Incremental Measurement of Structural Entropy for Dynamic Graphs

Runze Yang[a], Hao Peng[b,*], Chunyang Liu[c], Angsheng Li[a]

[a]*State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing, 100191, China*
[b]*School of Cyber Science and Technology, Beihang University, Beijing, 100191, China*
[c]*Didi Chuxing, Beijing, 100193, 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 best community partitioning for each snapshot. However, the current methods do not support dynamic encoding tree updating and incremental structural entropy computation. To address this issue, we propose *Incre-2dSE*, a novel incremental measurement framework that dynamically adjusts the community partitioning and efficiently computes the updated structural entropy for each updated graph. Specifically, *Incre-2dSE* includes incremental 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 support theoretical analysis of updated structural entropy and incrementally optimize community partitioning towards a lower structural entropy. We conduct extensive experiments on 3 artificial datasets generated by *Hawkes Process* and 3 real-world datasets. Experimental results confirm that our incremental 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

*Corresponding author
Email address: penghao@buaa.edu.cn (Hao Peng)

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] have been 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, 14], 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 total node degree and cut edge number of each community are counted to compute the structural entropy of each non-root node in the encoding tree. Thirdly, 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 graphs during time series [15]. To efficiently measure the quality of evolving 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 due to two challenges. 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[1], namely *the naive*

[1] "Two-dimensional encoding tree" means the height of the encoding tree is restricted to 2. The corresponding structural entropy of the two-dimensional encoding tree is named "two-dimensional structural entropy".

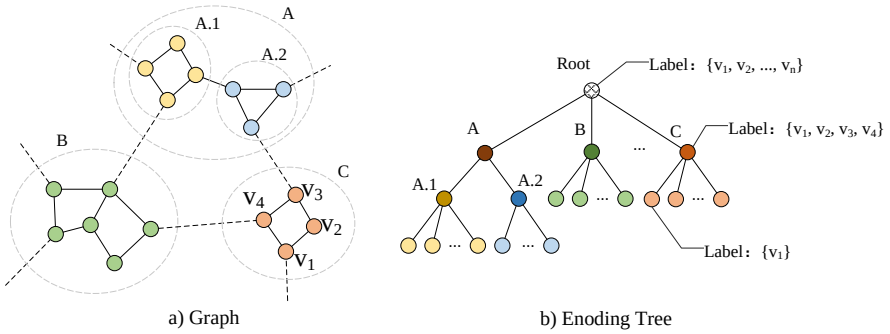

Figure 1: a) A graph containing three communities A, B, and C, where A is divided into two sub-communities A.1 and A.2. b) An encoding tree of the left graph. Each leaf node corresponds to a single graph node. Each branch node corresponds to a community. The root node corresponds to the graph node set.

adjustment strategy and *the node-shifting adjustment strategy*. The former strategy maintains the old community partitioning, and supports theoretical 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 challenge is the high time complexity of structural entropy computation by the traditional definition. To tackle this problem, we design an incremental framework, namely *Incre-2dSE*, for efficiently measuring the updated two-dimensional structural entropy. To be specific, *Incre-2dSE* first utilizes the two dynamic adjustment strategies to generate *Adjustments*, i.e., the changes of important statistics from the original graph to the updated graph and then uses the Adjustments to calculate the updated structural entropy by newly designed incremental formula. Additionally, we also generalize our incremental methods to undirected weighted graphs and conduct a detailed discussion on the calculation of one-dimensional structural entropy for directed weighted graphs.

We conduct extensive experiments on 3 artificial dynamic graph datasets generated by *Hawkes Process* and 3 real-world datasets on the application of real-time monitoring of community partitioning quality (two-dimensional structural entropy) and community optimization. Comprehensive experimental results show that our methods effectively capture the community evolution features and significantly reduce the time consumption with great interpretability. All source code and data of this project are publicly available at `https://github.com/SELGroup/IncreSE`.

The main contributions of this paper are as follows:

- Proposing two dynamic adjustment strategies for two-dimensional encoding trees to avoid the reconstruction of the encoding tree for every updated graph.
- Designing an incremental framework for efficiently measuring the updated two-dimensional structural entropy with low time complexity.
- Extending the proposed methods to weighted graphs and providing new incremental computation methods for directed weighted graphs.
- Conducting extensive experiments on artificial and real-world datasets to demonstrate the effectiveness and efficiency of our method in dynamic measurement of community partitioning quality.

The article is structured as follows: Section 2 outlines the definitions and notations. Section 3 describes the dynamic adjustment strategies. The algorithms are detailed in Section 4, and Section 5 gives discussion on more complex graphs. The experiments are discussed in Section 6. Section 7 presents the related works before concluding the paper in section 8.

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 represented by* $\xi = \{< (v_1, u_1), op_1 >, < (v_2, u_2), op_2 > , ..., < (v_n, u_n), op_n >\}$*, where* (v_i, u_i) *denotes an incremental edge* e_i *with two endpoints* v_i *and* u_i*. The operator* $op_i \in \{+, -\}$ *represents that the edge* e_i *will be added to or removed from a graph. 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* $\mathcal{G} = \{G_0, G_1, ..., G_T\}$*.* $G_0 = (\mathcal{V}_0, \mathcal{E}_0)$ *denotes the initial state and* $G_t = (\mathcal{V}_t, \mathcal{E}_t)$ *denotes the updated graph at time* t $(1 \leq t \leq T)$*. To describe the temporal dynamic graph, we suppose that an incremental sequence* ξ_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* ξ_t *with* G_{t-1}*, i.e.,* $G_t :=$

Table 1: Glossary of Notations.

Notation	Description
G	Graph
$\mathcal{V}; \mathcal{E}$	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
λ	The root node of an encoding tree
α	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_α	The label of α, i.e., the node community corresponding to α
V_α	The volume of T_α
g_α	The cut edge number of T_α
$\mathcal{G}$	Dynamic graph
G_0	Initial state of a dynamic graph
G_t	The updated graph at time t
ξ_t	Incremental sequence at time t
$\xi_{1\to t}$	Cumulative incremental sequence at time t
n	Incremental size
$\delta(v)$	The degree incremental of v
$\delta_V(\alpha)$	The volume incremental of T_α
$\delta_g(\alpha)$	The cut edge number incremental of T_α
ϕ_λ	The degree-changed node set
$\mathcal{A}$	The set of 1-height tree nodes whose volume or cut edge number change
$H^{\mathcal{T}}(G)$	The structural entropy of G by $\mathcal{T}$
$H^{\mathcal{T}}_{GI}(G, n)$	Global Invariant with incremental size n
ΔL	Local Difference, i.e., the approximation error between $H^{\mathcal{T}}(G)$ and $H^{\mathcal{T}}_{GI}(G, n)$

$CMB(G_{t-1}, \xi_t)$. We further define the cumulative incremental sequence at time t, denoted by $\xi_{1\to t}$, as the sequence formed by sequentially concatenating sequences $\xi_1, \xi_2, ..., \xi_t$, and then we have $G_t := CMB(G_0, \xi_{1\to t})$.

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 $\mathcal{T}$ of a graph $G = (\mathcal{V}, \mathcal{E})$ is an undirected tree that satisfies the following properties:*

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

Definition 4 (Structural Entropy)**.** *The structural entropy is defined by Li and Pan [6]. 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 $\mathcal{T}$, the structural entropy of G by $\mathcal{T}$ is defined as:*

$$H^{\mathcal{T}}(G) = \sum_{\alpha \in \mathcal{T}, \alpha \neq \lambda} -\frac{g_\alpha}{2m} \log \frac{V_\alpha}{V_{\alpha^-}}, \tag{1}$$

where m is the total edge number of G; g_α is the cut edge number of T_α, i.e., the number of edges between nodes in and not in T_α; V_α is the volume of T_α, i.e., the sum of the degrees of all nodes in T_α; $\log(\cdot)$ denotes logarithm with a base of 2. We name $H^{\mathcal{T}}(G)$ as the K-dimensional structural entropy if $\mathcal{T}$'s height is K.

The graph structural entropy of G is defined as:

$$H(G) = \min_{\mathcal{T}} \left\{ H^{\mathcal{T}}(G) \right\}, \tag{2}$$

where $\mathcal{T}$ ranges over all possible encoding trees.

If the height of $\mathcal{T}$ is restricted to K, the K-dimensional graph structural entropy of G is defined as:

$$H^K(G) = \min_{\mathcal{T}} \left\{ H^{\mathcal{T}}(G) \right\}, \tag{3}$$

where $\mathcal{T}$ 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.

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 which leads to lower structural entropy, and provide its theoretical proof. Finally, we give further discussion between the two proposed dynamic adjustment strategies.

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 realize the incremental computation of updated structural entropy. Finally, we analyze the boundedness and convergence of the Local Difference.

3.1.1. Strategy Description

Before we introduce the naive adjustment strategy, we first discuss the form of the two-dimensional encoding trees in detail and the formula of their corresponding structural entropy. For all possible two-dimensional encoding trees, whenever there is a leaf node γ ($T_\gamma = \{v_k\}$) whose height is 1 (like Fig. 2(b)), we can connect it with a child node γ^+ with the same label $T_{\gamma^+} = \{v_k\}$ to make all leaf nodes have height 2 (Fig. 2(c)). At this time, the structural entropy remains unchanged since the additional term induced by γ^+, i.e. $-\frac{d_k}{2m}\log\frac{d_k}{d_k}$, equals to 0.

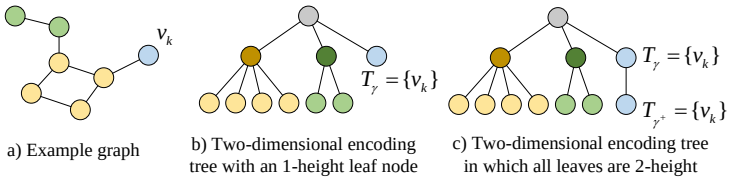

Figure 2: An example graph with its two equivalent two-dimensional encoding trees.

In other words, the two encoding trees in Fig. 2 are equivalent. Therefore, in this paper, we only consider the two-dimensional encoding trees where the

height of all leaf nodes is 2 for the sake of brevity. Given a graph G and its two-dimensional encoding tree $\mathcal{T}$, the two-dimensional structural entropy of G by $\mathcal{T}$ can uniformly be formulated as:

$$H^{\mathcal{T}}(G) = \sum_{\alpha_i \in A} (-\frac{g_{\alpha_i}}{2m} \log \frac{V_{\alpha_i}}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m} \log \frac{d_j}{V_{\alpha_i}}), \tag{4}$$

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

Now we present the description of the naive adjustment strategy. This 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. 3(a)), and u corresponds to a leaf node η in the two-dimensional encoding tree, i.e., $T_\eta = \{u\}$ (shown in Fig. 3(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. 3(d)), instead of another 1-height node (β in Fig. 3(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_α in Fig. 3(c)), rather than another arbitrary community (T_β in Fig. 3(e))*. Obviously, we can get the updated encoding tree, i.e., 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.

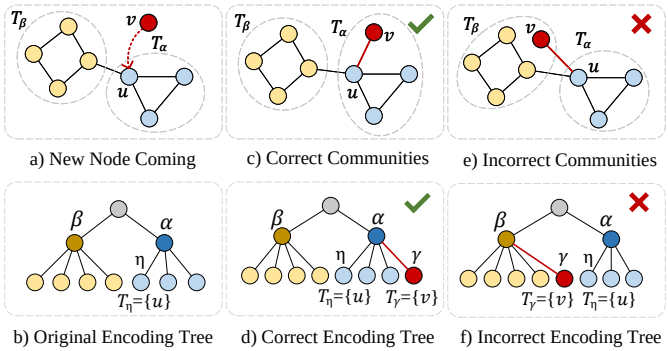

Figure 3: An example of the node strategy for adjusting two-dimensional encoding trees.

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:*

$$H^{\mathcal{T}'}_{v\to\alpha}(G') < H^{\mathcal{T}'}_{v\to\beta\neq\alpha}(G'), \tag{5}$$

where $H^{\mathcal{T}'}_{v\to\alpha}(G')$ denotes the updated structural entropy when the new node v is assigned to u's community T_α, i.e., $\{v\} \subset T_\alpha$, and $H^{\mathcal{T}'}_{v\to\beta\neq\alpha}(G')$ 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:

$$\begin{aligned}\Delta H^{\mathcal{T}'}(G') =& H^{\mathcal{T}'}_{v\to\alpha}(G') - H^{\mathcal{T}'}_{v\to\beta\neq\alpha}(G') \\ =& \frac{1}{2m+2}[\log\frac{V_\alpha+2}{2m+2} + (g_\alpha - V_\alpha)\log\frac{V_\alpha+1}{V_\alpha+2} - (g_\beta - V_\beta)\log\frac{V_\beta}{V_\beta+1}].\end{aligned} \tag{6}$$

Here we define:

$$f_1(g,V) = (g-V)\log\frac{V+1}{V+2}, \tag{7}$$

$$f_2(g,V) = -(g-V)\log\frac{V}{V+1}. \tag{8}$$

Let θ $(0 \leq \theta < 1)$ be the minimum proportion of the in-community edges in each community, i.e.,

$$\theta = \min_\alpha\{\frac{V_\alpha - g_\alpha}{V_\alpha}\}. \tag{9}$$

Since $1 \leq V < 2m$ and $0 < g \leq (1-\theta)V$, we have:

$$f_1(g,V) < -V\log\frac{V+1}{V+2} < \frac{1}{\ln 2} = \log e, \tag{10}$$

$$f_2(g,V) \leq \theta V\log\frac{V}{V+1} \leq \theta\log\frac{1}{2} = -\theta. \tag{11}$$

So

$$\begin{aligned}\Delta H^{\mathcal{T}'}(G') <& \frac{1}{2m+2}(\log\frac{V_\alpha+2}{2m+2} + \log e - \theta) \\ =& \frac{1}{2m+2}\log(\frac{V_\alpha+2}{2m+2}\cdot 2^{-\theta}e).\end{aligned} \tag{12}$$

Therefore, if the following condition holds:

$$\frac{2m+2}{V_\alpha+2} \geq \max\{2^{-\theta}e\} = e, \tag{13}$$

then Eq. (5) 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

In this part, we introduce two quantities, Global Invariant and Local Difference, to realize the approximation and the fast incremental calculation of the updated structural entropy by naive adjustment strategy. When an incremental sequence ξ with size n is applied to a graph G, resulting in a new graph G' and its corresponding two-dimensional encoding tree $\mathcal{T}'$ using the naive adjustment strategy, the updated two-dimensional structural entropy can be expressed as:

$$H^{\mathcal{T}'}(G') = \sum_{\alpha_i \in A}(-\frac{g'_{\alpha_i}}{2m+2n}\log\frac{V'_{\alpha_i}}{2m+2n} + \sum_{v_j \in T_{\alpha_i}} -\frac{d'_j}{2m+2n}\log\frac{d'_j}{V'_{\alpha_i}}). \tag{14}$$

An intuitive way to calculate the updated two-dimensional structural entropy is to update the variables in Eq. (4) and then compute via the updated formula Eq. (14). However, the incremental size n affects all terms in the summation equation in Eq. (14). Therefore, the updating and calculation process costs at least $O(|\mathcal{V}|)$, which is huge when the graph becomes extremely large. So how can we find an incremental formula with a smaller time complexity only related to the incremental size n? An intuitive attempt is to make a difference between the updated structural entropy and the original one to try to compute the incremental entropy in $O(n)$. Nevertheless, the fact that m changes to $m+n$ in all terms of Eq. (14) makes it difficult to derive a concise formula of $O(n)$ from the difference equation.

