A Temporal Network Embedding Method with Error Bound

Dezheng Li

Somerville College

University of Oxford

A thesis submitted for the degree of

MSc in Mathematical Modelling and Scientific Computing

Trinity 2025

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Prof. Renaud Lambiotte, for his generous encouragement and invaluable insights. I am immensely thankful for his patience; he is a truly nice person. I am also deeply thankful to Dr. Kathryn Gillow for her extensive support, for responding to my emails, and for organising the wonderful MMSC course.

I would like to extend my sincere thanks to my parents and to Yubo Cai at MIT, for their unwavering support during difficult times. Their encouragement played a crucial role in helping me fight against insomnia and depression. Without their encouragement and mental support, I would not be able to complete this dissertation.

Finally, I want to thank my computer for never crashing throughout this year because of my poorly optimised codes.

Abstract

Building upon the matrix decomposition-based static embedding method NetMF, we design an algorithm, ATNMF, which accelerates temporal network embedding by employing an online update and offline matrix decomposition. In the online phase, ATNMF incrementally maintains several small or sparse matrices that approximate the evolving NetMF matrix structure, which allows fast updates in response to incoming events. In the offline phase, we apply either randomised SVD (rSVD) or Adaptive Cross Approximation (ACA) to compute the final low-rank embeddings. We further examine the role of the filter function inherent in NetMF and extend it to alternative polynomial filter families, thereby enhancing its adaptability to various network structures. Beyond proposing the algorithm itself, we also provide explicit upper error bounds for both single-step and multi-step updates.

Numerical experiments demonstrate that ATNMF embeddings exhibit greater stability and comparable or superior quality compared to several baselines (DNE, CTDNE and TGN). Furthermore, without hardware acceleration, ATNMF reduces the average update time per event to sub-millisecond levels for real-world temporal networks. On certain networks with particular structures, ATNMF with appropriately selected filters achieves significantly better embedding quality than baseline methods.

Contents

List of Figures

Chapter 1

Introduction

Network embedding aims to map nodes in a graph onto a low-dimensional Euclidean space while preserving the original graph's structural relationships and semantic adjacency. Specifically, given a network's structural information (such as adjacency matrices or normalised Laplacian) and node attributes, embedding methods learn a function that assigns each node to a vector such that adjacent or structurally similar nodes are mapped to nearby points in the embedding space. These low-dimensional representations facilitate visualisation and analysis of network structures, and also serve as effective and efficient inputs for downstream tasks such as link prediction, node classification, clustering, and recommendation. For instance, one of the simplest embedding methods involves performing k-order truncated Eigenvalue Decomposition (EVD) on the Laplacian matrix[1].

Mainstream static network embedding methods can be roughly categorised into three classes: matrix factorisation, random walk, and deep neural networks[2]. Matrix factorisation methods are inherently suited to network embedding: network structures can be described using adjacency matrices (or other graph-based similarity matrices), and matrix factorisation precisely involves compressing high-dimensional matrices into low-rank matrices to learn low-dimensional representations of nodes. Inspired by natural language processing, random walk methods traverse networks as if performing a random walk, treating nodes as words and the generated node sequences as sentences. Subsequently, word2vec is employed to transform nodes (words) into vectors, technically utilising Skip-gram with Negative Sampling (SGNS) for efficient node vector training. Deep neural network (DNN) methods, on the other hand, leverage deep learning architectures that have achieved success in other domains to learn node embedding. Notably, Qiu et al. [3] showed that four popular SGNS-based methods—DeepWalk, LINE, node2vec, and PTE can all be interpreted as factorising specific target matrices. This theoretical unification bridges random walk-based

and matrix factorisation-based methods, offering a deeper understanding and new perspective of the former one.

However, many real-world networks are inherently dynamic, with evolving structures over time such as social networks and communication networks. In such settings, nodes and edges are constantly added, removed, or updated. Applying static embedding methods by collapsing time information into a snapshot often leads to inefficiencies: every network update would require a new embedding process, which is prohibitive for large-scale or rapidly evolving graphs.

To address this, temporal network embedding (TNE or DNE for dynamic network embedding) has emerged as a growing area of research, with the central goal of balancing embedding quality and update efficiency. Many dynamic embedding methods extend static techniques into the temporal domain, and thus can also be broadly categorised into matrix factorisation-based, random walk-based, and deep learning-based approaches.

Matrix decomposition methods on dynamic networks frequently employ matrix perturbation to model temporal changes in the network (e.g., DHPE[4], DANE[5]), whilst methods based on random walks often implement local updates through incorporating random walks in the temporal dimension (CTDNE[6]) or performing (or equivalently implementing) SGNS on affected nodes after updates (DNE[7]). Deep neural network-based approaches have also experienced explosive growth recently (e.g., DySAT[8] and DyHAN[9]), though these methods often lack theoretical analysis of errors.

1.1 Motivation

Despite significant advances, dynamic network embedding continues to face several fundamental challenges: the first one is the trade-off between accuracy and efficiency. Global updates offer high accuracy but are computationally expensive; localised updates improve efficiency but may lead to error accumulation. This tradeoff becomes particularly severe for high-frequency update streams and large-scale graphs. The second one is to incorporate high-order adjacency information into fast updates. In certain scenarios, such as short-lived bursts of social networks, relying only on first-order adjacency can amplify short-term noise, resulting in unstable and unreliable embeddings. Incorporating higher-order neighbourhood information mitigates this issue, yet this inherently conflicts with the requirement for rapid updates. The

final one is to provide interpretable approximation errors when utilizing approximate updates. These bounds can quantify representational drift and inform when a restart is necessary. NetMF, a static matrix factorisation method proposed by Qiu et al., naturally incorporates high-order proximity and outperforms other factorisation and walk-based methods in several benchmarks. It also has a close connection with DeepWalk, which enables analysis from both random walk and spectral perspectives. Furthermore, NetMF provides a solid theoretical foundation with explicit error bounds. However, the use of high-order information leads to a dense and complex target matrix, which increases computational costs. Thus, extending NetMF to the dynamic setting presents both a meaningful and significant technical challenge.

1.2 Contributions and Dissertation Structure

In this dissertation, we propose an Accelerated Temporal Embedding Method based on NetMF (ATNMF) for undirected dynamic networks without isolated nodes. Our method maintains decomposable low-rank structures of the evolving NetMF target matrix and derives cumulative error bounds over time. ATNMF operates in an online and offline strategy. In the online phase, it incrementally maintains three either compact or sparse matrices M^L, M^R and ΔM, and the volume of current graph, which enables millisecond-level updates, regardless of the size of networks. In the offline phase, when the number of processed events reaches a threshold or receive an embedding requirement, a matrix decomposition will reconstruct the full embedding, avoiding repeated computation while preserving interpretability and robustness.

This thesis is structured as follows:

Chapter 2 introduces background on temporal networks and the NetMF framework. Then Chapter 3 explains the spectral filter in NetMF and extends to other forms via providing new interpretations from the DeepWalk perspective. Chapter 4 is the main contribution of this dissertation. It presents one-step and multi-step update formulas for the target matrix, along with their corresponding theoretical error bounds. Chapter 5 formalises the full ATNMF algorithm and analyses its computational complexity. Chapter 6 presents numerical experiment results for link prediction and graph reconstruction tasks, showing that ATNMF is competitive in embedding quality with three other temporal embedding baselines. Finally in Chapter 7, we conclude the dissertation and outline directions for future work.

1.3 AI Usage and Code

1.3.1 Gen AI Usage

In this dissertation, ChatGPT-5 was used to check for typos and formatting, to provide example code (which is cited in uploaded codes), and for debugging.

1.3.2 Code

ATNMF algorithm code is uploaded to GitHub.

Chapter 2

Background

In this chapter, we briefly introduce temporal network models, Skip-Gram with Negative Sampling (SGNS), DeepWalk, and NetMF, and outline the central problem addressed in this dissertation along with its challenges.

2.1 Temporal Network Models

A temporal (or dynamic) network is, in essence, a network whose topology evolves over time. We begin with a standard definition of a static, unattributed network.

Definition 1 (Network)**.** *A (static and unattributed) network is defined as a tuple* $G = (V, E)$*, where* V *is the set of vertices in the graph,* $E \subseteq V \times V$ *is the set of edges between vertices in the graph.*

Unlike static networks, temporal networks do not admit a universally accepted formal definition. Based on how the evolution is represented, two common models are as follows:

Definition 2 (Snapshot Model of Temporal Network)**.** *A snapshot model of a temporal network* G *is a sequence of network snapshots* $G = \{G_1, G_2, \cdots, G_T\}$ *within a certain time interval, where every snapshot* $G_t = (V_t, E_t)$ *is a static network at time* t*.*

The snapshot model is typically adopted in long term network where structural changes occur at relatively coarse time scales. For example, the AS-733 dataset [10] records 733 autonomous system graphs collected daily from November 8, 1997 to January 2, 2000. In such cases, the graph usually undergoes significant changes between snapshots, and the time intervals are often on the scale of days or months.

In contrast, the event based model describes network evolution as a stream of timestamped events, making it well-suited for representing fine-grained and real-time updates in large-scale systems such as communication or financial networks. Notable examples include the email-Eu-core dataset [11], which records individual email exchanges, and the Bitcoin OTC network [12, 13], which captures peer-to-peer trust transactions. More formally, the event based model of temporal network is defined as:

Definition 3 (Event Based Model of Temporal Networks)**.** *An event based model of a dynamic network G consists of a stream of events with ordered timestamps $\{(u_i, v_i, t_i)|i = 1, 2, \cdots\}$, where u_i and v_i are the node pair on which the event occurs, and t_i is the timestamp.*

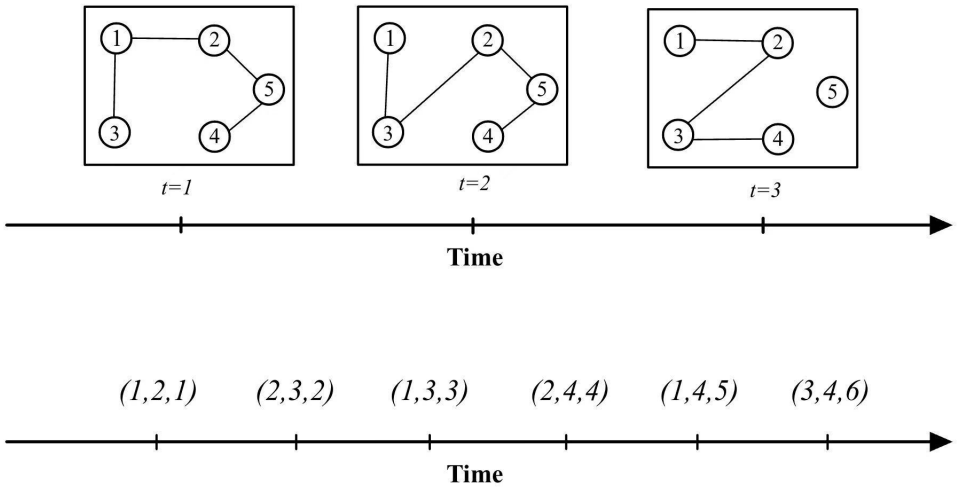

Figure 2.1: Snapshot model of temporal network (top) and event based temporal network model (bottom).

2.2 word2vec, SGNS and DeepWalk

To better understand the construction of NetMF's target matrix, we begin by reviewing the word2vec model, its training technique, Skip-Gram with Negative Sampling (SGNS), and its adaptation to network data via the DeepWalk algorithm.

2.2.1 word2vec and SGNS

The Skip-gram model was originally proposed by Mikolov et al. to learn vectorised representations of words [14] in the field of natural language processing (NLP). In their work, based on Skip-gram model, they aimed to maximise the likelihood of observing context words given a centre word. More formally, let $w_1, \cdots, w_L$ be training words, $v_w \in \mathbb{R}^k$ be words' vector representations, T be the size of training context and W be the number of words in vocabulary, then word2vec is maximising the following average log probability

$$\arg\max_{v_w} \frac{1}{L} \sum_{\ell=1}^{L} \sum_{-T \le j \le T, j \neq 0} \log p(w_{\ell+j} | w_\ell), \tag{2.1}$$

where p in Skip-gram is defined by softmax function

$$p(w_O | w_I) = \frac{\exp(v_{w_O}^\top v_{w_I})}{\sum_{w=1}^{W} \exp(v_w^\top v_{w_I})}. \tag{2.2}$$

Due to the prohibitively high computational cost of solving Eqn.(2.1) directly over a large vocabulary ($W \gg 1$), Mikolov et al. proposed the Skip-Gram with Negative Sampling (SGNS) technique to approximate and accelerate this optimisation. Instead of computing the full softmax, SGNS approximates $\log p(w_o | w_I)$ in the objective by introducing negative samples

$$\log \sigma(v_{w_O}^\top v_{w_I}) + \sum_{i=1}^{b_{\text{neg}}} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-v_{w_i}^\top v_{w_I}) \right], \tag{2.3}$$

in which $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, b_{neg} is the number of negative samples, P_n is a negative noise distribution describing the probability a word pair (w, c) does not exist and w_i is randomly drawn from P_n. This noise distribution $P_n(w)$ is treated as a free parameter in [14]. Mikolov et al. recommended that the $\frac{3}{4}$-rd power of the unigram distribution performs the best among other counterparts.

Intuitively, the first term $\log \sigma(v_{w_O}^\top v_{w_I})$ aims to increase the similarity between true word-context pairs, while the second term reduces the similarity between centre word and b_{neg} negatively sampled words. With this form it is possible to perform gradient descent (GD) or stochastic gradient descent (SGD) to optimise this objective function efficiently.

2.2.2 DeepWalk

DeepWalk extends the SGNS framework from NLP to graphs by interpreting nodes as words and sequences generated by random walks as sentences [15]. The key idea is to perform random walks on the graph to generate sequences of nodes, which serve as synthetic sentences. Sliding windows are then applied over these sequences to construct a node-context corpus, which is subsequently fed into the SGNS model for embedding learning. Alg.1 provides a detailed process of DeepWalk:
This conceptual bridge between random walks and word-context models sets the stage

Algorithm 1: DeepWalk[15]

Input: Graph $G = (V, E)$; walk length L; window size T; number of walks N; start-node distribution P on V; number of negative samples b_{neg}.
Output: Node embeddings $X \in \mathbb{R}^{|V| \times d}$ (via SGNS).

$\mathcal{D} \leftarrow \varnothing$
for $n = 1$ **to** N **do**
  Sample start node $w_1^{(n)} \sim P$
  Generate a random walk on G of length L: $(w_1^{(n)}, \ldots, w_L^{(n)})$
  for $j = 1$ **to** $L - T$ **do**
    for $r = 1$ **to** T **do**
      Add pair $\left(w_j^{(n)}, w_{j+r}^{(n)}\right)$ to $\mathcal{D}$
      Add pair $\left(w_{j+r}^{(n)}, w_j^{(n)}\right)$ to $\mathcal{D}$
Run SGNS on $\mathcal{D}$ with b_{neg} negative samples to obtain embedding X

for a matrix factorisation view of DeepWalk: an insight that underpins the NetMF framework discussed in the next section.

2.3 From DeepWalk to NetMF

On the surface, DeepWalk appears to be a randomised algorithm: it requires random walks to generate the corpus, and in SGNS, computational acceleration is also achieved through random negative sampling. This shows an apparent disparity with classical matrix based embedding methods. However, Qiu et al. demonstrated that, in the limit as the random-walk length $L \to \infty$, DeepWalk is asymptotically equivalent to an implicit factorisation of the so-called *DeepWalk matrix* in [3]. Building upon this, they subsequently proposed the NetMF algorithm and provided a rigorous error analysis.

This work originates from Levy and Goldberg [16], who showed that SGNS is, in fact, implicitly factorising a shifted PMI matrix.

Theorem 1 (Levy and Goldberg [16]). *Let $\mathcal{D}$ be a corpus, w be the centre word and c be its context, b_{neg} be the number of negative samples. Further denote $\#(\cdot)$ as the number of certain objects. Then SGNS is factorizing a shifted* Pointwise Mutual Information (PMI) *matrix, whose entries are*

$$\log\left(\frac{\#((w,c))|\mathcal{D}|}{\#(w)\#(c)}\right) - \log b_{\text{neg}}. \tag{2.4}$$

Here, the PMI is defined by

$$PMI(x,y) = \log\frac{P(x,y)}{P(x)P(y)},$$

which measures the strength of association between w and c.

Based on this, Qiu et al. showed that the first term in Eqn.(2.4) converges to a matrix in probability as $L \to \infty$, and provided the matrix DeepWalk is factorising, which we refer to as the *DeepWalk matrix* M:

Theorem 2 (Qiu et al. [3], Theorem. 2.3). *Let $G = (V, E)$ be an undirected network without isolated nodes. Then for DeepWalk $L \to \infty$,*

$$\frac{\#((w,c))|\mathcal{D}|}{\#(w)\#(c)} \xrightarrow{p} \frac{\text{vol}(G)}{2T}\left(d_c^{-1}\sum_{r=1}^{T}(P^r)_{w,c} + d_w^{-1}\sum_{r=1}^{T}(P^r)_{c,w}\right); \tag{2.5}$$

and matrix that DeepWalk implicitly factorises is

$$\log^\circ M = \log^\circ\left(\frac{\text{vol}(G)}{b_{\text{neg}}T}\sum_{r=1}^{T}P^rD^{-1}\right), \tag{2.6}$$

where $\log^\circ(\cdot)$ is the element-wise logarithm, $P = D^{-1}A$ is the transition probability matrix, D is the degree matrix, d_c, d_w are degree of c, w, A is the adjacency matrix, $\text{vol}(G) = \sum_i D[i,i]$ is the graph volume, b_{neg} is the number of negative samples, and T is the window size.

As the network is undirected, the *Normalised Adjacency Matrix (NAM)* $S = D^{-1/2}AD^{-1/2}$ admits an orthogonal diagonalisation $U\Lambda U^\top$, $U^\top U = I$. Thus, by retaining the top k eigenvalues, the DeepWalk matrix can be written as

$$\log^\circ M = \log^\circ\frac{\text{vol}(G)}{b_{\text{neg}}T}\left(D^{-1/2}\right)\left(U\left(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r\right)U^\top\right)\left(D^{-1/2}\right). \tag{2.7}$$

The term $\frac{1}{T}\sum_{r=1}^{T}\Lambda^r$ acts as a spectral filter that assigns distinct weights to different spectral components; its properties and implications will be analysed in the next chapter.

NetMF is thus proposed by efficiently approximating the DeepWalk matrix through adopting top h eigenpairs on undirected networks without isolated nodes.

