Genetic Algorithm for Optimization using Spiking Neurons

Siegfried Ludwig: s1028810[1], Joeri Hartjes: s4614542[1], Bram Pol: s4815521[1], and Gabriela Rivas: s1047701[1]

[1]SOW-MKI96 Neuromorphic Computing, Radboud University

January 17, 2020

Abstract

The use of evolution-inspired algorithms has been proven a viable solution for tackling problems of optimization. Considering the advantages in parallelism, memory collocation and energy efficiency of neuromorphic computing, it can be concluded that spiking neural networks could offer the possibility for the implementation of more efficient evolutionary algorithms. For this project, we designed and implemented scalable ensembles of spiking neurons to carry out the operations required for a genetic algorithm and implemented these in a simulator to successfully solve for an optimization problem. Two types of implementation have been explored that offer a complexity trade-off between computational space and time, with both designs having linear energy complexity.

1 Introduction

1.1 Background

Evolutionary algorithms have garnered increasing interest over the years, bringing in many advantages for optimisation over traditional methods. For instance, Genetic Algorithm (GA) systems may provide increased performance over conventional strategies and the opportunity for difficult problem solving such as multi-objective optimisation [1]. GAs draw their methods from processes in natural evolution, and are heuristic and stochastic approaches to provide optimal solutions. As in natural evolution, GAs work by modifying the characteristics of individuals in a population across several iterations. This is done by means of reproduction (crossover) and random gene mutation. With each run, individuals with an arrangement of genes with a greater similarity to those in the best solution (higher fitness value) are allowed to reproduce and carry over their genetic information onto the next iteration or 'generation'. In this study, populations are represented as groups of binary bit sequences, or chromosomes, in which each bit represents the value of a gene.

Spiking neural networks (SNNs) are inspired by the real world workings of biological neurons. These transmit information by means of timing and energy spikes, released when the potential difference inside a neuron reaches a certain threshold. Spiking neurons provide opportunities for massively parallel, highly energy efficient computation. This brings into question whether the implementation of genetic algorithms using a model of spiking neurons could lead to a faster and more efficient optimisation technique than that of a traditional GA implementation.

For this project, neurons are represented using the leaky integrate-and-fire (LIF) model. The LIF representation draws from real-life biological neurons' imperfect insulation of potential energy. Leaky integrate-and-fire imitates the process of energy diffusion through a neuron's membrane ('leak') that occurs if the charge inside the neuron is not large enough for firing to occur.

1.2 Project Description

The goal of the project was to realize a full genetic algorithm [2] with spiking neurons on neuromorphic hardware. Our implementation consists of processing inputs of binary genetic sequences,

which are represented as neuronal spikes. The optimization problem is the one max problem due to its simplicity, the objective of which is to deliver a system that, after some time, will learn to produce an output of sequences with optimum fitness, represented as complete gene activation, or a fully positive spike train.

First, the neural ensembles necessary to carry out the operations of a genetic algorithm were designed conceptually. Two ways of information encoding have been worked on concurrently. The binary genetic sequence can be represented sequentially as a spike train, with a spike indicating a 1 and no spike indicating a 0. An ensemble in this design processes one bit at a time. The second way of representing a binary genetic sequence is parallel, using a separate neuron for each position of the sequence, with spike to represent 1 and do not spike to represent 0. The two encodings offer a complexity trade-off between computational space and time.

The neuron type considered here is oriented at the common LIF neurons available on the Loihi chip [3]. Loihi's model of the integrate-and-fire neuron implements a synaptic response current, which refers to the weighted accumulation of input spikes and constant bias. The representation of a neuronal membrane potential 'leaks' potential energy over time and fires when this energy becomes greater than the set threshold.

The neural network was implemented and tested using a spiking neuron simulator[1]. Due to problems connecting to the Loihi neuromorphic research cloud and difficulties with implementing our highly custom setup in a Loihi-compatible language, the implementation was not ported to Loihi and consequently only tested in our simulator.

After generating and evaluating the initial population, the algorithm repeatedly performs selection, crossover, mutation and evaluation on the population until it is terminated (algorithm 1). A criterion for terminating the algorithm can be the solution quality.

Algorithm 1: The genetic algorithm

```
initialize population;
evaluation;
while not terminated do
    selection (bubble sort);
    crossover;
    mutation;
    evaluation;
end
```

Each chromosome is evaluated for its fitness value, which determines the number of offspring that are created from said parent chromosome and carried on to the next generation. For the one max problem considered in this project, the fitness of a chromosome is defined as its sum of active bits. The higher the number of active bits in an individual chromosome, the greater its fitness value. The population is ranked top to bottom, higher to lower fitness, by making use of a pairwise bubble sort step each generation. The use of only a limited number of bubble sort steps will lead to incomplete sorting. On the upside this is more efficient and leads to some variety in the ranking of the chromosomes while avoiding the removal of very promising individuals from the bottom.

Reproduction is implemented with a stochastic crossover method, which splits two sequences at a random point and swaps all subsequent genes between the individuals.

Mutation is carried out by assigning a probability for each bit in a given sequence to flip, turning 0 (no spike) into 1 (spike) and conversely 1 into 0. The mutation rate can be adjusted based on the fitness ranking, allowing chromosomes with higher fitness to experience less mutations.

[1] A modified version of the following simulator was used: https://github.com/HugoChateauLaurent/SNN-computing. Mainly, the implementation of randomness in the membrane potential of the LIF was modified, the refractoriness functionality was added to the LIF, and naming of the nodes for plotting was added.

