Incremental Measurement of Structural Entropy for Dynamic Graphs

Runze Yang\textsuperscript{a}, Hao Peng\textsuperscript{b}, Angsheng Li\textsuperscript{a}

\textsuperscript{a}State Key Laboratory of Software Development Environment, the School of Computer Science and Engineering, Beihang University, Beijing, 100191, China
\textsuperscript{b}School of Cyber Science and Technology, Beihang University, Beijing, 100191, China

Abstract

Structural entropy is a metric that measures the amount of information embedded in graph structure data under a strategy of hierarchical abstracting. To measure the structural entropy of a dynamic graph, we need to decode the optimal encoding tree corresponding to the optimal hierarchical community partitioning of the graph. However, the current structural entropy methods do not support efficient incremental updating of encoding trees. To address this issue, we propose \textit{Incr-2dSE}, a novel incremental measurement framework that dynamically adjusts the community partitioning and efficiently computes the updated structural entropy for each snapshot of dynamic graphs. \textit{Incr-2dSE} consists of an online module and an offline module. The online module includes dynamic measurement algorithms based on two dynamic adjustment strategies for two-dimensional encoding trees, i.e., the naive adjustment strategy and the node-shifting adjustment strategy, which supports theoretical analysis of the updated structural entropy and incrementally adjusts the community partitioning towards a lower structural entropy. In contrast, the offline module globally constructs the encoding tree for the updated graph using static community detection methods and calculates the structural entropy by definition. We conduct experiments on an artificial dynamic graph dataset generated by Hawkes Process and 3 real-world datasets. Experimental results confirm that our dynamic measurement algorithms effectively capture the dynamic evolution of the communities, reduce time consumption, and provide great interpretability.

Keywords:
Structural entropy, dynamic graph, boundedness and convergence analysis, incremental algorithm

Preprint submitted to Artificial Intelligence October 25, 2023
1. Introduction

In 1953, Shannon [1] proposed the problem of structural information quantification to analyze communication systems. Since then, many information metrics of graph structures [2, 3, 4, 5] were presented based on the Shannon entropy of random variables. In recent years, Li et al. [6, 7] proposed an encoding-tree-based graph structural information metric, namely structural entropy, to discover the natural hierarchical structure embedded in a graph. The structural entropy has been used extensively in the fields of biological data mining [8, 9], information security [10, 11], and graph neural networks [12, 13], etc.

The computation of structural entropy [6] consists of three steps: encoding tree construction, node structural entropy calculation, and total structural entropy calculation. Firstly, the graph node set is hierarchically divided into several communities (shown in Fig. 1(a)) to construct a partitioning tree, namely an encoding tree (shown in Fig. 1(b)). Secondly, the volume and cut edge number of each community are counted to compute the structural entropy of each non-root node in the encoding tree. Finally, the structural entropy of the whole graph is calculated by summing up the node structural entropy. In general, smaller structural entropy corresponds to better community partitioning. Specifically, the minimized structural entropy, namely the graph structural entropy, corresponds to the optimal encoding tree, which reveals the best hierarchical community partitioning of the graph.

In dynamic scenarios, a graph evolves from its initial state to many updated states during time series [14]. To efficiently measure the quality of community partitioning, we need to incrementally compute the updated structural entropy at any given time. Unfortunately, the current structural entropy methods [6, 7] do not support efficient incremental computation. The first challenge is that the encoding tree needs to be reconstructed for every updated graph, which leads to enormous time consumption. To address this issue, we propose two dynamic adjustment strategies for two-dimensional encoding trees, namely the naive adjustment strategy and the node-shifting adjustment strategy. The former strategy maintains the old community partitioning while supporting structural entropy analysis, while the latter dynamically adjusts the community partitioning by moving nodes between communities based on the principle of structural entropy minimization. The second
The main contributions of this paper are as follows:

• Proposing two dynamic adjustment strategies for two-dimensional encoding trees to address the issue of reconstructing the encoding tree for every updated graph.

• Designing an incremental framework for incrementally measuring the updated two-dimensional structural entropy with low time complexity.

• Conducting extensive experiments on artificial and real-world datasets to demonstrate the effectiveness of our method in dynamically measuring community partitioning quality.

The article is structured as follows: Section 2 outlines the definitions and notations. Section 3 describe the dynamic adjustment strategies. The algorithms are detailed in Section 4 and the experiments are discussed in Section 5. Section 6 presents the related works before concluding the paper in section 7.
2. Definitions and Notations

In this section, we summarize the notations in Table 1 and formalize the definition of Incremental Sequence, Dynamic Graph, Encoding Tree, and Structural Entropy as follows.

**Definition 1** (Incremental Sequence). An incremental sequence is a set of incremental edges which can be written as $\xi = \{< (v_1, u_1), op_1 >, < (v_2, u_2), op_2 >, ..., < (v_n, u_n), op_n >\}$, where $(v_i, u_i)$ denotes an incremental edge $e_i$ and the operator $op_i \in \{+, -\}$ represents that the edge is added or removed. The number of the incremental edges $n$ is named the incremental size.

**Definition 2** (Dynamic Graph). In this work, a dynamic graph is defined as a series of snapshots of a temporal, undirected, unweighted, and connected graph $G = \{G_0, G_1, ..., G_T\}$. $G_0 = (V_0, E_0)$ denotes the initial graph, and $G_t = (V_t, E_t)$ denotes the updated graph at time $t$ ($1 \leq t \leq T$). To describe the temporal dynamic graph, we suppose that an incremental sequence $\xi_t$ arrives at each non-zero time $t$. The updated graph $G_t$ is generated by orderly combining all new edges and nodes introduced by $\xi_t$ with $G_{t-1}$, i.e., $G_t := \text{CMB}(G_{t-1}, \xi_t)$. We further define the cumulative incremental sequence at time $t$, denoted by $\xi_{1\rightarrow t}$, as the sequence formed by sequentially concatenating sequences $\xi_1, \xi_2, ..., \xi_t$, and then we have $G_t := \text{CMB}(G_0, \xi_{1\rightarrow t})$. 

---

Figure 1: a) A graph containing three communities A, B, and C, where A is divided into two sub-communities A.1 and A.2. b) An encoding tree of the left graph. Each leaf node corresponds to a single graph node. Each branch node corresponds to a community. The root node corresponds to the graph node-set.
Definition 3 (Encoding Tree). The concept of the encoding tree is the same as “the Partitioning Tree” proposed in the previous work [6]. The encoding tree \( T \) of a graph \( G = (\mathcal{V}, \mathcal{E}) \) is an undirected tree that satisfies the following properties:

1. The root node \( \lambda \) in \( T \) has a label \( T_\lambda = \mathcal{V} \).
2. Each non-root node \( \alpha \) in \( T \) has a label \( T_\alpha \subset \mathcal{V} \).
3. For each node \( \alpha \) in \( T \), if \( \beta_1, \beta_2, \ldots, \beta_k \) are all immediate successors of \( \alpha \), then \( T_{\beta_1}, \ldots, T_{\beta_k} \) is a partitioning of \( T_\alpha \).
4. The label of each leaf node \( \gamma \) is a single node set, i.e., \( T_\gamma = \{v\} \).
5. For a node \( \alpha \) in \( T \), its height is denoted as \( h(\alpha) \). Let \( h(\lambda) = 0 \) and \( h(\alpha) = h(\alpha^-) + 1 \), where \( \alpha^- \) is the parent of \( \alpha \). The height of the encoding tree \( h(T) \), namely the dimension, is equal to the maximum of \( h(\gamma) \).

Definition 4 (Structural Entropy). The structural entropy [6] is defined by Li et al. We follow this work and state the definition below. Given an undirected, unweighted, and connected graph \( G = (\mathcal{V}, \mathcal{E}) \) and its encoding tree \( T \), the structural entropy of \( G \) by \( T \) is defined as:

\[
H^T(G) = \sum_{\alpha \in T, \alpha \neq \lambda} -\frac{g_\alpha}{2m} \log \frac{V_\alpha}{V_{\alpha^-}},
\]  

where \( m \) is the total edge number of \( G \); \( g_\alpha \) is the cut edge number of \( T_\alpha \), i.e., the number of edges between nodes in and not in \( T_\alpha \); \( V_\alpha \) is the volume of \( T_\alpha \), i.e., the sum of the degrees of all nodes in \( T_\alpha \); \( \log(\cdot) \) denotes logarithm with a base of 2. We name \( H^T(G) \) as the \( K \)-dimensional structural entropy if \( T \)’ height is \( K \).

The graph structural entropy of \( G \) is defined as

\[
H(G) = \min_T \{ H^T(G) \},
\]  

where \( T \) ranges over all possible encoding trees.

