Today’s Session Is Being Recorded

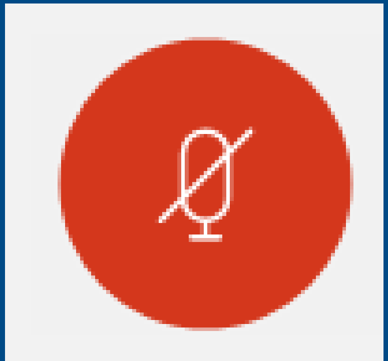
This session is being recorded and will be made available for viewing by others by invitation.

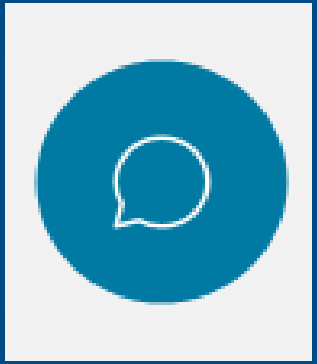
If you do not wish for your voice to be recorded, please place your phone on mute for the duration of the session.

This session may contain information that is deemed confidential under the terms of the INRC Participation Agreement and such information should be handled accordingly.

If you wish to ask a question or make a comment, please do not disclose any other information that is confidential to you, your employer, or any third party.

The views expressed in this session are those of the contributors and do not necessarily reflect the views of Intel Corporation.





RECORDING IN PROGRESS

Neuromorphic Solutions for Optimization Problems (II)

Future of Optimization from a Neuromorphic Perspective

February 11, 2021: Today's Panel

Udayan Ganguly (IIT Bombay) Johan Kwisthout (Radboud U)

Cengiz Pehlevan (Harvard U) Ojas Parekh (Sandia National Lab)

Gabriel Fonseca-Guerra (NCL) Prasad Joshi (NCL)

Slack channel: `#optimization`

Agenda

- A short introduction
- Individual thoughts (5 min each x 6 = approx. 30 min)
- Discussion as a panel (20 min)
- Q&A (10 min)

  - We can of course merge Panel discussion with Q&A and take Q's from chat as well as Slack.

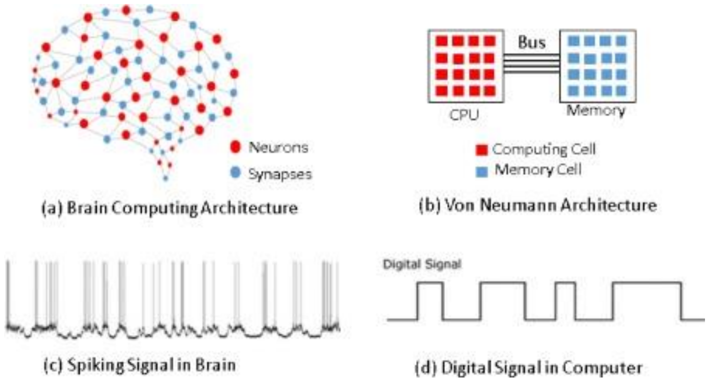
Individual thoughts

- Panelists approximately represent the following areas, but nothing is set in stone, obviously:

Johan Kwisthout	Genetic algorithms mapped to SNNs, Complexity theory for neuromorphic computing and its relation to mapping optimization problems (e.g. MIPs are NP-hard)
Cengiz Pehlevan	E-I balanced dynamics solving minimax problem
Prasad Joshi	Graph search on Loihi
Ojas Parekh	Dynamic programming mapped to SNNs, graph search in SNNs
Udayan Ganguly	Travelling salesman problem mapped to oscillator networks, hardware implementation of Boltzmann machine
Gabriel Fonseca-Guerra	Constraint Satisfaction Problem

Johan Kwisthout

Brain Inspired Computing research



- ***Neuromorphic architectures***
  novel brain-inspired hardware
  → new computing platform
  but also new **paradigm**

- Traditional computer architectures are **well-understood**:
  We know what we can do with limited resources and what not

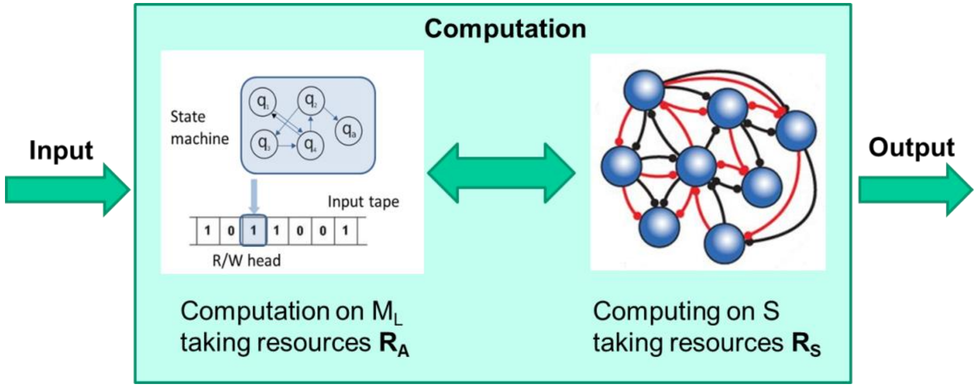
- Neuromorphic systems still **lack** such understanding

- We contribute ***theory of computing*** and ***algorithm design***
  both in abstract computation models and on the Loihi

- Some results on complexity theory and optimization algorithms

Neuromorphic Complexity Theory

- Computational model: spiking neural network



- Optimization problem: Input (e.g. CSP) → pre-processing leading to network configuration → computation
- Hybrid algorithm: assume neuromorphic co-processor that can be queried by regular CPU for specific tasks
- First formal results: NICE 2020 (to be presented this year)
- **No free lunch**! Neuromorphic architectures can speed up and save energy but not solve NP-hard problems in poly time

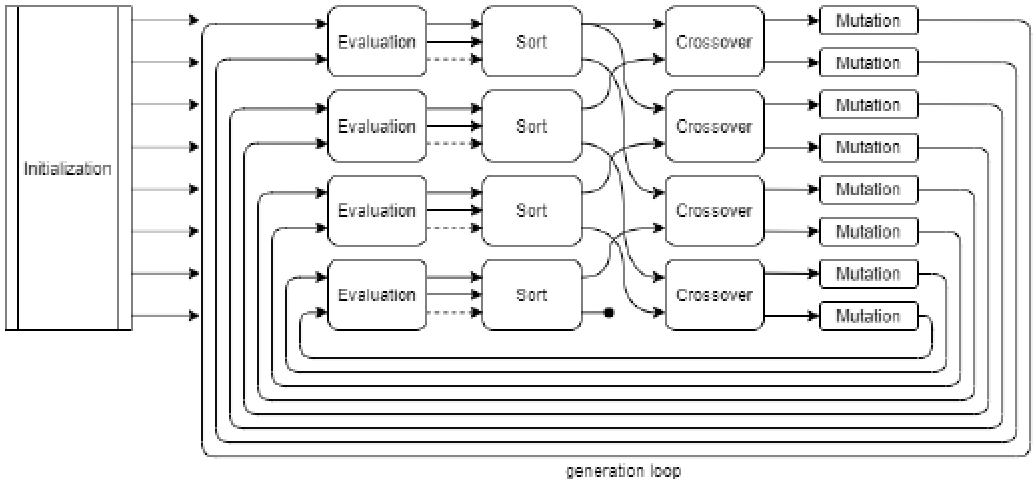
https://dl.acm.org/doi/abs/10.1145/3381755.3381760

Hybrid and SNN algorithms

- Hybrid algorithm for Max Network Flow – energy saving http://arxiv.org/abs/1911.13097 (uses Loihi for shortest paths)

- SNN implementation of genetic algorithms
  - Early work (student term project neuromorphic course)
- Proof-of-concept (one-max function)
- Approach not uncommon to Chris Yakopcic's SAT work
- Iteratively generating solutions, crossover, mutation
- Micro-circuits for sorting etc.
- Some results:

		n_chromosomes		len_chromosomes	
		sequential	parallel	sequential	parallel
space	neurons	$O(n)$	$O(n)$	$O(1)$	$O(n)$
	connections	$O(n)$	$O(n)$	$O(1)$	$O(n^2)$
time		$O(1)$	$O(1)$	$O(n)$	$O(1)$
energy		$O(n)$	$O(n)$	$O(n)$	$O(n)$



Cengiz Pehlevan

Minimax Dynamics of Optimally Balanced Spiking Networks of Excitatory and Inhibitory Neurons

Qianyi Li
Biophysics Graduate Program
Harvard University
Cambridge, MA 02138
qianyi_li@g.harvard.edu

Cengiz Pehlevan
John A. Paulson School of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
cpehlevan@seas.harvard.edu

NeurIPS, 2020

$$\min_{\mathbf{r}^E \geq 0} \max_{\mathbf{r}^I \geq 0} S(\mathbf{r}^E, \mathbf{r}^I), \quad S = -\frac{1}{2}\mathbf{r}^{E\top}\mathbf{W}^{EE}\mathbf{r}^E + \mathbf{r}^{E\top}\mathbf{W}^{EI}\mathbf{r}^I - \frac{1}{2}\mathbf{r}^{I\top}\mathbf{W}^{II}\mathbf{r}^I - \mathbf{x}^\top\mathbf{F}^\top\mathbf{r}^E$$

Leaky Integrate-and-Fire Dynamics:

$$\frac{dV_i^E}{dt} = -\frac{V_i^E}{\tau_E} + \sum_j W_{ij}^{EE} s_j^E - \sum_j W_{ij}^{EI} s_j^I + \sum_j F_{ij} s_j^0,$$

$$\frac{dV_i^I}{dt} = -\frac{V_i^I}{\tau_I} + \sum_j W_{ij}^{IE} s_j^E - \sum_j W_{ij}^{II} s_j^I \qquad s_j^{E(I)}(t) = \sum_k \delta(t - t_{j,k})$$

Instantaneous rate:

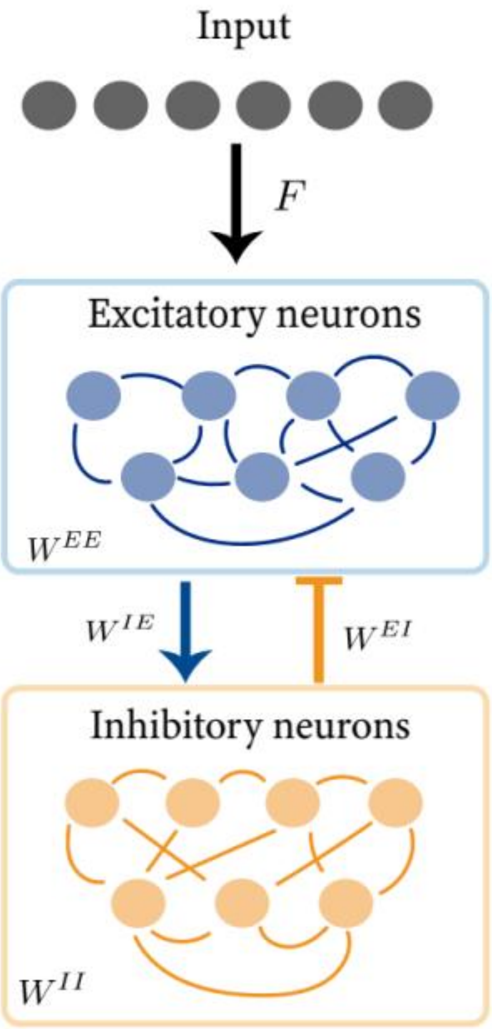
$$r_j^{E,I}(t) = \int_0^\infty e^{-(t-t')/\tau_{E,I}} s_j^{E,I}(t-t')dt'$$

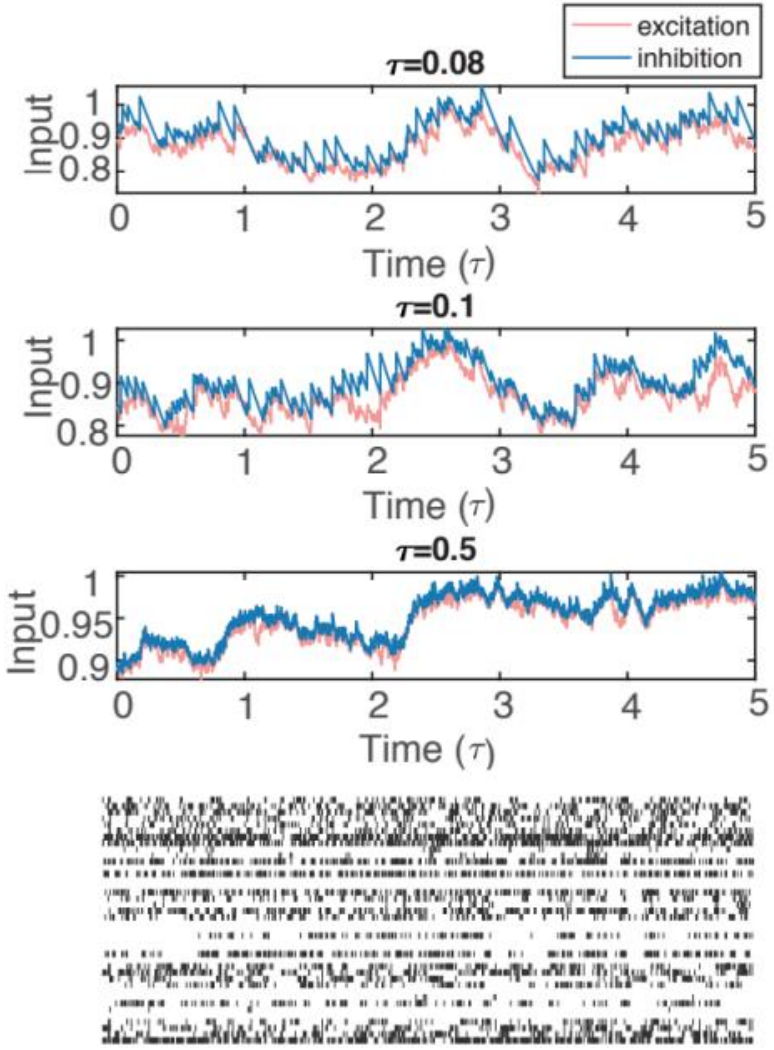
Greedy spiking:

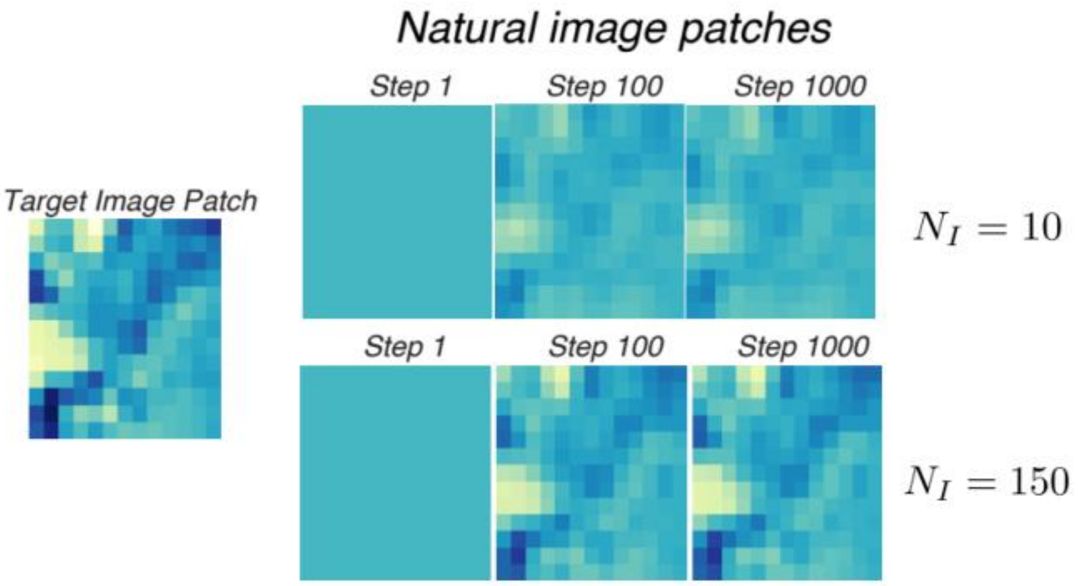
E: S(spike) < S(no spike)

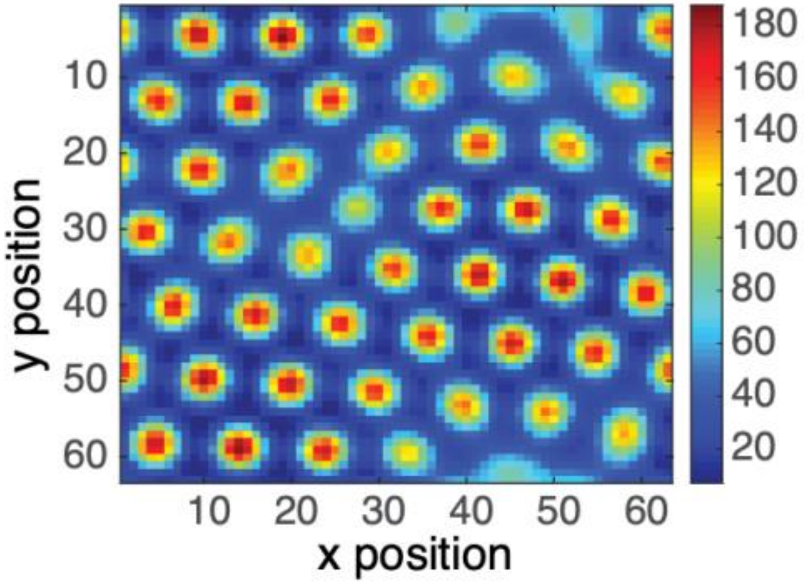
I: S(spike) > S(no spike)

E-I balance arises from saddle points

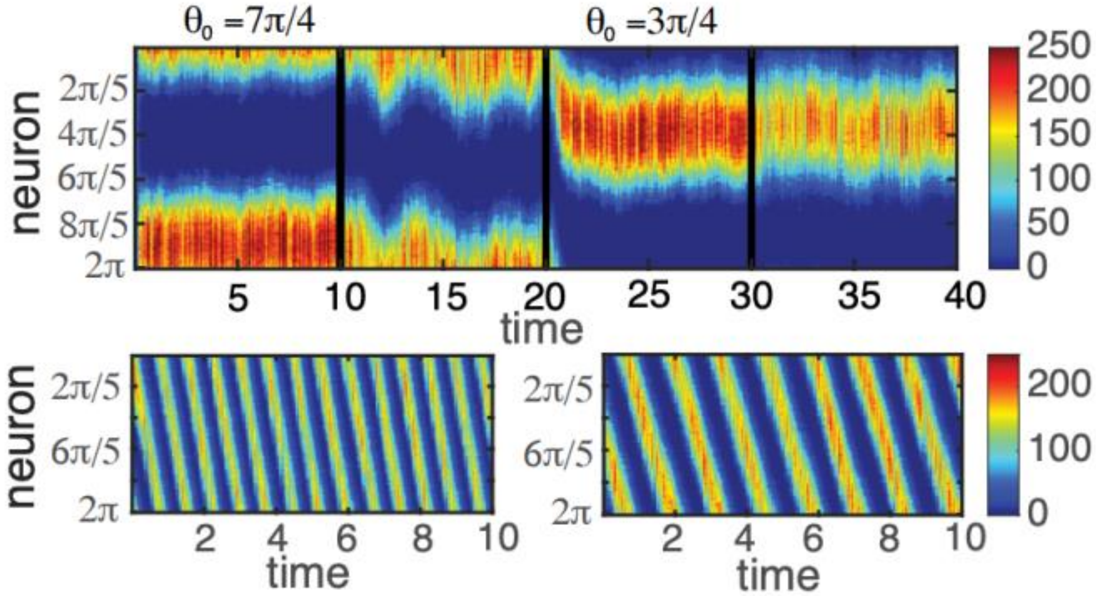


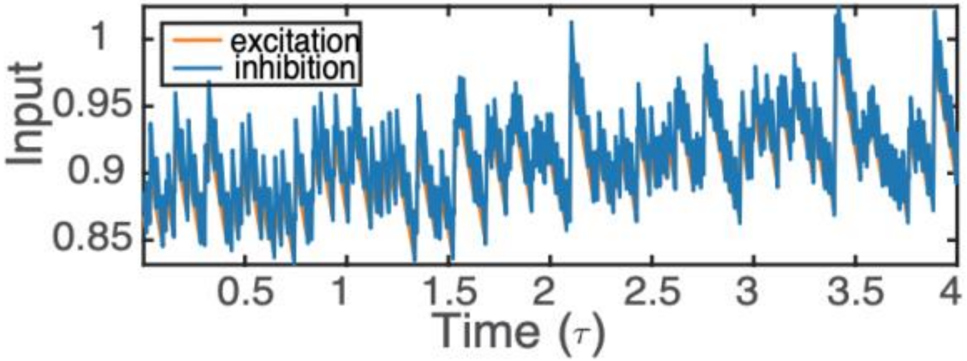



Natural image patches
Target Image Patch
Step 1
Step 100
Step 1000
$N_I = 10$
Step 1
Step 100
Step 1000
$N_I = 150$


y position
x position
10
20
30
40
50
60
180
160
140
120
100
80
60
40
20

-


$\theta_0 = 7\pi/4$
$\theta_0 = 3\pi/4$
neuron
$2\pi/5$
$4\pi/5$
$6\pi/5$
$8\pi/5$
2π
5 10 15 20 25 30 35 40
time
250
200
150
100
50
0
neuron
$2\pi/5$
$6\pi/5$
2π
2 4 6 8 10
time
$2\pi/5$
$6\pi/5$
2π
2 4 6 8 10
time
200
100
0


excitation
inhibition
Input
1
0.95
0.9
0.85
0.5 1 1.5 2 2.5 3 3.5 4
Time (τ)