NetMF provides implementations for both large and small window sizes T. Since the large-window variant aligns more closely with the spectral filtering and high-order modelling used later in this work, we focus on it in the main text (Alg.2); the small-window implementation is deferred to Appendix A. NetMF begins by computing the

Algorithm 2: NetMF for a Large Window Size T [3]

Input: Graph G, window size T, dimension k, rank h
Output: Embedding matrix $U_d\sqrt{\Sigma_d}$

1 Eigen-decomposition $D^{-1/2}AD^{-1/2} \approx U_k\Lambda_k U_k^\top$
2 Approximate $\widehat{M} = \frac{\mathrm{vol}(G)}{b_{\mathrm{neg}}} D^{-1/2}U_h\Big(\frac{1}{T}\sum_{r=1}^{T}\Lambda_h^r\Big)U_h^\top D^{-1/2}$
3 Compute $\widehat{M}' = \max(\widehat{M}, 1)$
4 Rank-k approximation by SVD: $\log^\circ \widehat{M}' = U_d\Sigma_d V_d^\top$
5 **return** $U_d\sqrt{\Sigma_d}$

top-h eigenpairs of the NAM to approximate original matrix, which can be accelerated by Arnoldi method as NAM is sparse. In the next step, the DeepWalk matrix $\log^\circ M$ is approximated by

$$\log^\circ \widehat{M} = \log^\circ \frac{\mathrm{vol}(G)}{b_{\mathrm{neg}}} D^{-1/2}U_h\Big(\frac{1}{T}\sum_{r=1}^{T}\Lambda_h^r\Big)U_h^\top D^{-1/2}. \tag{2.8}$$

We refer to this matrix as *NetMF matrix*. The truncation $\widehat{M}' = \max(\widehat{M}, 1)$ is applied to avoid excessive density after the logarithm. In the final step, a rank-d singular value decomposition is performed on $\log^\circ \widehat{M}'$ to get the embedding.

A distinguishing advantage of NetMF over other embedding methods is its theoretical interpretability: it provides an explicit upper bound on the approximation error between pre-logged NetMF matrix and pre-logged DeepWalk matrix. This error bound is formally stated in the following theorem.

Theorem 3 (Qiu et al. [3], Theorem 3.5)**.** *Let $\|\cdot\|_F$ be the matrix Frobenius norm. Then*

$$\big\|M - \widehat{M}\big\|_F \;\le\; \frac{\mathrm{vol}(G)}{b_{\mathrm{neg}} d_{\min}} \left(\sum_{j=k+1}^{n}\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_j^r\right|^2\right)^{\frac{1}{2}}, \tag{2.9}$$

and

$$\big\|\log^\circ M' - \log^\circ \widehat{M}'\big\|_F \;\le\; \big\|M' - \widehat{M}'\big\|_F \;\le\; \big\|M - \widehat{M}\big\|_F. \tag{2.10}$$

Both the authors' own experiments and independent benchmarks have demonstrated that NetMF achieves competitive or even superior embedding quality on a variety of downstream tasks such as link prediction and node classification [17–20].

Given these advantages, this dissertation aims to extend the NetMF framework to event based undirected temporal networks without isolated nodes, while preserving its strong theoretical guarantees and error bounds. We primarily face two major challenges:

1. High complexity The element-wise logarithmic transformation in NetMF can result in a dense matrix, pushing the computational complexity beyond $\mathcal{O}(n^2)$. Existing research has highlighted that NetMF's runtime may significantly exceed that of other embedding algorithms[20, 21]. In dynamic settings with high-frequency event streams, embedding updates must be completed within milliseconds to ensure real time responsiveness.

2. Local perturbations propagate globally The term $\sum_{r=1}^{T} P^r$ can cumulate and propagate an edge or node change to the whole graph, causing local changes in the graph to affect the entire embedding. The element-wise logarithm further amplifies this nonlocality, making localised updates ineffective. Efficient update strategies must therefore balance speed and approximation accuracy.

Chapter 3

The Filter in NetMF

In this chapter, we analyse the spectral structure underlying Eqn. (2.7), with particular focus on the term $\frac{1}{T}\sum_{r=1}^{T}\Lambda^r$. This term can be viewed as a spectral filter acting on the eigenvalues of the normalised adjacency matrix $S = D^{-1/2}AD^{-1/2}$. Qiu et al. interpreted this cumulative power term as a purely algebraic formulation of DeepWalk's window-based multi-step transitions, enabling a spectral filtering view of the embedding process. From this perspective, we shall observe how this filtering structure determines spectral preferences in the NAM. Integrating the DeepWalk matrix architecture, we will extend this filter to a more flexible family of filters, which will endow our subsequent algorithms with enhanced embedding quality. Later, the content of this chapter will also assist us with analysing errors.

3.1 Filter Applied in NetMF

We begin by defining the spectral domain in which our analysis takes place. Let $S = D^{-1/2}AD^{-1/2}$ denote the normalised adjacency matrix and $\mathcal{L} = I - S$ be the corresponding normalised Laplacian. Classical graph theory yields the following conclusion:

Theorem 4 ([22]). *For normalised graph Laplacian*

$$\mathcal{L} = I - D^{-1/2}AD^{-1/2},$$

all its eigenvalues are real numbers and lie in $[0, 2]$*, with* $\lambda_{\min}(\mathcal{L}) = 0$.

From which we immediately obtain the spectral radius of S:

Corollary 1. *For normalised adjacency matrix* $D^{-1/2}AD^{-1/2}$*, all its eigenvalues are real and lie in* $[-1, 1]$.

We now revisit the expression $\frac{1}{T}\sum_{r=1}^{T}\Lambda^r$ in Eqn. (2.7). This corresponds to applying a spectral filter of the form

$$f_T(x) = \frac{1}{T}\sum_{r=1}^{T} x^r$$

acts directly on the eigenvalues of S.

This filter has several useful properties:

$$f_T(1) = 1, \qquad f_T(0) = 0, \qquad f_T(-1) = \begin{cases} 0, & \text{if } T \text{ is even,} \\ -\frac{1}{T}, & \text{if } T \text{ is odd.} \end{cases}$$

Over the interval $[0, 1]$, $f_T(x)$ is strictly convex and monotonically increasing, as its second derivative satisfies

$$f_T''(x) = \frac{1}{T}\sum_{r=2}^{T} r(r-1)\,x^{r-2} > 0, \ \forall x \in [0, 1].$$

On the interval $(-1, 0)$, the filter is bounded below by $-1/T$, indicating a strong suppression of negative eigenvalues.

This observation shows that f acts as a spectral filter that emphasises large positive eigenvalues while attenuating both negative and small-magnitude components, with the effect becoming increasingly significant as T grows (Fig. 3.1).

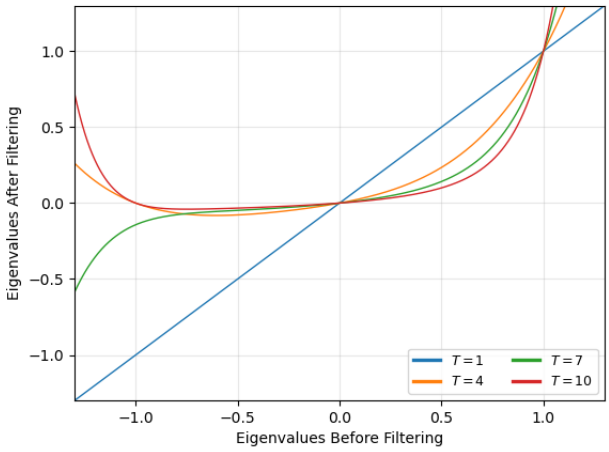

Figure 3.1: The filter $f(x) = \frac{1}{T}\sum_{r}^{T} x^r$. This indicates that higher-order power terms (longer paths) contribute more significantly to the filter output.

Consequently, this filtering mechanism supports the rationale behind NetMF's selection of the top-h eigenpairs: the filtering removes all non-positive eigenvalues,

retaining only the most informative components of the spectrum, which has the same logic as NetMF selecting the most effective eigenvalues.

However, assigning equal weights to all steps r can lead to overemphasis on higher-order terms, i.e., longer paths. This introduces two notable limitations:

1. **Lack of sensitivity to local structures.** Equal weighting treats distant and nearby nodes with the same importance, which dilutes the ability to distinguish local patterns. In networks with clear community structure or local clusters, this "long-range bias" can blur community boundaries and reduce performance on tasks such as node classification or community detection.

2. **Amplification of anomalous paths and noise.** Under uniform weighting, rare but structurally uninformative long-range connections, such as links to outlier nodes, are given equal emphasis. This can amplify noise and degrade the embedding's stability and structural accuracy.

3.2 Variations and Explanation of Filter Applied in NetMF

Back to the DeepWalk matrix and focus on the term:

$$\frac{1}{T}\sum_{r=1}^{T} P^r,\ P = D^{-1}A \tag{3.1}$$

Classical Markov chain theory tells us that the (i, j)-th entry of P^r corresponds to the probability of transitioning from node i to node j in exactly r steps. Thus, Eqn. (3.1) computes the average r-step transition probability over path lengths uniformly sampled from $\{1, \cdots, T\}$.

To generalise this filter, we consider replacing the uniform weights with an adjustable distribution over path lengths:

$$f(P) = \sum_{r=1}^{T} w_r P^r,\ \sum_{r=1}^{T} w_r = 1,\ w_r \geq 0. \tag{3.2}$$

Here, $\{w_r\}$ reflects a user-specified sampling distribution over path lengths. This allows the model to align better with the structural scales relevant to downstream tasks and offers greater flexibility.

Type	Unnormalised weights w_r $(r = 1, \dots, T)$	Notes
	Near-head	
Exponential (head)	$w_r = \exp\big(-\gamma\,(r-1)\big)$	Larger γ weights more on near-head.
Power (head)	$w_r = (T+1-r)^{\,a}$	Larger a weights more on near-head.
Sigmoid (head)	$w_r = \big(1+\exp\big(\frac{r-r_0}{\tau}\big)\big)^{-1}$	Cutoff at r_0; Smaller τ weights more on near-head.
	Far-head	
Power (tail)	$w_r = r^{\,p}$	Larger p weights more on far-head.
Gaussian (tail)	$w_r = \exp\Big(-\dfrac{(T-r)^2}{2\,\sigma^2}\Big)$	Reach maximum at T; smaller σ weights more on far-head.

Table 3.1: Weighting schemes for polynomial filters. Normalise before use: $\tilde{w}_r = w_r \big/ \sum_{s=1}^{T} w_s$; build $f(x) = \sum_{r=1}^{T} \tilde{w}_r\, x^r$.

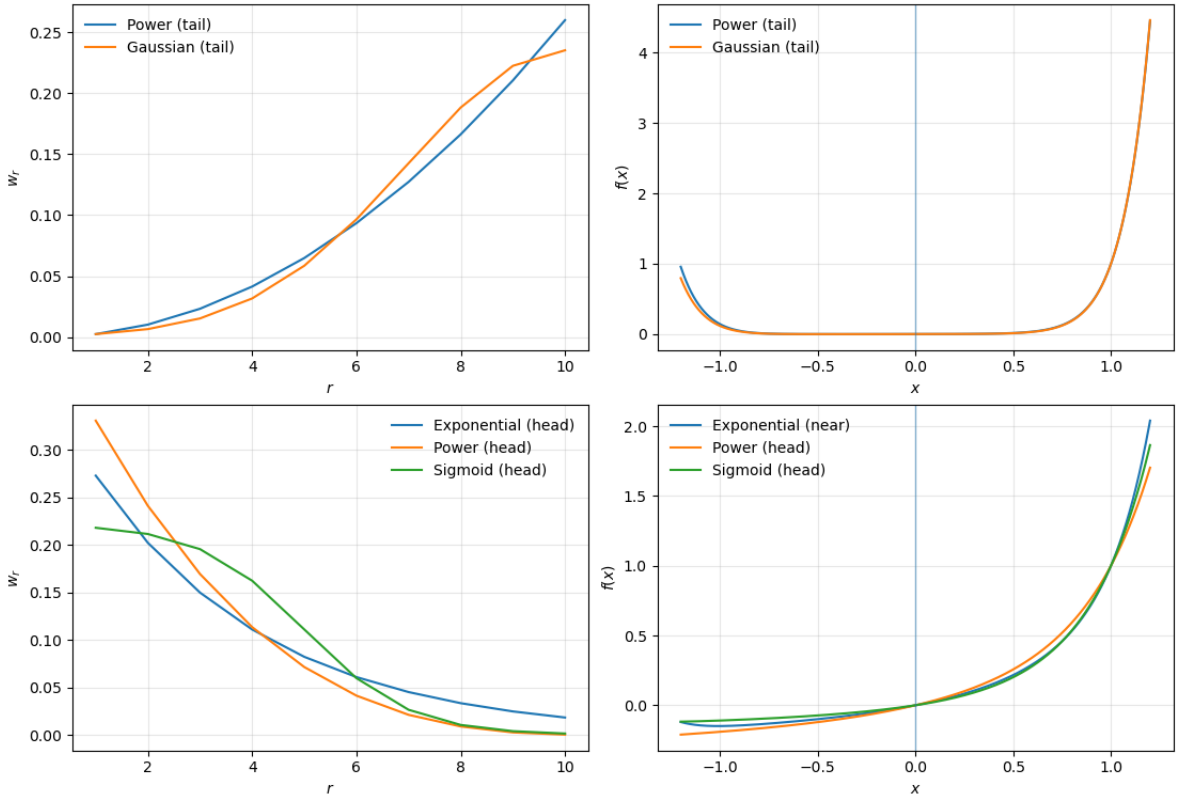

Figure 3.2: Normalised weights and filtering effects of different filters

Table 3.1 summarises several candidate weight distributions. Each of them defines

a polynomial filter with a distinct spectral shape, as illustrated in Fig. 3.2. These filters allow more flexible control over the relative importance of short and long range structures for a fixed window size T.

Filters with decaying weights produce smoother gain near $x = 1$, preserving sensitivity to small and medium eigenvalues. This enhances responsiveness to local structures by assigning more weight to short paths, thereby emphasizing nearby nodes and helping retain sharp features such as community boundaries. Such filters also reduce the influence of long-range noise and improve numerical stability.

In contrast, far-head weights concentrate their response near $x = 1$, highlighting high-order paths and suppressing weaker spectral components. These filters focus on global structures and are better suited for tasks requiring long-range relational modelling, especially in large, randomly connected networks.

Introduced f, we rewrite the NetMF matrix as

$$\log^{\circ} \frac{\mathrm{vol}(G)}{b_{\mathrm{neg}}} D^{-1/2} U_h f(\Lambda_h) U_h^{\top} D^{-1/2}$$

and will use this form for later discussion.

Chapter 4

Event Based Update Scheme

This chapter will present the main works of this dissertation: the update rules and theoretical error bounds for event based temporal networks. In our plan, rather than finding an explicit update scheme for NetMF matrix directly, we focus on approximating the updates to its components, i.e., orthonormal matrix U_h and diagonal matrix Λ_h. As we show later, these can be obtained efficiently by approximating orthogonal diagonalisation of the updated normalised adjacency matrix $S = D^{-1/2}AD^{-1/2}$.

In Sec.4.1 and 4.2, we will introduce the necessary preliminaries required for our analysis. We then derive the approximate update formula for the orthogonal diagonalisation of NAM after a single event, along with its associated upper error bound. Finally, we extend this to a multi-step setting and establish cumulative update rules together with corresponding error bounds.

4.1 Orthogonal Diagonalisation Updating Method

4.1.1 Simon-Zha's Updating Method

Latent Semantic Indexing (LSI), proposed by [23], is a mature text mining technique designed to retrieve documents relevant to user queries from a given collection. Zha and Simon (1999) give exact, projection-based partial-SVD updates for three natural LSI changes[24]:

(i) Appending new documents (add columns)

$$A \to \begin{pmatrix} A & D \end{pmatrix},$$

where $A \in \mathbb{R}^{m\times n}$ is the original term–document, $D \in \mathbb{R}^{m\times p}$ denotes p new documents.

(ii) Appending new terms (add rows)

$$A \to \begin{pmatrix} A \\ T \end{pmatrix}.$$

Here $T \in \mathbb{R}^{n\times q}$ denotes the q new terms.

(iii) Correcting term weights (structured low-rank edit)

$$A \to A + YZ^\top,$$

where $Z \in \mathbb{R}^{n\times j}$ indicates weight updates and $Y \in \mathbb{R}^{m\times j}$ selects terms need to adjust.

Their method unifies all three update types by following a three-step procedure: (i) expand the original SVD subspace with its orthogonal complement; (ii) form and perform SVD on a much smaller core matrix; (iii) map the results back to obtain the updated SVD. This update strategy is then understood as a Rayleigh-Ritz projection method, which approximates the SVD update within a reduced subspace [25]. We now present their methods as follows.

Let the current rank-k truncated SVD be

$$A_k \;=\; U_k \Sigma_k V_k^\top, \qquad U_k^\top U_k = V_k^\top V_k = I_k.$$

The aim is to find a triple $\widetilde{\Sigma}_k,\ \widetilde{U}_k,\ \widetilde{V}_k$ to approximate updated matrix as $\widetilde{U}_k \widetilde{\Sigma}_k \widetilde{V}_k^\top$. All QR factorisations below are thin; all SVDs of small cores are compact.

