Neuromorphic Graph Algorithms: Extracting Longest Shortest Paths and Minimum Spanning Trees

Bill Kay
kaybw@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Prasanna Date
datepa@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Catherine Schuman
schumancd@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

Neuromorphic computing is poised to become a promising computing paradigm in the post Moore's law era due to its extremely low power usage and inherent parallelism. Traditionally speaking, a majority of the use cases for neuromorphic systems have been in the field of machine learning. In order to expand their usability, it is imperative that neuromorphic systems be used for non-machine learning tasks as well. The structural aspects of neuromorphic systems (i.e., neurons and synapses) are similar to those of graphs (i.e., nodes and edges). However, it is not obvious how graph algorithms would translate to their neuromorphic counterparts. In this work, we propose a preprocessing technique that introduces fractional offsets on the synaptic delays of neuromorphic graphs in order to break ties. This technique, in turn, enables two graph algorithms: longest shortest path extraction and minimum spanning trees.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Graph algorithms analysis**; • **Hardware** → **Neural systems**.

KEYWORDS

Neuromorphic Computing, Graph Algorithms, Minimum Spanning Trees, Longest Shortest Path

ACM Reference Format:

Bill Kay, Prasanna Date, and Catherine Schuman. 2020. Neuromorphic Graph Algorithms: Extracting Longest Shortest Paths and Minimum Spanning Trees. In *Neuro-inspired Computational Elements Workshop (NICE '20)*, March 17–20, 2020, Heidelberg, Germany. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3381755.3381762>

Notice: This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

NICE '20, March 17–20, 2020, Heidelberg, Germany

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7718-8/20/03...\$15.00
<https://doi.org/10.1145/3381755.3381762>

1 INTRODUCTION

In computer science, we are witnessing the end of Moore's law in the twenty first century, and expect to saturate the gains that can be extracted from conventional computing platforms like CPUs and GPUs. However, the demand for faster and better computation seems to be increasing. To meet this ever increasing demand and to continue to make progress in computing, it is important to explore unconventional computing paradigms like neuromorphic computing and quantum computing [6, 7]. We focus on spiking neuromorphic computing systems (SNCs) in this work.

Traditionally, the majority of use cases for SNCs originate in the field of machine learning, for example, digit recognition, speaker recognition [9], classifying supercomputer failures [5], signal processing [21], etc. SNCs have been shown to perform extremely well on several machine learning tasks. For example, on inferencing tasks, SNCs not only perform faster than conventional computing platforms (CPUs and GPUs), but also use orders of magnitude less energy. These characteristics of SNCs, along with their inherent parallelism, make them an extremely promising computation paradigm in the post Moore's law era.

In order to expand the usability of SNCs, they must be leveraged to solve problems outside the field of machine learning, for example, graph problems, NP-complete problems, scientific computing, solving differential equations, etc. [1]. The structural aspects of SNCs, i.e., neurons and synapses, seem very similar to those of graphs, i.e., nodes and edges. However, it is not obvious if a direct mapping of graph algorithms to their neuromorphic counterparts exists. There has been a limited amount of work in the literature with respect to solving graph problems on SNCs [14–16, 23].

In this paper, we focus on two neuromorphic graph algorithms, single source longest shortest path (LSP) extraction and minimum spanning trees (MST). Specifically, we first propose a preprocessing technique that introduces fractional offsets to the synaptic delays so that no two neurons fire at precisely the same time without dramatically affecting the order in which spikes are received at their respective target neurons. We leverage our fractional offsets technique to propose neuromorphic versions of two graph algorithms. The first algorithm is used for extracting the longest shortest path in a graph from a given source node. The second algorithm is used for extracting the minimum spanning tree in a graph.

The related work has been described in Section 2. Section 3 describes the notation used in this paper. The fractional offsets preprocessing technique has been described in Section 4. Algorithms for extracting longest shortest path and obtaining minimum spanning tree are proposed in Section 5 and 6 respectively. Section 7 discusses the results obtained in this work and sheds light

on analyzing neuromorphic graph algorithms, along with further directions.