Prasad Joshi

Searching graphs with spikes

Runtime comparison to best Dijkstra optimizations:

- Neuromorphic: O($L \cdot \sqrt{V}$)
- Standard: O(E)

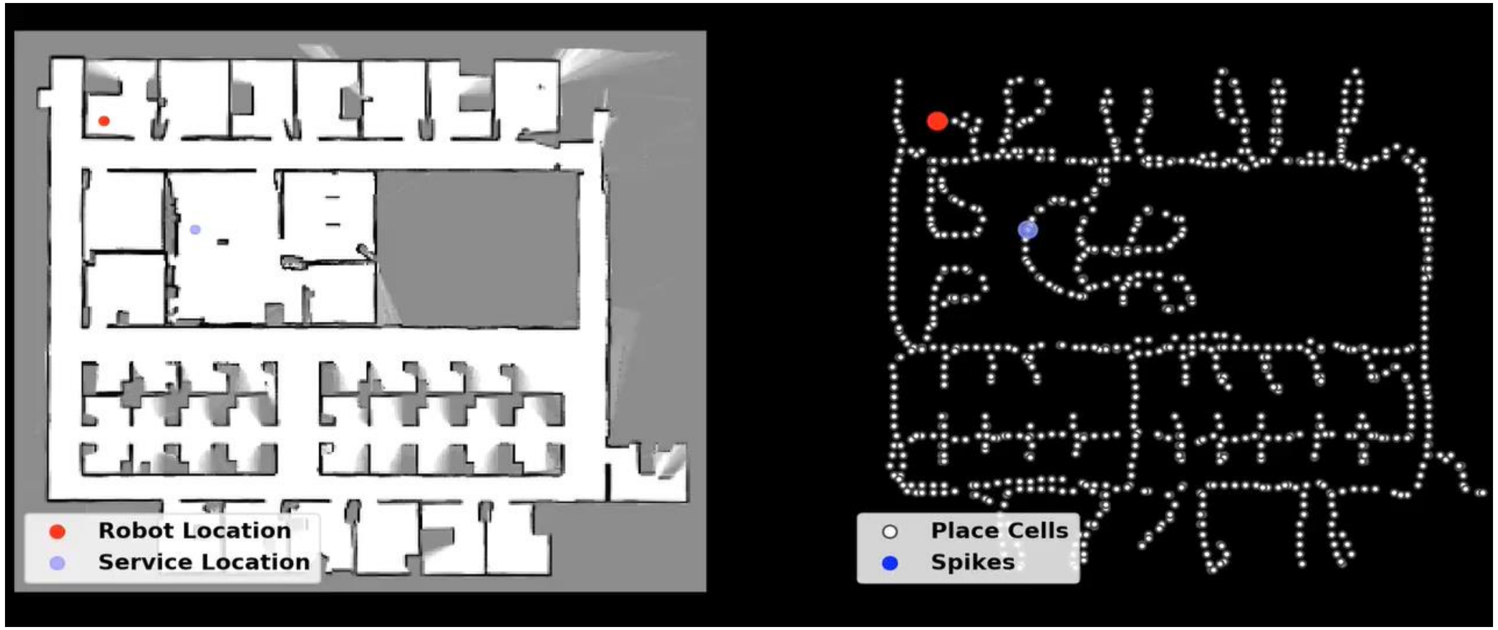
For most nontrivial problems:

- L<<E
- V<<E

Neuromorphic solution uses *fine-grain parallelism* an *temporal wavefront-driven computation* to potentially provide great performance gains for large problems.

Based on *Ponulak F., Hopfield J.J. Rapid, parallel path planning by propagating wavefronts of spiking neural activity. Front. Comput. Neurosci. 2013. V. 7. Article № e98.*

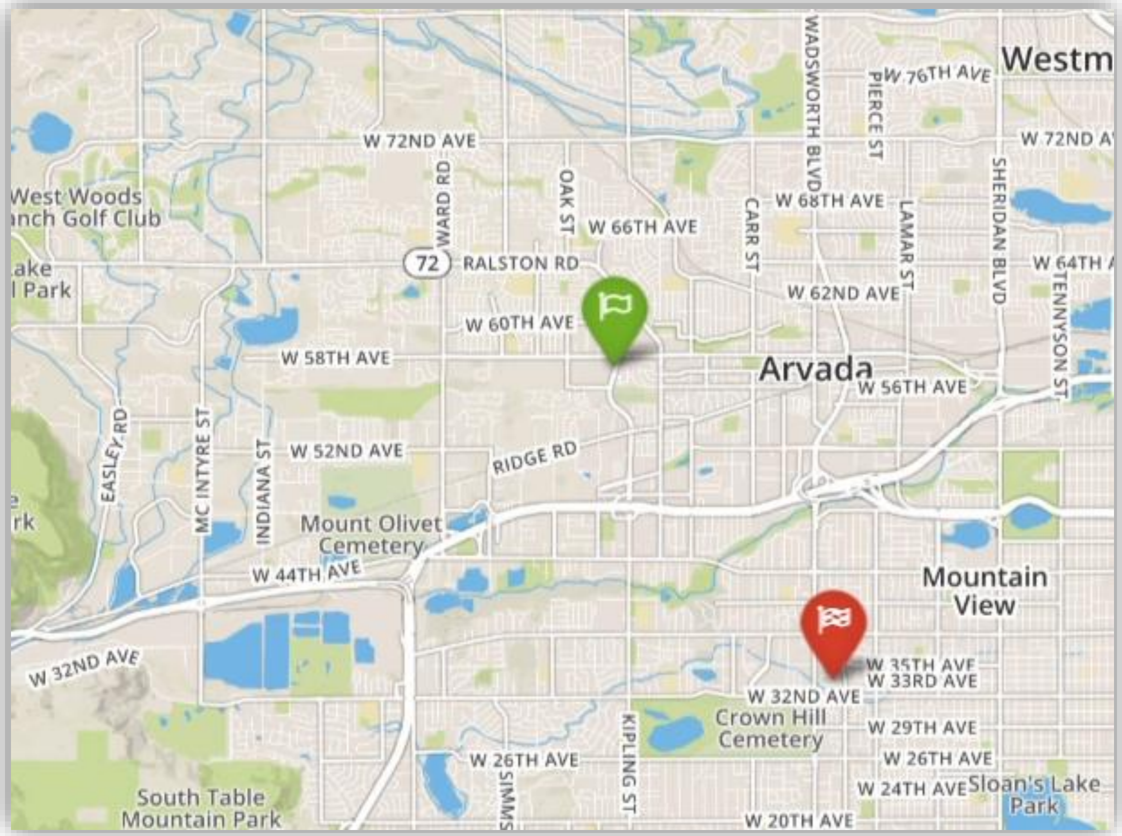
ROBOT MOTION | LOIHI REPRESENTATION



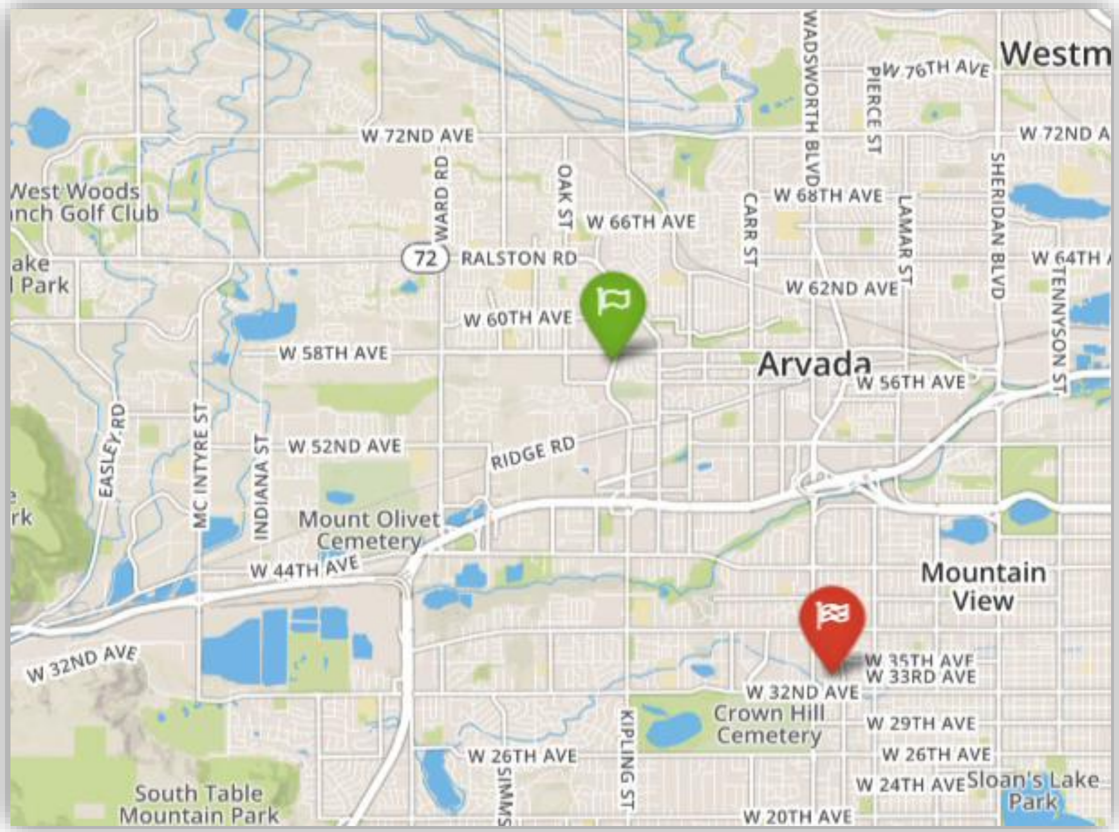
DARPA SDR Site B
(Data from Radish Robotics Dataset)

