Spike-based graph centrality measures

Kathleen Hamilton

Oak Ridge National Laboratory
Oak Ridge, Tennessee

Prasanna Date

Oak Ridge National Laboratory
Oak Ridge, Tennessee

Tiffany Mintz

Oak Ridge National Laboratory
Oak Ridge, Tennessee

Catherine D. Schuman

Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

We derive several spike-based routines that compute or establish bounds on radial centrality measures for undirected graphs and trees without the use of matrix multiplication. These spike-based centrality measures utilize a direct embedding of graph nodes and edges into neurons and synapses, can be implemented with static synapses or plastic synapses, and rely on minimal post-processing of spike rasters. This work contributes to the growing set of graphical applications for neuromorphic hardware.

CCS CONCEPTS

• **Hardware** → *Neural systems*; • **Theory of computation** → *Theory and algorithms for application domains*; **Graph algorithms analysis**; • **Applied computing** → *Consumer health*.

KEYWORDS

graph algorithms, spiking neural networks, neuromorphic applications

ACM Reference Format:

Kathleen Hamilton, Tiffany Mintz, Prasanna Date, and Catherine D. Schuman. 2020. Spike-based graph centrality measures. In *International Conference on Neuromorphic Systems 2020 (ICONS 2020)*, July 28–30, 2020, Oak Ridge, TN, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3407197.3407199>

1 INTRODUCTION

Spike-based algorithms designed to run on neuromorphic hardware rely on discrete electrical pulses transmitted via weighted connections to implement programmatic steps. While computational models of spiking neurons have been developed to describe

the behavior of biological neurons, our goal in this work is to fundamentally exploit the spiking behavior encoded by a given mathematical neuron model. Specialized processors optimized to implement spiking neurons and spike-based algorithms are known as spiking neuromorphic computing systems (SNCs) [28]. Many SNC platforms are under development (e.g., IBM TrueNorth [20], Intel Loihi [7], SpiNNaker [21], non-volatile memory devices [6]) with different capabilities [17] but similar strengths such as low communication costs. Developing new applications for neuromorphic hardware is a growing field of research [1], with algorithms being developed for community detection [14, 15, 22, 23], shortest path finding [26], Markovian random walks [29] and general scientific computing [30].

In network science the calculation of centrality measures is used to classify the “importance” of individual nodes on the overall network structure. Identifying highly important nodes is vital for studying dynamics on networks such as the spread of misinformation in social networks [33], spread of disease [2, 19] or identifying vulnerabilities in power systems [8]. There are a number of centrality measures in the network science literature [4, 5, 9, 10] and in [4, 5] centrality measures were classified according to graph theoretic concepts. Centrality measures are grouped into similar classes based on the type of graph walk, the properties of the walk (number of walks or lengths of walks) and whether the centrality is based on dynamics that emanate from a vertex (radial) or pass through a vertex (medial). Using the graph theoretic classifications of [4, 5]-radial, volume-based centrality measures can be extracted from to spiking neuron dynamics with minimal network engineering.

This paper contains the derivation of several spike-based algorithms for radial, volume-based centrality measures. In Section 2.3, we derive a set of spike-based primitives for computing various centrality measures for undirected graphs and present examples of hardware agnostic implementations that rely on deterministic routines and can be implemented with static or plastic synapses. In Section 3, we show how trails, walks and paths can be implemented via spiking neurons. In Section 4, we build our spike-based centralities from these implementations. In Section 5 we present preliminary results on the performance of these spike-based measures. Several spike-based centralities are simulated using the Brian2 [12] Python library and the standard graph centrality measures are computed using the routines available in NetworkX [13].

The spike-based methods we develop in this paper are “in-memory” approaches. Real-world graphs and networks can contain billions of vertices and edges. Current SNC hardware can feasibly implement millions of spiking neurons, but depending on the density of the target graph, the size of the spiking neuron system (SNS) that can

Research conducted while T. M. Mintz was affiliated with Oak Ridge National Laboratory
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.

ICONS 2020, July 28–30, 2020, Oak Ridge, TN, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8851-1/20/07...\$15.00
<https://doi.org/10.1145/3407197.3407199>

be realized on a particular SNC is limited by the sparsity of synapse connectivity. The computational cost of defining effective hardware embeddings for SNS remains an open question and in this work we are only considering graphs that can be fully constructed as a single SNS.

2 DEFINITIONS

In Section 2.1, we introduce the notation used for relevant graph quantities, and in Section 2.2, we define neuronal and synaptic parameters and describe how a graph is mapped onto spiking neurons. Our approach to designing spike based algorithms is based on the direct (1 : 1) mapping of graph nodes to spiking neurons and symmetric pairs of directed synapses, and as a result we use the same symbol (n) to refer to graph nodes and spiking neurons.

2.1 Graph definitions

An undirected graph $\mathcal{G}(n, E)$ is defined by a node set $n(\mathcal{G}) = [n_0, n_1, \dots, n_{|G|-1}]$ and a set of undirected edges $E(\mathcal{G})$, where an edge is a tuple labeled by the nodes it connects $e_{ij} = (i, j) = e_{ji}$. The edges of an undirected graph can be represented by a symmetric, binary matrix called the graph adjacency matrix $A_{\mathcal{G}}$. Throughout this work we only consider graphs that are undirected, connected and do not contain self-loops.

