Neuromorphic Evolutionary Computation

Jorge M. Cruz-Duarte • El-Ghazali Talbi

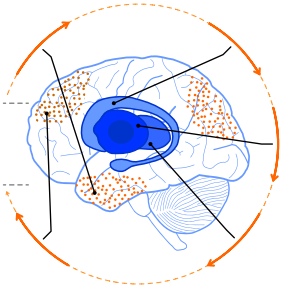
France 2030
RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité
Inria
Université de Lille
cnrs
CRIStAL

1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)



3. NEVO

(Guts & Examples)

5. Reflection

(Insights & Outlook)

4. Experiments

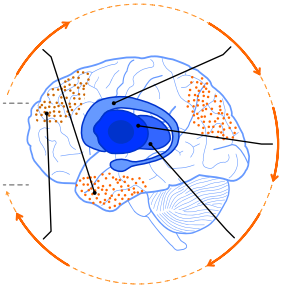
Some, (NeurOptimiser & NEVO)

1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)



3. NEVO

(Guts & Examples)

5. Reflection

(Insights & Outlook)

4. Experiments

Some, (NeurOptimiser & NEVO)

Energy Problem

Modern AI is consuming energy at a rate we cannot sustain!!!

- Energy and latency constraints increasingly limit where optimisation can be deployed,
- Embedded and autonomous systems are tightly budgeted.
- Separation between processing and memory makes data movement a dominant cost.
- Iterative BBO amplifies this cost via repeated evals. and pop. updates[1].

But this is the von Neumann realm!

[1] A. Eiben and J. Smith, "Introduction to evolutionary computing", Assembly Automation **24**, 324–324 (2004).

Implementation Gap

Neuromorphic chips exist, but the algorithms are still classical.

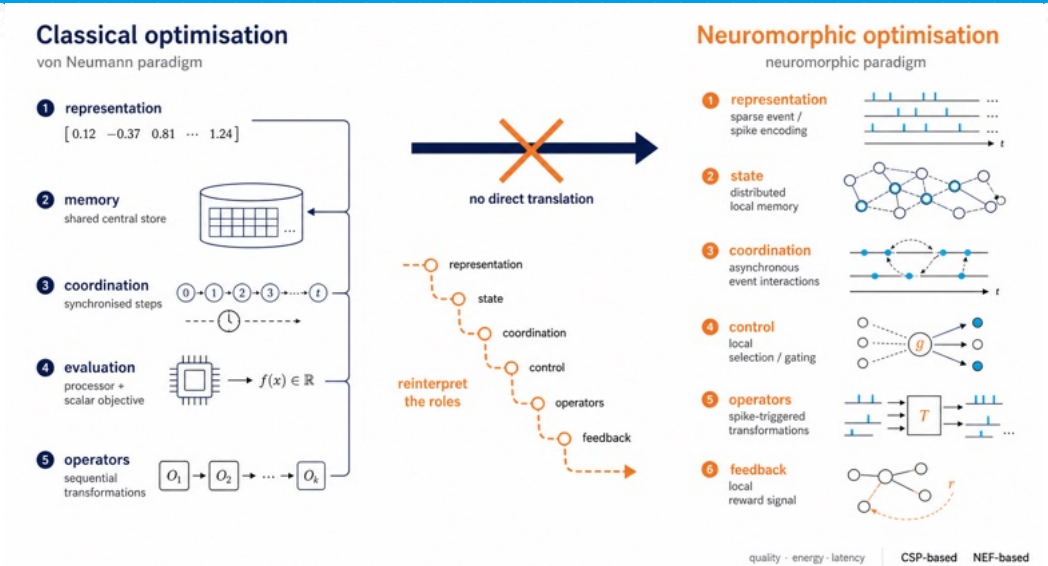
- Most current neuromorphic systems use **brain-inspired hardware** but run conventional, gradient-based software on top of it[1].
- **Evolutionary and swarm** intelligence algorithms (proven solvers for hard optimisation) have never been designed natively for spiking hardware[2].
- No framework **yet** unifies algorithm design, scalability, and hardware co-optimisation into a single coherent approach.

Neuromorphic ¿ ? **Optimisation**

[1] Sanaullah et al., "Exploring SNNs: a comprehensive analysis of mathematical models and app.", Front. Comput. Neuroscience **17**, 1215824 (2023).

[2] T. Sasaki and H. Nakano, "Swarm Intell. Alg. Spiking Neural-Osc. Netw., Coupling Interact. & Search Perform.", Nonlinear Theory Appl., IEICE **14**, 267–291 (2023).

Main Idea


Classical optimisation
von Neumann paradigm
1 representation
[0.12 −0.37 0.81 ⋯ 1.24]
2 memory
shared central store
3 coordination
synchronised steps
0 1 2 3 ⋯ t
4 evaluation
processor + scalar objective
f(x) ∈ ℝ
5 operators
sequential transformations
O1 O2 ⋯ Ok
no direct translation
representation
state
coordination
control
operators
feedback
reinterpret the roles
Neuromorphic optimisation
neuromorphic paradigm
1 representation
sparse event / spike encoding
t
2 state
distributed local memory
3 coordination
asynchronous event interactions
t
4 control
local selection / gating
g
5 operators
spike-triggered transformations
T
6 feedback
local reward signal
r
quality · energy · latency
CSP-based
NEF-based

Neuromorphic Programming Abstractions

Communicating Sequential Processes (CSP)

Abstraction level: Software architecture

- Concurrent **Processes** encapsulate state & behaviour
- Interaction via typed **Channels** (message passing, no shared memory)
- Directed **In / OutPorts** define process interfaces
- Execution is **asynchronous** and event-driven
- Same process, multiple **Process Models** (CPU, Loihi 2)

Neural Engineering Framework (NEF)

Abstraction level: Mathematical & alg.

- **Representation:** vectors encoded by neural ensembles via preferred-direction tuning curves
- **Transformation:** functions computed analytically via optimal decoders $\boldsymbol{d}_i^f$
- **Dynamics:** recurrent weights implement control-theoretic ODEs $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + \boldsymbol{Bu}$
- Weights derived by **least squares** (no manual tuning)

What we have been doing so far

NeurOptimiser
Neuromorphic Optimiser

Paradigm: Comm. Sequential Processes (CSP)
Framework/Target: Lava NC[1]/Intel Loihi 2

NEVO
Neuromorphic EVolutionary Optimiser

Paradigm: Neural Engineering Framework (NEF)
Framework/Target: Nengo[2]/Agnostic

[1] Intel Neuromorphic Computing Lab, *Lava: a software framework for neuromorphic computing*, `https://github.com/lava-nc`, 2023

[2] C. Eliasmith and T. Stewart, "Nengo and NEF: Connecting Cognitive Theory to Neuroscience", in Proc. Annu. Meet. Cogn. Sci. Soc. Vol. 33 (2011)

Key Concepts I

Minimisation Problem

We define the *minimisation problem* $(\mathfrak{X}, f, \min)$ as $\boldsymbol{x}_* \in \mathfrak{X}$, such as $\boldsymbol{x}_* = \operatorname{argmin}_{\boldsymbol{x}\in\mathfrak{X}} \{f(\boldsymbol{x})\}$, where $\boldsymbol{x}_* \in \mathfrak{X}$ is the optimal solution and $\mathfrak{X} \subseteq \mathbb{R}^d$ is the feasible domain.