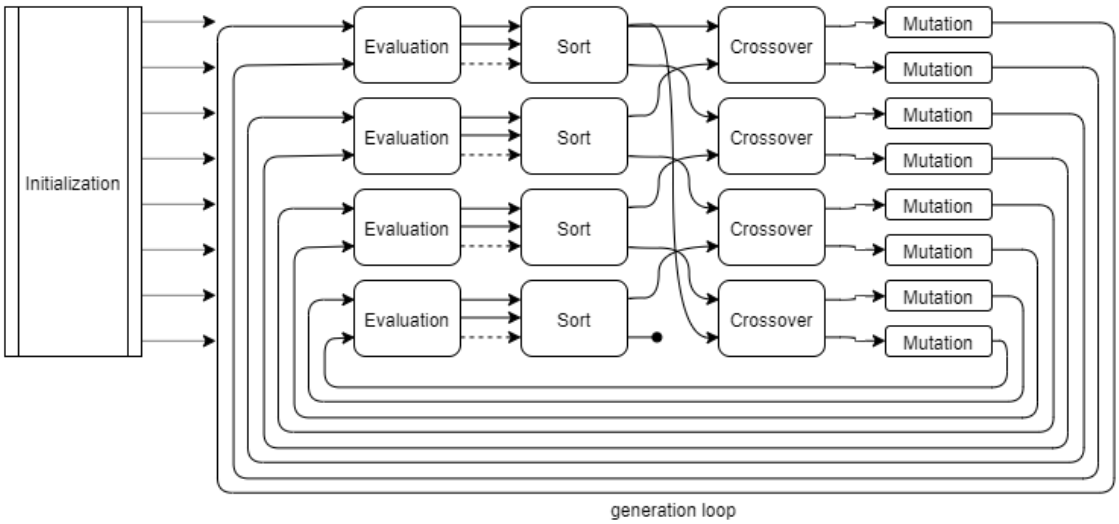

Figure 1: High-level architecture of the genetic algorithm network, depicted with 8 chromosome lanes as solid arrows (one arrow can represent multiple neural connections in the parallel design). To increase the number of chromosomes, the pattern of the second or equivalently the third of the four lanes is repeated.

2 High-Level Architecture

The genetic algorithm is implemented essentially as a single spiking recurrent neural network, consisting of specialized ensembles for each operation (Figure 1). The topology of the network gives a fitness hierarchy, with the fittest chromosomes being at the top and conversely the least fit chromosomes being at the bottom. The network architecture is static during runtime and no learning of the weights is required.

Scaling the network up for a larger population size or longer chromosomes is straight-forward beyond a small minimum size and length, by repeating whole ensembles and repeated elements within certain ensembles.

After initialization, the chromosomes enter the evaluation ensembles in pairs. This results in a decision whether the pair should be swapped in case the bottom chromosome has higher fitness or kept in the same lanes otherwise. The sorting ensemble then executes this step (note that the evaluation and sort ensembles are combined in the parallel design). This setup corresponds to a single pairwise bubble sort step and over time ranks chromosomes by their fitness, which is necessary for selection.

Selection is implemented in the connections from the sorting ensembles to the crossover ensembles, by eliminating the bottom chromosome and connecting the top chromosome twice. This results in better solutions propagating more successfully over time. In addition to potentially moving up one lane in the sorting ensemble itself, the winner of the pairwise evaluations moves up by another lane after sorting, which is done to ensure upwards mobility. Would the outputs of the sorting ensembles connect directly to the crossover within the same lane pair, a chromosome could never move up the hierarchy and every generation the same pairs would coincide. Conversely, the loser moves down another lane after sorting.

Crossover is then performed on each pair of chromosomes. After crossover, each chromosome is processed individually in a mutation ensemble. To close the generation loop, the outputs of the mutation ensembles connect back into the evaluation ensembles, forming a recurrent neural network.

3 Sequential Design

The sequential design of the genetic algorithm is based on the key idea that chromosomes can be processed sequentially one bit at a time, more closely resembling nature. The great advantage of this design is that a small ensemble can process arbitrary lengths of chromosomes without growing in size. The implementation relies on a lead bit, which precedes every chromosome and is always active (see figure 6 and 7). This allows the signalling to an ensemble that a chromosome arrived, which ensures correct processing. In the following, neurons will be ascribed different types based on their function in the ensemble. They are all based on the LIF neuron however.

3.1 Evaluation Ensemble

The sequential onemax evaluation ensemble makes use of 8 LIF neurons and 11 internal connections. It takes as input two chromosomes and gives as output two chromosomes plus a single spike on a separate output neuron indicating whether the chromosomes should be swapped or not. This corresponds to the question on whether the bottom chromosome has a higher fitness than the top chromosome. The membrane potential of the accumulator neuron (ACC) is increased with each active bit in the bottom chromosome, and decreased with each active bit in the top chromosome. The activation neuron (A), activated by the lead bit, then makes the ACC neuron fire or not based on the final membrane potential of the ACC neuron. If the membrane potential is greater than zero, it means that the bottom chromosome had more spikes than the top chromosome, and the two should be swapped. If the membrane potential is zero or less, the two chromosomes should stay in their ranking and the activation neuron is not strong enough to trigger a spike. The reset neuron (R) afterwards makes sure that the accumulator neuron is cleared by making it spike and suppressing the output, since otherwise the old membrane potential could interfere with operations on the next chromosome. Using no membrane leakage is necessary since chromosomes get processed over time.