A graph walk of length N can be described by a sequence of vertices and edges, indexed by the order in which they are visited (e.g., $\mathcal{L} = \{V_0, E_0, V_1, E_1, \dots, V_{N-1}\}$) and these traversals can be distinguished by the restrictions on which vertices and edges can be re-visited and re-used. This is the first characteristic used in [5] to classify centralities. A *walk* is the most general traversal; vertices and edges can be revisited (i.e., V_i and an edge E_i can occur multiple times in $\mathcal{L}$). A *trail* can revisit a vertex multiple times, but edges can only be traversed once. A *path* is the most restricted traversal: all vertices and edges in $\mathcal{L}$ are unique.

2.2 Spiking neuron system definitions

We define the SNS in graphical notation as $\mathcal{S}(n_l, s_k)$ which are connected, edge-weighted networks of spiking neurons (these models are also described in [27]). The internal state of the neurons (a time-dependent electrical potential) will control when spikes are fired and we assume that the neurons are fully deterministic.

The neuronal spiking behavior is controlled by two parameters: the spike threshold (v_{th}) and the refractory period (t_R). When the internal state of a neuron exceeds the spike threshold a spike is fired and the neuron enters a refractory period during which its internal state does not change. We distinguish the spikes fired by a neuron as “external” or “internal” firing; external firing is caused by an external stimulus applied to the neuron and internal firing is caused by one or more spikes arriving from other neurons.

Spikes are transmitted along synapses. The synapses are parameterized by weight (s_w), delay (δ), and plasticity (α). The plasticity value is the learning rate α . Static synapses cannot learn and are labeled with $\alpha = 0$. If $\alpha \neq 0$, then the synapses are plastic. Plasticity is not a requirement for the algorithms presented in this paper.

For the tasks presented in this paper the SNS are constructed from a given graph ($\mathcal{G}$) via *direct mapping*. A graph ($\mathcal{G}$ or $\mathcal{D}$) is directly mapped to an SNS by defining a neuron $n_i \in n$ for each

$v_i \in V$ and symmetric pairs of synapses $e_{ij} \rightarrow (s_{i \rightarrow j}, s_{j \rightarrow i})$ for each edge $e \in E$. Direct mapping preserves the connectivity of the original graph, and in particular ensures that graph elements such as triangles and cycles are also present in the SNS. A triangle on an undirected graph is defined by an unordered tuple of vertices $(v_i, v_j, v_k) \in V(\mathcal{G})$ where $(e_{ij}, e_{jk}, e_{ik}) \in E(\mathcal{G})$. Under direct mapping, any triangle (v_i, v_j, v_k) will map to a tuple of neurons $(n_i, n_j, n_k) \in \mathcal{S}(n, s)$.

Direct mapping defines the connections of an SNS; however, an additional non-trivial embedding step is needed to implement the SNS on a neuromorphic platform. For the local tests we implement in this paper we assume that the fan-in and fan-out connections of each neuron on the hardware are large enough, and the hardware contains sufficient neurons and synapses to realize the graph in the SNS.

2.3 Neuron driving

The synaptic connections defined from the graph mapping connect the neurons to each other. Each neuron is also connected to an external source of spikes via a set of directed connections. These are defined with large positive synaptic weights that exceed the firing threshold of a neuron, ensuring that a spike that is directed along the input connection will drive a neuron to internally fire for the application of stimuli (represented by a directed arc that terminates at a neuron) and a directed edge that allows for the recording of when a neuron fires a spike. A schematic representation showing the general input and output connections to an SNS is shown in Fig. 1.

Figure 1: Schematic showing how external stimuli can be applied to neurons of an SNS. Discrete spikes are fed to a neuron via an external source along directed arcs (solid red arrow) with synaptic weights defined to be greater than the neuron firing threshold. The arrival of an external spike at a neuron will immediately cause it to internally fire.

External spikes fed to the system ($\{\chi_i^{(T)}, \chi_j^{(T)}\}$) are indexed by neuron (i, j) and the number of spikes to be generated by the target neurons (T). These external spikes are used to initiate a spike-based algorithm by applying an external stimulus $\{\chi_i^{(T)}\}$ to a single neuron (single neuron driving) or applying external stimuli to multiple neurons simultaneously (multiple neuron driving) $\{\chi_i^{(T)}, \chi_j^{(T)}, \dots, \chi_n^{(T)}\}$. Single neuron driving can be used to extract the nearest neighbors of a neuron, and distance characteristics of a graph (e.g. vertex eccentricity, graph radius and diameter).

Multiple neuron driving can be used to implement parallelized routines, or to extract correlated behavior. For a set of external driving spikes sent to m neurons ($\{\chi_i^{(T)}, \chi_j^{(T)}, \dots, \chi_m^{(T)}\}$) at clock

time $t = 0$, the maximum number of spikes that can arrive at a (non-driven) neuron at clock time $t = 1$ one time step is m .