Population and Neighbourhood

A *population* $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ consists of a finite set of n candidate solutions.
A *neighbourhood* is defined as $\boldsymbol{N} \subseteq \boldsymbol{A}$, since $\boldsymbol{A}$ is an arbitrary set; e.g., $\boldsymbol{A} = \boldsymbol{X}, \boldsymbol{A} = \boldsymbol{N}_i^t, \boldsymbol{A} = \boldsymbol{P}^t$. (Considering that $\boldsymbol{N}_i^t = \{\boldsymbol{x}_i^\tau\}_{\tau=1}^{t}$ and $\boldsymbol{P}^t = \{\boldsymbol{p}_i^t\}_{i=1}^{n}$.)

Particular and Global Best

The i^{th} *particular best* candidate at the step t is obtained via $\boldsymbol{p}_i^t = \operatorname{argmin}_{\boldsymbol{x}\in\boldsymbol{N}_i^t}\{f(\boldsymbol{x})\}$.

The *global best* candidate solution at step t is defined as $\boldsymbol{g}^t = \operatorname{argmin}_{\boldsymbol{x}\in\boldsymbol{P}^t}\{f(\boldsymbol{x})\}$.

Key Concepts II

- Consider the generalised MH model[1] and adopt the low- and high-level scheme.
- It can be seen as a state-based Adaptive Operator Section or HH controller[2,3].
- At the low level:
  - **Problem**: Find $\boldsymbol{x}_* = \text{argmin}_{\boldsymbol{x}\in\mathfrak{X}} f(\boldsymbol{x})$, where $\mathfrak{X} \subseteq \mathbb{R}^D$ and $f : \mathfrak{X} \to \mathbb{R}$
  - **Solver**: A sequence of search operators from a collection, $h_o \triangleq h_s \circ h_g \in \mathfrak{H}_o \subset \mathfrak{H}$
- At the high level:
  - **Problem**: Find
$$h_*^k = \text{argmax}_{h_o\in\mathfrak{H}_o\subset\mathfrak{H}} \{U(h_o, \boldsymbol{s}^k)\},$$
where $\boldsymbol{s}^k$ is search state and U the expected utility
  - **Solver**: A sequence of ..., or NeurOptimiser, or NEVO!

[1] J. M. Cruz-Duarte et al., "Towards a generalised metaheuristic model for continuous optimisation problems", Mathematics **8**, 2046 (2020).

[2] J. Pei et al., "Adaptive operator selection for meta-heuristics: a survey", IEEE Transactions on Artificial Intelligence (2025).

[3] A. Hassan and N. Pillay, "Dynamic heuristic set selection for cross-domain selection hyper-heuristics", in Int. Conf. Theory Pract. Nat. Comput. (2021), pp. 33–44.

1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)

3. NEVO

(Guts & Examples)

5. Reflection

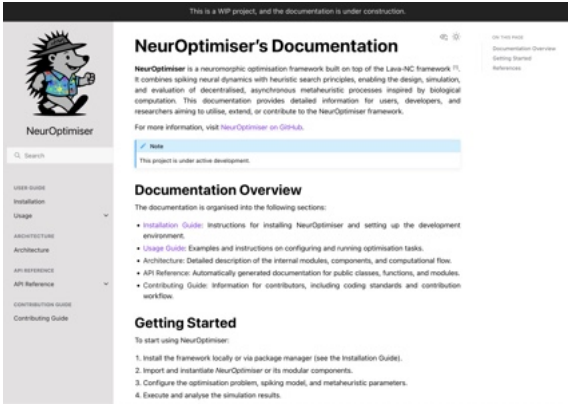
(Insights & Outlook)

4. Experiments

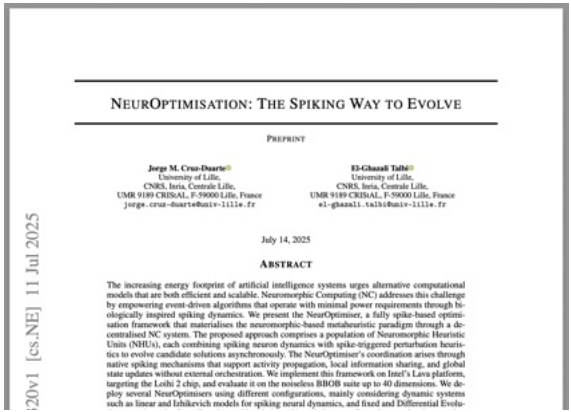
Some, (NeurOptimiser & NEVO)

Resources

https://neuroptimiser.github.io/



https://arxiv.org/abs/2507.08320



NEUROPTIMISATION: THE SPIKING WAY TO EVOLVE

PREPRINT

Jorge M. Cruz-Duarte
University of Lille,
CNRS, Inria, Centrale Lille,
UMR 9189 CRIStAL, F-59000 Lille, France
jorge.cruz-duarte@univ-lille.fr

El-Ghazali Talbi
University of Lille,
CNRS, Inria, Centrale Lille,
UMR 9189 CRIStAL, F-59000 Lille, France
el-ghazali.talbi@univ-lille.fr

July 14, 2025

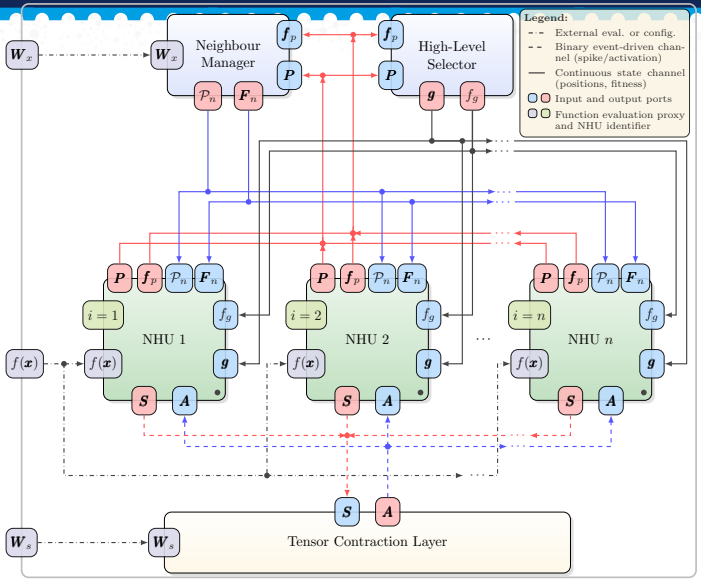
ABSTRACT

The increasing energy footprint of artificial intelligence systems urges alternative computational models that are both efficient and scalable. Neuromorphic Computing (NC) addresses this challenge by empowering event-driven algorithms that operate with minimal power requirements through biologically inspired spiking dynamics. We present the NeurOptimiser, a fully spike-based optimisation framework that materialises the neuromorphic-based metaheuristic paradigm through a decentralised NC system. The proposed approach comprises a population of Neuromorphic Heuristic Units (NHUs), each combining spiking neuron dynamics with spike-triggered perturbation heuristics to evolve candidate solutions asynchronously. The NeurOptimiser's coordination arises through native spiking mechanisms that support activity propagation, local information sharing, and global state updates without external orchestration. We implement this framework on Intel's Lava platform, targeting the Loihi 2 chip, and evaluate it on the noiseless BBOB suite up to 40 dimensions. We deploy several NeurOptimisers using different configurations, mainly considering dynamic systems such as linear and Izhikevich models for spiking neural dynamics, and fixed and Differential Evolu-

