Efficient solution validation of constraint satisfaction problems on neuromorphic hardware: the case of Sudoku puzzles

Riccardo Pignari, Vittorio Fra, Enrico Macii, *Fellow, IEEE*, Gianvito Urgese, *Senior Member, IEEE*

***Abstract*—Spiking neural networks (SNNs) offer an effective approach to solving constraint satisfaction problems (CSPs) by leveraging their temporal, event-driven dynamics. Moreover, neuromorphic hardware platforms provide the potential for achieving significant energy efficiency in implementing such models. Building upon these foundations, we present an enhanced, fully spiking pipeline for solving CSPs on the SpiNNaker neuromorphic hardware platform. Focusing on the use case of Sudoku puzzles, we demonstrate that the adoption of a constraint stabilization strategy, coupled with a neuron idling mechanism and a built-in validation process, enables this application to be realized through a series of additional layers of neurons capable of performing control logic operations, verifying solutions, and memorizing the network's state. Simulations conducted in the GPU-enhanced Neuronal Networks (GeNN) environment validate the contributions of each pipeline component before deployment on SpiNNaker. This approach offers three key advantages: i) Improved success rates for solving CSPs, particularly for challenging instances from the hard class, surpassing state-of-the-art SNN-based solvers. ii) Reduced data transmission overhead by transmitting only the final activity state from SpiNNaker instead of all generated spikes. iii) Substantially decreased spike extraction time. Compared to previous work focused on the same use case, our approach achieves a significant reduction in the number of extracted spikes (54.63% to 99.98%) and extraction time (88.56% to 96.41%).**

***Impact Statement*—This work presents a novel approach to address Constraint Satisfaction Problems through Spiking Neural Networks (SNNs) utilising neuromorphic tools like the GeNN framework and the SpiNNaker platform. We propose a new fully spiking pipeline that incorporates a constraint stabilization strategy, a neuron idling mechanism, and a built-in validation procedure. Our pipeline targets efficiency and performance of SNN-based solvers for Sudoku puzzles, leading to improvements in success rates and data transmission compared to previous solutions. Specifically, the reduction of extracted spikes, ranging from 54.63% to 99.98%, provides extraction time reduced by values between 88.56% and 96.41%. This results in significant enhancements in terms of energy efficiency and computational performance. Therefore, we show further evidence of the potential advantages of brain-inspired approaches that rely on neuromorphic HW for implementing effective and low-power solutions, which are suitable for real-world problems that characterise constrained key technological domains such as AI, IoT and Industry 4.0.**

***Index Terms*—Constraint satisfaction problem, GeNN, neuromorphic computing, spiking neural network, SpiNNaker, sudoku.**

I. Introduction

The authors are with the Electronic Design Automation (EDA) Group at Politecnico di Torino, Turin, Italy (e-mail: riccardo.pignari@polito.it, gianvito.urgese@polito.it).

CONSTRAINT satisfaction problems (CSPs) are mathematical problems involving a collection of elements whose state must satisfy a set of constraints. The definition of a CSP is formulated through a triplet describing the collection of variables, their respective domains, and the set of constraints. The applications of CSPs cover a wide range of scenarios such as logistics optimization [1]–[3], development of new drugs [4], [5], warehouse management [6]–[9], scheduling [10], structural optimization [11]–[13] and resource allocation [14]–[16]. One possible approach to solve CSPs involves the use of spiking neural networks (SNNs), a specific type of artificial neural networks (ANNs) that uses action potentials, or spikes, for neural communication [17]. SNNs are effective in solving CSPs like the traveling salesman problem (TSP) and the 3-SAT problem [18] as well as graph coloring, Latin squares, and Ising spin glasses [19]. The basic idea behind relying on SNNs as CSP solvers is to design a network of spiking neurons that evolves in time to search for viable solutions. Such a network is designed with a core set of interconnected neurons, or neuron populations, supplemented by a group of auxiliary neural units, either single neurons or populations, representing the specific problem domain.

A notable advantage of employing SNNs for solving CSPs lies in the possibility of deploying these models on neuromorphic hardware platforms. These specialized hardware architectures are designed to mimic the structural and functional characteristics of the human brain, enabling efficient computation through biologically inspired principles. While the deployment of an SNN-based CSP solver on neuromorphic platforms has been demonstrated in a previous work [19], the overall workflow was not entirely independent of external hardware resources. Specifically, the validation of solutions was typically performed after extracting the spiking activity from the neuromorphic hardware, requiring additional computational effort on external hardware components.

In this context, our work aims to address this limitation by proposing an enhanced, fully spiking pipeline, depicted in Figure 1, that directly incorporates solution identification and validation on the neuromorphic hardware platform, minimizing the reliance on external hardware. To achieve this, we designed a series of layers implementing logic operations to support the main network. These layers enable network activity filtering operations (*Polisher* block), validate solutions and interrupt neuronal activity (the *NetChecker* and the *If* blocks), and memorize the state representing the solution (the *Memory* block).

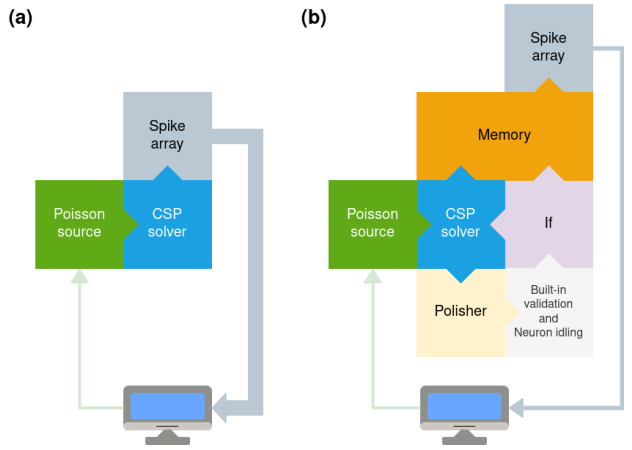


Fig. 1. Building blocks of the fundamental (a) and enhanced (b) pipeline. In (a), the solution proposed by [19] is depicted, with the CSP solver in charge of identifying the solution of the given problem. The *Poisson source* and *Spike array* blocks can be thought of as the spiking input and output respectively. The output collected from this latter needs to be transmitted out of the neuromorphic platform to be validated. The enhanced pipeline (b) introduces additional blocks to implement both a stop condition and the solution validation on neuromorphic hardware. By embedding such operations, and hence by taking advantage of entirely spike-based computation for both solving the problem and validating the solution, it largely reduces the amount of data to be transmitted back from the neuromorphic platform. Consequently, the time required for such operation is significantly lowered, and the computational cost of validating the found solution on non-neuromorphic hardware is completely removed.

To demonstrate the advantages of our SNN-based blocks, we focused on the Sudoku puzzle variant of the Latin square problem. The relevance of the latter is given by its variety of practical applications in different fields, including cryptography (Hill cipher algorithm) [20], scheduling and timetabling [21], error-correcting codes [22], [23], and agronomic research [24].

II. Background

Optimization problems target the identification of the best solution among a number of possible candidates, given a specific task. The complexity of such problems is proportionally related to the degrees of freedom of the considered system, which affect the solutions space to be investigated: the more complex the problem, the larger the number of possible solutions and the less feasible the brute force approach to find the optimal one.

Over the years, numerous methods have been developed to address optimization. Among them, analytical approaches, based on the Lagrange multipliers or on linear programming, and the most modern strategies relying on statistical approaches and machine learning (ML) methods. Additionally, a frontier application is represented by SNNs, which can explore the possible solutions by exploiting an attractor dynamics with a considerable reduction in computational and energy cost.