If the height of \( T \) is restricted to \( K \), the \( K \)-dimensional graph structural entropy of \( G \) is defined as

\[
H^K(G) = \min_T \{ H^T(G) \},
\]  

where \( T \) ranges over all the possible encoding trees of height \( K \). The encoding tree corresponding to \( H^K(G) \), which minimizes the \( K \)-dimensional structural entropy, is named the optimal \( K \)-dimensional encoding tree.
Table 1: Glossary of Notations.

| Notation | Description |
|----------|-------------|
| $G$      | Graph       |
| $V; E$   | Node set; Edge set |
| $v; e$   | Node; Edge |
| $d_i; d_v$ | Node degree of node $v_i$; Node degree of $v$ |
| $m$      | The total edge number of $G$ |
| $\mathcal{T}$ | Encoding tree |
| $\lambda$ | The root node of an encoding tree |
| $\alpha$ | The non-root node in an encoding tree, i.e., the community ID |
| $A$      | The set of all 1-height nodes in an encoding tree |
| $T_\alpha$ | The label of $\alpha$, i.e., the node community corresponding to $\alpha$ |
| $V_\alpha$ | The volume of $T_\alpha$ |
| $g_\alpha$ | The cut edge number of $T_\alpha$ |
| $\mathcal{G}$ | Dynamic graph |
| $G_0$    | Initial state of a dynamic graph |
| $G_t$    | The updated graph at time $t$ |
| $\xi_t$  | Incremental sequence at time $t$ |
| $\xi_{1\rightarrow t}$ | Cumulative incremental sequence at time $t$ |
| $n$      | Incremental size |
| $\delta(v)$ | The degree incremental of $v$ |
| $\delta_V(\alpha)$ | The volume incremental of $T_\alpha$ |
| $\delta_g(\alpha)$ | The cut edge number incremental of $T_\alpha$ |
| $\phi_\lambda$ | The degree-changed node set |
| $\mathcal{A}$ | The set of 1-height tree nodes whose volume or cut edge number change |
| $H^T(G)$ | The structural entropy of $G$ by $\mathcal{T}$ |
| $H^T_{GI}(G, n)$ | Global Invariant with incremental size $n$ |
| $\Delta L$ | Local Difference, i.e., the approximation error between $H^T(G)$ and $H^T_{GI}(G, n)$ |
3. Dynamic Adjustment Strategies for Two-Dimensional Encoding Trees

In this section, we first introduce the naive adjustment strategy and analyze the updated structural entropy under this strategy. Then, we describe the node-shifting adjustment strategy and provide its theoretical proof.

3.1. Naive Adjustment Strategy

In this part, we first provide a formal description of the naive adjustment strategy. Next, we introduce two metrics, Global Invariant and Local Difference, to evaluate the updated structural entropy’s approximate value and the deviation from the exact value. Finally, we analyze the boundedness and convergence of the Local Difference.

3.1.1. Strategy Description

The naive adjustment strategy comprises two parts: the edge strategy and the node strategy. The edge strategy dictates that *incremental edges do not alter the encoding tree’s structure*. On the other hand, the node strategy specifies that *when a new node v connects with an existing node u (shown in Fig. 2(a)), and u corresponds to a leaf node η in the two-dimensional encoding tree, i.e., \( T_\eta = \{u\} \) (shown in Fig. 2(b)), a new leaf node γ with a label \( T_\gamma = \{v\} \) will be set as an immediate successor of η’s parent (α in Fig. 2(d)), instead of another 1-height node (β in Fig. 2(f)).* We can describe the modification of the encoding trees from the community perspective. Specifically, the *incremental edges do not change the communities of the existing nodes while the new node is assigned to its neighbor’s community (\( T_\alpha \) in Fig. 2(c)), rather than another arbitrary community (\( T_\beta \) in Fig. 2(e)).* Obviously, we can get the updated encoding tree, in other words, the updated community partitioning in the time complexity of \( O(n) \) given an incremental sequence of size \( n \). To ensure that the node strategy minimizes the updated structural entropy most of the time, we give the following theorem.

**Theorem 1.** Suppose that a new graph node \( v \) is connected to an existing node \( u \), where \( \{u\} \subseteq T_\alpha \). If \( \frac{2m+2}{V_\alpha+2} \geq e \), we have

\[
H^{T_v}_{v \rightarrow \alpha}(G') < H^{T_v}_{v \rightarrow \beta \neq \alpha}(G'),
\]

where \( H^{T_v}_{v \rightarrow \alpha}(G') \) denotes the updated structural entropy when the new node \( v \) is assigned to \( u \)'s community \( T_\alpha \), i.e., \( \{v\} \subset T_\alpha \), and \( H^{T_v}_{v \rightarrow \beta \neq \alpha}(G') \) represents the updated structural entropy when \( v \) is allocated to another arbitrary community \( T_\beta \neq \alpha \), i.e., \( \{v\} \subset T_\beta \neq \alpha \).
Proof. Differentiating the updated structural entropy of the two cases above, we can obtain
\[
\Delta H^{T'}(G') = H_{v \rightarrow \alpha}^{T'}(G') - H_{v \rightarrow \beta \neq \alpha}(G')
\]
\[
= \frac{1}{2m + 2} \left[ \log \frac{V_\alpha + 2}{2m + 2} + (g_\alpha - V_\alpha) \log \frac{V_\alpha + 1}{V_\alpha + 2} - (g_\beta - V_\beta) \log \frac{V_\beta}{V_\beta + 1} \right].
\] (5)

Here we define
\[
f_1(g, V) = (g - V) \log \frac{V + 1}{V + 2},
\] (6)
\[
f_2(g, V) = - (g - V) \log \frac{V}{V + 1}.
\] (7)

Let \( \theta (0 \leq \theta < 1) \) be the minimum proportion of the in-community edges in each community, i.e.,
\[
\theta = \min \left\{ \frac{V_\alpha - g_\alpha}{V_\alpha} \right\}.
\] (8)

Since \( 1 \leq V < 2m \) and \( 0 < g \leq (1 - \theta)V \), we have
\[
f_1(g, V) < -V \log \frac{V + 1}{V + 2} < \frac{1}{\ln 2} = \log e,
\] (9)
\[
f_2(g, V) \leq \theta V \log \frac{V}{V + 1} \leq \theta \log \frac{1}{2} = -\theta.
\] (10)
\[
\Delta H^T(G') \leq \frac{1}{2m+2} (\log \frac{V_\alpha + 2}{2m+2} + \log e - \theta) \\
= \frac{1}{2m+2} \log\left(\frac{V_\alpha + 2}{2m+2} \cdot 2^{-\theta e}\right).
\]

Therefore, if the following condition holds:

\[
2m + 2 \geq \max\{2^{-\theta e}\} = e,
\]

then Eq. (11) holds, and thus Theorem 1 is proven. According to Theorem 1, our node strategy minimizes the updated structural entropy when the total volume of the whole graph \(2m\) is approximately larger than \(e\) times the maximum volume \(V_m\) of all communities. Usually, we have \(\theta \approx 1\), so the real condition is much looser.

3.1.2. Global Invariant and Local Difference

Given a graph \(G\) and its two-dimensional encoding tree \(T\), the two-dimensional structural entropy of \(G\) by \(T\) is defined as

\[
H^T(G) = \sum_{\alpha_i \in A} (-\frac{g_{\alpha_i}}{2m} \log \frac{V_{\alpha_i}}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m} \log \frac{d_j}{V_{\alpha_i}}),
\]

where \(A\) denotes the set of 1-height nodes in \(T\), i.e., \(A = \{\alpha \in T | h(\alpha) = 1\}\).

In this paper, we only consider the two-dimensional encoding trees where the height of all leaf nodes is 2. Based on Eq. (13), we give the definition of the Global Invariant and Local Difference as follows.

**Definition 5** (Global Invariant and Local Difference). Given an original graph \(G\) and its two-dimensional encoding tree \(T\), the Global Invariant is defined as an approximate value of the updated structural entropy after an incremental sequence with size \(n\), i.e.,

\[
H^T_{GI}(G, n) = \sum_{\alpha_i \in A} (-\frac{g_{\alpha_i}}{2m+2n} \log \frac{V_{\alpha_i}}{2m+2n} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m+2n} \log \frac{d_j}{V_{\alpha_i}})
\]

\[
= -\frac{1}{2m+2n} (S_N + S_C + S_G),
\]

where

\[
S_N = \sum_{v_i \in V} d_i \log d_i,
\]

9
\[ S_C = \sum_{\alpha_i \in A} (g_{\alpha_i} - V_{\alpha_i}) \log V_{\alpha_i}, \quad (16) \]
\[ S_G = -\sum_{\alpha_i \in A} g_{\alpha_i} \log (2m + 2n). \quad (17) \]

Given the updated graph \( G' \), the updated two-dimensional encoding tree \( T' \), and incremental size \( n \), the Local Difference is defined as the difference between the exact updated two-dimensional structural entropy and the Global Invariant, as shown below:

\[ \Delta L = H^T(G') - H^T_{GI}(G, n). \quad (18) \]

If an incremental sequence \( \xi \) with size \( n \) is applied to a graph \( G \), resulting in a new graph \( G' \) and its corresponding two-dimensional encoding tree \( T' \) using the naive adjustment strategy, the updated two-dimensional structural entropy can be expressed as:

\[ H^T(G') = \sum_{\alpha_i \in A} \left( -\frac{g'_{\alpha_i}}{2m + 2n} \log \frac{V'_{\alpha_i}}{2m + 2n} + \sum_{v_j \in T_{\alpha_i}} -\frac{d'_{v_j}}{2m + 2n} \log \frac{d'_{v_j}}{V'_{\alpha_i}} \right). \quad (19) \]

