25th Conference on Membrane Computing (CMC 2024)

An Evolutionary Approach to the Design of Spiking Neural P Circuits

Alberto Leporati and **Lorenzo Rovida**

University of Milano-Bicocca

Department of Informatics, Systems, and Communication

alberto.leporati@unimib.it, lorenzo.rovida@unimib.it

Nice – June 3rd, 2024

Our goals

- Define SN P circuits, for computing Boolean functions
- Investigate how well evolutionary algorithms discover SN P circuits that compute a given (possibly partially defined) Boolean function $f:\{0,1\}^n \to \{0,1\}^m$

  Partially defined = defined through some pairs $(\mathbf{x},\mathbf{y})$, with $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^m$
- Some classification problems can be seen as partially defined Boolean functions

For simplicity,
we start with Genetic Algorithms (GA)
and $m=1$

SN P gates

- An SN P gate is a spiking neuron containing a subset of the rules

$$a^i \to a \qquad \text{for } i \in \{0, 1, \ldots, \ell\}$$

  where $\ell \geq 1$ is the number of input lines

- An SN P gate computes a Boolean function $f : \{0,1\}^\ell \to \{0,1\}$, by encoding 0 as no spike, and 1 as one spike

---

- Differences with respect to standard spiking neurons:
  - ✓ No delays
  - ✓ Regular expressions have the simple form $E = a^i$
  - ✓ No (explicit) forgetting rules
  - ✓ Rule $a^0 \to a$ (can be avoided, using different encodings)

Symmetric gates

- The output of a symmetric gate only depends upon the number of 1's given as input
- A ℓ-input symmetric gate can thus be represented by a subset $G \subseteq \{0, 1, \ldots, \ell\}$
- Examples of symmetric gates:
  - ℓ-input AND gate: $\mathrm{AND} = \{\ell\}$
  - ℓ-input OR gate: $\mathrm{OR} = \{1, 2, \ldots, \ell\}$
  - (1-input) NOT gate: $\mathrm{NOT} = \{0\}$
  - (2-input) XOR gate: $\mathrm{XOR} = \{1\}$
  - ℓ-input PARITY gate: $\mathrm{PARITY} = \{1, 3, 5, \ldots, \ell\} \quad \text{for } \ell \text{ odd}$

$$\mathrm{PARITY} = \{1, 3, 5, \ldots, \ell - 1\} \quad \text{for } \ell \text{ even}$$

Symmetric gates

- The output of a symmetric gate only depends upon the number of 1's given as input
- A ℓ-input symmetric gate can thus be represented by a subset $G \subseteq \{0, 1, \dots, \ell\}$
- Examples of symmetric gates:
  - ℓ-input AND gate: $\mathrm{AND} = \{\ell\}$
  - ℓ-input OR gate: $\mathrm{OR} = \{1, 2, \dots, \ell\}$
  - (1-input) NOT gate: $\mathrm{NOT} = \{0\}$
  - (2-input) XOR gate: $\mathrm{XOR} = \{1\}$
  - ℓ-input PARITY gate: $\mathrm{PARITY} = \{1, 3, 5, \dots, \ell\} \quad \text{for } \ell \text{ odd}$

    $\mathrm{PARITY} = \{1, 3, 5, \dots, \ell - 1\} \quad \text{for } \ell \text{ even}$

Also the SN P gates

are symmetric !

SN P circuits

- Circuits composed of SN P gates
- Circuits = acyclic graphs (nodes = SN P gates, edges = synapses)
- W.l.o.g., circuits are made of layers of SN P gates
- A n-input/m-output SN P circuit computes a Boolean function $f : \{0, 1\}^n \to \{0, 1\}^m$

Evaluation of an SN P circuit:

- Start with a vector $\mathbf{x} \in \{0, 1\}^n$ encoding the n Boolean input values
- For each layer, compute its output vector by applying the rules in each SN P gate of the layer
- The output vector of the last layer is the output of the SN P circuit

SN P circuits

Observations:

- Each gate takes its input values from the input vector to the layer – which is the output vector of the previous layer (the input vector, for the first layer)
- There is no point in considering delays in the rules
- Each SN P gate starts with 0 spikes

⇒ If we want to reuse the circuit, we must « clean it » before evaluation

Otherwise, we assume implicit forgetting rules $a^i \to \lambda$

Boolean functions

Observations:

- The number of functions $f:\{0,1\}^n \to \{0,1\}^m$ is $(2^m)^{2^n}$
- Even restricting to $m = 1$, we have 2^{2^n} functions

$\Rightarrow$ Finding a specific function is like finding a needle in the haystack

Unfeasible already for $n = 8$

- If we are given $2^n - k$ input/output pairs that partially define $f:\{0,1\}^n \to \{0,1\}$, there are 2^k Boolean functions which are coherent with such a partial specification

  - If we are given $2^n - k$ input/output pairs that partially define $f:\{0,1\}^n \to \{0,1\}^m$ there are $(2^m)^k = 2^{k\cdot m}$ Boolean functions which are coherent with such a partial specification

Boolean functions

- Hence, the fewer input/output pairs given, the easier it is to find a Boolean function consistent with them
- Polynomial-size, constant depth circuits made of symmetric gates are more powerful than AND/OR/NOT circuits:

Theorem (Furst, Saxe, Sipser, 1985): No polynomial-size, constant depth AND/OR/NOT circuit can compute the PARITY function.

Research questions

RQ1: How well (or poorly) do evolutionary algorithms work, in finding the SN P circuits that compute a given Boolean function/solve a given classification task?

RQ2: How does the quality of the solutions found vary depending on some characteristics of the Boolean function (for example, non-linearity)?

Non-linear Boolean functions are interesting for cryptographic applications

Genetic Algoritms

- Genetic Algorithms (GA) are meta-heuristic optimization algorithms
- They solve constrained and unconstrained optimization problems
- A population of candidate solutions (individuals) is evolved through generations, driven by a fitness function
- Operators to be defined:
  - Selection
  - Crossover
  - Mutation

Parameters to be defined:

- Population size
- Probability of crossover
- Probability of mutation
- Number of generations / halting condition

Genetic Algoritms

- Fitness function: Given a list of pairs $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_k, \mathbf{y}_k)$ that partially define a function $f : \{0,1\}^n \to \{0,1\}^m$, the fitness of a circuit c is:

$$\text{fitness}(c, f) = 1 - \tfrac{1}{k} \left(\sum_{i=1}^{k} c(\mathbf{x}_i) \neq \mathbf{y}_i \right)$$

- Selection operator: *fitness proportionate selection*

  ➡ The *fittest* individuals are chosen for breeding

- Elitism: 10% of the best individuals are copied to the next generation

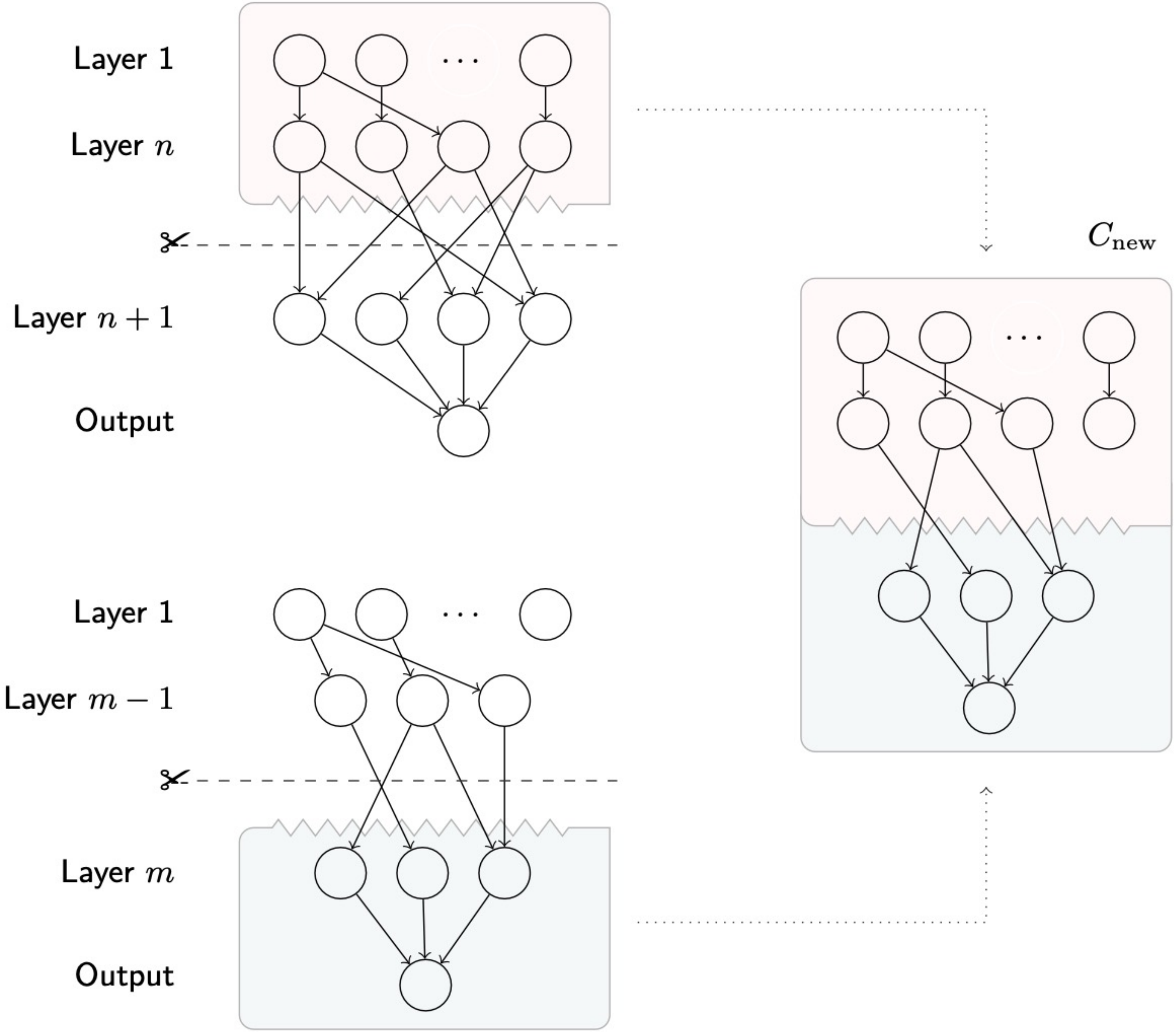
- Halting condition: *fixed number of generations*