Architecture

The NeurOptimiser comprises:

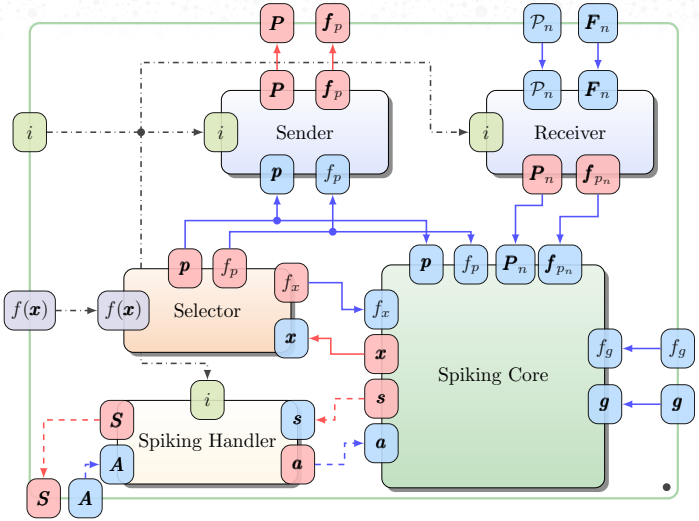
- a Tensor Contraction Layer,
- a Neighbourhood Manager,
- a High-Level Selector, and
- n Neuromorphic Heuristic Units (NHUs),



Neuromorphic Heuristic Unit (NHU)

The NHU comprises:

- a Spiking Handler,
- a Receiver, a Sender,
- a Selector, and
- a Spiking Core.



Spiking Core: Overview

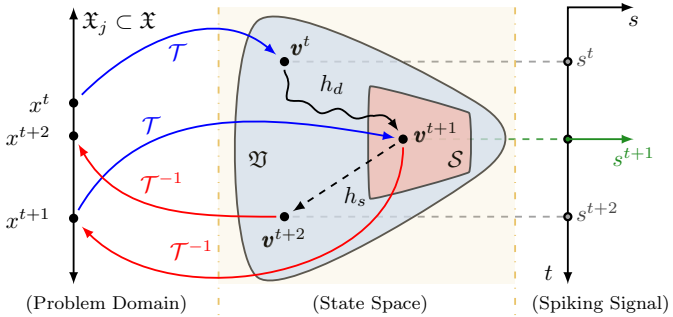


Figure: Sketch of three representative search steps performed by a spiking neuron.

Variables

- $x_j^t \in \mathfrak{X}_j$: Candidate sol. element
- $\boldsymbol{v}_j^t \in \mathfrak{V}$: Neuromorphic state
- s_j^t: Spike signal

State evolution

- $\mathcal{T}$: Bidirectional transformation
- $\mathcal{S}$: Spiking condition set
- h_d: Dynamic updating rule
- h_s: Spiking-triggered rule

Spiking Core: Key Components I

Bidirectional Transformation $\mathcal{T}$

This is defined as $\mathcal{T} : \mathfrak{X}_j \leftrightarrow \mathfrak{V}$. We implemented $\mathcal{T}$

$$\boldsymbol{v}_j = \mathcal{T}(x_j) \triangleq \Big(\alpha(x_j - x_{\text{ref},j}), \quad 2r - (1 - x_{\text{ref},j})\Big)^{\mathsf{T}},$$

where is a reference point $x_{\text{ref},j} \in \boldsymbol{x}_{\text{ref}} \in \mathfrak{X}$, a gain factor $\alpha \in \mathbb{R}$, and $r \sim \mathcal{U}(0, 1)$.

- We considered five distinct reference modes for defining $\boldsymbol{x}_{\text{ref}}$: `null` ($\mathbf{0}_d$), `p` ($\boldsymbol{p}$), `g` ($\boldsymbol{g}$), `pg` (midpoint)[1], and `pgn` (midpoint extended with neighbours).

[1] T. Sasaki and H. Nakano, "Swarm intelligence algorithm based on spiking neural-oscillator networks, coupling interactions and search performances", Nonlinear Theory and Its Applications, IEICE **14**, 267–291 (2023).

Spiking Core: Key Components II

Spiking Condition Set $\mathcal{S}$

This is defined through a characteristic function $\varphi_{\mathcal{S}}(\boldsymbol{v}_j) : \mathfrak{V} \mapsto \mathbb{Z}_2$, such that

$$\mathcal{S} = \left\{ \boldsymbol{v}_j \in \mathfrak{V} \mid \varphi_{\mathcal{S}}(\boldsymbol{v}_j) = 1 \right\}.$$

- We implemented four spiking modes (`spk_cond`):
  - `fixed` uses a membrane potential threshold, $\varphi_{\mathcal{S}}(\boldsymbol{v}_j) \triangleq |v_{j,1}| > \vartheta_j$[1];
  - `l1` and `l2` that generalise it, $||\boldsymbol{v}_j||_1 > \vartheta_j$ and $||\boldsymbol{v}_j||_2 > \vartheta_j$; and
  - `stable` for shrinking radius around the origin, $||\boldsymbol{v}_j||_2 < \varepsilon_{\text{spk}}/(1 + t)$.
- Considering the threshold parameter ϑ_j via distinct modes:
  - `fixed` mode considers the threshold constant $\vartheta_j \triangleq \alpha_{\text{thr}}$, where $\alpha_{\text{thr}} > 0$;
  - `diff_pg` strategy uses the global and particular best positions as $\vartheta_j \triangleq \alpha_{\text{thr}} |\boldsymbol{g}_j - \boldsymbol{p}_j|$; and
  - `diff_pref` mode defines it w.r.t. the reference position as $\vartheta_j \triangleq \alpha_{\text{thr}} |\boldsymbol{x}_{\text{ref},j} - \boldsymbol{p}_j|$

[1] E. M. Izhikevich, "Simple model of spiking neurons", IEEE Transactions on Neural Networks **14**, 1569 (2003).

Spiking Core: Key Components III

State variable $\boldsymbol{v}_j^t$ evolution

These govern the evolution of the state variable $\boldsymbol{v}_j^t$, which depends on whether the neuron is intrinsically spiking s_j^t or externally activated a_j^t by a neighbouring unit, such as,

$$\boldsymbol{v}_j^{t+1} \leftarrow \begin{cases} h_d(\boldsymbol{v}_j^t), & \text{if } s_j^t \vee a_j^t \neq 1, \\ h_s(\boldsymbol{v}_j^t), & \text{if } s_j^t \vee a_j^t = 1, \end{cases} \quad \text{and} \quad s_j^{t+1} \leftarrow \varphi_{\mathcal{S}}(\boldsymbol{v}_j^t),$$

- The dynamic updating rule h_d is defined by the evolution of a dynamic system. If continuous, $\dot{\boldsymbol{v}}_j = g(\boldsymbol{v}_j)$, an approximation method must be used.
- The spike-triggered rule h_s modifies the state using a heuristic operator.

