Abstract—A real-world graph often has frequently interacting nodes on less frequently updated edges. Each interaction activates an existing edge and changes the activeness of the edge. In such an activation network, nodes that are cohesively connected by active edges form a cluster in both structural and temporal senses. For activation networks, incrementally maintaining a structure for an efficient clustering query processing is thus important. This raises problems on maintaining the edge activeness, combining the structural cohesiveness and activeness for clustering, and designing indexes for online clustering queries. This paper considers the time-decay scheme in modelling the activeness and proposes a suite of techniques with great effort made on simplification and innovation for efficiency, effectiveness and scalability. The query time is only related to the query results as opposed to the graph. The index size is linear up to a logarithmic factor. Extensive experiments verify the quality of the clustering results and moreover, the update time is up to six orders of magnitude faster than the baseline.

I. INTRODUCTION

A graph with nodes and edges thereon can model the interconnections among real-world objects. Various graph-based applications host frequent interactions upon a relatively stable graph. For example, on a social network, nodes represent users and edges their friendships. An interaction between two nodes of an edge can be a live chat, a message or a comment on each other’s posts. Users interact largely with existing friends especially after the new user phase in which most friends have been recommended and added in a batch. Without interactions, two users along an edge drift apart with time. Lacking interactions sometimes reflects estrangement and even hostility: with polarized political ideas, even family members may not talk to each other just to avoid conflicts. Another example is on collaboration networks where a node denotes an academic and an edge a collaboration. Establishing a collaboration with new academics takes significantly more effort than reactivating existing collaborations. For two academics with existing collaboration, a long period without a collaboration may indicate diverged research interests or shifted research environments. We abstract, from these cases, an activation network which consists of 1) a relatively stable graph called relation network (term borrowed from [30]), 2) a sequence of activations each denoting an interaction along a relation network edge and 3) the character that without an activation, the activeness of an edge decays along time.

This paper studies scalable clustering of an activation network for efficient online queries. Specifically on an activation network, we aim at efficiently querying nodes that are cohesively connected by active edges (i.e., the edges that have been activated recently) – clusters in both structural and temporal senses. Such queries are of practical importance: at a time, a user of a social network may navigate his local active community with adequate zoom ins (i.e., reporting a slightly smaller community) and zoom outs; an academic in a collaboration network may explore his active research community with different granularities. Thus, efficiently reporting such clusters on an activation network is highly desirable.

Clustering has been extensively studied on both static and dynamic/temporal graphs (see surveys [5], [30]). On a static graph, a proliferation of research [5] optimizes one structural clustering measure such as modularity, density and conductance. Note that since each measure reflects different emphasis, the comparison among approaches usually uses a basket of measures (Section 3 of [30]). When it comes to dynamic/temporal graphs, to avoid expensive recomputation of the clustering for the snapshot of each time step, existing work [30] i) models the problem by either associating each edge a duration (e.g., a time interval) or constantly focusing on the activations within a temporal window (i.e., sliding window) and ii) at each time step, generate clustering from the previous time step’s results (see [43] as an entrance).

The barrier of applying existing clustering methods to activation network is the costly maintenance of the decaying activeness. With the time-decay scheme proposed by [19] (see Figure 1(a) for a specification), the activeness of all edges decay along the time even without any activation. Such an inevitable maintenance is costly to the clustering for online queries. To reduce the cost, an optimization was proposed by [19] which keeps, for each node at any time, only the top-$k$ ($k$ is a parameter) closest neighbors as opposed to the entire neighbor set; however, it introduces approximation and needs to recompute the clusters for each time step as well.

Even if the activeness of edges can be efficiently maintained, the main challenge of clustering activation network lies in efficient online clustering query and update processing. The state-of-the-art dynamic graph clustering [43] optimizes the modularity along updates on the edge weights. However, its update cost is still too large for graphs with billions of edges and cannot handle clustering queries with different granularities. This motivates us to consider a suite of indexing problems (Figure 1(b-c)): combining the structural cohesiveness and activeness for clustering, designing indexes for clustering and maintaining the indexes for online queries.

To enable an effective indexing for clustering, a metric that integrates both structural cohesiveness and activeness
(Figure 1(b)) is highly desirable; however, finding a maintainable and effective metric is non-trivial. An interesting work Attractor [33] on clustering static graph sheds light on our research. On a static graph (initially unit weighted), Attractor iteratively applies, on each edge \(e(x, y)\), “local” rules (rules related only to \(x, y\) and their common neighbors), to update the weight of \(e\), and truncates the weight to \([0, 1]\). The iteration terminates when all the weights are polarized (either 0 or 1). The clusters are generated as the connected components of the graph after removing all the 0-weighted edges. Attractor is not scalable since it requires empirically 50 iterations to terminate and each iteration takes quadratic time\(^1\). We note that the effectiveness of Attractor lies in the propagation of the local cohesiveness across iterations.

Our observation on Attractor raises a question: On a graph where the weight of an edge \(e(x, y)\) is the inverse of a “local similarity \(S(x, y)\) that combines both local cohesiveness and the activeness” of \(x\) and \(y\), can we use “shortest distance” to carry out the propagation of the local similarity for clustering? A strong benefit of using the shortest distance as the metric is the possibility for an efficient clustering index that allows scalable construction, efficient incremental maintenance and efficient processing of online clustering queries including zoom-in and zoom-out (Figure 1(c)). However, will the shortest distance be able to carry out the propagation? We provide a positive answer to this question: for any two nodes \(u\) and \(v\), if we define \(S(u, v)\) as the inverse of the shortest distance \(\text{dist}(u, v)\), then the cohesiveness of \(u\) and \(v\) is, the harmonic mean of the local similarity on edges on \(p\) paths \(u \rightarrow v\):

\[
\frac{\max_{p: \text{paths } u \rightarrow v}}{\text{the number of edges on } p}
\]

The propagation holds since \(S(u, v)\) reduces when either the shortest path \(p\) has more hops or the harmonic mean of the local similarity on edges on \(p\) is smaller. Our experimentation verifies the effectiveness and efficiency of using shortest distance as the metric.

The activation network maintenance and the distance-based indexing have to be considered holistically. In other words, the techniques used for efficiently maintaining the activeness must be compatible to the distance-based indexing. This poses extra challenge to our work. This paper provides a suite of techniques to efficiently clustering activation networks for online queries. Our contributions are summarized as follows.

- We propose the problem of indexing activation networks for online clustering queries – clustering activation networks – based on real-world applications.
- We provide solutions to activation network clustering including techniques for the following three problems.
  - We propose a global decaying factor to efficiently maintain the decaying activeness on the activation network, the distance metric and the clustering index.
  - We propose a new way of synthesizing structural cohesiveness and edge activeness into a distance metric that is easily maintainable under the decaying activeness and supports empirically verified high quality clustering.
  - We propose a scalable distance-based indexing called pyramids. Pyramids allow efficient query processing including zoom-in and zoom-out operations for clustering queries, and graph clustering of different granularities. Pyramids can also be efficiently updated. The index time and index size are linear (up to a logarithmic factor) to the number of nodes in the graph. The query time for clustering queries and update time are bounded [28]; in other words, the query time is determined by the nodes in the query results as opposed to the entire graph, and the update time is determined by the nodes affected (Section V) by the update.
- Extensive experiments verify the efficiency, effectiveness and scalability of our solution. The update time is up to six orders of magnitude faster than the baseline.

The paper is organized as follows. Section II introduces the related work. Section III defines the problem. Section IV proposes global decay factor for maintaining the activeness, and shows the distance metric that combines both the structural cohesiveness and edge activeness. The distance metric is designed such that it can be maintained with the global decay factor as well. Section V proposes the indexing structure, the index-based clustering algorithms, and the update process. Section VI shows extensive experiments in evaluating the effectiveness and efficiency of our solution. Section VII concludes the paper.

II. RELATED WORK

Clustering Evolving Networks. Graph clustering on evolving networks [4], [15], [30], [16] has been studied in the literature.
with two types of updates: updates on the network structure, e.g., insertion/deletion of edges and vertices [17, 11] and updates on the edge weights [7, 41], [6], [19]. The edge weight in [6] corresponds to the number of interactions between two nodes while [19] updates the edge weights with a time-decay scheme to focus more on recent interactions (adopted by us).