Therefore the Local Difference can be written as

\[ \Delta L = H^T(G') - H^T_{GI}(G, n) = -\frac{1}{2m + 2n} (\Delta S_N + \Delta S_C + \Delta S_G), \quad (20) \]

where \( \Delta S_N \), \( \Delta S_C \), and \( \Delta S_G \) are defined as:

\[ \Delta S_N = \sum_{v_k \in \phi_\lambda} [(d_k + \delta(v_k)) \log(d_k + \delta(v_k)) - d_k \log d_k], \quad (21) \]
\[ \Delta S_C = \sum_{\alpha \in A} [(g_{\alpha} + \delta_g(\alpha) - V_{\alpha} - \delta_V(\alpha)) \log(V_{\alpha} + \delta_V(\alpha)) - (g_{\alpha} - V_{\alpha}) \log V_{\alpha}], \quad (22) \]
\[ \Delta S_G = -\sum_{\alpha \in A} \delta_g(\alpha) \log(2m + 2n). \quad (23) \]

Here, \( \delta(v_k) \) denotes the incremental change in degree \( d'_k - d_k \), \( \delta_V(\alpha) \) represents the incremental change in volume \( V'_{\alpha} - V_{\alpha} \), \( \delta_g(\alpha) \) represents the incremental change in cut edge \( g'_{\alpha} - g_{\alpha} \), \( \phi_\lambda \) denotes the set of nodes that have changes in degree \( \{v_k \in V' | \delta(v_k) \neq 0 \} \), and \( A \) denotes the set of 1-height tree nodes that have changes in \( V_{\alpha} \) or \( g_{\alpha} \), i.e., \( A = \{ \alpha \in A' | \delta_V(\alpha) \neq 0 \text{ or } \delta_g(\alpha) \neq 0 \} \).
Based on the above, we can get the Global Invariant in the time complexity of $O(1)$ if $S_N$, $S_C$, and $S_G$ are saved. The Local Difference can also be computed in $O(n)$ given an incremental sequence with size $n$. Overall, the updated two-dimensional structural entropy can be calculated in the time complexity of $O(n)$ with the naive adjustment strategy.

### 3.1.3. Boundedness Analysis

According to Eq. (20), the bounds of $\Delta L$ can be obtained by analyzing its components, namely $\Delta S_N$, $\Delta S_C$ and $\Delta S_G$. First, we analyze the maximum and minimum values of $\Delta S_N$. We define

$$s_N(d, x) = (d + x) \log(d + x) - d \log d.$$  \hfill (24)

Since $s_N(d, n)$ is monotonically increasing with $d$, $\Delta S_N$ takes the maximum value when $n$ new incremental edges connect the two nodes with the largest degree. Therefore, we have

$$\Delta S_N \leq 2s_N(d_m, n),$$  \hfill (25)

where $d_m$ denotes the maximum degree in $G$. Since multiple edges are not allowed, the equality may hold if and only if $n = 1$. When each of the $n$ incremental edges connects a one-degree node and a new node, $\Delta S_N$ is minimized:

$$\Delta S_N \geq ns_N(1, 0).$$  \hfill (26)

Second, we analyze the bounds of $\Delta S_C$ and $\Delta S_G$. For convenience, we define

$$\Delta S_{CG} = \Delta S_C + \Delta S_G.$$  \hfill (27)

We commence by analyzing the bound of $\Delta S_{CG}$ when adding one new edge. If a new edge is added between two communities $T_{a_1}$ and $T_{a_2}$, we get

$$\Delta S_{CG} = (g_{a_1} - V_{a_1}) \log(V_{a_1} + 1) - (g_{a_1} - V_{a_1}) \log V_{a_1}$$
$$+ (g_{a_2} - V_{a_2}) \log(V_{a_2} + 1) - (g_{a_2} - V_{a_2}) \log V_{a_2} - 2 \log(2m + 2).$$  \hfill (28)

Thus we have

$$\Delta S_{CG} \geq 2V_m \log\left(\frac{V_m}{V_m + 1}\right) - 2 \log(2m + 2),$$  \hfill (29)
and
\[ \Delta S_{CG} \leq -2 \log(2m + 2), \]  
where \( V_m \) denotes the maximum volume of all \( T_\alpha \). If a new edge is added within a single community \( T_\alpha \) (or a new node is connected with an existing node in \( T_\alpha \)), we have
\[ \Delta S_{CG} = (g_\alpha - V_\alpha - 2) \log(V_\alpha + 2) - (g_\alpha - V_\alpha) \log V_\alpha. \]  
Then we can obtain
\[ \Delta S_{CG} \geq - (V_m + 2) \log(V_m + 2) + V_m \log V_m, \]  
and
\[ \Delta S_{CG} \leq - 2 \log(V_{\text{min}} + 2), \]  
where \( V_{\text{min}} \) denotes the minimum volume of all \( T_\alpha \). We next analyze the bound of \( \Delta S_{CG} \) when adding \( n \) new edges. When the \( n \) edges are all between the two communities with the largest volume, we have
\[ \Delta S_{CG} \geq 2V_m \log \left( \frac{V_m}{V_m + n} \right) - 2n \log(2m + 2n) \]
\[ > - 2n - 2n \log(2m + 2n), \]  
and \( \Delta S_{CG} \) takes the minimum value:
\[ \Delta S_{CG\text{min}} = - 2n - 2n \log(2m + 2n). \]  
When each of the \( n \) edges is added within \( n \) communities with the smallest volume, respectively, \( \Delta S_{CG} \) takes its maximum value:
\[ \Delta S_{CGm} = - 2n \log(V_{\text{min}} + 2). \]  
Finally, we can get a lower bound of \( \Delta L \) as
\[ \text{LB}(\Delta L) = - \frac{1}{2m + 2n} (2s_N(d_m, n) + \Delta S_{CGm}) \]
\[ = \frac{1}{m + n} [d_m \log d_m - (d_m + n) \log (d_m + n) + n \log(V_{\text{min}} + 2)]. \]  
An upper bound of \( \Delta L \) is as follows:
\[ \text{UB}(\Delta L) = - \frac{1}{2m + 2n} (ns_N(1, 0) + \Delta S_{CG\text{min}}) \]
\[ = \frac{n \log(m + n) + \frac{5}{2}n}{m + n}. \]
3.1.4. Convergence Analysis

In this section, we aim to analyze the convergence of the Local Difference as well as its first-order absolute moment. To denote that one function converges at the same rate or faster than another function, we use the notation $g(m) = O(f(m))$, which is equivalent to $\lim_{m \to \infty} \frac{g(m)}{f(m)} = C$, where $C$ is a constant.

**Theorem 2.** Given the incremental size $n$, the Local Difference converges at the rate of $O\left(\frac{\log m}{m}\right)$, represented as:

$$\Delta L = O\left(\frac{\log m}{m}\right). \quad (39)$$

**Proof.** The lower bound of $\Delta L$ is given by:

$$LB(\Delta L) = \frac{m \log m - (d_m + n) \log (d_m + n)}{m + n} + \frac{n \log (V_{min} + 2)}{m + n} \geq \frac{m \log m - (m + n) \log (m + n)}{m + n} + \frac{n \log (2 + 2)}{m + n},$$

$$\geq \frac{1}{m + n} [\log (1 - \frac{n}{m + n})^m - n \log (m + n)] = O\left(\frac{\log m}{m}\right). \quad (40)$$

Similarly, the upper bound is given by:

$$UB(\Delta L) = \frac{n \log (m + n) + \frac{5}{2} n}{m + n} = O\left(\frac{\log m}{m}\right). \quad (41)$$

Since

$$LB(\Delta L) \leq \Delta L \leq UB(\Delta L), \quad (43)$$

Theorem 2 is proved. It follows that the difference between the exact value of the updated two-dimensional structural entropy and the Global Invariant converges at the rate of $O\left(\frac{\log m}{m}\right)$. 
Definition 6. Let $X$ be a random variable representing the incremental size $n$. We remind that $\mathbb{E}[X] = \pi$.

Theorem 3. The first-order absolute moment of the Local Difference converges at the rate of $O\left(\frac{\log m}{m}\right)$:

$$\mathbb{E}[|\Delta L|] = O\left(\frac{\log m}{m}\right).$$  \hfill (45)

Proof. We can represent the expectation of the lower bound of $\Delta L$ as:

$$\mathbb{E}[|\text{LB}(\Delta L)|] = \mathbb{E}[\left| \frac{d_m \log d_m - (d_m + X) \log(d_m + X)}{m + X} \cdot \frac{n \log(V_{\text{min}} + 2)}{m + X} \right|]$$

$$\leq \mathbb{E}\left[ \frac{(m + X) \log (m + X) - m \log m}{m + X} \right] + \mathbb{E}\left[ \frac{X \log(m + 2)}{m + X} \right]$$

$$\leq \frac{m \log m - (m + \pi) \log (m + \pi)}{m + \pi} + \frac{\pi \log(m + 2)}{m + \pi}$$

$$= O\left(\frac{\log m}{m}\right),$$  \hfill (46)

Similarly, the expectation of the upper bound is given by:

$$\mathbb{E}[|\text{UB}(\Delta L)|] = \mathbb{E}\left[ \frac{X \log(m + X) + \frac{5}{2} X}{m + X} \right]$$

$$\leq \frac{\pi \log(m + \pi) + \frac{5}{2} \pi}{m + \pi}$$

$$= O\left(\frac{\log m}{m}\right).$$  \hfill (47)

Finally, since

$$0 \leq \mathbb{E}[|\Delta L|] \leq \max\{\mathbb{E}[|\text{LB}(\Delta L)|], \mathbb{E}[|\text{UB}(\Delta L)|]\},$$  \hfill (48)

Theorem 3 is proved. Thus, the expectation of the absolute value (i.e., the first-order absolute moment) of the Local Difference converges at the rate of $O\left(\frac{\log m}{m}\right)$. 