Using Loihi for Driving Directions in Colorado

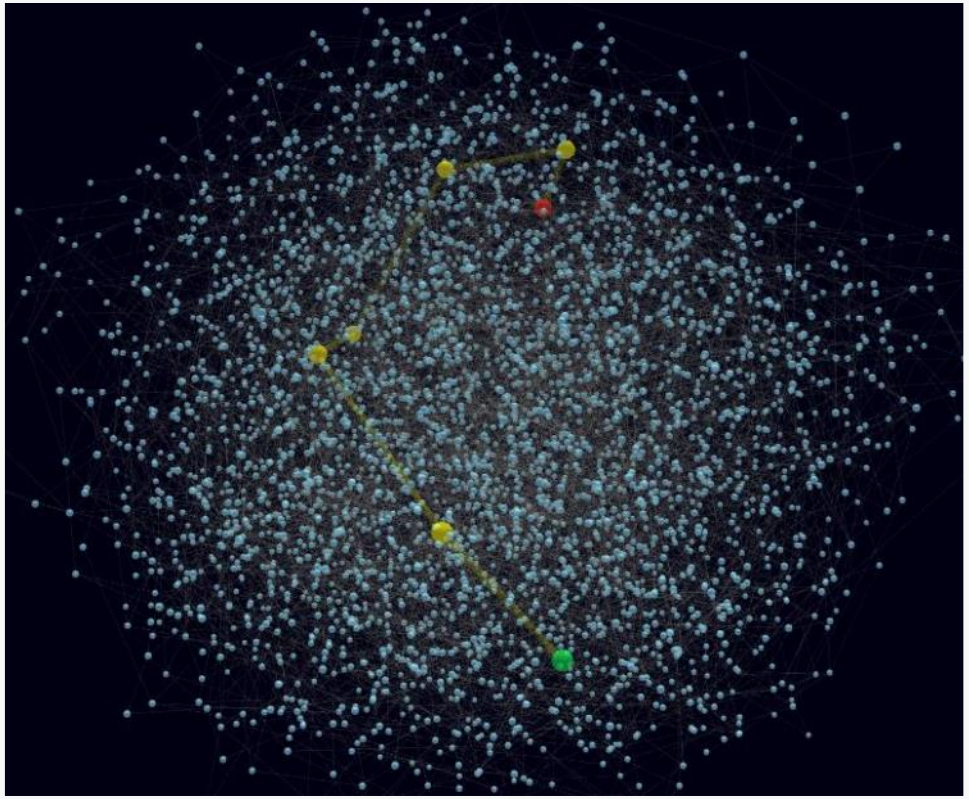
Loihi: Fine-Grain Parallel Search



Dijkstra: Sequential Breadth-First Search

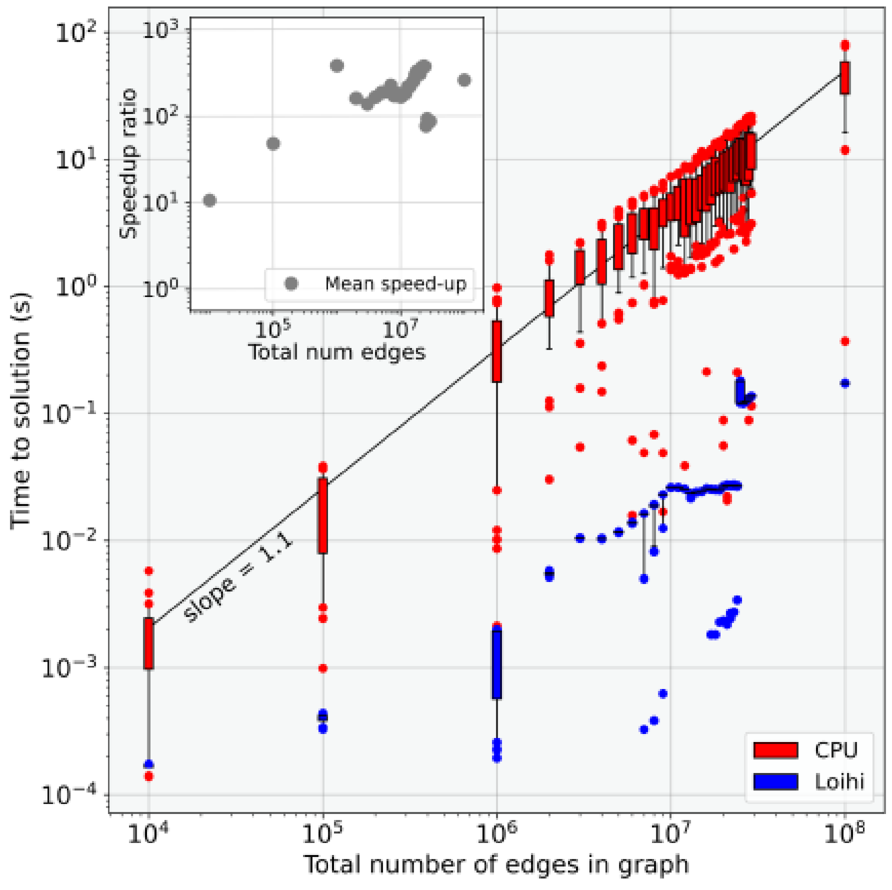


Searching Small World Graphs with Loihi



Small world networks up to 1M vertices mapped to Loihi
Spikes traverse graph and identify shortest path *in time*
(versus CPU search with optimized Dijsktra's Algorithm)

Loihi search has >100x latency advantage versus a CPU (Xeon Gold 6136)



From upcoming Proceedings of the IEEE publication; preprint available

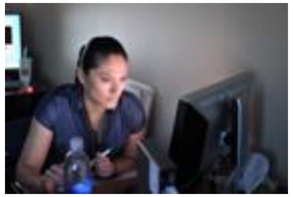
References and System Test Configuration Details

Loihi graph search algorithm based on *Ponulak F., Hopfield J.J. Rapid, parallel path planning by propagating wavefronts of spiking neural activity. Front. Comput. Neurosci. 2013.* Loihi: Nahuku and Pohoiki Springs systems running NxSDK 0.97. CPU: Intel Xeon Gold with 384GB RAM, running SLES11, evaluated with Python 3.6.3, NetworkX library augmented with an optimized graph search implementation based on Dial's algorithm. See also http://rpg.ifi.uzh.ch/docs/CVPR19workshop/CVPRW19_Mike_Davies.pdf

Ojas Parekh

Sandia National Laboratories

Provable neuromorphic advantages for graph algorithms



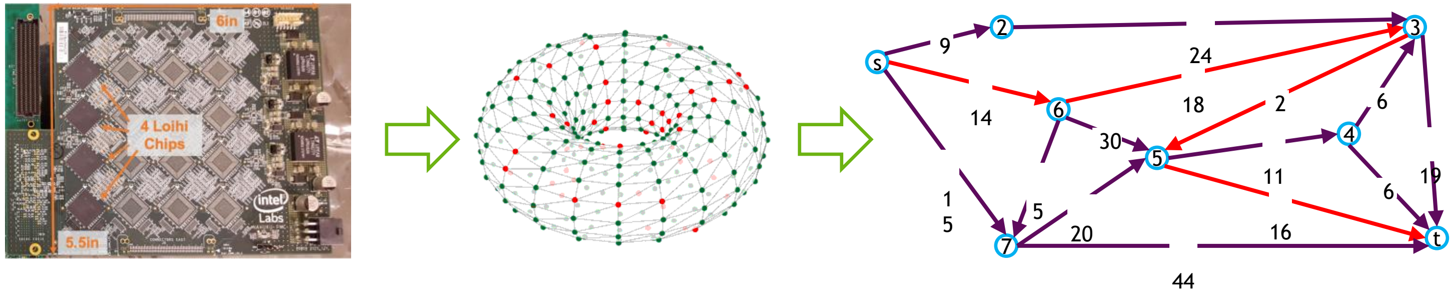
Presented by

Ojas Parekh

with James B. Aimone, Yang Ho, Cynthia Phillips, Ali Pinar, William Severa, and Yipu Wang

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Neuromorphic Graph Algorithms

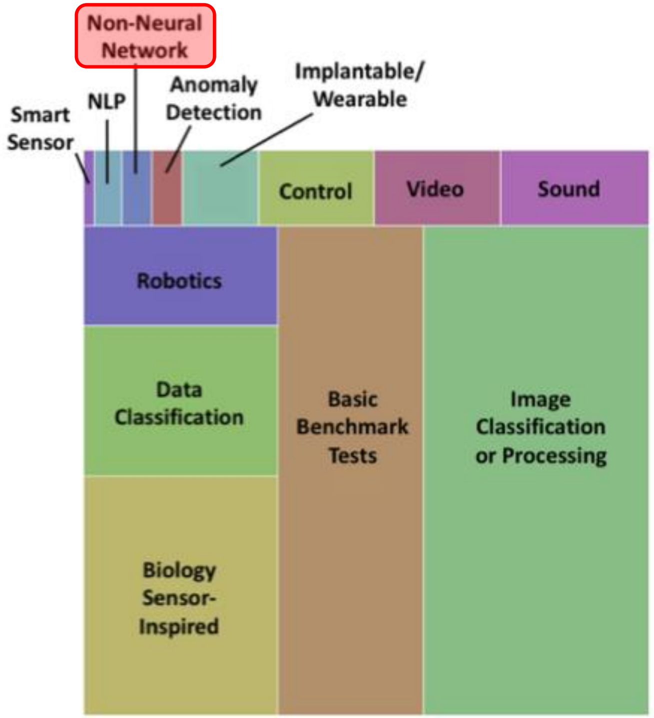


- 2017 survey by Schuman et al. of neuromorphic computing covering 2500+ references had only 8 citations of graph applications (see figure)