CSPs are a specific type of optimization problems which belong to the NP-complete set. Their mathematical formulation is based on the triplet $\langle X, D, C \rangle$ where:

- $X = \{X_1, X_2, ..., X_n\}$ describes the set of variables present in the problem;
- $D = \{D_1, D_2, ..., D_n\}$ describes the set of domains the variables belong to;
- $C = \{C_1, C_2, ..., C_n\}$ defines the set of constraints.

Among the above mentioned examples of CSP, the Latin Square problem consists of a matrix with $n \times n$ cells, partially filled with n different elements, such that once solved each symbol appears in each row and column only once. In the Sudoku variant, the structure is constituted by a square of 9×9 cells, subdivided into 9 blocks of 3×3 sub-squares. Each cell of the Sudoku puzzles must be filled in with numbers from 1 to 9. Solutions to Sudoku puzzles and other CSPs can be investigated by means of SNNs [18], [19], [25]–[28] by replacing the standard nonlinearity adopted in ANNs through perceptron-based units with bioinspired elements such as leaky integrate-and-fire (LIF) neurons. A generalized formulation for CSP mapping onto SNNs was shown by Jonke et al. in [18], where a methodology to model the energy landscape of a given problem through the topological structure of a neuromorphic model was proposed. The resulting dynamics of the network corresponds to a dynamical system with a temporal evolution described through an attractor model [29], with a continuous exploration of possible states related to the original problem, whose fixed points correspond to the possible solutions.

In [19], SNNs designed to translate the mathematical formulation of the problem into the number of neurons and their synaptic connections are used for a stochastic search of the solution. Precisely, three CSP classes are taken into account, namely Graph Coloring, Latin Square Problem and Ising Model, showing the feasibility of such approach. Nonetheless, some limitations can be identified which partially limit the prospective impact of the proposed methodology. Specifically, three main aspects can be outlined. First, the fixed simulation time hinders the possibility of stopping the process once a solution is found, translating into an unnecessary energy consumption. Similarly, performing the solution validation on an external platform with respect to the one onto which the SNN runs implies data preparation and transfer which induce further time and energy consumption. Third, reliability issues affecting the problem mapping can arise if specific design choices in the clues definition are not taken.

Further studies [30]–[32] have also investigated neuromorphic approaches to the solution of the Latin square problem, exploring different methodologies and platforms and validating the efficacy of novel neuron models in the domain of CSPs.

III. Implementation

A Sudoku puzzle consists of 81 cells arranged in a squared 9×9 matrix made of 9 blocks of 3×3 sub-squares. Each cell represents the elemental unit of the puzzle, namely the basic building block of the system. The relationships between different cells along rows and columns and within the sub-squares, together with the initial clues, define the specific characteristic of each CSP of this type.

A. Network for puzzle solution

1) Neuron populations as elemental units: Every single cell of the Sudoku puzzle is modelled by employing 9 populations

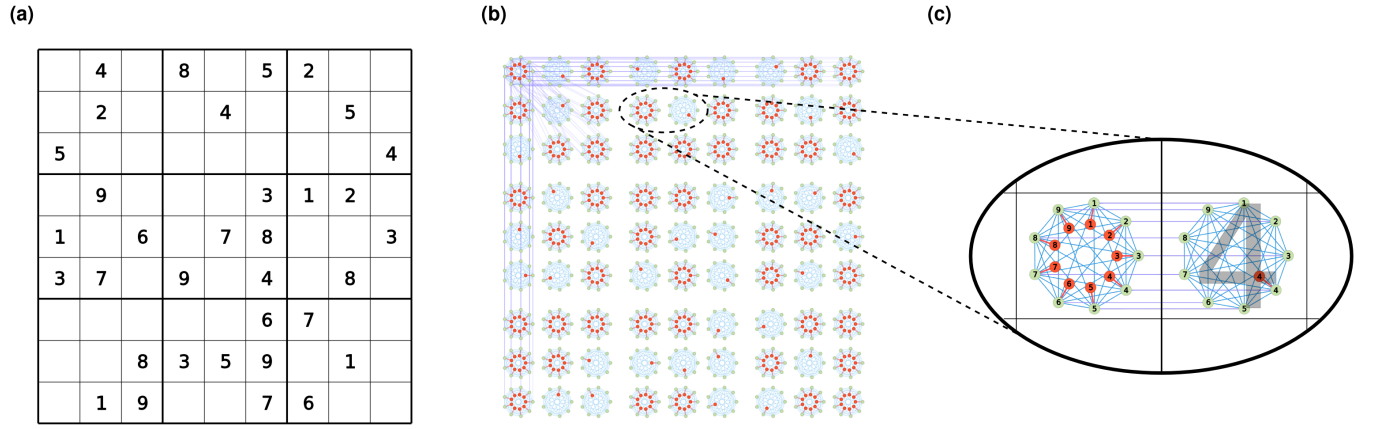


Fig. 2. Example of a Sudoku puzzle (a) with its corresponding SNN-based CSP solver (b). Connections for internal inhibition are depicted in *blue*, while *purple* lines represent lateral inhibition. All the stimulus populations are shown in *red* together with the corresponding connections. For the initial clues, single stimulus populations are shown according to the specific cell values. To improve clearness, lateral inhibition connections are shown for the top-left cell only. In (c), the 9 possible values for each cell of a Sudoku puzzle are modelled by means of 9 populations (depicted in *green*) of neurons. The winner-take-all configuration (achieved through the *blue* inter-population connections) ensures that only one population is active and only one value is taken by a cell. In *red*, the stimulus populations and the corresponding connections are shown. On the right-hand side, an example of clue modelling to fix a single cell value, 4 in this case, is reported.

of LIF neurons, one for each possible digit, connected one to another through a mechanism of internal inhibition resulting in the so-called winner-take-all configuration [29]: a single population can remain active while suppressing all the other ones, to ensure that each cell takes a single value at a time. Across cells, populations representing the same value are connected to implement lateral inhibition, ensuring that a single value can not appear more than once along a row, a column, or in a sub-square. Additionally, all the populations have a further synaptic connection to so-called stimulus populations, whose purpose is to keep available the number that the cell can assume. A schematic of the inter-population connections is reported in Figure 2. The specific neuron model implemented is a current-based (CuBa) LIF [19], [33], with membrane potential and synaptic current described in time by Equation 1 and Equation 2 respectively:

$$\frac{dV_m(t)}{dt} = \frac{I_{syn}(t) + I_{offset}}{C_m} - \frac{V_m(t) - V_{rest}}{\tau_m} \tag{1}$$

$$\tau_d \frac{dI_{syn}(t)}{dt} = -I_{syn}(t) + I_{spike}(t - t_s) \tag{2}$$

where $V_m(t)$ is the membrane potential; V_{rest} is the leaky component; C_m models the membrane capacitance; τ_m is the temporal decay constant for the membrane potential; I_{offset} is an additional bias current to control the internal dynamics; $I_{spike} = w\delta(t - t_s)$ is the contribution through the synaptic weight w of the incoming spiking activity occurring at time t_s; τ_d is the temporal decay constant for the synaptic components. Each time the membrane potential reaches the threshold voltage V_{th}, a spike is emitted and the membrane voltage is reset to a value V_{reset} for a time τ_{reset}. In Table I, the constant neuronal parameters of the above equations are reported with their values.

The SNN-based system describing the Sudoku puzzle is made stochastically evolve, in terms of spiking activity, to explore the configuration space with neural dynamics kept slightly above the firing threshold condition. Such a system is also referred to as the CSP solver, being the real responsible for the solution investigation. Its initial condition is defined through a random initialization of the synaptic weights, uniformly distributed within a given range of values, for the connections to the stimuli and for both internal and lateral inhibition, as reported in Table II.