14
3.2. Node-Shifting Adjustment Strategy for Two-Dimensional Encoding Trees

Although the naive adjustment strategy can quickly obtain an updated two-dimensional encoding tree and its corresponding structural entropy, we still need a more effective strategy to get a better community partitioning towards lower structural entropy. Therefore, we propose another novel dynamic adjustment strategy, namely node-shifting, by moving nodes to their optimal preference communities iteratively. Different from the naive adjustment strategy, edge changes can change the communities of the existing nodes to minimize the structural entropy. Besides, this strategy supports multiple incremental edges at the same time and the removal of the existing edges.

In the following, we first describe the node-shifting adjustment strategy in detail and then prove that the node’s movement towards its optimal preference community can get the lowest structural entropy. Finally, we discuss the limitations of this strategy.

3.2.1. Strategy Description

We first give the definition of the optimal preference community in Definition 7.

**Definition 7 (Optimal Preference Community (OPC)).** Given a graph \( G = (V, E) \) and a target node \( v_t \in V \), the optimal preference community (OPC) of \( v_t \) is defined as the community \( T_{\alpha^*} \) in which

\[
\alpha^* = \begin{cases} 
\arg \min_{\alpha} \left[ (g_\alpha - V_\alpha) \log \frac{V_\alpha}{V_\alpha + d_t} + 2d(\alpha) \log \frac{V_\alpha + d_t}{2m} \right], & v_t \notin T_\alpha; \\
\arg \min_{\alpha} \left[ (g_\alpha - V_\alpha + d_t + d(\alpha)) \log \frac{V_\alpha - d_t}{V_\alpha} + 2d(\alpha) \log \frac{V_\alpha}{2m} \right], & v_t \in T_\alpha,
\end{cases}
\]

where \( d(\alpha) \) denotes the number of the edges connected between \( v_t \) and \( T_\alpha \).

The node-shifting adjustment strategy is designed as follows: (1) let involved nodes be all nodes that appeared in the incremental sequence; (2) for each involved node, move it to its OPC; (3) update the involved nodes to all nodes connected with the shifted nodes but in different communities, then repeat step (2).

Fig. 3 and Fig. 4 give examples to illustrate the node-shifting adjustment strategy in different situations. Fig. 3 shows how incremental edges affect community partitioning. In Fig. 3(a), the graph is divided into 2 communities \( T_\alpha \) and \( T_\beta \). In Fig. 3(b), 4 incremental edges (red dotted) are inserted into the graph. Then all involved nodes (red outlined) are checked for moving
into their OPCs. In this step, one green node is shifted from $T_\alpha$ to $T_\beta$ (denoted by the red arrow). In Fig. 3(c), the shifted node in the previous step “sends messages” (red dotted arrows) to its neighbors in $T_\alpha$ (red outlined). The nodes that received the message (red outlined) are then checked for shifting. At this time, another green node moves into $T_\beta$. In Fig. 3(d)-(e), the graph follows the above process to continue the iterative update. The final community partitioning is shown in Fig. 3(f). Fig. 4 shows how new nodes are assigned to communities. Fig. 4(a) gives a 7 nodes graph with 2 communities. In Fig. 4(b), 3 new nodes (white filled) are added with 7 incremental edges and they belong to no community. Then all of them and their existing neighbors become involved nodes. Next, the upper new node is assigned to $T_\alpha$ because $T_\alpha$ is determined as its OPC. Also, the lower two new nodes are moved into their OPCs. In Fig. 4(c), the new involved nodes (red outlined) are checked. Fig. 4(d) shows the final state of this node-shifting process.

3.2.2. Theoretical Proof

In this part, we provide a simplified model to theoretically derive the OPC’s solution formula (Eq. (49)). In the graph of this model, there exists $r$ communities $T_{\alpha_1}, T_{\alpha_2}, \ldots, T_{\alpha_r}$. There is also a target node $v_t$ which does not belong to any community. The number of the edges connected between $v_t$ and $T_{\alpha_i}$ is denoted by $d^{(i)}$. The volume and the number of the cut edges of $T_{\alpha_i}$ are denoted by $V_i$ and $g_i$, respectively. Then we have Theorem 4.

Figure 3: An example of the node-shifting adjustment strategy for adding new edges.
Figure 4: An example of the node-shifting adjustment strategy for adding new nodes.

Figure 5: A simplified model for theoretical analysis.

**Theorem 4.** Suppose that the node $v_t$ is moving into community $T_{\alpha_k}, k \in \{1, 2, ..., r\}$. The updated structural entropy is minimized when $v_t$ moves into $T_{\alpha_k^*}$ where

$$k^* = \arg \min_k [(g_k - V_k) \log \frac{V_k}{V_k + d_t} + 2d^{(k)} \log \frac{V_k + d_t}{2m}].$$ \hspace{1cm} (50)

**Proof.** Let $H_k$ be the two-dimensional structural entropy after $v_t$ moves into $T_{\alpha_k}$. Then $H_k$ is given by

$$H_k = \sum_{\alpha_i \neq \alpha_k} \left( -\frac{g_i}{2m} \log \frac{V_i}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m} \log \frac{d_j}{V_i} \right) + \left( -\frac{g_k + d_t - 2d^{(k)}}{2m} \log \frac{V_k + d_t}{2m} \right)$$

$$+ \sum_{v_q \in T_{\alpha_k} \setminus \{v_t\}} -\frac{d_q}{2m} \log \frac{d_q}{V_k + d_t} - \frac{d_t}{2m} \log \frac{d_t}{V_k + d_t}. \hspace{1cm} (51)$$

Therefore, the structural entropy is minimized when $v_t$ moves into $T_{\alpha_k^*}$ where

$$k^* = \arg \min_k H_k. \hspace{1cm} (52)$$
Figure 6: An example where the node-shifting adjustment strategy does not converge. The left is a graph added with two incremental edges which cause two nodes to shift. The right shows the updated communities after the first iteration and the future movement at the second iteration. After the second iteration, the updated graph becomes the left again.

Let the structural entropy before the node movement be $\tilde{H}$, which is given by

$$\tilde{H} = \sum_{\alpha_i} \left( -\frac{g_i}{2m} \log \frac{V_i}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m} \log \frac{d_j}{V_i} \right) + \left( -\frac{d_t}{2m} \log \frac{d_t}{2m} - \frac{d_t}{2m} \log \frac{d_t}{d_t} \right).$$

Since $\tilde{H}$ is independent of $k$, we have

$$k^* = \arg \min_k 2m(H_k - \tilde{H})$$

$$= \arg \min_k \left[ (g_k - V_k) \log \frac{V_k}{V_k + d_t} + 2d_t^{(k)} \log \frac{V_k + d_t}{2m} \right].$$

Therefore Theorem 4 is proved. In practice, all nodes are belonging to their communities. We can first move the target node out of its community, and then use Eq. (50) to determine the OPC. This process is equivalent to directly using Definition 7 without moving out the target node.

### 3.2.3. Limitations

The limitations of the node-shifting adjustment strategy are listed below. First, this strategy can’t handle the birth of new communities and the dismission of the existing communities. Second, it is hard to give the bound of the gap between the Global Invariant and the updated structural entropy. Third, the node-shifting adjustment strategy may not converge in some cases (Fig. 6 gives an example), which forces us to set the maximum number of iterations.
4. Incre-2dSE: an Incremental Framework of Measuring the Updated Two-Dimensional Structural Entropy

4.1. Outline

The illustration of our incremental framework Incre-2dSE is shown in Fig. 7. This framework aims to measure the updated two-dimensional structural entropy given the original graph, the original encoding tree, and the incremental sequences. The framework includes two modules: the online module (the green and blue parts shown in Fig. 7) and the offline module (the red part shown in Fig. 7). The online module consists of two stages. The first stage is Initialization. In this stage, the Structural Data, which contains a graph’s essential data to compute the structural entropy, is extracted from the original graph and its encoding tree (Fig. 7i). Then the Structural Expressions, which are defined as the expressions of the Structural Data, are computed and saved (Fig. 7ii). For the same original graph, Initialization only needs to be performed once. The second stage is Measurement. In this stage, the Adjustment, which is a data structure storing the changes in degree, volume, and cut edge number from the original graph to the updated graph, is first generated and saved according to the structural data and the incremental sequence by the Adjustment generation algorithm with the naive adjustment strategy (NAGA) and the node-shifting adjustment strategy (NSGA) (Fig. 7iii). Then, the Adjustment-based incremental updating algorithm (AUIA) is called to gather the Structural Data, the Structural Expression, and the Adjustment to efficiently calculate the updated structural entropy and update the Structural Data and the Structural
Expressions (Fig. 7). The offline module includes a traditional offline algorithm (TOA). TOA commences by updating the graph using the incremental sequence (Fig. 7(a)). Next, the new encoding tree of the updated graph is re-constructed using a static community detection algorithm (Fig. 7(b)). Then, the updated Structural Data is extracted (Fig. 7(c)), and finally, the updated structural entropy is computed by definition (Fig. 7(d)).