This property can be used to implement a simple *clique test*. For a SNS $\mathcal{S}$ with synapses from the graph $\mathcal{G}$, if we want to test whether a group of neurons $Q = \{n_i\}$ forms a clique $\mathcal{K}_{|Q|}$ we can construct a spike algorithm that will cause all the neurons in Q to spike only if $Q = \mathcal{K}_{|Q|}$. First the neuronal parameters are defined as: $\{v_{th}(n_i) = |Q| - 1 \mid n_i \in Q\}$, $\{v_{th}(n_i) = |E| \mid n_i \notin Q\}$, $t_R = 0 \forall n$. Then, the synaptic parameters are defined as: $s_w = 1, \alpha = 0$. Finally, this SNS is incorporated into a spike algorithm ($\mathbb{P}_{\text{clique}}$) by applying multiple neuron driving to the neurons in set Q : $\{\chi\} = \{\chi_i^{(1)} \mid n_i \in Q\}$. After $t = 2$ time steps if all $n_i \in Q$ fire a spike then Q is a clique $\mathcal{K}_{|Q|}$.

Multiple neuron driving can be used to identify within 1 time step all neurons that are within 1 edge of any of the neurons being driven, but the spike raster alone is insufficient to distinguish which neurons are closest to which driven neuron. By introducing plasticity and a simple spike timing dependent learning rule, plasticity plays the role of an indicator function for quick identification of which neurons caused subsequent firing in the SNS. Then the nearest neighbors of each driven neuron can be identified from the synapses that have potentiated and have $s_w > 1$.

When a neuron fires a spike, that spike is transmitted along all outgoing synapses; this results in spikes simultaneously arriving at multiple target neurons. Simply increasing the number of neurons receiving external stimuli is not guaranteed to result in gaining more information from a graph (see Fig. 2) but by incorporating more complex neuron structures (heterogeneous spike thresholds, synapse plasticity) it is possible to design parallel implementations of algorithms or extract higher order correlated graph elements such as triangles.

Figure 2: [Left] Schematic showing external stimulus applied to multiple neurons in a system without synapse plasticity and neurons that internally fire within 1 time step. [Right] Schematic showing neurons that internally fire within 1 time step and how that affects individual synapse weights.

Information is extracted from an SNS by analyzing the spike behavior of individual neurons, by measuring the time dependent synaptic weights in the system, or both. The overall spiking behavior is dependent on the neuronal and synaptic parameters in the SNS and instigated by external spikes.

A spike-based algorithm defined on a set of neurons needs to specify how spikes are generated, how they propagate throughout the set of neurons, how and when they will be measured and finally how information is extracted. To fully define a spike-based algorithm ($\mathbb{P}$) we require the following elements: $\mathcal{S}$ (an SNS to define the connections), $\{\ell\}$ (the neuronal parameters), $\{\mathbf{k}\}$ (the synaptic parameters), $\{\chi\}$ a set of external stimuli, and (t) the number of time steps that spikes are allowed to propagate. Since the internal

spiking generated by external driving may not self-terminate, (t) is necessary to fully define a specific algorithm. By changing the parameters of an SNS, or by changing how external stimuli are applied, we can implement different algorithms on the same SNS.

Single neuron driving of n_i causes a series of spikes to travel outward from n_i through the system. If spikes are allowed to propagate through $\mathcal{S}$ for more than one time step, then it is possible for connected systems of spiking neurons to exhibit chaotic spike patterns [11, 31, 32]. For the purpose of exploring and analyzing the structure of a graph we need to control the direction in which new spikes are generated. For example, to extract information about how far a series of spikes has spread from the initially driven neuron n_i , and to identify all neurons reachable from n_i by at least a finite number of non-backtracking steps, we need to enforce a non-backtracking condition.

3 GENERATING GRAPH WALKS

We now describe how to generate spiking dynamics that share the characteristics of walks, trails and paths as defined in 2.1. In general, since spikes can only propagate along synaptic connections, it is apparent that if neuron n_i fires a spike at time step $t = 0$, and then neuron n_j fires at time step $t = r$ then n_j must be connected to n_i by path on the underlying graph $\mathcal{G}$. Whether that connection is made via a walk, trail or path is dependent on the neuron and synaptic parameters.

- **General walks:** With a short (or trivial) refractory period ($t_R = 0$) a neuron that has fired in a previous time step can fire again with the arrival of new spikes. Additionally with static synapses the same connections can be re-used multiple times. Thus a general walk in which vertices and edges are re-visited and re-used is straightforward to implement.
- **Trails:** With the use of plastic synapses with large synaptic weight decay that converts excitatory synapses into inhibitory synapses ($\alpha \ll 0$) we can enforce the constraint that any edge of the graph is traversed at most once, ensuring that spikes traverse the underlying graph via trails.
- **Paths:** With the use of neurons with large refractory periods ($t_R = |E(\mathcal{G})|$) we can enforce the constraint that any vertex of the graph is visited at most once, ensuring that spikes traverse the underlying graph via non-backtracking paths.

In addition to creating non-backtracking paths, using a large refractory period ($t_R = |E(\mathcal{G})|$) will also construct an SNS that has self-terminating spike sequences. The spikes generated in an SNS constructed from direct mapping of a connected, undirected graph $\mathcal{G}$ where each neuron has refractory period $t_R = |E(\mathcal{G})|$ will terminate once all neurons have fired once.

