Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

David Baumeister

born in Heidelberg (Germany)

2024

Placement strategies for multi-compartment neurons on BrainScaleS-2

This Bachelor Thesis has been carried out by David Baumeister at the
European Institute for Neuromorphic Computing (EINC), Kirchhoff-Institut für Physik (KIP)
in Heidelberg
under the supervision of
Dr. Johannes Schemmel

Abstract

BrainScaleS-2 (BSS-2) is a mixed-signal neuromorphic system that allows the emulation of spiking neural networks. These neural networks can be constructed out of neuron circuits, that partially replicate the behaviour of biological neurons, and their synaptic connections.

The neuron circuits on BSS-2 are arranged in a 2D grid and can be connected to form spatially extended neurons. To use these multicompartment neurons on the BSS-2 chip a mapping between an abstract neuron model and a switch configuration on the chip needs to be found. For small neurons the switch configuration can be performed manually, but for rather complex morphologies the manual approach is tedious and might even be unfeasible, since the mapping is non-trivial and time-consuming regarding the number of switches that need to be set.

To solve the issue of configuring the hardware switches to form a neuron consisting of multiple compartments, three algorithms are developed and implemented: a brute force algorithm, an evolutionary algorithm and a constructive algorithm following a set of rules. These are tested with randomly generated neurons to determine logical limitations and optimization possibilities.

While the brute force approach fails for the placement of larger neurons because of its scaling and therefore did not prove useful, the evolutionary and rule based approach both manage to place a significant amount of neurons. The rule based algorithm is superior in execution speed whereas the evolutionary algorithm is more flexible in terms of neuron topology. The success of theses two algorithms opens up new possibilities for the use of BSS-2.

Zusammenfassung

BSS-2 ist ein mixed-signal neuromorpher Chip, der die Emulation von neuronalen Netzen ermöglicht. Neuronale Netze werden aus elektrischen Neuron-Schaltkreisen, die teilweise die Funktion biologischer Neuronen replizieren, und synaptischen Verbindungen konstruiert.

Die Schaltkreise auf dem BSS-2-Chip sind in einem 2D Raster angeordnet. Um Multicompartment-Neuronen darauf verwenden zu können, muss eine Übersetzung des abstrakten Modell-Neurons in eine Konfiguration der Schalter auf der Hardware gefunden werden. Schalter einzeln zu setzen, ist für komplexe Neuronen zeitaufwändig und führt dazu, dass wenn keine triviale Lösung erkennbar ist, das Neuron nicht auf der Hardware verwendet werden kann.

Um das Problem der Konfiguration der Schalter auf dem Chip zu lösen, werden drei Algorithmen entwickelt und implementiert: eine Exhaustionsmethode, ein evolutionärer Algorithmus sowie ein konstruktiver Algorithmus, der Regeln folgend eine Lösung konstruiert. Die Algorithmen werden mit Hilfe zufällig generierter Neuronen getestet, um logische Beschränkungen und Optimierungsmöglichkeiten zu bestimmen.

Die Exhaustionsmethode gelangt für größere Neuronen an seine Grenzen, was auf die schlechte Skalierung des Algorithmus zurückzuführen ist. Sowohl der evolutionäre als auch der regelbasierte Algorithmus sind erfolgreich bei der Plazierung unterschiedlicher Neuronen. Dabei zeichnet sich der regelbasierte Algorithmus besonders durch seine schnelle Ausführung aus, während der evolutionäre eine höhere Flexibilität gegenüber Neuronentopologien aufweist. Diese Algorithmen eröffnen neue Möglichkeiten für die Verwendung von BSS-2.

Contents

1 Introduction **2**

2 Methods **3**

2.1 Multicompartment Neurons . . . 3
2.2 Hardware . . . 3
2.3 Algorithmic Concepts . . . 3
2.3.1 Brute Force . . . 5
2.3.2 Evolutionary Algorithms . . . 5
2.3.3 Greedy Algorithm . . . 6
2.3.4 Backtracking . . . 6

3 Results **7**

3.1 Neuron Abstraction Model . . . 7
3.2 Hardware Resources . . . 7
3.3 PyNN API . . . 8
3.4 Placement . . . 8
3.4.1 Brute Force Algorithm . . . 8
3.4.2 Rule-based Algorithm . . . 10
3.4.3 Evolutionary Algorithm . . . 15

4 Discussion **25**

5 Outlook **28**

1 Introduction

The analog mixed-signal neuromorphic hardware platform BrainScaleS-2 (BSS-2) emulates neural networks, partially recreating the working principle of biological neural networks. Each BSS-2-chip offers 512 neuron circuits that each replicate the dynamics of the AdEx neuron model [4]. These can be connected via synapses on the chip to form neural networks. The chip can be used to create Spiking neural network (SNN) as well as Artificial neural network (ANN) [11].

While biological neurons have a spacial structure consisting of structures like dendrites, axon and soma, the neuron circuits on BSS-2 have no spacial extent. For many experiments the point neuron models are sufficient, however some benefit from spatially structured neurons.

Spacial neurons have applications in the emulation of biological processes since their behaviour is more appropriate than point-neurons, but also for computational purposes spatially extended neurons are useful [9]. The structure of the dendrites can be used to pre-process a synaptic input before it reaches the soma. Another application is the coincidence detection between two synaptic inputs. Therefore, neurons with bipolar dendrites fire strongly when the synaptic inputs on the dendrites happen with a specific delay and much more weakly otherwise [9]. Another advantage in the use of multicompartment-neurons is the possibility to establish different learning rules on different parts of one neuron [3].

BSS-2 offers the possibility to emulate neurons with multiple compartments rather than simulating their behaviour. This leads to a reduction of computational cost. Spatially extended neurons can be realised on the BSS-2 hardware, by short-circuiting multiple neuron circuits to create a compartment. These compartments represent a part of the biological neuron and can be connected to each other via resistors. The conductance of the resistor influences the signal in the neuron, as the spacial structure in a biological neuron would.

The BSS-2 chip has its neuron circuits placed in two rows of 256. On the top and the bottom of these rows the synapse arrays are placed. The chip is divided in half vertically by routing structures. Therefore, a spatially extended neuron can only use the left or right half of the chip. The connections mentioned above are established by five switches in each neuron circuit. These allow to short circuit multiple neuron circuits to form a compartment and connect multiple compartments via resistors.

Setting these switches manually is sufficient for smaller neurons, but becomes challenging and time-consuming for larger neurons. Therefore, an algorithm which performs the switch configuration is developed. Three types of algorithms are implemented and will be discussed in the following. A brute force approach, a evolutionary approach and a constructive algorithm following a set of rules. To test the algorithms for the success rate of the placement and the execution time a test with randomly generated neurons is used.

2 Methods

2.1 Multicompartment Neurons

A neuron model can be divided into multiple compartments each serving a computational purpose, e.g. preprocessing data in dendritic structures [10], or emulating biological behaviour better than point neurons. These compartments are formed by connecting multiple neuron circuits on the BSS-2-chip. A compartment can use the shared resources of all connected circuits. These compartments are interconnected via resistors which create a spatially extended structure.

2.2 Hardware

The BSS-2 chip has 512 neuron circuits, placed in two rows of 256 circuits [11]. This pattern allows for synaptic input from synapses from the top for the top row and the bottom for the bottom row. The chip is separated in two halves vertically, see Figure 1. For the placement of the multicompartment-neurons a 2×128 grid is therefore available.

For each neuron-circuit five switches, see Figure 1, can be used to connect two circuits in two different ways.

A direct connection via the two inner switches allows to short circuit two neuron-circuits creating a single compartment. This allows to form compartments of different shapes if required for the placement and to use the resources of multiple neuron-circuits to fulfil the model requirements for a compartment. For example by connecting two neuron circuits with these two switches the total capacity of the compartment can be increased, since the membrane capacitances are connected in parallel. This type of connection will be called internal connection in the following.

The second type of connection uses the shared line of the chip to create a connection between two compartments. There are two shared lines on the chip, one in the top row and one in the bottom row. Each neuron circuit has three switches to use the shared line. It can connect to the shared line with a switch or via a resistor. The shared line can be interrupted or connected to the right. Therefore, a connection between two compartments can be created by letting one compartment connect one, or multiple neuron-circuits to the shared line directly and the other compartment connecting with one or multiple compartments to the shared line via a resistor. This type of connection will be called external connection in the following. An external connection between neuron circuits can only be formed in one row.

2.3 Algorithmic Concepts

For the placement of the neuron models different algorithmic approaches are taken. The basic concepts of the implemented algorithms are presented in the following.

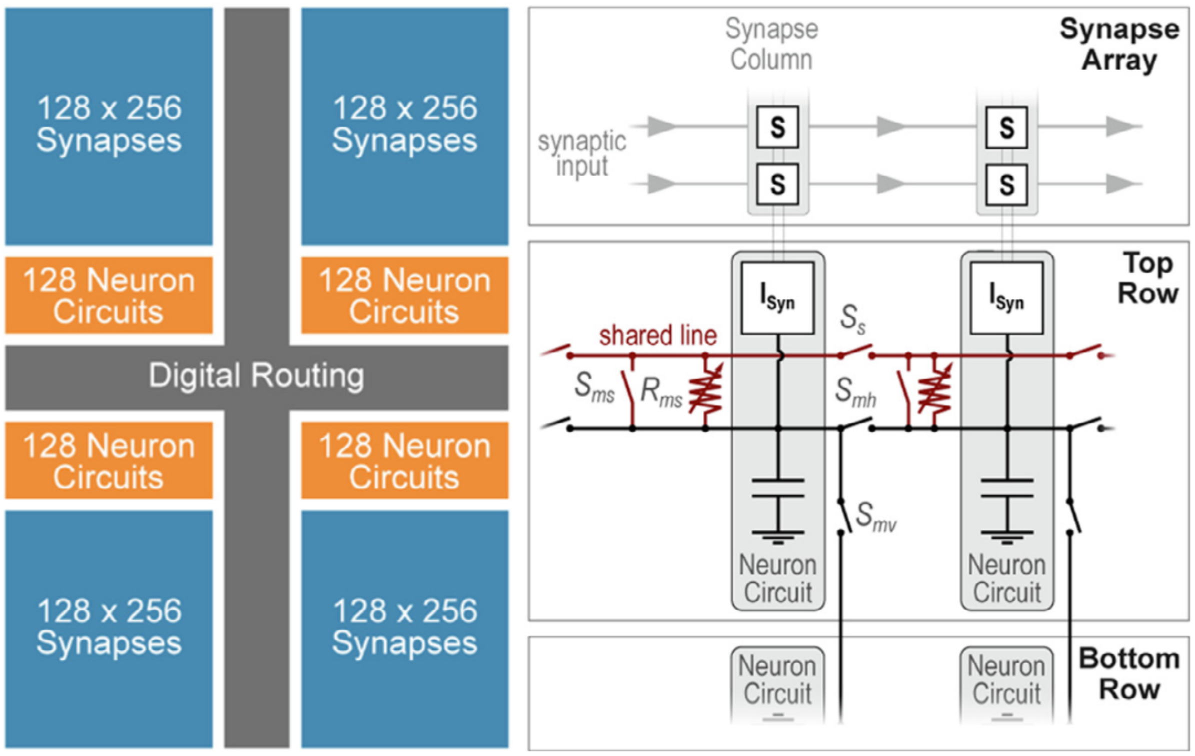


Figure 1: Left figure shows the basic structure of the BSS-2-chip. The two rows of 128 neuron circuits on one half of the chip can be connected to each other, but no connection between the two halves of the chip are possible because of the routing structure dividing the chip vertically. The right figure shows the five switches for each neuron circuit that can be used to connect two circuits, taken from [8]. The inner switches S_{mh} and S_{mv} are used to short circuit tow neuron circuits. The external switches S_{s}, S_{ms} and R_{ms} are used to connect two compartments.