2 RELATED WORK

SNCs have become increasingly popular as a viable complementary post-Moore’s law architecture in recent years. In particular, there are a variety of neuromorphic hardware systems that have been developed and that are available in some capacity to the research community, including SpiNNaker [11] and BrainScaleS [22], both of which were developed as part of the European Union’s Human Brain Project, and IBM’s TrueNorth [19] and Intel’s Loihi [8], both of which are research chips developed in industry in the United States. Moreover, there is a tremendous amount of work in investigating new and emerging devices for neuromorphic implementation, including memristors [4], spintronics [13], and optoelectronics [26]. The availability and continued development of new hardware makes it natural to now focus on the development of new algorithmic approaches for neuromorphic computers.

Much of the work in neuromorphic computing has been focused on applications in machine learning, specifically focusing on using neuromorphic systems to implement spiking neural networks that have been trained or that learn using machine learning approaches [24]. Another class of uses of neuromorphic systems is to understand how the brain operates through neuroscience simulations [29, 30]. However, there has also been a rise in interest in utilizing neuromorphic systems for non-machine learning and non-neuroscience applications, by utilizing some of the fundamental characteristics of computation in a neuromorphic system, including massively parallel computation, collocated processing and memory, and event-driven computation [1].

Because of the inherent network-like structure of SNCs, one set of non-machine learning applications that is most natural for SNCs is graph problems [15, 16]. There have been a variety of efforts to map different graph problems onto neuromorphic systems, including community detection [14] and shortest path [8, 23]. For these types of implementations, the graph in question is mapped into a corresponding spiking neural network, for example by creating a neuron per node and a synapse per edge. Then, neurons are selectively stimulated, the SNC is run to simulate activity in the network, and the resulting spiking activity or updated network (changed as a result of synaptic plasticity, for example) gives a solution or partial solution to the graph problem.

Neuromorphic systems have also been used for high performance computing (HPC) and scientific computing tasks. Highly programmable systems such as SpiNNaker [11] lend themselves to HPC-style tasks. One example of utilizing SpiNNaker for a task in HPC is for scalable and efficient image processing, such as image smoothing, edge detection, and image sharpening [27]. In terms of scientific applications, spiking algorithms have been presented for a variety of problems, including cross-correlation particle image velocimetry [25], finding the median [31], and solving diffusion partial differential equations by modeling the random walk process [2].

A variety of other tasks for neuromorphic systems have also been proposed, including spiking algorithms for a wide range of signal processing applications [28] and Markov Chain Monte Carlo

sampling on SpiNNaker [18]. In addition, there have been a variety of efforts to map quadratic unconstrained binary optimization (QUBO) problems onto neuromorphic systems as well [3] (including for graph partitioning [20]), as well as efforts to map constraint satisfaction problems in general onto SNCs [10, 17].

In this paper, we extend the work on neuromorphic graph algorithms by [14, 15, 23]. We first propose a preprocessing technique for mapping graphs and networks onto SNCs, and show that this preprocessing technique enables us to extract a single source longest shortest path (LSP) and minimum spanning tree (MST) from a given graph.

3 NOTATION FOR GRAPH THEORY

In Section 5, we will use neuromorphic architectures to implement graph algorithms on weighted, directed graphs. In Section 6, we will use neuromorphic architectures to implement graph algorithms on weighted, undirected graphs. We remark that undirected graphs can be viewed as directed graphs when undirected edges are viewed as bidirectional edges. Hence, our definitions are all in terms of directed graphs. A weighted, directed graph $G = (V, E, w)$ consists of:

- (1) A finite set of vertices or nodes, V .
- (2) A finite set of edges, i.e. a collection of ordered pairs of V , $E \subseteq V \times V$.
- (3) A weight function $w : E \rightarrow \mathbb{R}$.