With static synapses the spiking dynamics can be used to define an upper bound on the shortest path length between two vertices $(v_i, v_j) \in V(\mathcal{G})$. If single neuron driving is applied to n_i , and n_j fires within r time steps then the shortest path connecting $|\mathcal{P}(v_i \rightarrow v_j)| \leq r$.

Single neuron driving of n_i can be used to identify nearest neighbors, or find an upper bound on the shortest path between two vertices, or define the eccentricity $\epsilon(v_i)$. However with static synapses it will not identify the unique edges that exist in the neighborhood of n_i , or comprise $\mathcal{P}(v_i \rightarrow v_j)$. This behavior can also be used to

construct spike-based methods for shortest path finding but this requires synaptic plasticity [25]. In Sec. 2, we noted that plasticity is an effective indicator label but the lack of plasticity does not limit static synapses to only be used with single neuron driving. With higher spike threshold values, we can use multi-neuron driving to find local graph elements such as triangles.

3.1 Post-processing of spike rasters

Radial and medial centrality measures are computed using the total number of paths generated from a single node. When converted into a spike-based algorithm, this relaxes the control needed over individual spike dynamics, but additional post-processing is needed before computing a final centrality measure. Once a spike raster is generated from a spike algorithm, the raster is decoded using population-rate decoding. At each clock step the number of spiking neurons are counted.

4 SPIKE-BASED CENTRALITIES

4.1 Degree Centrality and the Sade k -path Centrality

The degree centrality of a vertex ($d(v_i)$) is the number of vertices it is connected to by a single edge, or equivalently the number of length-1 paths that connect to v_i . Measuring the degree centrality is equivalent to counting the number of nearest neighbor vertices on the graph. This is implemented via a spiking algorithm $\mathcal{P}_1$ that finds the equivalent neuron set, and operates by applying an external stimulus to a single neuron (χ_i^1) which will cause internal spiking from only the nearest neighbors in the next time step ($t = 1$). To complete the program $\mathbb{P}_1(\mathcal{S}, \chi_i^1, t = 1)$, the neuron and synaptic parameters are set: $\mathcal{S}(N_{v_{th}=1}, W_{sw=1}, \alpha=0)$. Population decoding is applied to the spike raster to compute the degree centrality.

To compute the k -path centrality measure [5, 24] we need to count the number of individual k -length paths emanating from an individual neuron. If the underlying graph $\mathcal{G}$ is a tree then the Sade k -path centrality can be measured by extending the time that spikes are allowed to propagate in $\mathcal{P}_1$. The spike dynamics generated in $\mathcal{P}_1$ will not self-terminate and the user must halt the spiking dynamics or simply readout the spike times after 1 time step (MCT=1). Using the neuron and synaptic parameters as above: $\mathcal{S}(N_{v_{th}=1}, W_{sw=1}, \alpha=0)$, but allowing the spikes to propagate for a number of time steps (k) then after k time steps, all neurons that can be reached with at least k non-backtracking steps will have fired at least once. We use this to define $\mathcal{P}_k(\mathcal{S}, \chi_i^1, t = k)$

After applying population decoding to the entire spike raster, the number of neurons that fire in each clock step (t) are the number of t -length paths emanating from n_i . After decoding of the full spike raster, each clock time step can identify the set of all neurons that can be reached via k -length paths. If $\mathcal{G}$ is a general graph, then the spike-based k -path centrality computed above will be a lower bound.

4.2 Vertex Eccentricity and Closeness Centrality

In graph theory, the eccentricity of a vertex $\epsilon(v_i)$ is the maximum length geodesic path $\mathcal{L}(v_i \rightarrow v_j)$ from v_i to any $v_j \in V(\mathcal{G})$ [16].

By setting $t_R = |E|$, the self-terminating spike sequences described in Sec. 3 can be used with single neuron driving to measure the eccentricity of a neuron $\epsilon(n_i) \in N(\mathcal{S})$, while also measuring the power and time needed to cause each neuron to fire once. Applying single neuron driving to a neuron n_i , and allowing spikes to propagate until the firing pattern self-terminates at time step R , defines then the eccentricity of the neuron as $\epsilon(n_i) = R$.

The closeness centrality of a vertex v_i is the normalized sum of all geodesic paths from v_i to all other vertices $v_j \in \mathcal{G}$ [5, 10]:

$$C_{close}(v_i) = \frac{|V(\mathcal{G})| - 1}{\sum_j d(v_i, v_j)}. \quad (1)$$

The eccentricity of n_i is computed using self-terminating spiking dynamics and ensures that all neurons in the SNS have fired. Once the spiking dynamics self-terminate the full spike raster can be decoded to extract the shortest geodesic distances between the initial driven neuron n_i and all other neurons in the SNS. After applying population decoding to the full spike raster, the summation over all clock steps, weighted by the number of neurons that fire in each clock step, can be used to compute the closeness centrality of $n_i(v_i)$.