2.3.1 Brute Force

Brute force algorithms solve problems by trying out every possible solution and checking for validity.

If a problem is solvable an algorithm always finds a correct solution when following the brute-force approach. Brute force algorithms are useful when applied to problems with few possible states, since the time complexity of the algorithm is in $O(n)$ where n is the number of possible states. If no further information about the correct state exists, in the worst case every possible state needs to be checked for validity.

2.3.2 Evolutionary Algorithms

An evolutionary algorithm follows the concept of biological evolution. Therefore a population of individuals goes through multiple generations where each individual undergoes selection, mating and mutation in each generation [13]. Each individual is represented by a genome, which in this case represents the states of the switches on the chip. For the selection an ordering of the individuals is required which implies how well it solves the given problem. This ordering is given by the so-called fitness of the individuals. The fitness indicates how close a solution is to being correct. Calculating the fitness to a result where the result is not explicitly known often poses the main difficulty when working with evolutionary algorithms.

During the selection stage a number of individuals are selected, to form a new population. Different procedures are available for this purpose. For example, the better half of the individuals is selected and copied twice to form the new population. Another selection procedure is the tournament where multiple randomly selected individuals out of the population compete against each other and the better fitting individuals proceeds to the next generation. The tournament is repeated N times, where N is the number of individuals in the old population. This results in a new population of the same size as the old one. With the tournament it is possible that a better fitting individual has multiple children in the next generation, and that bad fitting individuals proceed into the next generation, when competing against other worse fitting individuals. Bad fitting individuals serve the purpose of having a heterogeneous population which allows for different approaches on a valid solution. The rate of good fitting individuals in the next generation is influenced by the number of contestants during the tournament, since for a higher number of contestants the chance that a well fitting individual wins the tournament is higher.

During the mating stage two randomly selected individuals have a certain chance to interact with each other's genome. For example, one exchanges parts of its genomes with the other. This step allows combining two individuals that both have a good partial solution to one with a overall better solution and one with a possibly worse solution than before.

The last stage in each generation is the mutation. During the mutation one or multiple changes are randomly applied to a number of individuals. The most basic type of mutation is a random flip of a part of the bits in the genome of an individual. Different types of mutations can be applied depending on the structure of the genome of each individual.

After performing each of the stages mentioned above the fitness of each individual of the new population is calculated and a new generation starts. To achieve a valid result the average fitness of the population should be increasing in each step. Good fitting individuals have multiple copies of them in new generations with mutations or mated with other individuals. Therefore, on average the fitness of the best individuals of the population is increasing in each generation.

To prevent a mutation that makes the best fitting individuals in the population worse, a concept called "hall of fame" can be used to ensure a certain number of best fitting individuals proceeds to the next generation without participating in the two alternating stages mentioned above [6].

2.3.3 Greedy Algorithm

Greedy algorithms try solving problems by finding locally optimal partial solutions [7]. When problems can be divided into steps that are independent of each other greedy algorithms can find the optimal solution as a combination of locally optimal solutions. However often the single steps are not independent of each other and therefore a locally optimal solution can result in a bad overall solution or a state where no solution can be found.

For the placement of multicompartment neurons, local solutions are the placements of single compartments that have a optimal usage of resources but may block off connections between other compartments and therefore prevent a valid solution for the placement of the complete neuron.

2.3.4 Backtracking

A backtracking algorithm tries to find solutions to a problem in multiple steps. If the algorithm detects, that its current state can not lead to a valid solution the state is abandoned and the algorithm goes back n steps [14]. The algorithm than chooses another option for the next step as the last time. The number of steps n that the algorithm goes back in the solving process depends on the problem to solve. Typical choices are going back one step, which equals a depth first search or going back to the first step, which equals a breadth first search.

3 Results

3.1 Neuron Abstraction Model

The implementation of the neuron abstraction model was performed during a preceding internship by the author and will be shortly described in the following [2].

The neuron abstraction model allows for intuitive construction of a multicompartment neuron. The model consists of three main parts, that are hierarchically ordered. The neuron is implemented as a graph which contains its compartment as vertices and the connections between the compartment as edges. The compartments contain mechanisms, which correspond to hardware configurations e.g. the membrane-capacitance of a compartment, see Figure 2.

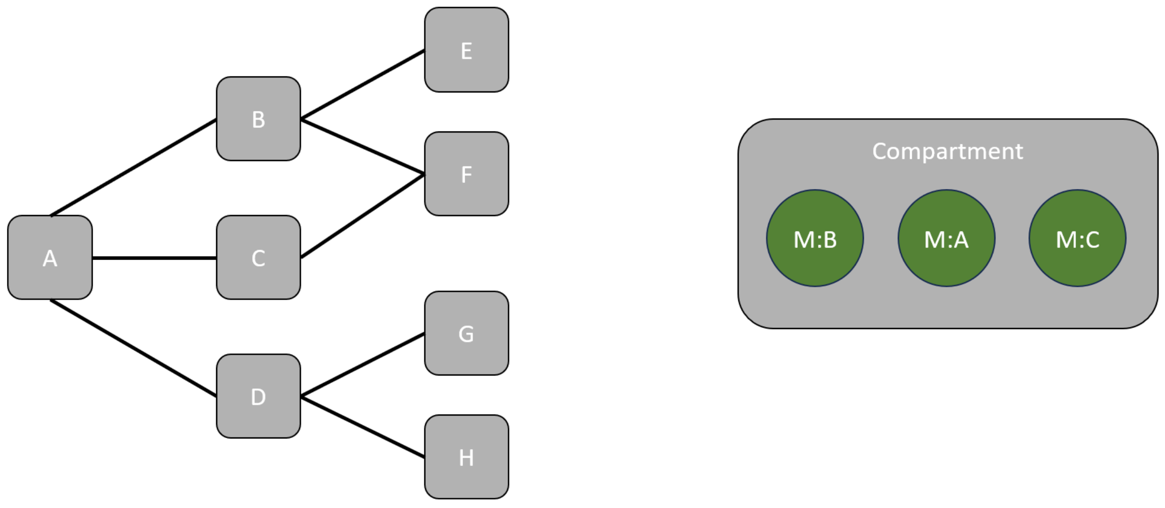


Figure 2: On the left hand an abstract neuron model in its graph representation with multiple compartments. On the right hand a single compartment containing its mechanisms.

The neuron model described above is the only required user input for the placement algorithm to work. No further information by the user is required. The created abstraction model was designed to not only work as an abstraction step for the specific problem of multicompartment-placement, but is generic and can be used in different use cases and on different hardware.

3.2 Hardware Resources

The implementation of the hardware resource determination step was performed during a preceding internship by the author and will be shortly described in the following [2].

To perform the placement of the multicompartment neuron, the given model parameters, e.g. membrane capacity, are not sufficient. Instead, the number of neuron circuits on the BSS-2 chip are required for the algorithm to perform placement.

To determine the hardware resources, the model parameters of the mechanism of each compartment are converted into the number of required neuron circuits. This information is provided during the placement step for each compartment. The required resources can be specifically linked to the top or the bottom row if required. This feature allows compartments to have guaranteed synaptic input from both the upper and lower synaptic array which can be required depending on the experimental setup.

3.3 PyNN API

PyNN is a Python library often used for experiments in computational neuroscience. To make the use of multicompartment neurons on BSS-2 accessible in a similar fashion as in the PyNN library [12] an Application programming interface (API) is created.

The API allows creating neurons of the abstraction model by defining their mechanism and adding them into a compartment builder, that returns classes of compartments. This allows to create templates of compartments, e.g. a spiking mechanism. These compartments are added to the morphology builder and then interconnected to form a neuron. The morphology builder returns a class of a neuron, which can again be used to create templates. This neuron is then converted to the abstract neuron described in section 3.1 to perform the placement step.

The neurons generated in Python can be used to create PyNN populations, with which experiments can be performed.

3.4 Placement

The placement of multicompartment-neurons is trivial for small neurons without complex structure. For increasing size and complexity the placement becomes increasingly challenging. The configuration of the chip consists of five switches for each of the 256 neuron circuits on each half of the chip. Setting them by hand is not sufficient for larger neurons and a automation of the placement is required. Therefore, three different algorithms are implemented and presented in the following.

3.4.1 Brute Force Algorithm

The brute force algorithm tries every possible combination of the five switches of each neuron circuit of the BSS-2 chip until a correct result is found, see Figure 3. This approach has the advantage that for every placeable neuron a solution is found. The disadvantage of the brute force concept lies in the number of possible combinations that need to be tested. One half of the BSS-2 chip has a grid of $2 \times 128 = 256$ neuron circuits and each of the circuits has five switches with two states each. Therefore, theoretically there are $(2^5)^{256} \sim 2 \cdot 10^{385}$ possible states for the setup. Trying every possible configuration is therefore not feasible.

The number of possible states of a neuron circuit in general are $2^5 = 32$. Since some states are not valid these states are not tested in the brute force algorithm. It is not needed to test for states where the neuron circuit is connected to the shared line directly and with a resistance at once, since the direct connection short circuits the resistor. This subtracts 8 possible states which leaves 24 combinations per neuron circuit and leads to a total of $24^{256} \sim 2 \cdot 10^{353}$ possible combinations for the whole coordinate-system of neuron circuits.

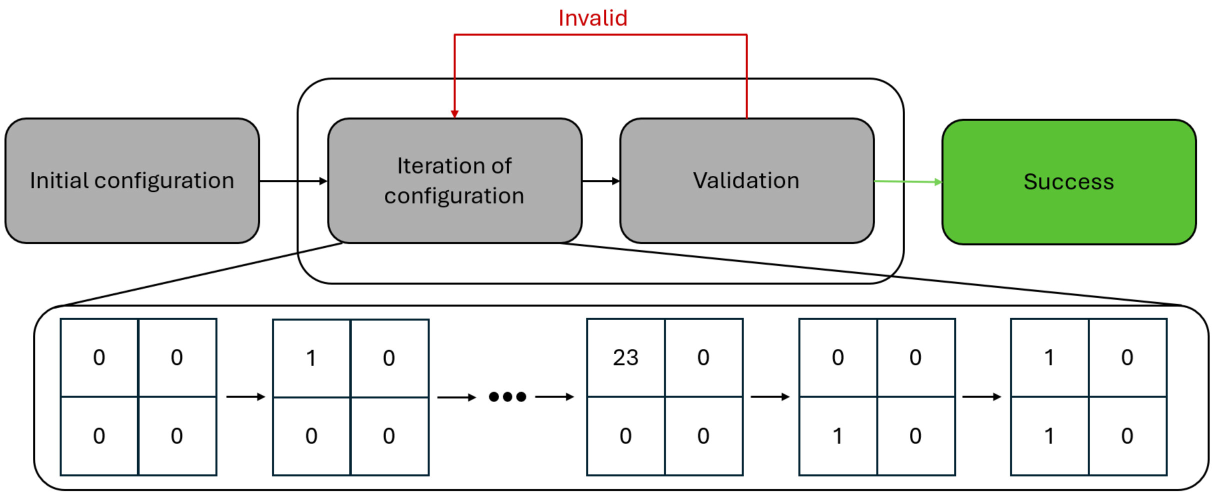


Figure 3: Working principle of the brute force algorithm.