Frequently the weight function w has some sort of physical interpretation, e.g., distances on a map or resistance along a circuit. Here we present a few notational conventions that we find useful throughout the paper:

- (1) $V := \{v_1, \dots, v_n\}$.
- (2) $E := \{e_{i_t j_t}^{(1)}, \dots, e_{i_m j_m}^{(m)}\}$, where $e_{i_t j_t}^{(t)} = (v_{i_t}, v_{j_t})$. We suppress either subscripts or superscripts (or both) when convenient or clear from context. We call v_{i_t} and v_{j_t} the *tail* and *head* of $e_{i_t j_t}^{(t)}$, respectively.

The *eccentricity* of a vertex v is the length of the longest minimum distance from v to any other vertex (often called the single source longest shortest path at v , or LSP at v). In [8, 23], for a graph G and a vertex v , SNCs were used to compute the eccentricity of v . Further, the edge weights of the SNC were used to extract a subgraph which contains *every* longest shortest path from v . If the longest shortest path from v is unique, then a longest shortest path from v can be determined. However, in some graphs every edge participates in a longest shortest path at v , and so the subgraph which is returned is the entire graph. For example, if every edge has weight 1, every edge in the graph in Figure 1 participates in the LSP from v_s .

For the LSP algorithm in [8, 23], the SNC was a direct embedding of the input graph, with one neuron per vertex, one synapse per edge, and each synaptic delay set exactly equal to the value of the weight function for the corresponding edge. In the next section, we will provide a preprocessing technique which produces an SNC whose topology is the same as G , but with different synaptic delay values which allow us to extract a single longest shortest path at v .

Before we describe our preprocessing technique and algorithms for single source longest shortest path extraction and minimum

Figure 1: A graph where every edge has unit weight, and is in an LSP from v_s to v_t

spanning trees, we would like to shed some light on the characteristics of SNCs, and answer questions like which operations are permitted and which ones aren't. The SNCs are comprised of a finite number of neurons and synapses, where all-to-all connectivity of neurons is possible, i.e. it is possible to connect any neuron to any other neuron on the hardware. Although the number of neurons is finite, from a practical point of view, this number is assumed to be large enough so that it does not impose any hardware limitations on neuromorphic algorithms being designed.

Each neuron in the SNC has the following configurable parameters associated with it:

- Spiking Threshold (v_{th}): A neuron spikes when its activation potential (v) is more than its spiking threshold (v_{th}).
- Refractory Period (t_R): After a neuron has spiked, it lays dormant and does not spike for the refractory period.

Each synapse in the SNC has the following configurable parameters associated with it:

- Synaptic Delay (δ): This is the time taken by a signal to travel from pre-synaptic neuron to post-synaptic neuron along a given synapse.
- Synaptic Weight (s_w): This is the weight of the synapse.

Furthermore, once a program is setup on an SNC and run, it is possible to stop all neuromorphic activity at any point of time if required. This is unlike quantum computing algorithms, where measurement of the quantum system destroys the state of the qubits. It is also possible to communicate with the classical processors (CPUs or GPUs and their memory) when a neuromorphic program is paused or stopped. Also, all neuron and synapse parameters can be individually configured before, during or after the execution of a neuromorphic program. Some operations that are not permitted include memory operations. Memory is not available implicitly on the SNC, and any algorithm that is memory intensive must be avoided if possible.

4 PREPROCESSING TECHNIQUE

In previous implementations of graph algorithms via SNCs, the graphs have been directly embedded into SNCs whose synaptic delays were precisely given edge weights. We introduce a preprocessing step which takes a weighted, directed graph $G = (V, E, w)$ and produces a new weight function h so that the graph $\hat{G} = (V, E, h)$ can be directly embedded into an SNC that allows us to solve an instance of either LSP or MST for G . We focus on the versions of these problems where the edge weights are non-negative integers. If they are not, we can take rational approximations of the weights and clearing denominators (but we don't factor this into our preprocessing runtime). Let $E = \{e^{(1)}, e^{(2)}, \dots, e^{(m)}\}$ be the edge set of G . We define the weight function h by adding a fractional offset o_i to the original edge weight:

$$h(e^{(i)}) := w(e^{(i)}) + \frac{1}{2^i} = w(e^{(i)}) + o_i$$

The collection of the o_i s satisfies two properties that we will exploit:

- (1) $\sum_{i=1}^m o_i < 1$ (since the infinite sum is precisely 1).
- (2) For any two distinct sets of indices A and B ,

$$\sum_{i \in A} o_i \neq \sum_{j \in B} o_j.$$

We will do all our analysis assuming the fractional offset weights are implementable in our neuromorphic system. If a neuromorphic system requires integer weights, simply multiply the new weight function by 2^m to clear fractions. Given an SNC which is a direct embedding of G , sometimes spiking a single vertex will result in several neurons spiking at the same time. It is for this reason that [8, 23] were able to extract a subgraph with every single longest shortest path, and also allowed for cycles to appear in subgraphs that were extracted. The two above observations allow us to directly embed $\hat{G}$ into an SNC where no two neurons spike at the same time, the order in which neurons spike does not change from the order that they would spike in a direct embedding of G (up to ties), and the runtime of an SNC for $\hat{G}$ is within one unit of the runtime of an SNC for G . This will be made explicit for LSP and MST in the subsequent sections. Notice, the preprocessing step has runtime $O(|E|)$, which is faster than either LSP or MST and hence does not increase the big O runtime of either algorithm.

5 LONGEST SHORTEST PATH EXTRACTION

Given a weighted directed graph $G = (V, E, w)$, we perform the aforementioned preprocessing step to obtain $\hat{G} = (V, E, h)$, where h is simply w with fractional offsets added. As in [8, 23], we directly embed $\hat{G}$ into an SNC which has one neuron per vertex, one synapse per edge, and system parameters ($v_{th} = 0, t_R = \alpha, \delta = h(e) + 1, s_w = 1$), where α is given by the expression:

$$\alpha := \left(\sum_{e \in E} h(e) + 1 \right) + 1.$$

Setting the refractory period this high ensures that each neuron fires only once. The algorithm to extract the longest shortest path from a source v_s is the same as in [8, 23], which we include as Algorithm 1

Algorithm 1: LSP Algorithm on SNC**Result:** Compute the length of a longest shortest path.Spike v_s ;**while** *Some neuron has not yet spiked* **do**

Run the SNC;

if *A neuron spikes* **then**

Record the time at which the neuron spikes;

end**end****return** Runtime

We note that we either implicitly assume that every vertex is reachable from v_s (and so this algorithm terminates in finite time), or we define the longest shortest path to be infinite when some vertex is not reachable from v_s . If we are interested in the longest shortest path reachable from v_s we can run the algorithm for α steps, and the last neuron which fires is the longest shortest path which is reachable from v_s . In the course of running the above algorithm, each time a post synaptic neuron spikes the weight of the synapse changes. Hence, we can examine the weights when the algorithm terminates and see which synapses caused which neurons to spike. In [8, 23], they showed that this algorithm terminates in precisely the eccentricity of v_s time. We have the following lemma which will allow us to read the longest shortest path from the SNC:

LEMMA 5.1. *After running the LSP Algorithm, the SNC induced on only the synapses which caused a neuron to spike is a tree.*

Once we establish Lemma 5.1, we can find the longest shortest path since each neuron spiked because of a *distinct* synapse, and so we can trace the path from the end to the beginning by looking at the last neuron to spike and moving backwards through the tree. We include now a proof of Lemma 5.1.