Consider two examples: a clique graph or a path graph. In the case of $\mathcal{G} = K_N$, then $\epsilon(n_i) = 1$ for all neurons. Applying single neuron driving to any neuron in $\mathcal{S}$ will terminate after 1 time step (MCT=1). Decoding the spike raster will return 1 neuron fires at clock step 0, and $N - 1$ neurons fire in clock step 1. Thus the (normalized) closeness centrality of any neuron in $\mathcal{G} = K_N$ is:

$$C_{close}(n_i) = \frac{(N - 1)}{(N - 1)} = 1.$$

If v_i is a terminal node of $\mathcal{G} = \mathcal{P}_N$, then the spiking in $\mathcal{S}$ will terminate after N time steps (MCT=N). Decoding the spike raster will return 1 neuron fires at clock step 0, and 1 neuron fires in each subsequent clock step. The normalized closeness centrality for the terminal node is then:

$$C_{close}(n_i) = \frac{(N - 1)}{\sum_{x=0}^{N-1} x} = \frac{2}{N}.$$

These routines rely on self-terminating spike dynamics and as the graph order increases the length of time needed for the computation may increase. In general, for a graph $\mathcal{G}$, the number of time steps needed to compute the eccentricity and consequently the closeness centrality is bounded by the radius and diameter of the graph and subsequently the SNS: $r(\mathcal{G}) \leq R \leq D(\mathcal{G})$.

4.3 Local Clustering Coefficient

On a given graph, the neighborhood of a single vertex $N(v_0)$ is the induced subgraph on $v_0 \cup \{v'\}$, where $\{v'\}$ are all vertices that are connected to v_0 by an edge. Contained in the neighborhood of a vertex is the *triangle neighborhood*: The triangle neighborhood $N_{\Delta}(v_i)$ is the induced subgraph of $N(v_i)$ that contains only the neighbors and edges of v_0 which form triangles $(0, i, j)$.

The triangle neighborhood can be used to define the local clustering coefficient of a vertex. This is a measure of how close the neighborhood of a vertex is to a fully connected clique and is defined in [34] as

$$C(v_i) = \frac{\text{all triangles that contain } v_i}{\text{all tuples that contain } v_i}, \quad (2)$$

where the number of tuples that contain v_i is just all possible pairs that can be formed amongst the neighbors of v_i . For example, a vertex in a d -regular tree would have $C(v_i) = 0$ and a vertex in a clique K_d would have $C(v_i) = 1$.

Eq. 2 can be re-written in terms of the triangle neighborhood and the degree of a neuron (n_i):

$$C(n_i) = \frac{|N_\Delta(n_i)|}{\binom{d_i}{2}} \quad (3)$$

In [25], it was shown that the neighborhood of a neuron n_i with degree d_i can be extracted with static synapses and $(d+1)$ evaluations. We modify this approach to extract information about the triangle neighborhood of n_i in order to define the clustering coefficient and show that this can be done with $(d+1)$ evaluations.

The degree enumeration method of Section 4.1 is first used to identify the neighbors ($\{n_j\}$) of n_i (and compute $d_i = |\{n_j\}|$). Once the neighbors of n_i are identified, we want to restrict any future spike responses to only the set $Q = n_i \cup \{n_j\}$. We update the spike thresholds of all neurons in the SNS as $\{v_{th} = |E| \mid n_i \notin Q, v_{th} = 2 \mid n_i \in Q\}$.

The order of the triangle neighborhood $|N_\Delta(n_i)|$ is the number of triangles that contain a vertex n_i . If we define $\Delta^{(i,j)}$ as the set of all triangles that contain an edge e_{ij} then by applying $\{\chi\} = \{\chi_i^{(1)}, \chi_j^{(1)}\}$, $|\Delta^{(i,j)}|$ is found by counting the neurons n_k that fire after $t = 1$ time steps (shown in Fig. 3). This process is repeated for all d edges connected to n_i , then all triangles that contain n_i is computed from

$$|N_\Delta(n_i)| = \frac{1}{2} \sum_{j \in n_j} |\Delta^{(i,j)}|. \quad (4)$$

The factor of $1/2$ in Eq. 4 compensates for overcounting. Thus the clustering coefficient of a neuron n_i can be computed with $(d+1)$ executions: 1 to compute the denominator and d to compute the numerator of Eq. 2. The structure of the SNS does not change but the neuronal parameters need to be updated once.

Alternatively we could compute the clustering coefficient with a method that lists all triangles that contain a given vertex n_i . The order of the triangle neighborhood is computed by explicitly identifying all triangles. After the set $Q = n_i \cup \{n_j\}$ is identified, all $\binom{d_i}{2}$ pairs of neighboring vertices $(n_k, n_l) \in \{n_j\}$ are formed and stored. Then the clique test from Sec. 2.2 is applied sequentially to all tuples (n_i, n_k, n_l) .

Figure 3: For $S(N_{v_{th}=2}, W_{s_w=1})$ [Left] External stimuli applied to two neurons connected by an edge which is not contained in any triangles on the graph does not generate any internal firing in S . [Right] External stimuli applied to two neurons connected by an edge which is contained in two triangles on the graph can cause internal firing in S .