To support zoom-in and zoom-out operations, hierarchical clustering algorithms (see survey [29]) establish the cluster hierarchy by iteratively either agglomerating clusters from bottom up or dividing clusters from top down. Each iteration greedily chooses the clusters to merge or a cluster to divide by optimizing a measure such as modularity, normalized cut, conductance, etc. However, the time-consuming optimization of each iteration is prohibitive to massive activation networks.

In clustering evolving networks, existing work (see survey [15]) focuses on updating the clusters based on the change in the underlying graph to avoid recomputing the clustering from scratch at each time step. With the time-decay scheme, all the activeness decays along the time even without interactions; hence, the algorithm of [19] recomputes the clustering upon each query and optimizes the clustering computation by maintaining a derived graph (summary) of the top-k closest neighbors of each node to perform clustering.

**Distances and Graph Clustering** Dynamic processes are engaged in measuring the similarity among nodes in a static unweighted graph. The dynamic processes include random walk [26], diffusion [42], and synchronization [13]. Pair-wise distances are relatively expensive to compute especially on massive graphs. A recent work [33] (will be elaborated in Section IV) iteratively updates the edge weights until they become binary (connect/disconnect) and then uses connected components as clusters. As far as we know, our solution is the first work in using shortest distance for graph clustering.

**Index Distances on Massive Graphs** Indexing distances exactly for small-world networks (PLL [8], the state-of-the-art) and road networks [24] adopts different approaches; updating these indexes on activation networks is expensive due to the collective updates on the activeness along the time. For example, the index time and index size of PLL [8] are bottlenecks on static massive graphs [21], let alone the update.

Approximate distance indexes that provide approximate guarantee [14], [32], [25] follow the seminal work of Thorup and Zwick [36]. This line of research reports approximate distance $d(u, v)$ of two nodes $u$ and $v$ with guarantee $d(u, v) \leq d(u, v) \leq rd(u, v)$ where $r \geq 1$ is a real number called the stretch. Trade-offs are made among the size of the index, the stretch, and the online query time. In the original work of [36], for any positive value $k$, an index with size $O(kn^{1+1/k})$ can provide stretch $2k - 1$ and query time $O(k)$; later work [14] improves trade-off to query time $O(1)$ and size $O(n^{1+1/k})$. A large constant factor in indexing is often hidden in this line of work; one exception is the structure proposed by Das Shama [32] with space $O(n \log(n))$, stretch $2\log(n) - 1$ and query time $O(\log(n))$. The structure can be easily parallelized/distributed in construction and query processing [31]. We adopt this structure for distance indexing.

Online computation, as a complement to the exact indexes and approximate indexes, can reduce the index size [9] and improve the precision [37], [27].

**III. Problem Formulation**

Let $G(V, E)$ be an undirected unweighted graph with the vertex set $V$ and edge set $E$. Denote by $n$ and $m$ the cardinalities of $V$ and $E$, respectively. For a node $v \in V$, denote by $N(v)$ the set of neighbors of $v$, i.e., $N(v) = \{u(u, v) \in E\}$, and by $\text{deg}(v) = |N(v)|$ the degree of $v$. For any function $f : E \rightarrow \mathbb{R}$ that maps the edges of $G$ to real numbers, the $f$-based distance $\text{dist}_f(u, v)$ between two nodes $u, v \in V$ is the length of the shortest path under $f$ between $u$ and $v$.

An activation is a pair $(e, t')$ of an edge $e \in E$ and a timestamp $t'$. An **activation stream** is an unlimited sequence of activations that arrive in the system sequentially in the following form $A_1, A_2, \ldots, A_i, \ldots$. Interaction $A_i = (e_i, t_i)$ arrives at time $t_i > 0$, $\forall i > 0$. Denote by $t$ the current time.

An **activation network** is a graph with an activation stream. We use the time-decay scheme to model that the impact of an activation on an edge decays exponentially along the time.

**Time-decay Scheme [19]. Decay factor** $\lambda$ is a non-negative parameter. The **activeness** at time $t$ is a function on $E$: 

$$a_t(e) = \sum_{A_i = (e_i, t_i) \leq t} e^{-\lambda(t - t_i)}, \forall e \in E. \quad (1)$$

**Example 1:** Let $\lambda = 0.1$. For edge $e(v_8, v_{11})$ in Figure 2(a), consider an interaction stream with $A_1 = (e, 0)$ and $A_2 = (e, 2)$. $a_0(e) = 1$. At time $t = 1$, $a_1(e) = 1 \times e^{-0.1 \times (1 - 0)} = 0.905$. At time $t = 2$, $a_2(e) = 1 \times e^{-0.1 \times (2 - 0)} + 1 \times e^{-0.1 \times (2 - 2)} = 1.818$.

**Problem 1 (Clustering Activation Networks):** Given an activation network $G(V, E)$ with an activation stream and a decay factor $\lambda$, answer the following clustering queries at any time $t$ considering both the activeness defined by the time-decay scheme and the structural cohesiveness:

1) **Report all clusters.** Report clustering results such that the total number of clusters is $\Theta(\sqrt{n})$ and based on which repetitive operations of

- Zoom in: report finer grained clustering results.
- Zoom out: report coarser grained clustering results.

The total number of different granularities is $O(\log_2 n)$.

2) **Report Local Clusters.** Given a query node $v$, report

- The smallest cluster that contains $v$, and then allow repetitive zoom-out operations.
- The cluster that contains $v$ in the granularity where the total number of clusters is $\Theta(\sqrt{n})$ and then allows zoom-in and zoom-out operations.

Observing that the activeness is a function on the current time $t$, constant maintenance is needed when $t$ increases which may trigger a high cost.

**IV. Establish Distance Metric**

This section proposes a global decay factor for maintaining the activeness, then shows the distance metric that i) combines both the structural cohesiveness and edge activeness, and ii) is maintainable under the global decay factor.
A. Global Decay Factor

The time-decay scheme constantly decays the activeness of all edges; however, we made the following observation.

**Observation 1:** The activeness of all unactivated edges decays at the same pace. Specifically, for any edge $e$ that has not been activated during time span $[t', t'']$, Equation 1 guarantees that $a_{e'}(t) = a_e(t) \times e^{-\lambda (t' - t)}$. The factor $e^{-\lambda (t' - t)}$ is edge independent, i.e., does not depend on $e$.

**Definition 1 (Global Decay Factor):** Let $t^*$ be a timestamp no larger than $t$ - the current time. Project the activeness of all edges at time $t$ to timestamp $t^*$ to generate anchored activeness $a^*_t(e) = a_t(e)/g(t, t^*)$ where $g(t, t^*) = e^{-\lambda (t^* - t)}$ is called the global decay factor and $t^*$ the anchor time.

For any $e$ edge that has not been activated during $[t', t]$, the activeness $a_t^*(e)$ of $e$ at time $t$ is $a_t^*(e) \times g(t, t^*)$. Thus, $a_t^*(e) = a_t(e)$ does not have to be updated along the time. Upon receiving an activation on edge $e$ at time $t$, the increase of $a_t(e)$ by $\Delta a_t^*(e) = 1$ (Equation 1) incurs the increase of $a_t^*(e)$ by $\Delta e^{-\lambda (t^* - t)} = 1/g(t, t^*)$ to uphold $a_t^*(e) = a_t^*(e) \times g(t, t^*)$ on all edges. Therefore, the anchored activeness $a_t^*(e)$ only updates upon the arrival of an activation on edge $e$. Besides, when a fixed number of activations accumulates, we let all anchored activeness absorb the global decay factor ($\times g(t, t^*)$) and update the anchor time ($t^* \leftarrow t$), which is called a **batched rescale**. The anchor time $t^*$ is initially 0 and updated periodically in batched rescale. The update cost of batched rescale can be amortized to the arrived activations that triggered the rescale.

**Lemma 1:** With global decay factor, the activeness can be maintained at a cost linear to the number of arrived activations. All the proofs can be found in our technical report [3].

In the following, we use $[a_t, g(t, t^*)]$ to denote the activeness kept under a global decay factor: we are essentially keeping the anchored activeness $a_t^* = g(t, t^*)^{-1}$ for each edge.