Genetic Algoritms

- Crossover: inspired by *one-point crossover*, in which parent circuits are cut before a (randomly chosen) layer
- Two new individuals are produced by taking the first (resp., second) part of the first circuit, and the second (resp., first) part of the second circuit
- The fitness of the two children is evaluated, and only the best one is kept

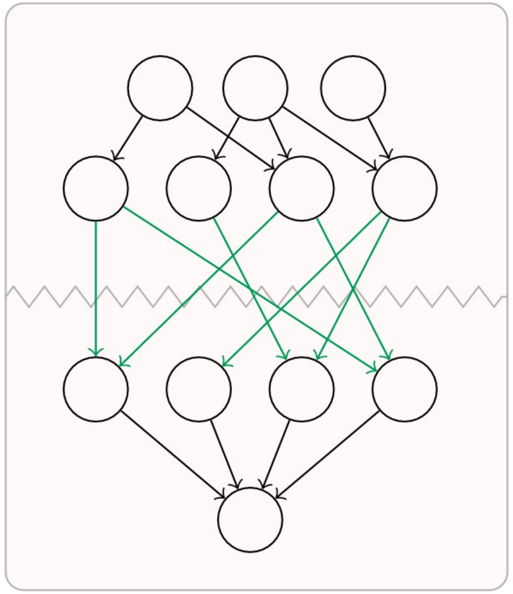
Genetic Algoritms

- Crossover operator:

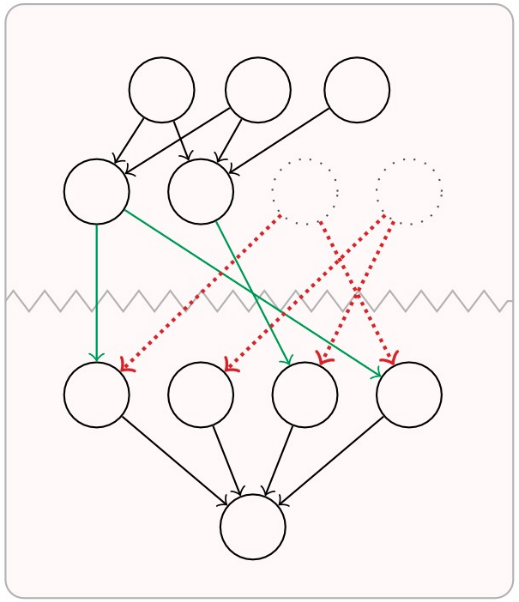


Genetic Algoritms

- Crossover: fixing (cleaning) the children circuit, if the two layers before the cut, in the parent circuits, have a different number of neurons