4.4 Propagation (Eigenvector) Centrality

Centrality measures are important in social network analysis for identifying influential nodes. The eigenvector centrality [3] was originally derived by considering the number of ℓ -length walks that can propagate from a given position on a graph and has been adapted for dynamical network models of epidemic spreading through social networks [18]. We consider how a similar metric can be defined via the spread of spike responses in a SNS. Since we are not computing the eigenvectors of the network's adjacency matrix, and to avoid any confusion, we call this spike-based centrality measure the "propagation centrality."

The eigenvector centrality of a vertex is dependent on the centralities of the adjacent vertices, as defined using the adjacency matrix (A_{ij}) . The propagation centrality of a neuron (n_0) is dependent on the number of internal spiking neurons that are generated when single neuron driving is applied to (n_0). The eigenvector centrality is higher for a vertex that is connected to other high-ranking vertices. The propagation centrality is high for a neuron that causes a large number of internal spiking neurons, which in turn cause a large number of internal spiking neurons.

A highly ranked neuron according to the propagation centrality will cause a large spike response in the remaining SNS neurons. If we configure the SNS for self-terminating spike sequences (see Sec. 2) then the internal spiking neurons are inherently non-backtracking. The final spike raster is decoded by counting the neurons that fire in each time window: n_0 will be the only neuron that fires in the window Δ_0 ; the secondary spiking caused by n_0 (the influence of the neuron) are the neurons that first in Δ_1 and the influence of these neurons is determined by the number of secondary spiking neurons in Δ_2 . For self-terminating spike dynamics this can be extended out to Δ_R . Using these neuron counts ($\{n_0, n_1, \dots, n_R\}$) the propagation centrality is computed as,

$$c(n_i) = \frac{\max\{n_0, n_1, \dots, n_R\}}{R} \prod_{i=0}^{i=R} n_i. \quad (5)$$

According to this measure, the highest ranking neuron will generate a large spike response in non-overlapping sets of neurons with spike dynamics that quickly terminate ($R \rightarrow 0$).

If we reduce the strictness of this condition by configuring the SNS for general walks, and allow neurons to fire multiple times, then the secondary spiking will not self-terminate. We can adapt the propagation centrality for this case by looking at only the local influence of a neuron n_0 . The local influence of a neuron (n_0) is measured by applying single neuron driving at time ($t = 0$), allowing secondary spikes to propagate through the SNS for 2 clock steps. Now the decoded spike raster has only: $\{n_0, n_1, n_2\}$ and the local influence is computed as,

$$\ell(n_i) = \prod_{i=0}^{i=2} n_i. \quad (6)$$

5 NUMERICAL RESULTS

We compare the accuracy of calculating several centrality measures through spike-based methods (simulated using Brian2) to the centrality scores computed using standard graph algorithms (computed using NetworkX). The degree centrality, vertex eccentricity and

Figure 4: The 10-vertex graph used for the numerical simulation results presented in Tables 1, 2, and 3.

closeness centralities are measured for a 10 vertex, undirected random graph (shown in Figure 4). These values are reported without any normalization in Tables 1, 2, and 3.

Table 1: Comparison of degree centrality scores computed with spiking neurons (Brian2) or standard graph algorithms (NetworkX) for a 10 vertex graph.

Vertex	Spiking Neurons	NetworkX
0	6	6
1	7	7
2	3	3
3	5	5
4	7	7
5	4	4
6	4	4
7	6	6
8	5	5
9	3	3

Table 2: Comparison of vertex eccentricity scores computed with spiking neurons (Brian2) or standard graph algorithms (NetworkX) for a 10 vertex graph.

Vertex	Spiking Neurons	NetworkX
0	2	2
1	2	2
2	3	3
3	2	2
4	2	2
5	2	2
6	2	2
7	2	2
8	2	2
9	3	3

Table 3: Comparison of closeness centrality scores computed with spiking neurons (Brian2) or standard graph algorithms (NetworkX) for a 10 vertex graph.

Vertex	Spiking Neurons	NetworkX
0	0.750	0.750
1	0.818	0.818
2	0.562	0.562
3	0.692	0.692
4	0.818	0.818
5	0.643	0.643
6	0.643	0.643
7	0.750	0.750
8	0.692	0.692
9	0.562	0.562

We also use the 10 vertex graph to compare the eigenvector centrality, the propagation centrality and the local influence score. These values are all normalized over the range $[0, 1]$ and reported in Table 2.

Table 4: Comparison of the eigenvector centrality (NetworkX) to the propagation centrality and the local influence computed with spiking neurons (Brian2) for a 10 vertex graph. The scores are all normalized to the range $[0, 1]$. Bold text used to highlight agreements in importance rankings.

Vertex	Eigenvector	Propagation	Local Influence
0	0.853	0.900	0.857
1	1.000	0.700	1.000
2	0.413	0.648	0.386
3	0.808	1.000	0.714
4	1.000	0.700	1.000
5	0.611	1.000	0.514
6	0.482	1.000	0.914
7	0.890	0.900	0.857
8	0.720	1.000	0.643
9	0.505	0.648	0.386

6 DISCUSSION

6.1 Performance

The values in Tables 1, 2, and 3 show that spike-based methods are able to obtain the correct centrality scores that standard graph algorithms compute. These quantities are computed from quantities that can be directly extracted from the graph (e.g. nearest neighbors or shortest paths).