Spiking Core: Key Components IV

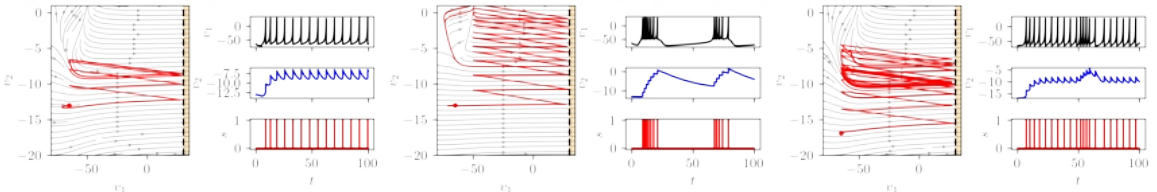
- h_d implemented from two-dimensional continuous systems with an RK4 approx.:
  - Izhikevich models[1], $g(\boldsymbol{v}_j) \triangleq \left(0.04v_{j,1}^2 + 5v_{j,1} + 140 - v_{j,2} + I_j, \quad a_j(b_j v_{j,1} - v_{j,2})\right)^\intercal$.
  - Linear Models[2], $g(\boldsymbol{v}_j) \triangleq \boldsymbol{A}_j \boldsymbol{v}_j$.
- h_s implemented with four options:
  - `random`, a random state.
  - `fixed`, the best-known state so far with low noise.
  - `directional`, a current-to-best directional state.
  - `differential` based on the mutation operators from Differential Evolution (DE)[3], using the neighbourhood for sampling random states.
    For example, `DE/rand`, and `/current-to-rand`, `/best-to-rand`, and `/current-to-best`.

[1] E. M. Izhikevich, "Simple model of spiking neurons", IEEE Transactions on Neural Networks **14**, 1569 (2003).

[2] H. S. Steven, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*, (CHAPMAN & HALL CRC, 2024).

[3] A. W. Mohamed et al., "Differential evolution mutations: taxonomy, comparison and convergence analysis", IEEE Access **9**, 68629–68662 (2021).

Spiking Core: Examples



(a) Izhikevich - Fast Spiking (b) Izhikevich - Chattering (c) Izhikevich - Resonator

(d) Linear - Sink Node (e) Linear - Spiral Source (f) Linear - Saddle Point

Figure: Portraits (v_1 vs. v_2) and time signals (v_1, v_2, s vs. t) of representative neuromorphic dynamics.

Some Parameters to Configure

Axis	Parameter	Code Name	Values
Dynamic Rule	Operator (h_d)	`hd_operator`	{`linear`†, `izhikevich`}
	Coefficient preset(s)	`coeffs_lin`	{`random`†, `sink`, `source`, `attractor`, `repeller`,`saddle`, `centre`}
		`coeffs_izh`	{`random`†, `RS`, `IB`, `CH`, `FS`, `TC`, `TCn`, `RZ`, `LTS`}
	Per-neuron diversity	`coeffs_per_neuron`	{`True`, `False`†}
	Numerical method	`approx`	{`euler`, `rk4`†}
	Small variation	`izh_small_var`	{`True`, `False`†}
Spiking Rule	Operator (h_s)	`hs_operator`	{`fixed`†, `random`, `directional`, `differential`}
	Variant(s) of h_s	`hs_variant_fixed`	{`fixed` — `pbest`†, `gbest`, `center`, `pgbest`}
		`hs_variant_random`	{`uniform`†, `normal`}
		`hs_variant_directional`	{`fixed` — `gbest`†, `pbest`, `center`}
		`hs_variant_differential`	{`rand`, `current`, `pbest`, `best`, `current-to-rand`, `best-to-rand`, `pbest-to-rand`, `current-to-best`†, `current-to-pbest`,`rand-to-best`, `rand-to-pbest`}
Topology	Neuron topology (W_s)	`neuron_topology`	{`1dr`†, `2dr`, `random`, `full`}
	Unit topology (W_x)	`unit_topology`	{`1dr`†, `2dr`, `random`, `full`}
	Number of neighbours (m)	`num_neighbours`	{2, . . . , 5†, . . . , 19} (from 2 to $n-1$)
Mapping	Threshold mode (ϑ)	`thr_mode`	{`fixed`†, `diff_pg`, `diff_pref`, `random`}
	Spiking condition (φ_s)	`spk_cond`	{`fixed`†, `11`, `12`, `random`, `adaptive`, `stable`}
	Reference mode (x_{ref})	`ref_mode`	{`None`, `p`, `g`, `pg`, `pgn`†}
Heterogeneity	Number of core types	`num_cores`	{1, 2, 3, 4, . . . }
	Per-type h_d	`c{k}_hd_operator`	{`linear`, `izhikevich`}
	Per-type h_s	`c{k}_hs_operator`	{`fixed`, `random`, `directional`, `differential`}
	Core weight	`c{k}_weight`	[0, 1]

Installation

- Use Python 3.10 (tested and recommended version)
- From PyPI

```
pip install neuroptimiser        # Using pip

uv pip install neuroptimiser     # Using uv (recommended)
```

- From GitHub

```
git clone https://github.com/neuroptimiser/neuroptimiser.git
cd neuroptimiser

pip install -e .     # Using pip

uv pip install -e . # Using uv (recommended)
```

Example 0: The Simplest NeurOptimiser I

1. Importing libraries

```
import matplotlib.pyplot as plt
import numpy as np

from neuroptimiser import NeurOptimiser
```

2. Define the problem

```
problem_function    = lambda x: np.linalg.norm(x)
problem_bounds      = np.array([[-5.0, 5.0], [-5.0, 5.0]])
```

Example 0: The Simplest NeurOptimiser II

3. Instantiate the Neuroptimiser with the default configurations

```
optimiser = NeurOptimiser()
```

3* Show the overall configuration parameters

```
print("DEFAULT CONFIG PARAMS:\n", optimiser.config_params, "\n")
print("DEFAULT CORE PARAMS:\n", optimiser.core_params)
print("DEFAULT HIGH-LEVEL ...", optimiser.selector_params, "\n")
```

```
DEFAULT CONFIG PARAMS:
 {'num_dimensions': 2, 'seed': 69, 'num_neighbours': 1, ...
DEFAULT CORE PARAMS:
 [{'spk_q_ord': 2, 'thr_max': 1.0, 'is_bounded': False, ...
DEFAULT HIGH-LEVEL SELECTOR PARAMS:
 {'mode': 'greedy'}
```

Example 0: The Simplest NeurOptimiser III

4. Run the `solve` method.

```
optimiser.solve(
    obj_func=problem_function,
    search_space=problem_bounds,
    debug_mode=True,                    # Verbose mode
    num_iterations=1000,
)
```

Examples

Let's continue with the tutorial in ...