- Most of above graph applications have a learning-oriented component (Hopfield networks or Boltzmann machines)

- Recent interest in spike-based graph algorithm papers (e.g., [arXiv: 1902.10369, 1903.10574, 1911.13097, 2001.08439, 2010.01423 https://doi.org/10.1145/3354265.3354285])

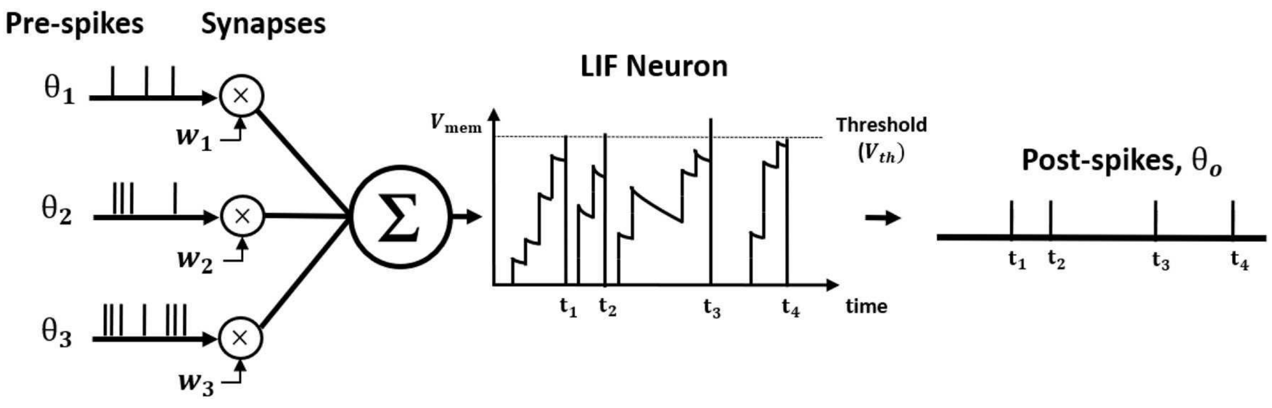
- None of these works demonstrate an asymptotic neuromorphic advantage over conventional computing



Landscape of current neuromorphic applications based on 2500+ references
[Schuman et al., https://arxiv.org/abs/1705.06963, 2017]

Shortest Paths, Neuromorphically

Networks of spiking neurons elegantly implement Dijkstra's algorithm

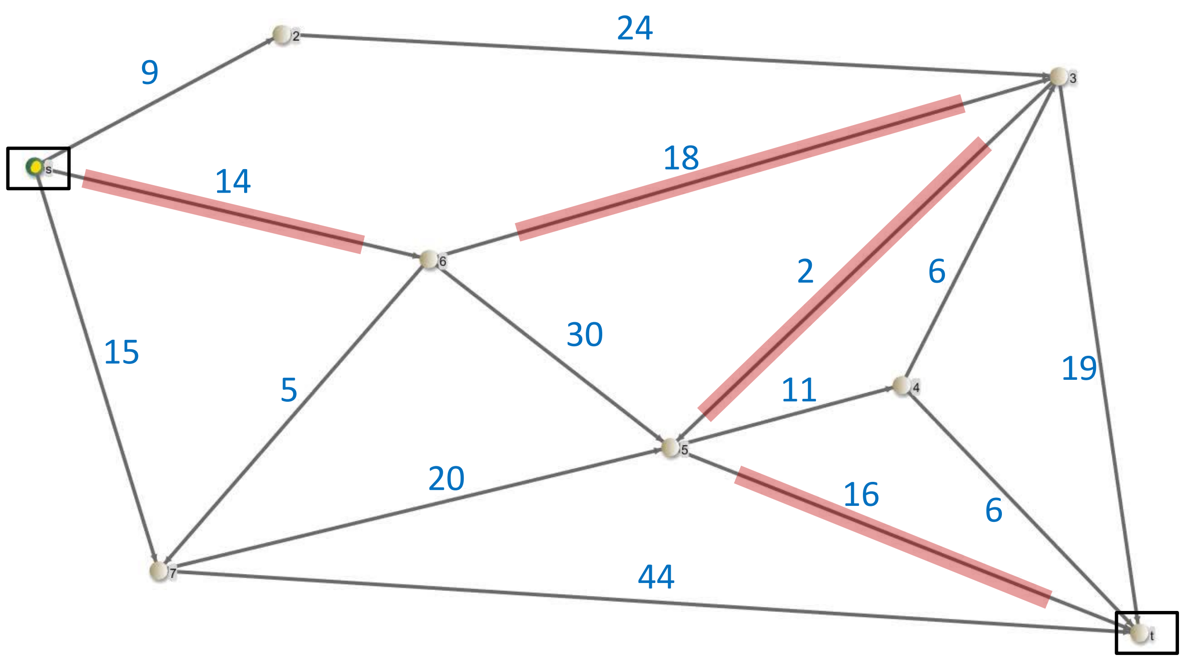


Leaky integrate and fire (LIF) neuron
Image from [Lee et al., https://doi.org/10.3389/fnins.2020.00119]

- Application to shortest paths: program all neurons to propagate any incoming spikes, with delays on synapses proportional to edge weights
- Initiate spike at node s, and terminate when node t first fires
- Although elegant, this is a pseudopolynomial-time, whose run time depends linearly on the edge weights
- We design polynomial-time algorithms for k-hop shortest paths

Spiking shortest paths algorithm

[Aibara et al., IEEE Int. Symp. on Circuits and Systems, 1991]

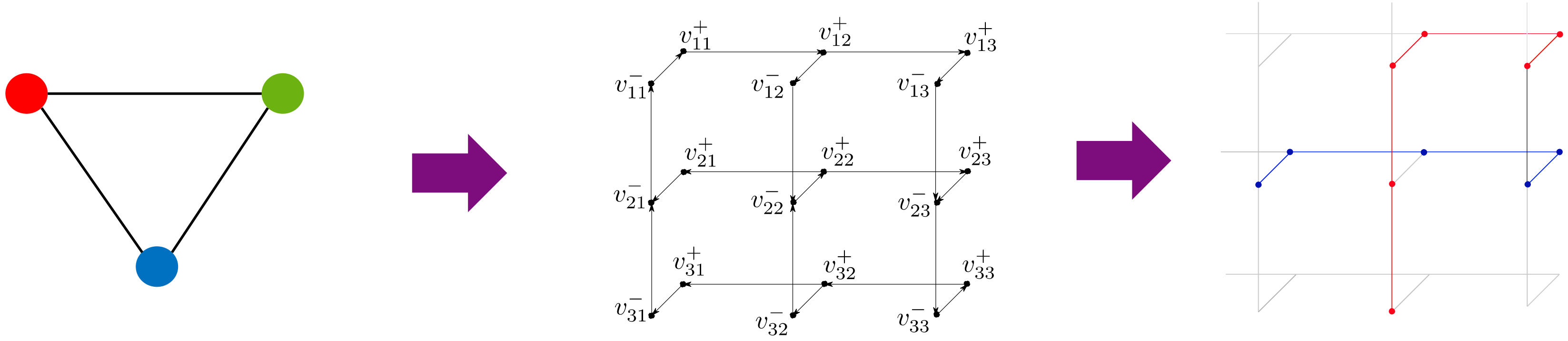


First Asymptotic Neuromorphic Advantages (for Shortest Paths)

n – number of nodes in graph
m – number of edges in graph
k – weighted shortest path with at most k edges
poly-log factors ignored in table

Algorithm type	k-hop single source shortest paths
Conventional (Floyd-Warshall)	$\tilde{O}(km)$
Neuromorphic implementation	$\tilde{O}(k)$

Assumes problem graph may be embedded without dilation in neuromorphic hardware



We also take embedding/data-movement costs into account by only assuming a simple 2d-grid-like "crossbar" architecture.

On conventional side, we introduce a geometric data-movement model.

Algorithm type	k-hop single source shortest paths
Conventional (Floyd-Warshall)	$\tilde{O}(km^{1.5})$
Neuromorphic implementation	$\tilde{O}(km)$

Data-movement aware running times.

Any conventional algorithm needs $\Omega(m^{1.5})$ to read input in our model.

More detailed presentation of results appeared in SPAA 2020 [https://doi.org/10.1145/3350755.3400258]

Dynamic Programming

Dynamic programming is a *general technique* for solving certain kinds of discrete optimization problems

- Recurrent solutions to lattice models for protein-DNA binding
- Backward induction as a solution method for finite-horizon discrete-time dynamic optimization problems
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite-horizon, discrete-time, discounted, time-invariant dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithmic problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Cocke–Younger–Kasami (CYK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth's word wrapping algorithm that minimizes raggedness when word wrapping text
- The use of transposition tables and refutation tables in computer chess
- The Viterbi algorithm (used for hidden Markov models, and particularly in part of speech tagging)
- The Earley algorithm (a type of chart parser)
- The Needleman–Wunsch algorithm and other algorithms used in bioinformatics, including sequence alignment, structural alignment, RNA structure prediction
- Floyd's all-pairs shortest path algorithm
- Optimizing the order for chain matrix multiplication
- Pseudo-polynomial time algorithms for the subset sum, knapsack and partition problems
- The dynamic time warping algorithm for computing the global distance between two time series
- The Selinger (a.k.a. System R) algorithm for relational database query optimization
- De Boor algorithm for evaluating B-spline curves
- Duckworth–Lewis method for resolving the problem when games of cricket are interrupted
- The value iteration method for solving Markov decision processes
- Some graphic image edge following selection methods such as the "magnet" selection tool in Photoshop
- Some methods for solving interval scheduling problems
- Some methods for solving the travelling salesman problem, either exactly (in exponential time) or approximately (e.g. via the bitonic tour)
- Recursive least squares method
- Beat tracking in music information retrieval
- Adaptive-critic training strategy for artificial neural networks
- Stereo algorithms for solving the correspondence problem used in stereo vision
- Seam carving (content-aware image resizing)
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the linear search problem
- Kadane's algorithm for the maximum subarray problem
- Optimization of electric generation expansion plans in the Wein Automatic System Planning (WASP) package

Wikipedia: 30 applications across diverse domains
[https://en.wikipedia.org/wiki/Dynamic_programming]

Another list with 50 applications
[https://blog.usejournal.com/top-50-dynamic-programming-practice-problems-4208fed71aa3]