**Example 2:** Consider edge $e_{v_9, v_11}$ of Figure 2(a) and the activation stream in Example 1. $t^*$ is initially and $a_0(e) = 1$. At time $t = 1$, $g(t, t^*)$ becomes $e^{-0.1 \times (1 - 0)} = 0.905$ and the edge weight $a_1^2(e) = a_0^2(e) \times g(1, 0)$ where $a_1(e) = a_0(e) = 1$. At time $t = 2$, $g(t, t^*)$ becomes 0.819 and then we derive $a_2^2(e) = a_1^2(e) + 1/g(t, t^*) = 2.221$. It can be verified that $a_2(e) = 1.819 = a_2^2(e) \times g(2, 0)$. If we update the anchor time at $t = 2$, then $t^* = 2$ and $a_2^* (e) = a_2(e) = 1.819$.

Derived functions of the activeness can be similarly maintained using the global decay factor.

**Definition 2 (Maintainable under the global decay factor):** Let $F_t$ be a derived function of $W_t$, i.e., $F_t(e)$ of any edge $e \in E$ is a function $f_t^e((a_t^*(e')) \in E)$. We say $F_t$ is

- PosM positively maintainable under the global decay factor if $F_t(e) = f_t^e((a_t^*(e')) \in E) \times g(t, t^*)$,
- NegM negatively maintainable under the global decay factor if $F_t(e) = f_t^e((a_t^*(e')) \in E)$,
- NeuM neutrally maintainable under the global decay factor if $F_t(e) = f_t^e((a_t^*(e')) \in E)$.

**Lemma 2:** Function $F_t$ is PosM if it is a linear combination, without constant term, of 1) $\{a_t(e') \in E\}$, or 2) functions that are PosM. Function $F_t$ is NegM if 1) it is the inverse of a PosM function, or 2) a linear combination, without constant term, of functions that are NegM. Clearly $a_t(e)$ is PosM. Lemma 2 will be used to quickly verify if a function is maintainable under the global decay factor.

B. Local Reinforcement

This section first designs a similarity function $S_t$ and then derives the distance metric. We start with examining Attractor [33], an algorithm on a closely related problem: clustering a static unweighted graph by iteratively updating edge weights based on the local graph topology.

Attractor clusters a static unweighted graph by iteratively (engaging up to 50 repetitions) incorporating the local structural cohesion into the edge weights (initially 1). Whenever the updated weight exceeds 1 (falls below 0, resp.), it truncates (rounds, resp.) the weight to 1 (0, resp.), until all edge weights become binary (0 or 1). Clusters are generated as the connected components with only 0-weight edges. The effectiveness of Attractor lies in propagating the local structural cohesion via **iteratively** updating the edge weights. The repetition number, 3 to 50 [33], is acceptable for offline computation but not for online computation: waiting for a 0-1 separation over all edge weights is impractical. This motivates us to use the shortest distance, instead of multiple repetitions, to propagate the local structural cohesion for clustering.

**Active similarity.** To reduce the number of repetitions, we first derive active similarity from Jaccard similarity to combine the structural correlation of two nodes $u$ and $v$ of an edge with its activeness. For $\forall (u, v) \in E$, its active similarity is

$$
\sigma(u, v) = \frac{\sum_{x \in N(u) \land N(v)} (a_t(u, x) + a_t(v, x))}{\sum_{x \in N(u)} a_t(u, x) + \sum_{x \in N(v)} a_t(v, x)}
$$

Fig. 2: Index a Graph with 13 nodes with $\mathcal{P}$ of 2 **pyramids**, each has $[\log_2(13)] = 4$ levels of granularities.
Note that the common friends of $u$ and $v$ that have active edges with $u$ and $v$ (collectively) will boost the similarity more than the common friends that are not so active with $u$ and $v$. Exclusive friends of $u$ and $v$ will reduce the similarity.

Jaccard similarity [35] is adopted for initializing the active similarity before applying the local reinforcement because i) it is widely used in clustering, e.g., SCAN [39] and Attractor [33], ii) local and inexpensive to compute from the activeness function, and most importantly iii) it is NeuM (Definition 2) – the global decay factor can all be cancelled. Property iii) is important since the initialized structural similarity should be irrelevant to the time decay function and furthermore, this enables the structural similarity to be embedded in the local reinforcement without spoiling the maintainability of $\mathcal{F}_t$.

**Active Neighbor Set.** Given the active similarity, we define the active neighbor set based on a threshold $\epsilon$.

$$N_\epsilon(v) = \{u \in N(v) | \sigma(u,v) \geq \epsilon\}.$$  

Because the active similarity is essentially a scale (the global factor term $\beta(t,t')$ can be cancelled), we have Lemma 3.

**Lemma 3:** Consider the anchored activeness $a_\epsilon^* t$ and its corresponding active similarity and active neighbor set $N_\epsilon^*(v)$ for each node $v \in V$, we have $N_\epsilon^*(v) = N_\epsilon(v)$.

**Core Node.** Based on the size of the active neighbor set, we define core nodes. Given a parameter $\mu$, a node $v \in V$ is a core if it has at least $\mu$ active neighbors, $|N_\epsilon(v)| \geq \mu$. Two node types that are related to core can be derived: a node is a p-core if it is not a core but has the potential to be a core – its degree is less than $\mu$. A node is a periphery if it can never be a core – its degree is less than $\mu$. Note that the three types of nodes, core, p-core, and periphery, disjointly partition the vertex set $V$, i.e., a node belongs to exactly one type.

**Local Reinforcement.** Based on the above concepts, we introduce a local reinforcement process in encoding both the cohesiveness and activeness to one metric. Right before an activation $A = (e,t)$ with edge $e(u,v)$ arrives at the system at time $t$, assume that there is a function $\mathcal{F}_t : E \rightarrow \mathbb{R}$ on the edge set. We call $e$ the trigger edge and $u, v$ the trigger nodes. The local reinforcement updates $\mathcal{F}_t$ to propagate the structural coherence based on the activation. Specifically, we define 3 different processes called direct consolidation, triadic consolidation and wedge stretch, for edge $e$ and one of its trigger nodes. In the following, we only describe the processes for the trigger node $u$ while the processes for $v$ are symmetric.

- **Direct consolidation.** The activation on $e$ consolidates the relationship between $u$ and $v$ by a factor proportional to the active similarity; moreover, the more neighbors $u$ or $v$ has, the less influence an interaction has on $u$ or $v$.

  $$A_{\mathcal{F}}(e) = \mathcal{F}_t(e) \frac{\sigma_{\mathcal{F}}(u,v)}{\deg(u)}.$$  

- **Triadic consolidation.** For common neighbors of $u$ and $v$, the consistency between the structural coherence and the activation reinforces the similarity.

  $$T_{\mathcal{F}}(e) = \sum_{w \in N(u) \cap N(v)} \sqrt{\mathcal{F}_t(u,w)\mathcal{F}_t(v,w)} \frac{\sigma_{\mathcal{F}}(w,u)}{\deg(u)}.$$  

- **Wedge stretch.** Exclusive friends of $u$ or $v$ introduce divergence, which may lead to a less impactful activation.

  $$WS_{\mathcal{F}}(e) = \sum_{w \in N(u) \cup N(v)} \mathcal{F}_t(w,u) \frac{\sigma_{\mathcal{F}}(w,u)}{\deg(u)}$$

With these three processes, we update $\mathcal{F}_t(e)$ with three formulations based on the node property (i.e., core, periphery or p-core) of the trigger node $u$. Specifically, if $u$ is a

- **Core:** The high number of active neighbors indicates that a core may lead a community and thus attract neighboring nodes in forming a cluster. For $e(u,v)$, we apply direct and triadic consolidation to update $\mathcal{F}_t(e)$:

  $$\mathcal{F}_t(e) \leftarrow \mathcal{F}_t(e) + A_{\mathcal{F}}(e,t) + T_{\mathcal{F}}(e,t).$$  

- **Periphery:** The weak connections of a periphery imply that it is more likely to be grouped into a cluster as a follower, but it lacks the ability of attracting its neighbors and may be distracted by exclusive neighbors. Therefore, we apply wedge stretch to update $\mathcal{F}_t(e)$:

  $$\mathcal{F}_t(e) \leftarrow \mathcal{F}_t(e) - WS_{\mathcal{F}}(e).$$

- **P-core:** In the situation between a core and a periphery, a p-core has the ability to consolidate the similarity through direct and triadic consolidations while it also follows its exclusive neighbors and diminishes the relationship with $v$ through wedge stretch. We update $\mathcal{F}_t(e)$ with

  $$\mathcal{F}_t(e) \leftarrow \mathcal{F}_t(e) + A_{\mathcal{F}}(e,t) + T_{\mathcal{F}}(e,t) - WS_{\mathcal{F}}(e,t).$$

We update $\mathcal{F}_t(e)$ for trigger node $v$ symmetrically as well. This completes the local reinforcement of $\mathcal{F}_t(e)$ upon an activation. Note that here the function $\mathcal{F}_t$ can be any function, we plug the similarity function in to derive the distance metric.