4.2. Online Module

4.2.1. Stage 1: Initialization

Firstly, we give the definition of the Structural Data and the Structural Expressions (Definition 8) of a graph $G$.

**Definition 8 (Structural Data and Structural Expressions).** Given a graph $G$, the Structural Data of $G$ is defined as follows:

1. (node level) the degree $d_i$ of each $v_i \in V$;
2. (community level) the volume $V_\alpha$ and the cut edge number $g_\alpha$;
3. (graph level) the total edge number $m$;
4. (node-community map) the community ID $v_i$ belongs to, denoted by $\alpha(v_i) \in A$ where $v_i \in T_\alpha(v_i)$;
5. (community-node map) the community $T_\alpha$ of each $\alpha \in A$.

The Structural Expressions of $G$ are defined as follows:

1. (node level)

\[
\hat{S}_N = \sum_{d \in D} k_d d \log d, \tag{55}
\]

where $k_d$ denotes the node number of each $d \in D$ while $D$ denotes the set of all distinct degrees in $G$;

2. (community level)

\[
\hat{S}_C = \sum_{\alpha \in A} (g_\alpha - V_\alpha) \log V_\alpha; \tag{56}
\]

3. (graph level)

\[
\hat{S}_G = - \sum_{\alpha \in A} g_\alpha. \tag{57}
\]
Given a graph \( G = (\mathcal{V}, \mathcal{E}) \) as a sparse matrix and its two-dimensional encoding tree represented by a dictionary like \{community ID 1: node list 1, community ID 2: node list 2, ...\}, the Structural Data can be easily obtained and saved in the time complexity of \( O(|\mathcal{E}|) \) (Fig. 7). Then the Structural Expressions can be calculated with the saved Structural Data in \( O(|\mathcal{V}|) \) (Fig. 7). Overall, the Initialization stage requires total time complexity \( O(|\mathcal{E}|) \).

4.2.2. Stage 2: Measurement

In this stage, we first need to generate the Adjustment (Definition 9) from \( G \) to \( G' \). We provide two algorithms for generating Adjustments by the proposed dynamic adjustment strategies, namely the naive adjustment generation algorithm (NAGA) and the node-shifting adjustment generation algorithm (NSGA) (Fig. 7). The input of both two algorithms is the Structural Data of the original graph and an incremental sequence with size \( n \). The pseudo-code of NAGA and NSGA are shown in Algorithm 1 and Algorithm 2, respectively. The time complexity of NAGA is \( O(n) \) because it needs to traverse \( n \) edges in the incremental sequence and it only costs \( O(1) \) for each edge. In NSGA, we first need \( O(n) \) to initialize the Adjustment (line 5-31). Second, in the node-shifting part (line 32-51), we need to determine the OPC for all \( |I| \) involved nodes, which costs \( O(|A||I|) \). This step is repeated \( N \) times and the time cost is \( O(|A||I_1| + |I_2| + ... + |I_N|) \), where \( I_i \) denotes the number of the involved nodes in the \( i \)-th iteration. Since \( |I_1| \leq n \) and \( |I_{i+1}| \leq |I_i| \) most of the time, the total time complexity of NSGA is \( O(nN|A|) \).

Definition 9 (Adjustment). The Adjustment from the original graph \( G \) to the updated graph \( G' \) is defined as follows:

1. (node level) the degree change \( \delta(d) = k'_d - k_d \) of each \( d \in D \), where \( D \) denotes the set of the degrees which have node number changes from \( G \) to \( G' \);
2. (community level) the volume change \( \delta_V(\alpha) \) and the cut edge number change \( \delta_g(\alpha) \) of each \( \alpha \in A \);
3. (graph level) the total edge number change \( n \);
4. (node-community map) the change list of the node-community map Structural Data denoted by \( J_{n-c} = \{(v_i, \alpha'(v_i)), \ldots\} \) where \( \alpha'(v_i) \) denotes the new community ID of \( v_i \);
Algorithm 1: Naive adjustment generation algorithm (NAGA)

**Input**: The Structural Data \( (d_i, V_{\alpha}, g_{\alpha}, m, \alpha(v_i), \text{and } T_{\alpha}) \) of \( G \), and an incremental sequence \( \xi \) from \( G \) to \( G' \).

**Output**: The Adjustment \( (\delta(v_i), \delta_k(d), \delta_V(\alpha), \delta_g(\alpha), n, J_n-c, \text{and } J_c-n) \) from \( G \) to \( G' \) by the naive adjustment strategy.

1. \( n := \text{GetLength}(\xi) \);
2. \( \delta(v_i) := 0, \delta_k(d) := 0, \delta_V(\alpha) := 0, \delta_g(\alpha) := 0, D := \emptyset, A := \emptyset, J_n-c := \emptyset, J_c-n := \emptyset \);
3. Let the proxy maps be \( \hat{\alpha}(v_i) := \alpha(v_i), v_i \in V \);
4. Let the proxy node level Structural Data be \( \hat{d}_v := d_v, v \in V \), where \( d_v \) denotes the degree of \( v \);
5. for \( e = (u, v, +) \in \xi \) do
6. \( D := D \cup \{d_u, d_v, d_u + 1, d_v + 1\} \);
7. \( \delta_k(\hat{d}_u) := \delta_k(\hat{d}_u) - 1, \delta_k(\hat{d}_u + 1) := \delta_k(\hat{d}_u + 1) + 1 \);
8. \( \hat{d}_v := \hat{d}_v - 1, \hat{d}_v + 1 := \hat{d}_v + 1 + 1 \);
9. \( \hat{d}_u := \hat{d}_u + 1, \hat{d}_v := \hat{d}_v + 1, \delta(u) := \delta(u) + 1, \delta(v) := \delta(v) + 1 \);
10. if \( \hat{\alpha}(u) == \text{NULL} \) then
11. \( \hat{\alpha}(u) := \hat{\alpha}(v) \);
12. \( J_n-c := J_n-c \cup \{(u, \hat{\alpha}(v))\}, J_c-n := J_c-n \cup \{(\hat{\alpha}(v), u, +)\} \);
13. end
14. if \( \hat{\alpha}(v) == \text{NULL} \) then
15. \( \hat{\alpha}(v) := \hat{\alpha}(u) \);
16. \( J_n-c := J_n-c \cup \{(v, \hat{\alpha}(u))\}, J_c-n := J_c-n \cup \{(\hat{\alpha}(u), v, +)\} \);
17. end
18. \( A := A \cup \{\hat{\alpha}(u), \hat{\alpha}(v)\} \);
19. if \( \hat{\alpha}(v) == \hat{\alpha}(u) \) then
20. \( \delta_V(\hat{\alpha}(v)) := \delta_V(\hat{\alpha}(v)) + 2 \);
21. end
22. if \( \hat{\alpha}(v) \neq \hat{\alpha}(u) \) then
23. \( \delta_V(\hat{\alpha}(v)) := \delta_V(\hat{\alpha}(v)) + 1, \delta_V(\hat{\alpha}(u)) := \delta_V(\hat{\alpha}(u)) + 1 \);
24. \( \delta_g(\hat{\alpha}(v)) := \delta_g(\hat{\alpha}(v)) + 1, \delta_g(\hat{\alpha}(u)) := \delta_g(\hat{\alpha}(u)) + 1 \);
25. end
26. end
27. return the Adjustment from \( G \) to \( G' \).
Algorithm 2: Node-shifting adjustment generation algorithm (NSGA)

Input: The Structural Data \((d_i, V_\alpha, g_\alpha, m, \alpha(v_i), \text{ and } T_\alpha)\) of the original graph \(G\), an incremental sequence \(\xi\) from \(G\) to \(G'\), and the iteration number \(N\).

Output: The Adjustment \((\delta(v_i), \delta_k(d), \delta_V(\alpha), \delta_g(\alpha), n, J_{n-c}, \text{ and } J_{c-n})\) from \(G\) to \(G'\).

1. \(n := \text{GetLength}(\xi), J_{n-c} := \emptyset, J_{c-n} := \emptyset\), \(\delta(v_i) := 0, \delta_k(d) := 0, \delta_V(\alpha) := 0, \delta_g(\alpha) := 0;\)
2. Let the proxy maps be \(\hat{\alpha}(v_i) := \alpha(v_i), \hat{\alpha}(v_i) := \alpha(v_i), v_i \in V;\)
3. Let the proxy node-level Structural Data be \(\hat{d}_v := d_v, v \in V\), where \(d_v\) denotes the degree of \(v;\)
4. Let the involved node set be \(I := \emptyset;\)
5. \// Initialize the Adjustment
6. for \((e = (u, v, op) \in \xi)\) do
7. \quad if \(op == +\) then
8. \quad \quad if \(u, v\) are both existing nodes in \(V\) then
9. \quad \quad \quad Update the node-level Adjustment (using the proxy node-level Structural Data, the same below);
10. \quad \quad \quad if \(\alpha(u)\) and \(\alpha(v)\) are both not None then
11. \quad \quad \quad \quad Update the community-level Adjustment without changing the community partitioning;
12. \quad \quad \quad else if \(\alpha(u)\) or \(\alpha(v)\) is None (suppose \(\alpha(u) == \text{None}\) then
13. \quad \quad \quad \quad \quad \delta_V(\alpha(u)) := \delta_V(\alpha(u)) + 1; \delta_g(\alpha(u)) := \delta_g(\alpha(u)) + 1;
14. \quad \quad \quad else if u or v does not exist (suppose u does not exist) then
15. \quad \quad \quad \quad \quad \alpha(u) := \text{None};
16. \quad \quad \quad \quad \quad Update the node-level Adjustment;
17. \quad \quad \quad \quad \quad \delta_V(\alpha(u)) := \delta_V(\alpha(u)) + 1; \delta_g(\alpha(u)) := \delta_g(\alpha(u)) + 1;
18. \quad \quad \quad \quad \quad \quad J_{n-c} := J_{n-c} \cup \{(u, \text{None})\}; J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\};
19. \quad \quad \quad else if u and v are both not existing then
20. \quad \quad \quad \quad \quad \alpha(u) := \text{None}, \alpha(v) := \text{None};
21. \quad \quad \quad \quad \quad Update the node-level Adjustment;
22. \quad \quad \quad \quad \quad J_{n-c} := J_{n-c} \cup \{(u, \text{None})\}; J_{n-c} := J_{n-c} \cup \{(v, \text{None})\};
23. \quad \quad \quad \quad \quad J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\}; J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\};
24. \quad \quad \quad Update the proxy node-level Structural Data as if the edge \(e\) is added into the graph;
25. \quad \quad else if \(op == -\) then
26. \quad \quad \quad Update the node-level and the community-level Adjustment without changing the community partitioning;
27. \quad \quad \quad Update the proxy node-level Structural Data as if the edge \(e\) is removed from the graph;
28. end
29. end