Neuromorphic Dynamic Programming

New neuromorphic algorithms for dynamic programming

Spike times encode dynamic programming table values

Example: Dynamic Program for Knapsack Problem

k \ W	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
2	0	1	2	3	3	3	3	3	3	3	3
3	0	1	2	4	5	6	7	7	7	7	7
4	0	1	2	4	5	6	7	8	9	9	9
5	0	1	2	4	5	6	7	8	9	10	11

Each table entry is value of best knapsack solution of weight at most W using items {1,…,k}

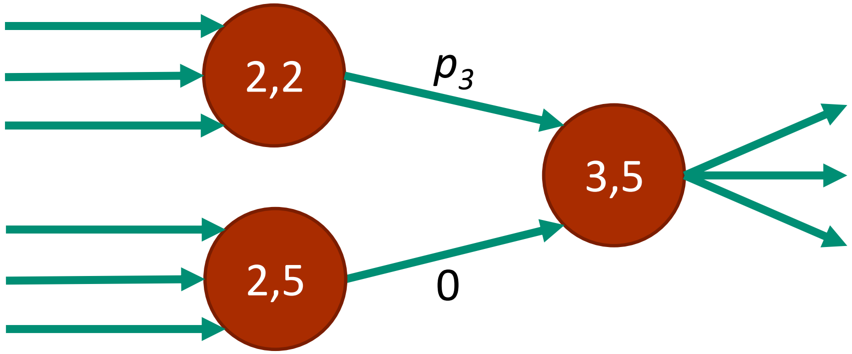
[ICONS 2019, https://doi.org/10.1145/3354265.3354285]

Knapsack Problem:

N items, each with weight w_i and value v_i

Goal: pick subset of items of weight at most W, maximizing total value.

$$T[3,5] = max\{\overbrace{T[2,5-w_3]+p_3}^{=6}, \overbrace{T[2,5]}^{=3}\}$$



Spiking approach: T[i,j] encoded as time neuron (i,j) receives incoming spike on last of its incoming links

Udayan Ganguly

intel labs

n-oscillator for n-city Traveling Salesman Problem

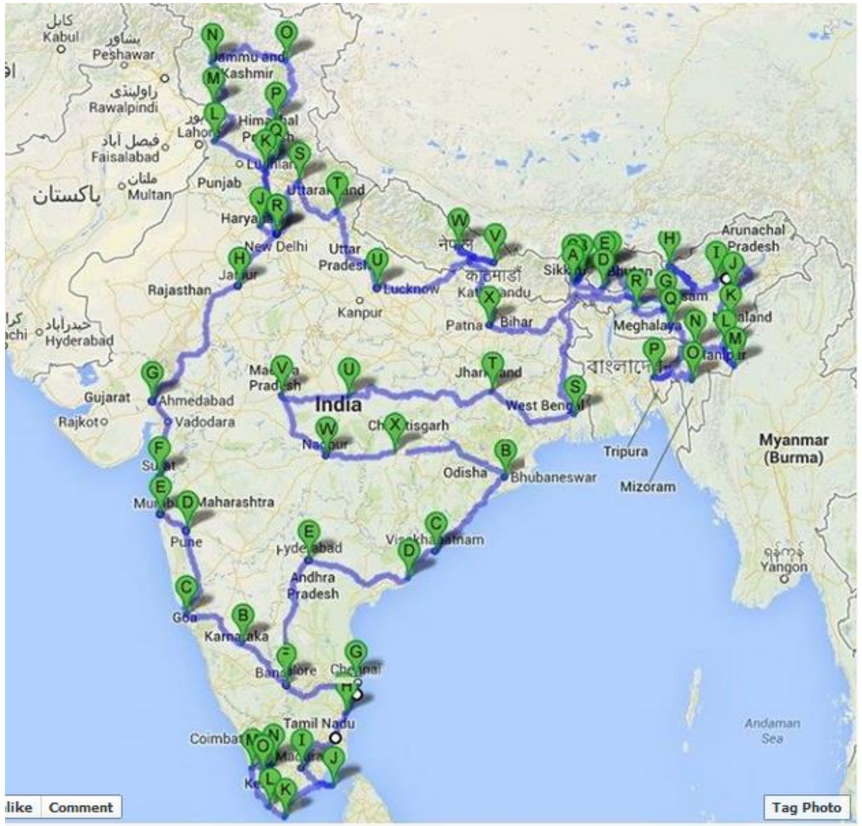
Udayan Ganguly
IIT Bombay

INRC Panel

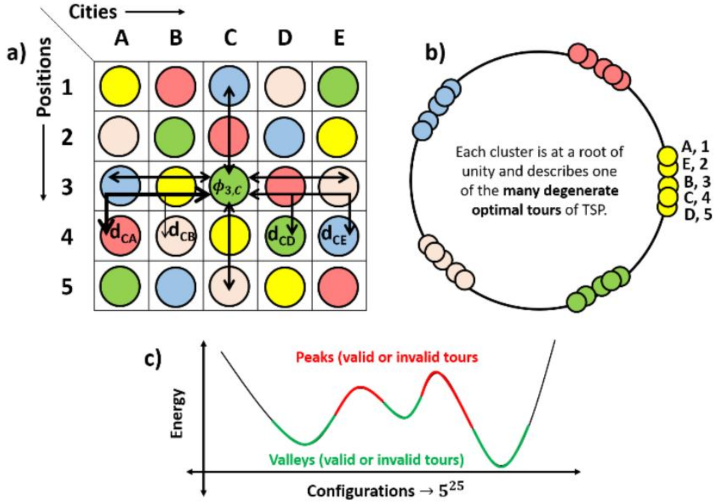
Feb 11, 2021

Oscillator Neural Network (ONN) Traveling Salesman Problem

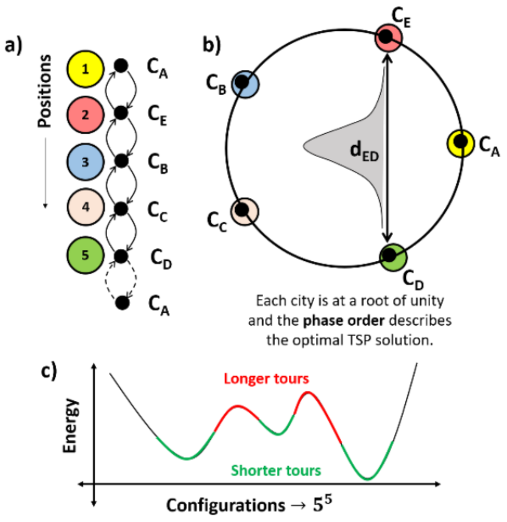
Stochastic Gradient Descent Neural Networks + Oscillator Phases Mapped Cost Functions ⇒ Efficient Solutions to TSP Optimization



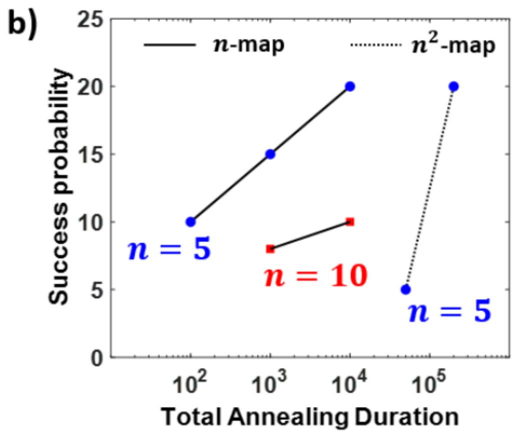
TSP is a very difficult NP Hard problem



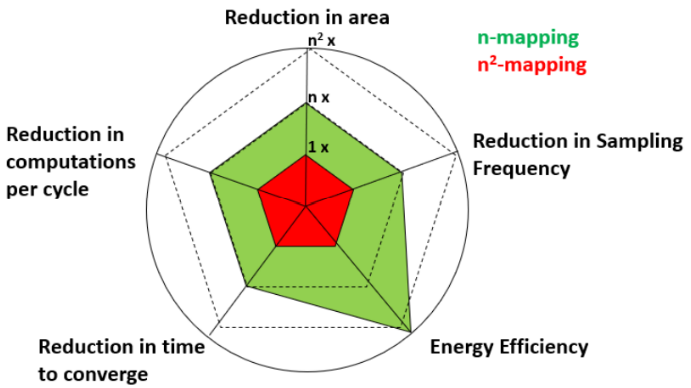
n^2 Oscillator Phase clustering determines tour; High degeneracy



n-Oscillator Phase determines tour; No degeneracy



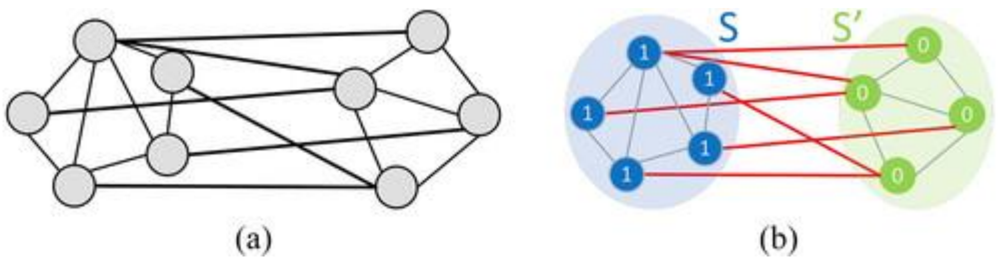
Performance



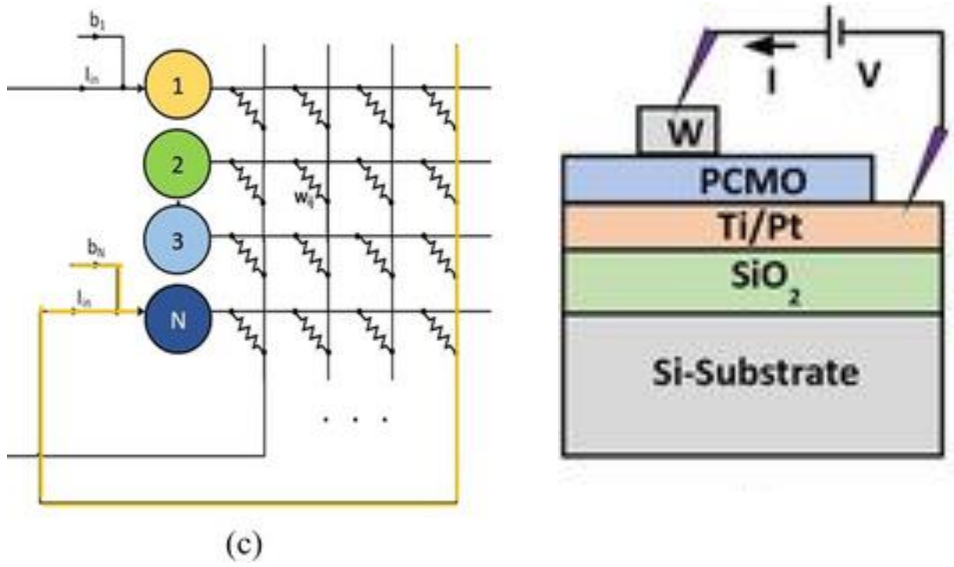
Comparison

A) Neuromorphic Hardware should be able to solve such problems approximately but efficiently!