**C. Distance Metric**

Given the local reinforcement process we have described, we define the similarity function $S_t$ on $E$ in both initializing $S_0 (t = 0)$ and updating $S_t (t > 0)$ along the time. The distance metric is defined upon the similarity function.

**Initialize $S_0$**. Note that when $t = 0$, no activation has arrived at the system, finding the similarity function $S_0$ serves the purpose of clustering a static graph. In other words, the similarity function should purely reflect the structural coherences of the graph. We start by setting $S_0 = 1, \forall e \in E$ first and then applying local reinforcement with a stream of activations. The activations are generated in the following way. i) Determine the number of repetitions rep. ii) The stream is initialized with activations over all edges in $E$ (in arbitrary order) – the timestamps are all $t = 0$. iii) For each repetition, append the stream with activations over all edges in $E$ – the timestamps are all $t = 0$. As shall be revealed in the experiment, 7 repetitions are enough for a high quality clustering while 0 repetition is enough for beating the baselines.

**Update the similarity function $S_t$ on activation stream.** For an edge $e \in E$, without activations, the similarity $S_t(e)$ decays at the same ratio $\lambda$ as the edge weight $a_{\mathcal{F}}(e)$; upon the arrival of an activation $(e,t)$ at time $t$, local reinforcement with trigger edge $e$ is applied to $S_t$. 


Lemma 4: If a similarity function is PosM (Definition 2, after applying the local reinforcement, the similarity function is still PosM).

Therefore, similar to the activeness (Lemma 1), the cost for maintaining a similarity function can be amortized to each arrived activation and we thus have Lemma 5.

Lemma 5: The cost of maintaining $S_t$ upon receiving an activation $e(u, v)$ is $O(|N(u)| + |N(v)|)$ involving only neighbors of $u$ and $v$.

$S_0$ can be used for clustering a static graph and thus can be used to verify the effectiveness of the local reinforcement in clustering static graphs with ground truth.

**Distance Metric.** Upon the similarity function on the edge set $E$, a distance metric based on shortest distance can be derived to propagate the structural coherence computed in $S_t$. Engaging in the shortest distance reduces the number of repetitions required for refining $S_t$. Specifically, at time $t$, given a similarity function $S_t$ and its reverse function $S_t^{-1}(e) = \frac{1}{\Theta(s)}$, $\forall e \in E$, define a distance metric $d : V \times V \rightarrow \mathbb{R}$ as the pairwise shortest distance, with weight $S_t^{-1}$ on graph $G$. In the following discussions on shortest distance, unless otherwise specified, the default edge weight is $S_t^{-1}$.

Consider two nodes $u$ and $v$, their attraction strength is defined as $\max_{p: \text{path between } u \text{ and } v} \frac{1}{\sum_{e \in p} S_t^{-1}(e)} = \frac{1}{\text{dist}(u, v)}$.

The attraction strength is the maximum harmonic mean, among all the paths from $u$ to $v$, of the active similarities divided by the number of hops on the path. It justifies the propagation hidden in the distance computation.

Lemma 6: The distance metric is NegM (Definition 2).

**V. DISTANCE INDEXING**

As discussed in Section II, it is expensive for massive graphs such as social networks to compute $M_t$ in $O(mn)$ time, building exact distance index has strong limitations in index time, index size and maintenance.

**A. Index (Pyramids) Construction**

Our index $P$, called pyramids, adopts, as the base structure, an oracle proposed by Sarma et al., for approximate distance indexing on web-scale graphs [32] (see Section II for details). We first construct $P$ and then show how our algorithms use $P$ to generate clusters and maintain $P$ upon activations.

A building block of $P$ is to construct, given an integer $l \in [1, \lceil \log_2(n) \rceil]$ called the granularity level, a Voronoi partition.

Seed set. Select a set $S$ of $2^l$ nodes without duplication from the node set $V$ of the graph uniformly at random. In other words, for any $S' \subseteq V$ with size $2^l$, the probability of $Pr(S = S') = \frac{1}{(2^l)}$.

Voronoi Partition. For each node $v \in V$, find the node $S[v] \in S$ that is closest to $v$. $S[v]$ is called the seed of $v$ in $S$. One can compute $S[v]$ for all $v \in V$ using one Dijkstra with set $S$ as the super source node. $2^l$ disjoint partitions can be obtained by grouping the nodes in $V$ by their seeds in $S$, each partition has a seed node. Store the shortest path tree from a seed node to all the nodes in the corresponding partition, a by-product of Dijkstra.

An index $P$ consists of a constant number $k$ (typically 4) of pyramids, $P_1, P_2, \ldots, P_k$, where each pyramid is a suite of $[\log_2(n)]$ Voronoi partitions with granularity level, respectively, $1, 2, \ldots, [\log_2(n)]$.

Lemma 7: [32] The construction time of an index $P$ is $O(n \log^2(n) + m \log(n))$ and the space is $O(n \log_2(n))$.

**Example 3:** In Figure 2, (a) shows an example of a graph with $n = 13$ nodes and (b), (c) are the $k = 2$ pyramids of its index $P$. For each pyramid, we show the Voronoi partition of the first 3 granularity levels as (d)-(f) (resp. (g)-(i)). Each pyramid consists of $[\log_2(13)] = 4$ granularity levels. From level 1 to level 3, $2^{l-1}, 2^{l-1}$ and $2^{l-1}$ nodes are selected as seeds (in gray color) respectively and each seed is the root of a shortest path tree. For example, on level 1 shown in (d) of pyramid (b), the only seed $v_1$ is the root node of the shortest path tree containing all the 13 nodes of the graph. On level 2 shown in (e) of pyramid (b), each node of the graph exclusively belongs to a partition of a seed node $v_4$ or $v_7$.

**B. Clustering with Pyramids**

Each Voronoi structure provides a center-based clustering: seed node $s$ attracts the surrounding nodes $v$ with strength defined in Section IV-C and dominates $v$ if this attraction is the strongest among all seeds. The seed with the smallest shortest distance to $v$ has the largest similarity to $v$.

Given the random nature of the seed selection, we engage multiple pyramids as a voting system to stabilize the clustering. Specifically, given a support threshold $\theta$ which is normally set to 0.7, two nodes $u, v$ are in the same cluster at granularity level $l$ if $u$ and $v$ are dominated by the same seed node in $\theta k$ or more pyramids of $P$. Formally, we define a voting function $H_l$ for granularity $l$ as $H_l(u, v) = \begin{cases} 1, & \text{if at least } \theta k \text{ pyramids (in } P \text{) have seed set } S_l \text{ with } S_l[u] = S_l[v] \text{ at level } l; \\ 0, & \text{otherwise.} \end{cases}$

**Example 4:** The $k = 2$ pyramids in Figure 2 (b) and (c) can be used as a voting system for clustering. Let the support threshold $\theta$ be 0.7 and take two connected nodes $v_4$ and $v_7$ as an example. At level $l = 2$, both pyramids put $v_4$ and $v_7$ under the partition of the same seed node, then $H_l(v_4, v_7) = 1$ as $2 \geq 2 \times 0.7 = 1.4$. At level $l = 3$, $v_4$ and $v_7$ are under the partition of the same seed node in pyramid (c) while in different seed nodes’ partitions in pyramid (b), then $H_l(v_4, v_7) = 0$ as $1 < 1.4$. Thus, the voting system votes $v_4$ and $v_7$ to be in the same cluster at level 2 but not in the same cluster at level 3.

With the voting function, we remove all edges in $G$ whose voting result is 0 and leave the edges with result 1. Then consider two ways of clustering.