Comparing the performance of spike-based propagation centralities to the eigenvector centrality is more complicated. We introduced the propagation centrality and local influence score in Eq. 4.4 not as a direct translation of the eigenvector centrality, but as an analogy. We are not using spiking neurons to compute the eigenvalues or identify eigenvectors of the adjacency matrix. We

are identifying high scoring vertices due to their relative connection to other high scoring vertices, but that score is based on the generation of secondary spikes in neighboring neurons.

The results presented in Table 4 are only generated on a single random graph and using these values we make some qualitative observations by comparing the relative ranking of each vertex according to the propagation centrality, local influence and the eigenvector centrality. First, there are several agreements between the local influence score and the eigenvector centrality; both assign the highest scores to vertices 1 and 4 while both assign vertex 2 the lowest score. There are also significant discrepancies in the 3 scores as well. For example the propagation centrality and local influence assign high scores to vertex 6, which has a low eigenvector centrality score. This discrepancy may be due to the dependence on non-backtracking walks in the propagation centrality, which is not present in the local influence score.

These are preliminary results on a newly proposed set of centralities measures which are purely spike-based and further study on a larger set of graphs is needed. We suggest that further study of these kinds of measures has merit. For example, one advantage of the spike-based measures is that the propagation centrality and local influence can be measured for any class of graph, whereas the eigenvector centrality cannot be found for trees.

6.2 Algorithmic Complexity

We estimate the scalability of a spike-based algorithm as the number of time steps needed to execute a program and the number of unique spiking neuron systems that have to be constructed and executed. There is an additional component in the complexity of these algorithms, which is writing and reading network information to and from the neuromorphic system. “Writing” the network corresponds to configuring the network topology and setting the appropriate parameters (synaptic weights and delays, neuronal thresholds) on the neuromorphic system. “Reading” corresponds to reading out the updated network from the neuromorphic system after activity has occurred. Reading a network is necessary only if plasticity occurs in the neuromorphic system, because plasticity will change the network’s parameters; if there is no plasticity, the network remained unchanged due to behavior. Because writing and reading networks is a non-trivial component of real execution time on neuromorphic hardware and because the way it is implemented (and thus how much time it takes) is implementation-specific, for each of the algorithms we simply specify the number of network reads and writes required to execute that algorithm.

The general scaling behavior for the routines introduced in this paper is summarized in Table 5. The maximum clock time (MCT) to execute each method listed in the third column is an upper bound on the number of clock steps or the amount of simulation time needed to simulate the spiking neurons. We use the ratio (v_{th}/s_w) of spike threshold to synaptic weight as an unique network characteristic and for routines that require multiple steps this value is listed for each step. Instantiating a set of spiking neurons, or reading the current neuron states can add significant computational overhead and are dependent on the choice of hardware. For these steps we report the number of times the current state of a system needs to

be measured (*Reads*) and the number of times a system needs to be instantiated (*Writes*).

Table 5: Summary of scaling bounds for centrality measures computed with static synapses.

Method	MCT	v_{th}/s_w	Reads	Writes
Degree Centrality	1	1	0	1
Vertex Eccentricity	N	1	0	1
Local Cluster Coefficient	$(d + 1)$	$1, 2(E)$	0	$d + 1$
Closeness Centrality	$D(\mathcal{G})$	1	0	$D(\mathcal{G})$
Propagation Centrality	$ E $	1	0	1
Local Influence	$3N$	1	0	1

7 CONCLUSIONS

The use of spiking neurons to perform tasks beyond neuroscience simulations and neural network style computation is a developing area of scientific computing. Characteristics of spiking neurons, such as the isotropic firing of neurons that lends itself to massive parallelization, make them very attractive candidates for graph analysis. We believe that by replacing matrix-based computing with spike-based computing there is the potential to speed up elements of graph analysis and in this work, we have proposed several ways in which spiking neurons can be incorporated into algorithmic primitives for graphs. Since it is likely that neuromorphic systems will be included in future heterogeneous computers, expanding their use cases for non-neural network algorithms will allow more effective utilization of these processors in the future.

Of all the neuromorphic hardware characteristics (built to efficiently execute spiking neuron models) the ability to implement synaptic plasticity poses the greatest advantage for graph algorithms. Because not all neuromorphic hardware systems implement synaptic plasticity, in this work we have focused only on static synapses. As such, though synaptic plasticity may be more difficult and energy intensive to implement in a neuromorphic system, it can reduce the computation time required for some of the tasks described here.

ACKNOWLEDGMENTS

Research sponsored in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy. 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 *Proceedings of the Third International Workshop on Post Moore’s Era Supercomputing*, Jeffery S. Vetter and Satoshi Matsuoaka (Eds.). Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN, 24–26. Issue FTGTR-2018-11.
- [2] David C Bell, John S Atkinson, and Jerry W Carlson. 1999. Centrality measures for disease transmission networks. *Social networks* 21, 1 (1999), 1–21.
- [3] Phillip Bonacich. 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of mathematical sociology* 2, 1 (1972), 113–120.