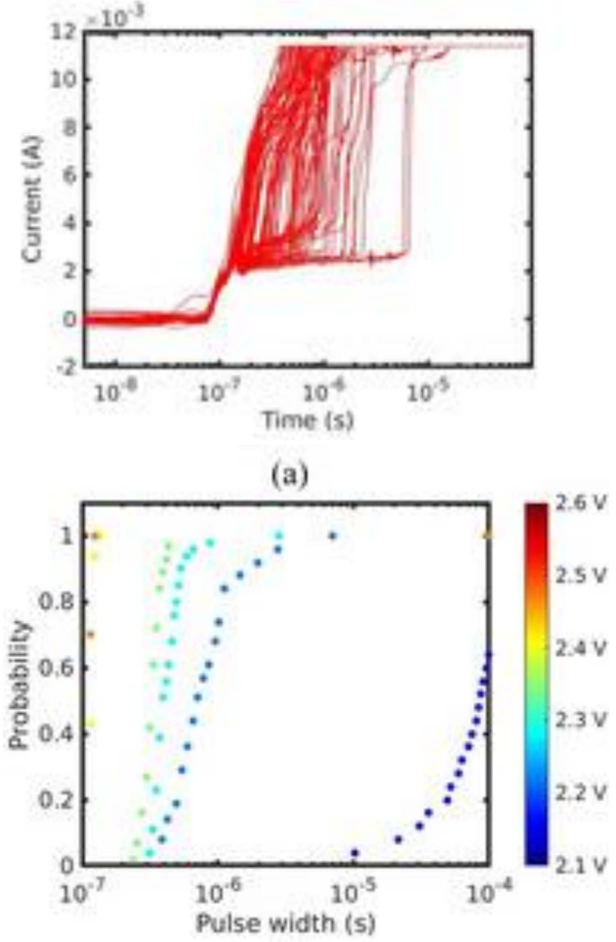
Boltzmann Machine with Stochastic Nanoscale RRAM for Max Cut Problem



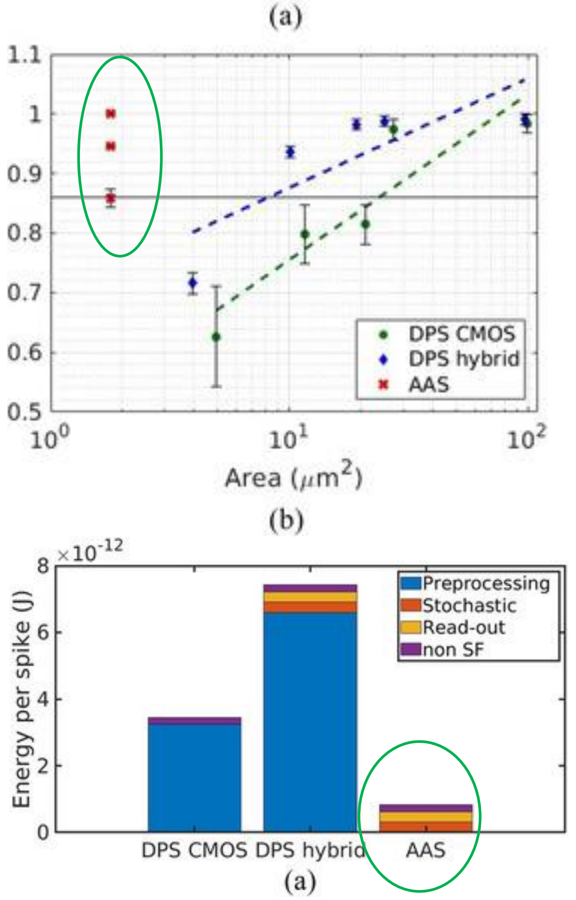
1. Max Cut problem is mapped to Boltzmann Machine



2. Boltzmann Machine is mapped to Cross Bar + RRAM



3. RRAM: Stochastic Switching Approx. Analog Sigmoid (AAS)



4. AAS has better Power Performance Area (PPA) trade-off compared to Digital Precision Sigmoid (DPS)

B) Neuromorphic Hardware based on nanoscale devices may enable PPA improvement!

Key Enablers for Neuromorphic Solvers

- Parallel communication in network
- Specific Network Structure/Design
- Stochasticity/Noise Mediated
- Others …(?)

Gabriel Fonseca-Guerra

CSPs and Their Encoding with SNNs

Problem definition:

Given:

Variable subsets

Value restrictions

$$CSP = \{X_i, D_i, \{\mathrm{Y_j}, \mathrm{R_j}\}\}$$

Variables

Constraints C_j

Value domains

Find assignment to X_i out of D_i that satisfies all constraints.

Optimization formulation in binary domain:

Minimize:

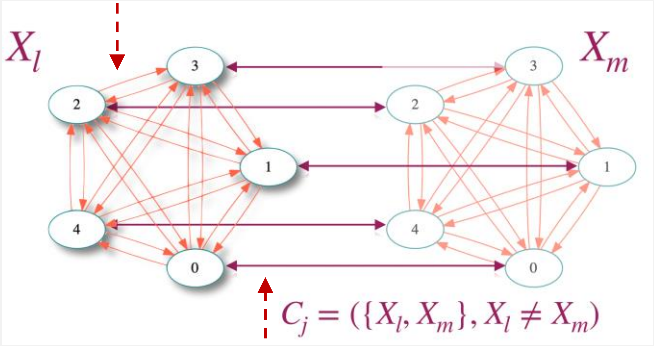
$$E = S^T \cdot W \cdot S \;\mid\; W \in \{-1,0,1\}^{N\times N}$$

Subject to:

$$S_{i\cdot|D_i|+k} = S'_{ik}, \qquad S'_{ik} \in \{0,1\}, \qquad \sum_k S'_{ik} = 1,$$

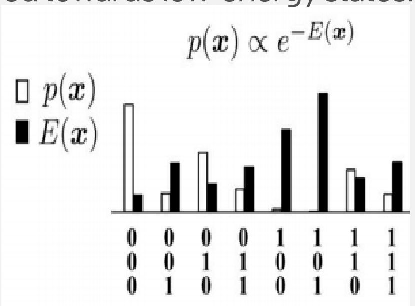
Encoding

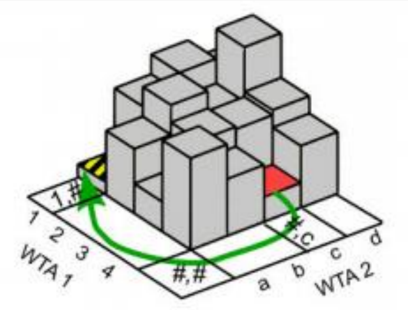
Variables represented by Winner-Take-All (WTA) circuits



Interconnectivity between WTAs
encode Constraints

[1]Minimization ⇔ Sampling from probability distribution $p(x)$ biased towards low-energy states:

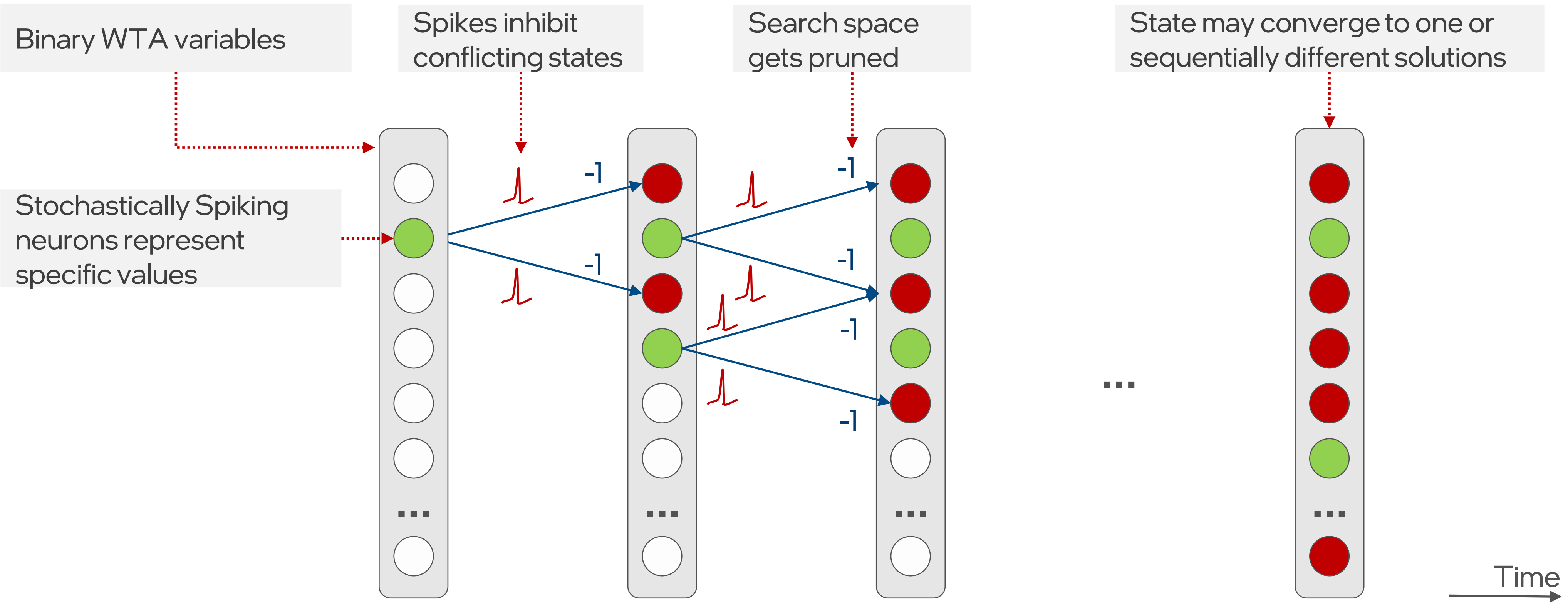




[1]Stochastic search via SNN enables faster convergence than pure gradient dynamics.