Same number of neurons = no cleaning



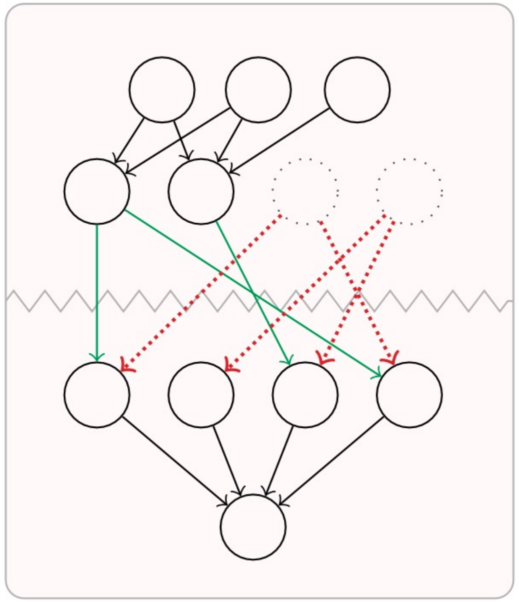
Red edges are invalid = cleaning required

Genetic Algoritms

- Crossover: fixing (cleaning) the children circuit, if the two layers before the cut, in the parent circuits, have a different number of neurons

Cleaning means:

- Removing invalid edges
- Removing gates whose output is not used in the next layer
- Removing empty layers



Red edges are invalid = cleaning required

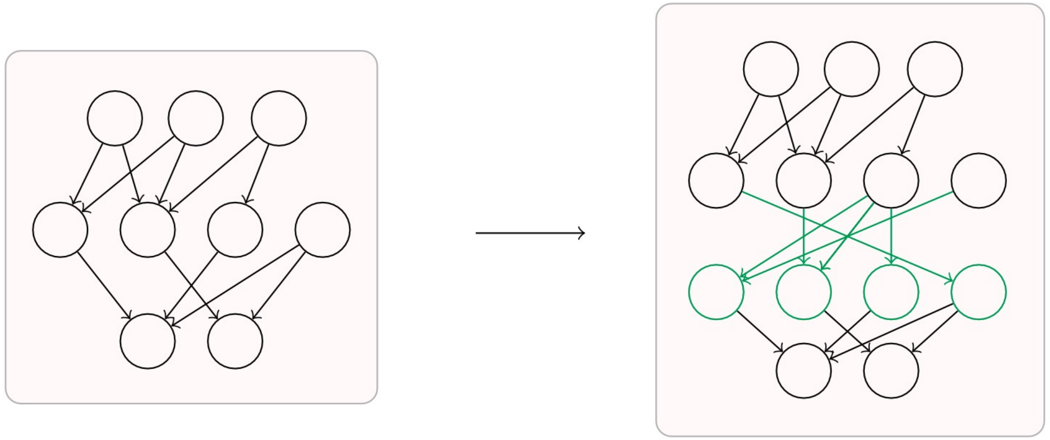
Genetic Algoritms

Several kinds of mutation:

- Adding a new rule to randomly chosen SN P gates
- Removing a rule to randomly chosen SN P gates
- Adding a new input line to randomly chosen SN P gates
- Removing an input line to randomly chosen SN P gates (requires cleaning)
- Adding a new gate in a randomly chosen layer (plus random edges)
- Removing a gate in a randomly chosen layer (requires cleaning)