1. Appending new documents (columns). Given $D \in \mathbb{R}^{m\times p}$, form

$$W_D \;=\; U_k^\top D, \qquad (I - U_k U_k^\top)\, D \;=\; Q_D R_D, \quad Q_D^\top Q_D = I_p.$$

Build the $(k{+}p) \times (k{+}p)$ core and perform SVD

$$H_D \;=\; \begin{bmatrix} \Sigma_k & W_D \\ 0 & R_D \end{bmatrix} \;=\; E\,\Theta\,H^\top \quad \text{(SVD)}.$$

Then the updated triplet is

$$\boxed{\widetilde{\Sigma}_k = \Theta, \quad \widetilde{U}_k = \begin{bmatrix} U_k & Q_D \end{bmatrix} E, \quad \widetilde{V}_k = \begin{bmatrix} V_k & 0 \\ 0 & I_p \end{bmatrix} H\ .}$$

2. Appending new terms (rows). Given $T \in \mathbb{R}^{q\times n}$, form

$$Y_T \;=\; T\,V_k, \qquad (I - V_kV_k^\top)\,T^\top \;=\; Q_T R_T, \quad Q_T^\top Q_T = I_q.$$

Build the $(k{+}q)\times(k{+}q)$ core and perform SVD

$$H_T \;=\; \begin{bmatrix}\Sigma_k & 0\\ Y_T & R_T\end{bmatrix} \;=\; E\,\Theta\,H^\top.$$

Then

$$\boxed{\widetilde{\Sigma}_k = \Theta, \quad \widetilde{U}_k = \begin{bmatrix}U_k & 0\\ 0 & I_q\end{bmatrix} E, \quad \widetilde{V}_k = \begin{bmatrix}V_k & Q_T\end{bmatrix} H\ .}$$

3. Correcting term weights (additive low-rank edit). Suppose $A_{\text{new}} = A + YZ^\top$ with $Y \in \mathbb{R}^{m\times j}$ and $Z \in \mathbb{R}^{n\times j}$. Compute QRs

$$(I - U_kU_k^\top)\,Y \;=\; Q_Y R_Y, \qquad (I - V_kV_k^\top)\,Z \;=\; Q_Z R_Z,$$

and set

$$E_1 \;=\; U_k^\top Y \in \mathbb{R}^{k\times j}, \qquad E_2 \;=\; Z^\top V_k \in \mathbb{R}^{j\times k}.$$

Build the $(k{+}j)\times(k{+}j)$ dense core and perform SVD

$$H_{YZ} \;=\; \begin{bmatrix}\Sigma_k & 0\\ 0 & 0\end{bmatrix} \;+\; \begin{bmatrix}E_1\\ R_Y\end{bmatrix}\begin{bmatrix}E_2 & R_Z^\top\end{bmatrix} \;=\; F\,\Theta\,G^\top.$$

Then

$$\boxed{\widetilde{\Sigma}_k = \Theta, \quad \widetilde{U}_k = \begin{bmatrix}U_k & Q_Y\end{bmatrix} F, \quad \widetilde{V}_k = \begin{bmatrix}V_k & Q_Z\end{bmatrix} G\ .}$$

4.1.2 Orthogonal Diagonalisation Methods for Low Rank Symmetric Updating

Inspired by Zha's work, we provide a Rayleigh–Ritz-styled approximation for orthogonal diagonalisation of symmetric adding one node or weight changing updates, along with an associated error bound.

Lemma 1 (Adding one node)**.** *Let $A \in \mathbb{R}^{n\times n}$ be symmetric with a known k-term orthogonal diagonalisation approximation*

$$A \;=\; U\Lambda U^\top + \mathrm{Res}, \qquad U^\top U = I_k, \quad \Lambda = \mathrm{diag}(\lambda_1,\ldots,\lambda_k), \quad \mathrm{Res} = \mathrm{Res}^\top,$$

and define the $(n{+}1)\times(n{+}1)$ augmentation with a vector $b \in \mathbb{R}^n$ by

$$A_+ = \begin{pmatrix}A & b\\ b^\top & 0\end{pmatrix}.$$

Write $v = (I - UU^\top)b$, $\alpha = \|v\|_2$, and set $r := \mathrm{rank}(v) \in \{0,1\}$.

(i) $r = 1$ *(i.e.,* $\alpha > 0$*). Let* $q = v/\alpha$ *and*

$$Q_+ = \begin{pmatrix} U & q & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(k+2)},$$

and define the core matrix

$$C = \begin{pmatrix} \Lambda & 0 & U^\top b \\ 0 & 0 & \alpha \\ b^\top U & \alpha & 0 \end{pmatrix} \in \mathbb{R}^{(k+2)\times(k+2)}.$$

(ii) $r = 0$ *(i.e.,* $\alpha = 0$*). Let*

$$Q_+ = \begin{pmatrix} U & 0 \\ 0 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(k+1)},$$

$$C = \begin{pmatrix} \Lambda & U^\top b \\ b^\top U & 0 \end{pmatrix} \in \mathbb{R}^{(k+1)\times(k+1)}.$$

In either case, let $C = U_C \Gamma U_C^\top$ *be an eigen-decomposition. Then*

$$(Q_+ U_C)\,\Gamma\,(Q_+ U_C)^\top$$

is an orthogonal diagonalisation of a rank-$(k + r + 1)$ *approximation to* A_+ *and, for every unitarily invariant norm* $\|\cdot\|$*,*

$$\left\| A_+ - (Q_+ U_C)\Gamma(Q_+ U_C)^\top \right\| = \|\mathrm{Res}\|.$$

Proof. We prove the two cases separately.

Case $r = 1$. As for the first claim, simply check that

$$(Q_+ U_C)^\top (Q_+ U_C) = I_{k+2},$$

which indicates $(Q_+ U_C)\,\Gamma\,(Q_+ U_C)^\top$ is a valid orthogonal diagonalisation of rank $k+2$ approximation to A_+.

For the second one, since $b = UU^\top b + v = UU^\top b + \alpha q$, a direct multiplication gives

$$Q_+ C Q_+^\top = \begin{pmatrix} U & q & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Lambda & 0 & U^\top b \\ 0 & 0 & \alpha \\ b^\top U & \alpha & 0 \end{pmatrix} \begin{pmatrix} U^\top & 0 \\ q^\top & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} U\Lambda U^\top & b \\ b^\top & 0 \end{pmatrix}.$$

Therefore,

$$A_+ - Q_+ C Q_+^\top = \begin{pmatrix} A & b \\ b^\top & 0 \end{pmatrix} - \begin{pmatrix} U\Lambda U^\top & b \\ b^\top & 0 \end{pmatrix} = \begin{pmatrix} \mathrm{Res} & 0 \\ 0 & 0 \end{pmatrix}.$$

By unitary invariance of the norm and block-diagonality, $\|A_+ - Q_+ C Q_+^\top\| = \|\mathrm{Res}\|$.

Case $r = 0$. Here $v = 0$ so $b \in \mathrm{range}(U)$, and with the given Q_+ and C we have

$$Q_+ C Q_+^\top = \begin{pmatrix} U & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \Lambda & U^\top b \\ b^\top U & 0 \end{pmatrix} \begin{pmatrix} U^\top & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} U\Lambda U^\top & b \\ b^\top & 0 \end{pmatrix}.$$

The same subtraction as above yields $A_+ - Q_+ C Q_+^\top = \begin{pmatrix} \mathrm{Res} & 0 \\ 0 & 0 \end{pmatrix}$, hence $\|A_+ - (Q_+ U_C)\Gamma(Q_+ U_C)^\top\| = \|\mathrm{Res}\|$ for any unitarily invariant norm. □

Lemma 2 (Rank-2 weights update)**.** *Let $A \in \mathbb{R}^{n\times n}$ be symmetric with a known k-term spectral approximation*

$$A \;=\; U\Lambda U^\top + \mathrm{Res}, \qquad U^\top U = I_k, \quad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_k), \quad \mathrm{Res} = \mathrm{Res}^\top,$$

and let $x, y \in \mathbb{R}^n$. Set $X = [\,x \;\; y\,] \in \mathbb{R}^{n\times 2}$ and define the symmetric rank-2 update

$$A_+ \;=\; A + xy^\top + yx^\top \;=\; A + XSX^\top, \qquad S := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Write a thin QR factorisation on the U-orthogonal component

$$(I - UU^\top)X \;=\; Q_X R_X, \qquad Q_X^\top Q_X = I, \quad U^\top Q_X = 0,$$

and set $r := \mathrm{rank}\big((I - UU^\top)X\big) \in \{0, 1, 2\}$. Let $B := U^\top X \in \mathbb{R}^{k\times 2}$.

(i) $r = 2$. Write $Q_X = \begin{pmatrix} q_1 & q_2 \end{pmatrix} \in \mathbb{R}^{n\times 2}$ and $R_X \in \mathbb{R}^{2\times 2}$ upper triangular with nonzero diagonal. Define

$$Q_+ \;=\; \begin{pmatrix} U & q_1 & q_2 \end{pmatrix} \in \mathbb{R}^{n\times(k+2)},$$

$$C \;=\; \begin{pmatrix} \Lambda & 0 \\ 0 & 0_{2\times 2} \end{pmatrix} \;+\; \begin{pmatrix} B \\ R_X \end{pmatrix} S \begin{pmatrix} B \\ R_X \end{pmatrix}^\top \in \mathbb{R}^{(k+2)\times(k+2)}.$$

(ii) $r = 1$. Write $Q_X = q \in \mathbb{R}^n$ and $R_X \in \mathbb{R}^{1\times 2}$. Define

$$Q_+ \;=\; \begin{pmatrix} U & q \end{pmatrix} \in \mathbb{R}^{n\times(k+1)},$$

$$C \;=\; \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} \;+\; \begin{pmatrix} B \\ R_X \end{pmatrix} S \begin{pmatrix} B \\ R_X \end{pmatrix}^\top \in \mathbb{R}^{(k+1)\times(k+1)}.$$

(iii) $r = 0$ (i.e., $x, y \subset \mathrm{span}(U)$). Set

$$Q_+ \;=\; U \in \mathbb{R}^{n\times k}, \qquad C \;=\; \Lambda \;+\; BSB^\top \in \mathbb{R}^{k\times k}.$$

In any case, let $C = U_C \Gamma U_C^\top$ be an eigen-decomposition. Then

$$(Q_+ U_C)\,\Gamma\,(Q_+ U_C)^\top$$

is an orthogonal diagonalisation of a rank-$(k+r)$ approximation to A_+ and, for every unitarily invariant norm $\|\cdot\|$,

$$\left\|A_+ - (Q_+U_C)\Gamma(Q_+U_C)^\top\right\| \;=\; \|\mathrm{Res}\|.$$

Proof. Similar with Lemma 1. □

4.2 Weyl's Inequality and Its Useful Corollary

In this section, we present essential tools for the subsequent analysis. We begin with the classical Weyl's inequality, which characterises how the eigenvalues of Hermitian matrices behave under additive perturbations:

Theorem 5 (Weyl)**.** *Let $A, B \in \mathbb{C}^{n\times n}$ be Hermitian matrices with eigenvalues*

$$\lambda_1(\cdot) \geq \lambda_2(\cdot) \geq \cdots \geq \lambda_n(\cdot).$$

Then for each $k = 1, \ldots, n$,

$$\lambda_k(A) + \lambda_n(B) \;\leq\; \lambda_k(A+B) \;\leq\; \lambda_k(A) + \lambda_1(B).$$

As a direct consequence, we obtain the following bound on eigenvalue shifts under symmetric perturbations:

Lemma 3. *If $A,\ B \in \mathbb{R}^{n\times n}$ are symmetric. If λ_i are eigenvalues of A, $\hat{\lambda}_i$ are eigenvalues of $A+B$, then*

$$|\hat{\lambda}_k - \lambda_k| \leq \|B\|_2. \tag{4.1}$$

4.3 One Step Update

We divide the event stream into two categories: *node updates* (insertions) and *edge updates* (weight changes). Since we assume the graph has no isolated nodes, *node deletions* are not considered.

4.3.1 Node Update of Normalised Adjacency Matrix

We study a one-step augmentation of the normalised adjacency matrix $S_t = D_t^{-1/2} A_t D_t^{-1/2}$, where a new node is added at time $t+1$.

The new node connects to the existing graph via an incidence vector $b_t \in \mathbb{R}^n$, which is sparse: it has $m = |\mathrm{supp}(b_t)| \ll n$ nonzero entries. This results in a local change in both adjacency and degree matrices:

$$A_t \to A_{t+1} = \begin{pmatrix} A_t & b_t \\ b_t^\top & 0 \end{pmatrix}, \quad D_t \to D_{t+1} = \begin{pmatrix} D_t + \mathrm{diag}(b_t) & \\ & \mathbf{1}^\top b_t \end{pmatrix}. \tag{4.2}$$

Suppose we already have a rank-k orthogonal normalisation approximation of S_t, denoted by $\widehat{S}_t = U_t \Lambda_t U_t^\top$, with approximation error $\|S_t - \widehat{S}_t\|$. Our goal is to update the approximation without recomputing the full decomposition of S_{t+1}.

Inspired by [26], rather than directly calculated the decomposition, we express the updated basis U_{t+1} as a transformed version of U_t via rotation, scaling, and translation:

$$U_{t+1} = M^L U_t M^R + \Delta M$$

This formulation allows for a computationally cheap update. As for the updated diagonal matrix, we define a diagonal approximation induced by Lemma 1.

The resulting approximation error $\|S_{t+1} - \widehat{S}_{t+1}\|$ naturally depends on both the previous error $\|S_t - \widehat{S}_t\|$ and the update vector $\|b\|$.

In the theorem below, we derive an explicit update formula for U_{t+1} and provide an upper bound for the new approximation error.

Theorem 6 (One Step Node Update of NAM)**.** *Let G_t be a symmetric network without isolated vertices at time t, $A_t \in \mathbb{R}^{n\times n}$ as its adjacency matrix and $D_t = \mathrm{diag}(\sum_j A[i,j])$ as degree matrix. At time $t+1$, a new node is added with incident vector $b \in \mathbb{R}^n$ having $m \ll n$ nonzeros, so that*

$$A_{t+1} = \begin{pmatrix} A_t & b \\ b^\top & 0 \end{pmatrix}.$$

Define $S_t := D_t^{-1/2} A_t D_t^{-1/2}$, and let a k-term orthogonal diagonalisation approximation of S_t as

$$\widehat{S}_t^{(k)} = U_t \Lambda_t^{(k)} U_t^\top, \ U_t \in \mathbb{R}^{n\times k}, \ \Lambda_t^{(k)} \in \mathbb{R}^{k\times k}$$

with $U_t^\top U_t = I_k$ and $\Lambda_t^{(k)}$ diagonal.

Further let $\tilde{D}_t := D_t + \mathrm{diag}(b)$, $d = \mathbf{1}^\top b$, $W := \tilde{D}_t^{-1/2} D_t^{1/2}$, $\hat{b} := d^{-1/2} \tilde{D}_t^{-1/2} b$, $\hat{b} \notin \mathrm{span}(WU_t)$ and $\|W\|_2 \le \beta_{t,\mathrm{node}}$.

Then for $S_{t+1} := D_{t+1}^{-1/2} A_{t+1} D_{t+1}^{-1/2}$ there exists a k-term orthogonal diagonalisation approximation $\widehat{S}_{t+1}^{(k)} = U_{t+1} \Lambda_{t+1}^{(k)} U_{t+1}^\top$ and matrices $M_t^L, M_t^R, \Delta M_t$ such that:

(i) The orthogonal matrix $U_{t+1} \in \mathbb{R}^{(n+1)\times k}$ can be updated as

$$\boxed{U_{t+1} = M_t^L U_t M_t^R + \Delta M_t.} \tag{4.3}$$

(ii) The error between S_{t+1} and $\widehat{S}_{t+1}^{(k)}$ is upper bounded by

$$\boxed{\|S_{t+1} - \widehat{S}_{t+1}^{(k)}\|_F \;\le\; \beta_{t,\text{node}}^2 \|S_t - \widehat{S}_t^{(k)}\|_F + \sqrt{2}\|\hat{b}\|_2.} \tag{4.4}$$

Proof. Define

$$\text{Res} := S_t - \widehat{S}_t^{(k)},$$

and

$$\widehat{A} = \tilde{D}_t^{-1/2} A_t \tilde{D}_t^{-1/2}.$$

A block computation yields

$$S_{t+1} = D_{t+1}^{-1/2} A_{t+1} D_{t+1}^{-1/2} = \begin{pmatrix} \widehat{A} & \hat{b} \\ \hat{b}^\top & 0 \end{pmatrix}. \tag{4.5}$$

Write the rank-k approximation at time t as

$$S_t = U_t \Lambda_t U_t^\top + \text{Res},$$

we obtain

$$\widehat{A} = \tilde{D}_t^{-1/2} D_t^{1/2}\, S_t\, D_t^{1/2} \tilde{D}_t^{-1/2} = W U_t \Lambda_t U_t^\top W^\top \;+\; E, \quad E := W \text{Res} W^\top.$$

Let the thin QR of WU_t be $WU_t = \widehat{U}_t R_t$ with $\widehat{U}_t^\top \widehat{U}_t = I_k$ and $R_t \in \mathbb{R}^{k\times k}$ invertible. Setting $B := R_t \Lambda_t R_t^\top$, we have

$$\widehat{A} = \widehat{U}_t\, B\, \widehat{U}_t^\top + E.$$

According to Lemma 1, there exists matrices

$$C = \begin{pmatrix} R_t \Lambda_t R_t^\top & 0 & z \\ 0 & 0 & \alpha \\ z^\top & \alpha & 0 \end{pmatrix} \in \mathbb{R}^{(k+2)\times(k+2)} \tag{4.6}$$

and

$$K = \begin{pmatrix} \widehat{U}_t & \alpha^{-1}(I - \widehat{U}_t \widehat{U}_t^\top)\hat{b} & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(k+2)}, \tag{4.7}$$

where

$$z := R_t^{-\top} c, \qquad c := (WU_t)^\top \hat{b}, \qquad \alpha^2 := \|(I - \widehat{U}_t \widehat{U}_t^\top)\hat{b}\|_2 > 0,$$

such that

$$S_{t+1} = K\,C\,K^\top + \begin{pmatrix} E & 0 \\ 0 & 0 \end{pmatrix}.$$

Let $C = U_C\,\Gamma^+\,U_C^\top$ be the EVD with $\Gamma^+ = \mathrm{diag}(\gamma_1, \ldots, \gamma_{k+2})$ in nonincreasing order. We thus construct the top k-term approximation of S_{t+1}, $\widehat{S}_{t+1}^{(k)}$, by truncating top k diagonal values of Γ^+ as

$$\widehat{S}_{t+1}^{(k)} := K\,U_C[:, 1\!:\!k]\;\mathrm{diag}(\gamma_1, \ldots, \gamma_k)\;U_C[:, 1\!:\!k]^\top K^\top. \tag{4.8}$$

A direct calculation shows that

$$K = \begin{pmatrix} WU_tR_t^{-1} & \alpha^{-1}(\hat{b} - WU_t(R_t^\top R_t)^{-1}c) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4.9}$$