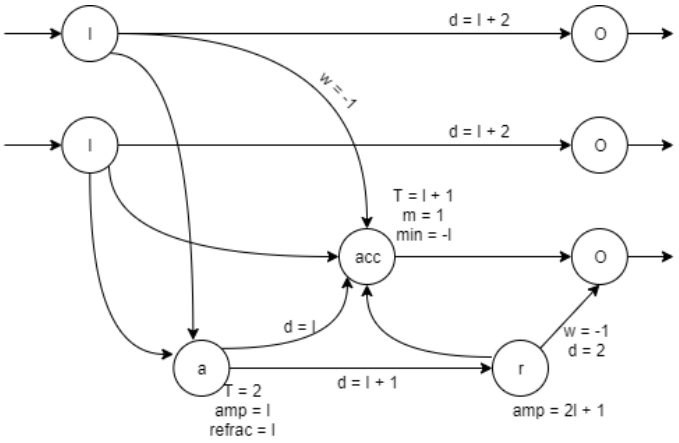

Figure 2: Onemax evaluation ensemble. (I) Input, (acc) Accumulator neuron, (a) Activation neuron, (r) Reset neuron, (o) Output.

3.2 Bubble Sort Ensemble

The bubble sort ensemble consists of 10 neurons and 15 internal connections. It takes two chromosomes plus a fitness indicator as input and gives two chromosomes as output. It uses gate (G) neurons to open or close the identity and swap lanes connecting input and output and thereby controlling whether the chromosomes are swapped or propagated as identity. This is achieved by giving the swap gate neurons a threshold of 2, which means they can only fire if an input comes from the gate control (GC) neuron. The GC neuron is activated by the gate control activation (GCA) neuron, which takes the fitness input coming from the evaluation ensemble indicating whether the chromosomes should be swapped or not. The GC neuron uses a recurrent connection to keep the swap gates open and the identity gates closed until the chromosomes passed through, at which point it is deactivated by a delayed spike coming from the GCA neuron.

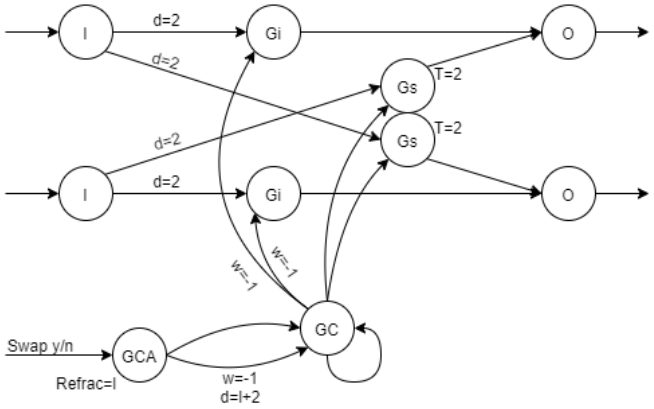

Figure 3: Bubble sort ensemble in the sequential design. (I) input, (Gi) identity gate, (Gs) swap gate, (GC) gate control, (GCA) gate control activation, (O) output.

3.3 Crossover Ensemble

The crossover ensemble works similarly to the bubble sort ensemble, except that identity and swap gates are not open or closed for the whole chromosome, but switch activation at a random point. It uses 13 LIF neurons and 27 internal connections. The ensemble could be simplified to only use one gate control (GC) neuron as in the bubble sort ensemble, but has been implemented with two in this project. Interesting here is the control head which enables the implementation of a random crossover point. It uses a stochastic (S) neuron and a stochasticity control (SC) neuron. The S neuron gets constant input from the SC neuron, which is sub-threshold however. In addition, the S neuron generates some random membrane potential each time step, which can cross the threshold and lead to a spike. If S spikes, the identity GC neuron is deactivated and the swap GC neuron is activated. The S neuron also deactivates the SC neuron, since only one crossover point is desired.

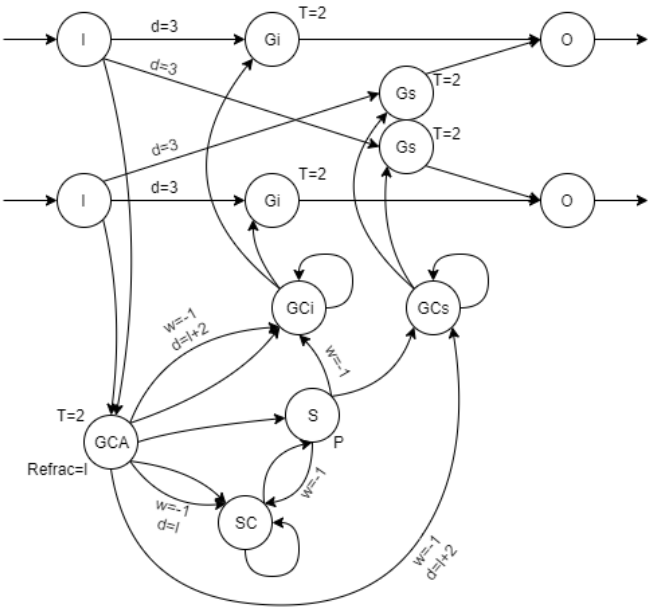

Figure 4: Crossover ensemble in the sequential design. (I) input, (Gi) identity gate, (Gs) swap gate, (GCi) identity gate control, (GCs) swap gate control, (GCA) gate control activation, (S) stochastic, (SC) stochasticity control, (O) output.