PROOF. Let G be the graph which is left after the LSP algorithm runs, and suppose there are two paths, P_1 and P_2 with the same starting and ending vertices v_1 and v_2 , and without loss of generality are the first such paths to arise while running the LSP algorithm. That is:

$$P_1 = u_1 e^{(i_1)} u_2 e^{(i_2)} \dots u_{r-1} e^{(i_{r-1})} u_r$$

$$P_2 = u'_1 e^{(j_1)} u'_2 e^{(j_2)} \dots u'_{s-1} e^{(j_{s-1})} u'_s$$

with $u_1 = u'_1 = v_1$ and $u_r = u'_s = v_2$, and each e is an edge in G . The edges $e^{(i_{r-1})}$ and $e^{(j_{s-1})}$ are both in G , which means they were each the reason v_2 spiked. This can happen if and only if P_1 and P_2 have the same length. However:

$$\begin{aligned} h(P_1) &= \sum_{k=1}^{r-1} h(e^{(i_k)}) \\ &= \left(\sum_{k=1}^{r-1} w(e^{(i_k)}) + o_{i_k} \right) \\ &= \left(\sum_{k=1}^{r-1} w(e^{(i_k)}) \right) + \left(\sum_{k=1}^{r-1} o_{i_k} \right) \end{aligned}$$

where the first sum is an integer, and the second sum is between 0 and 1 by the properties of the fractional offsets. By the same logic:

$$h(P_2) = \left(\sum_{k=1}^{s-1} w(e^{(j_k)}) \right) + \left(\sum_{k=1}^{s-1} o_{j_k} \right)$$

where the first sum is an integer, and the second sum is a *different* number between 0 and 1 from before (as P_1 and P_2 have different edges and thus different offsets). Hence P_1 and P_2 have different weights, and so only one of their final synapses caused v_2 to spike. $\square$

In the worst case, the algorithm runs in α steps, although the algorithm could terminate much quicker if every neuron spikes sooner than that. If the initial edge weights are bounded, the runtime is $O(|E|)$, and if the edge weights are allowed to grow with the size of the input the runtime is $O(\max_{e \in E} w(e) * |E|)$.

6 MINIMUM SPANNING TREES

Given a connected weighted graph G , a *Minimum Spanning Tree* is a tree T which is a subgraph of G on the entire vertex set of G (T *spans* G) and the sum of the edge weights of T are as small as possible. Every connected weighted graph G contains a minimum spanning tree. Among algorithms for computing a minimum spanning tree, one of the most well known is Prim's algorithm [12]. We present Prim's algorithm as Algorithm 2

Algorithm 2: Prim's Algorithm**Result:** Produce a minimum spanning tree of G .Initialize T as a single vertex;**while** T *does not span* G **do**

 Among all the edges with one vertex in T and one not in T , pick the one with smallest weight, breaking ties arbitrarily. Add the new vertex and edge to T ;

end**return** T

We now provide an SNC and spiking algorithm which performs exactly as in Prim's algorithm. Given a weighted, undirected graph $G = (V, E, w)$, we perform the preprocessing step described in Section 4 to obtain $\hat{G} = (V, E, h)$. We embed $\hat{G}$ directly into an SNC with system parameters ($v_{th} = 0, t_R = \alpha, \delta = h(e) + 1, s_w = 1$), where α is given by the expression:

$$\alpha := \left(\sum_{e \in E} h(e) + 1 \right) + 1.$$

We now provide Algorithm 3 for computing a minimum spanning tree using an SNC.

As before, the preprocessing step assures that each pass through the while loop after spiking every neuron in S , exactly one new neuron spikes. In particular, the post synaptic neuron which is adjacent to a neuron in S of minimum weight. Since the preprocessing step preserves the order of edge weights (breaking ties with fractional offsets), each synapse added to T corresponds to an edge in G which would have been a valid choice for Prim's algorithm. Moreover, by a similar argument as in the proof of Lemma 5.1, the