Accordingly, the rank-k approximation $\widehat{S}_{t+1}^{(k)}$ in Eqn.(4.8) admits an orthogonal diagonalisation of the form

$$\widehat{S}_{t+1}^{(k)} = U_{t+1}\mathrm{diag}(\gamma_1, \ldots, \gamma_k)U_{t+1}^\top,$$

where the column orthonormal matrix $U_{t+1} := KU_C[:, 1:k]$ can be expanded explicitly as

$$\begin{aligned} U_{t+1} &= \begin{pmatrix} WU_t(R_t^{-1}U_C[1:k, 1:k] - \alpha^{-1}(R_t^\top R_t)^{-1}cU_C[k+1, 1:k]) + \alpha^{-1}\hat{b}U_C[k+1, 1:k] \\ U_C[k+2, 1:k] \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} W \\ 0_{1\times n} \end{pmatrix}}_{:=M_t^L \in \mathbb{R}^{(n+1)\times n}} U_t \underbrace{\left(R_t^{-1}U_C[1:k, 1:k] - \alpha^{-1}(R_t^\top R_t)^{-1}cU_C[k+1, 1:k]\right)}_{:=M_t^R \in \mathbb{R}^{k\times k}} \\ &\qquad + \underbrace{\begin{pmatrix} \alpha^{-1}\hat{b}U_C[k+1, 1:k] \\ U_C[k+2, 1:k] \end{pmatrix}}_{:=\Delta M_t \in \mathbb{R}^{(n+1)\times k}}. \end{aligned} \tag{4.10}$$

The associated column orthogonal matrix update is then of the affine form

$$U_{t+1} = M_t^L U_t M_t^R + \Delta M_t.$$

It remains to prove the error bound. Subtracting (4.8) from (4.5) and using the triangle inequality gives

$$\|S_{t+1} - \widehat{S}_{t+1}^{(k)}\|_F \;\le\; \underbrace{\|E\|_F}_{\text{(I)}} \;+\; \underbrace{\left\|K\,U_C\,\mathrm{diag}(0, \ldots, 0, \gamma_{k+1}, \gamma_{k+2})\,U_C^\top K^\top\right\|_F}_{\text{(II)}}. \tag{4.11}$$

For (I), referring the definition of E shows

$$\|E\|_F = \|W \text{ Res } W^\top\|_F \le \beta_{t,\text{node}}^2 \| \underbrace{\text{Res}}_{=S_t - \widehat{S}_t^{(k)}} \|_F, \quad (\|W\|_2 \le \beta_{t,\text{node}}). \tag{4.12}$$

For (II), since K and U_C have orthonormal columns,

$$\text{(II)} \ \le \ \left(\gamma_{k+1}^2 + \gamma_{k+2}^2\right)^{1/2}. \tag{4.13}$$

To seek for the range of γ_{k+1} and γ_{k+2}, let us rewrite C in Eqn.(4.6) as

$$C = \text{diag}(R_t \Lambda_t R_t^\top, 0, 0) + U,$$

where

$$U = \begin{pmatrix} 0 & 0 & z \\ 0 & 0 & \alpha \\ z^\top & \alpha & 0 \end{pmatrix}.$$

Direct calculation shows nonzero eigenvalues of U are $\pm\sqrt{\|z\|_2^2 + \alpha^2}$, hence $\|U\|_2 = \sqrt{\|z\|_2^2 + \alpha^2} = \|\hat{b}\|_2$. By Lemma 3 applied to $C = \text{diag}(B, 0, 0) + U$, the two tail eigenvalues obey

$$|\gamma_{k+1}|, \ |\gamma_{k+2}| \ \le \ \|U\|_2 \ = \ \|\hat{b}\|_2. \tag{4.14}$$

Combining (4.13)–(4.14) yields

$$\text{(II)} \ \le \ \sqrt{2}\|\hat{b}\|_2. \tag{4.15}$$

Putting (4.12) and (4.15) into (4.11) gives the Frobenius bound

$$\|S_{t+1} - \widehat{S}_{t+1}^{(k)}\|_F \ \le \ \beta_{t,\text{node}}^2 \|S_t - \widehat{S}_t^{(k)}\|_F + \sqrt{2}\,\|\hat{b}\|_2, \tag{4.16}$$

which is the claimed bound. This completes the proof. □

In our analysis, we assume that $\hat{b} \notin \text{span}(WU_t)$, thereby avoiding the degenerate case $\alpha = 0$. This assumption is reasonable in practice: since $n \gg k$, the span of WU_t forms only a low-dimensional subspace in a very high-dimensional space, making it highly unlikely for $\hat{b}$ to lie exactly within it (Fig. 4.1).

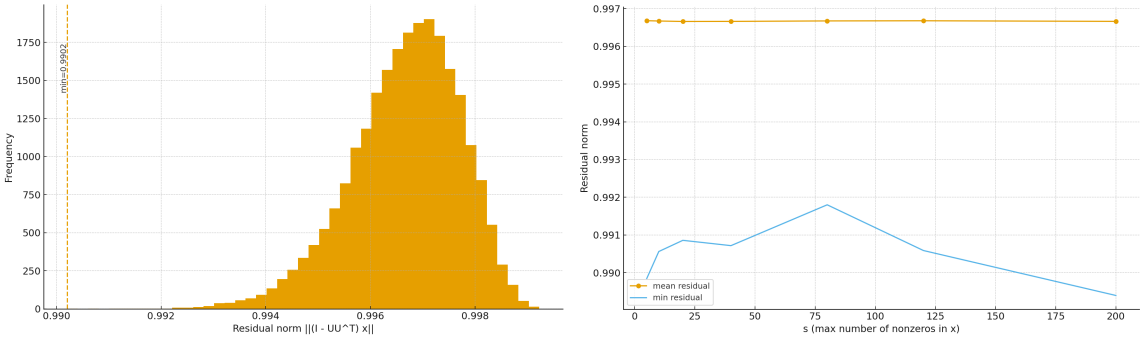

Figure 4.1: Left: histogram of residual norm $\|(I - UU^\top)x\|$ for sparse unit vectors x, with random orthonormal. Dashed line indicates the minimum residual. Right: mean and minimum residuals versus s. Residuals stay well above zero and are insensitive to s, suggesting that sparse vectors rarely fall in span(WU) when $k \ll n$.

For completeness, when $\alpha = 0$, we directly use the case $r = 0$ from Lemma 1: Let

$$K = \begin{pmatrix} WU_t R_t^{-1} & 0 \\ 0 & 1 \end{pmatrix}, \; C = \begin{pmatrix} R_t \Lambda_t R_t^\top & z \\ z^\top & 0 \end{pmatrix} \tag{4.17}$$

and substitute them into Eqn.(4.7) and Eqn.(4.6) respectively will give the updating scheme. In this case, the upper bound can be tighter, but still upper bounded by Eqn.(4.4).

Remark (Practical computation and complexity)**.** *To improve efficiency in the node update, we compute key quantities using a series of tailored tricks. Below we list the main terms and their computational costs, assuming the support size* $m = |\mathrm{supp}(b)| \ll n$ *and the cost of accessing each* u_i *is* $O(T(u_i))$*:*

- *Diagonal matrix* W*:*

$$W[i,i] = \sqrt{\frac{D_t[i,i]}{D_t[i,i] + b[i]}} \qquad \textit{Cost: } O(m). \tag{4.18}$$

- *Projection vector:*

$$c = d^{-1/2} \sum_{i \in \mathrm{supp}(b)} \frac{\sqrt{D_t[i,i]}b[i]}{D_t[i,i] + b[i]} u_i^\top \qquad \textit{Cost: } O(mT(u_i)) + O(mk). \tag{4.19}$$

- *Normalised vector and its norm:*

$$\hat{b}[i] = d^{-1/2} . \frac{b[i]}{\sqrt{D_t[i,i] + b[i]}}, \qquad \|\hat{b}\|_2^2 = d^{-1} \sum_{i \in \mathrm{supp}(b)} \frac{b[i]^2}{D_t[i,i] + b[i]} \qquad \textit{Cost: } O(m). \tag{4.20}$$

- *Gram matrix for R_t:*

$$G := U_t^\top W^2 U_t = I - \sum_{i \in \mathrm{supp}(b)} \frac{b[i]}{D_t[i,i] + b[i]} u_i^\top u_i \qquad \textit{Cost: } O(mT(u_i)) + O(mk^2), \tag{4.21}$$

followed by a Cholesky decomposition to obtain R_t, which costs $O(k^3)$.

The remaining components can be computed as follows:

- *Construction of C and its k-term EVD: $O(k^3)$.*
- *M_t^L from updated W: $O(m)$.*
- *M_t^R from R_t^{-1} and linear correction: $O(k^3)$.*
- *ΔM_t (nonzero only in $m+1$ rows) : $O(mk)$.*

In total, the full node update to get M_t^L, M_t^R and ΔM_t from scratch costs

$$O(k^3 + mk^2 + mT(u_i)).$$

As long as $T(u_i)$ remains moderate, this is significantly cheaper than recomputing the rank-k eigendecomposition of S_{t+1}.

4.3.2 Edge Update of Normalised Adjacency Matrix

Edge updates are very similar to node updates, differing only slightly in their computation. We present the formula for the one-step update and its error in the theorem below.

Theorem 7 (One Step Edge Update of NAM)**.** *Let G_t be a symmetric network without isolated vertices at time t, $A_t \in \mathbb{R}^{n\times n}$ its adjacency, and $D_t = \mathrm{diag}(\sum_j A[i,j])$. At time $t{+}1$, a symmetric rank-2 edge update occurs:*

$$A_{t+1} = A_t + xy^\top + yx^\top \;=\; A_t + XSX^\top, \qquad X = \begin{pmatrix} x & y \end{pmatrix}, \quad S = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

where $x, y \in \mathbb{R}^n$ have at most $m \ll n$ nonzeros.

Define normalised adjacency $S_t := D_t^{-1/2} A_t D_t^{-1/2}$, and let a k-term orthogonal diagonalisation approximation of S_t be

$$\widehat{S}_t^{(k)} = U_t \Lambda_t^{(k)} U_t^\top,\ U_t \in \mathbb{R}^{n\times k},\ \Lambda_t^{(k)} \in \mathbb{R}^{k\times k}$$

with $U_t^\top U_t = I_k$ *and* $\Lambda_t^{(k)}$ *diagonal. Let*

$$D_{t+1} = D_t + \operatorname{diag}\big((\mathbf{1}^\top y)\, x + (\mathbf{1}^\top x)\, y\big), \quad W := D_{t+1}^{-1/2} D_t^{1/2}, \quad \widehat{X} := D_{t+1}^{-1/2} X = (\hat{x}\ \hat{y}).$$

Form the thin QR $WU_t = \widehat{U}_t R_t$ *with* $\widehat{U}_t^\top \widehat{U}_t = I_k$*, and another QR*

$$(I - \widehat{U}_t \widehat{U}_t^\top)\widehat{X} \;=\; Q_X R_X, \qquad Q_X^\top Q_X = I_r.$$

We further assume that $\operatorname{rank}\big((I - \widehat{U}_t \widehat{U}_t^\top)\widehat{X}\big) = 2$ *and* $\|W\|_2 \le \beta_{t,\mathrm{edge}}$*.*
Then for $S_{t+1} := D_{t+1}^{-1/2} A_{t+1} D_{t+1}^{-1/2} = W S_t W \;+\; \widehat{X} S \widehat{X}^\top$ *there exists a* k*-term orthogonal diagonalisation approximation* $\widehat{S}_{t+1}^{(k)} = U_{t+1}\Lambda_{t+1}U_{t+1}^\top$ *and matrices* $M_t^L, M_t^R, \Delta M_t$ *such that:*

(i) The orthogonal matrix $U_{t+1} \in \mathbb{R}^{n\times k}$ *can be updated as*

$$\boxed{U_{t+1} \;=\; M_t^L\, U_t\, M_t^R \;+\; \Delta M_t.} \tag{4.22}$$

(ii) The error between S_{t+1} *and* $\widehat{S}_{t+1}^{(k)}$ *is upper bounded by*

$$\boxed{\|S_{t+1} - \widehat{S}_{t+1}^{(k)}\|_F \;\le\; \beta_{t,\mathrm{edge}}^2 \|S_t - \widehat{S}_t^{(k)}\|_F \;+\; \sqrt{2}\,\|\widehat{X}\|_2^2.} \tag{4.23}$$

Proof. As in the node case, we firstly define

$$\mathrm{Res} := S_t - S_t^{(k)}.$$

Write

$$S_{t+1} = D_{t+1}^{-1/2} A_{t+1} D_{t+1}^{-1/2} = \underbrace{W\, S_t\, W^\top}_{W U_t \Lambda_t U_t^\top W^\top + W \mathrm{Res} W^\top} + \widehat{X} S \widehat{X}^\top. \tag{4.24}$$

With $WU_t = \widehat{U}_t R_t$ and $(I - \widehat{U}_t\widehat{U}_t^\top)\widehat{X} = \widehat{Q}_X \widehat{R}_X$, based on Lemma 2, define $K = (\widehat{U}_t\ \widehat{Q}_X)$ and

$$C = \begin{pmatrix} R_t \Lambda_t R_t^\top & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ \widehat{R}_X \end{pmatrix} S \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ \widehat{R}_X \end{pmatrix}^\top \tag{4.25}$$

then we have

$$S_{t+1} = K C K^\top \;+\; E, \qquad E := W\ \mathrm{Res}\ W^\top. \tag{4.26}$$

Let $C = U_C \Gamma^+ U_C^\top$ with $\Gamma^+ = \operatorname{diag}(\gamma_1, \ldots, \gamma_{k+2})$ in nonincreasing order, and the k-term truncation

$$\widehat{S}_{t+1}^{(k)} := K\, U_C[:, 1\!:\!k]\, \operatorname{diag}(\gamma_1, \ldots, \gamma_k)\, U_C[:, 1\!:\!k]^\top K^\top.$$

As $\operatorname{rank}(R_X) = 2$, we have

$$\begin{aligned} U_{t+1} = KU_C &= (WU_tR_t^{-1}, \widehat{X}R_X^{-1} - WU_tR_t^{-1}\widehat{U}_t^\top \widehat{X}R_X^{-1}) \begin{pmatrix} U_C[1:k, 1:k] \\ U_C[k+1:, 1:k] \end{pmatrix} \\ &= \underbrace{W}_{:=M_t^L} U_t \underbrace{R_t^{-1}(U_C[1:k,1:k] - R_t^{-\top} c R_X^{-1} U_C[k+1:,1:k])}_{:=M_t^R} \\ &\qquad + \underbrace{\widehat{X}R_X^{-1}U_C[k+1:,1:k]}_{:=\Delta M_t}, \end{aligned} \tag{4.27}$$

where $c := U_t^\top W \widehat{X}$.

Then for the error, as in the node-update proof,

$$\|S_{t+1} - \widehat{S}_{t+1}^{(k)}\|_F \;\le\; \underbrace{\|E\|_F}_{\text{(I)}} \;+\; \underbrace{\left(\gamma_{k+1}^2 + \gamma_{k+2}^2\right)^{1/2}}_{\text{(II)}}, \tag{4.28}$$

where $E = W$ Res W and thus

$$\text{(I)} \le \beta_{t,\text{edge}}^2 \|\text{Res}\|_F.$$

Write $C = \operatorname{diag}(R_t\Lambda_t R_t^\top, 0, 0) + U$ and

$$U := \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix} S \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix}^\top.$$

By Lemma 3, $|\gamma_{k+i}| \le \|U\|_2$ for $i = 1, 2$, hence

$$\text{(II)} \;\le\; \sqrt{2}\,\|U\|_2 \;\le\; \sqrt{2}\left\| \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix} \right\|_2 \|S\|_2 \left\| \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix}^\top \right\|_2 \le \sqrt{2}\left\| \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix} \right\|_2^2 = \sqrt{2}\,\|\widehat{X}\|_2^2.$$

The last equality uses

$$\left\| \begin{pmatrix} \widehat{U}_t^\top \widehat{X} \\ R_X \end{pmatrix} \right\|_2^2 = \lambda_{\max}\big(\widehat{X}^\top(\widehat{U}_t\widehat{U}_t^\top + I - \widehat{U}_t\widehat{U}_t^\top)\widehat{X}\big) = \lambda_{\max}(\widehat{X}^\top\widehat{X}) = \|\widehat{X}\|_2^2.$$

This gives (ii). □

Based on reasoning similar to the node update case, the vectors $\hat{x}, \hat{y}$ are highly unlikely to lie in the column space of WU_t when $k \ll n$. Hence we abolish cases rank is less than 2.

Remark (Practical computation and complexity for edge update)**.** *To simplify the edge update process and estimate complexity, we outline the main intermediate terms and their costs below (with $i \in \operatorname{supp}(x) \cup \operatorname{supp}(y)$):*

- *Updated degree matrix:*

$$D_{t+1} = D_t + \mathrm{diag}(s_y x + s_x y), \qquad s_x = \sum_i x_i, \quad s_y = \sum_i y_i \qquad \textit{Cost: } O(m). \tag{4.29}$$

- *Normalised update vectors:*

$$\hat{x}_i = \frac{x_i}{\sqrt{D_{t+1}[i,i]}}, \qquad \hat{y}_i = \frac{y_i}{\sqrt{D_{t+1}[i,i]}}, \qquad \widehat{X} = (\hat{x}\ \hat{y}) \qquad \textit{Cost: } O(m). \tag{4.30}$$

- *Diagonal matrix W:*

$$W[i,i] = \sqrt{D_t[i,i]/D_{t+1}[i,i]} \qquad \textit{Cost: } O(m). \tag{4.31}$$

- *Projection matrix:*

$$c = \begin{pmatrix} \sum_{i\in\mathrm{supp}(x)} W_{ii}\hat{x}_i u_i^\top & \sum_{i\in\mathrm{supp}(y)} W_{ii}\hat{y}_i u_i^\top \end{pmatrix} \qquad \textit{Cost: } O(mT(u_i)) + O(mk). \tag{4.32}$$

- *Gram matrix for R_t:*

$$G := U_t^\top W^2 U_t = I - \sum_{i\in\mathrm{supp}(x)\cup\mathrm{supp}(y)} \frac{s_y x_i + s_x y_i}{D_t[i,i] + s_y x_i + s_x y_i} u_i^\top u_i = R_t^\top R_t, \tag{4.33}$$

  Cost: $O(mT(u_i)) + O(mk^2) + O(k^3)$ *(for Cholesky).*

- *Gram matrix for R_X:*

$$H := \widehat{X}^\top \widehat{X} - c^\top G^{-1} c = R_X^\top R_X, \qquad \textit{Cost: } O(m) + O(k^2). \tag{4.34}$$

Overall, the additional cost compared to node update lies in the construction of the 2×2 matrix H, which is negligible. Therefore, the total time complexity for computing M_t^L, M_t^R, and ΔM_t remains

$$O(k^3 + mk^2 + mT(u_i)).$$

This confirms that edge updates can be executed as efficiently as node updates in practice.