3.4 Mutation Ensemble

Finally the mutation ensemble (Figure 5) has a chance of turning a 0 into a 1 and conversely a 1 into a 0, independently for each bit excluding the lead bit. It uses 6 LIF neurons and 12 internal connections. The first stochastic neuron (S1) gets a positive input from each spike in the input and adds a random membrane potential, which can cross the threshold and lead to a spike. A spike

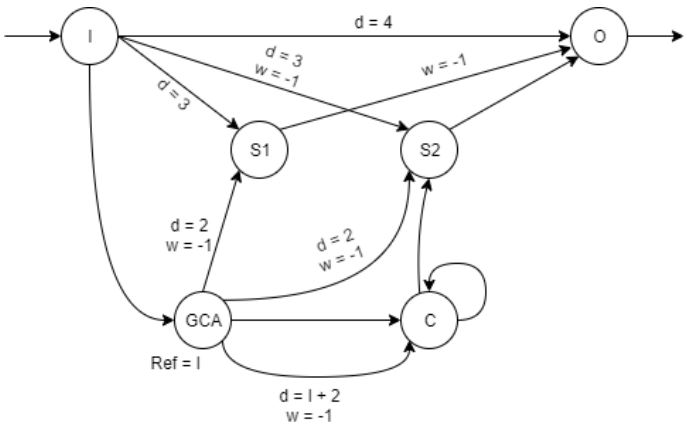

Figure 5: Mutation ensemble in the sequential design. (I) input, (S1) stochastically flips 1 to 0, (S2) stochastically flips 0 to 1, (C) constant, (GCA) gate control activation, (O) output.

from S1 suppresses the ensemble output, thereby turning a 1 into a 0. The other stochastic neuron (S2) always gets input from the control (C) neuron and adds a random membrane potential, but it is suppressed by every spike in the input. If no spike comes from the input, it has a chance of firing and turning the ensemble output from a 0 to a 1. The control neuron is activated and finally deactivated by the control activation (CA) neuron.

3.5 Full Network Behavior

A raster plot of the inputs and all ensemble outputs over time can be seen in Figure 6. The top lanes show the inputs, which have been set to all zeros here with a lead bit. The inputs only fire once in the beginning, giving the chromosome initialization. The chromosomes then pass through the ensembles where they are processed. Note that the end of a chromosome is not particularly visible here, as it is not as well marked as the beginning with the lead bit. A chromosome can still be processed in one ensemble while already entering into the next (e.g. crossover to mutation), which is why the spike clusters can overlap between ensembles.

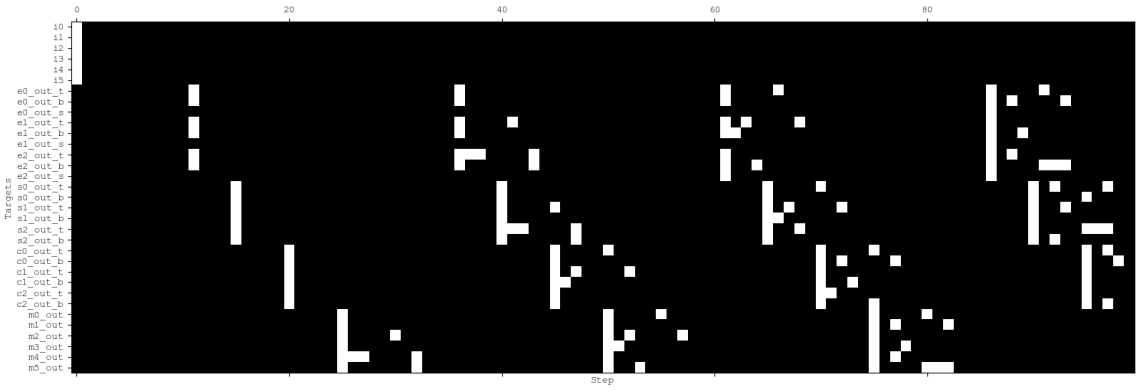

Figure 6: Raster plot of inputs and all ensemble output neurons over time in the sequential design (6 chromosomes of length 8, for 100 steps). Chromosomes are initialized as all zeros. Inputs (i), evaluation (e), bubble sort (s), crossover (c), mutation (m).

An exception to this is the evaluation ensemble, which needs to accumulate the full chromosome to make an evaluation. It therefore breaks the time-constant flow through the other ensembles and leads to a time dependency on the chromosome length. On the upside this prevents chromosomes

from being longer than the execution cycle, which could otherwise lead to the beginning of the next generation interfering with the end of the last for long chromosomes.

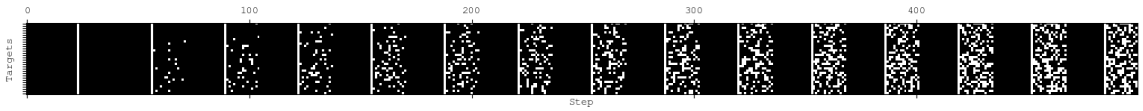

Figure 7: Raster plot of bubble sort output neurons over time in the sequential design (32 chromosomes of length 16, for 500 steps). Chromosomes are initialized as all zeros. The solution quality is improving over time.

A raster plot of only the bubble sort outputs is given in Figure 7. It shows the improving solution quality over time. Interestingly the fitness ranking is not directly apparent. This is caused by the fact that in this implementation the top chromosome gets paired both with another top chromosome and the bottom chromosome. Thereby, good solutions get introduced to the bottom of the hierarchy at every generation.

4 Parallel Design

The idea behind the parallel implementation is that every gene of the chromosome gets processed at the same time. Instead of using a spike train to represent the chromosome, like in the sequential design, we will be using multiple neurons that each represent one gene of the chromosome. A set of neurons can then represent the binary code of the chromosome by either spiking or not. This way, the binary code of the chromosome can be processed in only 1 time-step, but requires more neurons as chromosomes get longer. A generation of the entire algorithm 1 always takes 11 time-steps to run with the parallel design. However, the amount of neurons and connections of the ensemble will grow as the chromosomes or the population size get bigger. The different ensembles used in the algorithm will be explained below.