To address this issue, we here introduce Global Invariant and Local Difference. We define the Global Invariant (Definition 5) as an approximate updated structural entropy that only updates m to $m+n$ in Eq. (4) and keeps other variables unchanged. The Local Difference (Definition 6) is defined as the difference between the updated structural entropy and the Global

Invariant, which can also be regarded as the approximate error. Obviously, 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 the necessary incremental changes. Overall, the updated two-dimensional structural entropy can be calculated in $O(n)$ by computing and summing up the Global Invariant and the Local Difference. In the experimental part, we use a more explicit and practiced form of the structural entropy update formula (Eq. (55)) derived from the Global Invariant and the Local Difference.

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

$$\begin{aligned} H_{GI}^{\mathcal{T}}(G,n) &= \sum_{\alpha_i \in A} (-\frac{g_{\alpha_i}}{2m+2n} \log \frac{V_{\alpha_i}}{2m+2n} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m+2n} \log \frac{d_j}{V_{\alpha_i}}) \\ &= -\frac{1}{2m+2n}(S_N + S_C + S_G), \end{aligned} \tag{15}$$

where

$$S_N = \sum_{v_i \in \mathcal{V}} d_i \log d_i, \tag{16}$$

$$S_C = \sum_{\alpha_i \in A} (g_{\alpha_i} - V_{\alpha_i}) \log V_{\alpha_i}, \tag{17}$$

$$S_G = -\sum_{\alpha_i \in A} g_{\alpha_i} \log(2m+2n). \tag{18}$$

Definition 6 (Local Difference)**.** *Given the updated graph G', the updated two-dimensional encoding tree $\mathcal{T}'$, 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:*

$$\Delta L = H^{\mathcal{T}'}(G') - H_{GI}^{\mathcal{T}}(G,n) = -\frac{1}{2m+2n}(\Delta S_N + \Delta S_C + \Delta S_G), \tag{19}$$

where

$$\Delta S_N = \sum_{v_k \in \phi_\lambda} [(d_k + \delta(v_k)) \log(d_k + \delta(v_k)) - d_k \log d_k], \tag{20}$$

$$\Delta S_C = \sum_{\alpha \in \mathcal{A}} [(g_\alpha + \delta_g(\alpha) - V_\alpha - \delta_V(\alpha)) \log(V_\alpha + \delta_V(\alpha)) - (g_\alpha - V_\alpha) \log V_\alpha], \tag{21}$$

$$\Delta S_G = - \sum_{\alpha \in \mathcal{A}} \delta_g(\alpha) \log(2m + 2n). \tag{22}$$

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 $g'_\alpha - g_\alpha$, ϕ_λ denotes the set of nodes that have changes in degree $\{v_k \in \mathcal{V}' | \delta(v_k) \neq 0\}$, and $\mathcal{A}$ denotes the set of 1-height tree nodes that have changes in V_α or g_α, i.e., $\mathcal{A} = \{\alpha \in A' | \delta_V(\alpha) \neq 0$ or $\delta_g(\alpha) \neq 0\}$.

3.1.3. Boundedness Analysis

According to Eq. (19), the bounds of ΔL can be obtained by analyzing its components, namely ΔS_N, ΔS_C and ΔS_G. First, we analyze the maximum and minimum values of ΔS_N. We define

$$s_N(d, x) = (d + x) \log(d + x) - d \log d. \tag{23}$$

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

$$\Delta S_N \leq 2 s_N(d_m, n), \tag{24}$$

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, ΔS_N is minimized:

$$\Delta S_N \geq n s_N(1, 0). \tag{25}$$

Second, we analyze the bounds of ΔS_C and ΔS_G. For convenience, we define

$$\Delta S_{CG} = \Delta S_C + \Delta S_G. \tag{26}$$

We commence by analyzing the bound of ΔS_{CG} when adding one new edge. If a new edge is added between two communities T_{α_1} and T_{α_2}, we can get

$$\begin{aligned} \Delta S_{CG} =& (g_{\alpha_1} - V_{\alpha_1}) \log(V_{\alpha_1} + 1) - (g_{\alpha_1} - V_{\alpha_1}) \log V_{\alpha_1} \\ &+ (g_{\alpha_2} - V_{\alpha_2}) \log(V_{\alpha_2} + 1) - (g_{\alpha_2} - V_{\alpha_2}) \log V_{\alpha_2} - 2 \log(2m + 2). \end{aligned} \tag{27}$$

Thus we have

$$\Delta S_{CG} \geq 2V_m \log(\frac{V_m}{V_m+1}) - 2\log(2m+2), \tag{28}$$

and

$$\Delta S_{CG} \leq -2\log(2m+2), \tag{29}$$

where V_m denotes the maximum volume of all T_α. If a new edge is added within a single community T_α (or a new node is connected with an existing node in T_α), we have

$$\Delta S_{CG} = (g_\alpha - V_\alpha - 2)\log(V_\alpha + 2) - (g_\alpha - V_\alpha)\log V_\alpha. \tag{30}$$

Then we can obtain

$$\Delta S_{CG} \geq -(V_m+2)\log(V_m+2) + V_m \log V_m, \tag{31}$$

and

$$\Delta S_{CG} \leq -2\log(V_{min}+2), \tag{32}$$

where V_{min} denotes the minimum volume of all T_α. We next analyze the bound of ΔS_{CG} when adding n new edges. When the n edges are all between the two communities with the largest volume, we have:

$$\begin{aligned} \Delta S_{CG} \geq & 2V_m \log(\frac{V_m}{V_m+n}) - 2n\log(2m+2n) \\ > & -2n - 2n\log(2m+2n), \end{aligned} \tag{33}$$

and ΔS_{CG} takes the minimum value:

$$\Delta S_{CGmin} = -2n - 2n\log(2m+2n). \tag{34}$$

When each of the n edges is added within n communities with the smallest volume, respectively, ΔS_{CG} takes its maximum value:

$$\Delta S_{CGm} = -2n\log(V_{min}+2). \tag{35}$$

Finally, we can get a lower bound of ΔL as

$$\mathrm{LB}(\Delta L) = -\frac{1}{2m+2n}(2s_N(d_m, n) + \Delta S_{CGm})$$

$$=\frac{1}{m+n}[d_m \log d_m - (d_m+n)\log(d_m+n) + n\log(V_{min}+2)]. \tag{36}$$

An upper bound of ΔL is as follows:

$$\begin{aligned}\mathrm{UB}(\Delta L) &= -\frac{1}{2m+2n}(ns_N(1,0) + \Delta S_{CGmin}) \\ &= \frac{n\log(m+n)+\frac{5}{2}n}{m+n}.\end{aligned} \tag{37}$$

Discussion: The boundedness analysis gives the lower and upper bound of the Local Difference. This suggests that when we compute Global Invariant to quickly get an approximate value of the updated structural entropy, the approximate error is bounded and decreases as the graph grows larger and thus the validity and accuracy of the approximation are guaranteed.

3.1.4. Convergence Analysis

In this section, we 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 $O(\frac{\log m}{m})$, represented as:*

$$\Delta L = O(\frac{\log m}{m}). \tag{38}$$

Proof. The lower bound of ΔL is given by:

$$\begin{aligned}\mathrm{LB}(\Delta L) =& \frac{d_m \log d_m - (d_m+n)\log(d_m+n)}{m+n} + \frac{n\log(V_{min}+2)}{m+n} \\ \geq& \frac{m\log m - (m+n)\log(m+n)}{m+n} + \frac{n\log(2+2)}{m+n}, \\ \geq& \frac{1}{m+n}[\log(1-\frac{n}{m+n})^m - n\log(m+n)] \\ =& O(\frac{\log m}{m}).\end{aligned} \tag{39}$$

Similarly, the upper bound is given by:

$$\begin{aligned}\mathrm{UB}(\Delta L) =& \frac{n\log(m+n)+\frac{5}{2}n}{m+n}\\ =& O(\frac{\log m}{m}).\end{aligned} \tag{40}$$

Since

$$\mathrm{LB}(\Delta L) \leq \Delta L \leq \mathrm{UB}(\Delta L), \tag{41}$$

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 $O(\frac{\log m}{m})$.

Definition 7. *Let X be a random variable representing the incremental size n. We remind that $\mathbb{E}[X] = \overline{n}$.*

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

$$\mathbb{E}[|\Delta L|] = O(\frac{\log m}{m}). \tag{42}$$

Proof. We can represent the expectation of the lower bound of ΔL as:

$$\begin{aligned}\mathbb{E}[|\mathrm{LB}(\Delta L)|] =& \mathbb{E}[|\frac{d_m\log d_m-(d_m+X)\log(d_m+X)}{m+X}+\frac{n\log(V_{min}+2)}{m+X}|]\\ \leq& \mathbb{E}[\frac{(m+X)\log(m+X)-m\log m}{m+X}]+\mathbb{E}[\frac{X\log(m+2)}{m+X}]\\ \leq& \frac{m\log m-(m+\overline{n})\log(m+\overline{n})}{m+\overline{n}}+\frac{\overline{n}\log(m+2)}{m+\overline{n}}\\ =& O(\frac{\log m}{m}).\end{aligned} \tag{43}$$

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

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

$$\leq \frac{\overline{n}\log(m+\overline{n}) + \frac{5}{2}\overline{n}}{m+\overline{n}}$$
$$= O(\frac{\log m}{m}). \tag{44}$$

Finally, since

$$0 \leq \mathbb{E}[|\Delta L|] \leq \max\{\mathbb{E}[|\mathrm{LB}(\Delta L)|], \mathbb{E}[|\mathrm{UB}(\Delta L)|]\}, \tag{45}$$

Theorem 3 is proved. □

Discussion: The convergence analysis gives proof of the convergence of the Local Difference and its first-order absolute moment. That is, when we use Global Invariant to approximate the updated structural entropy, the approximate error and its absolute value's expectation both converge to 0 *no slower than* $O(\frac{\log m}{m})$ *as the graph edge number* m *grows larger, suggesting a high level of confidence that our approximation is reliable. Additionally, the convergence analysis also demonstrates that the updated structural entropy is mainly contributed by the incremental size* n *other than the position of the incremental edges when the graph is extremely large. It is because when the graph grows larger, we can approximate the updated structural entropy with an approximate error that converges to* 0 *by simply changing* m *to* $m+n$*.*

3.1.5. Limitations

The limitations of the naive adjustment strategy are listed below. First, this strategy cannot deal with multiple incremental edges at the same time, e.g. a new node appears connecting with two different existing nodes. An alternative solution is to arrange all incremental edges at a certain time stamp into a sequence with which we can add the edges one by one while keeping the connectivity of the graph. In this way, the community of newly introduced nodes is inevitably related to the input order of the incremental edges. Second, it cannot handle edge or node deletions. Third, the community of the existing nodes remains unchanged, which is sub-optimal in most cases.

3.2. Node-Shifting Adjustment Strategy

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 (Definition 8) 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. Therefore, the node-shifting adjustment strategy fully overcomes the limitations of the naive adjustment strategy listed in Section 3.1.5. 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 greedily. Finally, we discuss the limitations of this strategy.

3.2.1. Strategy Description

We first define the optimal preference community (OPC) (Definition 8) as the best community for a target node, i.e., if the target node moves into its OPC, the overall two-dimensional structural entropy must be the lowest compared to other community other than OPC. Then the node-shifting adjustment strategy can be described 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).*

Definition 8 (Optimal Preference Community (OPC))**.** *Given a graph* $G = (\mathcal{V}, \mathcal{E})$ *and a target node* $v_t \in \mathcal{V}$*, the optimal preference community of* v_t *is defined as the community* T_{α^*} *in which*

$$\alpha^* = \begin{cases} \arg\min_\alpha[(g_\alpha - V_\alpha)\log\frac{V_\alpha}{V_\alpha+d_t} + 2d^{(\alpha)}\log\frac{V_\alpha+d_t}{2m}], & v_t \notin T_\alpha; \\ \arg\min_\alpha[(g_\alpha - V_\alpha + d_t + d^{(\alpha)})\log\frac{V_\alpha-d_t}{V_\alpha} + 2d^{(\alpha)}\log\frac{V_\alpha}{2m}], & v_t \in T_\alpha, \end{cases} \tag{46}$$

where $d^{(\alpha)}$ *denotes the number of the edges connected between* v_t *and* T_α*.*

Fig. 4 and Fig. 5 give examples to illustrate the node-shifting adjustment strategy in different situations. Fig. 4 shows how incremental edges affect community partitioning. In Fig. 4(a), the graph is divided into 2 communities T_α and T_β. In Fig. 4(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_α to T_β (denoted by the red arrow). In Fig. 4(c), the shifted node in the previous step "sends messages" (red dotted arrows) to its neighbors in T_α (red outlined).