Starting from such random initialization, the network evolves, at each simulation step, determining the most active population for each cell of the Sudoku puzzle as a result of the winner-take-all configuration. In order to allow the enabling of the possible values that a cell can take on, stimulus populations are used. In Figure 2a, a generic cell is shown with its connectivity structure involving all the stimulus populations. This represents the condition of the initially empty cells of a Sudoku puzzle, whose specific value is not yet determined. To ensure the initial clues are preserved, namely to model cells with predefined values that cannot be changed during the system evolution, stimulus populations are instead employed as depicted in Figure 2c: given the value to be kept, 4 in the example, only the corresponding population is connected to the stimulus.

2) Attractor dynamics: Using the above-mentioned architecture, a Sudoku puzzle is mapped onto a sparsely connected graph, as the one depicted in Figure 2b, by encoding the properties and the constraints of the problem through the population connections. As a result of such connections, it is not possible to deduce the global behavior of the entire network from the functioning of the individual components. The SNN evolution can be instead described through an attractor dynamics, where the attractive fixed point corresponds to the solution of the puzzle. However, although there is only one possible solution to the puzzle, the mapping process may result in other minor attractive fixed points which do not represent a solution but can be interpreted as local minima trapping the network evolution.

This dynamics strongly depends on the neuron population size: too small results in a noisy network without solving capability; excessively large makes populations far too active, with the consequence of corrupting the correct interplay between different cells.

B. Constraint stabilization

A possible effect induced by large neuron populations is the change of the initial clues during the evolution of the system. The very origin of such phenomenon is however to be ascribed to the spiking activity rather than to the population dimension. We indeed observed that changes of the initial clues can be reproduced by acting on neuron parameters like the threshold voltage and the bias current, or on the synaptic connections, which induce a higher spiking activity.

Interestingly, if the initial clues are changed, the network can still find the correct solution to the new problem. In a sense, the network demonstrates its ability as a problem generator. However, such reformulation and adaptation capability must actually be seen as an issue: the solution found by the network is not related to the problem we want to solve but to a different one.

Among the potential causes for the alteration of the initial clues, including threshold voltage, bias current, and synaptic connections, it was determined that addressing the latter would be the most effective solution to resolve this issue.

As outlined in [19], lateral inhibition connections, depicted in *purple* in Figure 2, are established to relay the presence and value of the initial clues throughout the network via bidirectional connections between Sudoku cells subject to the same constraints. To mitigate the alteration of states, lateral inhibition connections from the initial clue populations are made unidirectional towards empty cells, thus preventing lateral inhibition from other populations and avoiding the modification of the initial states corresponding to the original constraints.

C. Neuron idling and built-in validation

To address the limitations found in the CSP solver of [19], as well as in the solution validation process, we introduced an additional element playing the role of an auxiliary network. Such a network is composed of 4 blocks, depicted in Figure 3 together with the CSP solver. It is responsible of multiple actions aimed at avoiding unnecessary activity and validating the solution found by the main network, namely the CSP solver. First, the *Polisher* block is used to collect and filter out the activity of the main network, so that only the spikes from the most active population are kept; then, the *NetChecker* block verifies if the original constraints have been preserved during the system evolution; third, the *If* block, upon validating the found solution, interrupts the activity of the CSP solver and it activates the final block, called *Memory*, in charge of keeping the activity state corresponding to the validated solution.

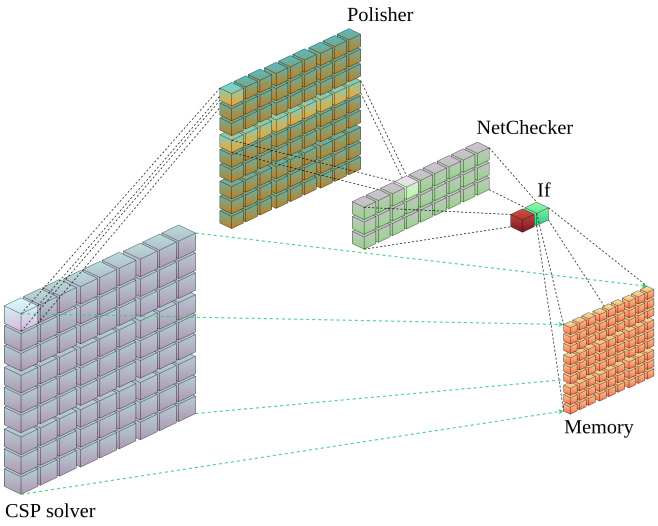


Fig. 3. The four blocks (*Polisher*, *NetChecker*, *If* and *Memory*) are introduced to implement the neuron idling mechanism and the built-in validation in the enhanced, fully spiking pipeline. The dashed lines are used to emphasize the inter-block interatcion scheme by highliting the connections of individual subunits. The connection between CSP solver and *Memory* is depicted in *green* as one of the two components of the *If* block to specify that such connection is active in specific conditions only.

TABLE I
SUMMARY OF THE NEURONAL PARAMETERS USED TO IMPLEMENT THE CUBA-LIF MODEL

Neuronal parameter	Value
C_m	0.25 nF
I_{offset}	0.1 (*) nA
V_{rest}	-65.0 mV
V_{th}	-50.0 mV
V_{reset}	-70.0 mV
τ_m	25.0 ms
τ_{refrac}	2.0 ms
τ_d	5.0 ms

(*) For the CSP solver, 0.3 is used to ensure neural activity in the absence of input stimuli.

The *Polisher* block, introduced with the objective of eliminating the background noise produced by the less active populations, consists of a layer whose structure is inspired by that of the CSP solver, with two key differences. First, a reduced number of neurons per population is used, as detailed in Table II. Second, each population receives stimuli from its counterpart in the CSP solver.

The *NetChecker* is composed by 3×9 units containing 10 populations each. Analyzing the structure in more detail, the units are defined as follows: 3 as the number of constraint types, namely on rows, on columns and for sub-squares; 9 for the number of rows, columns and sub-squares. The 10 populations that each cell is made up of are structured with the objective of verifying that all the constraints are respected. This is achieved through the use of 9 control populations for replicated digits and an additional population, designated as the 'check population', which is activated if all the constraints are satisfied. The 9 control populations are connected in a one-to-one scheme with the corresponding populations in the *Polisher* block: each control population models a digit and

each unit is connected to all the Sudoku cells along a row, along a column or in a sub-square. For instance, the unit in position (1,4) of the *NetChecker* in Figure 3 contains the 9 control populations connected to the populations of the fourth row in the *Polisher* according to the digit they represent, and the check population connected to all the populations of the same row.

Once all constraints are satisfied, the *If* block verifies the solution validity with respect to the initial problem. Such block consists of a single layer with two populations of neurons which are activated depending on the `True` or `False` outcome of the constraints conservation check. `False` is active if at least one of the 9 constraint-related populations of the *NetChecker* is active; `True` is instead obtained if all the 'check populations' of the *NetChecker* are active. To ensure that the two populations for `True` and `False` are not active at the same time, a winner-take-all scheme is adopted.

If the solution found is validated, the *If* block stops the evolution of the system by activating the *True* population that inhibits all the cells in the CSP solver, with the subsequent deactivation of the *False* population which enables the *Memory* block to save the last state of the CSP solver. Both of these mechanisms are implemented through inhibitory synapses.

The structure of the *Memory* block is analogous to that of the CSP solver, with the exception of the change in the number of neurons in each population, which is reduced to 3. As in the *Polisher*, each population receives stimuli from its counterpart in the CSP solver.

The structure of the four blocks, which facilitate the process of neuron idling and built-in validation, is the consequence of a topological mapping onto a spiking network of the type of constraints that a generic Sudoku puzzle must adhere to. This conversion, specifically tailored for problems belonging to the Latin square class, is nonetheless independent of the specific puzzles being analysed, and it can be readily scaled up for the identification of new forms of constraints by acting on the *NetChecker*.

D. Step-by-step implementation

This section provides an overall view of the proposed fully spiking pipeline, with the aim of giving a step-by-step description of the end-to-end implementation.