// See the rest on the next page
32 // Start node-shifting
33 \tau := 1;
34 \textbf{while} \ \tau \leq N \text{ and } I \neq \emptyset \textbf{do}
35 \hspace{1em} \tilde{I} := \emptyset, \ X := \emptyset;
36 \hspace{1em} \textbf{for each node } v \in I \textbf{ do}
37 \hspace{2em} \text{Determine the OPC of } v \text{ denoted by } \alpha^* \text{ using } \hat{\alpha}(v);
38 \hspace{2em} X := X \cup \{(v, \alpha^*)\};
39 \hspace{2em} \textbf{if } \hat{\alpha}(v) \neq \alpha^* \textbf{ then}
40 \hspace{3em} \text{Update the Adjustment as if } v \text{ moves into } T_{\alpha^*} \text{ using } \hat{\alpha}(v);
41 \hspace{3em} \textbf{for each node } z \in \text{Neighbor}(v) \textbf{ do}
42 \hspace{4em} \textbf{if } \hat{\alpha}(z) \neq \alpha^* \textbf{ then}
43 \hspace{5em} \tilde{I} := \tilde{I} \cup \{z\};
44 \hspace{4em} \textbf{end}
45 \hspace{3em} \hat{\alpha}(v) := \alpha^*;
46 \hspace{2em} \textbf{end}
47 \hspace{1em} \textbf{end}
48 \hspace{1em} \textbf{for each } (v, \alpha) \in X \textbf{ do}
49 \hspace{2em} \hat{\alpha}(v) := \alpha;
50 \hspace{1em} \textbf{end}
51 \hspace{1em} I := \tilde{I}, \ \tau := \tau + 1;
52 \textbf{end}
53 \textbf{return} \text{ the Adjustment from } G \text{ to } G'.

5. \textit{(community-node map)} the change list of the community-node map Structural Data denoted by \( J_{c-n} = \{ ..., (\alpha_i, v_j, +/-), ... \} \) where \((\alpha_i, v_j, +/-)\) denotes that community \( T_{\alpha_i} \) is updated as \( T_{\alpha_i} \cup \{v_j\} \) or \( T_{\alpha_i} / \{v_j\} \).

After getting the Adjustment, the updated two-dimensional structural entropy of \( G' \) can then be incrementally calculated by

\[
H^T(G') = -\frac{1}{2m+2n} [\hat{S}'_N + \hat{S}'_C + \hat{S}'_G \log(2m+2n)],
\]

(58)

where \( \hat{S}'_N, \hat{S}'_C, \text{ and } \hat{S}'_G \) denote the incrementally updated Structural Expres-
sions:
\[
\hat{S}_N' = \hat{S}_N + \sum_{d \in \mathcal{D}} \delta_k(d) d \log d;
\]
\[
\hat{S}_C' = \hat{S}_C + \sum_{\alpha \in \mathcal{A}} [(g_{\alpha} + \delta_g(\alpha) + V_{\alpha} + \delta_V(\alpha)) \log(V_{\alpha} + \delta_V(\alpha)) - (g_{\alpha} + V_{\alpha}) \log V_{\alpha}];
\]
\[
\hat{S}_G' = \hat{S}_G + \sum_{\alpha \in \mathcal{A}} -\delta_g(\alpha).
\]

(59)

To implement the above incremental calculation process, we provide the Adjustment-based incremental updating algorithm (AIUA) (Fig. 7). Given the input, i.e., the Structural Data and Structural Expressions of the original graph and an Adjustment to the updated graph, we can compute the updated two-dimensional structural entropy incrementally, and update the Structural Data and Structural Expressions efficiently preparing for the next AIUA process when a new Adjustment comes. The pseudo-code of AIUA is shown in Algorithm 3. The time complexity of updating the Structural Data is \( O(|\phi| + |\mathcal{A}| + |J_n-c| + |J_{c-n}|) \leq O(n) \). The time complexity of updating the Structural Expressions is \( O(|\mathcal{D}| + |\mathcal{A}|) \leq O(n) \). The time complexity of calculating the updated two-dimensional structural entropy is \( O(1) \). In summary, the total time complexity of AIUA is \( O(n) \).

4.3. Offline Module

In this module, we implement the traditional offline algorithm (TOA), which reconstructs the encoding tree of the updated graph and calculates the updated two-dimensional structural entropy by definition. TOA consists of the following four steps. Firstly, it generates the updated graph by combining the original graph and the incremental sequence (\( \text{a} \) in Fig. 7). Secondly, it partitions the graph node set into communities using several different community detection algorithms, e.g., Infomap [15], Louvain [16], and Leiden [17], to construct the two-dimensional encoding tree (\( \text{b} \) in Fig. 7). Thirdly, the node-level, community-level, and graph-level Structural Data of the updated graph is counted and saved (\( \text{c} \) in Fig. 7). Finally, the updated structural entropy is computed via Eq. (19) (\( \text{d} \) in Fig. 7). The total time cost of TOA is \( O(|\mathcal{E}| + n) \) plus the cost of the chosen community detection algorithm. The pseudo-code of TOA is shown in Algorithm 4.
Algorithm 3: Adjustment-based incremental updating algorithm (AIUA)

**Input:** The Structural Data \((d_i, V_\alpha, g_\alpha, m, \alpha(v_i), \text{ and } T_\alpha)\) and the Structural Expressions \((\hat{S}_N, \hat{S}_C, \text{ and } \hat{S}_G)\) of the original graph \(G\), and the Adjustment \((\delta(v_i), \delta_k(d), \delta_V(\alpha), \delta_g(\alpha), n, J_{n-c}, \text{ and } J_{c-n})\) from \(G\) to \(G'\).

**Output:** The updated two-dimensional structural entropy \(H^{T'}(G')\), the updated Structural Data \((d'_i, V'_\alpha, g'_\alpha, m', \alpha'(v_i), \text{ and } T'_\alpha)\), and the updated Structural Expressions \((\hat{S}'_N, \hat{S}'_C, \text{ and } \hat{S}'_G)\).

// Update the Structural Data
1. for each \(v_i \in \phi_\lambda\) do
2. \(d'_i := d_i + \delta(v_i);\)
3. end
4. for each \(\alpha \in \mathcal{A}\) do
5. \(V'_\alpha := V_\alpha + \delta_V(\alpha); g'_\alpha := g_\alpha + \delta_g(\alpha);\)
6. end
7. \(m' = m + n;\)
8. for each \((v, \alpha) \in J_{n-c}\) do
9. \(\alpha'(v) := \alpha;\)
10. end
11. \(T'_\alpha := T_\alpha\) for all \(\alpha \in \mathcal{A};\)
12. for each \((\alpha, v, \text{op}) \in J_{c-n}\) do
13. \(\text{if } \text{op} == + \text{ then}\)
14. \(\quad T'_\alpha := T'_\alpha \cup \{v\};\)
15. \(\text{end}\)
16. \(\text{if } \text{op} == - \text{ then}\)
17. \(\quad T'_\alpha := T'_\alpha / \{v\};\)
18. \(\text{end}\)
19. end

// Update the Structural Expressions
20. \(\hat{S}'_N := \hat{S}_N; \hat{S}'_C := \hat{S}_C; \hat{S}'_G := \hat{S}_G;\)
21. for each \(d \in \mathcal{D}\) do
22. \(\quad S'_N := S'_N + \delta_k(d)d\log d;\)
23. end
24. for each \(\alpha \in \mathcal{A}\) do
25. \(\quad \hat{S}'_C := \hat{S}'_C + (g_\alpha + \delta_g(\alpha) + V_\alpha + \delta_V(\alpha))\log(V_\alpha + \delta_V(\alpha)) - (g_\alpha + V_\alpha)\log V_\alpha;\)
26. \(\quad \hat{S}'_G := \hat{S}'_G - \delta_g(\alpha);\)
27. end

// Calculate the updated two-dimensional structural entropy
28. \(H^{T'}(G') := -\frac{1}{2m+2n}[\hat{S}'_N + \hat{S}'_C + \hat{S}'_G \log(2m + 2n)];\)
29. return \(H^{T'}(G'),\) the updated Structural Data, and the updated Structural Expressions.
Algorithm 4: Traditional Offline Algorithm (TOA)

Input : The original graph $G$ and an incremental sequence $\xi$.
Output: The updated two-dimensional structural entropy $H^T'(G')$.
1 Get the updated graph $G'$ by combining $G$ and $\xi$;
2 Construct the two-dimensional encoding tree $T'$;
3 Get the degree $d'_i$ of each node $v_i \in V'$;
4 Get the volume $V'_\alpha$ and the cut edge number $g'_\alpha$ of $\alpha \in A'$;
5 Get the total edge number $m'$ of $G'$;
6 Obtain $H^T'(G')$ via Eq. (19);
7 return $H^T'(G')$;

5. Experiments and Evaluations

In this section, we conduct extensive experiments on the application of dynamic graph real-time monitoring and community optimization. Below we first provide the description of the artificial dynamic graph dataset and the 3 real-world datasets. Then we give the experimental results and analysis.