Algorithm 3: MST Algorithm on SNC

Result: Produce a minimum spanning tree of G .
Initialize $S = \{v_s\}$, $T = \{\}$. ;
while $S \neq V$ **do**
 Spike every neuron in S ;
 if v spikes as the post synaptic neuron of synapse e **then**
 Stop neuromorphic activity;
 Add v to S ;
 Add e to T ;
 end
end
return T

collection of synapses T is a tree at every pass through the while loop. Thus, the tree T returned when $S = V$ at the end of the algorithm is a tree which could have been selected by Prim's algorithm, and is hence minimal. Note that while the delays on the tree T differ from the weights in the original graph G by a fractional offset, when computing the weight of the minimum spanning tree in G we can simply look at the integer parts of the synaptic delays in T . The while loop executes exactly $|V|$ times, and on the t th execution we have to set up t spikes. If weights of the graph are bounded while the input size grows, this yields a runtime of $O(|V|^2)$. If the weights are allowed to grow with the input, we have to wait at most $\max_{e \in E} h(e)$ time steps per execution of the while loop, yielding a big O runtime of $O(\max_{e \in E} w(e) * |V|^2)$.

7 DISCUSSION

First of all, we would like to shed some light on how we went about computing the runtime of Algorithms 1 and 3. Algorithm 1 would run for α amount of time in the worst case. Note that α is a measure of the *weight* of the edges ($h(e)$) and not the *number* of edges of the graph. So, the runtime of Algorithm 1 would depend on the actual values of the weights. For example, if a three node clique has edge weights 1, 1000 and 2000, then the neuromorphic LSP algorithm would run for at least 1001 units of time. This is unlike a classical algorithm, which might run for three units of time to scan each of the three edges. Note that for the neuromorphic algorithm, the dependency of runtime is on the *values* of weights, while that for a classical algorithm is on the *number* of weights. Thus, to evaluate the big O expression for runtimes of neuromorphic algorithms, we need to take this dependency into account. For Algorithm 1, in the worst case, it would run for α units of time, and since α is $O(\max_{e \in E} w(e) * |E|)$, the exact worst case runtime of the algorithm is also $O(\max_{e \in E} w(e) * |E|)$. Now, consider a situation where all weights are bounded, i.e. we know beforehand that none of the weights are going to be more than a particular value, say w_{max} . In this case, the running time becomes $O(w_{max}|E|)$. Now, since we already know w_{max} beforehand, it can be treated as a constant, in which case the running time becomes $O(|E|)$. This is only possible when we know a priori that the weights are bounded. If no such information is available about the weights, then we revert back to $O(\max_{e \in E} w(e) * |E|)$ as the running time of Algorithm 1.

We can apply a similar argument for evaluating the running time of Algorithm 3. The while loop in Algorithm 3 runs for $O(|V|)$

time, and spiking every neuron in the set S takes $O(|V|)$ time. So, the while loop essentially takes $O(|V|^2)$ time. Note that, we have not yet accounted for the time taken for spikes to travel across the graph, which is a measure of the synaptic delays (equal to edge weights of the original graph). When we take these synaptic delays into account, and apply a similar line of reasoning as above, we end up with the expression $O(\max_{e \in E} w(e) * |V|^2)$. If somehow, we were able to get a bound on the edge weights of the original graph, then the running time would be $O(|V|^2)$. We can compare this running time to the running time of Prim's algorithm, which is $O(|E| \log |V|)$. Note that for extremely dense graphs, or complete graphs, $|E| = O(|V|^2)$. In this case, the run time of Prim's algorithm actually becomes $O(|V|^2 \log |V|)$. Note that the running time for our neuromorphic algorithm under bounded weights is $O(|V|^2)$, which is better than $O(|V|^2 \log |V|)$. Thus, when dealing with extremely dense graphs or complete graphs, for instance, social network graphs, the neuromorphic MST algorithm might be a better choice than Prim's algorithm, when the edge weights are known to be bounded.