1) **Even Clustering.** Report the connected components of the remaining graph.

2) **Power Clustering.** Set a direction to each edge that heads from high degree node to low degree node (use node id
to break ties) and label all nodes as “unclustered”. Search from the high degree node to low degree node: for each unclustered node \( v \), find all unclustered nodes that are reachable from \( v \) and put them into a single cluster.

The drawback of the even clustering is the amplification of any unexpected error in the voting results: a cluster can be over-expanded due to any mis-clustering of two nodes of an edge. This problem can be avoided by power clustering.

**Lemma 8:** The complexity of both even clustering and power clustering in clustering a graph is \( O(m \log(n)) \).

Note that both even clustering and power clustering are search based, we thus have Lemma 9.

**Lemma 9:** Given a query node \( v \), the cluster at a granularity level \( l \) can be obtained by the time proportional to the size of the neighbors of the reported nodes.

Zoom-in and zoom-out operations can be supported by adjusting the granularity level \( l \).

**Example 5:** With the same voting system as Example 4 (Figure 2), edges \( e(v_1, v_2) \), \( e(v_1, v_3) \), \( e(v_4, v_5) \), \( e(v_5, v_6) \), \( e(v_6, v_7) \), \( e(v_6, v_10) \), \( e(v_7, v_12) \) and \( e(v_8, v_11) \) are voted to be within the same cluster at level 3. Power clustering ranks all the nodes based on degree and breaks ties with node id: \( v_6, v_1, v_4, \cdots, v_12, v_{13}, v_7 \). Search from \( v_6 \), reach unclustered nodes \( v_5, v_9, v_{10} \) and produce cluster \( \{v_6, v_5, v_9, v_{10}\} \). A search from \( v_1 \) reaches unclustered nodes \( v_2 \) and \( v_3 \) and produce cluster \( \{v_1, v_2, v_3\} \), \( \{v_4, v_{11}\} \), \( \{v_8, v_{11}, v_{12}\} \) and \( \{v_7\} \), are found.

**C. Efficient Update.**

According to Lemma 2, the global decay factor is applicable to \( S_t \), \( M_t \), as well as \( \mathcal{P} \). The global decay factor allows the update cost of \( \alpha_t \) to be \( O(1) \) per activation and that of \( S_t \) on edge \( e(u, v) \) to be \( O(deg(u) + deg(v)) \).

**Lemma 10:** The global decay factor for \( S_t^{-1} \), \( M_t \) and \( \mathcal{P} \) is \( g^{-1}(t, t^*) \) where \( t^* \) is the anchor time.

Upon the arrival of an activation \( e(u, v) \), the update of \( \mathcal{P} \) boils down to two steps:

1. Update \( S_t(e) \) in \( O(deg(u) + deg(v)) \) time;
2. Update the edge weight of \( e \) to \( S_t^{-1}(e) \) in the Voronoi partition of each level of each pyramid of \( \mathcal{P} \).

Algorithms 1-3 update an edge weight \( e \) in a Voronoi partition. The essence of these algorithms is to search, from the two ends of the edge \( e \), the nodes whose distance to their closest seed nodes are subject to change (affected). Algorithm 1 handles the case when \( S_t^{-1}(e) \) decreases during the update and Algorithm 3 handles an increasing \( S_t^{-1}(e) \). Both algorithms store the impacted nodes together with their upper bound distances to the seed nodes in a queue, and terminate when the queue becomes empty.

A decreasing \( S_t^{-1}(e) \) can only incur a reduction of the distances of other nodes to their seeds (proved formally for Lemma 12). The decrease of a node \( x \) to its seed node must be triggered by the decrease of a neighbor of \( x \) to the corresponding seed node. Thus, we pass to Algorithm 2 a potential “neighbor who may cause the update” (the input node named \( b \)) and use Algorithm 2 to detect/update the infected nodes for Algorithm 1. Algorithm 1 firstly updates the reverse similarity on \( e \) in Line 1. Line 2 initializes the priority queue with an empty queue. The queue stores triples, each triple includes i) the upper bound distance from a node \( z \) to its closest seed node, ii) the node \( z \), and iii) the current seed node \( S[z] \) of \( z \). Line 3 tests if its distance upper bound to its seed node is updated in the recalculation. \( u \) and \( v \) are pushed into the queue after the probing test (Line 3-4). As long as the queue is not empty (Line 5), the node \( x \) whose triple has the minimum distance upper bound in the queue will be popped (Line 6). If \( x \) can update the distance upper bound of any of its neighbors (Line 8), its corresponding neighbors will be enqueued.

When \( w^{-1}(e(u, v)) \) increases, if the edge \( e \) is not on the shortest path tree before the update, then nothing will be impacted by the update. If \( v \) (or \( u \)) is the parent of \( a \) (or \( v \), resp.) in the shortest path tree before update, then all the affected nodes must be in the shortest path tree rooted at \( u \) (or \( v \), resp.) (see the proof in Lemma 12 in the technical report). Algorithm 3 takes an edge \( e(u, v) \) with increasing weight \( w^{-1} \). It firstly updates the reverse similarity on \( e \) in Line 1 then decides the impacted subtree \( T_u \) in Line 2-5. Line 6 initializes a priority queue \( Q \). In Line 7-9, both distances and seeds of the nodes \( x \) in \( T_u \) are reset to a node \( -1 \) with infinite distance. Besides, the nodes that are adjacent to \( x \) but not in \( T_u \) are pushed to \( Q \). In Line 10-14, Dijkstra’s-like procedures are applied to reconstruct the shortest path tree(s).

The following lemma shows that by conducting the above bounded search [28], all the nodes with updated distances/seed nodes will be pushed into the queue:

Let \( e(u, v) \) be an edge whose \( w^{-1}(u, v) \) is updated. Consider a Voronoi partition with seed set \( S \). Denote by \( S[x] \) the seed of each node \( x \) in \( V \) in the graph \( G \) under \( S_t^{-1} \) before the update of the weight of \( e(u, v) \) and \( S'[x] \) the seed of each node

---

**Algorithm 1: Update-Decrease**

**Input:** Edge \( e(u, v) \in E \), similarity \( w \) on \( e \) to update, Voronoi partition of the level of interest in \( \mathcal{P} \)

**Output:** Updated Voronoi partition

1. \( S_t^{-1}(e) \leftarrow w^{-1} \);
2. Initialize a priority queue \( Q \leftarrow \emptyset \);
3. if \( \text{Probe}(u, \mathcal{P}) \) then \( Q.push(\text{dist}(S[u], u), u, S[u]) \);
4. if \( \text{Probe}(v, \mathcal{P}) \) then \( Q.push(\text{dist}(S[v], v), v, S[v]) \);
5. while \( Q \) is not empty do
6. \( \langle \text{dist}(S[x], x), x, S[x] \rangle \leftarrow Q.pop() \);
7. for each \( y \in N(x) \) do
8. if \( \text{Probe}(y, x, \mathcal{P}) \) then \( Q.push(\text{dist}(S[y], y), y, S[y]) \);
9. return the updated Voronoi partition;

**Algorithm 2: Probe**

**Input:** Two nodes \( a, b \in V \) with \( (a, b) \in E \), the shortest path tree of the Voronoi partitions at the granularity of interest in \( \mathcal{P} \)

**Output:** Return true if \( \text{dist}(S[a], a) \) or \( S[a] \) changed triggered by \( b \), return false otherwise

1. \( o \leftarrow S[b] \);
2. \( d_{o,a} \leftarrow \text{dist}(S[b], b) + S^{-1}(a, b) \);
3. if \( \text{dist}(S[a], a) > d_{o,a} \) then
4. \( S[a] \leftarrow x \);
5. \( \text{dist}(S[a], a) \leftarrow d_{o,a} \);
6. return true;
7. return false;
Algorithm 3: Update-Increase

Input: Edge \( e(u, v) \in E \), similarity \( w \) on \( e \) to update, Voronoi partition of the level of interest in \( \mathcal{P} \)