- Example 0: https://neuroptimiser.github.io/examples/Example0.html
- Example 1: https://neuroptimiser.github.io/examples/Example1.html
- Example 2: https://neuroptimiser.github.io/examples/Example2.html

1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)

3. NEVO

(Guts & Examples)

4. Experiments

Some, (NeurOptimiser & NEVO)

5. Reflection

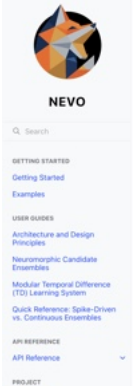
(Insights & Outlook)

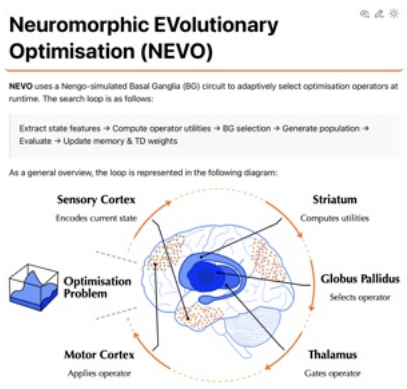
Resources

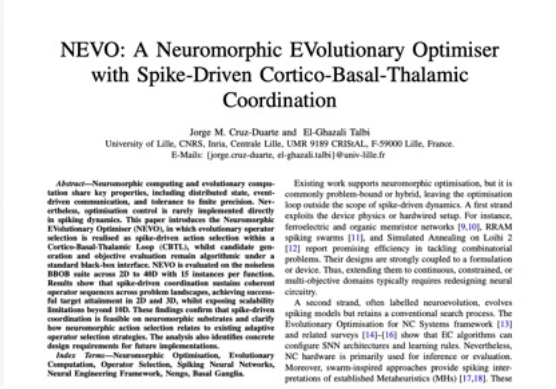
https://jcrvz.github.io/nevo

cec_pap596

Fri 26, 17h20 · 0.11 Cape Town







NEVO: A Neuromorphic EVolutionary Optimiser with Spike-Driven Cortico-Basal-Thalamic Coordination

Jorge M. Cruz-Duarte and El-Ghazali Talbi
University of Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France.
E-Mails: {jorge.cruz-duarte, el-ghazali.talbi}@univ-lille.fr

***Abstract*—Neuromorphic computing and evolutionary computation share key properties, including distributed state, event-driven communication, and tolerance to finite precision. Nevertheless, optimisation control is rarely implemented directly in spiking dynamics. This paper introduces the Neuromorphic EVolutionary Optimiser (NEVO), in which evolutionary operator selection is realised as spike-driven action selection within a Cortico-Basal-Thalamic Loop (CBTL), whilst candidate generation and objective evaluation remain algorithmic under a standard black-box interface. NEVO is evaluated on the noiseless BBOB suite across 2D to 40D with 15 instances per function. Results show that spike-driven coordination sustains coherent operator sequences across problem landscapes, achieving successful target attainment in 2D and 3D, whilst exposing scalability limitations beyond 10D. These findings confirm that spike-driven coordination is feasible on neuromorphic substrates and clarify how neuromorphic action selection relates to existing adaptive operator selection strategies. The analysis also identifies concrete design requirements for future implementations.**

***Index Terms*—Neuromorphic Optimisation, Evolutionary Computation, Operator Selection, Spiking Neural Networks, Neural Engineering Framework, Nengo, Basal Ganglia.**

Existing work supports neuromorphic optimisation, but it is commonly problem-bound or hybrid, leaving the optimisation loop outside the scope of spike-driven dynamics. A first strand exploits the device physics or hardwired setup. For instance, ferroelectric and organic memristor networks [9,10], RRAM spiking swarms [11], and Simulated Annealing on Loihi 2 [12] report promising efficiency in tackling combinatorial problems. Their designs are strongly coupled to a formulation or device. Thus, extending them to continuous, constrained, or multi-objective domains typically requires redesigning neural circuitry.

A second strand, often labelled neuroevolution, evolves spiking models but retains a conventional search process. The Evolutionary Optimisation for NC Systems framework [13] and related surveys [14]–[16] show that EC algorithms can configure SNN architectures and learning rules. Nevertheless, NC hardware is primarily used for inference or evaluation. Moreover, swarm-inspired approaches provide spiking interpretations of established Metaheuristics (MHs) [17,18]. These

NEVO: Algorithmic Architecture

Domain Transformation

An affine transform $\mathcal{T} : \mathfrak{V} \to \mathfrak{X}$ with $\mathfrak{V} = [-1, 1]^D$ and $\mathfrak{X} = [\boldsymbol{l}_b, \boldsymbol{u}_b]$.

Memory Management

A fixed-size archive of capacity M storing pairs $(\boldsymbol{v}_i, f_i)$.

Utility Modelling

Each operator has utility $u_o : [0, 1]^3 \to \mathbb{R}$ and adaptive weight w_{h_o}. Thus, utilities are computed as $\boldsymbol{u}_o^k = \boldsymbol{W}(\boldsymbol{A}\boldsymbol{s}^{k+1} + \boldsymbol{b})$.

Search State

It is given by features (diversity, improv. rate, and conv.), as

$$\boldsymbol{s}^k = (\phi_d^k, \phi_i^k, \phi_c^k)^\intercal \in [0, 1]^3$$

where

$$\phi_d^k = \sqrt{3} D^{-1} \sum_{j=1}^{D} \mathrm{Std}\{\boldsymbol{v}_{:,j}^k\},$$

$$\phi_i^k = W^{-1} \sum_{q=k-W+1}^{k} \mathbb{I}(f_\star^q < f_\star^{q-1}),$$

$$\phi_c^k = \left(1 + \lambda_c \Delta f^k / (\,|f_\star^k| + \eta_c\, \Delta f^k + \epsilon)\right).$$

NEVO: Algorithmic Architecture

Search Operator Collection

Thirteen search operators (each generates N candidate solutions) from well-known MHs a priori categorised* as for:

Exploration

- Random Search (RS)
- Lévy Flight (LF)
- Genetic Crossover (GX)
- Differential Evolution (DE)
- Central Force Optimisation (CF)
- Firefly Algorithm (FA)
- Gravitational Search (GS)

Exploitation

- Local Random Walk (LW)
- Genetic Mutation (GM)
- Simulated Annealing (SA)
- Tabu Search (TS)
- Particle Swarm Optimisation (PS)
- Spiral Optimisation (SO)

*This is a coarse description of the intended role of each implementation and does not imply a strict dichotomy.

NEVO: CBTL

The operator selection is implemented as a **Cortico-Basal-Thalamic Loop (CBTL)**[1] circuit.

- Utility ensembles (one per operator) compute $u_o(\boldsymbol{s}(t))$
- Basal ganglia WTA network performs competition
- Thalamus gates action vector $\boldsymbol{a}(t) \in \mathbb{R}^{13}$ for operator selection
- After each evaluation, reward $\boldsymbol{r}(t)$ updates weights and the action selection applies ε-greedy policy[2] over basal ganglia output
- The closed loop

$$\boldsymbol{s}(t) \rightarrow \boldsymbol{u}_o(t) \rightarrow \boldsymbol{a}(t) \rightarrow \boldsymbol{X}(t) \rightarrow r(t)$$

updates every $\Delta \boldsymbol{t}_{\text{eval}}$.

[1] D. Joseph and R. S. Bapi, "What do the basal ganglia do? A modeling perspective", Biological cybernetics **103**, 237–253 (2010).