It is easy to skip these minute details while computing the running times for neuromorphic algorithms, especially given its inherent parallelism. For example, in the classical world, if a neuron were to spike and send signals to all of its N neighbors, this process would take $O(N)$ time. However, the same process in neuromorphic world takes only $O(1)$ time, because of the inherent parallelism of SNCs. Once a neuron spikes, the signal is sent to all of its N neighbors in one shot. This happens in parallel, and unlike the classical world, we do not have to loop over all its neighbors, which would have taken $O(N)$ time. While on the one hand, SNCs allow for this intrinsic parallelism, on the other hand, one must be extremely careful and account for the time taken for signals to travel across the graph by taking into consideration the synaptic delays and other such parameters. So far, all our neuromorphic algorithms have been comparable to their classical counterparts in terms of running times. However, one can think of algorithms and applications which are better suited on neuromorphic systems than classical systems or vice versa. It is important to know the pros and cons of these hardware systems for extreme heterogeneous computing applications.

8 CONCLUSION

Here we presented a pair of neuromorphic graph algorithms for SNCs – the extraction of a longest shortest path and finding a minimum spanning tree. Work on graph algorithms such as computing the length of a longest shortest path, extracting a subgraph which contained *every* longest shortest path, and extracting the neighborhood of a vertex [8, 23] were all implementable via directly embedding the graph in question into an SNC. Hence, there was no need for classical hardware (CPUs and / or GPUs). However, the set of graph algorithms which can be computed via direct embedding into SNCs seems to be limited. The results in this paper introduce a preprocessing step which made possible two graph algorithms which, a priori, were not obviously computable via direct embedding. The preprocessing step would be performed on classical hardware, but did not increase the big O runtime of either algorithm. It is also worth noting that while the LSP algorithm

returns an LSP as well as the length of the path (given by the runtime of the algorithm), the MST algorithm returns the tree but does *not* return the weight of the tree – this requires a postprocessing procedure which also does not increase the big O runtime of the algorithm ($O(|E|)$ steps to add up the weights from the extracted tree). While preprocessing and postprocessing steps have allowed us to perform more graph algorithms via SNC, our procedure only changes the edge weights. In the future, we would like to explore what is possible when the preprocessing of G produces an SNC with entirely different topology. A common technique in the field of complexity theory is to turn an instance of one problem into another by building gadgets that accomplish different tasks in different settings. It is likely that by introducing gadgets, we can build SNCs that compute more complex graph algorithms.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725.