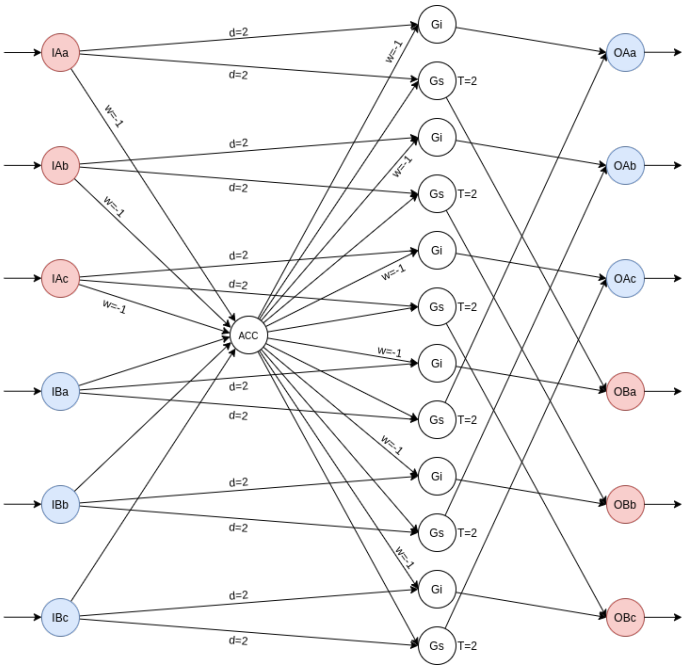

Figure 8: Ensemble responsible for the evaluation and sorting in the parallel design, applied to a pair of chromosomes consisting of three genes.

4.1 Evaluation & Bubble Sort Ensemble

The input to the parallel evaluation and bubblesort is again done by having the fitness of each gene of a pair of chromosomes be represented by the activity of a dedicated input neuron. The goal of the evaluation is to have the chromosome with the highest fitness be the chromosome who's activation will be transferred to the first n output neurons where n is the length of the chromosome. One lead bit is present for the pair of chromosomes which enables functionality in the other segments of the GA, however for this segment it is of no use and therefore linked directly

to its corresponding output neuron. Also for this reason the decision was made to omit the lead bit altogether in the figure 8. A design decision was made to incorporate the parallel evaluation and bubblesort in the same ensemble due to the nature of the parallel implementation in that the fitness of all genes in both chromosomes is available at the same time. This enables the comparison of the fitness through the use of one Accumulator neuron (ACC) to which all input genes are connected (excluding the lead bit).

As can be seen in the figure 8 the weights of the connections from the upper chromosome have a value of -1 while the lower connections have the standard +1 value. This leads to the ACC becoming active only if the lower chromosome has a greater fitness than the top chromosome by at least one gene. Subsequently the activation of the ACC will determine whether either the identity gates, or the swap gates are activated. These are responsible for transferring the activity from the input to the output neuron of the same, or the 'adversarial chromosome', respectively. Each of the input neurons is connected to both a dedicated identity gate and a dedicated swap gate, with these being connected to the identity neuron or the neuron on the other chromosome in the same position. The connection from the input to the gates is delayed by one time step, however, to allow for synchronous arrival of the spike to the spike coming from the ACC. The connections between the ACC and the gates are weighted such that by default the identity gates have a threshold low enough that a spike from the input neurons will be enough to spike the gate as well while the threshold of the swap gates is too high. As soon as the ACC is activated however this spike is no longer enough for the identity gates, while the extra activation coming from the ACC to the swap gates lowers their threshold enough to let the spike pass from the input neuron to the correct output neuron on the side of the 'adversarial chromosome'.

4.2 Crossover Ensemble

The parallel crossover ensemble can be seen in Figure 9. The first gene of every chromosome is always identity and will end up in the same output chromosome. The last gene will always be crossed over and end up in the opposite chromosome. To decide where the genes in between go, a 'random point maker' has been made (see Figure 11). When the crossover ensemble is activated, the random point maker will also be activated by the leadbit. The input to the second layer of the random point maker will spike with a probability of $p = \frac{1}{n-2}$, where n is the chromosome length. This is because the first gene is identity and the last gene is always crossed over, so these will not be taken into account. When the second layer (ia, ib, ic) get a spike from the input, it will give a spike to every neuron in the third layer that is at the same level or below. This will ensure that once a gate is opened, the ones below will also open. The third layer of the random point maker (the gates) connect to the i's and c's in the crossover ensemble. When a gate neuron of the