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, (A Bradford Book, Cambridge, MA, USA, 2018).

Installation

- Use Python 3.10 (tested and recommended version)
- From GitHub

```
git clone https://github.com/jcrvz/nevo.git
cd nevo

uv pip install -e . # Using uv (recommended)
uv pip install -e . --extra dev     # includes dev dependencies
```

- From PyPI (not available on PyPI yet, but will be soon)

```
pip install nevo        # Using pip

uv pip install nevo     # Using uv (recommended)
```

Example 0: The Simplest NEVOptimiser I

1. Importing libraries.

```
from nevo import NEVOptimiser
from ioh import get_problem
```

2. Define the problem (now using IOH[1]).

```
problem = get_problem(fid=1, instance=1, dimension=10)
problem.reset()
```

3. Instantiate the NEVOptimiser

```
optimiser = NEVOptimiser(
    objective_function=problem,
    bounds=(problem.bounds.lb, problem.bounds.ub),
    dimension=10, population_size=50, memory_size=25,
)
```

[1] J. de Nobel et al., "IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics", arXiv (2021).

Example 0: The Simplest NEVOptimiser II

4. Run the `run` method.

```
optimiser.run(time=20.0)
x_best, f_best = optimiser.get_best_solution()
print(f"Best fitness: {f_best:.6e}")
```

```
Starting NEVO optimisation for 5.0s...
Problem: 10D
Population size: 50
Operators: ['LevyFlight', 'DifferentialEvolution', ...
Operator mode: trad
Expected evaluations: ~250,000
Build finished in 0:00:01.
Simulation finished in 0:00:20.
```

Example 0: The Simplest NEVOptimiser III

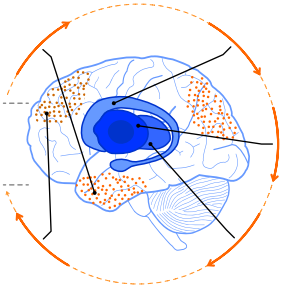
```
================================================...
OPTIMISATION RESULTS
================================================...

Total evaluations: 250,000
Best fitness: 7.948005e+01

Operator usage:
  LevyFlight                :      38 calls (  0.8%) ...
  DifferentialEvolution     :      40 calls (  0.8%) ...
  ParticleSwarm             :      32 calls (  0.6%) ...

...

Best solution (original space):
  [ 0.25156777 -1.15294226 -0.72089242  1.92949977 ...
Best fitness: 7.948005e+01
```

1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)



3. NEVO

(Guts & Examples)

5. Reflection

(Insights & Outlook)

4. Experiments

Some, (NeurOptimiser & NEVO)

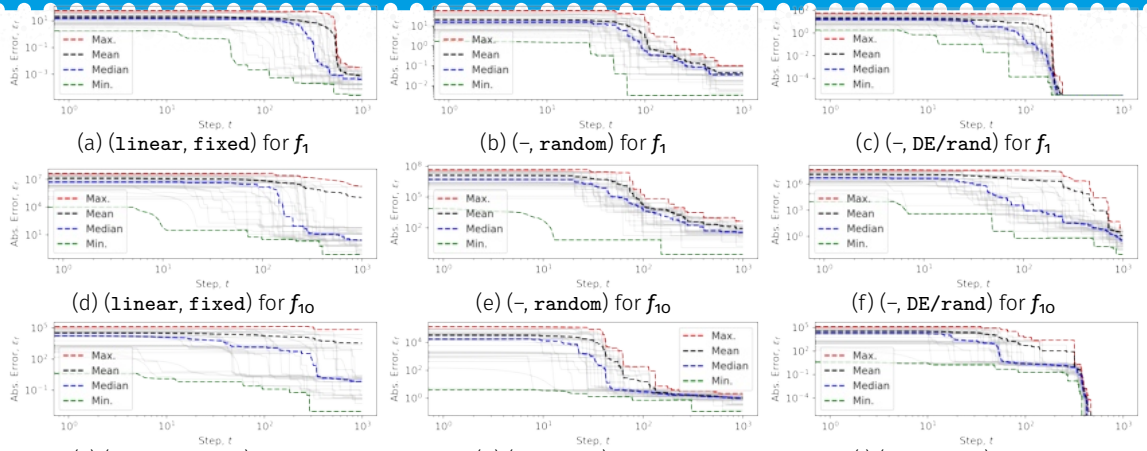
with NeurOptimiser: Description

- **Objective:**
  To test operational correctness using IOH[1] with five 2D problems (f_1, f_6, f_{10}, f_{15}, and f_{20}), one per function class. Then, we use the BBOB noiseless suite from COCO[2].
- **Details:**
  - Each configuration used 30 NHUs for 1000 steps.
  - Two dynamic models for h_d: `linear` or `izhikevich`
  - Four heuristic operators as h_s: `fixed`, `random`, `directional`, and `DE/rand`.
  - The spike condition was set to `fixed`.
  - Both topologies (W_s & W_x) were defined as one-directional rings.

[1] J. de Nobel et al., "IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics", arXiv (2021).

[2] N. Hansen et al., "COCO: a platform for comparing continuous optimizers in a black-box setting", Optimization Methods and Software **36**, 114–144 (2021).

with NeurOptimiser: Absolute Error



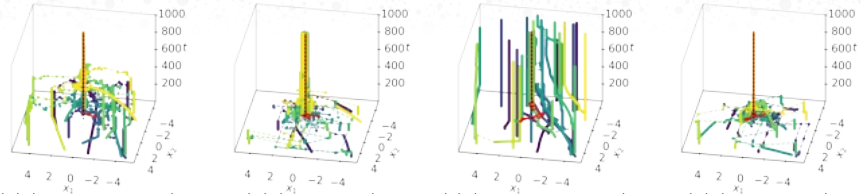
(a) (`linear`, `fixed`) for f_1 (b) (–, `random`) for f_1 (c) (–, `DE/rand`) for f_1

(d) (`linear`, `fixed`) for f_{10} (e) (–, `random`) for f_{10} (f) (–, `DE/rand`) for f_{10}

(g) (`linear`, `fixed`) for f_{20} (h) (–, `random`) for f_{20} (i) (–, `DE/rand`) for f_{20}

Figure: Absolute error $\varepsilon_f = |f(\boldsymbol{p}_i) - f_*|$ over 1000 steps for three (of five) 2D BBOB functions, each representing a different subgroup. Each row corresponds to a function, and each column to a h_s.

with NeurOptimiser: Positions

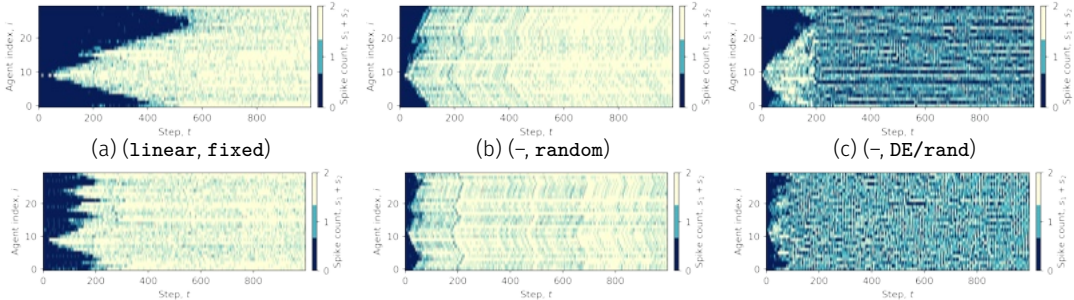


(a) (`linear`, `fixed`) (b) (–, `random`) (c) (–, `directional`) (d) (–, `DE/rand`)