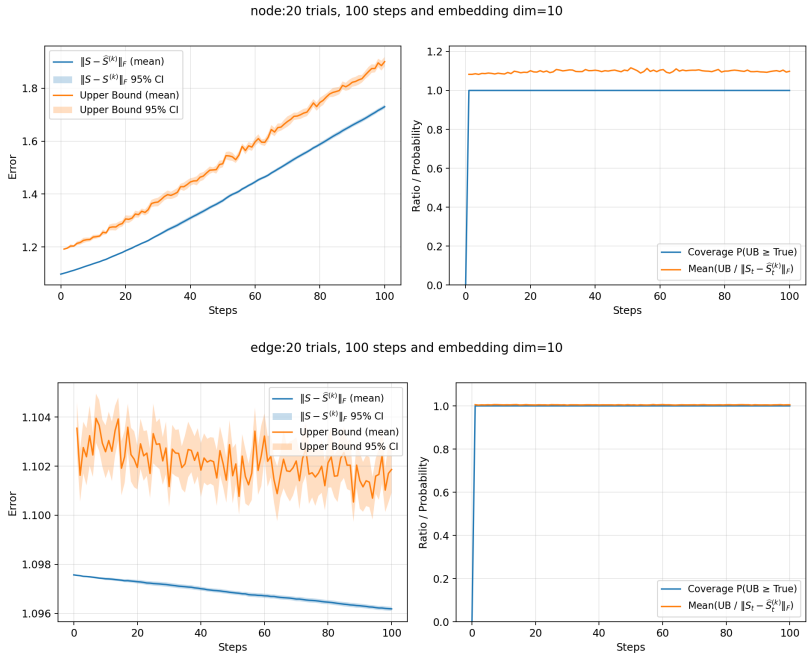

Figure 4.2: A comparison of real S error and bounds in one step update. Left panels show mean $\pm 95\%$ CI of $\|S_t - \widehat{S}_t^{(k)}\|_F$ with mean upper bound. Right: probability of real error is bounded and mean upper bound/real error. $(k = 10)$

Fig. 4.2 shows the node/edge upper error bound and real error in randomly generated temporal networks. It is obvious that our upper bound is valid and rather tight.

4.3.3 One-Step Update Error for Pre-log NetMF Matrix

It should be noted that the orthogonal-diagonalisation approximation above is not equivalent to a rank-k truncation of the true matrix; consequently, the spectrum of the approximate NAM may exceed the eigenvalue interval $[-1, 1]$ of the true NAM. We now provide bounds on the spectrum of the approximate NAM produced by the update scheme.

Lemma 4. *In both node update and edge update, the absolute value of eigenvalues of* $\widehat{S}_{t+1}^{(k)}$ *are bounded in*

$$[-1 - \|E\|_2, 1 + \|E\|_2],$$

where $E = W$ Res W *as defined in proofs of Theorem 6 and 7.*

Proof. Denote $\widehat{S}^{(k+2)} := KCK^\top$ for K, C defined as in proofs of Theorem 6 and 7. According to Corollary 1, $\|S_{t+1}\|_2 \leq 1$. Then by Weyl inequality,

$$\lambda_{\max}(\widehat{S}_{t+1}^{(k+2)}) = \lambda_{\max}(S_{t+1} - E) \leq \lambda_{\max}(S_{t+1}) + \|E\|_2 \leq 1 + \|E\|_2.$$

Similarly we have

$$\lambda_{\min}(\widehat{S}_{t+1}^{(k+2)}) \geq -1 - \|E\|_2.$$

As eigenvalues of $\widehat{S}_{t+1}^{(k)}$ is a subset of eigenvalues of $\widehat{S}_{t+1}^{(k+2)}$, we thus conclude the expected result. □

This lemma shows that the spectrum of the updated low-rank approximation $\widehat{S}_{t+1}^{(k)}$remains confined to a symmetric, bounded interval, expanding by at most $\|E\|$ beyond $[-1, 1]$.

Building on this result, we present a one-step error bound as

Theorem 8 (One Step Error)**.** *For both node update and edge update, the pre-logged NetMF error is upper bounded as*

$$\|M_{t+1} - \widehat{M}_{t+1}\|_F \leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \left(\beta_t^2 \|S_t - \widehat{S}_t^{(k)}\|_F + B_t\right) \sum_{r=1}^{T} r w_r (1 + \beta_t^2 \|S_t - \widehat{S}_t^{(k)}\|_F)^{r-1}, \tag{4.35}$$

where $d_{\min,t+1}$ is the minimum degree of all nodes at time $t+1$, β_t is the upper bound of $\|W\|_2$ and

$$B_t = \begin{cases} \sqrt{2}\|\hat{b}\|_2, & \text{node update,} \\ \sqrt{2}\|\widehat{X}\|_2^2, & \text{edge update.} \end{cases} \tag{4.36}$$

Proof. Directly write the error in Frobenius norm as

$$
\begin{aligned}
\|M_{t+1} - \widehat{M}_{t+1}\|_F &= \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}}} \left\| D_{t+1}^{-1/2} \left(\sum_{r=1}^{T} w_r (S_{t+1}^r - \widehat{S}_{t+1}^{(k)r}) \right) D_{t+1}^{-1/2} \right\|_F \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \left\| \sum_{r=1}^{T} w_r (S_{t+1}^r - \widehat{S}_{t+1}^{(k)r}) \right\|_F \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \sum_{r=1}^{T} w_r \|S_{t+1}^r - \widehat{S}_{t+1}^{(k)r}\|_F \text{ (Triangle)} \\
&= \frac{\mathrm{vol}(G_{t+1})}{b d_{\min,t+1}} \sum_{r=1}^{T} w_r \left\| \sum_{i=0}^{r-1} S_{t+1}^{r-1-i} (S_{t+1} - \widehat{S}_{t+1}^{(k)}) \widehat{S}_{t+1}^{(k)i} \right\|_F \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \sum_{r=1}^{T} w_r \sum_{i=0}^{r-1} \left\| S_{t+1} - \widehat{S}_{t+1}^{(k)} \right\|_F \|\widehat{S}_{t+1}^{(k)}\|_2^i \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \left\| S_{t+1} - \widehat{S}_{t+1}^{(k)} \right\|_F \sum_{r=1}^{T} r w_r \|\widehat{S}_{t+1}^{(k)}\|_2^{r-1} \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \left\| S_{t+1} - \widehat{S}_{t+1}^{(k)} \right\|_F \sum_{r=1}^{T} r w_r (1 + \beta_t^2 \|S_t - \widehat{S}_t^{(k)}\|_2)^{r-1} \text{ (By Lemma 4)} \\
&\leq \frac{\mathrm{vol}(G_{t+1})}{b_{\mathrm{neg}} d_{\min,t+1}} \left(\beta_t^2 \|S_t - \widehat{S}_t^{(k)}\|_F + B_t \right) \sum_{r=1}^{T} r w_r (1 + \beta_t^2 \|S_t - \widehat{S}_t^{(k)}\|_F)^{r-1}.
\end{aligned}
$$

The first inequality uses $\|AB\|_F \leq \|A\|_2 \|B\|_F$, $\|D_{t+1}^{-1/2}\|_2 = d_{\min,t+1}^{-1/2}$, and the last inequality uses Eqn.(4.4), Eqn.(4.23) and $\|A\|_2 \leq \|A\|_F$. □

The left figure in Fig. 4.3 compares the one-step update error and upper bound provided in the last theorem. The empirical error (blue) is always below the bound (orange), confirming validity.

4.4 Multi-Step Update of Pre-log NetMF Matrix Approximation

With all the above preparations, we are now ready to extend our update rule and error analysis to multiple steps of pre-logged NetMF matrix approximation. Theorem 9 provides an explicit expression for pre-logged NetMF matrix approximation at time $t + s$, as well as its associated error bound.

Theorem 9. *Starting at time t, if we use the update scheme described in Theorem 6 and 7, then at time $t + s$*

(i) the pre-logged NetMF matrix approximation $\widehat{M}_{t+s}$ can be expressed as

$$\widehat{M}_{t+s} = \frac{\mathrm{vol}(G_{t+s})}{b_{\mathrm{neg}}} D_{t+s}^{-1/2} U_{t+s}^{(k)} f(\Lambda_{t+s}^{(k)}) U_{t+s}^{(k)\top} D_{t+s}^{-1/2}, \tag{4.37}$$

where the orthogonal matrix $U_{t+s}^{(k)}$ is given by

$$U_{t+s}^{(k)} = M_{t:t+s}^{L} U_t^{(k)} M_{t:t+s}^{R} + \Delta M_{t:t+s}, \tag{4.38}$$

$$M_{t:t+s}^{L} = \prod_{i=1}^{s} M_{t+s-i}^{L}, \tag{4.39}$$

$$M_{t:t+s}^{R} = \prod_{i=1}^{s} M_{t+i-1}^{R}, \tag{4.40}$$

$$\Delta M_{t:t+s} = \sum_{j=0}^{s-1} \left(\prod_{i=1}^{s-j-1} M_{t+s-i}^{L} \right) \Delta M_{t+j} \left(\prod_{i=1}^{s-j-1} M_{t+j+i}^{R} \right). \tag{4.41}$$

If the upper limit of the product notation is 0 we set the product to be 1;

(ii) the error at $t+s$ is upper bounded by

$$\| \log^{\circ} M_{t+s} - \log^{\circ} \widehat{M}_{t+s} \|_F \leq \frac{\mathrm{vol}(G_{t+s})}{b_{\mathrm{neg}} d_{\min,t+s}} \left(\prod_{i=0}^{s-1} \beta_{t+i}^2 E_t + \sum_{i=0}^{s-1} \prod_{j=i+1}^{s-1} \beta_{t+j}^2 B_{t+i} \right) \cdot \sum_{r=1}^{T} r w_r \left(1 + \prod_{i=0}^{s-1} \beta_{t+i}^2 E_t + \sum_{i=0}^{s-2} \prod_{j=i+1}^{s-1} \beta_{t+j}^2 B_{t+i} \right)^{r-1}, \tag{4.42}$$

where

$$E_t = \| S_t - \widehat{S}_t^{(k)} \|_F. \tag{4.43}$$

Proof. (i) We prove the first claim by induction. If $s = 1$, then we have

$$U_{t+1}^{(k)} = M_t^L U_t^{(k)} M_t^R + \Delta M_t, \tag{4.44}$$

which is the result from Theorem 6 and 7.

Assume the claim is true at $t+s$, then by Theorem 6 and 7,

$$\begin{aligned}
U_{t+s+1}^{(k)} &= M_{t+s}^{L} U_{t+s}^{(k)} M_{t+s}^{R} + \Delta M_{t+s} \\
&= M_{t+s}^{L} \prod_{i=1}^{s} M_{t+s-i}^{L} U_{t}^{(k)} \prod_{i=1}^{s} M_{t+i-1}^{R} M_{t+s}^{R} \\
&\quad + \Delta M_{t+s} + M_{t+s}^{L} \sum_{j=0}^{s-1} \left(\prod_{i=1}^{s-j-1} M_{t+s-i}^{L} \right) \Delta M_{t+j} \left(\prod_{i=1}^{s-j-1} M_{t+j+i}^{R} \right) M_{t+s}^{R} \\
&= \prod_{i=1}^{s+1} M_{t+s+1-i}^{L} U_{t}^{(k)} \prod_{i=1}^{s+1} M_{t+i-1}^{R} \\
&\quad + \sum_{j=0}^{s} \left(\prod_{i=1}^{s-j} M_{t+s+1-i}^{L} \right) \Delta M_{t+j} \left(\prod_{i=1}^{s-j} M_{t+j+i}^{R} \right),
\end{aligned} \tag{4.45}$$

and this is exactly (4.23) - (4.41) at $s+1$.

(ii) According to Theorem 6 and 7, by induction, $\forall j \geq 1$,

$$E_{t+s} \leq \prod_{i=0}^{s-1} \beta_{t+i}^{2} E_t + \sum_{i=0}^{s-1} \prod_{j=i+1}^{s-1} \beta_{t+j}^{2} B_{t+i}. \tag{4.46}$$

Substitute back into Theorem 8, we have

$$\begin{aligned}
\|M_{t+s} - \widehat{M}_{t+s}\|_F &\leq \frac{\mathrm{vol}(G_{t+s})}{b_{\mathrm{neg}} d_{\min,t+s}} \left(\beta_{t+s-1}^{2} E_{t+s-1} + B_{t+s-1}\right) \sum_{r=1}^{T} r w_r (1 + \beta_{t+s-1}^{2} E_{t+s-1})^{r-1} \\
&\leq \frac{\mathrm{vol}(G_{t+s})}{b_{\mathrm{neg}} d_{\min,t+s}} \left(\prod_{i=0}^{s-1} \beta_{t+i}^{2} E_t + \sum_{i=0}^{s-1} \prod_{j=i+1}^{s-1} \beta_{t+j}^{2} B_{t+i} \right) \\
&\qquad \cdot \sum_{r=1}^{T} r w_r \left(1 + \prod_{i=0}^{s-1} \beta_{t+i}^{2} E_t + \sum_{i=0}^{s-2} \prod_{j=i+1}^{s-1} \beta_{t+j}^{2} B_{t+i} \right)^{r-1}.
\end{aligned}$$

Hence by Theorem 3, the error between logged matrices satisfies

$$\| \log^{\circ} M_{t+s} - \log^{\circ} \widehat{M}_{t+s} \|_F \leq \|M_{t+s} - \widehat{M}_{t+s}\|_F. \tag{4.47}$$

Thus the claim is proved.

□

The bound in Eqn.(4.42) consists of three interpretable factors:

- The first term $\frac{\mathrm{vol}(G_{t+s})}{b_{\mathrm{neg}} d_{\min,t+s}}$ captures the current global graph information, and is significantly affected by $d_{\min,t+s}$, the weakest connectivity in the network;

- The second term $\prod_{i=0}^{s-1} \beta_{t+i}^2 E_t + \sum_{i=0}^{s-1} \prod_{j=i+1}^{s-1} \beta_{t+j}^2 B_{t+i}$ grows linearly with the number of steps s and reflects both the initial approximation error and accumulated update noise, which is controllable.
- The third term $\sum_{r=1}^{T} rw_r \left(1 + \prod_{i=0}^{s-1} \beta_{t+i}^2 E_t + \sum_{i=0}^{s-2} \prod_{j=i+1}^{s-1} \beta_{t+j}^2 B_{t+i}\right)^{r-1}$ reflects how the structure of the filter amplifies the error, especially when long paths are heavily weighted. And this is the only term can be adjusted by the user.

To control this last factor, one should prefer filters that emphasis on short-range paths (i.e., with decaying weights) to suppress spectral noise amplification. However, in settings where capturing long-range dependencies is essential (e.g., in randomly connected graphs), filters with an emphasis on long-range paths may be preferred. This creates a trade-off between representation power and numerical stability.

The product $\prod_i \beta_i^2$ can exceed 1 only when weight decreases occur. Thus, limiting the number of such events during each work loop helps control error propagation. The right panel of Fig. 4.3 shows that the derived upper bound is effective, yet somewhat loose. This looseness mainly arises from the amplification of $\frac{\text{vol}(G)}{d_{\min}}$ (Table.B.1).

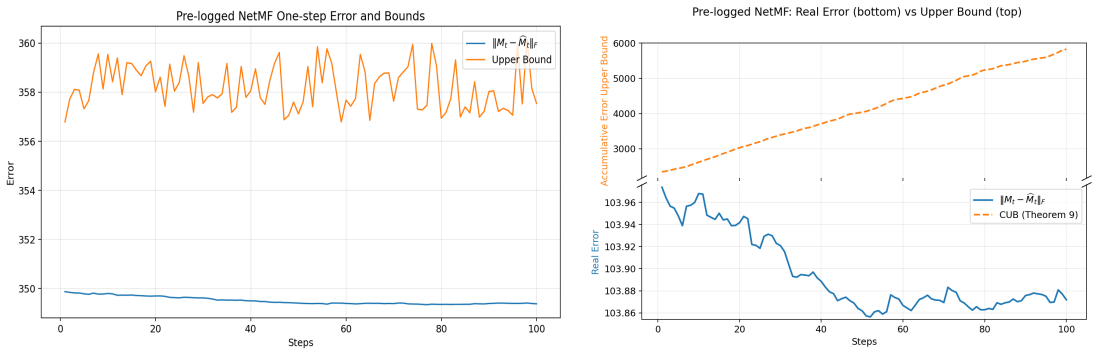

Figure 4.3: A comparison of real pre-logged NetMF error and bounds. Left: real one-step error $\|M_t - \widehat{M}\|_F$ vs per-step upper bound. Right: real multi-step error $\|M_{t+s} - \widehat{M}_{t+s}\|_F$ and cumulative upper bound given by Theorem 9 in a randomly generated temporal network. ($T = 3,\ k = 10$)

Chapter 5

An Accelerated Temporal Embedding Method based on NetMF (ATNMF)

5.1 General Description

The key idea behind our algorithm is to separate the expensive whole matrix decomposition step from the frequent arrival of temporal events in a dynamic network. Instead of recomputing embeddings from scratch at each update, we maintain a compact approximation of the normalised adjacency matrix online, and defer the actual embedding generation to an offline stage that is triggered on demand and can be computed efficiently.

Specifically, starting from the initial normalised adjacency matrix

$$S_0 \;=\; D_0^{-1/2} A_0 D_0^{-1/2}$$

we compute and store its top-k eigenpairs (U_0, Λ_0). Then

- **Online Stage:**
  During the time interval $[0, t]$, we maintain four smaller or sparse matrices $M^L,\ M^R,\ \Delta M$ and the diagonal matrix Λ_t along with a scalar $\mathrm{vol}(G_t)$ and an index set $\mathrm{Ind}_{\Delta M}$ indicating nonzero row indices in ΔM, such that at time t

$$S_t \approx U_t \Lambda_t U_t^\top, \quad U_t \;\approx\; M^L\, U_0\, M^R \;+\; \Delta M,$$

- **Offline Stage:**
  When triggered (either by exceeding a fixed number of updates or by external query), the algorithm uses maintained variables to construct an approximation

of the NetMF matrix, and then perform a low-rank decomposition to produce node embeddings.

This design accelerates NetMF on temporal networks; we therefore refer to it as an Accelerated Temporal embedding method based on NetMF (ATNMF).

The entire workflow is illustrated in Fig. 5.1. For the reader's convenience, Table.C.1 summarises the dimensions of all matrices used in the ATNMF algorithm.

5.2 Online Update

The goal of the online update stage is to efficiently maintain an approximate orthogonal diagonalisation of the evolving normalised adjacency matrix S_t under a stream of events.

To realise this goal, we provide three algorithms in this section. `NodeUpdate` and `EdgeUpdate` implement the incremental update schemes derived from Theorem 6 and Theorem 7, respectively, each handling one type of event. `OneStepUpdateCompose` composes the updates accumulated across multiple events into a unified form, preparing for next event and downstream embedding computation in the offline stage.