The implementation of this algorithm iterates through all possible states of the switches of each neuron circuit in the coordinate system and checks after each iteration if the current state is a valid solution. Starting at the left end of the grid, it firstly iterates through all states of the neuron circuit in the top row. If all valid 24 combination of switches have been tried out, the state of the first neuron circuit is set to zero, which equals no switches set, and the second neuron circuit, in the first column in the second row is iterated by one. Then all iterations of the first neuron circuit are repeated. This can be continued until the maximum size of 256 used neuron circuits (128×2) are reached.

The validation of the current state consists of different steps which aim to detect a wrong solution quickly. First the coordinate system is scanned to detect compartments. This is done by finding neuron circuits that have a state unequal to zero, which means they have switches set, and therefore are connected to other neuron circuits. The validation performs the following checks on the possible solution in the given order since they have increasing computational costs and therefore aim at a fast invalidation through computational cheap checks.

1. Number of compartments: compares the number of detected compartments with the number of compartments in the target neuron.
2. Number of compartment connections: compares the number of detected compartment-connections with the number of compartment-connections in the target neuron.
3. Isomorphism of neuron constructed from current solution and target neuron.
4. Minimal number of neuron circuits per compartment.

Once a valid solution is found, the algorithm assigns the compartment-IDs of the target neuron to the corresponding neuron circuits on the coordinate system. The conversion happens through a mapping of the compartment-IDs created during the validation and the compartment-IDs of the target neuron.

For larger neurons, finding a valid solution for the placement takes many iterations of the algorithm and is therefore time-consuming. The algorithm has a time limit after which it is terminated without success and without information if the neuron model is placeable on the hardware. Neurons with up to four compartments have successfully been placed using the brute-force algorithm.

Since multiple states of the coordinate-system can be tested in parallel, it is useful to run the brute force algorithm in parallel by creating blocks of configurations. This happens by iterating the state of a neuron circuit further on the right, e.g. the fifth which creates blocks of configurations with 24^4 configurations, that can be tested in parallel.

This feature is implemented by creating s starting states, where higher-valued circuits have a state unequal zero. Then these s states are validated and, while no solution is found, iterated as in the single-threaded case. The time required to test for n possible states of the coordinate system is therefore reduced by the factor s. The parallelisation is done on a machine with 64 cores therefore $s = 64$ is chosen.

When performing the brute force approach on the whole coordinate-system with the parallelisation explained above, this leads to $\frac{24^{256}}{64} \sim 3 \cdot 10^{351}$ successive iterations.

The time a single iteration of the algorithm, including the validation which requires the most time, was measured after several optimisations to be in the order of 20 ns. With this the placement of a more complex neuron as shown in Figure 13, with 34 circuits that need to be partially iterated, would take the algorithm $1.8 \cdot 10^{33}$ years without the parallelisation.

In the time from the creation of the universe to now the algorithm would be able to iterate up to 19 circuits at the given speed [5]. For the algorithm to perform in a suitable time of 10 min for the mentioned complex neuron, 10^{38} parallel runs would be required. Therefore, the brute-force algorithm as a solving strategy for the placement is discarded. For smaller neurons the solution is rather trivial and other algorithms can find a good to optimal solution and for larger neurons the brute-force approach can not find a solution, in a suitable time span.

3.4.2 Rule-based Algorithm

A basic version of this algorithm was developed during the previous internship [2]. During the bachelor thesis some functionalities were modified to increase the amount of placeable neuron models. At the end of the internship the algorithm was capable of placing neurons with up to four adjacent compartments.

The basic concept of this algorithm can be classified as a sequential greedy algorithm. The algorithm picks the most complex compartment of the model-neuron and places it at the center of the coordinate system. It then places compartments, which are connected to this central compartment in the model neuron, adjacent to the central compartment on the coordinate system, see Figure 4, following given rules. Examples for rules given to the algorithm are the following

- The compartment with the highest number of connected compartments is placed in the center

– The neighbour with the highest number of connected compartments is placed before neighbours with fewer connections

– The default placement happens in the top row of the coordinate system. If the top row is full, compartments are placed in the bottom row.

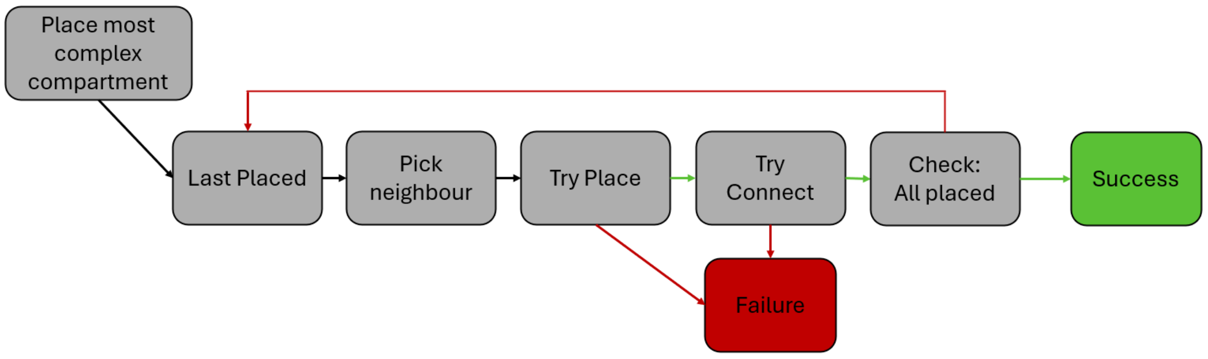


Figure 4: Working principle of the rule based algorithm.

With the concept of rules for the algorithm the amount of neurons placeable with the algorithm is small at the beginning of the development since only fundamental rules are known. During the development model neurons with increasing complexity in terms of size, shape and number of connections, are tested. If the test for a model neuron fails a rule is implemented to allow the placement of these neurons. For neurons with a higher number of compartments the number of possible topologies increases. Therefore, testing each neuron topology by hand is not suitable and the algorithm is tested with randomly generated neurons. The individual steps of rule implementation are described in the following.

As a first step the rule-based algorithm assigns the descriptor of a compartment to one neuron circuit. This represents a neuron consisting of one compartment with the requirement for one neuron circuit. This is the most fundamental assignment which does not include multicompartment neurons. See the result Figure 5.

Next a compartment with the requirement for multiple neuron-circuits is placed. Therefore, an internal connection between two neuron-circuits is established by setting the switches to short circuit these two compartments, see Figure 6.

In the next step a neuron with two compartments that are connected to each other is placed. Therefore, the algorithm gets a rule on establishing a external connection between two neuron circuits belonging to different compartments, see Figure 7.

A more complex structure is a chain of compartments, where each compartment is connected to a maximum of two other compartments. This can be realised by placing the compartments in the top row adjacent to each other and establish a external connection between them, see Figure 8.

If a neuron contains a compartment that is connected to more than two other compartments the placement pattern of the compartment chain is no longer sufficient. Therefore, the compartment with more than two connections needs to change its shape to a two times two grid to be able to connect to up to four compartments, see Figure 9 and Figure 10. The choice of the shape is not the only possible. A solution with fewer neuron circuits would be, to allocate one circuit in the top and one in the bottom row, however this pattern requires for a more complex treatment of the connections between the compartment to ensure that two compartments are not short-circuited via the shared line. Therefore, the solution using the two times two shape is used to simplify the connection process.

For some cases, especially for synaptic inputs, a further restriction on the location of the neuron circuits of a compartment is required. Therefore, it can be specified to place compartments in the top row or in the bottom row. An example can be seen in Figure 11.

To increase the maximal amount of connections per compartment a bridge-shaped structure that has inlaying chains is implemented. At this state the algorithm can construct compartments with arbitrary number of connected compartments. However, the structure of the connected compartments and their neighbours is limited. Only two of the neighbours can branch into multiple neighbours. One of them is placed on the right of the bridge-structure one on the left. Additionally, the amount of compartments with synaptic input from both directions is limited due to their property to occupy both the top and the bottom lane of the coordinate system, making it impossible to place more than one on each side. See an example in Figure 12.

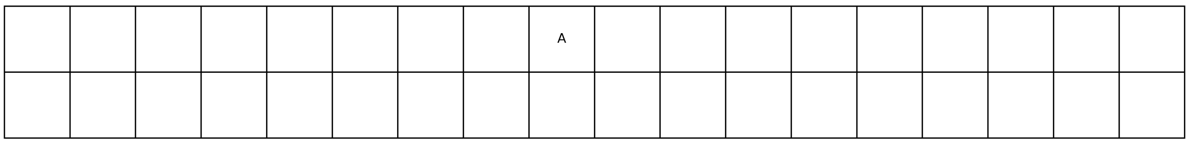


Figure 5: Placement of a single compartment with requirement for one neuron circuit.

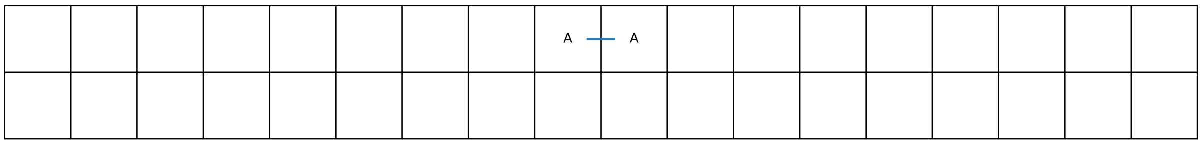


Figure 6: Placement of a single compartment with requirement for multiple neuron circuit.

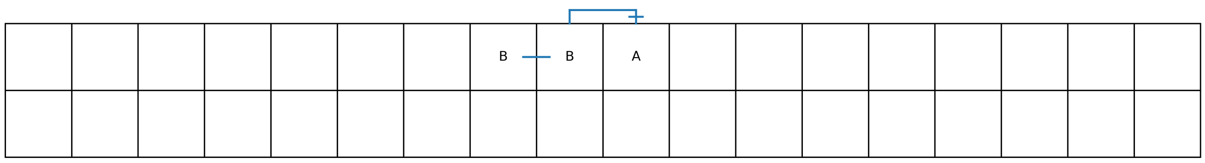


Figure 7: Placement of two compartments of different size that are connected to each other.

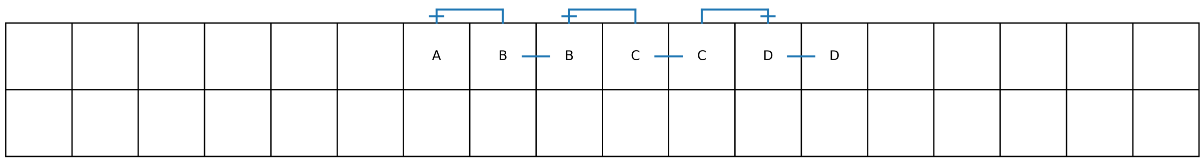


Figure 8: Placement of a chain of compartments.