The core element, namely the CSP solver, is built to replicate the 9×9 grid of the Sudoku puzzle. Each of the 81 cells is made of 9 populations of LIF neurons, connected one to another through inhibitory synapses designated as internal inhibition. Such connectivity is adopted to employ a winner-take-all scheme, which eventually ensures a single active population per cell as required by the rules of Sudoku puzzles. In order to determine which digit corresponds to a cell, each of them is connected to a single stimulus population representing values from 1 to 9. Cells with stimulus populations for the same digit are also inter-connected for lateral inhibition, to ensure that a given digit will not appear more than once along a row, a column, or in a sub-square. Figure 2c depicts an example of mapping by focusing on an empty cell in position (2,4) and an initial clue in position (2,5). The former has all the populations connected to a stimulus, while the latter imposes 4 as cell value by enabling only the connection of that specific stimulus population.

Connected to this main part is the *Polisher* block. It is structured in the same way as the CSP solver except for the absence of the stimulus populations and the reduced number of neurons per population (Table II). Stimuli arrive to the cells of this block through individual direct connections with the CSP solver cells, and the aim is to suppress possible residual background noise produced by the non-dominant populations within the CSP solver.

The *Polisher* block is then connected to the *NetChecker*, which is responsible for verifying that the spiking activity collected from the CSP solver adheres to the constraints of the Sudoku puzzle. The structure of this block is made of 3×9 units of 10 populations of neurons. Its role is to verify the three types of constraints: the first 9 units are responsible for the constraints on the rows, the next 9 ones for verifying the constraints on the columns, and the final 9 ones for checking the constraints on the sub-squares. The 10 populations are instead implemented as 9 control populations and one additional population, designated as the 'check population', which is activated if all the constraints are satisfied. Connections between the *NetChecker* and the *Polisher* block have the following scheme: the control populations refer to single cell values and every unit is connected to an entire row, column or sub-square.

Subsequent to the *NetChecker* is the *If* block, which consists of two populations inter-connected with a winner-take-all strategy: the `True` and the `False` population. The former is connected to the 'check populations' of the *NetChecker*, while

TABLE II
SUMMARY OF THE PARAMETERS USED FOR THE DIFFERENT COMPONENTS OF THE WHOLE MODEL RUN IN GENN. WHERE RANGES INSTEAD OF VALUES ARE REPORTED, UNIFORM DISTRIBUTION OF VALUES WITHIN SUCH RANGES ARE CONSIDERED.

	Neurons per population	**Synaptic connection weight**				
		Stimulus	**Internal**	**Lateral**	**to CSP solver**	**to Memory**
CSP solver	27 (*)	[1.4, 1.6]	[-0.08, 0.00]	[-0.08, 0.00]	/	/
Polisher	10	1.0	-1.0	/	/	/
NetChecker	10	1.00 0.15 (check pop.)	/	-1.2	/	/
If	10	11.00 0.02 (val. pop.)	0.5 (`True`) 0.0 (`False`)	-1.0	-2.0	-0.6
Memory	3	1.0	0.4	-0.3	/	/

(*) The solution for two of the three puzzles belonging to the easy class (#2 and #3 specifically) has been simulated by using 28 neurons per populations in the CSP solver due to an observed gain in performance.

TABLE III
SUMMARY OF THE SOLUTIONS FOUND PERFORMING 300 SIMULATIONS IN GENN FOR THREE DIFFERENT PUZZLES OF EACH CLASS

Class		**Easy**				**Medium**				**Hard**			
Constraint stabilization		✗		✓		✗		✓		✗		✓	
Neuron idling and built-in validation		✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Puzzle:	#1	293	289	292	297	5	5	210	202	0	0	121	147
	#2	188	203	216	233	0	1	126	136	0	0	0	0
	#3	248	245	273	282	0	0	7	7	0	0	91	89

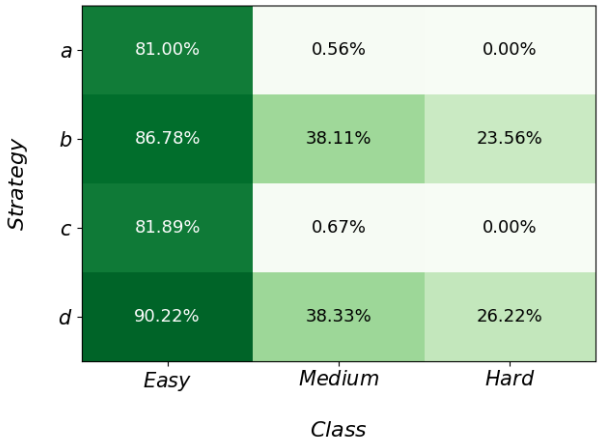


Fig. 4. Comparison of the different strategies investigated through simulations in GeNN. The values correspond to the success rate, evaluated as the percentage of solutions found for three different puzzles of each class simulated 300 times.

the latter is connected to all the control populations within the *NetChecker*. Additionally, the `False` population is connected to all the cells of the *Memory* block.

As final element, the *Memory* block has the same structure and connections with the CSP solver as the *Polisher* block, with the only exception of just 3 neurons per population. It is also connected to the `False` population of the *If* block.

IV. EVALUATION

We tested our fully spiking approach on nine Sudoku puzzles belonging to three different difficulty classes: easy, medium, and hard. Specifically, we used the ones from [19] belonging to the easy and hard classes and added a selection of puzzles from [34].

At the software level, we leveraged the capabilities of the GeNN environment [35] on a hardware platform equipped with an Intel 11th Gen i7-11700KF (16) @ 5 GHz processor, 32GB of RAM, NVIDIA RTX A4000 graphics card with 16GB of VRAM and Ubuntu 20.04.5 LTS x86_64 operating system, while on the neuromorphic hardware side we relied on SpiNNaker [36]. Compared to [19] we were forced to use the `sPyNNaker8` library for PyNN instead of `sPyNNaker7` due to deprecation of the latter.

In order to produce a comprehensive analysis, and a valuable comparison, we solved the puzzles with four different strategies, all of them based on the same CSP solver:

a) as in [19] to obtain the baselines for our fully spiking approach;
b) by including constraint stabilization only;
c) by including neuron idling and built-in validation only;
d) by merging *b* and *c*.

By performing 300 simulations to solve each puzzle with all configurations, we collected results from a total of 10800 experiments.

The adoption of a constraint stabilization mechanism resulted in the complete fix of the issue affecting the original implementation. It was indeed verified that the initial clues were never changed when including such additional element for the puzzle solution, in comparison with the 171 constraint changes identified using the solver of [19]. From Table III and Figure 4, the impact of constraint stabilization, which corresponds to the above mentioned strategy *b*, can be appreciated in detail. Especially for the medium and hard classes, the number of successful solutions increases by more than one order of magnitude, and the success rate does accordingly:

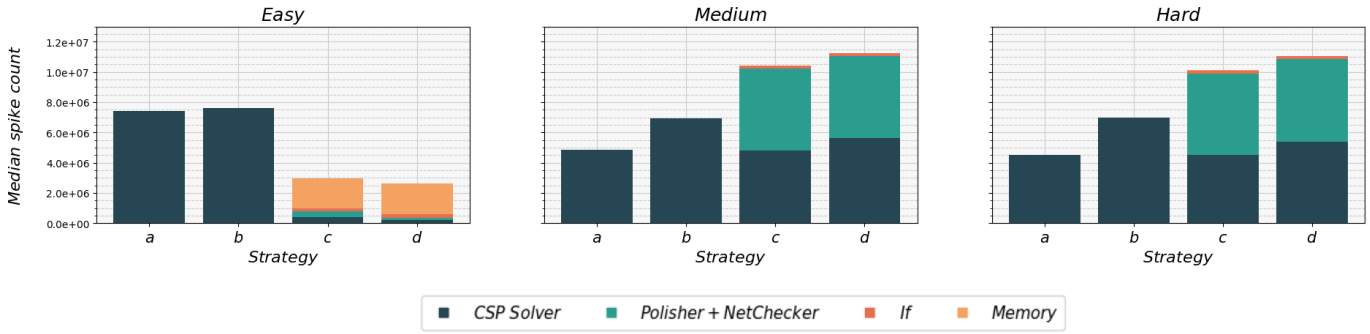


Fig. 5. By introducing the neuron idling mechanism and the built-in validation, the median number of spike produced by the whole pipeline is significantly reduced for puzzles of the easy class. For the medium and the hard class, the poor solving capability of the CSP solver, which leads to the exploitation of the whole simulation time in most of the cases, hides such effect.