Algorithm 3: NodeUpdate

Input: Initial orthonormal matrix U_0, $\mathrm{vol}(G)$, $d_{\min}$, M^L, M^R, ΔM, Λ_t, D_t, added node b
Output: $M_t^L, M_t^R, \Delta M_t, \Lambda_{t+1}$, $\mathrm{vol}(G)$, nonzero indices in this update Ind

Initialise:
  $W \leftarrow I_n$; $G \leftarrow I_k$; $c \leftarrow 0_k$; $\|\hat{b}\|_2^2 \leftarrow 0$
$\mathrm{Ind} \leftarrow \mathrm{supp}(b)$; $d \leftarrow \mathbf{1}^\top b$
$\mathrm{vol}(G) \leftarrow \mathrm{vol}(G_t) + 2d$
for $i \in \mathrm{Ind}$ **do**
  Construct W by Eqn.(4.18); Update G by Eqn.(4.21)
  Construct c by Eqn.(4.19); Construct $\hat{b}$ and $\|\hat{b}\|_2^2$ by Eqn.(4.20)
Cholesky $G = R^\top R$; calculate $z \leftarrow R^{-\top} c$
$\alpha^2 \leftarrow \|\hat{b}\|_2^2 - \|z\|_2^2$
$C \leftarrow \begin{pmatrix} R\Lambda_t R^\top & 0 & z \\ 0 & 0 & \alpha \\ z^\top & \alpha & 0 \end{pmatrix}$
$(U_C, \Gamma^+) \leftarrow \mathrm{EVD}(C, k)$; $\Lambda_{t+1} \leftarrow \Gamma^+[1\!:\!k, 1\!:\!k]$; $M_t^L \leftarrow \begin{pmatrix} W \\ 0 \end{pmatrix}$
$M_t^R \leftarrow R^{-1} U_C[1\!:\!k, 1\!:\!k] - \alpha^{-1}(R^\top R)^{-1} c U_C[k+1, 1:k]$
$\Delta M_t \leftarrow \begin{pmatrix} \alpha^{-1}\hat{b} U_C[k+1, 1:k] \\ U_C[k+2, 1:k] \end{pmatrix}$;
return $M_t^L, M_t^R, \Delta M_t, \Lambda_{t+1}, \mathrm{vol}(G), \mathrm{Ind}$

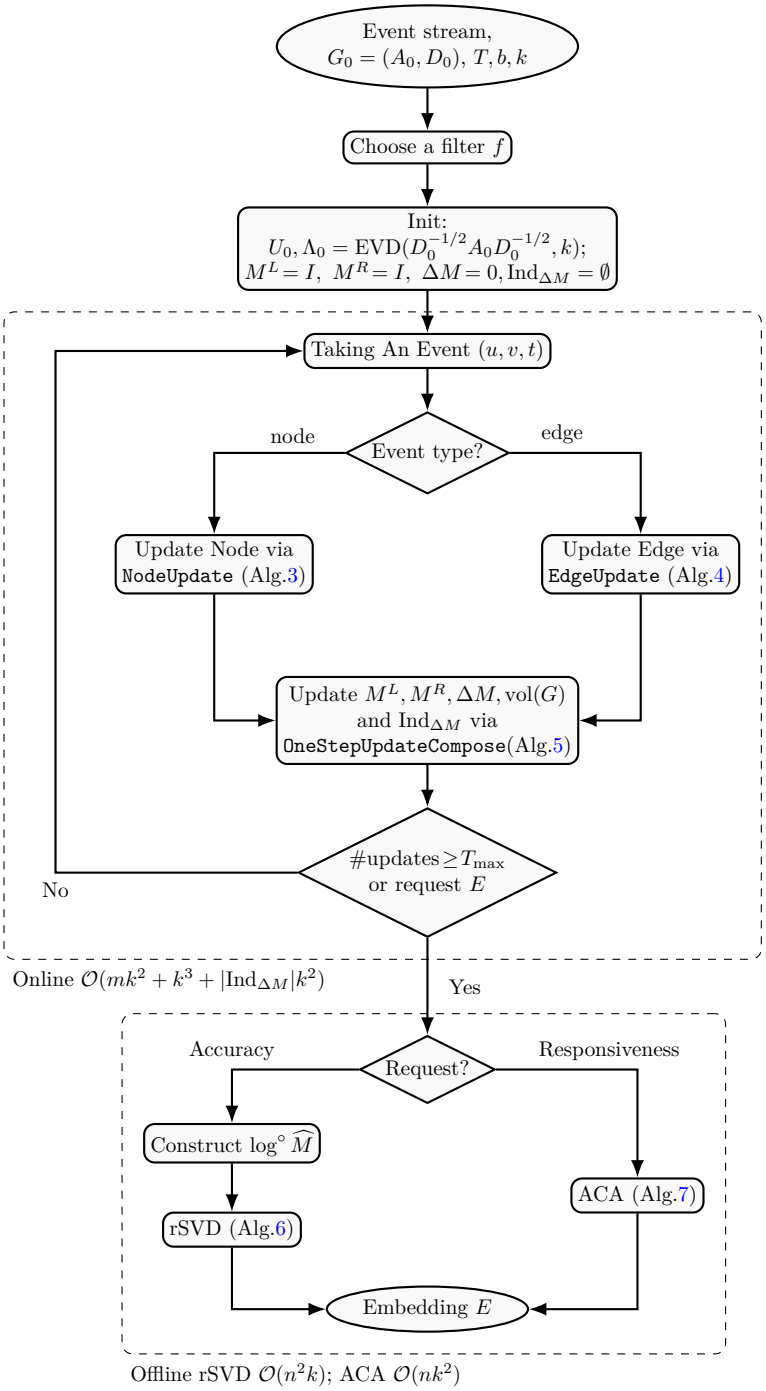

Figure 5.1: Overview of ATNMF. Online stage maintains series of lightweight components. Upon a trigger (step cap or embedding request), the offline stage computes an embedding via either rSVD or ACA, then outputs E for downstream tasks.

Algorithm 4: EdgeUpdate

Input: Initial orthonormal matrix U_0, $\mathrm{vol}(G_t)$, $M^L, M^R, \Delta M, \Lambda_t, D_t, x, y$
Output: $M_t^L, M_t^R,\ \Delta M_t,\ \Lambda_{t+1},\ \mathrm{vol}(G)$, nonzero indices in this update Ind

Initialise:
  $W \leftarrow I_n$; $G \leftarrow I_k$; $c \leftarrow 0_{k\times 2}$; $T \leftarrow 0_{2\times 2}$
$\mathrm{vol}(G) \leftarrow \mathrm{vol}(G_t) + 2\sum_{i,j}(xy^\top)[i,j]$
$\mathrm{Ind}_x \leftarrow \mathrm{supp}(x)$; $\mathrm{Ind}_y \leftarrow \mathrm{supp}(y)$; $\mathrm{Ind} \leftarrow \mathrm{Ind}_x \cup \mathrm{Ind}_y$;
$s_x \leftarrow \mathbf{1}^\top x$; $s_y \leftarrow \mathbf{1}^\top y$;
for $i \in \mathrm{Ind}$ **do**
  Calculate D_{t+1} by Eqn.(4.29); Construct $\widehat{X} = (\hat{x}\ \hat{y})$ by Eqn.(4.30)
  Construct W by Eqn.(4.31); Update G by Eqn.(4.33)
  Construct c by Eqn.(4.32)
  $T_{11} \mathrel{+}= \hat{x}_i^2$; $T_{22} \mathrel{+}= \hat{y}_i^2$; $T_{12} \mathrel{+}= \hat{x}_i\hat{y}_i$
Cholesky $G = R^\top R$; $z \leftarrow R^{-\top}c$;
$H \leftarrow \begin{pmatrix} T_{11} & T_{12} \\ T_{12} & T_{22} \end{pmatrix} - z^\top z$;
Cholesky $H = R_X^\top R_X$;
$C \leftarrow \begin{pmatrix} R\Lambda_t R^\top & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} z \\ R_X \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} z \\ R_X \end{pmatrix}^\top$;
$(U_C,\ \Gamma^+) \leftarrow \mathrm{EVD}(C,\ k)$; $\Lambda_{t+1} \leftarrow \Gamma^+[1\!:\!k, 1\!:\!k]$;
$M_t^L \leftarrow W$;
$M_t^R \leftarrow R^{-1}\big(U_C[1\!:\!k, 1\!:\!k] - R_t^{-\top} c R_X^{-1} U_C[k{+}1\!:\!k{+}2, 1\!:\!k]\big)$;
$\Delta M_t \leftarrow \widehat{X}_r\, R_X^{-1}\, U_C[k{+}1\!:\!k{+}r, 1\!:\!k]$
return $M_t^L, M_t^R, \Delta M_t, \Lambda_{t+1}, \mathrm{vol}(G), \mathrm{Ind}$

A notable observation is that each update step, whether for nodes or edges, incurs only moderate memory overhead. In the updated matrix, only M_t^R requires dense storage, with a size of $\mathcal{O}(k^2)$.

By contrast, M_t^L is a diagonal matrix where all but at most m entries are equal to one (in the node update case, the top $n \times n$ is such a diagonal matrix). Hence, it can be compactly stored using only $\mathcal{O}(m)$ space. Similarly, ΔM_t is an $n(+1) \times k$ matrix with at most m nonzero rows, requiring only $\mathcal{O}(mk)$ space.

Combining these properties yields a total space complexity of $\mathcal{O}(k^2 + mk)$ per update, demonstrating exceptional efficiency in large-scale scenarios.

The update procedure for maintaining these quantities is summarised in Algorithm 5.

Algorithm 5: OneStepUpdateCompose

Input: $M_t^L, M_t^R, \Delta M_t$, index set Ind, left total update matrix M^L, right total update M^R, total transformation ΔM, nonzero indices of ΔM $\mathrm{Ind}_{\Delta M}$

Output: $M^L, M^R, \Delta M$, $\mathrm{Ind}_{\Delta M}$

for i *in* Ind **do**
  $M^L[i,i] \leftarrow M_t^L[i,i]M^L[i,i]$
  $\Delta M[i,:] \leftarrow M_t^L[i,i]\Delta M[i,:]$
for i *in* $\mathrm{Ind}_{\Delta M}$ **do**
  $\Delta M[i,:] \leftarrow \Delta M[i,:]M_t^R$
for i *in* Ind **do**
  $\Delta M[i,:] \leftarrow \Delta M[i,:] + \Delta M_t[i,:]$
$\Delta M \leftarrow (\Delta M;\ \Delta M_t[\mathrm{end},:])$
if *this step is node update* **then**
  $M^L \leftarrow (M^L;\ 0)$
$\mathrm{Ind}_{\Delta M} \leftarrow \mathrm{Ind}_{\Delta M} \cup \mathrm{Ind}$
$M^R \leftarrow M^R M_t^R$
return $M^L, M^R, \Delta M, \mathrm{Ind}_{\Delta M}$

Thanks to the sparsity of both M_t^L and ΔM_t in each update, we can update the cumulative matrices M^L and ΔM within $\mathcal{O}(m)$ and $\mathcal{O}(|\mathrm{Ind}_{\Delta M}|k^2 + mk)$, respectively. The time required to obtain M^R is simultaneously bounded by $\mathcal{O}(k^3)$, which is typically substantially smaller than $\mathcal{O}(n)$ under our assumptions. However, as the number of update steps increases, $|\mathrm{Ind}_{\Delta M}|$ also grows monotonically. Consequently, the worst-case complexity could be $\mathcal{O}(nk^2 + mk)$. Ttherefore a periodic restart is recommended to prevent ΔM from becoming dense.

5.3 Offline Embedding

To obtain the final embedding, we construct the matrix $\log^\circ \widehat{M}$ and perform low-rank decomposition on it. However, the matrix $\log^\circ \widehat{M}$ is symmetric but dense, making full SVD computation impractical for large n.

To address this, we adopt randomised low-rank approximation techniques. In this work, we implement two methods: randomised SVD (rSVD) and adaptive cross approximation (ACA). Below, we briefly describe both algorithms.

Randomised SVD (rSVD) was introduced by N. Halko et al. in 2011 [27] and has become one of the most widely used and effective randomised low-rank approximation algorithms. It has found applications in signal processing, statistical learning, and network analysis, and is particularly well-suited to the structure of our problem. Given a dense matrix $A \in \mathbb{R}^{m\times n}$ and the target rank k, the basic rSVD procedure can be summarised as follows:

Algorithm 6: Randomised-SVD (rSVD)[27]

Input: $A \in \mathbb{R}^{m\times n}$, target rank k
Output: $U \in \mathbb{R}^{m\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$, $V \in \mathbb{R}^{n\times k}$

Draw a random matrix Ω;
$Y \leftarrow A\Omega$;
Compute QR: $Q, R \leftarrow \mathrm{QR}(Y)$;
$B \leftarrow Q^\top A$;
Compute SVD of the small matrix: $B = \tilde{U}\Sigma V^\top$;
$U \leftarrow Q\tilde{U}[:, 1:k]$, $\Sigma \leftarrow \Sigma[1:k, 1:k]$, $V \leftarrow V[:, 1:k]$;
return U, Σ, V;

The algorithm has a time complexity of $\mathcal{O}(mnk)$, which is a significant improvement over the $\mathcal{O}(mn^2)$ cost of exact SVD. Randomised SVD also offers theoretical error guarantees; for details, we refer the reader to Theorem 9.1 in [27].

However, although rSVD provides order-of-magnitude speedups when $n \gg 1$, the cost of computing the embedding is still impractical for time-sensitive applications. Moreover, applying rSVD requires explicitly constructing the dense matrix $\log^\circ \widehat{M}$. This step alone incurs $\Omega(n^2)$ memory, potentially leading to excessive runtime and out-of-memory issues.

To address this, we adopt an alternative method known as *Adaptive Cross Approximation (ACA)*, initially proposed by M. Bebendorf *et al.* in 2003 [28] for solving Fredholm integral equations, and later extended to matrix approximation in [29].

Algorithm 7: Adaptive Cross Approximation (ACA)[29]

Input: $A \in \mathbb{R}^{n\times m}$, tolerance $tol > 0$, target rank $k \geq 1$
Output: $U \in \mathbb{R}^{n\times r}$, $V \in \mathbb{R}^{r\times m}$ s.t. $A \approx UV$

Initialise:
  $U, V \leftarrow$ null matrix, $r \leftarrow 0$, $s \leftarrow 0$
while $r < k$ **do**
  Choose a row index $i \in \{1, \ldots, n\}$ not used before
  $\tilde{v} \leftarrow A[i, :] - U[i, :]V$
  if $\|\tilde{v}\|_2 = 0$ **then break**
  $j \leftarrow \arg\max_{1\leq j'\leq m} |\tilde{v}[j']|$
  if $\tilde{v}[j] = 0$ **then break**
  $v \leftarrow \tilde{v}/\tilde{v}[j]$
  $u \leftarrow A[:, j] - U\,V[:, j]$
  if $\|u\|_2 = 0$ **then break**
  $U \leftarrow (\,U \;\; u\,), \quad V \leftarrow \begin{pmatrix} V \\ v \end{pmatrix}, \quad r \leftarrow r + 1$
  $s \leftarrow \sqrt{s^2 + \|u\|_2^2 \|v\|_2^2}$
  if $\|u\|_2 \, \|v\|_2 \leq tol \cdot s$ **then break**
return U, V

The core idea of ACA is to construct a low-rank approximation of a matrix by selecting only a small number of informative rows and columns (i.e., "crosses"), thereby significantly reducing both computational and memory costs. In each iteration, ACA

selects a pivot row i (typically based on a specific pivoting strategy) from the residual matrix, which has not yet been well approximated. It then uses the corresponding row and column to construct a rank-one outer product that approximates the residual. This process is repeated recursively until the approximation error falls below a specified tolerance.

A rigorous pivots choice that has proven to have an exact error upper bound is to select i such that

$$A[i, j'] - A[i, J](A[I, J])^{-1}A[I, j'] \neq 0,$$

for some j, where I, J are row and column indices have been selected. In this strategy, Theorem 3.35 in [29] shows the element-wise error is upper bound in functional sense. However, this pivots choice can be tedious and time consuming; some articles have shown that if we select row indices i as $\arg\max_{1\leq i'\leq m} |\tilde{u}[i']|$ can also work [30–32].

The complexity of ACA lies in $O(k^2(m+n))$ [31]. Indeed, dominate operations in Alg.7, line `5` and `10`, need $\mathcal{O}(r(m+n))$ flops in each loop, which gives $\mathcal{O}(k^2(m+n))$ in total k loops. This enables us to obtain approximate embeddings in rational time for network-scale problems. Furthermore, ACA requires only partial rows and columns of the original matrix, meaning we need not explicitly construct the entire dense matrix $\log^\circ \widehat{M}$, thereby saving considerable computational time. Computations yield

$$\log^\circ \widehat{M}[i, j] = \log\left(\frac{\mathrm{vol}(G)}{b}\right) - \frac{1}{2}\log D[j, j] - \frac{1}{2}\log D[i, i] + \log\left(u_i f(\Lambda^{(k)})u_j^\top\right). \quad (5.1)$$

and requires $\mathcal{O}(T(u_i) + T(u_j) + k^2)$.

Naturally, as a trade-off, ACA also incurs greater error compared to randomised-SVD (Fig. 15). Therefore, when rapid response is required, ACA may be selected; whereas in scenarios demanding higher precision and more time, rSVD is a better choice. Furthermore, since the obtained low-rank approximations U and V are not unique, an additional step is required to achieve more stable results: $U = Q_U R_U$, $V^\top = Q_V R_V$, followed by SVD of the $k \times k$ submatrix $R_U R_V^\top$: $U_k \Sigma_k V_k^\top$. The embedding is then expressed as $Q_U U_k \sqrt{\Sigma_k}$. This process requires $\mathcal{O}(nk^2)$ computational complexity. We therefore present the final algorithm for computing the embedding (Alg.8).