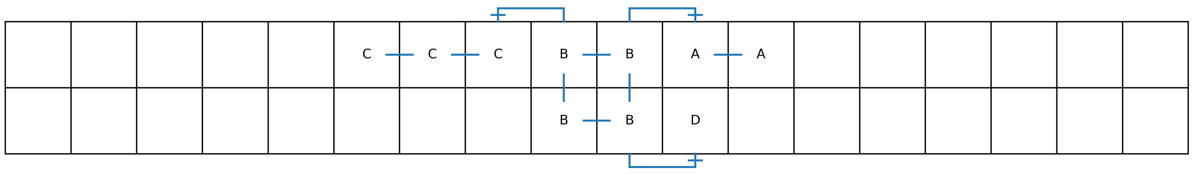


Figure 9: Placement of a compartment which needs to change its shape to allow for connections adjacent compartments.

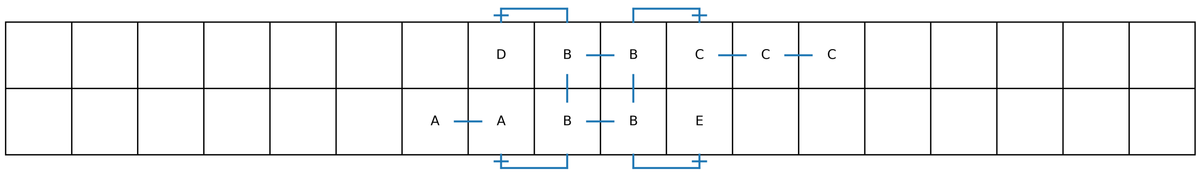


Figure 10: Placement of a compartment with four connected compartments.

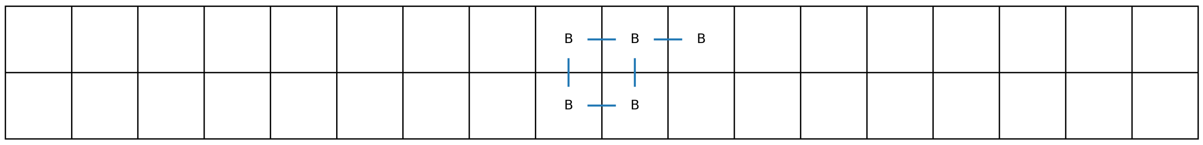


Figure 11: Placement of a compartment with specified synaptic input.

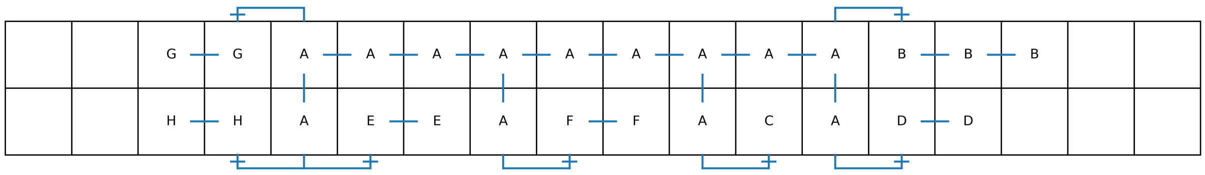


Figure 12: Placement of a compartment with seven connected compartments via a bridge structure.

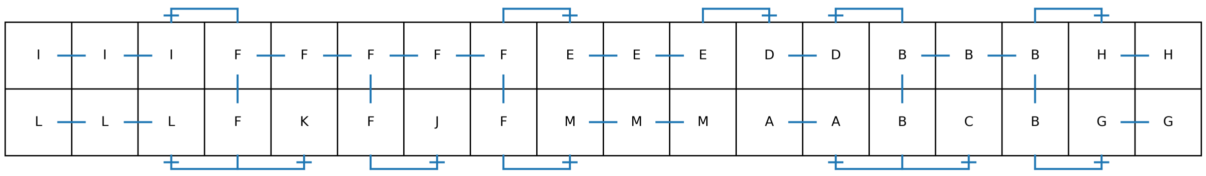


Figure 13: Placement of a neuron with a combination of different topological structures. Compartment F creates a bridge structure to connect to seven compartments. Compartment D increases its size to allow two connections. Compartment B creates a bridge structure to connect to five compartments. Compartment E allocates a third neuron circuit not required by the minimal resource requirements or the number of connections but to allow for the placement of compartment M in the bottom row.

Evaluation

The rule-based algorithm is tested regarding to the execution time and the rate of successful placements. Therefore, random neurons are generated with a given number of compartments and compartment connections. The algorithm then performs the placement. During the test the time required for the placement of one neuron as well as the success rate of the algorithm is measured over the iterations of the test.

The rule based algorithm has logical limitations for placeable neuron topologies. Therefore, when finding a logical error in placement the algorithm aborts and a fail is recorded.

The test can be executed with cyclic or acyclic neurons. For the acyclic neurons only $m = n - 1$ compartment-connections are possible (with n the number of compartments). For $m > n - 1$ a cycle would occur and for $m < n - 1$ the neuron would not be connected (not every compartment has a path to every other compartment on the neuron). If cyclic neurons are allowed by the test the number of compartment-connections is within the range $n \leq m \leq \frac{n \cdot (n-1)}{2}$.

Since the rule-based algorithm can only connect loops if the compartments are placed adjacent in one row by coincidence, limiting the test to acyclic neurons made the test results more representative. Another reason to limit to acyclic neurons is the fact, that in many cases where biological neurons are reviewed these neurons have a acyclic structure [1]. The results of the test for the rule-based algorithm in its current state and its state at the end of the internship are displayed in Figure 14. The testing shows that during this work the amount of placeable neurons has increased. The execution time of the algorithm, see Figure 15, has increased for higher numbers of compartments which results from the implementation of the backtracking. Overall the execution time of the algorithm depends linearly on the number of compartments. The peaks for higher number of compartments result from the backtracking which can require the algorithm to place compartments multiple times with different allocated resources.

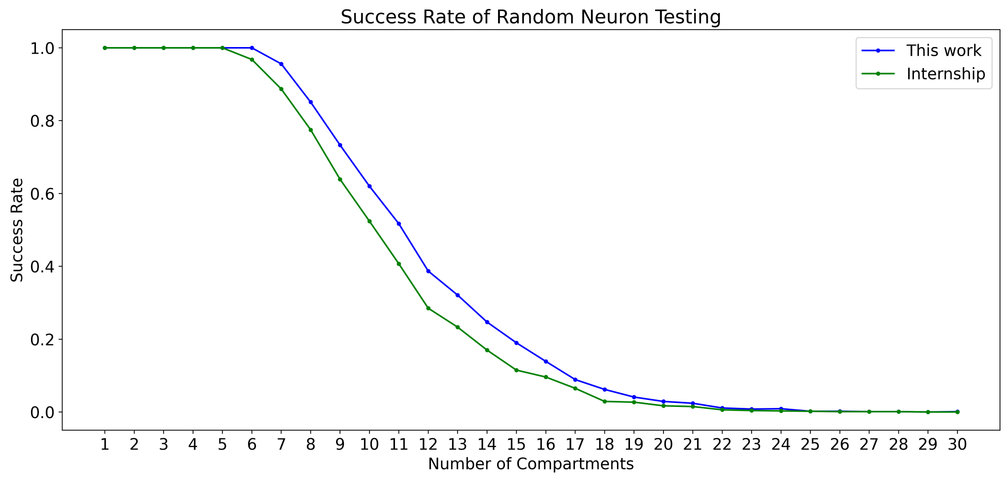


Figure 14: Testing of the ruleset algorithm for random neurons at its current state and its state at the end of the internship. The algorithm is tested with 1000 randomly generated neurons for each neuron size.

The topology of the neurons failed to be placed are used to determine future changes on the

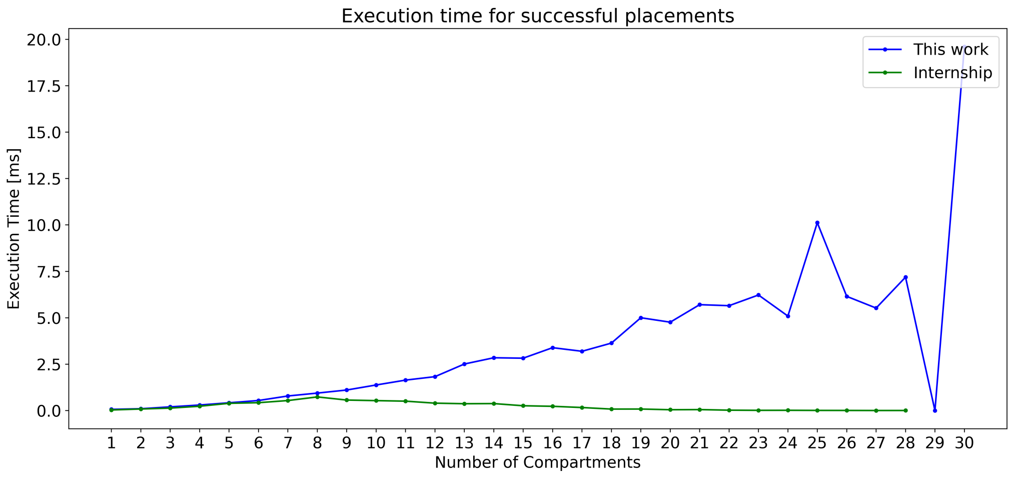


Figure 15: Testing of the ruleset algorithm for random neurons at its current state and its state at the end of the internship. The algorithm is tested with 1000 randomly generated neurons for each neuron size.

algorithm. The test was therefore repeated multiple time to accomplish the changes mentioned above.

During the testing phase limitations to the current state of the algorithm are recognized. The random neuron testing failed for some neurons that have different topology but share substructures that cause the algorithm to fail. An example for two failing neurons with eight and twelve compartments are shown in Figure 16. The failure of the algorithm for these two similar neuron topologies results from a wrong order of placed compartments during the placement step. Another reason for the failed placement is the inability to treat leaf compartments and chains, which are single compartments and compartments connected to a maximum of two other compartments, differently than branches that split up in multiple chains or leafs. Since the algorithm only places compartments adjacent to already placed compartments, for some combinations placement slots can get blocked.

This can be solved by either using backtracking and changing the placement order or by allowing changing the shape of already placed compartments. The second issue of leaf compartments and chains can be fixed by giving the algorithm more knowledge over the structure of connected branches instead of only the next connected compartment during the placement. When taking substructures into consideration leafs and chains can be placed differently, so they are not blocking for other more complex structures. A plot of a failed placement, due to the described issue, and a possible fix is shown in Figure 17.

3.4.3 Evolutionary Algorithm

The evolutionary algorithm developed uses a probabilistic approach to find valid solutions for the placement of the multicompartment neuron models. The algorithm creates a population of individuals which undergo multiple changes and then form a new generation. This process is repeated until a solution is found or a specific time limit is exceeded, see section 2.3.2.