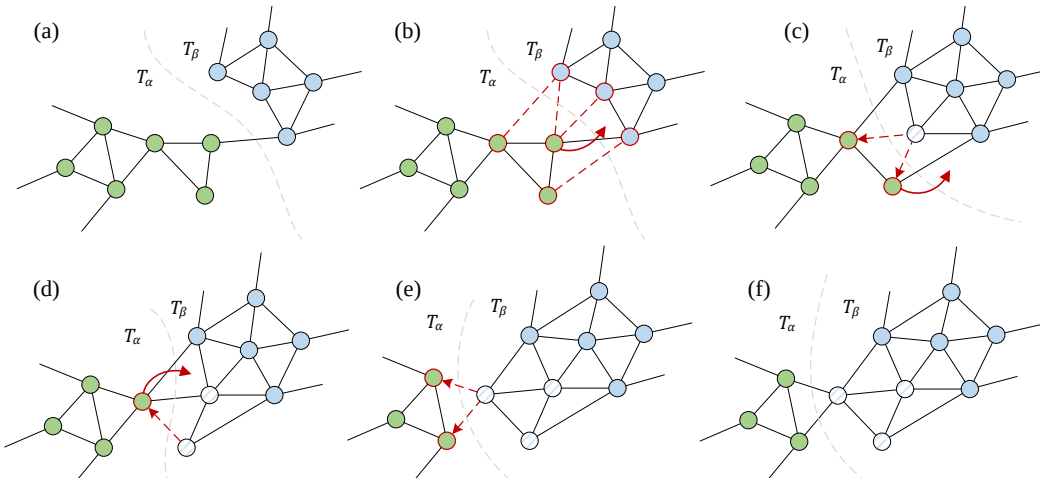

Figure 4: An example of the node-shifting adjustment strategy for adding new edges.

The nodes that received the message (red outlined) are then checked for shifting. At this time, another green node moves into T_β. In Fig. 4(d)-(e), the graph follows the above process to continue the iterative update. The final community partitioning is shown in Fig. 4(f). Fig. 5 shows how new nodes are assigned to communities. Fig. 5(a) gives a 7 nodes graph with 2 communities. In Fig. 5(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_α because T_α is determined as its OPC. Also, the lower two new nodes are moved into their OPCs. In Fig. 5(c), the new involved nodes (red outlined) are checked. Fig. 5(d) shows the final state of this node-shifting process.

3.2.2. Theoretical Proof

In this part, we provide a simplified model (Fig. 6) to theoretically derive the OPC's solution formula (Eq. (46)). In the graph of this model, there exists r communities $T_{\alpha_1}, T_{\alpha_2}, ..., T_{\alpha_r}$. 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_{α_i} is denoted by $d^{(i)}$. The volume and the number of the cut edges of T_{α_i} are denoted by V_i and g_i, respectively. Then we have Theorem 4.

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*

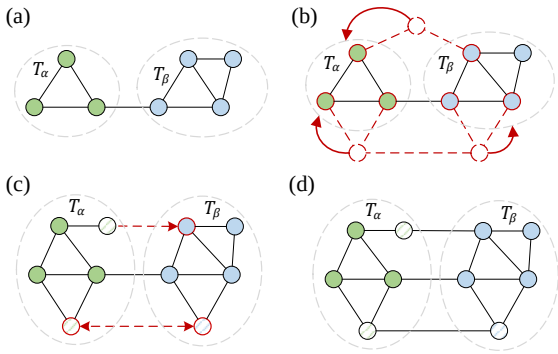

Figure 5: An example of the node-shifting adjustment strategy for adding new nodes.

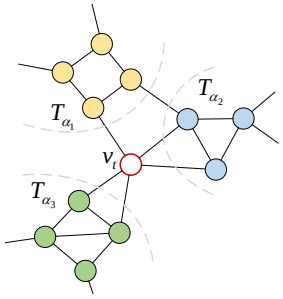

Figure 6: A simplified model for theoretical analysis.

$T_{\alpha_{k^*}}$ *where*

$$k^* = \arg\min_k [(g_k - V_k)\log\frac{V_k}{V_k + d_t} + 2d^{(k)}\log\frac{V_k + d_t}{2m}]. \tag{47}$$

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

$$\begin{aligned} H_k = &\sum_{\alpha_i \neq \alpha_k} (-\frac{g_i}{2m}\log\frac{V_i}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m}\log\frac{d_j}{V_i}) + (-\frac{g_k + d_t - 2d^{(k)}}{2m}\log\frac{V_k + d_t}{2m} \\ &+ \sum_{v_q \in T_{\alpha_k}/\{v_t\}} -\frac{d_q}{2m}\log\frac{d_q}{V_k + d_t} - \frac{d_t}{2m}\log\frac{d_t}{V_k + d_t}). \end{aligned} \tag{48}$$

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

$$k^* = \arg\min_k H_k. \tag{49}$$

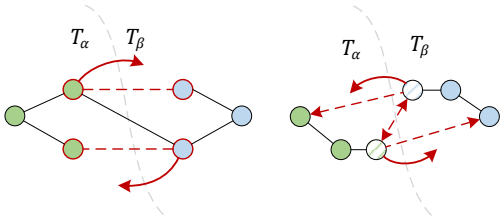

Figure 7: An example where the node-shifting adjustment strategy does not converge. The left is a graph added with two incremental edges which cause two nodes to shift. The right shows the updated communities after the first iteration and the future movement at the second iteration. After the second iteration, the graph becomes the left again.

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

$$\tilde{H} = \sum_{\alpha_i}(-\frac{g_i}{2m}\log\frac{V_i}{2m} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{2m}\log\frac{d_j}{V_i}) + (-\frac{d_t}{2m}\log\frac{d_t}{2m} - \frac{d_t}{2m}\log\frac{d_t}{d_t}). \tag{50}$$

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

$$\begin{aligned} k^* &= \arg\min_k 2m(H_k - \tilde{H}) \\ &= \arg\min_k [(g_k - V_k)\log\frac{V_k}{V_k + d_t} + 2d^{(k)}\log\frac{V_k + d_t}{2m}]. \end{aligned} \tag{51}$$

Therefore Theorem 4 is proved. □

In practice, all nodes belong to their communities. We can first move the target node out of its community, and then use Eq. (47) to determine the OPC. This process is equivalent to directly using Definition 8 without moving out the target node.

3.2.3. Limitations

The limitations of the node-shifting adjustment strategy are listed below. First, it is hard to give the bound of the gap between the Global Invariant and the updated structural entropy for further theoretical analysis. Second, the node-shifting adjustment strategy may not converge in some cases (Fig. 7 gives an example), which forces us to set the maximum number of iterations.

3.3. Further Discussion between the Two Dynamic Adjustment Strategies

Similarities: (1) Both dynamic adjustment strategies are designed to incrementally change the original two-dimensional encoding trees to adapt

the incremental edges and nodes in dynamic scenarios. (2) The time complexities for computing the updated structural entropy of both strategies are significantly lower than the original calculation formula (detailed analysis is shown in Section 4.3). (3) Both strategies cannot handle the birth of new communities and the dismission of the existing communities.

Differences: (1) The focuses of the two strategies are different. The naive adjustment strategy emphasizes theoretical analysis, such as boundedness and convergence analysis, and acts as a fast incremental baseline in experimental evaluations. By contrast, the node-shifting adjustment strategy mainly focuses on addressing the limitations of the naive strategy (Section 3.1.5) and the dynamic optimization of the existing communities towards a lower structural entropy. (2) The ways of updating encoding trees, or updating community partitionings, of these two strategies are different. In the naive adjustment strategy, new edges do not change the communities of the existing nodes, and new nodes are assigned to the direct neighbors' communities. While in the node-shifting adjustment strategy, the influence on community adjustment of new edges is considered and the new nodes' community is also determined by the incremental edges. (3) The time complexity of the naive adjustment strategy is fixed while that of the node-shifting strategy grows nearly linearly with the iteration number N. Experiments show that the naive strategy is faster than the node-shifting strategy with $N \geq 5$ in most cases (Fig. 13).

4. Incre-2dSE: an Incremental Measurement Framework of the Updated Two-Dimensional Structural Entropy

4.1. Definitions

In this part, we present the definitions of Structural Data, Structural Expressions, and Adjustment, which will be employed in subsequent sections.

Definition 9 (Structural Data)**.** *Given a graph G, the Structural Data of G is defined as follows:*

1. *(node level) the degree d_i of each $v_i \in \mathcal{V}$;*
2. *(community level) the volume V_α and the cut edge number g_α;*
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_α of each $\alpha \in A$.*

Definition 10 (Structural Expressions)**.** *The Structural Expressions of G are defined as follows:*

1. *(node level)*
$$\hat{S}_N = \sum_{d \in D} k_d d \log d, \tag{52}$$
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)*
$$\hat{S}_C = \sum_{\alpha \in A} (g_\alpha - V_\alpha) \log V_\alpha; \tag{53}$$
3. *(graph level)*
$$\hat{S}_G = -\sum_{\alpha \in A} g_\alpha. \tag{54}$$

Definition 11 (Adjustment)**.** *The Adjustment from the original graph G to the updated graph G' is defined as follows:*

1. *(node level) the degree change $\delta(v)$ for each node $v \in \phi_\lambda$ and the node number change $\delta_k(d) = k'_d - k_d$ of each $d \in \mathcal{D}$, where $\mathcal{D}$ 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 $\alpha \in \mathcal{A}$;*
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 $J_{n-c} = \{..., (v_i, \alpha'(v_i)), ...\}$ where $\alpha'(v_i)$ denotes the new community ID of v_i;*
5. *(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_{α_i} is updated as $T_{\alpha_i} \cup \{v_j\}$ or $T_{\alpha_i} / \{v_j\}$.*

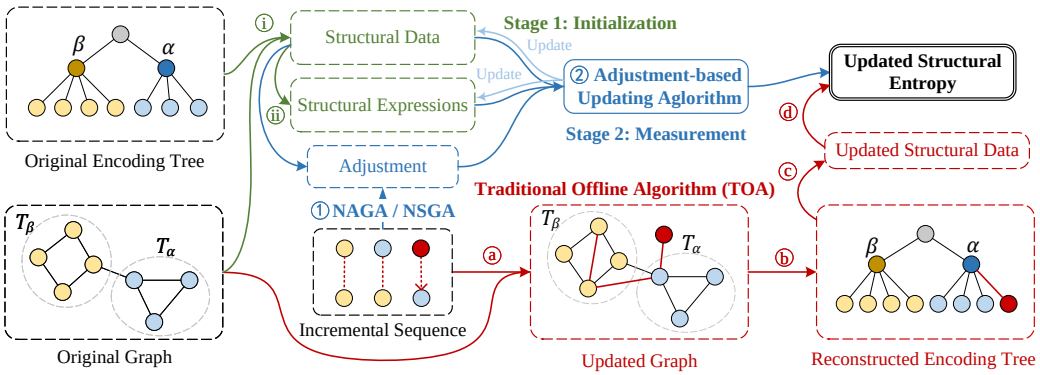

Figure 8: The outline of *Incre-2dSE* (including two stages, Initialization and Measurement) and the traditional offline algorithm.

4.2. Outline

The illustration of our incremental framework *Incre-2dSE* and its static baseline, the traditional offline algorithm (TOA), is shown in Fig. 8. *Incre-2dSE* aims to efficiently measure the updated two-dimensional structural entropy while dynamically adjusting the community partitioning given the original graph, the original encoding tree, and the incremental sequences. This framework consists of two stages, *initialization* and *measurement*. In the initialization stage, the Structural Data (Definition 9), which contains a graph's essential data to compute the structural entropy, is extracted from the original graph and its encoding tree (Fig. 8ⓘ). Then the Structural Expressions (Definition 10), which are defined as the expressions of the Structural Data, are computed and saved (Fig. 8ⓘⓘ). For the same original graph, Initialization only needs to be performed once. In the measurement stage, the Adjustment (Definition 11), which is defined as 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) or the node-shifting adjustment strategy (NSGA) (Fig. 8①). 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. 8②). As the baseline, TOA commences by updating the graph using the incremental sequence (Fig. 8ⓐ). Next, the new encoding

tree of the updated graph is reconstructed using a static community detection method (Fig. 8ⓑ). Then, the updated Structural Data is extracted (Fig. 8ⓒ), and finally, the updated structural entropy is computed by definition (Fig. 8ⓓ).

4.3. The Incremental Framework

4.3.1. Stage 1: Initialization

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 (Definition 9) can be easily obtained and saved in the time complexity of $O(|\mathcal{E}|)$ (Fig. 8ⓘ). Then the Structural Expressions (Definition 10) can be calculated with the saved Structural Data in $O(|\mathcal{V}|)$ (Fig. 8ⓘⓘ). Overall, the Initialization stage requires total time complexity $O(|\mathcal{E}|)$.

4.3.2. Stage 2: Measurement

In this stage, we first need to generate the Adjustment (Definition 11) from G to G'. We provide two algorithms for generating Adjustments by the proposed two dynamic adjustment strategies, namely *the naive adjustment generation algorithm (NAGA)* and *the node-shifting adjustment generation algorithm (NSGA)* (Fig. 8①). The input of both two algorithms is the Structural Data of the original graph and an incremental sequence and the output is an Adjustment. 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 $O(nN|A|)$.