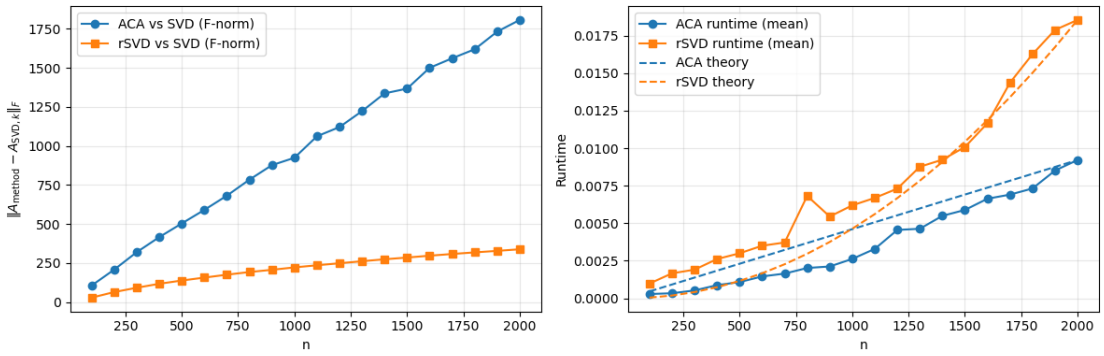

Figure 5.2: Comparison of ACA and randomised SVD (rSVD) against a truncated SVD baseline on random symmetric Gaussian matrices. Left: mean Frobenius gap to SVD-k over 10 trials with $k = 20$. Right: mean runtimes; dashed curves show scaled theoretical growth (rSVD $\sim n^2k$, ACA $\sim nk^2$).

Algorithm 8: EstimateEmbedding

Input: Initial orthonormal matrix U_0, $M^L, M^R, \Delta M, \Lambda, D, b_{\text{neg}}$, filter f, and embedding dimension k

Output: E

Calculate $\text{vol}(G) \leftarrow \sum D_t$
$U \leftarrow M^L U_0 M^R + \Delta M$
if *Fast response required* **then**
  Construct an oracle, orc, according to Eqn.(5.1)
  $U, V \leftarrow \text{ACA}(\log^\circ \widehat{M}, k, \text{orc})$;
  $Q_U, R_U \leftarrow \texttt{QR}(U)$; $Q_V, R_V \leftarrow \texttt{QR}(V)$
  $U_k, \Sigma_k, V_k \leftarrow \texttt{SVD}(R_U R_V^\top)$
  $E \leftarrow Q_U U_k \sqrt{\Sigma_k}$
else if *Precision required* **then**
  $\widehat{M} \leftarrow \frac{\text{vol}(G)}{b} D^{-1/2} U f(\Lambda) U^\top D^{-1/2}$
  $U, \Sigma, V \leftarrow \text{rSVD}(\log^\circ \widehat{M}, k)$
  $E \leftarrow U\sqrt{\Sigma}$
return E

5.4 ATNMF Algorithm and Complexity Analysis

Based on Theorem 9, we now present the complete ATNMF algorithm.

Algorithm 9: ATNMF

Input: Event stream *events*, initial graph $G = (A_0, D_0)$, filter f, window size T, number of negative samples b, embedding dimension k, max steps T_{max}

Output: Embedding E

Initialise:

  $(U_0, \Lambda_0) \leftarrow EVD(D_0^{-1/2} A_0 D_0^{-1/2}, k)$

  $M^L \leftarrow I$; $M^R \leftarrow I$

  $\Delta M \leftarrow 0$; counter $\leftarrow 0$

for *e in events* **do**

  `/* Online update */`

  if *e is node update* **then**

    $M_t^L, M_t^R, \Delta M_t, \Lambda_t, \mathrm{vol}(G), \mathrm{Ind} \leftarrow$ `NodeUpdate`$(U_0, \mathrm{vol}(G), M^L, M^R, \Delta M, \Lambda_0, b)$

  else

    $M_t^L, M_t^R, \Delta M_t, \Lambda_0, \mathrm{vol}(G), \mathrm{Ind} \leftarrow$ `EdgeUpdate`$(U_0, \mathrm{vol}(G), M^L, M^R, \Delta M, \Lambda_t, x, y)$

  $M^L, M^R, \Delta M, \mathrm{Ind}_{\Delta M} \leftarrow$ `OneStepUpdateCompose`$(M_t^L, M_t^R, \Delta M_t, \mathrm{Ind}, M^L, M^R, \Delta M, \mathrm{Ind}_{\Delta M})$

  counter $\leftarrow$ counter $+1$

  if counter $> T_{max}$ *or* requiring embedding **then**

    `/* Offline update */`

    $E \leftarrow$ `EstimateEmbedding`$(U_0, M^L, M^R, \Delta M, \Lambda_t, D, b, f)$

    return E

The overall computational complexity of ATNMF can be divided into two main components. The first is the online update phase, which includes two subroutines: node/edge updates and update composition. As detailed in Section 4.3, a single node/edge update costs $\mathcal{O}(mk^2 + k^3 + m\,T(u_i))$. Under our storage scheme, a row of U is reconstructed via

$$U[i,:] \;=\; M^L[i,i]\,U_0[i,:]\,M^R \;+\; \Delta M[i,:]$$

takes $\mathcal{O}(k^2)$ operations; hence $T(u_i) = \mathcal{O}(k^2)$ and the per–event cost becomes $\mathcal{O}(mk^2 + k^3)$. Section 5.2 further shows that the composition step (Alg. 5) costs $\mathcal{O}(|\mathrm{Ind}_{\Delta M}|\,k^2 + mk + k^3)$. Aggregating these terms yields an online complexity of

$$\mathcal{O}\big(mk^2 \;+\; |\mathrm{Ind}_{\Delta M}|\,k^2 \;+\; k^3\big).$$

The second part is the offline embedding phase. For rSVD, we first construct the dense matrix

$$\log^\circ \widehat{M} = \log^\circ \frac{\mathrm{vol}(G)}{b} D^{-1/2} U f(\Lambda) U^\top D^{-1/2},$$

which requires $\mathcal{O}(n^2 k)$ operations. As rSVD itself also needs $\mathcal{O}(n^2 k)$, the total cost of the rSVD-based offline phase is $\mathcal{O}(n^2 k)$. For ACA, each loop needs extra $\mathcal{O}(k^2)$ flops for extracting u_i, the total complexity of the ACA-based offline phase is $\mathcal{O}(nk^2 + k^3) \approx \mathcal{O}(nk^2)$ under the assumption $k \ll n$.

Chapter 6

Numerical Experiments

In this chapter, we evaluate the performance of our proposed algorithm ATNMF against three representative baseline methods across two common downstream tasks on four temporal graph datasets. Our results demonstrate that ATNMF is a competitive approach for dynamic network embedding, especially in scenarios where the graph exhibits specific structural patterns.

6.1 Experiment Setup

6.1.1 Data Sets

We use four datasets, including two real-world temporal networks and two synthetic temporal graphs:

- **MathOverflow Temporal Network**[11]
  A dynamic network collected from the MathOverflow platform on Stack Exchange, capturing timestamped user interactions. The dataset consists of 24,818 nodes and 506,550 temporal edges spanning over 2,350 days.

- **email-Eu-core Temporal Network**[11]
  A temporal communication network representing timestamped email exchanges between members of a European research institution. The network includes 986 nodes and 332,334 temporal edges, covering a timespan of 803 days.

- **Comms-Bridge (Synthetic, GitHub)**
  A synthetic graph consisting of multiple densely connected communities, designed to evaluate the algorithm's ability to capture community evolution and inter-community dynamics. The network contains 1,000 nodes and 5,040 temporal edges.

- **Random-Regular (Synthetic, GitHub)**
  A synthetic dynamic graph based on a connected 10-regular random graph, with edges sequentially added over time. This setup is used to assess the algorithm's capability to learn long-range structural dependencies. The graph consists of 1,000 nodes and 5,000 temporal edges. (See Fig. 6.1)

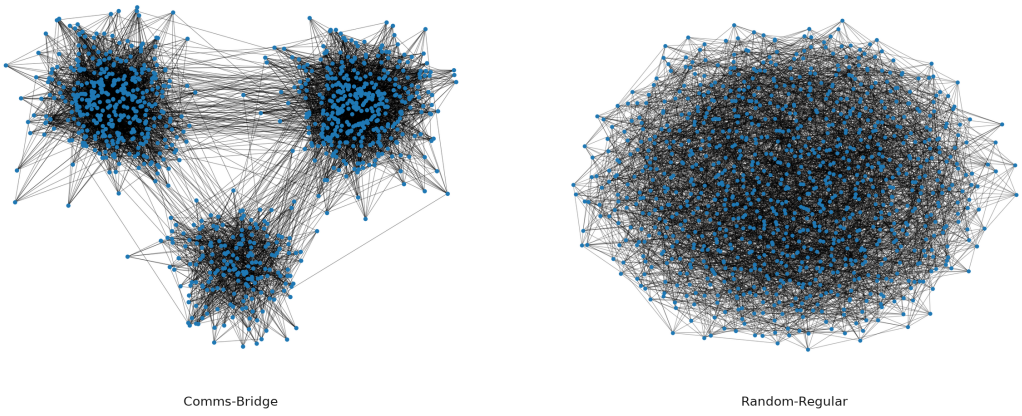

Figure 6.1: The network diagrams of two synthetic networks at the end of evolution.

In order to reflect realistic prediction scenarios and avoid evaluation bias from random splits, a time-respecting partition is employed.

6.1.2 Compared Methods

We compare ATNMF against the following baseline methods:

- **DNE** [7] A Skip-Gram based dynamic network embedding method that learns node representations incrementally by selectively updating only the most affected nodes.
- **CTDNE** [33] A method that performs time-respecting random walks, strictly following the chronological order of edge timestamps. The resulting walk sequences are then used to train node embeddings via the Skip-Gram model.
- **TGN** [34] A general framework for dynamic graph representation learning based on event streams. Each node maintains a learnable memory, which is updated via a message-passing mechanism upon arrival of a new event, enabling time-sensitive embeddings.

6.1.3 Tasks Performed and Evaluation Metrics

We evaluate method performance on two downstream tasks: Link Prediction (LP) and Graph Reconstruction (GR) and use Average Precision (AP) and ROC-AUC as metrics.

LP ranks currently unconnected node pairs by their likelihood of forming an edge given a snapshot or history, thus predicting missing or future links. While GR uses learned node embeddings with a simple decoder to recover the observed adjacency matrix, assessing how well structure is preserved.

As for metrics adopted, ROC-AUC is the area under the receiver operating characteristic curve and summarises how well scores separate positives from negatives across thresholds; average precision summarises the precision-recall curve as a weighted mean of precision over recall increments and is especially informative for imbalanced data. For both metric, larger values indicate stronger discrimination.

6.2 Experimental Results

Throughout all experiments, we set the negative sampling parameter $b = 1$.

6.2.1 Effectiveness

In this subsection, we assess the effectiveness of our algorithm with 3 other baselines on two real-world datasets.

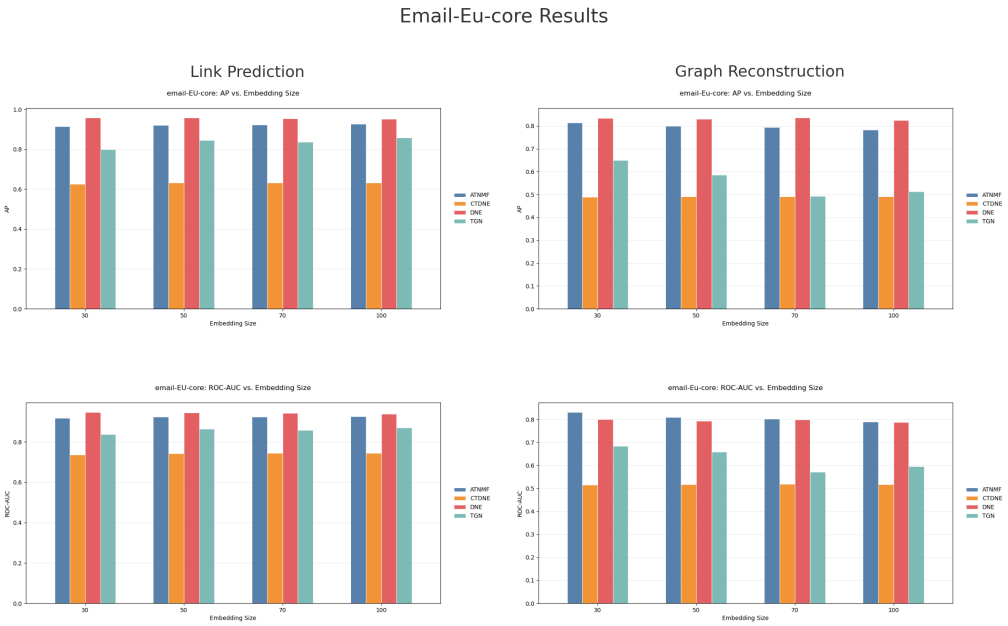

Figure 6.2: Comparison of AP and ROC-AUC across different embedding dimensions in link prediction and graph reconstruction tasks on email-Eu-core. Window size $T = 50$, filter: uniform.

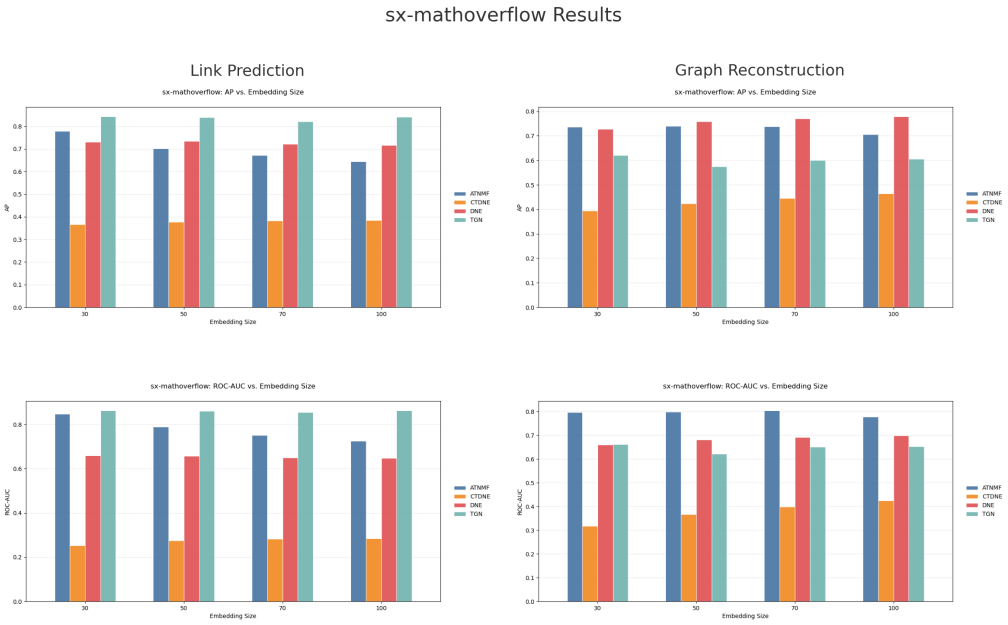

Figure 6.3: Comparison of AP and ROC-AUC across different embedding dimensions in link prediction and graph reconstruction tasks on sx-mathoverflow. Window size $T = 50$, filter: uniform.

Figure 6.2 shows that on the Email-Eu-core dataset, ATNMF consistently delivers

stable performance across various embedding dimensions, outperforming CTDNE and competitive with rest 2 baseline methods overall. In LP task, ATNMF performs slightly below TGN and is roughly on par with DNE, while in GR task, ATNMF achieves better results. Specifically, with lower embedding dimensions, ATNMF's AP matches that of DNE and surpasses the other two baselines; at higher dimensions, AP of ATNMF is slightly lower than that of DNE.

The results on the sx-MathOverflow dataset, presented in Figure 6.3, display a different pattern. Notably, in LP task, ATNMF performs very well in low dimension, and is surpassed by DNE as the dimension grows in AP but remain a relatively high ROC-AUC. In GR task, ATNMF outperforms other algorithms in the metric of ROC-AUC, while quite competitive when measured in AP.

In summary, ATNMF exhibits strong adaptability on both small and large scale networks and its great awareness of preserving structure information on real world datasets.

6.2.2 Update Efficiency

		Average ms per event	
Algorithm	**Dim.**	**email-EU-core**	**sx-mathoverflow**
ATNMF	10	0.268	0.256
	30	0.266	0.287
	50	0.264	0.799
	70	0.261	0.270
	90	0.259	0.259
	110	0.254	0.253
DNE	30	0.010	0.027
CTDNE		0.007	0.023
TGN		0.068	0.090

Table 6.1: Per-event processing time across datasets and embedding dimensions. Window size $T = 50$, filter: uniform.

Table 6.1 presents the average per-event processing times of each algorithm across various datasets and embedding dimensions. Notably, ATNMF exhibits consistent update times across embedding dimensions ranging from 10 to 110, with average durations remaining within approximately $0.25 \sim 0.27$ milliseconds. This indicates that ATNMF's update cost is largely insensitive to embedding dimension and demonstrates strong scalability. We observe that DNE, CTDNE, and TGN achieve much

lower per-event times than ATNMF. We think this is partly due to their implementation in Torch, which leverages parallel computation and hardware acceleration. Despite ATNMF being slower compared to the other methods, its per-event processing time still falls within the millisecond level, meeting our design goal.

6.2.3 Filter Functions' Effect on Embedding Quality

Filters & Baseline	**Comms–Bridges**		**email–EU–Core**		**sx–mathoverflow**		**Random**	
	ROC–AUC	AP	ROC–AUC	AP	ROC–AUC	AP	ROC–AUC	AP
Uniform	0.766	0.755	0.921	0.911	0.885	0.847	0.496	0.509
Exp($\gamma = 1.0$)	**0.831**	**0.881**	0.918	0.917	0.678	0.640	0.484	0.492
HeadPower($a = 2$)	0.787	0.804	0.917	0.913	0.787	0.712	0.488	0.499
SigmoidHead($r_0 = 60,\ \tau = 30$)	0.779	0.795	0.917	0.912	0.868	0.811	0.492	0.505
TailPower($a = 2.5$)	0.611	0.534	0.608	0.567	0.885	0.863	**0.576**	**0.576**
GaussianTail($\sigma = 40$)	0.582	0.515	0.589	0.552	**0.897**	**0.879**	0.524	0.533
DNE	0.562	0.534	**0.948**	**0.959**	0.658	0.731	0.501	0.496
CTDNE	0.832	0.856	0.736	0.626	0.252	0.366	0.491	0.479
TGN	0.606	0.577	0.867	0.848	0.845	0.818	0.489	0.490

Table 6.2: Performance of different weight function families (top, $T = 200$) and baselines (bottom) across four datasets. Higher is better; bold numbers are the best in each column in the top block.

The results in Table 6.2 indicate that different filter functions offer distinct advantages depending on the network type. On the Comms–Bridges dataset, which features well-defined community structure, exponential weight filters perform best by emphasizing short-range connectivity. In the communication-rich email–EU–Core network, both uniform and exponential filters effectively balance local and global information. In the heterogeneous, long-tailed interaction patterns of the sx–MathOverflow network, tail-weighted filters better preserve long-range dependencies, resulting in the best performance. In the unstructured random graph, which lacks discernible patterns, the TailPower filter performs relatively better.