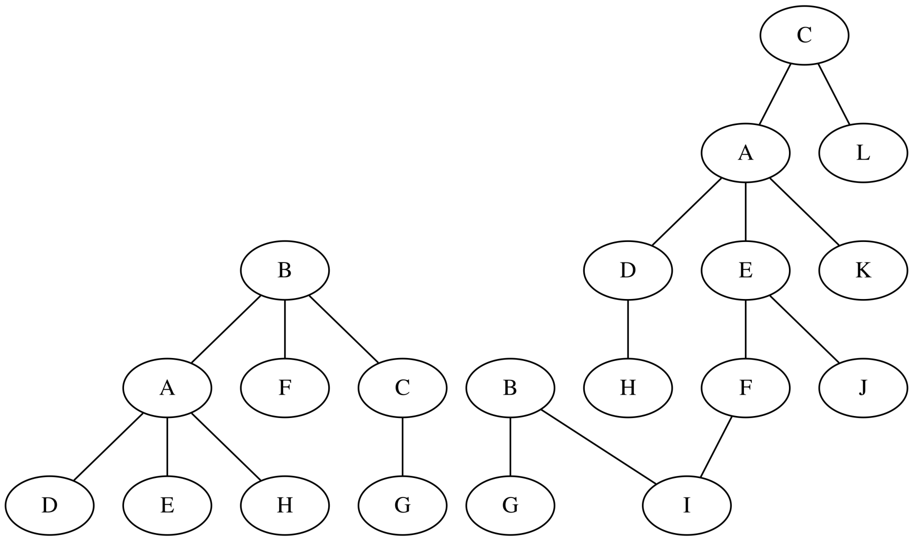


Figure 16: The neuron on the left with 8 compartments failed to be placed, because the compartments have been placed in the wrong order. Compartments with more connections get placed before compartments with lower numbers of compartments possibly blocking neuron circuits on the chip. The neuron on the right with 12 compartments failed to be placed because the algorithm cannot handle leafs or chains of compartments differently than splitting branches. Therefore, they are placed equally, which blocks adjacent neuron circuits on the hardware. This leads to a failure in placement.

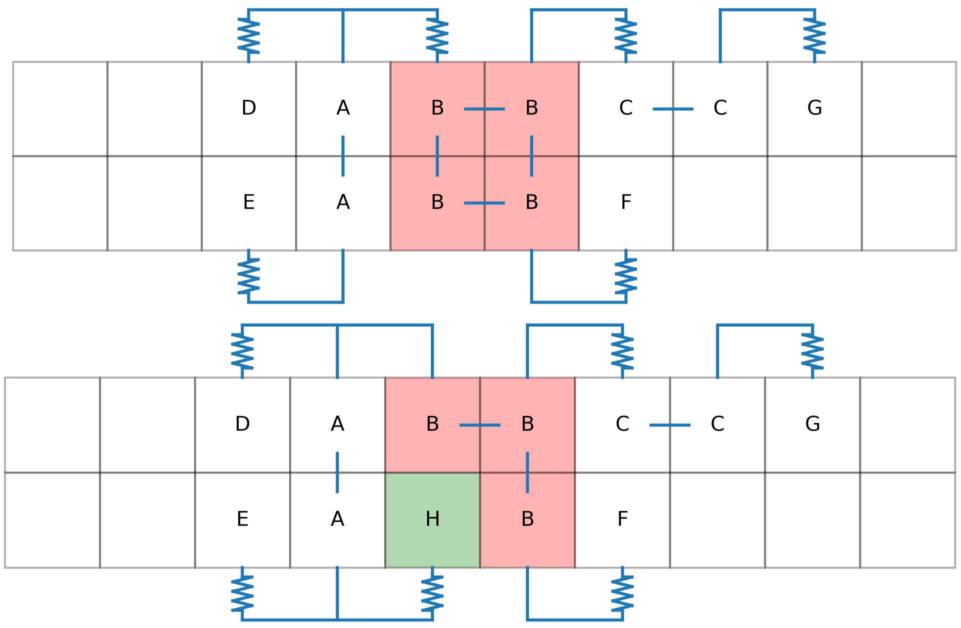


Figure 17: In the plot on top the placement done by the algorithm is shown in this case the compartment H can not be placed since it requires a free slot adjacent to compartment A. In the bottom plot a simple fix for this issue is shown.

The operations of each generation can be classified in four steps. First a selection step makes a selection of individuals from the last generation which continue to the new generation. In the second step mating between individuals happens, where parts of the genomes are exchanged. In the third step multiple different mutating operations are applied to the individuals in the population. As a last step the fitness of each individual of the population is calculated, see Figure 18.

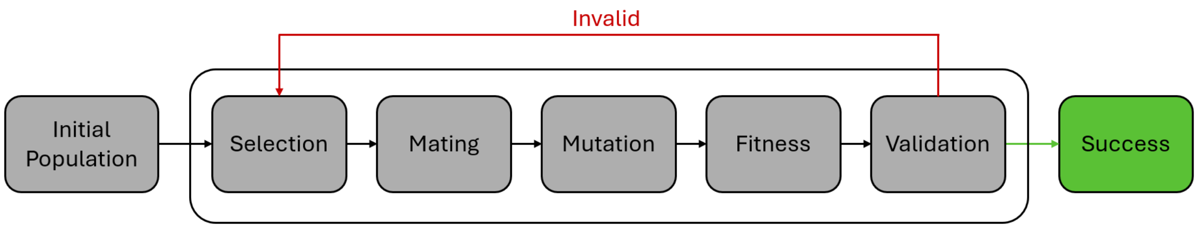


Figure 18: Basic working principle of an evolutionary algorithm.

The genome of each individual is a bit-string, where each bit represents one switch or the connection between the two rows of the coordinate-system, which is formed by two switches, see Figure 19. Therefore, each neuron circuit requires 4.5 bit to represent its switches. The genome for one half of the BSS-2-chip has a length of 1152 bits. It follows a alternating pair pattern so that two adjacent parts of the genome are also adjacent on hardware, see Figure 19. The population is an unordered list of individuals, each represented by its genome.

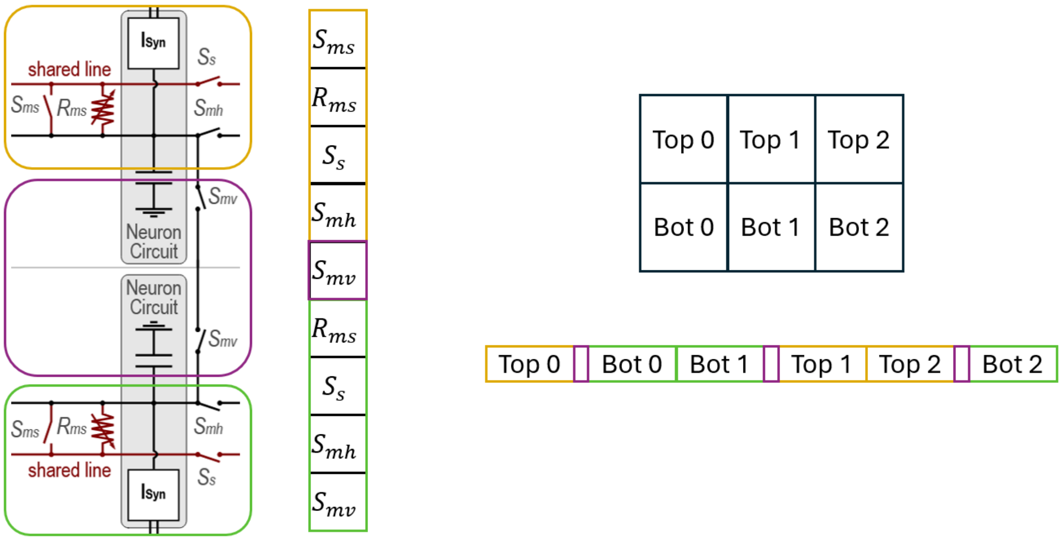


Figure 19: On the left the genome as a representation of the single switches of the neuron circuits on hardware is shown. The switches to connect the top and the bottom row are represented in a single bit since both need to be set to establish a valid connection. On the right the pattern in which the genome is formed is shown.

The first population is initiated with individuals where for each genome random bit flips are applied. During the testing a rate of $p_{\text{flip,inital}} = 0.8$ is found optimal.

Selection

The selection process happens in two steps, see Figure 20. First a number of individuals with the best fitness are selected, which will continue to the next generation unchanged. This so called hall-of-fame is used to prevent the mating and mutation steps happening later to manipulate the best individuals and making their fitness worse. Without the hall-of-fame concept it is possible that e.g. because of a high mutation rate each well fitting individual gets mutated to a worse fitting one. In the implementation the top five percent of the population were preserved in the hall-of-fame.

During the second step a selection of the size of the previous generation minus the number of hall-of-fame individuals is performed via a tournament. In a tournament multiple randomly drawn individuals compete against each other. The one with the highest fitness wins the tournament and proceeds to the new population. A higher number of contestants leads to more well fitting individuals in the new population. This possibly leads to changes in the right direction to find a valid solution but can as well cause a homogeneous population which does not try out enough different configurations and therefore is stuck with an invalid solution.

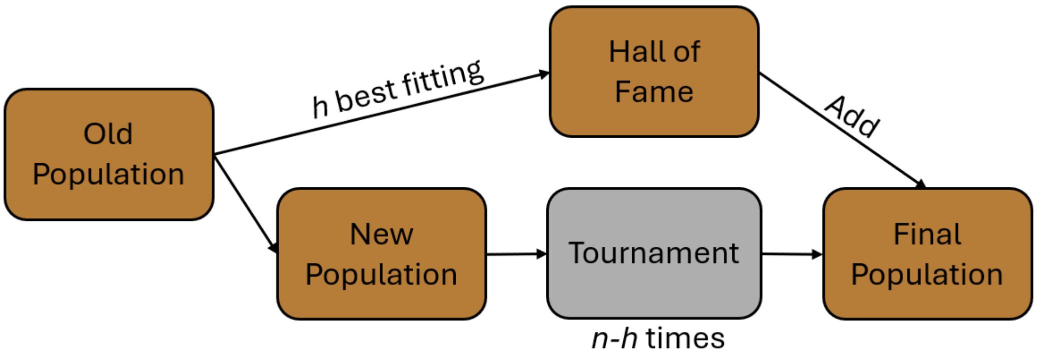


Figure 20: Selection step of evolutionary algorithm.

Mating

During the mating step two individuals exchange parts of their genome, see Figure 21. Therefore, two individuals are randomly selected and with a given chance randomly selected parts of their genomes are exchanged. This allows to combine solutions that are partially good. The mating can be improved by performing the mutational operations that change the genome in a way that the exchange of parts of it create a better solution.

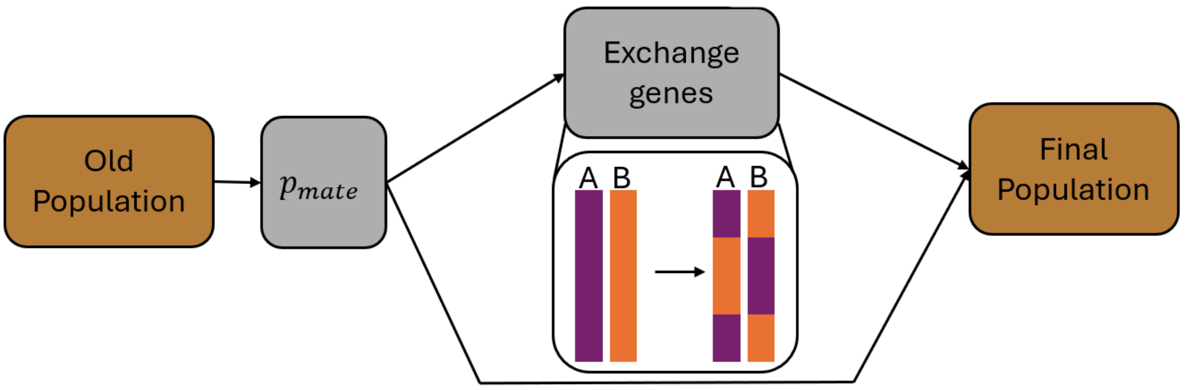


Figure 21: Mating step of evolutionary algorithm.