Output: Updated Voronoi partition

1. \( S'^{-1}_t(v) \leftarrow w^{-1} \);
2. \( o \leftarrow 1 \);
3. if \( e(u, v) \) and \( v \) are in shortest path \( T_u \) rooted at \( u \) then \( o \leftarrow u \);
4. else if \( e(u, v) \) and \( u \) are in shortest path \( T_v \) rooted at \( v \) then \( o \leftarrow v \);
5. else return the updated Voronoi partition;
6. Initialize a priority queue \( Q \leftarrow \emptyset \);
7. foreach \( x \in \text{shortest tree } T_o \) rooted at \( o \) do
   8. \( \text{dist}(S[x], x) \leftarrow \infty, S[x] \leftarrow -1 \);
9. foreach \( y \in N(x) \setminus T_o \) then \( Q.push(dist(S[y], y), y, S[y]); \)
10. while \( Q \) is not empty do
    11. if \( \text{dist}(S[x], x), x, S[x]) \leftarrow Q.pop(); \)
    12. for each \( y \in N(x) \) do
        13. if \( \text{Probe}(v, y, \mathcal{P}) \) then \( Q.push(dist(S[y], y), y, S[y]); \)
14. return the updated Voronoi partition;

Fig. 3: Updates on a Voronoi Partition

in the graph under \( S'^{-1}_t \) right after the update. Let \( U \subseteq V \) be the set of nodes \( x \) whose distance \( \text{dist}(x, S[x]) \) under \( S'^{-1}_t \) is different from \( \text{dist}(x, S'[x]) \) under \( S'^{-1}_t \) or \( S[x] \neq S'[x] \). Let \( U' \leftarrow U \cup \{u, v\} \), we have Lemma 11 and Lemma 12.

Lemma 11 (Local Update): The graph \( G'_U(U', E \cap U' \times U') \) is connected.

Lemma 12 (Complexity of Algorithm 1 and 3): The costs of both Algorithm 1 and Algorithm 3 are \( O(\sum_{x \in U'} \deg(x)) \) up to a logarithmic factor.

Example 6: Figure 3(a-e) shows 5 update examples on the Voronoi partition of Figure 2(e). In (a), the weight under \( S'^{-1}_t \) of \( e(v_2, v_3) \) is decreased by 1. Update-Decrease first pushes \( v_5 \) and \( v_6 \) into \( Q \) after updating the edge weight and then runs in iterations. Repetition 1, \( v_5 \) is popped and probed. Its neighbors \( v_3, v_7 \) cannot pass the Probe checking even; \( v_6 \) is reinserted to updated (due to different implementation of the priority queue) in \( Q \). Iteration 2, \( v_6 \) is popped out, updated by Probe and its neighbors \( v_3, v_9, v_10 \) pass Probe checking and are pushed into \( Q \). Iteration 3, \( v_9 \) is popped and updated but none of its neighbors, \( v_4 \) and \( v_9 \), shall be pushed into \( Q \). Iteration 4, \( v_9 \) is popped and updated but no new nodes can have better distances triggered by \( v_9 \) and therefore \( Q \) becomes empty, the algorithm terminates. In (b), the weight of \( e(v_1, v_3) \) is increased by 1. Only \( v_3 \) is changed. In (c), the weight of \( e(v_4, v_7) \) is increased by 1. In this case, only \( v_3 \) is changed. In (d), the weight of \( e(v_8, v_1) \) is increased by 5. Then seed \( v_4 \) is closer to \( v_7 \) than seed \( v_8 \). In (e), the weight of \( e(v_8, v_7) \) is decreased by 8, then seed \( v_8 \) now is closer to \( v_7 \) than seed \( v_4 \).

Remarks. Due to the “local” feature of the update, we can maintain a voting count (among Pyramids) for each level, each edge in real time. This allows us to report changes on user specified nodes at a cost equal to the reporting. Due to the space limit, we leave the detailed algorithm in the technical report of the paper [3]. It is also worth mentioning that the \( \log_2(n) \times k \) Voronoi partitions in \( \mathcal{P} \) are mutually independent in storage, update and query processing.

Lemma 13: The update of \( \mathcal{P} \) is embarrassingly parallel and can be deployed to achieve a speedup up to \( \log_2(n) \times k \).

VI. EXPERIMENTS

This section evaluates the efficiency and effectiveness of the proposed approach on various real-world networks.

Our Methods. We proposed the following 3 versions of our Activation Network Clustering (ANC) method supported by the update algorithm UPDATE (Section V) and clustering algorithm DirectedCluster (Section V).

1) ANCF: An offline method updates the index \( \mathcal{P} \) for each snapshot of \( S_t \) (Section IV-C) with a certain number of repetitions of local reinforcement, rep. The default value of parameter rep is 7. Its time complexity is \( O(k \cdot m + n \cdot \log n) \).

2) ANCO: An online method updates the index \( \mathcal{P} \) for initial \( S_0 \) with rep rounds (7 rounds by default) of local reinforcement. Then at each timestamp \( t \) on an activation stream, ANCO first updates \( S_{t-1} \) to \( S_t \) (with no more local reinforcement), then updates \( \mathcal{P} \) with \( S_t \). The initial edge activeness is 1. Its time complexity is \( O(\sum_{x \in U'} \deg(x)) \).

3) ANCOR: An online method that is similar to ANCO while it applies local reinforcement at intervals (5 timestamps by default) to update \( S_t \), then updates \( \mathcal{P} \) with \( S_t \).

Baseline Methods. The following baselines are compared with our methods listed above.

1) DYNA: An online method proposed by [43]. Its time complexity is \( O(|\Delta E| \cdot \frac{|E|}{2}) \) where \(|\Delta E|\) is the number of edges that are activated [43].

2) LWEF: An online method proposed by [38]. Its time complexity is \( O(d \cdot |\Delta E| \cdot n^2) \) where \( d \) is the average degree.

3) LOUV: An offline method proposed by [12] which attempts to maximize the modularity using a greedy optimization approach. It is used as the offline method of DYNA. Its time complexity is \( O(m) \) [43].

4) SCAN: An online method proposed by [39]. Its time complexity is \( O(m) \) [39].

5) ATTR: An offline method proposed by [33] with time complexity \( O(k \cdot d \cdot m) \), where \( k \) is number of repetitions. All algorithms were implemented in Java with library JavaSE-9. All experiments were conducted on a machine with Intel XeonE5-2697 CPU, 504GB main memory and Linux(centos). Only one core is engaged for all the algorithms.

Data sets. The experiments were conducted on 17 real-world graphs in Table I. The largest graph has more than 41M nodes and 1.2B edges. The data sets include social networks, collaboration networks, and email networks downloaded from [1],
### Table I: Data Set Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
<th>Vertex Size</th>
<th>Edge Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO</td>
<td>CollegeMsg</td>
<td>1,893</td>
<td>13,835</td>
<td>social</td>
</tr>
<tr>
<td>FB</td>
<td>fb-combine</td>
<td>4,039</td>
<td>88,234</td>
<td>social</td>
</tr>
<tr>
<td>CA</td>
<td>ca-GtQC</td>
<td>4,158</td>
<td>13,422</td>
<td>collaboration</td>
</tr>
<tr>
<td>MI</td>
<td>socfb-MIT</td>
<td>6,402</td>
<td>251,230</td>
<td>social</td>
</tr>
<tr>
<td>LA</td>
<td>lastfm-asia</td>
<td>7,624</td>
<td>27,806</td>
<td>social</td>
</tr>
<tr>
<td>CM</td>
<td>ca-CondMat</td>
<td>21,363</td>
<td>91,286</td>
<td>collaboration</td>
</tr>
<tr>
<td>IE</td>
<td>ia-email-eu</td>
<td>32,430</td>
<td>54,397</td>
<td>email</td>
</tr>
<tr>
<td>GI</td>
<td>git-web-ml</td>
<td>37,770</td>
<td>289,003</td>
<td>social</td>
</tr>
<tr>
<td>EA</td>
<td>email-Enron</td>
<td>224,832</td>
<td>339,925</td>
<td>social</td>
</tr>
<tr>
<td>DB</td>
<td>dblp</td>
<td>317,080</td>
<td>1,049,866</td>
<td>collaboration</td>
</tr>
<tr>
<td>AM</td>
<td>amazon</td>
<td>334,863</td>
<td>925,872</td>
<td>product</td>
</tr>
<tr>
<td>YT</td>
<td>youtube</td>
<td>1,314,890</td>
<td>2,987,624</td>
<td>product</td>
</tr>
<tr>
<td>DB2</td>
<td>dblp-2020</td>
<td>2,617,981</td>
<td>14,796,582</td>
<td>collaboration</td>
</tr>
<tr>
<td>OK</td>
<td>orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
<td>social</td>
</tr>
<tr>
<td>LI</td>
<td>li</td>
<td>3,597,962</td>
<td>34,681,189</td>
<td>social</td>
</tr>
<tr>
<td>TW2</td>
<td>twitter</td>
<td>4,713,138</td>
<td>17,601,953</td>
<td>social</td>
</tr>
<tr>
<td>TW</td>
<td>twitter-rv</td>
<td>41,652,230</td>
<td>1,202,513,046</td>
<td>social</td>
</tr>
</tbody>
</table>

### Table II: Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>2, 4, 8, 16</td>
<td>rep</td>
<td>0, 1, 3, 5, 7, 9</td>
</tr>
<tr>
<td>ε</td>
<td>0.2, 0.5, 0.6, 0.8, 0.9</td>
<td>μ</td>
<td>2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
</tbody>
</table>

### Parameters

We also constructed two real data sets DB2 and TW2. The construction details are described in [3].