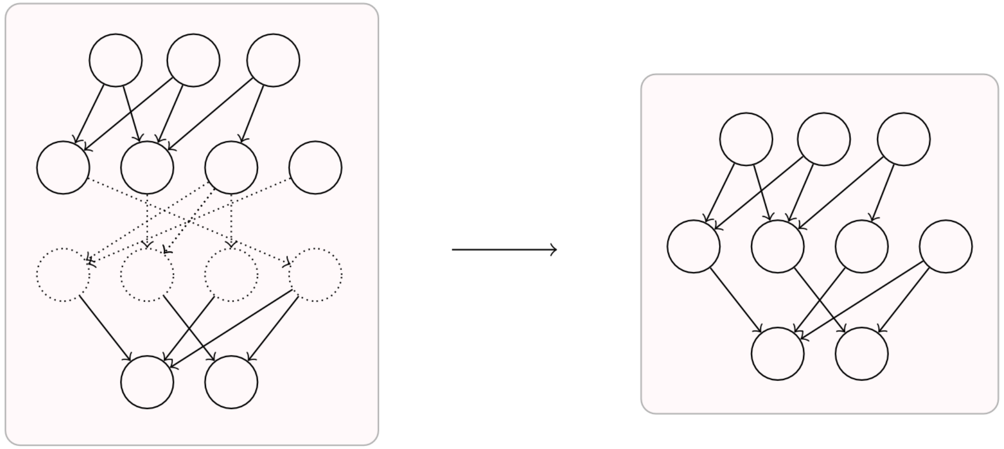
Genetic Algoritms

More drastic mutations:



Adding a new layer to the circuit

(plus random edges)



Removing a layer from the circuit

(requires cleaning)

Genetic Algoritms

- Pseudocode:

Algorithm 3 Genetic Algorithm for finding a SN P circuit that computes a given Boolean function f

```
procedure EVOLVE(f)
    ▷ f is (possibly partially) defined by a list of pairs (x, y) ∈ {0,1}^n × {0,1}^m
    P ← initial population of randomly generated SN P circuits
    FIT ← fitness values of each circuit in P
    save the maximum and average fitness values in a list
    while stopping criterion is not verified do
        ▷ P′ and FIT′ constitute the new generation
        P′ ← best p percent circuits from P                  ▷ p typically is in [0, 10]
        FIT′ ← fitness values of circuits in P′
        for i ← 1 to (1 − p/100) · |P| do
            parent1, parent2 ← FITNESSPROPORTIONATESELECTION(P)
            child ← CROSSOVER(parent1, parent2)              ▷ only the fittest among
                                                               the two children is kept
            add MUTATION(child) to P′
            add the fitness of child to FIT′
        end for
        save the maximum and average fitness values
        P ← P′                                                ▷ substitute P with P′
    end while
end procedure
```

Experiments

The following mutation probabilities have been experimentally found, and have been used in all experiments:

	Add layer	Remove layer	Add neuron	Remove neuron	Add rule	Remove rule	Random input lines
Probability	0.008	0.080	0.030	0.200	0.005	0.080	0.010

For every simulation, the best and the average fitness of individuals were recorded

First experiment: PARITY (XOR) function

- 5 inputs, 1 output
- Function entirely specified (all input/output pairs provided)
- Population size $|P| \in \{20, 30, 40\}$
- Minimum number of intermediate layers in randomly generated circuits: $l_{min} \in \{1,2\}$
- Maximum number of intermediate layers in randomly generated circuits: $l_{max} \in \{l_{min}, l_{min} + 1\}$
- Stopping criterion: 200 generations

- 15 simulations ⟹ Computed the average between the best fitness in each simulation

First experiment: Parity (XOR) function

Population	Min layers	Max layer	Mean Fitness	Successful simulations
40	3	4	0.93125	2/15
40	1	2	0.9125	1/15
30	3	3	0.9125	2/15
30	3	4	0.9125	2/15
40	2	3	0.910417	1/15
20	3	4	0.908333	5/15
30	2	3	0.908333	4/15
		⋮		
20	1	1	0.879167	1/15

Second experiment: PARITY (XOR) function

- 8 inputs, 1 output
- Function entirely specified (all input/output pairs provided)
- Population size $|P| \in \{60, 80, 100\}$
- Minimum number of intermediate layers in randomly generated circuits: $l_{min} \in \{1,2,3\}$
- Maximum number of intermediate layers in randomly generated circuits: $l_{max} \in \{l_{min}, l_{min} + 1\}$
- Stopping criterion: 500 generations

  - 15 simulations for each set of parameters

Second experiment: PARITY (XOR) function

Population	Min layers	Max layer	Mean Fitness	Successful simulations
100	2	2	0.889844	0/15
100	3	3	0.835417	0/15
60	3	4	0.835156	1/15
80	3	3	0.827604	1/15
60	3	3	0.819792	0/15
60	1	2	0.811979	0/15
100	1	1	0.810156	0/15
		⋮		
60	1	1	0.765104	0/15

Second experiment: PARITY (XOR) function

Grid search over the following mutation probability values:

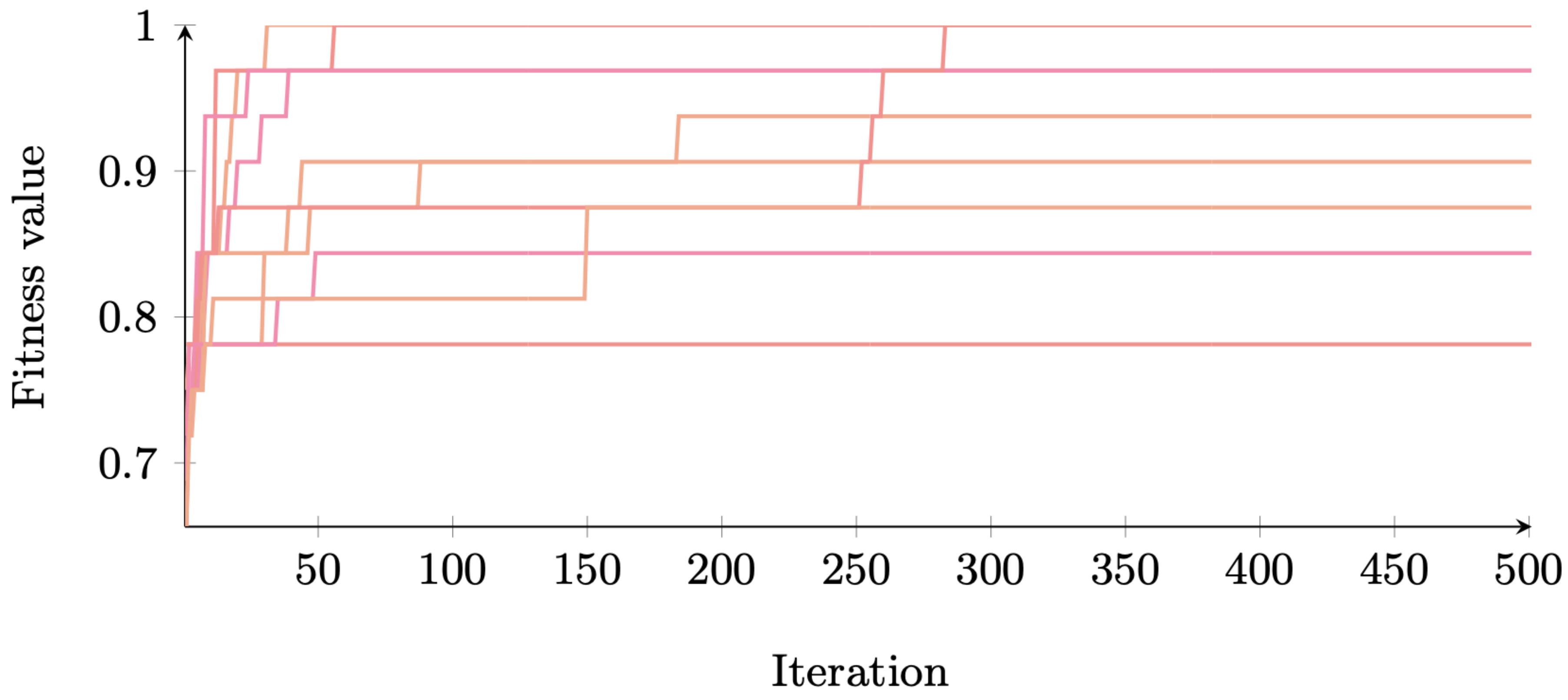
- Probability of adding a new layer at each iteration ∈ $\{0.005, 0.01, 0.02\}$
- Probability of adding a new neuron in a random layer ∈ $\{0.03, 0.05\}$
- Probability of removing a neuron from a random layer ∈ $\{0.05, 0.15, 0.25\}$
- 15 simulations for each set of parameters

Second experiment: PARITY (XOR) function

Add layer probability	Add neuron probability	Remove neuron probability	Average Fitness	Successful simulations
0.01	0.05	0.05	0.9125	2/15
0.01	0.05	0.25	0.8930	0/15
0.01	0.03	0.15	0.8883	0/15
0.01	0.03	0.25	0.8628	0/15
0.01	0.05	0.15	0.8529	1/15
0.02	0.05	0.05	0.8487	0/15
0.005	0.03	0.05	0.8482	0/15
		⋮		
0.02	0.03	0.05	0.8016	0/15

Second experiment: PARITY (XOR) function

Growth of fitness value for the best 15 circuits, using the best set of parameters:



Third experiment: ANF Boolean functions

We considered two Boolean functions expressed in the Algebraic Normal Form (ANF):

$$
\begin{aligned}
f_1(x_1, \ldots, x_5) = 1 &\oplus x_2 \oplus (x_1 \wedge x_2) \oplus (x_1 \wedge x_4) \oplus (x_1 \wedge x_5) \oplus (x_2 \wedge x_3) \oplus \\
&(x_3 \wedge x_5) \oplus (x_1 \wedge x_2 \wedge x_3) \oplus (x_1 \wedge x_2 \wedge x_5) \oplus (x_1 \wedge x_3 \wedge x_4) \oplus \\
&(x_1 \wedge x_3 \wedge x_5) \oplus (x_1 \wedge x_4 \wedge x_5) \oplus (x_3 \wedge x_4 \wedge x_5) \oplus \\
&(x_1 \wedge x_3 \wedge x_4 \wedge x_5) \oplus (x_2 \wedge x_3 \wedge x_4 \wedge x_5) \oplus \\
&(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5)
\end{aligned}
$$

$$
\begin{aligned}
f_2(x_1, \ldots, x_5) = x_1 &\oplus (x_1 \wedge x_4) \oplus (x_1 \wedge x_5) \oplus (x_2 \wedge x_4) \oplus (x_3 \wedge x_5) \oplus \\
&(x_4 \wedge x_5) \oplus (x_1 \wedge x_2 \wedge x_3) \oplus (x_1 \wedge x_2 \wedge x_4) \oplus (x_1 \wedge x_2 \wedge x_5) \oplus \\
&(x_2 \wedge x_3 \wedge x_4) \oplus (x_2 \wedge x_3 \wedge x_5) \oplus (x_1 \wedge x_2 \wedge x_3 \wedge x_4) \oplus \\
&(x_1 \wedge x_3 \wedge x_4 \wedge x_5) \oplus (x_2 \wedge x_3 \wedge x_4 \wedge x_5) \oplus \\
&(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5)
\end{aligned}
$$

Third experiment: ANF Boolean functions

- 5 inputs, 1 output
- Function partially specified (80% input/output pairs provided)
- Population size $|P| \in \{75, 100, 125\}$
- Minimum number of intermediate layers in randomly generated circuits: $l_{min} \in \{1,2,3\}$
- Maximum number of intermediate layers in randomly generated circuits: $l_{max} \in \{l_{min}, l_{min} + 1\}$
- Stopping criterion: 500 generations

- 20 simulations for each set of parameters

Third experiment: ANF Boolean functions

	Input set size	Population	Min layers	Max layer	Mean Fitness	Successful simulations
f_1	Partial (22/32)	100	2	2	0.91818	3/20
		125	2	2	0.91591	1/20
		100	2	3	0.91136	1/20
	Complete (32/32)	125	1	1	0.89531	1/20
		125	3	3	0.89375	0/20
		100	1	1	0.88906	0/20
f_2	Partial (22/32)	100	3	4	0.92500	1/20
		100	2	3	0.925	1/20
		125	1	1	0.92273	1/20
	Complete (32/32)	125	2	3	0.86406	0/20
		125	3	3	0.86406	0/20
		125	1	1	0.86094	0/20

Third experiment: ANF Boolean functions

Grid search over the following mutation probability values:

	Add layer probability	Add neuron probability	Remove neuron probability	Average Fitness	Successful simulations
f_1	0.001	0.01	0.25	0.90469	$0/30$
	0.003	0.01	0.25	0.90156	$0/30$
	0.003	0.03	0.15	0.89687	$1/30$
f_2	0.003	0.03	0.05	0.87187	$0/20$
	0.005	0.01	0.05	0.86875	$0/20$
	0.003	0.03	0.25	0.8625	$0/20$

Future work

- Perform further experiments, on more complicated Boolean functions
- Implementing different crossover operations, and stopping criteria
- Implementing other evolutionary algorithms
  - Grammatical Evolution, Evolution Strategies, Memetic Algorithms
  - Augmenting structures?
- Perform an ablation study
- Explore the fitness landscape

- Apply evolutionary techniques to standard SN P systems (and their extensions)
- Compare SN P circuits with other kinds of Boolean circuits

Thank you
for your attention !

Alberto Leporati and **Lorenzo Rovida**

alberto.leporati@unimib.it, lorenzo.rovida@unimib.it