REFERENCES

- [1] James B Aimone, Kathleen E Hamilton, Susan Mniszewski, Leah Reeder, Catherine D Schuman, and William M Severa. 2018. Non-neural network applications for spiking neuromorphic hardware. In *3rd International Workshop on Post-Moore's Era Supercomputing (PMES 2018)*, Dallas, TX.
- [2] James Bradley Aimone, Aaron Jamison Hill, Richard B Lehoucq, Ojas D Parekh, Leah Reeder, and William Mark Severa. 2018. *Neural Algorithms for Low Power Implementation of Partial Differential Equations*. Technical Report. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States).
- [3] Md Zahangir Alom, Brian Van Essen, Adam T Moody, David Peter Widemann, and Tarek M Taha. 2017. Quadratic Unconstrained Binary Optimization (QUBO) on neuromorphic computing system. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 3922–3929.
- [4] Gangotree Chakma, Md Musabbir Adnan, Austin R Wyer, Ryan Weiss, Catherine D Schuman, and Garrett S Rose. 2017. Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 1 (2017), 125–136.
- [5] Prasanna Date, Christopher D Carothers, James A Hendler, and Malik Magdon-Ismael. 2018. Efficient Classification of Supercomputer Failures Using Neuromorphic Computing. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 242–249.
- [6] Prasanna Date, Robert Patton, Catherine Schuman, and Thomas Potok. 2019. Efficiently embedding QUBO problems on adiabatic quantum computers. *Quantum Information Processing* 18, 4 (2019), 117.
- [7] Prasanna Date, Catherine Schuman, Robert Patton, and Thomas Potok. 2019. A Classical-Quantum Hybrid Approach for Unsupervised Probabilistic Machine Learning. In *Future of Information and Communication Conference*. Springer, 98–117.
- [8] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham China, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.
- [9] Steve K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, et al. 2013. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–10.
- [10] Gabriel A Fonseca Guerra and Steve B Furber. 2017. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience* 11 (2017), 714.
- [11] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. 2014. The spinnaker project. *Proc. IEEE* 102, 5 (2014), 652–665.
- [12] Harvey J Greenberg. 1998. Greedy algorithms for minimum spanning tree. *University of Colorado at Denver* (1998).
- [13] Julie Grollier, Damien Querlioz, and Mark D Stiles. 2016. Spintronic nanodevices for bioinspired computing. *Proc. IEEE* 104, 10 (2016), 2024–2039.
- [14] Kathleen E Hamilton, Neena Imam, and Travis S Humble. 2017. Community detection with spiking neural networks for neuromorphic hardware. In *Proceedings of the Neuromorphic Computing Symposium*. ACM, 9.
- [15] Kathleen E Hamilton, Tiffany M Mintz, and Catherine D Schuman. 2019. Spike-based primitives for graph algorithms. *arXiv preprint arXiv:1903.10574* (2019).
- [16] Kathleen E Hamilton, Catherine D Schuman, Steven R Young, Neena Imam, and Travis S Humble. 2018. Neural Networks and Graph Algorithms with Next-Generation Processors. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1194–1203.
- [17] Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. 2016. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in neuroscience* 10 (2016), 118.
- [18] Daniel R Mendat, Sang Chin, Steve Furber, and Andreas G Andreou. 2015. Markov Chain Monte Carlo inference on graphical models using event-based processing on the SpiNNaker neuromorphic architecture. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 1–6.
- [19] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.
- [20] Susan M Mniszewski. 2019. Graph Partitioning as Quadratic Unconstrained Binary Optimization (QUBO) on Spiking Neuromorphic Hardware. In *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 4.
- [21] Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. 2019. Low-power neuromorphic hardware for signal processing applications. *arXiv preprint arXiv:1901.03690* (2019).
- [22] Johannes Schemmel, Daniel Brüderle, Andreas Gribbl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. 2010. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 1947–1950.
- [23] Catherine D Schuman, Kathleen Hamilton, Tiffany Mintz, Md Musabbir Adnan, Bon Woong Ku, Sung-Kyu Lim, and Garrett S Rose. 2019. Shortest path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*. ACM, 3.
- [24] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).
- [25] William Severa, Ojas Parekh, Kristofor D Carlson, Conrad D James, and James B Aimone. 2016. Spiking network algorithms for scientific computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 1–8.
- [26] Jeffrey M Shainline, Sonia M Buckley, Richard P Mirin, and Sae Woo Nam. 2017. Superconducting optoelectronic circuits for neuromorphic computing. *Physical Review Applied* 7, 3 (2017), 034013.
- [27] Indar Sugiarto, Gengting Liu, Simon Davidson, Luis A Plana, and Steve B Furber. 2016. High performance computing on spinnaker neuromorphic platform: A case study for energy efficient image processing. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.
- [28] Olivier Temam and Rodolphe Hélot. 2011. Implementation of signal processing tasks on neuromorphic hardware. In *The 2011 International Joint Conference on Neural Networks*. IEEE, 1120–1125.
- [29] Sacha J Van Albada, Andrew G Rowley, Michael Hopkins, Maximilian Schmidt, Johanna Senk, Alan B Stokes, Francesco Galluppi, Dave R Lester, Markus Diesmann, and Steve B Furber. 2016. Full-scale simulation of a cortical microcircuit on SpiNNaker. In *Frontiers in Neuroinformatics Conference Abstract: Neuroinformatics*, Vol. 10.
- [30] Sacha Jennifer van Albada, Andrew G Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan Barry Stokes, David R Lester, Markus Diesmann, and Steve B Furber. 2018. Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Frontiers in neuroscience* 12 (2018), 291.
- [31] Stephen J Verzi, Craig M Vineyard, Eric D Vugrin, Meghan Galiardi, Conrad D James, and James B Aimone. 2017. Optimization-based computation with spiking neurons. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015–2022.