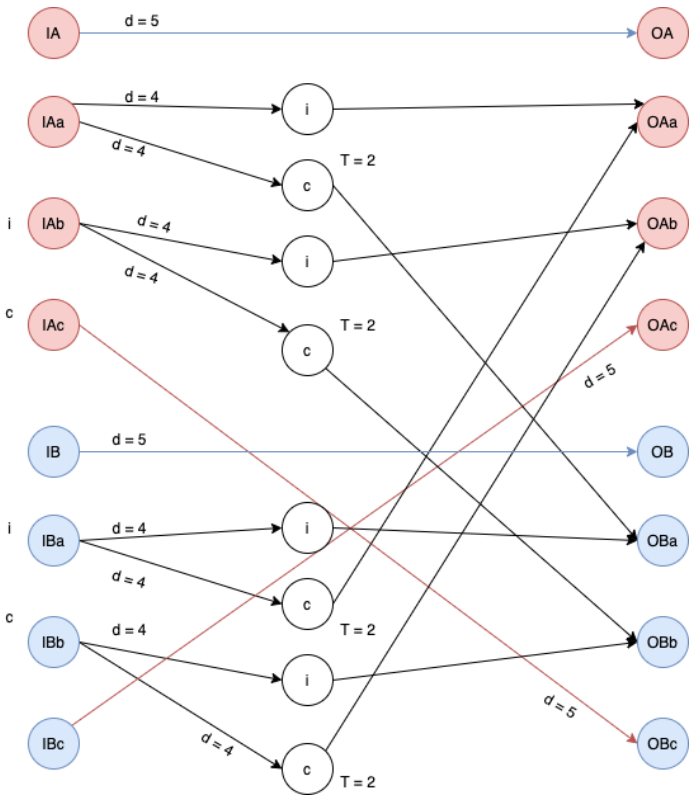

Figure 9: The parallel crossover ensemble.

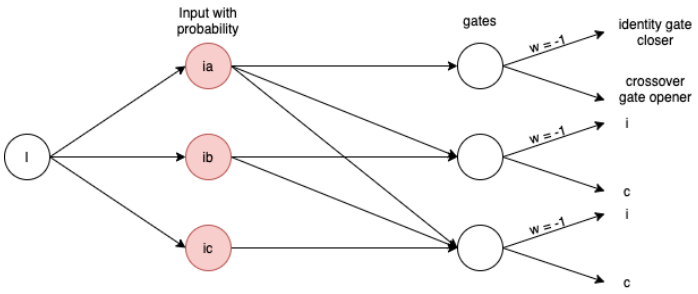

Figure 11: The random point maker that connects to the gates of the crossover ensemble.

random point maker gets a spike, it will close the identity gate and open the crossover gate of both chromosomes at that level. This way, the crossover ensemble starts with sending spikes of genes to the same output chromosome, but has a random point where it will turn into crossing over to the other output chromosome.

The crossover ensemble, together with the random point maker, takes 5 time steps to run for every chromosome length. The number of neurons in the ensemble is $n \cdot 10 - 11$ which is linear growth. However, the amount of connections is not a linear growth, because the connections between the second and third layer of the random point maker grows with $(n^2 + n)/2$, which is exponential growth. Because of this, the number of connections in the whole ensemble grows exponentially.

4.3 Mutation Ensemble

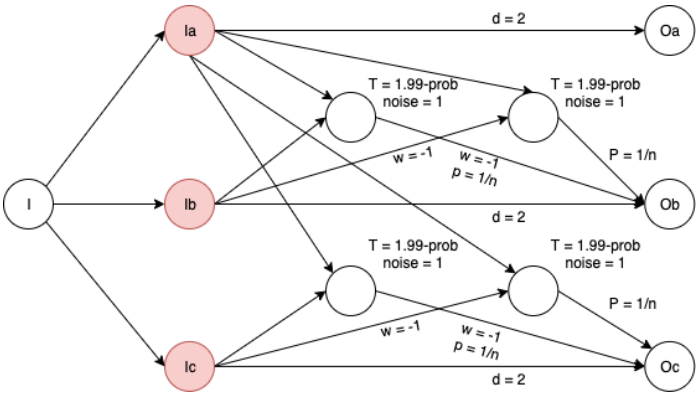

Figure 10: Parallel mutation ensemble

The final ensemble in the parallel design is responsible for the stochastic mutation of the genes in the chromosomes, meaning turning a 1 into a 0 or vice versa (Figure 10). The way it is implemented is through assigning a probability P to each of the genes, and therefore neurons, to switch their activity. The probability is dependent on the length of the chromosome n, and is given by $P = 1/2n$. Except for the lead bit (Ia), every input-neuron (Ib, Ic) is connected to two LIF neurons, and both of their thresholds are influenced by the switching-probability through $T = 2 - P$. A noise factor is present in both intermediate neurons, its function being to add extra randomness as to whether a neuron will mutate or not. In the diagram 10 the first of the two intermediate neurons is responsible for potentially turning off the activation in case that the input neuron has spiked, and the other is responsible for the opposite. Each of the input neurons is connected directly to its corresponding output neuron, however this connection is delayed such that its spike is delivered synchronously to the potential spike of one of the two intermediate neurons. The role of the lead bit is essential to the mutation ensemble, as the lead bit is connected to all of the intermediate neurons in the ensemble. Its guaranteed activity allows for the potential activation of the two intermediate neurons, which otherwise have no chance of reaching their threshold. Combining the stochastic nature of the intermediate neurons, together with the configuration of the intermediate neurons then has the desired effect of a random mutation of the gene together with the appropriate switching of its value.

4.4 Full Network Behavior

The separate ensembles of the parallel implementation described above are connected together in a modular fashion. Each of the ensembles is implemented in a separate function in such a way that both the function parameters and its output are identical to the other ensemble's function parameters and output in terms of type and ordering. This standardized way of handling the in- and outputs of the functions allows for easy connection of the ensembles as if making a puzzle, while also avoiding confusion. As mentioned above, not all of the ensembles require the lead bit, and because it is vital in other ensembles the decision was made to forward it as a separate parameter. The other parameters include the length of the chromosome, and a list of the neurons of both chromosomes. The output of each of the ensemble functions is as follows: *neurons, connections, lead bit, output*. *Neurons* represents a list of all created by the ensemble, *connections* is a list containing all newly-made connections, the *lead bit* represents the created output lead bit, and *output* contains just the output neurons in which the new neurons containing the values for the genes are stored.