TABLE IV
SUMMARY OF THE SOLUTIONS FOUND PERFORMING 100 EXPERIMENTS ON SPINNAKER FOR THREE DIFFERENT PUZZLES OF EACH CLASS

Class		**Easy**		**Medium**		**Hard**	
Constraint stabilization		✗	✓	✗	✓	✗	✓
Neuron idling and built-in validation		✗	✓	✗	✓	✗	✓
Puzzle:	#1	97	85	0	12	0	14
	#2	69	64	0	0	0	0
	#3	81	87	0	0	0	5

from 0.56% to 38.11% for the medium class and from 0.00% to 23.56% for the hard one. Concerning the puzzles of the easy class, the success rate increases instead from 81.00% to 86.78%.

By adopting the strategy *c*, we instead showed the suitability of a neuron idling mechanism in a fully spiking pipeline that comprises both the puzzle solution and its validation. Specifically, such mechanism significantly reduces the energy consumption if a solution is found early in the simulation time. For the easy puzzles, it resulted in a decrease of the median number of spikes during the simulations of about 59.87% with respect to strategy *a*. As it is reported in Figure 5, this improvement was not observed for the other classes, the reason being the much smaller success count which implies a much longer time spent by the CSP solver in its dynamical evolution. The idling mechanism is indeed not triggered until a solution to the puzzle is found, which consequently means that it does not act on the spiking activity of the system if a valid solution is not identified. The relevant increase of spikes produced with the strategy *c* for the medium and hard classes, shown in Figure 5, hence arises from the reduced capability of the CSP solver of finding valid solutions, which in turn translates into longer simulations with increased computational efforts. In terms of success count, and success rate accordingly, as it can be appreciated from Table III and Figure 4, the strategy *c* provides a slight improvement: 0.89% and 0.11% for easy and medium puzzles respectively. This increase in the number of valid solutions found, given that the CSP solver is not changed, can be directly ascribed to the built-in validation, which replaces the bin-based approach of strategy *a*.

Finally, the coupling of strategies *b* and *c* gives rise to the complete pipeline we label as strategy *d*, where constraint stabilization and neuron idling with built-in validation are all used. As it is reported in Figure 4, such synergy eventually results in relevant improvements for all the classes compared to the original strategy *a*: from 81.00% to 90.22% for the easy puzzles, from 0.56% to 38.33% for the medium class and from 0.00% to 26.22% for the hard one.

Following the simulations in GeNN, we assessed the actual performance of our fully spiking pipeline on neuromorphic hardware by running it on SpiNNaker. Specifically, we compared strategy *a* and *d* with 100 simulations for each puzzle of each class in both cases, thus collecting results from a total of 1800 experiments. As it is well known, the fundamental and distinguishing characteristic of neuromorphic platforms like SpiNNaker compared to GPUs is the spike-based computation. As a consequence, while the results of the simulations performed in GeNN could be deeply investigated by directly probing the different populations activity, any analysis of the activity produced when running the model on SpiNNaker could be performed exclusively after dedicated spike collection and transmission. Therefore, the most reliable approach to inspect the results from SpiNNaker without introducing any overhead was to collect and transmit spikes only at the end of the simulations.

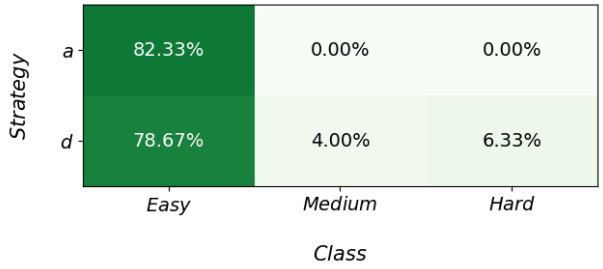


Fig. 6. Comparison between the original pipeline from [19] (strategy *a*) and the enhanced, fully spiking pipeline (strategy *d*). The values correspond to the success rate from 100 experiments run on SpiNNaker for three different puzzles of each class.

Table IV and Figure 6 show the success count and the corresponding success rate respectively. Coherently with the results achieved in GeNN, startegy *d* turns out to introduce improvements in the success rate for the medium and hard classes, although they are smaller than on GPU: from 0.00% to 4.00% and from 0.00% to 6.33% respectively. On the contrary, a reduction of solving capability from 82.33% to 78.67% was reported for the easy class. Such differences with respect to the behaviour observed on GPU, despite being undesired, are not completely unexpected, and they can be ascribed to the reduced numeric precision of the fixed-point representation introduced by the SpiNNaker platform during the synaptic weights quantization process.

Such phenomenon was reported by Ostrau et al. [31] also, whose work investigated three neuromorphic hardware platforms, namely SpiNNaker, Spikey and BrainScaleS, and the NEST software framework. A discrepancy between the software framework and the hardware platforms in terms of their capacity to solve Sudoku problems was shown, as a consequence of the different numeric precision employed. To verify whether this hypothesis could hold for our fully spiking pipeline as well, we performed additional simulations in GeNN to adopt two different numeric precisions other than the default one of 32-bit floating point: `float16` and `float64`. For all the classes, each puzzle solution was simulated with strategy *d* performing 300 runs, which led to a total of 5400 new experiments.

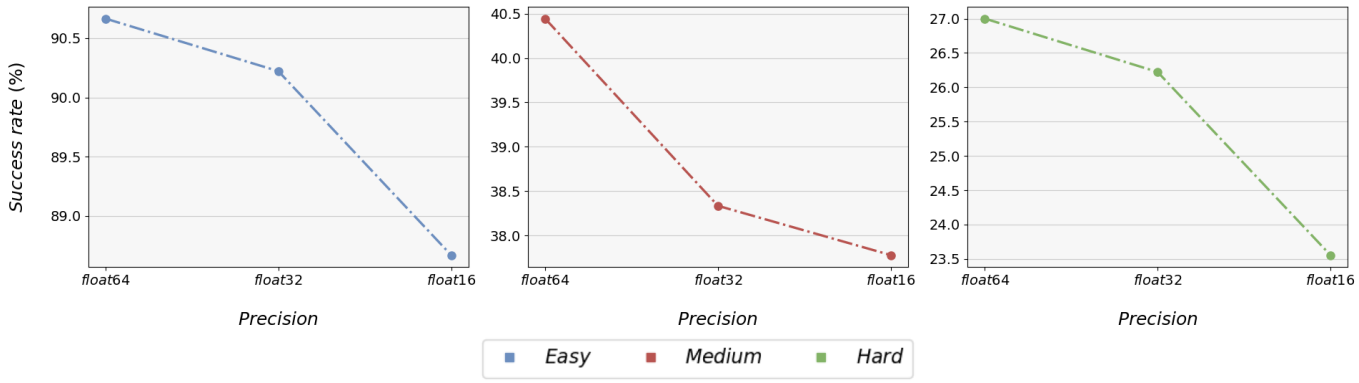


Fig. 7. Success rate as a function of the numeric precision employed for simulations in GeNN. By varying the floating point precision, changes in the success rate have been observed for all the classes: moving from `float64` down to `float16`, the percentage of successfully solved Sudoku puzzles decreased.