- [4] Stephen P Borgatti. 2005. Centrality and network flow. *Social networks* 27, 1 (2005), 55–71.
- [5] Stephen P Borgatti and Martin G Everett. 2006. A graph-theoretic perspective on centrality. *Social networks* 28, 4 (2006), 466–484.
- [6] Geoffrey W Burr, Robert M Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, et al. 2017. Neuromorphic computing using non-volatile memory. *Advances in Physics: X* 2, 1 (2017), 89–124.
- [7] 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.
- [8] Ajendra Dwivedi and Xinghuo Yu. 2011. A maximum-flow-based complex network approach for power system vulnerability analysis. *IEEE Transactions on Industrial Informatics* 9, 1 (2011), 81–88.
- [9] Ove Frank. 2002. Using centrality modeling in network surveys. *Social networks* 24, 4 (2002), 385–394.
- [10] Linton C Freeman. 1978. Centrality in social networks conceptual clarification. *Social networks* 1, 3 (1978), 215–239.
- [11] Pranay Goel and Bard Ermentrout. 2002. Synchrony, stability, and firing patterns in pulse-coupled oscillators. *Physica D: Nonlinear Phenomena* 163, 3–4 (2002), 191–216.
- [12] Dan Goodman and Romain Brette. 2008. Brian: A Simulator for Spiking Neural Networks in Python. *Frontiers in Neuroinformatics* 2 (2008).
- [13] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.
- [14] Kathleen E. Hamilton and Travis S. Humble. 2018. Spiking spin-glass models for label propagation and community detection. *arXiv preprint arXiv: 1801.2128201* (2018).
- [15] 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 (Knoxville, Tennessee) (NCS '17)*. ACM, New York, NY, USA, Article 9, 8 pages. <https://doi.org/10.1145/3183584.3183621>
- [16] Frank Harary. 1969. *Graph theory*. Addison-Wesley Publishing Company.
- [17] Giacomo Indiveri and Shih-Chii Liu. 2015. Memory and information processing in neuromorphic systems. *Proc. IEEE* 103, 8 (2015), 1379–1397.
- [18] Jian-Guo Liu, Jian-Hong Lin, Qiang Guo, and Tao Zhou. 2016. Locating influential nodes via dynamics-sensitive centrality. *Scientific reports* 6 (2016), 21380.
- [19] Yang Liu, Bo Wei, Yuxian Du, Fuyuan Xiao, and Yong Deng. 2016. Identifying influential spreaders by weight degree centrality in complex networks. *Chaos, Solitons & Fractals* 86 (2016), 1–7.
- [20] 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.
- [21] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. 2013. SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits* 48, 8 (2013), 1943–1953.
- [22] Marcos G Quiles, Liang Zhao, Fabricio A Breve, and Anderson Rocha. 2010. Label propagation through neuronal synchrony. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 1–8.
- [23] Marcos G Quiles, Ezequiel R Zorzal, and Elbert EN Macau. 2013. A dynamical model for community detection in complex networks. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 1–8.
- [24] Donald Stone Sade. 1989. Sociometrics of macaca mulatta III: N-path centrality in grooming networks. *Social Networks* 11, 3 (1989), 273–292.
- [25] 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.
- [26] Catherine D Schuman, Kathleen E. Hamilton, Tiffany Mintz, Md Musabbir Adnan, Bon Woong Ku, Sung-Kyu Lim, and Garrett S. Rose. 2018. Shortest Path and neighborhood subgraph extraction on a spiking memristive neuromorphic implementation. In *Proceedings of the Third International Workshop on Post Moore's Era Supercomputing*, Jeffery S. Vetter and Satoshi Matsuoka (Eds.). Future Technologies Group, Oak Ridge National Laboratory, Oak Ridge, TN, 27–29. Issue FTGTR-2018-11.
- [27] Catherine D Schuman, Raphael Pooser, Tiffany Mintz, Md Musabbir Adnan, Garrett S Rose, Bon Woong Ku, and Sung Kyu Lim. 2017. Simulating and Estimating the Behavior of a Neuromorphic Co-Processor. In *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*. ACM, 8–14.
- [28] 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).
- [29] William Severa, Rich Lehoucq, Ojas Parekh, and James B Aimone. 2018. Spiking Neural Algorithms for Markov Process Random Walk. *arXiv preprint arXiv:1805.00509* (2018).
- [30] William Severa, Ojas Parekh, Kristofer D Carlson, Conrad D James, and James B Aimone. 2016. Spiking network algorithms for scientific computing. In *Rebooting Computing (ICRC), IEEE International Conference on*. IEEE, 1–8.
- [31] Carl van Vreeswijk and Haim Sompolinsky. 1998. Chaotic balanced state in a model of cortical circuits. *Neural computation* 10, 6 (1998), 1321–1371.
- [32] Carl Van Vreeswijk, Haim Sompolinsky, et al. 1996. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 5293 (1996), 1724–1726.
- [33] Senzhang Wang, Xiaojian Zhao, Yan Chen, Zhoujun Li, Kai Zhang, and Jiali Xia. 2013. Negative influence minimizing by blocking nodes in social networks. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*.
- [34] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *nature* 393, 6684 (1998), 440–442.