(e) (`izh`, `fixed`) (f) (–, `random`) (g) (–, `directional`) (h) (–, `DE/rand`)

Figure: Best particular positions ($x_1, x_2 \in \boldsymbol{p}_i^t$) of runs with two neurOptimisers with 30 NHUs searching on a 2D Sphere (f_1) problem domain.

with NeurOptimiser: Spiking Trains

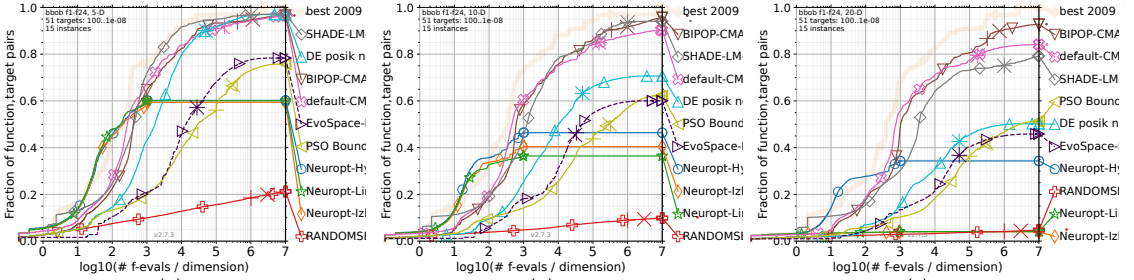


(a) (`linear`, `fixed`) (b) (–, `random`) (c) (–, `DE/rand`)

(d) (`izhikevich`, `fixed`) (e) (–, `random`) (f) (–, `DE/rand`)

Figure: Spiking signal count ($s_1 + s_2$) for each one of the 30 NHUs, implemented in neurOptimisers based on `linear` and `izhikevich` models, searching on a 2D Sphere (f_1) problem domain.

with NeurOptimiser: Extension

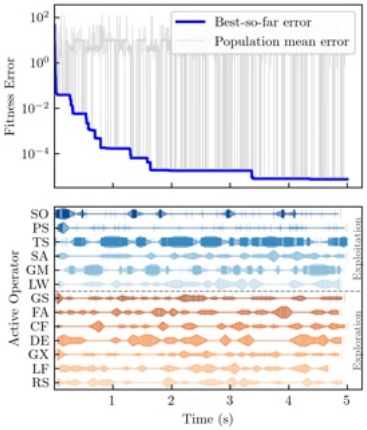


(a) 5D (b) 10D (c) 20D

Figure: Bootstrapped Empirical Cumulative Distribution Function (ECDF) of the number of f-evaluations divided by dimension for 51 targets in $\mathbf{10}^{[-8..2]}$ across all function subgroups, with 15 instances per dimension, shown for dimensions 5, 10, and 20. The best algorithm from BBOB 2009 is shown as a light, thick reference line. Legend: ☆ Neuropt-Lin, ♢ Neuropt-Izh, ◯ Neuropt-Hyb.

with NEVO: Preliminary I

Sphere (f01) in 10D: Best: 79.5, Error: 7.57×10^{-6}



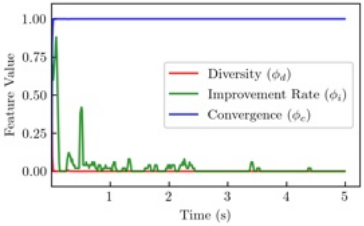
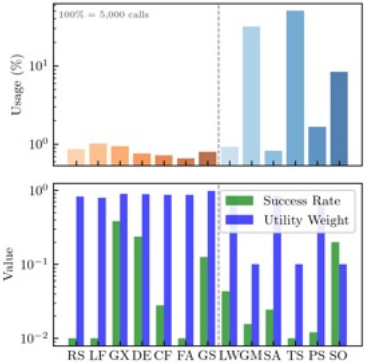


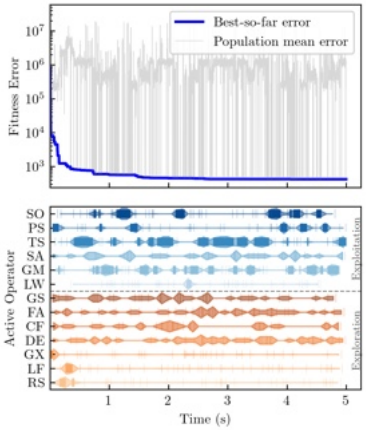
Figure: Representative single-run optimisation dynamics of the CBTL arch. on **Sphere (f01)** in 10D.

Left: Fitness Error / Active Operator
Centre: Search-State Features
Right: Operator Usage / Success Rates & Weights



with NEVO: Preliminary II

Ellipsoidal Rotated (f10) in 10D: Best: 365, Error: 420



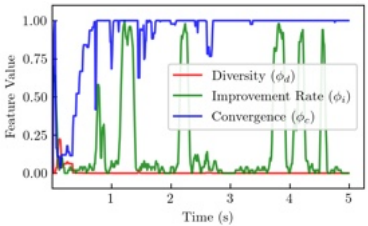
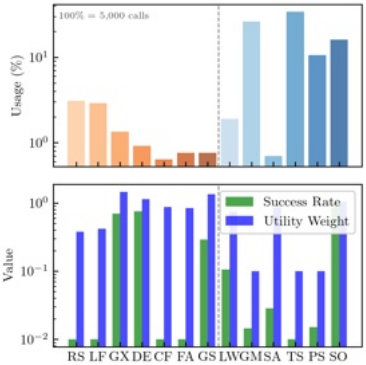


Figure: Representative single-run optimisation dynamics of the CBTL arch. on **Ellipsoidal Rotated (f10)** in 10D.

Left: Fitness Error / Active Operator
Centre: Search-State Features
Right: Operator Usage / Success Rates & Weights

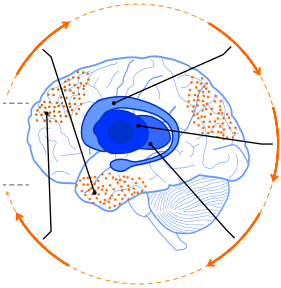


1. Preamble

(Motivation Recap)

2. NeurOptimiser

(Guts & Examples)



3. NEVO

(Guts & Examples)

5. Reflection

(Insights & Outlook)

4. Experiments

Some, (NeurOptimiser & NEVO)

Insights & Outlooks

Insights

- Spike-driven dynamics can drive BB search w/o gradients; convergence is achievable.
- Neuron model, topology, & encoding have no direct classical counterpart and matter algorithmically.
- Sub-watt operation estimated on Loihi 2 ($\leq$1.35 W); Other NC platforms remain largely untested for optimisation.
- Off-chip steps prevent full energy gains; end-to-end on-chip optimisation remains open.