The results, summarized in Figure 7, confirmed that the success rate is indeed influenced by the numeric precision, with linear reduction as precision decreases. The transition from `float64` to `float32` entails an overall reduction of 1.11% (from 52.70% to 51.59%) when transitioning from `float64` to `float32` and a further reduction of 1.59% (from 51.59% to 50.00%) when moving from `float32` to `float16`.

The further advantage provided by our fully spiking pipeline when running on SpiNNaker to implement strategy *d* can be appreciated from Figure 8, where both the number of spikes to extract and the corresponding extraction time is shown with respect to strategy *a*. The reported reductions, ranging from 54.63% to 99.98% for the number of spikes and from 88.56% to 96.41% for the extraction time, highlight that strategy *d* allows to significantly increase the overall efficiency of the pipeline: once the simulation is completed, the amount of data to be transmitted from the SpiNNaker platform is strongly reduced, and so are the time and the energy required for such operation.

V. Discussion

The attractor network investigated in this work originates from [19] and stems from the possibility of even further exploiting the efficiency of spike-based computation. It introduces two major changes: the adoption of a constraint stabilization mechanism and the use of the four blocks *Polisher*, *NetChecker*, *If* and *Memory* to build a fully spiking pipeline for the solution of Sudoku puzzles and its validation entirely on neuromorphic hardware. Simulations in GeNN are performed to assess the impact of such changes individually.

The constraint stabilization mechanism, implemented by modifying the lateral inhibition scheme within the neuron populations that model the Sudoku cells, and investigated by the experiments carried out with strategy *b*, has a strong impact on the success rate, as it is highlighted in Figure 4. Its role is to preserve the problem formulation throughout the dynamical evolution of the network. Changes of the initial clues during the system evolution were indeed observed when reproducing the original implementation of [19]. The origin of such effect

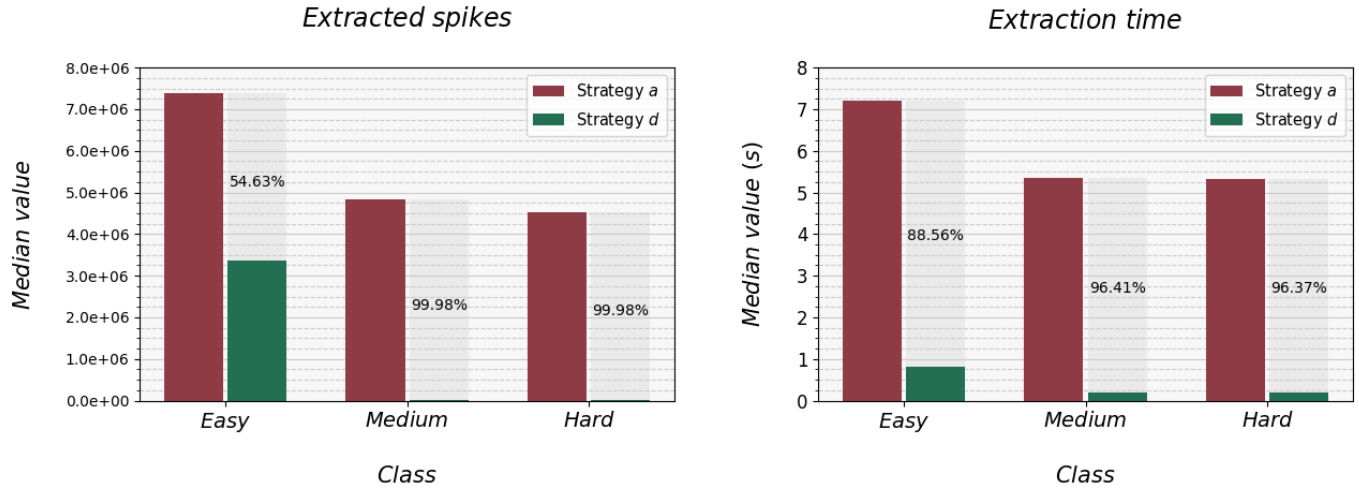


Fig. 8. Comparison of strategy *a* and strategy *d* in terms of spike extraction from SpiNNaker. The 100 experiments run for three different puzzles of each class highlighted that both the number of spikes to be extracted and, consequently, the extraction time required are strongly reduced thanks to the combined effect of the built-in validation and the neuron idling mechanism. This translates into a significant gain in efficiency thanks to the reduction in the computational cost required to external, non-neuromorphic, platforms.

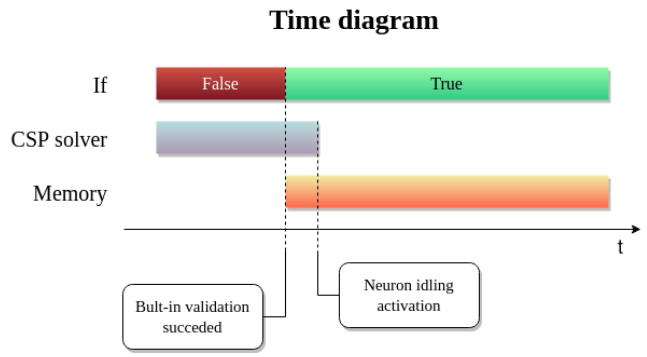


Fig. 9. In order to correctly save the final activity state of the CSP solver when the found solution is validated (i.e. the `True` population of the *If* block is activated), a temporal overlap between the functioning of this latter before it is shut down and the activation of the *Memory* block is needed. Such delay refers to the time elapsing between a successful built-in validation and the activation of the neuron idling, and it is implemented through the synaptic connections between the *If* block and the CSP solver.

was traced back to the inhibition of the Sudoku cells containing the initial values to be kept unchanged. Specifically, being their synaptic connectivity of inhibitory type, the state of such cells can be corrupted if other neuron populations produce a high enough activity. Therefore, to avoid the initial clue changes, this possibility must be hindered by removing the inhibitory synaptic connections towards these cells.

By introducing the constraint stabilization, the original problem cannot be changed during the stochastic evolution of the system, thus improving the rate of the CSP solver in finding the correct solution to the correct Sudoku puzzle. From Figure 4, it can be appreciated that the constraint stabilization plays a major role for all the classes and it is vitally important in the solution of hard Sudoku puzzles.

Neuron idling and built-in validation are instead the two mechanisms implemented by means of the four blocks *Polisher*, *NetChecker*, *If* and *Memory*. They represent the distinguishing elements with respect to [19] in the definition of our fully spiking pipeline, and they are investigated by adopting strategy *c*. As it is summarized in Figure 4 and Figure 5, their effect is directly related to the efficiency of a successful CSP solver. From Figure 5, it can indeed be appreciated how such contribution disruptively appears for the easy puzzles, which are the ones with the highest success rate (Figure 4).

The explanation of this result is in the specific behaviour, and activation conditions, of the *If* block. Given a CSP solver for a certain Sudoku puzzle, the *Polisher* block is operated to continuously receive the activity of the evolving network and to identify the most active populations. It can be thought of as a noise suppression mechanism applied to the spiking activity of the CSP solver. The cleaned activity state of all the populations is then passed to the *NetChecker*, which verifies if all the constraints of the Sudoku, i.e. along the rows, along the columns and within the sub-squares, are satisfied. This is the newly introduced built-in validation. When all the conditions are verified, one of the two components of the *If* block, namely the `True` population, is activated, thus enabling the writing of the current state of the CSP solver into the *Memory* block and the subsequent idling of the solver itself. The writing step represents the copy into the *Memory* block of the last activity state of the CSP solver populations, and it is possible thanks to an increased synaptic delay introduced in the connections between the *If* block and the latter.