5.1. Datasets

5.1.1. Artificial Datasets

First, we generate the initial state of the dynamic graph by utilizing the random_partition_graph method in “Networkx” [18] (a python library). This method has three parameters. The first parameter is a list of community sizes $S = [s_1, s_2, ...]$, which denotes the node number of each community of the initial state. The other two parameters are two probabilities $p_{hn}$ and $p_{ac}$. Nodes in the same community are connected with $p_{hn}$ and nodes across different communities are connected with $p_{ac}$.

Second, we generate the incremental sequences and the updated graphs by Hawkes Process [19] referring to some settings of Zuo et al. [20]. Hawkes Process [19] models discrete sequence events by assuming that historical events can influence the occurrence of current events. In this process, we first use node2vec [21] to get the embedding vectors of all nodes. Second, we randomly choose node $x$ to be the target node. With probability $p_{hn}$, we connect a new node with $x$, whose embedding vector is set to be the same as $x$'s. With probability $1 - p_{hn}$, we use the embedding vectors to calculate the conditional intensity function $\Lambda_{y_i|x}$ of $x$ and each of its non-neighboring nodes $y_i$, and then we add an edge between $x$ and $y_i$ with the conditional probabil-
Table 2: Brief Description of the Dynamic Graph Datasets.

| Dataset       | $|V_0|$  | $|E_0|$ | $E(\Delta V)$ | $E(\Delta E)$ | # of snapshots |
|---------------|--------|--------|-------------|--------------|----------------|
| Hawkes        | 2,000  | 150,187| 372         | 7,509        | 20             |
| Cit-HepPh     | 24,869 | 118,530| 2,743       | 11,406       | 20             |
| Cit-HepTh     | 308    | 335    | 3,432       | 2,522        | 20             |
| Facebook      | 13,062 | 62,348 | 22,853      | 26,487       | 20             |


\[ p_{yi|x} = \frac{\Lambda_{yi|x}}{\sum_y \Lambda_{y|x}} \]

We repeat the above two steps to generate the incremental sequence.

Finally, we describe the chosen parameter settings and the generated artificial datasets as follows.

- **Initial state settings**: The total node number is 2000, $S = [200, 300, 400, 500, 600]$, $p_{in} = 0.3$, and $p_{ac} = 0.01$.

- **Hawkes Process settings**: We set the dimension of the embedding vectors as 16 and $p_{hn}$ as 0.05.

The generated initial state, according to the above parameters, has 150187 edges. We generate a cumulative incremental sequence $\xi_{1\rightarrow T}$ including 150187 incremental edges. Next, we let $T = 20$ and divide $\xi_{1\rightarrow T}$ into 20 sub-sequences $\xi_1, \xi_2, ..., \xi_{20}$ of equal length. Then we generate each updated graph $G_t$ by combining the incremental sequence $\xi_t$ with the original graph $G_{t-1}$. The generated artificial dataset, namely the Hawkes dataset, is shown in Table 2.

### 5.1.2. Real-World Datasets

For the real-world datasets, we choose Cit-HepPh\cite{22}, Cit-HepTh\cite{22}, and Facebook\cite{23} to conduct our experiments. Cit-HepPh and Cit-HepTh are two citation networks in the field of high-energy physics phenomenology and theory from 1993 to 2003. Facebook records the establishment process of the user friendship relationship of about 52% Facebook users in New Orleans from 2006 to 2009. For each dataset, we cut out 21 consecutive snapshots (an initial state and 20 updated graphs). Since structural entropy is only defined on connected graphs, we only preserve the largest connected component for each snapshot. The used real-world datasets are briefly shown in Table 2.
5.2. Results and Analysis

5.2.1. Application: Dynamic Graph Real-Time Monitoring and Community Optimization

In this application, we incrementally optimize the community partitioning and monitor the corresponding two-dimensional structural entropy by our incremental framework to quantify the community quality in real-time for each snapshot of the dynamic graph. As shown in Fig. 8, we monitor the structural entropy on all 4 datasets using static community detection methods including Infomap [15], Louvain [16], and Leiden [17]. In every single test, we first use the chosen static method to generate the initial community partitioning. Then we use the online module, i.e., \textit{NAGA+AIUA} and \textit{NSGA+AIUA}, and the offline module, namely \textit{TOA} with the chosen static method, to respectively measure the updated two-dimensional structural entropy at each time stamp. The iteration number of \textit{NSGA} is set as 5.

Overall, the structural entropy obtained by \textit{NSGA+AIUA} based on the node offset strategy, is significantly smaller than that obtained by \textit{NSGA+AIUA} on the naive adjustment strategy, in all the 12 sets of experiments. In particular, on the Cit-HepPh, Cit-HepTh, Facebook, and Hawkes datasets, \textit{NSGA+AIUA} is able to reduce the structural entropy value by up to about 26%, 18%, 15% and 9%, respectively. This verifies that the node-shifting strategy is theoretically able to reduce the structural entropy and represents that the strategy is able to obtain significantly better coding tree/community partitioning than the plain adjustment strategy, which proves that the strategy is an effective improvement and supplement to the naive adjustment strategy.

While maintaining high computational efficiency, the online algorithm proposed in this paper still exhibits a performance that is not weaker than the offline algorithm. Specifically, both online algorithms outperform the offline algorithm in the experiments applying Leiden, where \textit{NSGA+AIUA} even achieves a structural entropy reduction of up to about 30%. In the experiments using Louvain, \textit{NSGA+AIUA} beat the offline algorithm on all datasets except Cit-HepTh. Although the offline algorithm is optimal in all four sets of experiments with Infomap, \textit{NSGA+AIUA}’s structural entropy curves on the Cit-HepPh, Facebook, and Hawkes datasets almost closely follow the offline’s while maintaining the computational efficiency advantage. Overall, these two online algorithms, especially \textit{NSGA+AIUA} based on the node-shifting strategy, significantly outperform the offline algorithm in terms
The reason is that, as the number of iterations increases, more nodes will shift to their OPC, which leads to the further reduction of the structural entropy. This experiment also demonstrates that our node-shifting adjustment strategy has excellent interpretability.

5.2.2. Hyperparameter Study

In this part, we evaluate the influence on the updated structural entropy of different iteration numbers of the node-shifting adjustment strategy. We use $NSGA + AIUA$ with iteration number $N = 3, 5, 7, 9$ to measure the mean updated structural entropy of the 20 snapshots, respectively, on each situation in Section 5.2.1. As we can see from Table 3, the updated structural entropy decreases as the number of iterations increases most of the time. The reason is that, as the number of iterations increases, more nodes will shift to their OPC, which leads to the further reduction of the structural entropy. This experiment also demonstrates that our node-shifting adjustment strategy has excellent interpretability.
Table 3: The Updated Structural Entropy by Node-Shifting Adjustment Strategy with Different Number of Iterations. (Bold Number Denotes the Lowest Structural Entropy in the Current Setting)

| # of iterations (N) | N = 3   | N = 5   | N = 7   | N = 9   |
|---------------------|---------|---------|---------|---------|
| Cit-HepPh           | Infomap | 9.7223  | 9.7193  | 9.7184  | **9.7183** |
|                     | Louvain | 10.8133 | 10.8115 | 10.8108 | **10.8102** |
|                     | Leiden  | 10.0074 | 9.9566  | 9.9370  | **9.9190** |
| Cit-HepTh           | Infomap | 7.9056  | 7.9072  | 7.9007  | **7.8993** |
|                     | Louvain | 8.1996  | 8.2023  | 8.1999  | **8.1987** |
|                     | Leiden  | 7.7155  | 7.6316  | 7.6113  | **7.6040** |
| Facebook            | Infomap | 10.2422 | 10.2355 | 10.2337 | **10.2331** |
|                     | Louvain | 11.2824 | 11.2785 | 11.2767 | **11.2757** |
|                     | Leiden  | 10.4421 | 10.3920 | 10.3872 | **10.3939** |
| Hawkes              | Infomap | 10.0029 | 10.0023 | 10.0020 | **10.0017** |
|                     | Louvain | 10.0029 | 10.0023 | 10.0020 | **10.0017** |
|                     | Leiden  | 10.5357 | 10.5286 | 10.5281 | **10.5279** |

5.2.3. Time Consumption Evaluation

Fig. 9 shows the time consumption of all used structural entropy measurement methods used above on all 4 datasets. The vertical axis in the figure represents the mean time consumption of the chosen dynamic method across all 20 snapshots. The horizontal axis represents the selected static method and its average time cost. As we can see from the results, all our proposed dynamic methods are significantly faster than the existing static methods. Specifically, for example, our dynamic methods obtain over 33x, 13x, 37x, and 7x speed up on average on Cit-hepPh, Cit-hepTh, Facebook, and Hawkes, in contrast with Infomap as the static method (the first figure in Fig. 9). In addition, the time cost of the NSGA+AIUA increases as the increase of iteration number N. The time consumption of NAGA+AIUA is always less than NAGA+AIUA with N = 7.