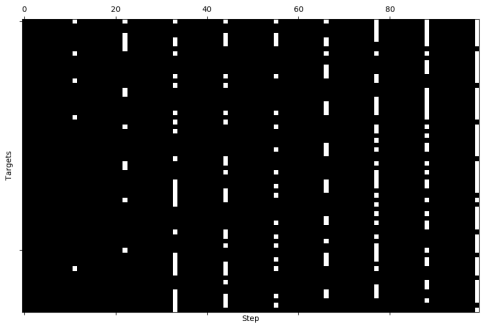

Figure 12: The output for the parallel ensemble for every generation with 8 chromosomes of length 8, with zeros as initialization. The x-axis represents the time. The y-axis represent the genes of the chromosomes (top 8 of y-axis is the best chromosome). A generation takes 11 time steps so it only shows activation at amounts multiple to 11.

In Figure 12 a raster plot of the output is given in which the improvement of the chromosomes can be seen over time. The plot itself shows the power that the mutation has over the improving fitness of the chromosomes. The plot nicely shows that the fitness level of the chromosomes in the upper part is larger than the bottom part, especially in the last few generations since sorting has had more time to affect the ranking. It also shows how significant the effect of the mutation is, seeing that the third generation from the start compared to its previous generation contains a lot more 'fit' genes. The third generation can almost be described as ranked upside down, which could only have been the result of the mutation as none of the other ensembles are able to generate fitness out of nothing (especially in the lowest ranks).

5 Analysis

The quality of solutions increases over time, which validates our general design and implementation (Figure 13). The top chromosome in the hierarchy mostly has a higher than average fitness, which specifically shows that even the single pairwise bubble sort step at each generation is enough to at least approximate a fitness ranking.

Both the top and the average solutions converge around sub-optimal fitness levels. This is explained by the mutation applied at every generation, which can decrease solution quality. It is a problem of the specifics of the genetic algorithm, not of the fact that it is implemented as a spiking neural network.

Computational complexity for neuromorphic computing is considered in terms of space, time and energy. For this analysis, all three complexities have been considered with regard to the number of chromosomes and the chromosome length. Comparing the complexity of the sequential and parallel design shows a space-time trade-off between the two (Figure 14), with the sequential design requiring less space but more time.

All tests with regard to a growing number of chromosomes have been performed with a fixed

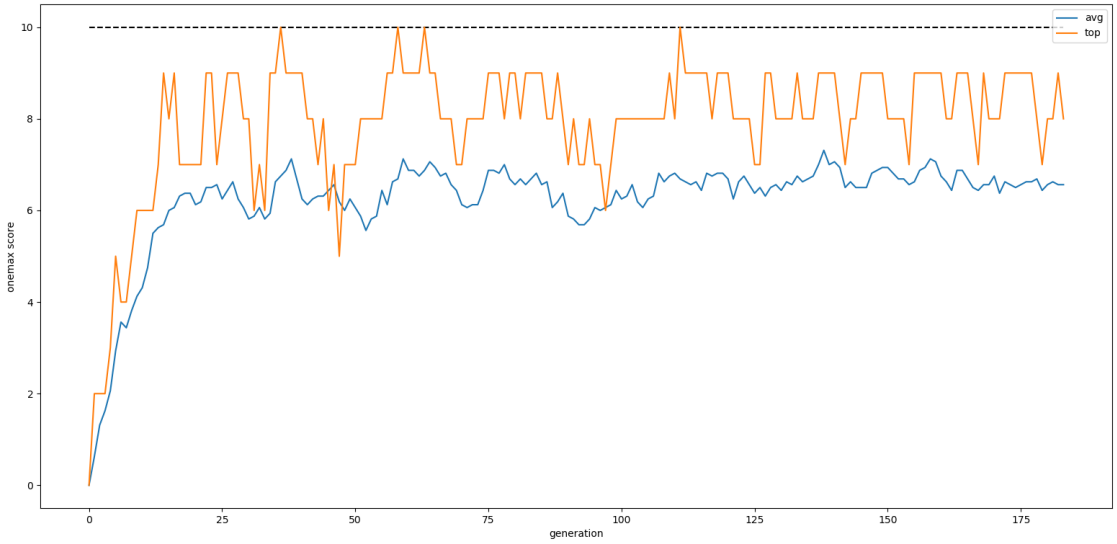

Figure 13: Average and best solution quality over generations (16 chromosomes of length 10, for 5000 steps). The theoretical best solution has fitness score 10 and is only sporadically reached due to mutation. The more stable convergence limit appears to be around 0.85. This is a problem of the specifics of the chosen genetic algorithm itself, not necessarily of the implementation as a spiking neural network. While this plot comes from the sequential design, the parallel design behaves similarly.

chromosome length of 16, while conversely all tests with regard to a growing chromosome length have been performed with a fixed number of chromosomes of 16.

Both designs have linear space complexity in the number of chromosomes, both in terms of the number of neurons and the number of connections. The sequential design has much lower space requirements however.

Regarding the chromosome length, the sequential design has constant space complexity, while the parallel design is linear in the number of neurons and exponential in the number of connections. This is the least favorable of all measured behaviors. It is specifically caused by the current implementation of randomly determining a crossover point.

Both designs have constant time complexity in the number of chromosomes, with time measured in simulation steps per generation.

While the parallel design also has constant time complexity in the chromosome length, the sequential design has linear time complexity. The sequential design inherently needs to have at least linear time complexity in the chromosome length, as the full chromosome needs to be accessed before an evaluation can be made. This is an advantage for the parallel design, as the full chromosome is available at once.