Mutation

Three different mutations are applied to the population during this step, see Figure 22.

Two different more directed types of mutations are implemented as well; a shifting operation and a mutation that adds and removes columns of the configuration.

The shifting operation translates the genome by a random distance in a specified range either to the left or to the right with a probability p_{shift}. The translation happens as a rotation, so circuits represented by bits in the genome that are shifted out of the structure on one end are inserted on the other end. This mutation can be beneficial for the mating of two individuals as it repositions partial solutions that can form a better solution when mating.

The mutation of adding and removing columns, with probability $p_{\text{add/remove}}$, of the configuration grid allows to change configurations in the middle quite easily compared to the mutation that flips random bits. To change the structure in the middle of a larger structure in the configuration and shift the existing well fitting parts further to the sides requires a lot of bit flips which first result in a worse fitness and are therefore unlikely to happen. This issue can be solved by inserting a blank column at a random position that can be mutated during the next mutation. The removal of columns serves the same purpose. By deleting blocking structures in the middle of the configuration new connections can be formed by the undirected mutation that have better fitness.

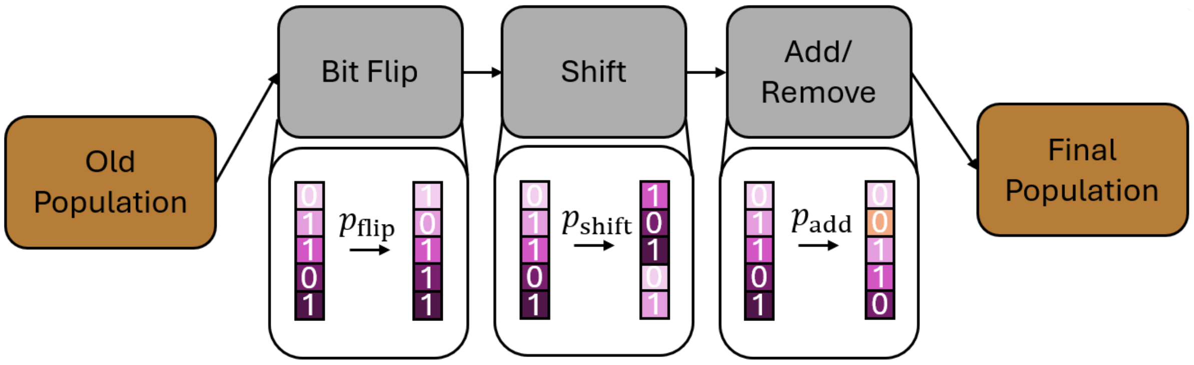


Figure 22: Mutation step of evolutionary algorithm.

Fitness

After the operations that manipulate the population and create a new generation of individuals the fitness of each individual is calculated. Therefore, multiple factors are taken into account. To calculate the fitness first the genome represented as a bit-string is converted into a coordinate-system and a model neuron is constructed by looking on the switches set in the coordinate system. This neuron is compared to the target neuron in terms of number of compartments, number of compartment connections, allocated hardware-resources and the largest subgraph-isomorphism.

The number of compartments and connections is simply compared to the number of required compartments and connections of the target-neuron. The higher the difference the lower the fitness from this parameter. $F_{\text{compartments/ connections}} = 1 - \frac{|N_{\text{target}} - N_{\text{allocated}}|}{\max(N_{\text{target}}, N_{\text{allocated}})}$

The fitness calculated by hardware resources uses the difference between allocated resources and required hardware-resources. The number of required resources is the minimum number of resources determined by the requirements of the mechanisms of the target-neuron. If the number of allocated resources is lower than the required the fitness of this parameter is between zero and one $F_{\text{resources}} = 1 - \frac{|N_{\text{target}} - N_{\text{allocated}}|}{N_{\text{target}}}$. If the number of allocated resources is higher than the required ones plus the additional resources allowed the fitness has a value between one and two $F_{\text{resources}} = 2 - \frac{|N_{\text{target}} - N_{\text{allocated}}|}{N_{\text{allocated}}}$. This distinction between more or less allocated resources than required is done because a solution with minimal number of hardware resources can not always be found. So the optimal solution can require more than the minimal number but should be still as small as possible. On the other hand with fewer resources no valid solution is possible.

The subgraph isomorphism method performs a mapping of compartment-identifiers from the neuron constructed out of the genome to the target-neuron. If a valid solution is found this mapping is complete. Otherwise the largest subgraph is determined. The fitness is then calculated by the difference of the number of compartments of the subgraph-neuron and the target-neuron. $F_{\text{isomorphism}} = 1 - \frac{|N_{\text{target}} - N_{\text{subgraph}}|}{N_{\text{target}}}$

Each comparison returns a value which is higher for neurons closer to the target neuron. These factors are weighted and added up to form the total fitness of an individual Figure 23 by $F = \sum_i w_i \cdot F_i = w_{\text{compartments}} \cdot F_{\text{compartments}} + w_{\text{connections}} \cdot F_{\text{connections}} + w_{\text{resources}} \cdot F_{\text{resources}} + w_{\text{isomorphism}} \cdot F_{\text{isomorphism}}$.

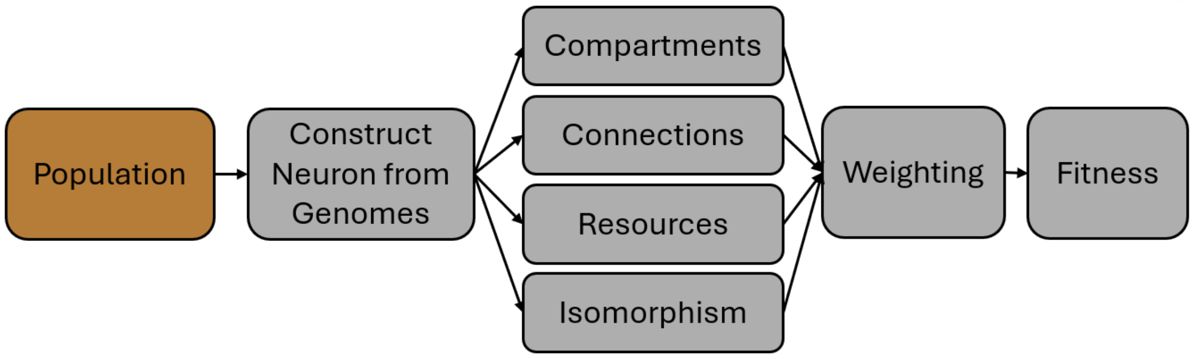


Figure 23: Fitness calculation during evolutionary algorithm.

During the development of the algorithm multiple other fitness values are taken into account but discarded later. The main goal of these values is to keep the growth of the result local and with minimal spatial extend to prevent disjunct solutions spread over the grid. Therefore, a fitness value that is larger for a lower number of switches set in total is introduced as well as a value that measures the distance between the neuron circuit with switches set on the furthest left and the circuit with switches set furthest on the right. This value is smaller for larger distances. Both these fitness values have the consequence that setting switches and having larger structures in the configuration results in a smaller fitness and therefore inhibits the growth of the configuration strongly so that no solution can be found.

Valid solutions

The operations described above are repeated over multiple generations. Therefore, the order in which the operations are performed is arbitrary. In the current implementation after each generation only the best fitting individual in the population is checked for validity since this check is computationally expensive, see Figure 24. However, this check is sufficient since the fitness is always higher for a valid individual with worse e.g. allocated resources, than for an invalid individual.

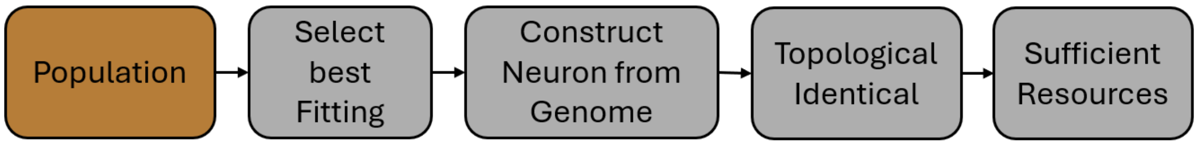


Figure 24: Validation in each generation of evolutionary algorithm.

Parameter optimization

The different operations applied to the population explained above have parameters like the mating probability or the mutation probability for different types of mutations. To find an optimal parameterization for the evolutionary algorithm, it is tested by different randomly generated neurons. The test is run for each of the parameterizations in parallel and the results are compared regarding to the success-rate which correlates to the execution time, since a failed placement is equal to reaching the run-time limit. Throughout multiple tests the following parameters have proven useful as they create the highest overall success rate. With multiple test runs with different combinations of run parameters the in total best performing parameterization can be seen in Table 1. Up to 128 different parameterizations are tested in parallel to distinguish whether some parts of parameterizations are superior to others and to find the best combination. A comparison between the success rates of multiple different parameterizations can be seen in Figure 25. It can be seen, that some parameterizations are overall superior to others. The number of tested combinations is limited since the algorithm requires a time limit of multiple minutes to achieve sufficient success rates and the testing therefore is time-consuming. Additionally after setting fixed parameters for the mutation and mating probabilities the population size is varied and different seeds for the random number generator are tested. The results of this testing steps are shown in Figure 26 and Figure 27. While for the population size an increase in the success rate for larger populations can be observed as well as a shorter execution time, no dependency between the seed for the random number generator and the success rate can be seen. Therefore, a population size of 10 000 is chosen and the choice of the seed remains arbitrary.