#### Parameters

Table II summarizes the parameters used in the experiments. The default values of k (number of pyramids in $P$) and rep (number of repetitions of local reinforcement) are in bold. $\epsilon$ and $\mu$ determine the core nodes and are graph-dependent, their value setting on data sets and sensitivity tests are reported in [3]. Unless otherwise specified, all parameters in the experiments follow this setting.

### A. Effectiveness

#### Evaluating the Clustering Results Based on Ground Truth.

Static graphs LA, DB, AM, YT have 18, 11117, 11941, 3337 ground truth communities respectively. On activation graphs with varying $S_t$, we use Spectral Clustering [22] to obtain the clusters as ground truth. The number of ground truth clusters on simulated activation graphs CO, FB, CA, MI, and LA are set as $2 \times \sqrt{n}$, i.e., 87, 127, 129, 160, 175 respectively. Though the cluster number of all 5 baselines is fixed, for fair and consistent comparison, the cluster number of all our methods will select to be close to the ground truth number among granularity.

### Exp 1: Performance on Static Networks.

Table III shows the scores obtained by ANCF and 4 baselines when varying rep on static networks. For ANCF, increasing the parameter rep from 1 to 9 constantly improves the scores of all 4 measures.

The improvements illustrate that the local reinforcement can effectively capture the structural cohesiveness, which helps to improve the performance.

The overall performance of ANCF on structural measures is comparable to baselines. On Modularity, our method constantly outperforms all 4 baselines except LOUV: the average Modularity of ANCF9 over 4 datasets is 5, 8 and 55 times that of SCAN, ATTR and LWEP respectively. Although LOUV is an algorithm that specifically optimizes the Modularity, ANCF9 achieves only 18% lower Modularity than LOUV. The Conductance of ANCF9 is comparable to that of the baselines, specifically, our method is 20% lower and 12% higher than SCAN and ATTR. LOUV reports significantly smaller cluster number than the ground truth and our methods (will elaborate this in the last paragraph of Exp 1) and thus achieves biased 35% lower conductance since the number of inter-cluster edges [10], [40] is greatly reduced.

ANCF substantially outperforms all 4 baselines on ground truth-based measures. On NMI, except on LA, where LOUV is 3% higher than us, our method ANCF9 outperforms SCAN, ATTR, LOUV and LWEP by an average of 847%, 159%, 21%, 1192% respectively. Even with 1 repetition of local reinforcement only, ANCF1 can gain higher NMI than the baselines. On Purity, except on LA, where LOUV is 4% higher than us, ANCF9 is on average 137% higher than the best baseline LOUV. On F1-Measure, ANCF gains the highest scores on LA and AM and gives comparable scores on DB and YT to the best one.

LOUV tends to find a small cluster number. For example, on DB, it finds 190 clusters while ground truth number is 11187 and that of ours is 11115. On AM, LOUV finds 350 times fewer clusters than ground truth. The extremely small cluster number found by LOUV may not conform to the real communities as the real-world networks tend to consist of a large number of small-sized clusters [20].

### Exp 2: Performance on Activation Networks.

We generated activation networks ($\lambda = 0.1$) for timestamps 0-100, each timestamp randomly activated 5% of the edges on the graph. For graph snapshot of each timestamp, we generate the ground truth clusters using spectral clustering with uniform cluster numbers (Section VI-A). We evaluated 8 methods on 5 activation networks: 4 offline methods that need to re-compute the clusters of the snapshot at each timestamp, 4 online methods that incrementally update the clusters/indices.

Table IV shows the amortized time costs for handling each activation (of an edge per granularity level). ANCO is constantly the fastest method and ANCOR is the second. This conforms our theoretical analysis (Lemma 12). Among offline methods, ANCF is 25% and 14% faster than LOUV and SCAN on average. Among online methods, ANCO is constantly faster than DYNA by 3 orders of magnitude on average, and ANCOR is 99.97% faster than DYNA. Furthermore, our offline method ANCF is 18% faster than DYNA on average. The inferior performance of DYNA on activation networks is due to the edge update – the weight of all edges has to be updated at every timestamp even with no activation. This shows that the
TABLE III: Performance on Static Networks

<table>
<thead>
<tr>
<th></th>
<th>Modularity</th>
<th>NMI</th>
<th>Purity</th>
<th>F1-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN</td>
<td>0.25</td>
<td>0.41</td>
<td>0.47</td>
<td>0.03</td>
</tr>
<tr>
<td>ATTR</td>
<td>0.03</td>
<td>0.19</td>
<td>0.72</td>
<td>0.18</td>
</tr>
<tr>
<td>LOUV</td>
<td>0.81</td>
<td>0.72</td>
<td>0.83</td>
<td>0.71</td>
</tr>
<tr>
<td>LWEPI</td>
<td>0.03</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>ANCF5</td>
<td>0.67</td>
<td>0.68</td>
<td>0.72</td>
<td>0.40</td>
</tr>
<tr>
<td>ANCF0</td>
<td>0.69</td>
<td>0.69</td>
<td>0.74</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Fig. 4: Performance on Activation Networks (The x-axis is the timestamp and the y-axis is the score)

Table IV: Time Costs on Activation Networks

<table>
<thead>
<tr>
<th>Offline Recomputation</th>
<th>CO</th>
<th>FB</th>
<th>CA</th>
<th>MI</th>
<th>LA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN</td>
<td>0.01906</td>
<td>0.38861</td>
<td>0.01425</td>
<td>1.47617</td>
<td>0.02500</td>
</tr>
<tr>
<td>ATTR</td>
<td>5.92000</td>
<td>42.78333</td>
<td>0.28167</td>
<td>1140.557</td>
<td>0.00033</td>
</tr>
<tr>
<td>LOUV</td>
<td>0.06393</td>
<td>0.14704</td>
<td>0.05593</td>
<td>11.7213</td>
<td>0.183650</td>
</tr>
<tr>
<td>ANCF</td>
<td>0.02291</td>
<td>0.29376</td>
<td>0.01052</td>
<td>1.26654</td>
<td>0.02630</td>
</tr>
</tbody>
</table>

Online Update per Activation

<table>
<thead>
<tr>
<th>DTVA</th>
<th>LWEPI</th>
<th>ANCF</th>
<th>ANCOR</th>
<th>ANCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.002379</td>
<td>1.06833</td>
<td>0.00001</td>
<td>0.00001</td>
<td>0.00001</td>
</tr>
<tr>
<td>0.28283</td>
<td>5.09067</td>
<td>0.00006</td>
<td>0.00001</td>
<td>0.00001</td>
</tr>
</tbody>
</table>

Figure 4 shows the clustering quality, each row for a data set. For each measure, we use two adjacent subfigures to present the results for a better readability: the left shows the comparison between our methods and online baseline methods and the right shows the comparison between our methods and offline baselines. Overall, for all online methods, their performances deteriorate as time goes by. All offline methods have stable performance except SCAN due to its graph-sensitive parameter setup.