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

$$H^{\mathcal{T}'}(G') = -\frac{1}{2m+2n}[\hat{S}'_N + \hat{S}'_C + \hat{S}'_G \log(2m+2n)], \tag{55}$$

Algorithm 1: Naive adjustment generation algorithm (NAGA)

Input : The Structural Data (d_i, V_α, g_α, m, $\alpha(v_i)$, and T_α) of G, and an incremental sequence ξ from G to G'.

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

$n :=$ `GetLength`(ξ);
$\delta(v_i) := 0$, $\delta_k(d) := 0$, $\delta_V(\alpha) := 0$, $\delta_g(\alpha) := 0$, $\mathcal{D} := \emptyset$, $\mathcal{A} := \emptyset$, $J_{n-c} := \emptyset$, $J_{c-n} := \emptyset$;
Let the proxy maps be $\hat{\alpha}(v_i) := \alpha(v_i), v_i \in \mathcal{V}$;
Let the proxy node level Structural Data be $\hat{d}_v := d_v, v \in \mathcal{V}$, where d_v denotes the degree of v;
for $e = (u, v, +) \in \xi$ **do**
    $\mathcal{D} := \mathcal{D} \cup \{d_u, d_v, d_u + 1, d_v + 1\}$;
    $\delta_k(\hat{d}_u) := \delta_k(\hat{d}_u) - 1$, $\delta_k(\hat{d}_u + 1) := \delta_k(\hat{d}_u + 1) + 1$;
    $\delta_k(\hat{d}_v) := \delta_k(\hat{d}_v) - 1$, $\delta_k(\hat{d}_v + 1) := \delta_k(\hat{d}_v + 1) + 1$;
    $\hat{d}_u := \hat{d}_u + 1$, $\hat{d}_v := \hat{d}_v + 1$, $\delta(u) := \delta(u) + 1$, $\delta(v) := \delta(v) + 1$;
    if $\hat{\alpha}(u) ==$ `NULL` **then**
        $\hat{\alpha}(u) := \hat{\alpha}(v)$;
        $J_{n-c} := J_{n-c} \cup \{(u, \hat{\alpha}(v))\}$, $J_{c-n} := J_{c-n} \cup \{(\hat{\alpha}(v), u, +)\}$;
    end
    if $\hat{\alpha}(v) ==$ `NULL` **then**
        $\hat{\alpha}(v) := \hat{\alpha}(u)$;
        $J_{n-c} := J_{n-c} \cup \{(v, \hat{\alpha}(u))\}$, $J_{c-n} := J_{c-n} \cup \{(\hat{\alpha}(u), v, +)\}$;
    end
    $\mathcal{A} := \mathcal{A} \cup \{\hat{\alpha}(u), \hat{\alpha}(v)\}$;
    if $\hat{\alpha}(v) == \hat{\alpha}(u)$ **then**
        $\delta_V(\hat{\alpha}(v)) := \delta_V(\hat{\alpha}(v)) + 2$;
    end
    if $\hat{\alpha}(v) \neq \hat{\alpha}(u)$ **then**
        $\delta_V(\hat{\alpha}(v)) := \delta_V(\hat{\alpha}(v)) + 1$, $\delta_V(\hat{\alpha}(u)) := \delta_V(\hat{\alpha}(u)) + 1$;
        $\delta_g(\hat{\alpha}(v)) := \delta_g(\hat{\alpha}(v)) + 1$, $\delta_g(\hat{\alpha}(u)) := \delta_g(\hat{\alpha}(u)) + 1$;
    end
end
return the Adjustment from G to G'.

Algorithm 2: Node-shifting adjustment generation algorithm (NSGA)

Input : The Structural Data (d_i, V_α, g_α, m, $\alpha(v_i)$, and T_α) of the original graph G, an incremental sequence ξ 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}, and J_{c-n}) from G to G'.

$n :=$ `GetLength`(ξ), $J_{n-c} := \emptyset$, $J_{c-n} := \emptyset$ $\delta(v_i) := 0$, $\delta_k(d) := 0$, $\delta_V(\alpha) := 0$, $\delta_g(\alpha) := 0$;

Let the proxy maps be $\hat{\alpha}(v_i) := \alpha(v_i), \hat{\hat{\alpha}}(v_i) := \alpha(v_i), v_i \in \mathcal{V}$;

Let the proxy node-level Structural Data be $\hat{d}_v := d_v, v \in \mathcal{V}$, where d_v denotes the degree of v;

Let the involved node set be $I := \emptyset$;

`// Initialize the Adjustment`

for $e = (u, v, op) \in \xi$ **do**

  $I := I \cup \{u, v\}$;

  if $op == +$ **then**

    if *u, v are both existing nodes in $\mathcal{V}$* **then**

      Update the node-level Adjustment (using the proxy node-level Structural Data, the same as below);

      if *$\alpha(u)$ and $\alpha(v)$ are both not None* **then**

        Update the community-level Adjustment without changing the community partitioning;

      else if *$\alpha(u)$ or $\alpha(v)$ is None (suppose $\alpha(u) == None$)* **then**

        $\delta_V(\alpha(u)) := \delta_V(\alpha(u)) + 1$; $\delta_g(\alpha(u)) := \delta_g(\alpha(u)) + 1$;

    else if *u or v does not exist (suppose u does not exist)* **then**

      $\alpha(u) :=$ None;

      Update the node-level Adjustment;

      $\delta_V(\alpha(u)) := \delta_V(\alpha(u)) + 1$; $\delta_g(\alpha(u)) := \delta_g(\alpha(u)) + 1$;

      $J_{n-c} := J_{n-c} \cup \{(u, \text{None})\}$; $J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\}$;

    else if *u and v are both not existing* **then**

      $\alpha(u) :=$ None, $\alpha(v) :=$ None;

      Update the node-level Adjustment;

      $J_{n-c} := J_{n-c} \cup \{(u, \text{None})\}$; $J_{n-c} := J_{n-c} \cup \{(v, \text{None})\}$;

      $J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\}$; $J_{c-n} := J_{c-n} \cup \{(\text{None}, u, +)\}$;

    Update the proxy node-level Strucutual Data as if the edge e is added into the graph;

  else if $op == -$ **then**

    Update the node-level and the community-level Adjustment without changing the community partitioning;

    Update the proxy node-level Strucutual Data as if the edge e is removed from the graph;

  end

end

`// See the rest on the next page`

```
   // The rest of Algorithm 2
   // Start node-shifting
32 τ := 1;
33 while τ ≤ N and I ≠ ∅ do
34 |  Ĩ := ∅, X := ∅;
35 |  for each node v ∈ I do
36 |  |  Determine the OPC of v denoted by α* using α̂(v);
37 |  |  X := X ∪ {(v, α*)};
38 |  |  if α̂(v) ≠ α* then
39 |  |  |  Update the Adjustment as if v moves into T_α* using α̂(v);
40 |  |  |  for each node z ∈ Neighbor(v) do
41 |  |  |  |  if α̂(z) ≠ α* then
42 |  |  |  |  |  Ĩ := Ĩ ∪ {z};
43 |  |  |  |  end
44 |  |  |  end
45 |  |  |  α̂̂(v) := α*;
46 |  |  end
47 |  end
48 |  for each (v, α) ∈ X do
49 |  |  α̂(v) := α;
50 |  end
51 |  I := Ĩ, τ := τ + 1;
52 end
53 return the Adjustment from G to G'.
```

where $\hat{S}'_N$, $\hat{S}'_C$, and $\hat{S}'_G$ denote the incrementally updated Structural Expressions:

$$\begin{aligned}
\hat{S}'_N &= \hat{S}_N + \sum_{d\in\mathcal{D}} \delta_k(d) d \log d; \\
\hat{S}'_C &= \hat{S}_C + \sum_{\alpha\in\mathcal{A}} [(g_\alpha + \delta_g(\alpha) + V_\alpha + \delta_V(\alpha)) \log(V_\alpha + \delta_V(\alpha)) - (g_\alpha + V_\alpha) \log V_\alpha]; \\
\hat{S}'_G &= \hat{S}_G + \sum_{\alpha\in\mathcal{A}} -\delta_g(\alpha).
\end{aligned} \tag{56}$$

To implement the above incremental calculation process, we provide *the Adjustment-based incremental updating algorithm (AIUA)* (Fig. 8②). 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 $O(|\phi_\lambda| + |\mathcal{A}| + |J_{n-c}| + |J_{c-n}|) \leq O(n)$. 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 $O(n)$.

4.4. Baseline: the Traditional Offline Algorithm

The traditional offline algorithm (TOA) reconstructs the encoding tree for each 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 (ⓐ in Fig. 8). Secondly, it partitions the graph node set into communities using several different static community detection algorithms, e.g., Infomap [16], Louvain [17], and Leiden [18], to construct the two-dimensional encoding tree (ⓑ in Fig. 8). Thirdly, the node-level, community-level, and graph-level Structural Data of the updated graph is counted and saved (ⓒ in Fig. 8). Finally, the updated structural entropy is computed via Eq. (14) (ⓓ in Fig. 8). 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.

5. Extensions on More Complex Graphs

In this section, we discuss the feasibility of the extension to weighted or directed graphs of our methods. First, we argue that the method for undirected weighted graphs can be extended naturally from that of undirected unweighted graphs. Second, we analyze the fundamental differences between the paradigm of structural entropy incremental computation for directed graphs and that for undirected graphs and present new methods for calculating one-dimensional structural entropy incrementally on directed weighted graphs.

5.1. Undirected Weighted Graphs

The incremental measurement method of structural entropy for undirected weighted graphs can be intuitively and easily extended from the methods for undirected unweighted graphs proposed earlier. In the following, we

Algorithm 3: Adjustment-based incremental updating algorithm (AIUA)

Input : The Structural Data (d_i, V_α, g_α, m, $\alpha(v_i)$, and T_α) and the Structural Expressions ($\hat{S}_N$, $\hat{S}_C$, 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}, and J_{c-n}) from G to G'.

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

`// Update the Structural Data`

for *each* $v_i \in \phi_\lambda$ **do**
  $d'_i := d_i + \delta(v_i)$;
end
for *each* $\alpha \in \mathcal{A}$ **do**
  $V'_\alpha := V_\alpha + \delta_V(\alpha)$; $g'_\alpha := g_\alpha + \delta_g(\alpha)$;
end
$m' = m + n$;
for *each* $(v, \alpha) \in J_{n-c}$ **do**
  $\alpha'(v) := \alpha$;
end
$T'_\alpha := T_\alpha$ for all $\alpha \in A$;
for *each* $(\alpha, v, op) \in J_{c-n}$ **do**
  if $op == +$ **then**
    $T'_\alpha := T'_\alpha \cup \{v\}$;
  end
  if $op == -$ **then**
    $T'_\alpha := T'_\alpha / \{v\}$;
  end
end

`// Update the Structural Expressions`

$\hat{S}'_N := \hat{S}_N$; $\hat{S}'_C := \hat{S}_C$; $\hat{S}'_G := \hat{S}_G$;
for *each* $d \in \mathcal{D}$ **do**
  $S'_N := S'_N + \delta_k(d) d \log d$;
end
for *each* $\alpha \in \mathcal{A}$ **do**
  $\hat{S}'_C := \hat{S}'_C + (g_\alpha + \delta_g(\alpha) + V_\alpha + \delta_V(\alpha)) \log(V_\alpha + \delta_V(\alpha)) - (g_\alpha + V_\alpha) \log V_\alpha$;
  $\hat{S}'_G := \hat{S}'_G - \delta_g(\alpha)$;
end

`// Calculate the updated two-dimensional structural entropy`

$H^{\mathcal{T}'}(G') := -\frac{1}{2m+2n}[\hat{S}'_N + \hat{S}'_C + \hat{S}'_G \log(2m + 2n)]$;
return $H^{\mathcal{T}'}(G')$, the updated Structural Data, and the updated Structural Expressions.

Algorithm 4: Traditional Offline Algorithm (TOA)

Input : The original graph G and an incremental sequence ξ.
Output: The updated two-dimensional structural entropy $H^{\mathcal{T}'}(G')$.

Get the updated graph G' by combining G and ξ;
Construct the two-dimensional encoding tree $\mathcal{T}'$.;
Get the degree d'_i of each node $v_i \in \mathcal{V}'$;
Get the volume V'_α and the cut edge number g'_α of $\alpha \in A'$;
Get the total edge number m' of G';
Obtain $H^{\mathcal{T}'}(G')$ via Eq. (14);
return $H^{\mathcal{T}'}(G')$;

first introduce the definition of two-dimensional structural entropy of undirected weighted graphs. Then we update the definition of the Adjustment and propose an incremental formula for structural entropy computation under the new circumstance.

Two-dimensional structural entropy of undirected weighted graphs. An undirected weighted graph can be denoted as $G_W = (\mathcal{V}, \mathcal{E}, w)$, where $w(e) \in \mathbb{R}^+$ represents the weight of $e \in \mathcal{E}$. For any $e \notin \mathcal{E}$, let $w(e) = 0$. According to Li and Pan [6], the definition of the encoding tree for an undirected weighted graph remains unchanged, and the two-dimensional structural entropy of $G_W = (\mathcal{V}, \mathcal{E}, w)$ by its two-dimensional encoding tree $\mathcal{T}$ is defined as

$$H^{\mathcal{T}}(G_W) = \sum_{\alpha_i \in A} (-\frac{g_{\alpha_i}}{V_\lambda} \log \frac{V_{\alpha_i}}{V_\lambda} + \sum_{v_j \in T_{\alpha_i}} -\frac{d_j}{V_\lambda} \log \frac{d_j}{V_{\alpha_i}}), \tag{57}$$

where new degree $d_j = \sum_{v_i \in \mathcal{N}(v_j)} w(v_j, v_i)$ ($\mathcal{N}(v)$ denotes the set of neighbors of node v), new volume $V_\alpha = \sum_{v_i \in T_\alpha} d_i$, and new cut edge number g_α is replaced with the sum of the weights of the edges with exactly one endpoint in T_α.

