowe, J.E. (2002). Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory. Kluwer Academic Publishers Group, Dordrecht, The Netherlands.

16. Freitas, R. A. Jr., Havukkala, I. (2005). Current Status of Nanomedicine and Medical Nanorobotics. Journal of Computational and Theoretical Nanoscience 2 (4): 1–25. doi:10.1166/jctn.2005.001.

17. Hodgkin, A., and Huxley, A. (1952).A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. 117:500–544.

18. Wilson, H.R., and Cowan, J.D. (1972).Excitatory and inhibitory interactions in localized populations of model neurons. Biophys. J., 12:1-24

19. Chrostowski, M., Yang, L., Wilson, H.R., Bruce, I.C., and Becker, S. (2011). Can homeostatic plasticity in deafferented primary auditory cortex lead to travelling waves of excitation? J. Computational Neuroscience 30: 279-299.

20. Berger, D., Borgelt, C., Louis, S., Morrison, A., and Grun, S. (2010). Efficient identification of assembly neurons within massively parallel spike trains. J. Computational Intelligence and Neuroscience 10.1155/2010/439648.

21. Meunier, D., Lambiotte, R., Fornito, A., Ersche, K.D., Bullmore, E.T. (2009).Hierarchical modularity in human brain functional networks. J. Frontiers in Neuroinformatics 10.3389/neuro.11.037.2009.

22. Foeller, E., Vater, M., Kossl, M. (2001). Laminar analysis of inhibition in the gerbil primary auditory cortex. Journal of the Association for Research in Otolaryngology, 2(3), 279–296.

23. Gerken, G. (1996). Central tinnitus and lateral inhibition: An auditory brainstem model. Hearing Research, 97, 75–83.

24. Lu, Y., Jen, P. (2001). GABAergic and glycinergic neural inhibition in excitatory frequency tuning of bat inferior collicular neurons. Experimental Brain Research, 141, 331-339.

25. Douglas, R., Martin, K. (1998). Neocortex. In G. M. Shepherd (Ed.). The synaptic organization of the brain (pp. 459–509). New York: Oxford University Press.

26. Read, H., Winer, J., Schreiner, C. (2002). Functional architecture of the auditory cortex. Current Opinion in Neurobiology, 12, 433–440.

27. Newman, M.E.J., and Girvan, M. (2004). Finding and evaluating community structure in networks. Phys. Rev. E69, 026113.

28. Matthews, P.M., Jezzard, P. (2004) Functional magnetic resonance imaging. J. Neurol. Neurosurg. Psychiatry 2004;75:6-12.

29. Balasubramaniam, S., Boyle, N.T., Della-Chiesa, A., Walsh, F., Mardinoglu, A., Botvich, D., and Prina-Mello, A. (2011).Development of artificial neuronal networks for molecular communication. Nano Communication Networks, doi: 10.1016/j.nancom.2011.05.004.

30. Walsh, F., Boyle, N.T., Mardinoglu, A., Chiesa, A.D, Botvich, D., Prina-Mello, A., Balasubramaniam, S. (2011). Artificial backbone neuronal network for nano scale sensors. 1st IEEE International Workshop on Molecular and Nano Scale Communication (MoNaCom).

31. Montana, D.J., Davis, L. (1989). Training feedforward neural networks using genetic algorithms. Machine Learning 762-767. BBN Systems and Technologies Corp. Cambridge, MA 02138

32. Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986). Learning Representations by Back Propagating Errors. Nature 323, pp. 533-536.

33. Tulai, A., Oppacher, F. (2002). Combining competitive and cooperative coevolution for training cascade neural networks.GECCO-02, Genetic and Evolutionary Computation Conference 2002, pp.618-625, New York.

34. Wah, B.W., Qian, M. (2000). Constrained Formulations for Neural Networks Training and Their Applications to Solve the Two-Spiral Problem. Proceedings of the Fifth International Conference on Computer Science and Informatics.

35. Dasgupta, D., and McGregor, D.R. (1992).Designing application-specific neural networks using the structured genetic algorithm. Combinations of Genetic Algorithms and Neural Networks 1992, COGANN-92: 87-96.

36. Dasgupta, D., and McGregor, D.R. (1992). A structured genetic algorithm: The model and the first results. Presented at AISB PG-Workshop, January. (Tech. Report NO. IKBS-2-91).

37. Guo, Z., Uhrig, R.E. (1992).Using genetic algorithms to select inputs for neural networks. Combinations of Genetic Algorithms and Neural Networks, 1992, COGANN-92. International Workshop on Digital Object, Identifier: 10.1109/COGANN.1992.273937, Page(s): 223 – 234.

38. Betts, R.P., Johnston, D.M., and Brown, B.H. (1976). Nerve fibre velocity and refractory period distributions in nerve trunks. Journal of Neurology, Neurosurgery and Psychiatry 39: 694-700.

39. Blinkov, S.M., and Glezer, I.I. (1968). The human brain in figures and tables: a quantitative handbook. New York, USA: Plenum Press.

40. Izhikevich, E.M. (2004). Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks 15: 1063-1070.

41. Curtis, D. R., Eccles, J. C. (1959) The time courses of excitatory and inhibitory synaptic actions. J Physiol. 12; 145(3): 529–546.

42. Miller, L., Escabi, M., Read, H., Schreiner, C. (2001). Functional convergence of response properties in the auditory thalamocortical system. Neuron, 32, 151–160.

43. Aertsen, A., Diesmann, M., Gewaltig, M.O. (1996). Propagation of synchronous spiking activity in feedforward neural networks. Journal of Physiology-Paris, Volume 90, Issues 3–4, 243-247.

44. Kistler, W.M., Gerstner, W. (2002). Stable Propagation of Activity Pulses in Populations of Spiking Neurons. Neural Computation, Vol. 14, 987-997.

45. Daube, J. R. Handbook of Clinical Neurophysiology, Volume 8. Elsevier, Toronto, Canada, 2008.

46. Wardi, Y. (1989) Random Search Algorithms with Sufficient Descent for Minimization of Functions. Mathematics of Operations Research 14:343-354.

47. Miller, B.L., and Goldberg, D.E. (1995). Genetic Algorithms, Tournament Selection, and the Effects of Noise. Complex Systems 9:193- 212.

48. Knuth, D.E. (1998). The art of computer programming. Vol.3., 2nd ed., Boston: Addison-Wesley, ISBN 0201896850.

49. Nachman, M. W., Crowell, S. L. (2000). Estimate of the Mutation Rate per Nucleotide in Humans. Genetics 156, 297-304.

50. De Jong, K.A., Spears, W.M. (1992) A formal analysis of the role of multi-point crossover in genetic algorithms. Annals of Mathematics and Artificial Intelligence Volume 5, Issue 1, 1-26.