5.2.4. Convergence Evaluation

In this part, we conduct a statistical experiment to confirm the convergence of the Local Difference and its first-order absolute moment. We first generate artificial original graphs with increasing total edge numbers from 480 to 24000. Based on each original graph, we generate 30 incremental sequences with size n sampling from a normal distribution with a mean of 100.
and a standard deviation of 10. These incremental sequences are generated by repeatedly adding edges within a community with probability \( p_{\text{pin}} \in [0, 0.8) \), adding edges across two random communities with probability \( p_{\text{pac}} \in [0, 0.1) \), and adding nodes with probability \( p_n = 1 - p_{\text{pin}} - p_{\text{pac}} \). We then count the Local Difference and its upper bound. The results are shown in Fig. 10. As the total edge number increases from 1 to 50 times, the mean Local Difference gradually decreases by 95.98\% (from 1.08 to 0.04), respectively, and is always positive. This solidly supports the convergence of the Local Difference and its first-order absolute moment. Moreover, the Local Difference is always below its upper bound, which confirms the validity of our bound.

Figure 9: Time consumption on each dataset.

Figure 10: The statistics of the Local Difference and its upper bound.

6. Related Works

Graph Entropy. Many efforts have been devoted to measuring the information in graphs. The graph entropy was first defined and investigated by Rashevsky [2], Trucco [3], and Mowshowitz [4]. After that, a different graph entropy closely related to information and coding theory was proposed by
Körner [5]. Later, Bianconi [24] introduced the concept of “structural entropy of network ensembles,” known as the Gibbs entropy. Anand et al. [25] proposed the Shannon entropy for network ensembles. Braunstein et al. [26] proposed the von Neumann graph entropy based on the combinatorial graph Laplacian matrix. These three metrics [24, 25, 26] are defined by statistical mechanics and are used to compare different graph models. However, most of the current graph entropy measures are based on the Shannon entropy definition for probability distributions, which has significant limitations when applied to graphs [27]. Recently, many efforts have been devoted to capturing the dynamic changes of graphs, e.g., the research based on the Algorithmic Information Theory [28, 29]. The structural entropy method used in this paper proposed by Li et al. [6] provides an approach to measuring the high-dimensional information embedded in graphs and can further decode the graph’s hierarchical abstracting by an optimal encoding tree.

**Fast Computation for Graph Entropy.** Chen et al. [30] proposed a fast incremental von Neumann graph entropy computational framework, which reduces the cubic complexity to linear complexity in the number of nodes and edges. Liu et al. [31, 32] used the structural entropy [6] as a proxy of von Neumann graph entropy for the latter’s fast computation, and also implemented an incremental method for one-dimensional structural entropy. In this paper, we mainly focus on the incremental computation for two-dimensional structural entropy based on our dynamic adjustment strategies for encoding trees.

### 7. Conclusion

In this paper, we propose two novel dynamic adjustment strategies, namely the naive adjustment strategy and the node-shifting adjustment strategy, to analyze the updated structural entropy and incrementally adjust the original community partitioning towards a lower structural entropy. We also implement an incremental framework, i.e., supporting the real-time measurement of the updated two-dimensional structural entropy. In the future, we aim to develop dynamic adjustment strategies for hierarchical community partitioning and incremental measurement algorithms for higher dimensional structural entropy.
References

[1] C. Shannon, The lattice theory of information, Transactions of the IRE Professional Group on Information Theory 1 (1) (1953) 105–107.

[2] N. Rashevsky, Life, information theory, and topology, The Bulletin of Mathematical Biophysics 17 (3) (1955) 229–235.

[3] E. Trucco, A note on the information content of graphs, The Bulletin of Mathematical Biophysics 18 (2) (1956) 129–135.

[4] A. Mowshowitz, Entropy and the complexity of graphs, Tech. rep. (1967).

[5] J. Körner, Coding of an information source having ambiguous alphabet and the entropy of graphs, in: Proceedings of the Prague Conference on Information Theory, 1973, pp. 411–425.

[6] A. Li, Y. Pan, Structural information and dynamical complexity of networks, IEEE Transactions on Information Theory 62 (6) (2016) 3290–3339.

[7] A. Li, Mathematical Principles of Information World: Calculuses of Information, Preprint., 2022.

[8] A. Li, X. Yin, B. Xu, D. Wang, J. Han, Y. Wei, Y. Deng, Y. Xiong, Z. Zhang, Decoding topologically associating domains with ultra-low resolution hi-c data by graph structural entropy, Nature Communications 9 (1) (2018) 1–12.

[9] Y. W. Zhang, M. B. Wang, S. C. Li, Supertad: robust detection of hierarchical topologically associated domains with optimized structural information, Genome Biology 22 (1) (2021) 1–20.

[10] Y. Liu, J. Liu, Z. Zhang, L. Zhu, A. Li, Rem: From structural entropy to community structure deception, in: Proceedings of the NeurIPS, Vol. 32, 2019, pp. 12918–12928.

[11] A. Li, X. Zhang, Y. Pan, Resistance maximization principle for defending networks against virus attack, Physica A: Statistical Mechanics and its Applications 466 (2017) 211–223.
[12] G. Luo, J. Li, J. Su, H. Peng, C. Yang, L. Sun, P. S. Yu, L. He, Graph entropy guided node embedding dimension selection for graph neural networks, in: Proceedings of the IJCAI, 2021, pp. 2767–2774.

[13] J. Wu, X. Chen, K. Xu, S. Li, Structural entropy guided graph hierarchical pooling, in: Proceedings of the ICML, Vol. 162, 2022, pp. 24017–24030.

[14] F. Harary, G. Gupta, Dynamic graph models, Mathematical and Computer Modelling 25 (7) (1997) 79–87.

[15] M. Rosvall, C. T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proceedings of the national academy of sciences 105 (4) (2008) 1118–1123.

[16] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment 2008 (10) (2008) P10008.

[17] V. A. Traag, L. Waltman, N. J. Van Eck, From louvain to leiden: guaranteeing well-connected communities, Scientific reports 9 (1) (2019) 5233.

[18] A. Hagberg, P. Swart, D. S Chult, Exploring network structure, dynamics, and function using networkx, Tech. rep. (2008).

[19] A. G. Hawkes, Spectra of some self-exciting and mutually exciting point processes, Biometrika 58 (1) (1971) 83–90.

[20] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, J. Wu, Embedding temporal network via neighborhood formation, in: Proceedings of the SIGKDD, 2018, pp. 2857–2866.

[21] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the SIGKDD, 2016, pp. 855–864.

[22] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: densification laws, shrinking diameters and possible explanations, in: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, 2005, pp. 177–187.
[23] B. Viswanath, A. Mislove, M. Cha, K. P. Gummadi, On the evolution of user interaction in Facebook, in: Proceedings of the 2nd ACM workshop on Online social networks, 2009, pp. 37–42.

[24] G. Bianconi, Entropy of network ensembles, Physical Review E 79 (3) (2009) 036114.

[25] K. Anand, G. Bianconi, Entropy measures for networks: Toward an information theory of complex topologies, Physical Review E 80 (4) (2009) 045102.

[26] S. L. Braunstein, S. Ghosh, S. Severini, The laplacian of a graph as a density matrix: a basic combinatorial approach to separability of mixed states, Annals of Combinatorics 10 (3) (2006) 291–317.

[27] H. Zenil, N. A. Kiani, J. Tegnér, Low-algorithmic-complexity entropy-deceiving graphs, Physical Review E 96 (1) (2017) 012308.

[28] H. Zenil, N. A. Kiani, A. A. Zea, J. Tegnér, Causal deconvolution by algorithmic generative models, Nature Machine Intelligence 1 (1) (2019) 58–66.

[29] H. Zenil, N. A. Kiani, F. Marabita, Y. Deng, S. Elias, A. Schmidt, G. Ball, J. Tegner, An algorithmic information calculus for causal discovery and reprogramming systems, Iscience 19 (2019) 1160–1172.

[30] P.-Y. Chen, L. Wu, S. Liu, I. Rajapakse, Fast incremental von Neumann graph entropy computation: Theory, algorithm, and applications, in: Proceedings of the ICML, 2019, pp. 1091–1101.

[31] X. Liu, L. Fu, X. Wang, C. Zhou, On the similarity between von Neumann graph entropy and structural information: Interpretation, computation, and applications, IEEE Transactions on Information Theory 68 (4) (2022) 2182–2202.

[32] X. Liu, L. Fu, X. Wang, Bridging the gap between von Neumann graph entropy and structural information: Theory and applications, in: Proceedings of the WWW, 2021, pp. 3699–3710.