Adjustment definition of undirected weighted graphs. Due to the continuity of edge weights, the original node level and graph level Adjustment definitions (Definition 11) no longer apply. We renew the Adjustment definitions of the two levels as follows:

1. (node level) the total weight change $\delta(v_i) = d'_i - d_i$ for each node $v_i \in \phi_\lambda$;
2. (graph level) the total weight change of the whole graph denoted by $\delta_V(\lambda)$.

Incremental formula for structural entropy computation. According to Eq. (57), the incremental computation formula for undirected weighted graphs can then be formulated as (extended from Eq. (55))

$$H^{\mathcal{T}'}(G'_W) = -\frac{1}{V_\lambda + \delta_V(\lambda)}[\hat{S}'_N + \hat{S}'_C + \hat{S}'_G \log(V_\lambda + \delta_V(\lambda))], \tag{58}$$

where

$$\begin{aligned}
\hat{S}'_N &= \hat{S}_N + \sum_{v_k \in \phi_\lambda} [(d_k + \delta(v_k)) \log(d_k + \delta(v_k)) - d_k \log d_k]; \\
\hat{S}'_C &= \hat{S}_C + \sum_{\alpha \in \mathcal{A}} [(g_\alpha + \delta_g(\alpha) + V_\alpha + \delta_V(\alpha)) \log(V_\alpha + \delta_V(\alpha)) - (g_\alpha + V_\alpha) \log V_\alpha]; \\
\hat{S}'_G &= \hat{S}_G + \sum_{\alpha \in \mathcal{A}} -\delta_g(\alpha).
\end{aligned} \tag{59}$$

5.2. Directed Weighted Graphs

The main method proposed in this paper is difficult to transfer to directed graph scenarios since the measurement of the structural entropy of directed graphs is fundamentally different from that of undirected graphs. The key difference is that the directed graph needs to be transferred into a transition matrix and stationary distribution needs computed. In this part, we briefly present an incremental scheme for measuring the one-dimensional structural entropy of directed weighted graphs since the incremental computation of two-dimensional structural entropy is quite complex. Specifically, we first define the directed weighted graph and its non-negative matrix representation. After that, we introduce the formula of the structural entropy of directed weighted graphs [6]. Finally, we review the traditional methods, namely Eigenvector Calculation and Global Aggregation, for accurately or approximately calculating the one-dimensional structural entropy of directed weighted graphs and then propose an incremental iterative approximation algorithm, i.e., Local Propagation.

5.2.1. Directed Weighted Graph and Non-Negative Matrix

Definition 12 (Directed Weighted Graph)**.** *A directed weighted graph can be denoted as* $G_{DW} = (\mathcal{V}, \mathcal{E}, f)$*, which satisfies the following properties:*

(1) $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ *is the node set;*

(2) *$\mathcal{E}$ denotes the set of directed edges $e = (v_i, v_j)$;*
(3) *for each directed edge $e = (v_i, v_j) \in \mathcal{E}$, $f(e) = f(v_i, v_j) > 0$ is the edge weight from v_i to v_j. For $e = (v_i, v_j) \notin \mathcal{E}$, we denote $f(e) = f(v_i, v_j) = 0$.*

The definition of directed weighted graphs is shown in Definition 12. Given a directed weighted graph $G_{DW} = (\mathcal{V}, \mathcal{E}, f)$ with N nodes, we fix the nodes in $\mathcal{V}$ in an order like $v_1, v_2, ..., v_N$. For $i, j \in \{1, 2, .., N\}$, we define $a_{ij} = f(v_i, v_j)$. Then we can obtain a non-negative matrix $\mathbf{A} = (a_{ij})$, named as a matrix of G_{DW}. In other words, a directed weighted graph can be represented by a non-negative matrix. The definition of the non-negative matrix representation of directed weighted graphs is presented in Definition 13.

Definition 13 (Directed Weighted Graph Represented by Non-Negative Matrix)**.** *A directed weighted graph $G_{DW} = (\mathcal{V}, \mathcal{E}, f)$ can be denoted as an $N \times N$ non-negative matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}_{\geq 0}^{N \times N}$, where for each $i, j \in \{1, 2, ..., N\}$, $a_{ij} = f(v_i, v_j) \geq 0$.*

5.2.2. One-Dimensional Structural Entropy of Directed Weighted Graphs

For direct graphs, the structural entropy is an uncertainty measurement defined on strongly connected graphs whose matrices must be irreducible (Definition 14).

Definition 14 (Irreducible Matrix)**.** *Given a non-negative matrix $\mathbf{A} \in \mathbb{R}_{\geq 0}^{N \times N}$, if there exists a permutation matrix $\mathbf{P}$ such that*

$$\mathbf{PAP} = \begin{bmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix}, \tag{60}$$

where $\mathbf{X}$ and $\mathbf{Z}$ are square matrices, then $\mathbf{A}$ is said to be a reducible matrix, otherwise $\mathbf{A}$ is an irreducible matrix.

Normalize each row in an irreducible matrix $\mathbf{A}$, i.e., for each i and j, let

$$b_{ij} = \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}}. \tag{61}$$

Then we can get a normalized irreducible matrix $\mathbf{B} = (b_{ij})$ where the sum of elements in each row is 1. b_{ij} is called the normalized edge weight. There is a fundamental theorem for normalized irreducible matrices like $\mathbf{B}$.

Theorem 5 (Perron Forbenius). *If* $\mathbf{B} \in \mathbb{R}_{\geq 0}^{N \times N}$ *is an irreducible matrix where for each* i, $\sum_{k=1}^{N} b_{ik} = 1$, *then*

(1) *the maximum eigenvalue of* $\mathbf{B}$ *is* 1*;*
(2) *the maximum eigenvalue of* $\mathbf{B}$ *has a unique left eigenvector;*
(3) *The unique left eigenvector of the maximum eigenvalue of* $\mathbf{B}$ *is a probability distribution, denoted by* $\pi = [\pi_1, \pi_2, ..., \pi_N]$.

According to Theorem 5, the stationary distribution of $\mathbf{A}$ can then be defined in Definition 9.

Definition 15 (Stationary Distribution). *Let* $\mathbf{A} \in \mathbb{R}_{\geq 0}^{N \times N}$ *be an irreducible matrix and* $\mathbf{B}$ *represents the normalized matrix of* $\mathbf{A}$ *where for each* i, $\sum_{k=1}^{N} b_{ik} = 1$. *The stationary distribution of* $\mathbf{A}$ *is defined as the unique left eigenvector of the maximum eigenvalue of* $\mathbf{B}$, *denoted by* $\pi = [\pi_1, \pi_2, ..., \pi_N]$.

Finally, the one-dimensional structural entropy (Definition 16) of an irreducible non-negative matrix (or a directed weighted graph) is defined as the Shannon entropy of the stationary distribution, which measures the total amount of uncertainty embedded in a directed weighted graph.

Definition 16 (One-Dimensional Structural Entropy). *Let* $\mathbf{A} \in \mathbb{R}_{\geq 0}^{N \times N}$ *be an irreducible matrix and* $\pi = [\pi_1, \pi_2, ..., \pi_N]$ *is the stationary distribution of* $\mathbf{A}$. *The one-dimensional structural entropy is defined as*

$$\mathcal{H}^1(\mathbf{A}) = -\sum_{i=1}^{N} \pi_i \log_2 \pi_i. \tag{62}$$

5.2.3. Incremental Measurement of One-Dimensional Structural Entropy

Generally, the exact value of one-dimensional structural entropy can be obtained by Eq. (62) in $O(n)$ given the stationary distribution. However, calculating the exact stationary distribution needs to solve the eigenvectors of the normalized irreducible matrix, which usually costs $O(n^3)$. In dynamic scenarios, the directed weighted graph evolves as time passes by. When an irreducible matrix $\mathbf{A} \in \mathbb{R}_{\geq 0}^{N \times N}$ representing a directed weighted graph gets an incremental $\Delta\mathbf{A}$, it becomes an updated irreducible matrix $\mathbf{A}' = \mathbf{A} + \Delta\mathbf{A}$. Let $n = ||\Delta\mathbf{A}||_0$ denote the incremental size. Let $\mathbf{B}$ and $\mathbf{B}'$ be the normalized

matrices of $\mathbf{A}$ and $\mathbf{A}'$. Define $\Delta\mathbf{B} = \mathbf{B}' - \mathbf{B}$. We can calculate $\Delta\mathbf{B} = (\Delta b_{ij})$ from $\mathbf{A}$ and $\Delta\mathbf{A}$ by

$$\Delta b_{ij} = b'_{ij} - b_{ij} = \frac{a'_{ij}}{\sum_{k=1}^{N} a'_{ik}} - \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}} = \frac{a_{ij} + \Delta a_{ij}}{(\sum_{k=1}^{N} a_{ik} + \Delta a_{ik})} - \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}}. \tag{63}$$

Since a non-zero element of $\Delta\mathbf{A}$ may influence at most all N elements of a row in $\mathbf{B}$, $||\Delta\mathbf{B}||_0$, named as the normalized incremental size, will be no more than nN. In addition, the number of the influenced rows in $\mathbf{B}$, denoted by n_B, will be no more than n. In the following, three ways are listed to compute the updated one-dimensional structural entropy.

Exact value calculation by Eigenvector Calculation. The first way is to calculate the exact value of the updated one-dimensional structural entropy by Definition 15 and 16. Specifically, the new exact stationary distribution π' is first computed by solving the left eigenvector of the maximum eigenvalue of $\mathbf{B}'$. Then the updated one-dimensional structural entropy can be calculated by

$$\mathcal{H}^1(\mathbf{A}') = -\sum_{i=1}^{N} \pi'_i \log_2 \pi'_i. \tag{64}$$

Generally, the total time complexity of this way is $O(N^3)$.

Approximate value calculation by Global Aggregation. According to the theory of the Markov chain, the approximate stationary distribution can be approximated by iteratively right-multiplying $\mathbf{B}'$, i.e.,

$$\pi^{(\theta)} = \pi\mathbf{B}'^{\theta} = \pi(\mathbf{B} + \Delta\mathbf{B})^{\theta}, \tag{65}$$

where $\theta \in \mathbb{N}$ is the iteration number. Whenever $\pi^{(\theta)}$ updates to $\pi^{(\theta+1)}$ by right-multiplying $\mathbf{B} + \Delta\mathbf{B}$, the updating process is equivalent to an information aggregation operation on graphs along directed edges, i.e., for each node v_i,

$$\pi_i^{(\theta+1)} = \sum_{v_j \in \mathcal{N}(v_i)} \pi_j^{(\theta)} b_{ji}, \tag{66}$$

where $\mathcal{N}(v_i)$ denotes the neighbors of v_i. Since for each iteration, all nodes need to be updated, we name this calculation method as "Global Aggregation". Finally, we compute the Shannon entropy of $\pi^{(\theta)}$ as the approximate one-dimensional structural entropy:

$$\mathcal{H}^1(\mathbf{A}') \approx -\sum_{i=1}^{N} \pi_i^{(\theta)} \log_2 \pi_i^{(\theta)}. \tag{67}$$

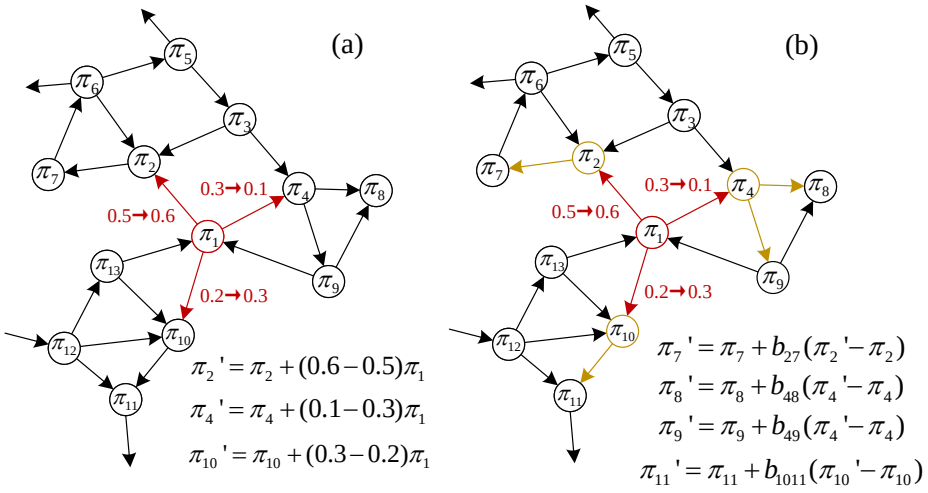

Figure 9: An illustration of Local Propagation. Note that all edge weights are normalized. In the first step (a), the red arrows denote the edges whose weight changes after getting an incremental, i.e., the involved edges. The stationary distributions of the involved nodes (π_2, π_4, π_{10}) are updated. In the second step (b), the direct successors of the last involved nodes update their stationary distributions and become new involved nodes. Then the procedure of (b) is repeated until the maximum iteration number is reached.

The total computational complexity of Global Aggregation is about $O(\theta N^2)$ in the form of matrix multiplication using Eq. (65). Utilizing graph perspective (Eq. (66)), the time complexity can be reduced to $O(\theta N\overline{d}) = O(\theta|\mathcal{E}|)$, where $\overline{d}$ denotes the mean direct successor number of all graph nodes.

Fast approximate value calculation by Local Propagation. In Global Aggregation, all nodes and edges need to be traversed in each iteration, which leads to high computational redundancy. In this part, we propose a new method for fast approximation of the updated one-dimensional structural entropy, namely Local Propagation. As the name suggests, the key idea is to use an information propagation scheme involving only changed local nodes to further reduce the redundancy of the aggregation process in Eq. (66).