Both designs have linear energy complexity in the number of chromosomes and in the chromosome length, when measuring energy as the average number of spikes required to process one generation. This has been tested with randomly initialized chromosomes and averaged over 30 test.

6 Discussion

A fully functioning genetic algorithm has been successfully implemented as a spiking neural network with two different designs, representing chromosomes sequentially as a spike train over time or as parallel spikes at a single time step. Both implementations are freely scalable beyond a small minimum number of chromosomes, with arbitrary chromosome lengths.

The complexity analysis space, time and energy shows the tractability of this approach with

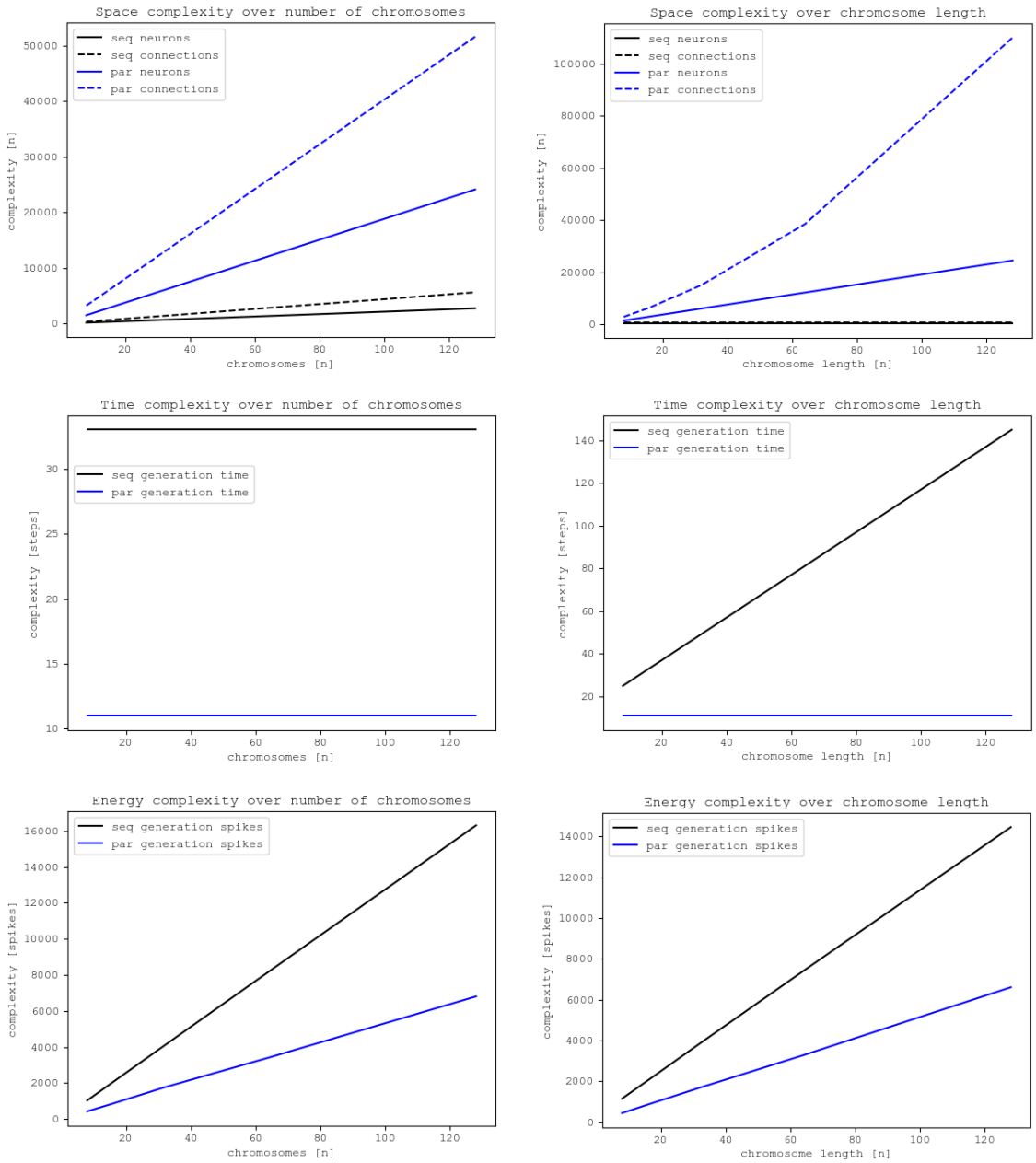

Figure 14: Complexity analysis of space, time and energy behavior for both the sequential and the parallel design. There is a trade-off between space and time comparing the two designs, with the sequential design requiring less space but more time. The worst behavior can be seen in the number of connections growing exponentially with the chromosome length in the parallel design. The sequential design has at most linear complexity.

the exception of the exponentially growing number of connections required for the parallel design when increasing chromosome length. The sequential design is at most linear in any of the analyzed complexities.

The design has not been implemented on neuromorphic hardware. Since fairly standard leaky-integrate-and-fire neurons were used however and no learning is required, translating the design to an implementation in neuromorphic hardware should be possible.

For future work, the design of the crossover ensembles could be adapted to support gene lengths of more than one bit (a chromosome consists of a number of genes, which itself could consist of a number of bases/bits). Practically this just means that the random crossover point should only be allowed at transition points between genes, so at fixed intervals.

References

[1] Kalyanmoy Deb. *multi-objective optimization using evolutionary algorithms*. 2001.

[2] John R Koza. *Genetic programming*. 1997.

[3] Chit-Kwan Lin et al. "Programming spiking neural networks on Intel's Loihi". In: *Computer* 51.3 (2018), pp. 52–61.