Figure 9 visually summarizes the fundamental role played by such delay. When the `True` population of the *If* block is activated, the `False` population is deactivated, the *Memory* block is activated and the CSP solver is shut down. The two latter operations must be properly delayed in order to correctly retrieve the final activity state of the CSP solver and to write it into the *Memory* block: only if the two blocks are active at the same time the writing operation can be successfully performed. It is hence fundamental the definition of an overlap between the solver shutting down and the *Memory* activation. The synaptic delay between the `True` population of the *If* block and the CSP solver produces it.

A crucial aspect in the development of our fully spiking pipeline and its deployment on SpiNNaker is tightly related to such synaptic delay, and a tailored implementation was needed due to the hardware specifications. We realized it by emulating an axonal structure through a *Delay* population made of a series of smaller, concatenated neuron populations with the characteristics summarized in Table V. As the biological axon does, the *Delay* population introduces a delay: the synaptic

TABLE V
SUMMARY OF THE PARAMETERS USED FOR THE DIFFERENT COMPONENTS OF THE WHOLE MODEL RUN ON SPINNAKER. WHERE RANGES INSTEAD OF VALUES ARE REPORTED, UNIFORM DISTRIBUTION OF VALUES WITHIN SUCH RANGES MUST BE CONSIDERED. THE OPTIMAL VALUES CHANGED WITH RESPECT TO TABLE II ARE HIGHLIGHTED IN BOLD. FOR THE *Delay* POPULATION, BOTH THE NUMBER OF INNER POPULATIONS (18) AND THEIR DIMENSION (10) ARE REPORTED.

	Neurons per population	**Synaptic connection weight**				
		Stimulus	**Internal**	**Lateral**	**to CSP solver**	**to Memory**
CSP solver	27 (*)	[1.4, 1.6]	[-0.08, 0.00]	[-0.08, 0.00]	/	/
Polisher	10	1.0	-1.0	/	/	/
NetChecker	10	1.00 0.15 (check pop.)	/	-1.2	/	/
If	10	**1.00** **0.11** (val. pop.)	**1.1** (`True`) 0.0 (`False`)	-1.0	/	-0.6
Delay	**10** (×**18**)	**2.5**	**/**	**-1.0**	**-2.0**	**/**
Memory	3	1.0	**0.8**	-0.3	/	/

(*) The solution for two of the three puzzles belonging to the easy class (#2 and #3 specifically) has been simulated by using 28 neurons per populations in the CSP solver due to an observed gain in performance.

delay needed during the spike propagation from the *If* block to the CSP solver. The axonal length, i.e. the number of concatenated populations, was designed to overcome the upper limit of 144 time steps imposed by single synapses in SpiNNaker and insufficient to reproduce the behaviour simulated in GeNN.

To effectively incorporate such changes with respect to the GPU-based simulations in GeNN, where it was actually possible to effortlessly implement the synaptic delay as a single variable, an optimization of the *Delay* population dimension and of the synaptic weights for the *If* and *Memory* block was performed through the Neural Network Intelligence (NNI) toolkit [37]. The optimal values found are highlighted in bold in Table V to highlight the changes with respect to Table II.

The relevance of the neuron idling mechanism is particularly evident in the comparison of strategy *a* and strategy *d* on SpiNNaker. As it is reported in Figure 8, the fully spiking pipeline provides huge gains, especially for the medium and the hard class, for the number of extracted spikes the extraction time. Both of these quantities are related to the CSP solver activity through the state stored in the *Memory* block: the number of extracted spikes represents the amount of spikes produced by the latter; the extraction time refers to the internal operations performed by SpiNNaker to extract packages of these spikes.

The built-in validation introduced with our fully spiking pipeline allows to avoid the binning procedure adopted by [19] to validate the solutions. By replacing it with the *NetChecker*, the computational effort required to analyze the extracted spikes on external hardware is completely prevented. As a result, the overall efficiency is improved from a twofold perspective: on the one hand, a significantly reduced data transmission is needed from SpiNNaker, on the other hand, the computational cost for solution validation on external hardware is reduced to zero.

The process of creating the CSP solver is ultimately contingent upon the specific puzzle under analysis and the synaptic connections defined through probability distributions. Each puzzle configuration would require a custom mapping in the CSP solver populations. On the other hand, our enhanced pipeline employs a process of topological mapping of the constraints and their validation, which allows the blocks producing neuron idling and built-in validation to have a univocal definition of the synaptic connections and their weights. This enables an independent validation process for the puzzle. Clearly, as our enhanced pipeline offers the potential to create a bespoke validation network for Latin square problems, its applicability is constrained to such a specific class. To extend and adapt the proposed approach to other classes, tailored solutions must be identified that take into account the specific characteristics and metrics of the targeted problem.

VI. Conclusions

Starting from the SNN-based solver for CSPs presented in [19], we have shown how the use of neuromorphic hardware can be extended to achieve further beneficial effects. By implementing not only the problem solver but also the solution validation on SpiNNaker, together with the adoption of a mechanism to reduce unnecessary energy consumption, we have presented an enhanced, fully spiking pipeline. As a result, we have reported a strong reduction in data transmission from the platform and the consequent computational effort required to process such data. Compared to [19], we indeed exploited a built-in validation mechanism within the neuromorphic hardware, so that it is not needed to extract the whole spiking activity produced by the network, and its verification also is not required. With our enhanced, fully spiking pipeline, only the final state of the solver network is sent out from SpiNNaker, thus dramatically reducing the extraction time. Additionally, we have introduced a constraint stabilization mechanism that ensures that the initial clues are preserved, so that the problem definition and characteristics are not affected by the stochastic evolution of the solver. With such a mechanism, we have shown that the success rate in finding proper solutions for the selected use case of Sudoku puzzles increases. Specifically, puzzles from the hard class also can be solved.

In order to provide a more comprehensive and general overview of the alternative methodologies for solving Latin Square problems, a detailed analysis of the current state of the art methods is presented in the Supplementary Material. This facilitates a thorough comparison between different approaches and techniques.

Through our enhanced, fully spiking pipeline, we demonstrated the potential of leveraging neuromorphic hardware for efficient and effective solution of CSPs, paving the way for further advancements in the field of brain-inspired computing and its applications.

Acknowledgment

This research is funded by the European Union - NextGenerationEU Project 3A-ITALY (PE0000004, CUP E13C22001900001) and the Fluently project with Grant Agreement No. 101058680. We acknowledge a contribution from the Italian National Recovery and Resilience Plan (NRRP), M4C2, funded by the European Union – NextGenerationEU (Project IR0000011, CUP B51E22000150006, "EBRAINS-Italy").

References

[1] B. Machado, C. Pimentel, and A. de Sousa, "Integration planning of freight deliveries into passenger bus networks: Exact and heuristic algorithms," *Transportation Research Part A: Policy and Practice*, vol. 171, p. 103645, 2023.

[2] W. Hu, J. Dong, K. Yang, R. Ren, and Z. Chen, "Network planning of metro-based underground logistics system against mixed uncertainties: A multi-objective cooperative co-evolutionary optimization approach," *Expert Systems with Applications*, p. 119554, 2023.

[3] X. Bai, A. Fielbaum, M. Kronmüller, L. Knoedler, and J. Alonso-Mora, "Group-based distributed auction algorithms for multi-robot task assignment," *IEEE Transactions on Automation Science and Engineering*, 2022.

[4] L. Hu, C. Fu, Z. Ren, Y. Cai, J. Yang, S. Xu, W. Xu, and D. Tang, "Sselm-neg: spherical search-based extreme learning machine for drug–target interaction prediction," *BMC bioinformatics*, vol. 24, no. 1, p. 38, 2023.

[5] H. Chen and J. Bajorath, "Designing highly potent compounds using a chemical language model," *Scientific Reports*, vol. 13, no. 1, p. 7412, 2023.