[1] Adapted from: Jonke Z, et. al (2016) Solving Constraint Satisfaction Problems with Networks of Spiking Neurons. Front. Neurosci. 10:118. doi: 10.3389/fnins.2016.00118

How SNNs Solve the Problems



Previous approaches rely on costly sampling of complete high-dimensional system at every (other) step:

- Binas et al., (2016) Spiking Analog VLSI Neuron Assemblies as CSP solvers, ISCAS, 2094-2097.
- Fonseca et al., (2017) Using stochastic spiking neural networks on SpiNNaker to solve constraint satisfaction problems, Front. Neurosci. 11:714.

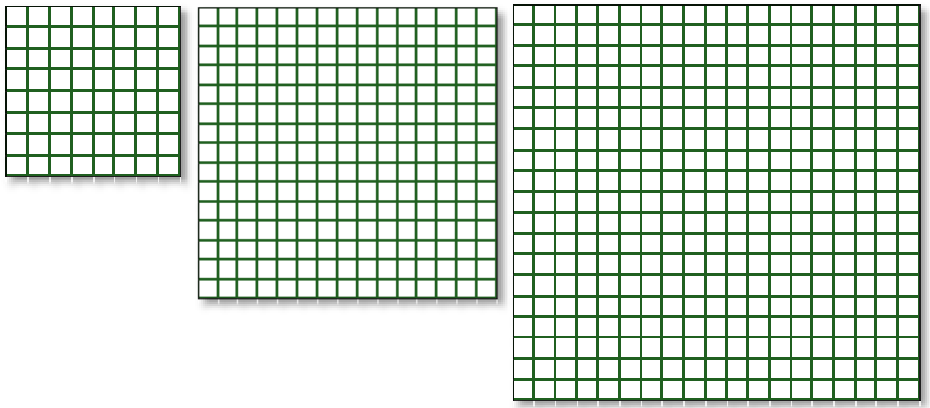
Latin Squares Benchmarking and Scaling

Task description

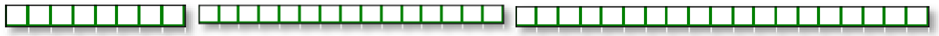
- A Latin square consists of an N × N array of variables each of which can take on N possible values such that no number is repeated in a row or column of the grid.
- Benchmark against the open-source state-of-the-art solver.

Scaling up

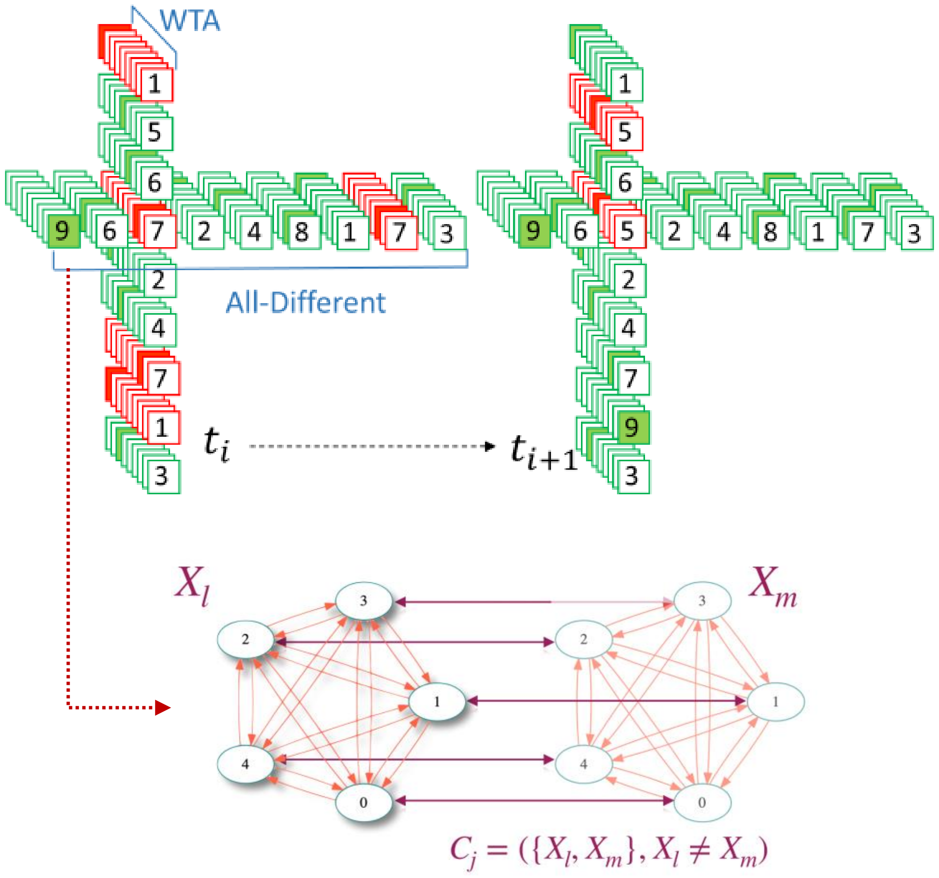
Variables: 81 256 400



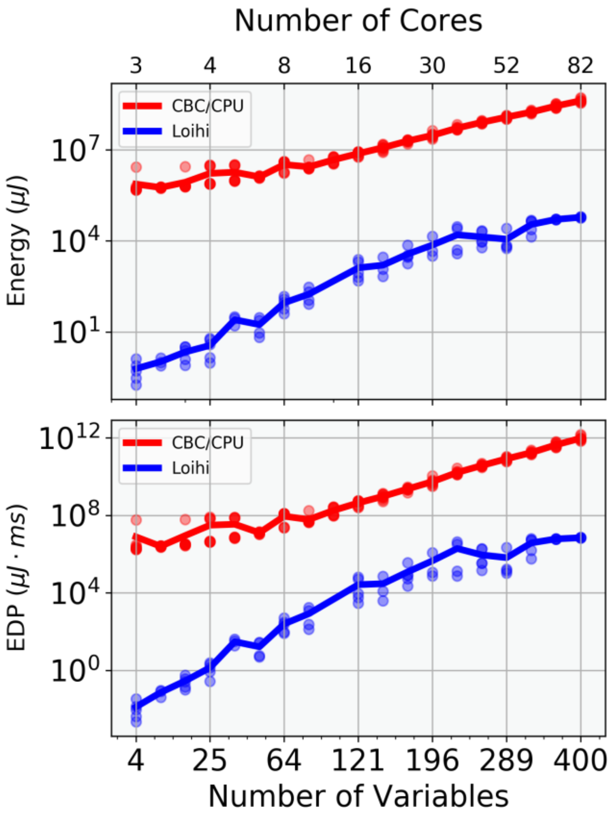
Domain size: 9 16 20

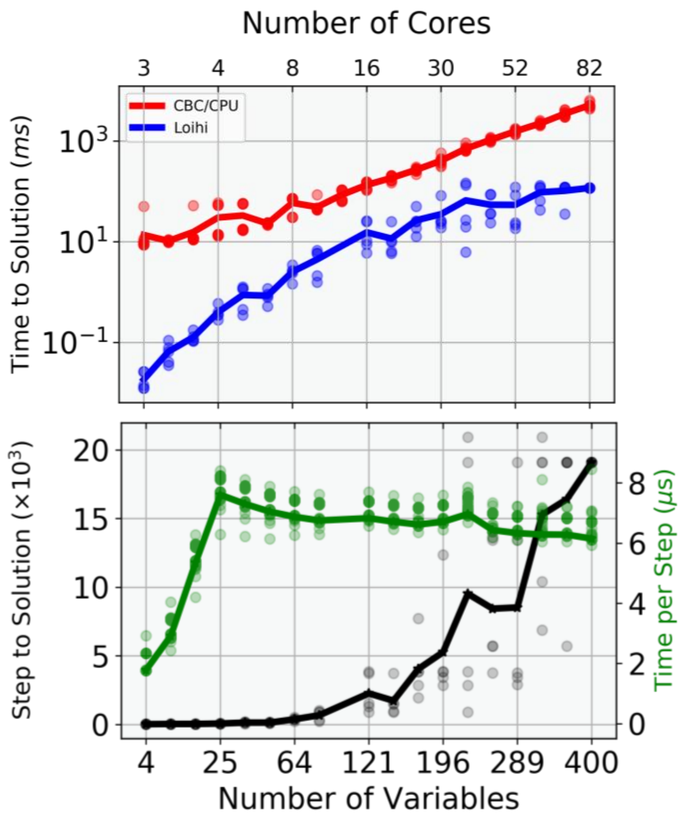


Encoding



Latin Squares Benchmarking and Scaling





Scaling:

Latin Square size is scaled up. Bigger problem implies:

- more neurons, cores and synapses.
- increased difficulty by exponential growth of the state space

Take away:

Compared with the state-of-the-art CBC solver[1], Loihi:

- is up to 44 times faster
- has 3-5 orders of magnitude lower EDP
- solves Latin Squares in the range 2x2 to 20x20
- can find different solutions for the same problem.

Largest CSPs solved with neuromorphic HW.[2]

Loihi: Nahuku board running with NxSDK 0.95 on a host Intel Core i7-9700K with 128GB RAM, running Ubuntu 16.04.6 LTS
CPU: Intel Core i7-9700K, RAM: 128GB, running Ubuntu 16.04.6 LTS.

[1] www.coin-or.org/projects/
[2] Davies et.al Proceedings of the IEEE, in review.

Performance results are based on testing as of December 2020 and may not reflect all publicly available security updates. Results may vary.

Legal Information

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Results have been estimated or simulated.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Food for thought to get started

- Why SNNs and/or neuromorphic platforms show good performance on the optimization problems?
- How would we classify the landscape of optimization problems?
  - Are some classes more amenable to acceleration by event-based algorithms than others?
- How can we create a unified framework for looking at SNN-friendly versions of optimization algorithms?
- Is there an architectural feature that we may think of, which might further enhance the performance?