In particular, Local Propagation contains two steps. In the first step (Fig. 9(a)), we define the set of directed edges in $\mathbf{B}$ influenced by $\Delta\mathbf{B}$ as "involved edges" (red edges in Fig. 9(a)). We then let $\pi^{(1)} = \pi^{(0)}$. For each involved edge (v_i, v_j), the stationary distribution value of the pointed node v_j is updated by

$$\pi_j^{(1)} \leftarrow \pi_j^{(1)} + \Delta b_{ij}\pi_i, \tag{68}$$

and v_j is added into an "involved nodes" set denoted by $I^{(1)}$. The time complexity of the first step is linearly correlated to the size of the involved edge set, i.e., $O(nN)$. In the second step, we repeat the following procedure for $\theta - 1$ times. Let $\pi^{(\theta+1)} = \pi^{(\theta)}$. For each node v_i in $I^{(\theta)}$, we update the stationary distribution values of all v_i's direct successors (like Fig. 9(b)) by

$$\pi_j^{(\theta+1)} \leftarrow \pi_j^{(\theta+1)} + b'_{ij}(\pi_i^{(\theta)} - \pi_i^{(\theta-1)}), v_j \in \mathcal{N}_s(v_i), \tag{69}$$

where $\mathcal{N}_s(v_i)$ denotes the direct successors of v_i. After all nodes in $I^{(\theta)}$ are traversed, $I^{(\theta)}$ is updated as $I^{(\theta+1)}$, i.e., all the direct successors of nodes of the original set $I^{(\theta)}$. For each iteration, the time complexity is $O(|I^{(\theta)}|\overline{d}^{(\theta)})$, where $\overline{d}^{(\theta)}$ denotes the mean direct successor number of nodes in $I^{(\theta)}$. In other words, let $E^{(\theta)}$ ($\theta \geq 1$) denote the set of edges whose starting points belong to $I^{(\theta)}$, we have $|I^{(\theta)}|\overline{d}^{(\theta)} = |E^{(\theta)}|$. Therefore, the time complexity of each iteration is $O(|E^{(\theta)}|)$ which must be less or equal to the complexity of Eq. (66), $O(|\mathcal{E}|)$.

6. Experiments and Evaluations

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

6.1. Artificial Datasets

First, we generate 3 different initial states of the dynamic graphs by utilizing the *random_partition_graph* (Random) [19], *gaussian_random_partition_graph* (Gaussian) [20], and *stochastic_block_model* (SBM) [21] methods in "Networkx" [22] (a Python library). Parameter descriptions of the three methods are listed below:

- **Random parameters:** This method has 3 parameters. The first parameter is a list of community sizes $\mathcal{S} = [s_1, s_2, ...]$, which denotes the node number of each community of the initial state. The other two parameters are two probabilities p_{in} and p_{ac}. Nodes in the same community are connected with p_{in} and nodes across different communities are connected with p_{ac}.

- **Gaussian parameters:** This method creates k communities each with a size drawn from a normal distribution with mean s and variance s/v. Nodes are connected within communities with probability p_{in} and between communities with probability p_{ac}.
- **SBM parameters:** This method partitions the nodes in communities of given sizes $\mathcal{S} = [s_1, s_2, ...]$, and places edges between pairs of nodes independently, with a probability that depends on the communities.

After that, we generate incremental sequences and updated graphs for each initial state by Hawkes Process [23] referring to some settings of Zuo et al. [24]. Hawkes Process [23] models discrete sequence events by assuming that historical events can influence the occurrence of current events. In this process, we first randomly choose a node x to be the target node (Fig. 10($t = 1$)). Second, we add edges or nodes with given probabilities. Specifically, with probability p_{hn} (Fig. 10(Node Addition)), we connect a new node with x. With probability $1 - p_{\text{hn}}$ (Fig. 10(Edge Addition)), we (1) use Node2vec [25] to get embedding vectors of all nodes; (2) calculate the conditional intensity function $\Lambda_{y_i|x}$ between x and each of its non-neighboring nodes y:

$$\Lambda_{y|x}(t) = -||\mathbf{e}_x - \mathbf{e}_y||^2 + \sum_{t_h<t} -||\mathbf{e}_h - \mathbf{e}_y||^2 \exp\left(-\delta_x \left(t - t_h\right)\right), \tag{70}$$

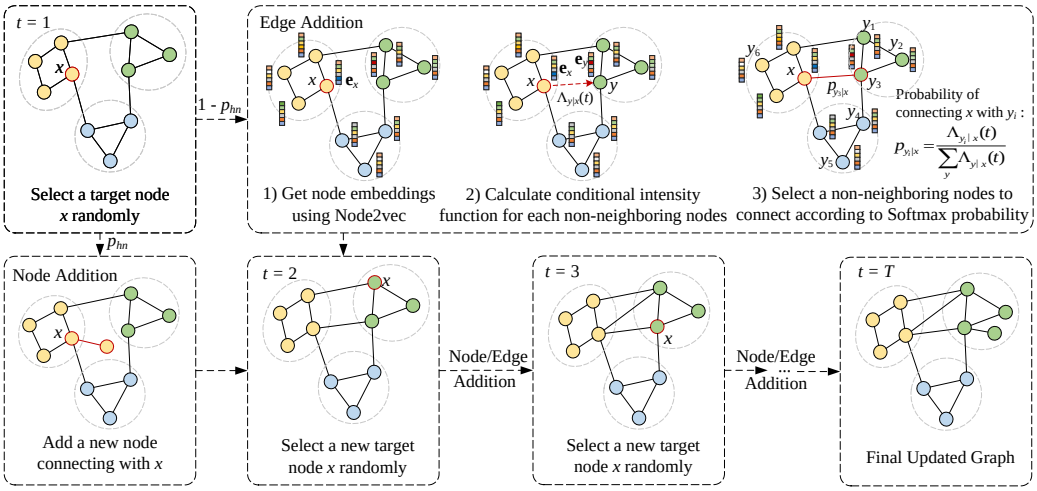

Figure 10: The generation process of the artificial Hawkes datasets.

Table 2: Parameter Values of the Generated Initial States and Hawkes Process.

	Parameter values
Random initial state	$\mathcal{S} = [800, 1000, 1200, 1400, 1600]$, $p_{\text{in}} = 0.05$, and $p_{\text{ac}} = 0.001$.
Gaussian initial state	The total node number is 6000, $s = 1000$, $v = 100$, $p_{\text{in}} = 0.05$, and $p_{\text{ac}} = 0.001$.
SBM initial state	$\mathcal{S} = [800, 1000, 1200, 1400, 1600]$, $p_{\text{in}} \sim^{*}\mathcal{U}(0.002, 0.01)$, and $p_{\text{ac}} \sim \mathcal{U}(0.01, 0.05)$.
Hawkes process	The dimension of the embedding vectors is set as 16 and p_{hn} is 0.05.

* $\mathcal{U}$ denotes uniform distribution.

Table 3: Statistics Description of the Artificial and Real-World Datasets.

Datasets	$\lvert\mathcal{V}_0\rvert$	$\lvert\mathcal{E}_0\rvert$	$\mathbb{E}(\lvert\Delta\mathcal{V}\rvert)$	$\mathbb{E}(\lvert\Delta\mathcal{E}\rvert)$	# of snapshots
Random-Hawkes	6,000	203,659	1,017	20,365	20
Gaussian-Hawkes	6,000	164,482	596	11,942	20
SBM-Hawkes	6,000	176,526	592	11,942	20
Cit-HepPh	25,656	132,119	235	10,727	20
DBLP	7,184	13,451	3,379	7,907	20
Facebook	14,094	72,809	2,233	26,000	20

where $\mathbf{e}_x, \mathbf{e}_y$ refers to the embedding vectors of nodes x and y, h refers to the historical neighbor nodes connected to x at time t_h before t, and δ_x refers to the discount rate, defined in this paper as the number of neighbors of x; and (3) add an edge between x and y_i with the Softmax conditional probability:

$$p_{y_i|x} = \frac{\Lambda_{y_i|x}}{\sum_y \Lambda_{y|x}}. \tag{71}$$

Subsequently, we repeat the above two steps to generate incremental sequences and updated graphs (Fig. $10(t = 2, 3, ..., T)$). The chosen parameter settings of the initial states and Hawkes Process are described in Table 2.

6.2. Real-World Datasets

For the real-world datasets, we choose Cit-HepPh [26], DBLP [27], and Facebook [28] to conduct our experiments. Cit-HepPh is a citation network in the field of high-energy physics phenomenology from 1993 to 2003. DBLP contains a co-authorship network of computer science papers from 1954 to 2015 in which authors are represented as vertices and co-authors are linked by an edge. 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. Overall, the statistics of the artificial and real-world datasets are briefly shown in Table 3.

6.3. Results and Analysis

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

In this application, we aim to optimize the community partitioning and monitor the corresponding two-dimensional structural entropy by our incremental algorithms, i.e., *NAGA+AIUA* and *NSGA+AIUA*, and the baseline *TOA* to quantify the community quality in real-time for each snapshot of a dynamic graph. Specifically, for each dataset, we first choose a static community detection method (referred to as static methods) from Infomap [16], Louvain [17], and Leiden [18] to generate the initial state's community partitioning. In this paper, Louvain is complemented by the *louvain_communities* method in "Networkx" [22]. Infomap and Leiden are complemented by the *community_infomap* and the *community_leiden* methods from the Python library "igraph" [29], respectively. Louvain and Infomap algorithms take default parameters while in Leiden we use "Modularity" as the objective instead of the original setting "Constant Potts Model (CPM)". This is because Leiden with CPM cannot effectively partition the communities on our datasets, as all partitions generated by Leiden with CPM contain only one node. Then we use *NAGA+AIUA*, *NSGA+AIUA*, and *TOA* to respectively measure the updated two-dimensional structural entropy at each time stamp. The default maximum iteration number of *NSGA* is set as 5 and the reason is discussed in Section 6.3.3.

The experimental results are shown in Fig. 11 (on real-world datasets) and Fig. 12 (on artificial datasets). Overall, the structural entropy obtained by *NSGA+AIUA* based on the node-shifting strategy, is completely smaller than that obtained by *NAGA+AIUA* based on the naive adjustment strategy, in all settings. For example, compared with *NAGA+AIUA*, *NSGA+AIUA* reduces the updated structural entropy by up to about 12% and 10% on Facebook and DBLP respectively. This verifies that the node-shifting strategy is theoretically and practically able to reduce the structural entropy and represents that the strategy can obtain a significantly better encoding tree (community partitioning) than the naive adjustment strategy.

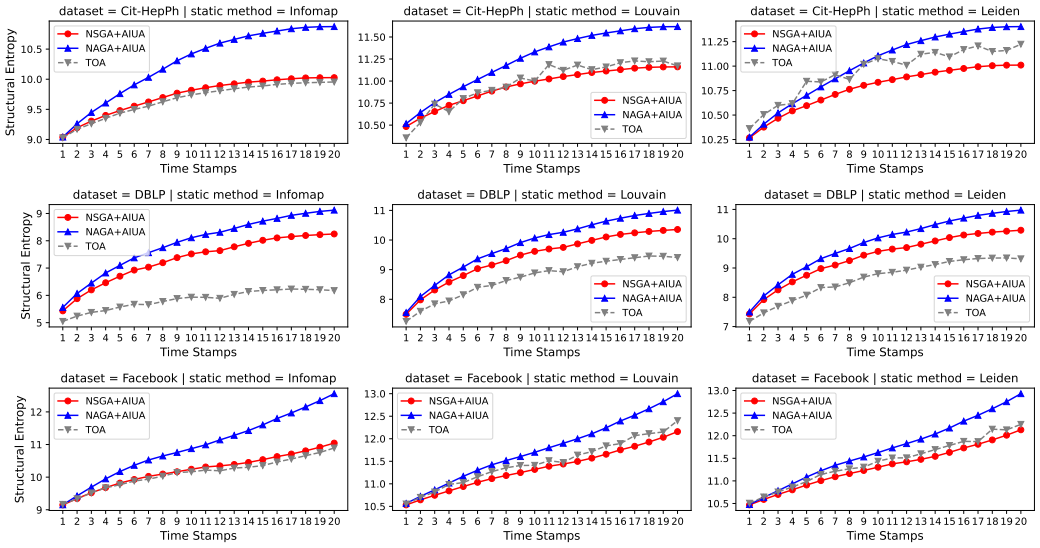

Figure 11: The updated structural entropy measured by *NAGA+AIUA*, *NSGA+AIUA*, and *TOA* on real-world datasets with different static methods. Lower structural entropy represents better performance.

While maintaining high efficiency (evaluated in Section 6.3.3), our incremental algorithms still exhibit a performance that is close to or better than *TOA*. In Fig. 11, the performance of *NSGA+AIUA* is only slightly weaker than *TOA* with Infomap on Cit-HepPh and Facebook. Further, it even achieves lower structural entropy than the offline algorithm when Louvain and Leiden are chosen. Two main reasons are listed as follows. (1) In the incremental algorithms, the new community partitioning is obtained by locally modifying the original partitioning of the initial state. While in *TOA* the community partitioning is reconstructed globally from snapshots (updated graphs) by the static methods. Although *TOA* spends a large amount of time searching for optimal partitioning globally, there is no theoretical proof that it must be better than local and greedy optimization used in incremental algorithms. (2) Additionally, the optimization objectives of incremental algorithms and *TOA* are different. The former focuses on the structural entropy itself while the latter aims to optimize modularity (Louvain and Leiden) or minimize the expected length of information transmission (Infomap).