Outlooks

- **Native operators:** Design for spike dynamics, not just port to classical operators.
- **On-chip memory:** Store & retrieve solution pop. directly in spiking substrates to scale beyond low dims.
- **Joint benchmarking:** Report quality, energy, & latency together; no shared protocol exists yet.
- **Theory:** Convergence, No-Free-Lunch, & complexity in synaptic ops are entirely open.
- **Open tooling:** A hardware-agnostic NEC library to lower the barrier for the EC community.

NeurOptim: Our research initiative

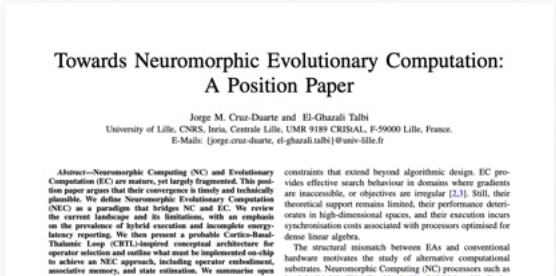
- NeurOptim[a] is a research initiative exploring how **Neuromorphic Computing** and **Computational Intelligence** can give rise to a new class of **Optimisation Algorithms**.
- Collective progress:
  - Open Science
  - Transparent methodologies
  - Reproducible experiment.

[a] `https://neuroptim.org`

Complementary Information

cec_pos106

Mon 22, 17h20 · 0.04 Brussels



Towards Neuromorphic Evolutionary Computation: A Position Paper

Jorge M. Cruz-Duarte and El-Ghazali Talbi
University of Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France.
E-Mails: {jorge.cruz-duarte, el-ghazali.talbi}@univ-lille.fr

***Abstract*—Neuromorphic Computing (NC) and Evolutionary Computation (EC) are mature, yet largely fragmented. This position paper argues that their convergence is timely and technically plausible. We define Neuromorphic Evolutionary Computation (NEC) as a paradigm that bridges NC and EC. We review the current landscape and its limitations, with an emphasis on the prevalence of hybrid execution and incomplete energy-latency reporting. We then present a probable Cortico-Basal-Thalamic Loop (CBTL)-inspired conceptual architecture for operator selection and outline what must be implemented on-chip to achieve an NEC approach, including operator embodiment, associative memory, and state estimation. We summarise open challenges spanning numerical precision, scalability, learning rules, benchmarking, theory, and tooling, including the need for convergence analysis framed in terms of the best-so-far process. We close with a phased roadmap and concrete reporting requirements towards a reproducible NEC ecosystem.**

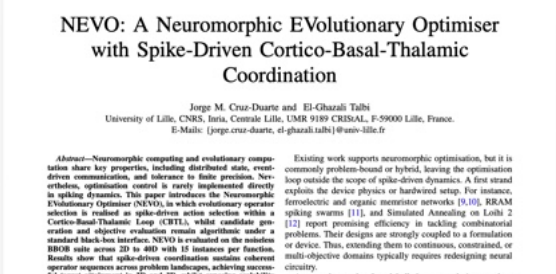
***Index Terms*—Neuromorphic Evolutionary Computation, Event-driven Optimisation, Neuromorphic Computing, Spiking Neural Networks, Operator Selection, Energy-Latency Benchmarking, Best-so-far Convergence**

constraints that extend beyond algorithmic design. EC provides effective search behaviour in domains where gradients are inaccessible, or objectives are irregular [2,3]. Still, their theoretical support remains limited, their performance deteriorates in high-dimensional spaces, and their execution incurs synchronisation costs associated with processors optimised for dense linear algebra.

The structural mismatch between EAs and conventional hardware motivates the study of alternative computational substrates. Neuromorphic Computing (NC) processors such as Intel Loihi [4], IBM TrueNorth [5], SpiNNaker [6], BrainScaleS [7], SynSense Xylo [8], and BrainChip Akida [9] integrate memory and computation, communicate via sparse asynchronous spikes, and operate in an event-driven regime rather than through dense synchronous updates [10]. These architectures achieve substantial energy efficiency. For instance, Loihi 2 operates within a power envelope of a few hundred milliwatts for networks of millions of neurons, far below the

cec_pap596

Fri 26, 17h20 · 0.11 Cape Town



NEVO: A Neuromorphic EVolutionary Optimiser with Spike-Driven Cortico-Basal-Thalamic Coordination

Jorge M. Cruz-Duarte and El-Ghazali Talbi
University of Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France.
E-Mails: {jorge.cruz-duarte, el-ghazali.talbi}@univ-lille.fr

***Abstract*—Neuromorphic computing and evolutionary computation share key properties, including distributed state, event-driven communication, and tolerance to finite precision. Nevertheless, optimisation control is rarely implemented directly in spiking dynamics. This paper introduces the Neuromorphic EVolutionary Optimiser (NEVO), in which evolutionary operator selection is realised as spike-driven action selection within a Cortico-Basal-Thalamic Loop (CBTL), whilst candidate generation and objective evaluation remain algorithmic under a standard black-box interface. NEVO is evaluated on the noiseless BBOB suite across 2D to 40D with 15 instances per function. Results show that spike-driven coordination sustains coherent operator sequences across problem landscapes, achieving successful target attainment in 2D and 3D, whilst exposing scalability limitations beyond 10D. These findings confirm that spike-driven coordination is feasible on neuromorphic substrates and clarify how neuromorphic action selection relates to existing adaptive operator selection strategies. The analysis also identifies concrete design requirements for future implementations.**

***Index Terms*—Neuromorphic Optimisation, Evolutionary Computation, Operator Selection, Spiking Neural Networks, Neural Engineering Framework, Nengo, Basal Ganglia.**

Existing work supports neuromorphic optimisation, but it is commonly problem-bound or hybrid, leaving the optimisation loop outside the scope of spike-driven dynamics. A first strand exploits the device physics or hardwired setup. For instance, ferroelectric and organic memristor networks [9,10], RRAM spiking swarms [11], and Simulated Annealing on Loihi 2 [12] report promising efficiency in tackling combinatorial problems. Their designs are strongly coupled to a formulation or device. Thus, extending them to continuous, constrained, or multi-objective domains typically requires redesigning neural circuitry.

A second strand, often labelled neuroevolution, evolves spiking models but retains a conventional search process. The Evolutionary Optimisation for NC Systems framework [13] and related surveys [14]–[16] show that EC algorithms can configure SNN architectures and learning rules. Nevertheless, NC hardware is primarily used for inference or evaluation. Moreover, swarm-inspired approaches provide spiking interpretations of established Metaheuristics (MHs) [17,18]. These

The convergence of Evolutionary Computation and Neuromorphic Computing is timely, technically plausible, and open for us, as a community, to shape.

Bedankt!
Thanks!
Merci!
Grazie!
¡Gracias!
Obrigado!
Mulţmesc!

NeurOptim

neuroptim.org

Contact information

Jorge M. Cruz-Duarte
jorge.cruz-duarte@univ-lille.fr

El-Ghazali Talbi
el-ghazali.talbi@univ-lille.fr

Supported by the France 2030 PIQ project ANR-24-RRII-0002