[6] R. De Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European journal of operational research*, vol. 182, no. 2, pp. 481–501, 2007.
[7] O. Badejo and M. Ierapetritou, "A mathematical modeling approach for supply chain management under disruption and operational uncertainty," *AIChE Journal*, vol. 69, no. 4, p. e18037, 2023.
[8] P. Srinivasarao and A. R. Satish, "Multi-objective materialized view selection using flamingo search optimization algorithm," *Software: Practice and Experience*, vol. 53, no. 4, pp. 988–1012, 2023.
[9] L. Zhen, J. Wu, H. Li, Z. Tan, and Y. Yuan, "Scheduling multiple types of equipment in an automated warehouse," *Annals of Operations Research*, vol. 322, no. 2, pp. 1119–1141, 2023.
[10] B. Çaliş and S. Bulkan, "A research survey: review of ai solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, pp. 961–973, 2015.
[11] J. Li, C. Zhao, F. Jia, S. Li, S. Ma, and J. Liang, "Optimization of injection molding process parameters for the lining of iv hydrogen storage cylinder," *Scientific Reports*, vol. 13, no. 1, pp. 1–15, 2023.
[12] S. Yang, H. Wang, Y. Pang, Y. Jin, and B. Linares-Barranco, "Integrating visual perception with decision making in neuromorphic fault-tolerant quadruplet-spike learning framework," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.
[13] S. Yang, H. Wang, Y. Pang, M. R. Azghadi, and B. Linares-Barranco, "Nadol: Neuromorphic architecture for spike-driven online learning by dendrites," *IEEE Transactions on Biomedical Circuits and Systems*, 2023.
[14] S. Bouajaja and N. Dridi, "A survey on human resource allocation problem and its applications," *Operational Research*, vol. 17, pp. 339–369, 2017.
[15] N. Risi, A. Aimar, E. Donati, S. Solinas, and G. Indiveri, "A spike-based neuromorphic architecture of stereo vision," *Frontiers in neurorobotics*, vol. 14, p. 568283, 2020.
[16] M. Abarca, G. Sanchez, L. Garcia, J. G. Avalos, T. Frias, K. Toscano, and H. Perez-Meana, "A scalable neuromorphic architecture to efficiently compute spatial image filtering of high image resolution and size," *IEEE Latin America Transactions*, vol. 18, no. 02, pp. 327–335, 2020.
[17] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
[18] Z. Jonke, S. Habenschuss, and W. Maass, "Solving constraint satisfaction problems with networks of spiking neurons," *Frontiers in neuroscience*, vol. 10, p. 118, 2016.
[19] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in neuroscience*, vol. 11, p. 714, 2017.
[20] S. Saeednia, "How to make the hill cipher secure," *Cryptologia*, vol. 24, no. 4, pp. 353–360, 2000.
[21] A. Hilton, "The reconstruction of latin squares with applications to school timetabling and to experimental design," *Combinatorial Optimization II*, pp. 68–77, 1980.
[22] C. J. Colbourn, T. Klove, and A. C. Ling, "Permutation arrays for powerline communication and mutually orthogonal latin squares," *IEEE Transactions on Information Theory*, vol. 50, no. 6, pp. 1289–1291, 2004.
[23] S. Huczynska, "Powerline communication and the 36 officers problem," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 364, no. 1849, pp. 3199–3214, 2006.
[24] N. Lakić, "The application of latin square in agronomic research." *Journal of Agricultural Sciences (Belgrade)*, vol. 46, pp. 71–77, 2001.
[25] R. Malaka and S. Buck, "Solving nonlinear optimization problems using networks of spiking neurons," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6. IEEE, 2000, pp. 486–491.
[26] S. Habenschuss, Z. Jonke, and W. Maass, "Stochastic computations in cortical microcircuit models," *PLoS computational biology*, vol. 9, no. 11, p. e1003311, 2013.
[27] M. Z. Alom, B. Van Essen, A. T. Moody, D. P. Widemann, and T. M. Taha, "Quadratic unconstrained binary optimization (qubo) on neuromorphic computing system," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3922–3929.
[28] Z. Chen, Z. Xiao, M. Akl, J. Leugring, O. Olajide, A. Malik, N. Dennler, C. Harper, S. Bose, H. A. Gonzalez *et al.*, "On-off neuromorphic ising machines using fowler-nordheim annealers," *arXiv preprint arXiv:2406.05224*, 2024.
[29] M. Khona and I. R. Fiete, "Attractor and integrator networks in the brain," *Nature Reviews Neuroscience*, pp. 1–23, 2022.
[30] B. Boreland, G. Clement, and H. Kunze, "Set selection dynamical system neural networks with partial memories, with applications to sudoku and kenken puzzles," *Neural Networks*, vol. 68, pp. 46–51, 2015.
[31] C. Ostrau, C. Klarhorst, M. Thies, and U. Rückert, "Comparing neuromorphic systems by solving sudoku problems," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019, pp. 521–527.
[32] L. Tao, P. Li, M. Meng, Z. Yang, X. Liu, J. Hu, J. Dong, S. Qiao, T. Ye, and D. Shang, "Blended glial cell's spiking neural network," *IEEE Access*, vol. 11, pp. 43 566–43 582, 2023.
[33] M. Tsodyks, A. Uziel, and H. Markram, "Synchrony generation in recurrent networks with frequency-dependent synapses," *The Journal of neuroscience*, vol. 20, no. 1, p. RC50, 2000.
[34] T. Mantere and J. Koljonen, "Solving, rating and generating sudoku puzzles with ga," in *2007 IEEE congress on evolutionary computation*. IEEE, 2007, pp. 1382–1389.
[35] E. Yavuz, J. Turner, and T. Nowotny, "Genn: a code generation framework for accelerated brain simulations," *Scientific reports*, vol. 6, no. 1, p. 18854, 2016.
[36] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE transactions on computers*, vol. 62, no. 12, pp. 2454–2467, 2012.
[37] Microsoft, "Neural Network Intelligence," 2021. [Online]. Available: https://github.com/microsoft/nni



Riccardo Pignari is a Ph.D. student at Politecnico di Torino in Department of Control and Computer Engineering (DAUIN) and member of SmartData@Polito research center. He obtained the B.Sc. in Physical Engineering and the M.Sc. in Physics Of Complex Systems both at Politecnico di Torino, (Torino, Italy). His main focus at the Electronic Design Automation (EDA) Group as a researcher is on neuromorphic and neuro-inspired computing.
Email address: riccardo.pignari@polito.it



Vittorio Fra is a Researcher and Assistant Professor at Politecnico di Torino in the Interuniversity Department of Regional and Urban Studies and Planning (DIST). He holds a B.Sc. in Physical Engineering and a M.Sc. in Nanotechnologies for ICTs, received from Politecnico di Torino where he also obtained his Ph.D. in Physics. In the Electronic Design Automation (EDA) Group, he focuses his activity on neuromorphic and neuro-inspired computing.
Email address: vittorio.fra@polito.it



Enrico Macii is a Full Professor of Computer Engineering with the Politecnico di Torino, Torino, Italy. He holds a Laurea degree in electrical engineering from the Politecnico di Torino, a Laurea degree in computer science from the Università di Torino, Turin, and a PhD degree in computer engineering from the Politecnico di Torino. His research interests are in the design of electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects. He is a Fellow of the IEEE.
Email address: enrico.macii@polito.it



Gianvito Urgese is an Assistant Professor with the Politecnico di Torino (Italy) where he received a PhD in Computer and Systems Engineering in 2016. His main research interests are Neuromorphic Engineering, Parallel and Heterogeneous Architectures, AIoT application development, and Algorithm Optimisation focused on Bioinformatics and Embedded-Systems domains. He is a Senior Member of the IEEE.
Email address: gianvito.urgese@polito.it