From the experimental results on artificial datasets (Fig. 12), we can conclude that our incremental algorithms have higher stability in the dynamic calculation of structural entropy. In most cases, there are many mutations

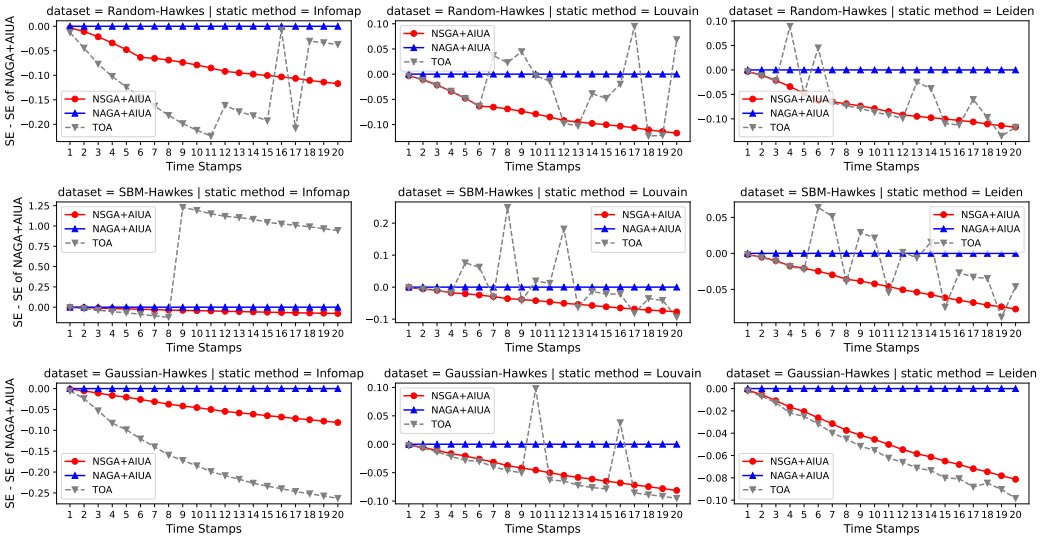

Figure 12: The updated structural entropy measured by *NAGA+AIUA*, *NSGA+AIUA*, and *TOA* on artificial datasets with different static methods. Since the three curves for the artificial datasets are closer to each other than that for the real-world datasets, all displayed structural entropy values are subtracted from the structural entropy value of *NAGA+AIUA* to better show the differences between the curves.

in the structural entropy value using *TOA*, while our algorithms maintain a continuous and stable decrease. It is because *TOA* globally reconstructs the community for each iteration so that small node or edge changes may cause more influence. By contrast, our algorithms dynamically adjust the community partitioning from the last snapshot, which affects only locally involved nodes.

6.3.2. 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 updated graphs, respectively, on each situation in Section 6.3.1. As we can see from Table 4, 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 4: The Updated Structural Entropy by Node-Shifting Adjustment Strategy with Different Number of Iterations. **Bold** Number Denotes the Lowest Structural Entropy.

# of iterations (N)		$N=3$	$N=5$	$N=7$	$N=9$
Cit-HepPh	Infomap	9.7155	9.7126	9.7122	**9.7120**
	Louvain	10.7804	10.7791	10.7786	**10.7784**
	Leiden	10.7802	10.7792	10.7788	**10.7786**
DBLP	Infomap	7.3274	7.3255	7.3244	**7.3242**
	Louvain	9.4851	9.4842	9.4841	**9.4838**
	Leiden	9.3919	9.3917	9.3912	**9.3909**
Facebook	Infomap	10.2134	10.2075	10.2060	**10.2055**
	Louvain	11.2946	11.2898	11.2876	**11.2864**
	Leiden	11.3087	11.3052	11.3038	**11.3030**
Random-Hawkes	Infomap	11.7836*	11.7832	11.7831	**11.7831**
	Louvain	11.7836*	11.7832	11.7831	**11.7831**
	Leiden	11.7836*	11.7832	11.7831	**11.7831**
Gaussian-Hawkes	Infomap	11.3352	11.3348	11.3347	**11.3346**
	Louvain	11.3352	11.3348	11.3347	**11.3346**
	Leiden	11.3352	11.3348	11.3347	**11.3346**
SBM-Hawkes	Infomap	11.6596	11.6595	11.6595	**11.6595**
	Louvain	11.6596	11.6595	11.6595	**11.6595**
	Leiden	11.6596	11.6595	11.6595	**11.6595**

* The structural differences between different static methods on artificial Hawkes datasets are really small indicating that the initial community partitionings of the three methods are almost the same.

6.3.3. Time Consumption Evaluation

Fig. 13 shows the time consumption comparison between our incremental algorithms, i.e. *NAGA+AIUA* and *NSGA+AIUA* ($N = 3, 5, 7, 9$), on all 6 datasets. The vertical axis in the figure represents the mean time consumption of the chosen incremental algorithm across all 20 snapshots. The horizontal axis represents 3 selected static methods. As we can see, the time cost of *NSGA+AIUA* increases as the increase of iteration number N. In addition, the time consumption of *NAGA+AIUA* is less than *NSGA+AIUA* with $N = 5$ in most cases.

Table 5 shows the time comparison between our online algorithm *NSGA+AIUA* ($N = 5$) and the offline algorithm *TOA*. As we can see from the results, all our proposed incremental algorithms are significantly faster than the existing static methods. Specifically, for example, *NSGA+AIUA* ($N = 5$) obtain over 140.93x and 77.81x speed up on average on DBLP and SBM-Hawkes, respectively, in contrast with the static method using Infomap.

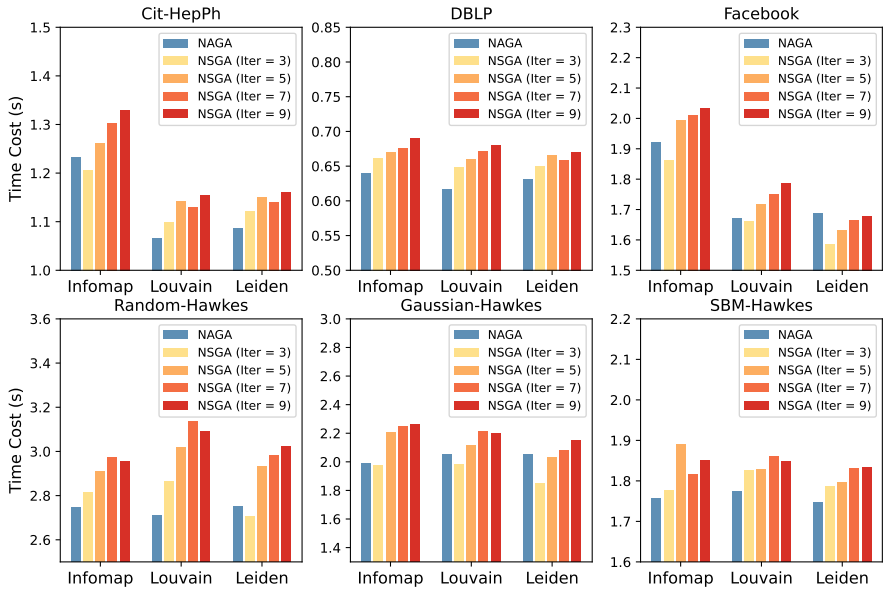

Figure 13: Mean time consumption of *NAGA+AIUA* and *NSGA+AIUA* ($N = 3, 5, 7, 9$) over 20 time stamps on each dataset under different static methods.

Table 5: Time Consumption Comparison of Our Incremental Algorithms (Online Time) and the Baseline Traditional Offline Algorithm (Offline Time).

Dataset	Cit-HepPh			DBLP			Facebook		
Static Method	Infomap	Louvain	Leiden	Infomap	Louvain	Leiden	Infomap	Louvain	Leiden
Online Time[1]	1.23s[2]	1.07s	1.09s	0.64s	0.62s	0.63s	1.92s	1.67s	1.69s
Offline Time	49.96s	6.42s	5.05s	49.80s	4.72s	12.30s	80.76s	8.15s	7.43s
Speedup[3]	↑ 40.62x	↑ 6.00x	↑ 4.63x	↑ 77.81x	↑ 7.61x	↑ 19.52x	↑ 42.06x	↑ 4.88x	↑ 4.40x
Dataset	Random-Hawkes			Gaussian-Hawkes			SBM-Hawkes		
Static Method	Infomap	Louvain	Leiden	Infomap	Louvain	Leiden	Infomap	Louvain	Leiden
Online Time	2.21s	2.48s	1.32s	1.48s	1.52s	1.55s	1.76s	1.55s	1.05s
Offline Time	58.37s	6.06s	3.02s	18.72s	4.08s	3.69s	248.04s	4.01s	3.02s
Speedup	↑ 26.41x	↑ 2.44x	↑ 2.28x	↑ 12.65x	↑ 2.68x	↑ 2.38x	↑ 140.93x	↑ 2.59x	↑ 2.88x

[1] Time cost of *NSGA+AIUA* ($N = 5$).
[2] Mean time consumption over 20 time stamps.
[3] Speedup = Offline Time/Online Time.

The reason why we choose $N = 5$ as the default parameter is as follows. As shown in Fig. 13, the time consumption of the node-shifting strategy rises linearly with N. However, Table 4 shows that the rate of decline of structural entropy gradually decreases from $N = 3$ to $N = 9$, and the structural entropy

values of $N = 7$ and $N = 9$ are very close. That is, the optimization efficiency of structural entropy decreases with increasing N. To seek a balance between efficiency and effectiveness, we take the compromise value $N = 5$.

6.3.4. Update Threshold Analysis

In scenarios with minimal changes, updating structural information might not be necessary. In this part, we set a threshold for the magnitude of graph changes before initiating updates to cut total time consumption. Specifically, the updated structural entropy will not be calculated until the incremental edge number exceeds a certain percentage (referred to as the update threshold θ) of the edge number of the last updated graph. As we can see from Table 6, the total time reduces 63%-87% with *NAGA+AIUA* and 47%-72% with *NSGA+AIUA* when θ is set from 5% to 20%. Meanwhile, the fluctuation of the final structural entropy remains within 0.15%, which indicates that the threshold has little impact on the precision of structural entropy. Overall, the setting of the updated threshold leads to improved efficiency and better adaptation to graphs undergoing frequent alterations.

Table 6: The Influence on Total Time Consumption and Structural Entropy of Different Updated Threshold θ on Cit-HepPh dataset with Infomap static method.

Updated Threshold		$\theta = 0\%$	$\theta = 5\%$	$\theta = 10\%$	$\theta = 15\%$	$\theta = 20\%$
NA[1]	Total Time	$32.89s$	$12.20s$ ($\downarrow$ 63%) [2]	$9.16s$ ($\downarrow$ 72%)	$5.64s$ ($\downarrow$ 83%)	$4.44s$ ($\downarrow$ 87%)
	Final SE	10.9736	10.9687 ($\leq$ 0.1%) [3]	10.9690 ($\leq$ 0.1%)	10.9668 ($\leq$ 0.1%)	10.9681 ($\leq$ 0.1%)
NS[1]	Total Time	$24.83s$	$13.11s$ ($\downarrow$ 47%)	$10.52s$ ($\downarrow$ 58%)	$8.27s$ ($\downarrow$ 67%)	$6.93s$ ($\downarrow$ 72%)
	Final SE	10.0269	10.0243 ($\leq$ 0.1%)	10.0186 ($\leq$ 0.1%)	10.0352 ($\leq$ 0.1%)	10.0419 ($\leq$ 0.15%)

[1] "NA/NS" represents *NAGA/NSGA+AIUA*.
[2] The percentage reduction in time consumption compared to the case where $\theta = 0\%$.
[3] The percentage fluctuation of the final structural entropy compared to the case where $\theta = 0\%$.

6.3.5. Robustness Analysis

In this subsection, we evaluate the robustness of the proposed framework on 5 new artificial datasets with different noise conditions. Specifically, we generate 5 initial states using the random partition method mentioned in Section 6.1 with p_{ac} (representing the noise) ranging from 0.01 to 0.05 and $\mathcal{S} = [100, 100, 200, 200, 200]$. For the convenience of comparison, the total edge number of the initial state is kept nearly the same, fluctuating between 37357 and 38175, by manually selecting the appropriate p_{in}. After the generation of the 5 initial states, we use the Hawkes Process to generate

74714 incremental edges, around 2 times the initial edges, and split them into 20 sub-sequences as the incremental of 20 time stamps. Then we use *NAGA/NSGA+AIUA* and *TOA* to calculate the structural entropy for all 20 updated graphs and the results are shown in Fig. 14.

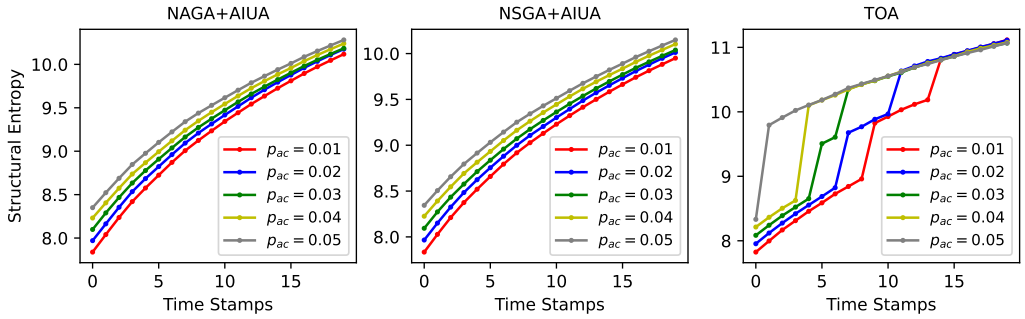

Figure 14: The structural entropy of *NAGA/NSGA+AIUA* and *TOA* with Infomap under different initial state parameter p_{ac} representing different noise conditions.

As we can see from Fig. 14, the structural entropy curves using *NAGA/NSGA+AIUA* are smooth no matter what p_{ac} takes. However, one or two jumps in structural entropy take place when it comes to *TOA*. Furthermore, the higher the noise (p_{ac}) is, the earlier and larger the structural entropy mutates, and the more unstable the *TOA* algorithm is. Additionally, upon examination, the number of communities does not change, meaning that these abrupt changes do not stem from changes in the number of communities, but due to dramatic changes in their content. Therefore, we can conclude that our algorithm maintains the structural entropy at a stable and lower level due to the property of keeping the original community structure from changing drastically, showing high robustness to the increasing noise.