Among online methods, ANCOR constantly performs the best over time except on F1 Measure: F1-scores by DYNA are higher at the beginning while they gradually drop by an
average of 16%, ANCF then overtakes it at timestamp 60. Compared with DYNA, ANCOR and ANCO respectively gain 13% and 9% higher NMI on average, 62% and 83% higher Purity, and 6% and 12% lower F1-Measure. As time goes by, DYNA deteriorates (scores drop 0.12% per timestamp on average). This is because DYNA is a rule-based method that fully depends on the current updates and clusters, and the activation would gradually trap it to suboptimal solutions [43]. Among offline methods, ANCF constantly performs the best over time except on F1-Measure where ANCF is comparable to LOUV. Compared with SCAN and LOUV respectively, ANCF gains 48% and 19% higher NMI, 108% and 100% higher Purity, and 28% and 1.6% higher F1-Measure. Furthermore, compared with offline method LOUV, our online methods ANCOR and ANCO also gain 8% and 13% higher NMI, 80% and 60% higher Purity respectively. On F1-Measure, they are 9% and 16% lower than LOUV.

We also compare 3 versions of our methods. They have the same performance at time 0 but ANCOR and ANCO deteriorate as time goes by as the captured structural cohesiveness dissipates due to edge activation. The performance of ANCOR is constantly better than that of ANCO and it has less deterioration than ANCO over time. The better performance (an average improvement of 9% on measures) with the cost of longer update time (but still on average 69% faster than DYNA) suggests that there is a trade-off between cluster quality and frequency of local reinforcement.

**B. Efficiency**

**Exp 3: Index Time.** Figure 5 shows the index time of $\mathcal{P}$ with varying numbers of pyramids, $k$. We can see that the index time increases linearly with $k$. Although the vertex sizes of both OK and LJ are million-scale, the time for indexing OK is 3.5 times more than that of LJ, because OK is denser. This verifies Lemma 7. The indexing for TW with a billion edges takes only 1 hour.

**Exp 4: Index Size.** Figure 6 shows the index size of $\mathcal{P}$ with $k = 4$ to 16 pyramids (the space for storing the graph is excluded). The index memory usage increases linearly with $k$ and is highly related to the vertices number in the graph, echoing Lemma 7. We computed the data size and our index size on all real data sets in Table I with more than 1M edges. The average ratio of dataset/index size is 0.53.

**Exp 5: Query Time.** Figure 7 shows the running time of DirectedCluster at different levels on different graphs. The clustering time grows linearly as the graph edge size increases as its complexity is $O(m \log(n))$. On different levels, the extraction time is basically the same which verifies Lemma 8.

**Exp 6: Update Time.** Figure 8 shows the update time of UPDATE and RECONSTRUCT with the batch size varying from 1 to 1024. The cut-off update time was 1 hour. The update time of UPDATE grows linearly with the activation number in the batch. On average, the single update UPDATE is up to six orders of magnitude faster than RECONSTRUCT, e.g., UPDATE is $197296 \times$ faster than RECONSTRUCT on LJ, attributing to the local update property in Lemma 11.

Figure 9 shows, on TW2 over a day ($\lambda = 0.01$), the update time of UPDATE. We split the activations from June 25 to June 26, 2019 into 1440 sets, each set arrives as a batch per minute. As figure shows, though there are some bursty activations, 95% (marked by a dashed line) of the sets can be processed within 6.5 seconds. Note that we are processing the activations on a single core, which is not compatible with the large application of twitter. Given Lemma 13, we can easily handle the bursty cases/sets by engaging multiple cores.

Figure 10 shows the total time costs that online methods take to process the workload over the day on TW2. To simulate the workload, we randomly replace real activations with simulated queries by varying the percentage in 1%-32%. For each query node, we report the cluster it belongs to with the batch size varying $(\lambda)$.

**C. Case Study on Clustering with Index $\mathcal{P}$**

In order to show the effectiveness of our solution in real applications, especially the support of the three operations described in Problem 1, we conduct a case study on a subgraph of real data set DB2. The subgraph consists of 29 nodes and 735 activations in the time span of 30 years. Since the data provides only yearly information, we treat each year as a time step. This graph is an activation network (Section I) instead of only an/a evolving/dynamic network in a broad sense because: (1) there is no edge/node insertion/deletion, (2) there is a sequence of activations along the network, and
(3) the edge weights globally decay. We focus on node $v_8$ and 5 of its neighbors $v_0$, $v_5$, $v_7$, $v_{11}$ and $v_{26}$, which are plotted in larger size for easy recognition. Specifically, we inspect three questions. 1) Given a granularity level $l$, what are all the clusters and how do these clusters evolve? 2) How do the clusters that our target users belong to change over 30 years under the update of our proposed index? 3) How do different granularity levels reflect the information on different scales?

Figure 11 shows the monitoring of the clusters with user input nodes of interest $v_8$. Figure 11 (1)(3)(5) are the subgraphs at time $t_{10}$, $t_{20}$, $t_{30}$ respectively, on granularity level $l_2$. Figure 11 (2)(4)(6) are the subgraphs on $l_3$, where $t_{10}$ corresponds to 1995 and $t_{20}$ corresponds to 2005. The node-author mappings are shown in [3]. We show the dis-similarities (initially 1 at $t_0$) on the edge between $v_8$ and its interested neighbors, e.g., the dis-similarity between $v_8$ and $v_0$ decrease from 20.0 at $t_{10}$ to 0.4 at $t_{20}$.

From $t_0$ to $t_{10}$, ground truth discloses that $v_8$ collaborated with $v_7$ at $t_{5}$-$t_{11}$, while other neighbors collaborated with others, e.g., $v_5$ collaborated with $v_4$, $v_6$ and $v_0$ collaborated with $v_1$, $v_2$, $v_3$ etc.. Figure 11 (1),(2) show that, till $t_{10}$, the dis-similarity of $v_8$ to $v_7$ decreases while to other neighbors increase because no collaboration results in the decay of activeness and therefore the increase of dis-similarity. On both $l_2$ and $l_3$, our approach reports $v_8$ is in the same cluster with $v_7$ but not $v_0$, $v_5$, $v_{11}$, which is consistent with the ground truth. From $t_{10}$ to $t_{20}$, the ground truth discloses that $v_8$ collaborated with $v_{11}$, $v_0$ and $v_5$ at time $t_{11}$-$t_{22}$, $t_{11}$-$t_{35}$ and $t_{17}$-$t_{26}$ respectively. Figure 11 (3),(4) show that, till $t_{20}$, on both granularities, our approach reports $v_8$ is changed to be in the same cluster with $v_0$ and $v_{11}$ instead of $v_7$. For $t_{20}$-$t_{30}$, $v_8$ starts to collaborate with $v_{26}$ at $t_{23}$ till $t_{32}$. Figure 11 (5),(6) show that, till $t_{30}$, the dis-similarities of $v_8$ to $v_{25}$, $v_5$ and $v_{11}$ increase as they did not collaborate with $v_8$ since $t_{26}$, $t_{18}$ and $t_{22}$ respectively and therefore their activeness decays.

Lower level of granularity observes more detailed information. For example, for $v_8$ and $v_{26}$, on $l_2$, they are constantly in the same cluster regardless of the truth that $v_8$ has not collaborated with $v_{26}$ until $t_{23}$. In comparison, the changes can be captured on $l_3$: $v_0$ and $v_{26}$ are in different clusters at both $t_{10}$ and $t_{20}$ and they are put into the same cluster at $t_{30}$.

VII. CONCLUSIONS

This paper proposes a concise solution to scalable and effective clustering on massive dynamic graphs with time-decay edge weights. To avoid decay-triggered updates, a global decay factor is proposed to provide low cost activation-triggered updates. A framework that integrates the local structural coherence and decaying edge weights into a consistent distance metric is also proposed for hierarchical clustering with bounded updates and efficient query processing. Extensive experiments verify the effectiveness and efficiency of our solution on real datasets.

Acknowledgment. The work was supported by grants from NSFC Grant No. U1936205, the Research Grant Council of the Hong Kong Special Administrative Region, China [Project No.: CUHK 14205618], and CUHK Direct Grant No. 4055159. Miao Qiao was partially supported by Marsden Fund UOA1732 from Royal Society of New Zealand, and the Catalyst: Strategic Fund NZ-Singapore Data Science Research Programme UOAX2001, from the Ministry of Business Innovation and Employment, New Zealand.
REFERENCES