Another parameter that was part of the testing is the size of the genome. The algorithm was

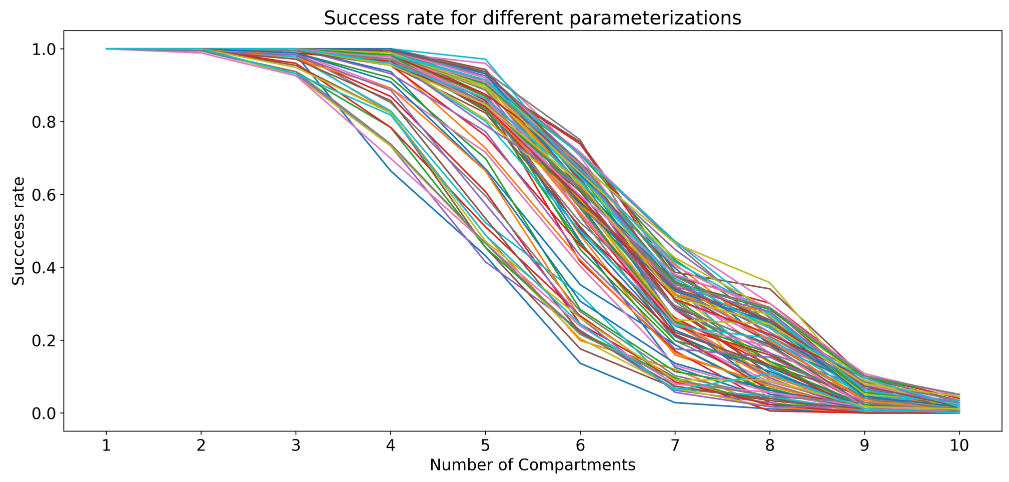
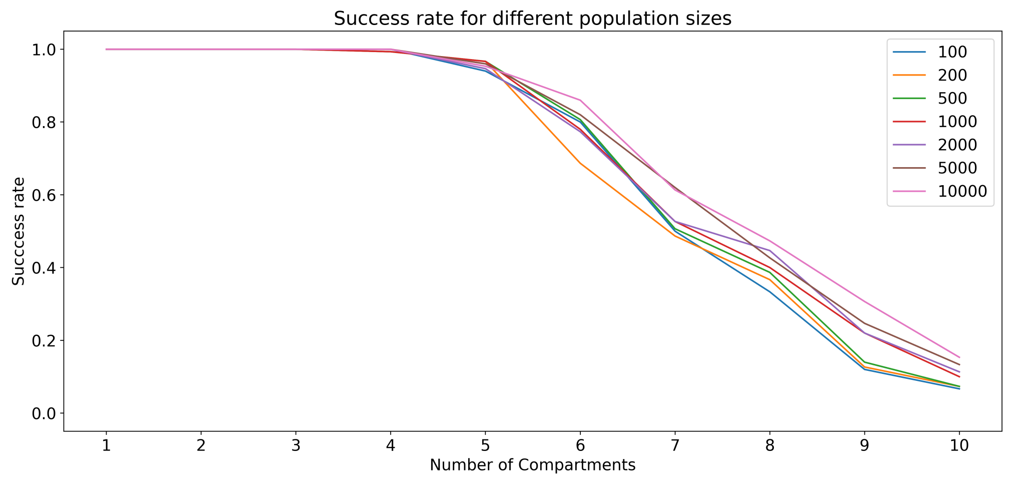


Figure 25: Comparison of different run parameter configurations. The test is performed with an average of 150 randomly generated neurons per number of compartments. The varied parameters are the probability for the mutational steps, the bit flip, the shift and the add/ remove mutation as well as the number of contestants in a tournament, the number of members in the hall of fame and the mating probability. It can be seen that some parameterizations have a overall better success rate and are therefore preferable to be used.

Population size	1000
Contestants in tournament	2
Member in hall-of-fame	50
Probability for random bit flips in individual	0.4
Probability for single bit to flip	0.01
Probability for mating	0.2
Probability for adding or removing a column	0.1
Probability for shifting the genome	0.1

Table 1: Optimal parameterization of the evolutionary algorithm.



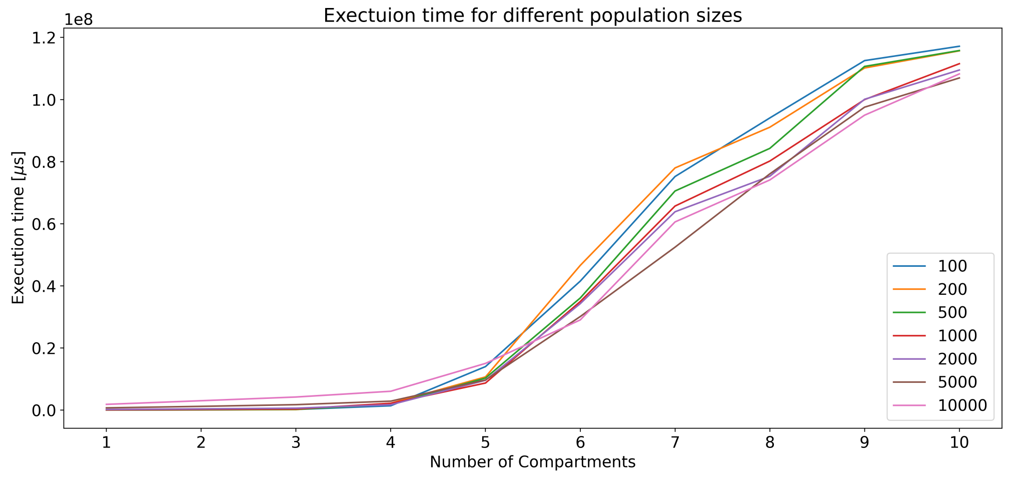


Figure 26: The plots show the dependency of the success rate and execution time of different population sizes. The test is performed with 150 different randomly generated neurons for each neuron size, with the optimal parameterization shown in Table 1. The results show, that a larger population size is beneficial for the success rate and the execution time of the algorithm.

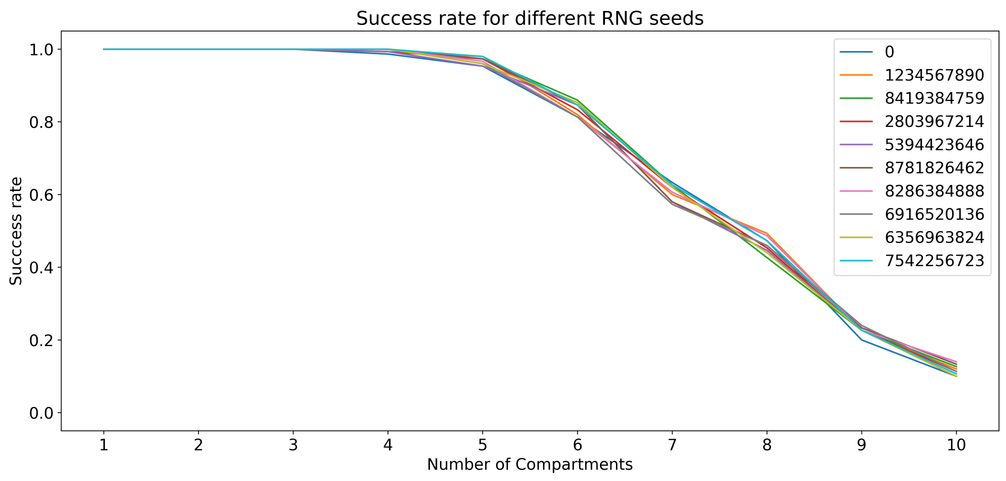


Figure 27: The plot shows the dependency of the success rate of different seeds for the random number generator. The test is performed with 150 different randomly generated neurons for each neuron size, with the optimal parameterization shown in Table 1. The results show no dependency between the success rate of the algorithm and the seed for the random number generator.

tested with different fixed lengths where for smaller neurons, with fewer compartments, a smaller genome has shown beneficial. Therefore, an optimization of the algorithm that varies the length of the genome or changes the limits in which mutations are applied during the execution is likely to improve the algorithms performance and success rate. This change allows to have a more local growth in the beginning but does not limit the number of neuron circuits used during the placement.

In the current implementation the first valid solution is taken as a result. In the future further steps can be introduced to optimise the valid result. This can happen in a constructive way by determining unused neuron circuits and flipping switches to free them. Another option is to optimise probabilistically by creating a population consisting out of individuals with the valid result as genome and the algorithm is run for a specific amount of time to find a solution with a higher fitness. Since a growth of fitness always means an improvement this method cannot invalidate the given results.

4 Discussion

This work aims at automating the placement of multicompartment neurons on BSS-2. Multicompartment neurons are useful since they replicate the behaviour of biological neurons better than point neurons do, but also because of their computational power [9]. For example, the structure of the dendrites can be used to pre-process a synaptic input before it reaches the soma or the separation into multiple compartments can be used to establish different learning rules on each compartment [3]. Since the mapping of an abstract multicompartment neuron model to the hardware configuration of the BSS-2-chip is non-trivial for larger neurons, three algorithms were implemented to automate the mapping.

During a preceding internship a neuron abstraction model was implemented that is used in this work. It is used to create neuron models that consist of multiple logical compartments that each fulfil a computational purpose. The neuron is represented in a graph structure and is the only input required by the placement algorithm to perform the placement.

During this work two different new algorithms were developed and one, which was developed during a preceding internship, was improved to solve the problem of placing multicompartment neurons onto the BSS-2-chip.

The simplest approach is the brute force algorithm, which tries every possible combination of switches, which control connections between neuron circuits on the chip. With $24^{256} \sim 2 \cdot 10^{353}$ possible configurations of the switches, the complexity of the algorithm is not useful for neurons with more than four compartments. Since the placement for neurons with few compartments is trivial, the algorithm has no use.

Another approach is the rule based algorithm which constructs the neuron onto the coordinate system starting with a center compartment and moving to the sides. For the general testing with randomly generated neurons this algorithm performed best, see Figure 28, but has logical limitations. For every structure this algorithm is unable to place, new rules for the placement need to be established.

Therefore, a third algorithm with an evolutionary approach was developed with the goal to create an algorithm capable of placing a large number of neurons while not being limited by logical problems but rather by performance or a time limit.

In direct comparison the rule-based algorithm performs better than the evolutionary algorithm, see Figure 28. The rule-based algorithm has execution times in the order of milliseconds whereas the evolutionary algorithm requires multiple seconds up to multiple minutes to find a valid solution. Another advantage of the rule-based algorithm is the higher success rate compared to the evolutionary approach. It finds solutions for some neurons with more than 30 compartments, if their topologies are simple enough.

The testing results of random neurons on the brute force algorithm, the evolutionary algorithm with the optimal parameterization as well as the rule based approach in its final state are shown in Figure 28. The success rate displayed in the plots does not take into account whether a neuron is placeable at all on the BSS-2-chip. Taking this into account would require a reliable algorithm that finds a valid solution for every placeable neuron, which was not achieved yet. The brute force algorithm was intended to be used for this purpose but found no use since its execution time is too high.