In addition, we note that the structural entropy curve gradually shifts upward as p_{ac} increases. This is because the numbers of edges of the initial states have slight errors. Specifically, the larger the p_{ac}, the more initial edges there are (from 37357 to 38175).

6.3.6. Gap between Incre-2dSE and the Current Static Structural Entropy Measurement Method

In this part, we try to study the gap between *Incre-2dSE* and the current static algorithms. The current mainstream static algorithm for structural entropy measurement is named *structural entropy minimization (SEM)* [6],

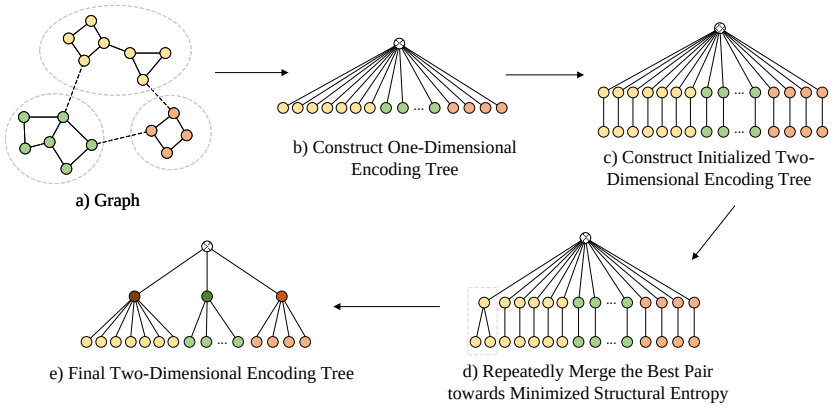

Figure 15: An illustration of two-dimensional structural entropy minimization process.

which is a greedy k-dimensional encoding tree construction algorithm for static graphs whose objective function is structural entropy. Fig. 15 gives an illustration of the *SEM* algorithm in two-dimensional cases (*2d-SEM*). Fig. 15(a) is an example graph. We first construct a one-dimensional encoding tree for this graph (Fig. 15(b)). The one-dimensional encoding tree has one root and $|\mathcal{V}|$ leaf nodes. Next, we add a successor node to each leaf node to construct an initialized two-dimensional encoding tree (Fig. 15(c)). Then we select the best pair of 1-height nodes to merge them which minimizes the structural entropy (Fig. 15(d)). At last, We repeat this merging step until the structural entropy doesn't go down to get the final two-dimensional encoding tree (Fig. 15(e)).

We measure the structural entropy of *Incre-2dSE* (*NAGA/NSGA+AIUA*) and *2d-SEM*[2] through all timestamps like Section 6.3.1 on the six datasets mentioned in Table 3. According to our experimental results (Fig. 16), there is a gap between the structural entropy of *2d-SEM* and our dynamic framework *Incre-2dSE* in some cases. Specifically, on all datasets except DBLP, the structural entropy of *NAGA+AIUA* is higher than but very close to *2d-SEM*. On DBLP, *NAGA+AIUA* is better than *2d-SEM*. On the contrary, *NSGA+AIUA* performs significantly better than *2d-SEM* on all datasets. In addition, the gap between *NSGA+AIUA* and *2d-SEM* is gradually increasing on Cit-HepPh and Facebook, is gradually decreasing on Random-Hawkes and Gaussian-Hawkes, and remains almost unchanged on DBLP and SBM-

[2]https://github.com/RingBDStack/SITool/tree/main/python

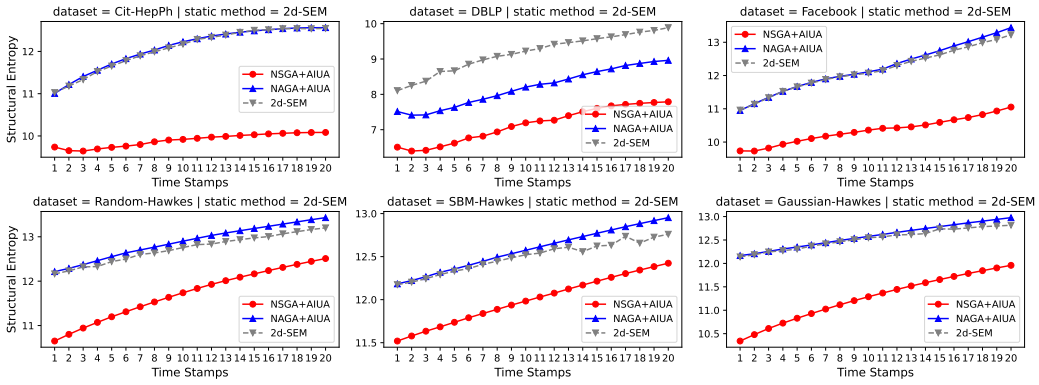

Figure 16: An illustration of two-dimensional structural entropy minimization process.

Hawkes. The reason why there is a gap between *Incre-2dSE* and *2d-SEM* is the same as the reason why there is a gap between *Incre-2dSE* and *TOA* in Section 6.3.1—*Incre-2dSE* updates the community and encoding tree locally and incrementally while *2d-SEM* constructs encoding trees globally from scratch for each snapshot.

6.3.7. 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 $p_{\text{n}} = 1 - p_{pin} - p_{pac}$. We then count the Local Difference and its upper bound. The results are shown in Fig. 17. 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.

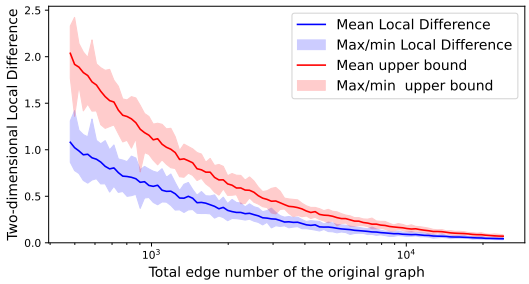

Figure 17: The statistics of the Local Difference and its upper bound.

6.3.8. One-Dimensional Structural Entropy Measurement of Directed Weighted Graphs

In this subsection, we evaluate the time consumption of the two approximate one-dimensional structural entropy measurement methods, Global Aggregation and Local Propagation, on two artificial datasets, i.e., Erdős-Rényi (ER) [30] and Cycle.

ER initial state. The initial state of the ER dataset is created by the Erdős-Rényi model implemented by *erdos_renyi_graph* in Networkx [22] which has two main parameters. One is the number of nodes n and the other is the probability of edge creation p. In this dataset, the Erdős-Rényi model chooses each of the $n(n-1)$ possible directed edges with probability p.

Cycle initial state. The Cycle dataset's initial state contains cyclically connected nodes, where the edge direction is in increasing order. For example, a cycle dataset $\{v_0, v_1, ..., v_n\}$ has $n+1$ direct edges $\{(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n), (v_n, v_0)\}$.

Weight and incremental settings. For the initial states of ER and Cycle datasets, we first set all edge weights as random integers in $[1, 10]$. Then we generate several incremental sequences with sizes 500, 2000, 5000, and 10000, respectively, corresponding to four different updated graphs from each initial state. Specifically, for each incremental edge, we randomly choose two nodes to add a direct edge between them with random weight in $[1, 10]$. If the edge exists, we randomly change its weight as another value in $[1, 10]$.

Experimental settings. For each initial state, we use Global Aggregation with iteration number 50 to calculate the initial stationary distribution from a uniform distribution. For each initial state, we then use Global Aggregation and Local Propagation to iteratively calculate the structural entropy of the 4 updated graphs and record the time consumption of each iteration.

Experimental results. The time consumption experimental results are shown in Fig. 18. On the ER dataset, Local Propagation is faster than Global Aggregation only in the first several iterations. Besides, the time consumption is lower when n is smaller. This is because the time consumption of each iteration of Local Propagation is approximately linear with the size of the involved node set. The smaller the incremental size n, the number of the involved nodes is less. However, since the mean number of the successors of ER graphs is more than 1, the involved nodes will rapidly expand into the entire node set. Therefore, when the iteration number is larger than 5, the efficiency of Local Propagation is nearly the same as that of Global Aggregation. On the contrary, in the Cycle dataset, the mean number of the direct successors of each node is nearly 1. That is, the size of the involved node set will not change, so the time consumption of Local Propagation will not increase as the iteration number grows. In addition, obviously the larger the incremental size n, the higher the time consumption whatever the iteration number is.

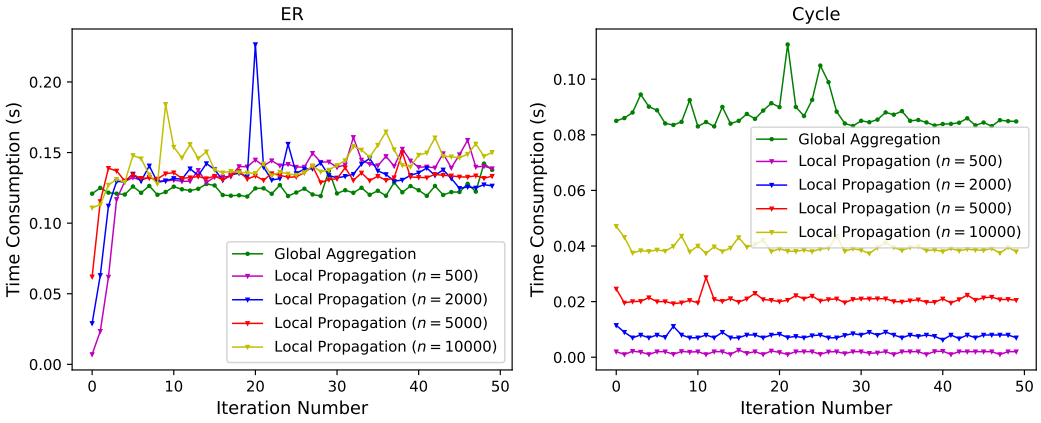

Figure 18: Time consumption of Global Aggregation and Local Propagation on ER and Cycle dataset. Note that only the time consumption of Global Aggregation of $n = 500$ is shown since the theoretical time consumption of different incremental size n is the same.

7. 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 [31] introduced the concept of "structural entropy of network ensembles", known as the Gibbs entropy. Anand et al. [32] proposed the Shannon entropy for network ensembles. Braunstein et al. [33] proposed the von Neumann graph entropy based on the combinatorial graph Laplacian matrix. These three metrics [31, 32, 33] 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 [34]. Recently, many efforts have been devoted to capturing the dynamic changes of graphs, e.g., the research based on the Algorithmic Information Theory [35, 36]. 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. This method is widely used in the fields of graph learning [14], reinforcement learning [37], and social networks [38].

Fast Computation for Graph Entropy. Chen et al. [39] 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. [40, 41] 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 on undirected graphs. In this paper, we mainly focus on the incremental computation for two-dimensional structural entropy based on our dynamic adjustment strategies for encoding trees. Besides, we also discuss the computation method for one-dimensional structural entropy on directed weighted graphs.

8. 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. Further, we discuss the extension of our proposed methods to weighted graphs and the

one-dimensional structural entropy computation on directed and weighted graphs. In the future, we aim to develop more dynamic adjustment strategies for hierarchical community partitioning and incremental measurement algorithms for higher dimensional structural entropy.

Acknowledgments

This work is supported by the National Key R&D Program of China through grant 2022YFB3104703, NSFC through grants 62322202 and 61932002, Beijing Natural Science Foundation through grant 4222030, Guangdong Basic and Applied Basic Research Foundation through grant 2023B1515120020, CCF-DiDi GAIA Collaborative Research Funds for Young Scholars, and the Fundamental Research Funds for the Central Universities.

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 NeuIPS, 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] Z. Yang, G. Zhang, J. Wu, J. Yang, Q. Z. Sheng, H. Peng, A. Li, S. Xue, J. Su, Minimum entropy principle guided graph neural networks, in: Proceedings of the sixteenth ACM international conference on web search and data mining, 2023, pp. 114–122.

[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] D. Zou, H. Peng, X. Huang, R. Yang, J. Li, J. Wu, C. Liu, P. S. Yu, Segsl: A general and effective graph structure learning framework through structural entropy optimization, in: Proceedings of the ACM Web Conference 2023, 2023, pp. 499–510.

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

[16] 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.

[17] 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.

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

[19] S. Fortunato, Community detection in graphs, Physics Reports 486 (3-5) (2009).

[20] U. Brandes, M. Gaertler, D. Wagner, Experiments on graph clustering algorithms, in: European symposium on algorithms, Springer, 2003, pp. 568–579.

[21] P. W. Holland, K. B. Laskey, S. Leinhardt, Stochastic blockmodels: First steps, Social networks 5 (2) (1983) 109–137.

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

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

[24] 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.

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

[26] 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.

[27] D. A. Bader, H. Meyerhenke, P. Sanders, D. Wagner, Graph partitioning and graph clustering, Vol. 588, American Mathematical Society Providence, RI, 2013.

[28] 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.

[29] G. Csardi, T. Nepusz, The igraph software, Complex syst 1695 (2006) 1–9.

[30] P. ERDdS, A. R&wi, On random graphs i, Publ. math. debrecen 6 (290-297) (1959) 18.

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

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

[33] 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.

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

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

[36] 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.

[37] X. Zeng, H. Peng, A. Li, Effective and stable role-based multi-agent collaboration by structural information principles, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 37, 2023, pp. 11772–11780.

[38] Y. Cao, H. Peng, Z. Yu, P. S. Yu, Hierarchical and incremental structural entropy minimization for unsupervised social event detection, in: Proceedings of the AAAI conference on artificial intelligence, 2024, pp. 1–9.

[39] 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.

[40] 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.

[41] 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.