Compared to baseline methods, ATNMF consistently achieves superior performance across datasets. This underscores the pivotal role of filter function design in enhancing the performance and adaptability of temporal network embeddings.

Chapter 7

Conclusion and Future Work

This dissertation addresses the problem of efficient and theoretically grounded dynamic network embedding, motivated by the computational bottlenecks and error accumulation challenges posed by real-time updates in evolving graphs. While classical static methods such as NetMF offer spectral insight into random-walk-based proximity, they become computationally prohibitive in the dynamic setting, where even small perturbations may require full recomputation. Moreover, existing dynamic embedding approaches often lack transparent error control or fail to capture the underlying matrix structure of the proximity operator.

To overcome these limitations, we propose ATNMF, a method uses online update and offline decomposition strategy to accelerate the whole embedding process and maintain excellent embedding quality of NetMF. In the online stage, our method focuses on efficient updates of multipliers in the NetMF matrix: an orthonormal matrix and a diagonal matrix. Our work shows the former one can be updated and approximated by the formula $M^L U_0 M^R + \Delta M$, where only small matrices are updated at each time step; while the diagonal matrix can be estimated by a small core matrix eigen-decomposition. In the offline stage, rSVD and ACA are utilised to achieve the final embedding. Beyond the algorithm, we also derive explicit error bounds for both single-event and multi-step updates, showing that the overall error accumulates in a controlled manner under certain growth assumptions.

Empirical evaluations across both real-world and synthetic temporal graphs confirm that ATNMF holds a balance between efficiency and accuracy. It achieves competitive or superior performance in link prediction and graph reconstruction tasks compared to 3 baselines, DNE, CTDNE, and TGN, while constraining per-event runtime within milliseconds. The performance of different filter functions also shows the excellent flexibility of ATNMF on networks with certain structures.

Several promising extensions of the ATNMF framework remain to be explored. First, our current spectral filter is constructed using non-negative-weights polynomials, which amplifies the contribution of higher-order walks and may exacerbate error propagation. An alternative direction is to design *non-monotonic filters* to suppress contributions from certain transition orders. From the perspective of random walks, this corresponds to penalising certain path lengths that may be detrimental in graphs with specific structural patterns. From a spectral perspective, this approach selectively attenuates eigenvalues outside a desired spectral band $[-1, 1]$, potentially reducing the deviation between the filtered proximity matrix and its NetMF-inspired approximation. More generally, introducing *non-polynomial filters* (e.g., rational or exponential kernels) may further enhance flexibility, although their theoretical analysis would be significantly more involved.

Second, the current formulation of ATNMF assumes undirected graphs without isolated nodes, as required in NetMF. Extending the framework to support *directed networks* and handle temporary or persistent *isolated vertices* is nontrivial. For isolated nodes, one possible solution is to dynamically prune rows and columns corresponding to zero-degree nodes at each time step. For directed graphs, the matrix $\sum_{r=1}^{T} P^r D^{-1}$ no longer admits symmetric diagonalisation, so approximating it would require alternative matrix factorisation or augmentation strategies.

Finally, constrained by hardware, our experiments were not able to perform evaluation on substantially larger graphs. Future work should evaluate ATNMF's embedding quality and at much larger scales, and incorporate parallel or distributed execution to further reduce latency and memory.

Bibliography

[1] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and Models for Network Data and Link Analysis*. Cambridge University Press, 1 edition, July 2016. ISBN 978-1-107-12577-3 978-1-316-41832-1 978-1-107-56481-7. doi: 10.1017/CBO9781316418321. URL https://www.cambridge.org/core/product/identifier/9781316418321/type/book.

[2] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2019. doi: 10.1109/TKDE.2018.2849727.

[3] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 459–467, February 2018. doi: 10.1145/3159652.3159706. URL http://arxiv.org/abs/1710.02971. arXiv:1710.02971 [cs].

[4] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2134–2144, 2018.

[5] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed Network Embedding for Learning in a Dynamic Environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396, Singapore Singapore, November 2017. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3132919. URL https://dl.acm.org/doi/10.1145/3132847.3132919.

[6] Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. Continuous-time link prediction via temporal dependent graph neural network. In *Proceedings of the web conference 2020*, pages 3026–3032, 2020.

[7] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2086–2092, Stockholm, Sweden, July 2018. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-2-7. doi: 10.24963/ijcai.2018/288. URL https://www.ijcai.org/proceedings/2018/288.

[8] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*, pages 519–527, 2020.

[9] Luwei Yang, Zhibo Xiao, Wen Jiang, Yi Wei, Yi Hu, and Hao Wang. Dynamic heterogeneous graph embedding using hierarchical attentions. In *European conference on information retrieval*, pages 425–432. Springer, 2020.

[10] Jure Leskovec, Jon Kleinberg, and Christos 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*, pages 177–187, 2005.

[11] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 601–610, 2017.

[12] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 221–230. IEEE, 2016.

[13] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 333–341. ACM, 2018.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[16] Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/hash/b78666971ceae55a8e87efb7cbfd9ad4-Abstract.html.

[17] Yuyang Xie, Yuxiao Dong, Jiezhong Qiu, Wenjian Yu, Xu Feng, and Jie Tang. Sketchne: Embedding billion-scale networks accurately in one hour. *IEEE Transactions on Knowledge and Data Engineering*, 35(10):10666–10680, 2023.

[18] Sadamori Kojaku, Jisung Yoon, Isabel Constantino, and Yong-Yeol Ahn. Residual2vec: Debiasing graph embedding with random graphs. *Advances in Neural Information Processing Systems*, 34:24150–24163, 2021.

[19] Jundong Li, Liang Wu, Ruocheng Guo, Chenghao Liu, and Huan Liu. Multi-level network embedding with boosted low-rank matrix approximation. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*, pages 49–56, 2019.

[20] Yuntian He, Saket Gurukar, and Srinivasan Parthasarathy. Efficient fair graph representation learning using a multi-level framework. In *Companion Proceedings of the ACM Web Conference 2023*, pages 298–301, 2023.

[21] Asan Agibetov. Neural graph embeddings as explicit low-rank matrix factorization for link prediction. *Pattern Recognition*, 133:108977, 2023.

[22] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[23] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[24] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing*, 21(2), October 1999. ISSN SJOCE3. doi: 10.1137/S1064827597329266. URL https://www.osti.gov/biblio/20015659. Institution: Pennsylvania State Univ., University Park, PA (US).

[25] Eugene Vecharynski and Yousef Saad. Fast updating algorithms for latent semantic indexing. *SIAM Journal on Matrix Analysis and Applications*, 35(3): 1105–1131, 2014.

[26] Haoran Deng, Yang Yang, Jiahe Li, Haoyang Cai, Shiliang Pu, and Weihao Jiang. Accelerating Dynamic Network Embedding with Billions of Parameter Updates to Milliseconds. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 414–425, August 2023. doi: 10.1145/3580305.3599250. URL http://arxiv.org/abs/2306.08967. arXiv:2306.08967 [cs].

[27] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[28] Mario Bebendorf and Sergej Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[29] Mario Bebendorf. *Hierarchical matrices: a means to efficiently solve elliptic boundary value problems*. Springer, 2008.

[30] Vladislav A Yastrebov and Camille Noûs. Adaptive cross approximation with a geometrical pivot choice: Aca-gp method. *arXiv preprint arXiv:2502.03886*, 2025.

[31] Kezhong Zhao, Marinos Vouvakis, and Jin-Fa Lee. Application of the multilevel adaptive cross-approximation on ground plane designs. In *2004 International Symposium on Electromagnetic Compatibility (IEEE Cat. No. 04CH37559)*, volume 1, pages 124–127. IEEE, 2004.

[32] Kezhong Zhao, M.N. Vouvakis, and Jin-Fa Lee. The adaptive cross approximation algorithm for accelerated method of moments computations of emc problems. *IEEE Transactions on Electromagnetic Compatibility*, 47(4):763–773, 2005. doi: 10.1109/TEMC.2005.857898.

[33] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-Time Dynamic Network Embeddings. In *Companion of the The Web Conference 2018 on The Web Conference 2018 -*

WWW '18, pages 969–976, Lyon, France, 2018. ACM Press. ISBN 978-1-4503-5640-4. doi: 10.1145/3184558.3191526. URL http://dl.acm.org/citation.cfm?doid=3184558.3191526.

[34] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs, October 2020. URL http://arxiv.org/abs/2006.10637. arXiv:2006.10637 [cs].

[35] Jelena Smiljanic and Christopher Blöcker. Similarity-based Link Prediction from Modular Compression of Network Flows.

[36] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, January 2020. ISSN 09507051. doi: 10.1016/j.knosys.2019.06.024. URL https://linkinghub.elsevier.com/retrieve/pii/S0950705119302916.

[37] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. Dynamic network embedding survey. *Neurocomputing*, 472:212–223, February 2022. ISSN 09252312. doi: 10.1016/j.neucom.2021.03.138. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231221016234.

[38] Xiaoran Zhang, Xikun Huang, Yuexing Han, Gouhei Tanaka, and Bing Wang. Local Topology-Based Reweighted Incremental Representation Learning for Dynamic Networks, April 2025. URL https://papers.ssrn.com/abstract=5217562.

[39] Zenan Xu, Zijing Ou, Qinliang Su, Jianxing Yu, Xiaojun Quan, and Zhenkun Lin. Embedding Dynamic Attributed Networks by Modeling the Evolution Processes, October 2020. URL http://arxiv.org/abs/2010.14047. arXiv:2010.14047 [cs].

[40] Hao Peng, Jianxin Li, Hao Yan, Qiran Gong, Senzhang Wang, Lin Liu, Lihong Wang, and Xiang Ren. Dynamic Network Embedding via Incremental Skip-gram with Negative Sampling, June 2019. URL http://arxiv.org/abs/1906.03586. arXiv:1906.03586 [cs].

[41] Jianshu Zhao, Jean Pierre Both, and Rob Knight. Ultra-fast and Efficient Network Embedding for Gigascale Biological Datasets, June 2025. URL https://www.biorxiv.org/content/10.1101/2025.06.18.660497v1. Pages: 2025.06.18.660497 Section: New Results.

[42] Ryan N. Lichtenwalter, Jake T. Lussier, and Nitesh V. Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 243–252, New York, NY, USA, July 2010. Association for Computing Machinery. ISBN 978-1-4503-0055-1. doi: 10.1145/1835804.1835837. URL https://doi.org/10.1145/1835804.1835837.

[43] Ryan N. Lichtenwalter, Jake T. Lussier, and Nitesh V. Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 243–252, New York, NY, USA, July 2010. Association for Computing Machinery. ISBN 978-1-4503-0055-1. doi: 10.1145/1835804.1835837. URL https://dl.acm.org/doi/10.1145/1835804.1835837.

[44] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed Network Embedding for Learning in a Dynamic Environment. pages 387–396, November 2017. doi: 10.1145/3132847.3132919. URL http://arxiv.org/abs/1706.01860. arXiv:1706.01860 [cs].

[45] Horst D. Simon and Hongyuan Zha. Low-Rank Matrix Approximation Using the Lanczos Bidiagonalization Process with Applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274, January 2000. ISSN 1064-8275, 1095-7197. doi: 10.1137/S1064827597327309. URL http://epubs.siam.org/doi/10.1137/S1064827597327309.

[46] Yuji Nakatsukasa. Eigenvalue perturbation bounds for Hermitian block tridiagonal matrices. *Applied Numerical Mathematics*, 62(1):67–78, January 2012. ISSN 0168-9274. doi: 10.1016/j.apnum.2011.09.010. URL https://www.sciencedirect.com/science/article/pii/S0168927411001838.

[47] Zhuoming Li and Darong Lai. Dynamic Network Embedding via Temporal Path Adjacency Matrix Factorization. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, pages 1219–1228, New York, NY, USA, October 2022. Association for Computing Machinery.

ISBN 978-1-4503-9236-5. doi: 10.1145/3511808.3557302. URL https://dl.acm.org/doi/10.1145/3511808.3557302.

[48] Roy Mitz, Nir Sharon, and Yoel Shkolnisky. Symmetric rank-one updates from partial spectrum with an application to out-of-sample extension, July 2019. URL http://arxiv.org/abs/1710.02774. arXiv:1710.02774 [math].

[49] HyungSeon Oh and Zhe Hu. Multiple-rank modification of symmetric eigenvalue problem. *MethodsX*, 5:103–117, January 2018. ISSN 2215-0161. doi: 10.1016/j.mex.2018.01.001. URL https://www.sciencedirect.com/science/article/pii/S2215016118300062.

[50] Eugene Vecharynski and Yousef Saad. Fast Updating Algorithms for Latent Semantic Indexing. *SIAM Journal on Matrix Analysis and Applications*, 35(3): 1105–1131, January 2014. ISSN 0895-4798, 1095-7162. doi: 10.1137/130940414. URL http://epubs.siam.org/doi/10.1137/130940414.

[51] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *The World Wide Web Conference*, pages 1509–1520, San Francisco CA USA, May 2019. ACM. ISBN 978-1-4503-6674-8. doi: 10.1145/3308558.3313446. URL https://dl.acm.org/doi/10.1145/3308558.3313446.

[52] Fast and Accurate Randomized Algorithms for Linear Systems and Eigenvalue Problems. URL https://epubs.siam.org/doi/epdf/10.1137/23M1565413.

[53] Cameron Musco and Christopher Musco. Recursive Sampling for the Nystrom Method. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/a03fa30821986dff10fc66647c84c9c3-Abstract.html.

[54] Yatian Wang, Nian-Ci Wu, Yuqiu Liu, and Hua Xiang. A generalized Nyström method with subspace iteration for low-rank approximations of large-scale nonsymmetric matrices. *Applied Mathematics Letters*, 166:109531, July 2025. ISSN 08939659. doi: 10.1016/j.aml.2025.109531. URL https://linkinghub.elsevier.com/retrieve/pii/S0893965925000813.

[55] Yuji Nakatsukasa. Fast and stable randomized low-rank matrix approximation, September 2020. URL http://arxiv.org/abs/2009.11392. arXiv:2009.11392 [math].

Appendix A

NetMF Small Algorithm

Algorithm 10: NetMF for a Small Window Size T [3]

Input: Graph G, window size T, dimension d
Output: Embedding matrix $U_d\sqrt{\Sigma_d}$

for $r = 1$ **to** T **do**
    Compute P^r
Compute $M = \frac{\mathrm{vol}(G)}{bT}\left(\sum_{r=1}^{T} P^r\right)D^{-1}$;
Compute $M' = \max(M, 1)$;
Rank-d approximation by SVD: $\log M' = U_d\Sigma_d V_d^\top$;
return $U_d\sqrt{\Sigma_d}$ as network embedding;

Appendix B

Additional Data

t	fro_error_M	upper_bound_M_cum	vol/dmin	rest term
0	–	–	–	–
1	103.974	2344.094	300.004	7.814
2	103.964	2362.080	300.013	7.873
3	103.957	2388.439	300.024	7.961
4	103.955	2417.254	300.035	8.057
5	103.948	2440.878	300.042	8.135
6	103.939	2462.082	300.050	8.206
7	103.956	2493.139	300.064	8.309
8	103.957	2536.734	300.081	8.453
9	103.960	2575.739	300.092	8.583
10	103.968	2615.518	300.109	8.715
11	103.967	2656.963	300.121	8.853
12	103.948	2698.063	300.138	8.989
13	103.946	2735.384	300.147	9.113
14	103.945	2772.058	300.163	9.235
15	103.950	2817.548	300.178	9.386
16	103.944	2861.162	300.192	9.531
17	103.945	2901.545	300.205	9.665
18	103.939	2943.525	300.219	9.805
19	103.939	2989.949	300.235	9.959

Table B.1: First 20 rows of the simplified error table.

Appendix C

Matrix Dimensions Summary

Symbol	Shape	Description
Base at time t (with n nodes, rank k)		
A_t	$n \times n$	Adjacency matrix.
D_t	$n \times n$ (diagonal)	Degree matrix.
S_t	$n \times n$	Normalised adjacency (NAM).
$U_t,\ \Lambda_t$	$n \times k,\ k \times k$	Rank-k eigentriple s.t. $S_t \approx U_t \Lambda_t U_t^\top$.
W	$n \times n$	A diagonal matrix.
Node update (add one node; new size n+1)		
b	$n \times 1$	New node's incident-edge vector (sparse).
$\hat{b}$	$(n+1) \times 1$	Normalised/augmented update vector.
$\widehat{U}_t,\ R_t$	$n \times k,\ k \times k$	Thin QR of WU_t. $R_t \succ 0$.
$c,\ z,\ \alpha$	$k \times 1,\ k \times 1,\ 1 \times 1$	Intermediate quantities.
G	$k \times k$	$G \succ 0$.
$M_t^L,\ M_t^R,\ \Delta M_t$	$(n+1) \times n,\ k \times k,$ $(n+1) \times k$	Matrices s.t. $U_{t+1} = M_t^L U_t M_t^R + \Delta M_t$. Top n rows of M_t^L is diagonal, the last row is 0s.
Edge update (rank-2 perturbation; size stays n)		
x, y	$n \times 1$	Endpoint indicator (sparse) vectors.
$X,\ \widehat{X}$	$n \times 2$	Stacked update matrix.
S	2×2	A constant matrix.
$\hat{U}_t,\ R_t$	$n \times k,\ k \times k$	Thin QR of $WU_t = \hat{U}_t R_t$. $R_t \succ 0$.
$Q_X,\ R_X$	$n \times 2,\ 2 \times 2$	Thin QR of $(I - \hat{U}_t \hat{U}_t^\top)\hat{X} = Q_X R_X$.
c	$k \times 2$	An intermediate quantity.
G	$k \times k$	An intermediate matrix. $G \succ 0$.
H	2×2	An intermediate matrix. $H \succ 0$.
$M_t^L,\ M_t^R,\ \Delta M_t$	$n \times n,\ k \times k,\ n \times k$	Matrices s.t. $U_{t+1} = M_t^L U_t M_t^R + \Delta M_t$.
Filtering / NetMF		
$f(\Lambda)$	$k \times k$	Spectral filter. A diagonal matrix
$\widehat{M}$	$n \times n$	Pre-logged NetMF-style matrix.
Output		
E	$n \times k$	Final node embedding.

Table C.1: ATNMF matrix/vector dimensions. Here n is the number of nodes at time t and k is the retained rank / embedding dimension. After a node addition, the row dimension becomes n+1.