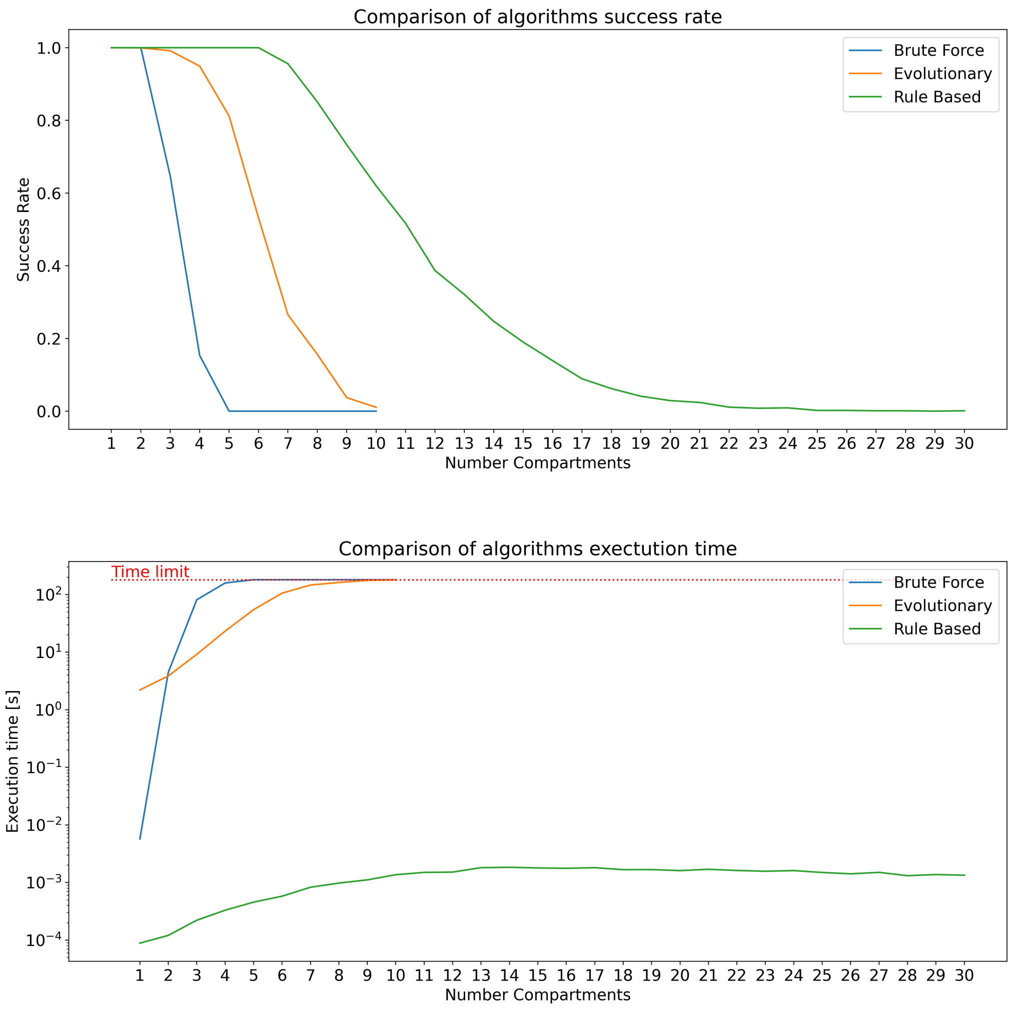


Figure 28: Comparison of the different algorithms. The evolutionary and the brute force algorithm were tested with 150 neurons per number of compartments since they have higher execution times. For both algorithms a time limit of 180 s is chosen to be able to test a sufficient amount of different neuron topologies. The ruleset algorithm was tested with 1000 neurons per number of compartments since it executes faster.

In direct comparison the bad performance of the brute force algorithm can be seen. The evolutionary algorithm can be ranked between the brute force and the rule based algorithm in terms of success rate. With the current parameterization it is capable of placing neurons with up to twelve compartments during the test. The rule based algorithm performs best in terms of its success rate as well as its execution time. It is capable of placing some neurons with up to 30 compartments during this test. The execution time is by far superior to the two other algorithms. Even for small numbers of compartments the evolutionary algorithm requires seconds for the placement and for larger neurons minutes are required to find a valid solution. The rule based algorithm finds solutions in milliseconds.

However, the evolutionary algorithm still has its advantage in the theoretical power to solve the placement problem for any placeable neuron topology. The limiting factor is the time given to find a solution. To make more neurons placeable during the given time a further optimization of the single algorithmic steps is required.

The problem of automating the placement of multicompartment neurons has been solved for a significant amount of neuron topologies by the implemented algorithms where the rule based and the evolutionary algorithm have proven useful for the placement whereas the brute force approach can not be used because of its bad scaling for larger neurons.

The brute force algorithm has a hard upper limit, since its execution time scales with the number of possible states of the hardware configuration. Therefore, the brute force algorithm is limited in use for small neurons.

The rule based algorithm is limited by the implemented rules. As mentioned above there are structures that cannot be placed at the current state of implementation.

The evolutionary algorithm faces two challenges. First an optimal parameterization needs to be found. Through the testing of different parameter combinations some parameter combinations are found superior to others, but since not every combination was tested, because of the large number of possible combinations, there might be a better configuration, that was not part of the testing. Secondly the single steps in the algorithm can be further optimized to improve the time a single generation requires. In combination with a longer run time during testing, a higher success rate can be achieved.

In addition to the development of the three algorithm described above an API comparable to the `PyNN`-multicompartment-interface was created. The abstract neuron models can be constructed in Python and can be used to create `PyNN`-populations of neurons.

Solutions to overcome some of the limitations of the currently implemented algorithms mentioned above will be presented in the following.

5 Outlook

The integration into `PyNN` will be continued. Currently, populations of the neurons can be created but no experiment can be run. Therefore, the placement where the algorithm is performed needs to be added before the run execution. To be able to use multicompartment neurons on the hardware a final translation of the model parameters to calibration parameters for the hardware needs to be performed.

The brute force algorithm will not receive any optimization in the future, since it was shown that the scaling of the algorithm defies every optimization.

The rule based algorithm is limited by the currently implemented placement rules. In future changes more rules for the placement can be established to make the placement of currently not placeable neuron topologies possible. Promising options are a change of shapes during the placement or a different treatment for different substructures. However, multiple iterative steps with testing for unplaceable structures in the neuron topology will be necessary to bring the algorithm to a state where it finds a valid solution for every neuron model placeable on the BSS-2-chip.

The evolutionary algorithm is limited by its parameterization and its runtime. To improve the parameterization further additional combinations of parameters can be tested to find superior solutions than the current. To improve the runtime of the algorithm the single steps that are performed during each iteration can be optimized to shorten the time a single generation needs. These single steps can be performed in parallel for multiple individuals of a population at once, however after each step and especially after each generation it is required to sync up the whole population. Therefore, the parallelisation is limited to single steps in each generation.

Another improvement is the implementation of a variable genome length. The algorithm starts applying mutations in a small area of the chip first and then increases the genomes' length to use more neuron circuits for the placement. This is useful since it preferably creates small solutions if possible but has no hard upper limit for the size of the placed structure other than the chip size. Mutations in a compact area of the grid are more likely to improve the result, since the placement grows locally and forms substructures that grow to increase the fitness. With this step there will be no need to lower the fitness if more resources are used than required by hardware resources, since the variable length of the genome limits the size of the configuration.

Independent of the algorithmic concept used to find a valid solution for the placement an additional step to optimize the result can be introduced. The goal of this step is to create a solution which uses fewer neuron circuits and has fewer unused neuron circuits in between used ones. This makes it possible to use more multicompartment neurons on one chip. This can be implemented in a constructive way, similar to the rule based algorithm, where rules are implemented that determine whether circuits are not required for a correct placement or if compartments can be repositioned to decrease the overall size of the neuron. Another implementation option is to let the evolutionary algorithm run to further maximize the fitness. The new population can be initialized with the valid solution which then can undergo the same steps as shown in section 3.4.3, to improve the result.

Equipped with the changes mentioned above either the evolutionary or the rule based algorithm are promising to solve the placement problem for a greater number of neurons and enable experiments with even more complex neuron topologies than currently possible.

References

[1] Wybo Willem AM. Data-driven reduction of dendritic morphologies with preserved dendrosomatic responses. *eLife*, 2021.

[2] David Baumeister. Placement algorithm for multicompartment neurons. Internship, Heidelberg University, 2024.

[3] Jacopo Bono, Katharina A. Wilmes, and Claudia Clopath. Modelling plasticity in dendrites: from single cells to networks. *Current Opinion in Neurobiology*, 46:136–141, October 2017.

[4] R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94:3637 – 3642, 2005.

[5] Chaboyer Brian. The age of the universe. *Physics Reports*, 307:23–30, 1998.

[6] Rosin Christopher D. New methods for competitive coevolution. *Evolutionary Computation*, 5, 1997.

[7] Dieter Jungnickel. *The Greedy Algorithm*, pages 129–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[8] Jakob Kaiser, Sebastian Billaudelle, Eric Müller, Christian Tetzlaff, Johannes Schemmel, and Sebastian Schmitt. Emulating dendritic computing paradigms on analog neuromorphic hardware. *Neuroscience*, 489:290–300, 2022.

[9] C. Koch and I. Segev. The role of single neurons in information processing. *Nat Neurosci*, 3 Suppl:1171–1177, November 2000.

[10] M. London and M. Häusser. Dendritic computation. *Annu. Rev. Neurosci.*, 28:503–532, 2005.

[11] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity. *Front. Neurosci.*, 16, 2022.

[12] Pynn documentation. `http://neuralensemble.org/docs/PyNN/`.

[13] Bartz-Beielstein Thomas. Evolutionary algorithms. *WIREs Data Mining and Knowledge Discovery*, 4:178—–195, 2014.

[14] Peter van Beek. Chapter 4 - backtracking search algorithms. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 85–134. Elsevier, 2006.

Acknowledgement (Danksagung)

Ich danke Philipp Spilger, Jakob Kaiser, Eric Müller, Simon Jonscher und Yannik Stradmann für das Korrekturlesen meiner Arbeit.

Ich danke Philipp für die umfangreiche und stets verfügbare Hilfe bei jeglichen Problemen in der C++ Entwicklung. Insbesondere für die erste fünfstündige Debugging-Session in der mehrere tausend Zeilen Compiler und Linker Fehler behoben wurden, aber auch für die teils zeitintensive Suche nach Logikfehlern.

Ich danke Jakob für die Ideen und Diskussionen zu Konzepten der evolutionären Algorithmen, sowie für den Versuch mich für die Schönheit von Vektorgrafiken zu begeistern.

Ich danke allen Mitspielern an der Dartscheibe und dem Kickertisch für eine erfrischende Mittagspause, insbesondere Eric, Joscha, Hartmut und Yannik am Kickertisch für erbarmungslose und teils kunstvolle Spiele.

The work carried out in this bachelor thesis used systems, which received funding from the EC Horizon 2020 Framework Programme under grant agreement 945539 (HBP SGA3), the EC Horizon Europe Framework Programme under grant agreement 101147319 (EBRAINS 2.0) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2182/1-390900948 (the Heidelberg STRUCTURES Excellence Cluster).

Appendix

repository	git hash	commits
grenade	3f7d61693290b686146a707ff13a0130ab488196	22996
		22890
		22528
code-format	09f3a985a6f264359b10a6a129dd6dce7e55c9e8	
logger	73dadb3ce413c521845ef7d36f818073eee4fefa	
halco	85e685aab6e8ea68ff78ae4273e17a75420813f9	
haldls	2dd2b267f1e283160972aa0190a4ea0cfc0f0f56	
hate	35b3cb211cabbbc5c01036ae7878a73e338166c4	
libnux	fc3b137384596ea5adbd5d4ee1ddfc9761a2aabc	
fisch	6120fc0ac0d90b3c66a212b3cc5cc25034bf584e	
hxcomm	95abf25670bd8cb7cc5b499cde56f653130cf20c	
rant	722edd57c9e42462a660db8a1febb0211ffad07c	
ztl	b6745261d8bfdce44516d58d632c3c73834839d2	
pywrap	5e2af30e9593882b471d3cd02df00b93f13ff479	
lib-boost-patches	136c5b41cb046afe2c726aa4646928bf5190622e	
sctrltp	1d854f953f7e8c8ead44406a22bb80421ca3857c	
hwdb	9607ff1f3090ec18e75b76ed90592be643b4cab2	
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3	
flange	28e729d59df3b4ff380f84351c40d4da3086bed8	
lib-rcf	21fbcb0a7c30efed98278ee997754f28092b9736	
bss-hw-params	b7be7827b51536804f0bda76f8ba4be693df23a8	
pynn-brainscales	d51bbdabc14fc5da5cc63a0a1110781acef9c4e0	22827
calix	a72f41e4a594ce82ab81d0d43f0ce2f3f67108ff	

Table 2: Current software state of modified repositories.

key	value
path	/containers/stable/2024-04-17_ 1.img
link	https://openproject.bioai.eu/containers

Table 3: Current state of the singularity container.

Declaration

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

[illegible]

Heidelberg, den 13.08.2024,