Everything Evolves in Personalized PageRank

Zihao Li*
University of Illinois at Urbana-Champaign
Illinois, USA
zihaoli5@illinois.edu

Dongqi Fu*
University of Illinois at Urbana-Champaign
Illinois, USA
dongqif2@illinois.edu

Jingrui He
University of Illinois at Urbana-Champaign
Illinois, USA
jingrui@illinois.edu

ABSTRACT

Personalized PageRank, as a graphical model, has been proven as an effective solution in many applications such as web page search, recommendation, etc. However, in the real world, the setting of personalized PageRank is usually dynamic like the evolving World Wide Web. On the one hand, the outdated PageRank solution can be sub-optimal for ignoring the evolution pattern. On the other hand, solving the solution from the scratch at each timestamp causes costly computation complexity. Hence, in this paper, we aim to solve the Personalized PageRank effectively and efficiently in a fully dynamic setting, i.e., *every component in the Personalized PageRank formula is dependent on time*. To this end, we propose the EvePPR method that can track the exact personalized PageRank solution at each timestamp in the fully dynamic setting, and we theoretically and empirically prove the accuracy and time complexity of EvePPR. Moreover, we apply EvePPR to solve the dynamic knowledge graph alignment task, where a fully dynamic setting is necessary but complex. The experiments show that EvePPR outperforms the state-of-the-art baselines for similar nodes retrieval across graphs.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**;

KEYWORDS

Personalized PageRank, Similarly Retrieval, Knowledge Graphs

ACM Reference Format:
Zihao Li, Dongqi Fu, and Jingrui He. 2023. Everything Evolves in Personalized PageRank. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583474

1 INTRODUCTION

Personalized PageRank, as a classic graphical model, has been proven as an effective solution in many application domains, such as web page search, recommendation, disinformation detection, and social network analysis [6, 8, 13, 32, 38, 40, 49]. However, in the real world, the setting of Personalized PageRank is dynamic, i.e., the latent structure for random walks is dependent on time [1, 7, 9, 15] such like the evolving World Wide Web (WWW). On the one hand, an outdated (i.e., obtained from the previous structure) PageRank solution may be sub-optimal for the current structure by ignoring the graph evolution patterns [42]. On the other hand, just solving the PageRank solution at each timestamp from the scratch causes unaffordable computation complexity, especially when the input graph structure is large [25].

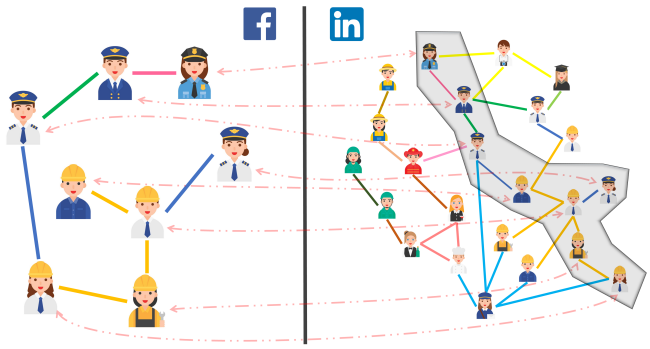


Figure 1: Example of the alignment between Facebook and LinkedIn, where each node is an attributed user and different color solid lines stand for different attributed edges. The red dashed line denotes the alignment of node pairs cross two graphs, the reason of alignment is that the shaded area in LinkedIn is similar to the given Facebook graph, in terms of topological structure, node attributes, and edge attributes.

To this end, we aim to study how to track the personalized PageRank vector at each timestamp effectively and efficiently. Moreover, different from previous PageRank tracking methods [25, 42], we extend the difficulty of tracking personalized PageRank to a **fully dynamic setting**, i.e., *every component in the Personalized PageRank formula is dependent on time* such as transition probability matrix and stochastic vector. To the best of our knowledge, the state-of-the-art tracking methods [10, 24, 25, 27, 42] only allow the transition matrix evolves between two consecutive timestamps.

A fully dynamic setting for the personalized PageRank tracking still keeps untouched because of the difficulty but is very important to many downstream tasks. **Firstly**, in this paper, we propose the EvePPR method, which is proven to track the exact personalized PageRank vector for each timestamp in the fully dynamic setting, and we prove the tracking accuracy and demonstrate the time complexity. **Secondly**, to show the importance of the fully dynamic setting, we start from proving that the personalized PageRank can solve the *knowledge graph alignment* task, as shown in Figure 1, which is a task of searching similar nodes across two knowledge graphs (e.g., given a user in LinkedIn network and try to find the

*Both authors contributed equally to this research.

WWW '23, April 30-May 4, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583474

same (or similar) user in Facebook). The fully dynamic setting of personalized PageRank allows topological structures, node attributes, and edge attributes of the two given knowledge graphs evolve simultaneously, which is complex and has no effective solutions until now [11, 36, 37, 46, 50] but more realistic and similar to the real world scenario. Furthermore, to make EvePPR solve the dynamic knowledge graph alignment task more effectively, we propose to encode the structural knowledge into EvePPR by crafting its stochastic vector, and this crafting operation can be fast updated by leveraging the temporal dependency of dynamic graphs.

Our contributions can be summarized as follows.

- We propose a personalized PageRank tracking method, named EvePPR, which is able to track the exact solution at each timestamp in a fully dynamic setting, i.e., each component in the standard personalized PageRank equation can evolve.
- For the proposed EvePPR, we theoretically and empirically prove the tracking accuracy and the time complexity.
- We show the necessity of the fully dynamic setting by adapting EvePPR to the dynamic knowledge graph alignment task, which allows topological structures, node attributes, and edge attributes of two knowledge graphs ti change simultaneously.
- We design extensive experiments to show the effectiveness and efficiency of the proposed EvePPR in different tasks.

The rest of the paper is organized as follows. In Section 2, we illustrate the problem setting and background knowledge, paving the way for proposing our EvePPR. In Section 3, we formally propose our personalized PageRank tracking method named EvePPR, which is proved to have computational efficiency to track the exact personalized PageRank vector at each timestamp in the fully dynamic setting. To show the necessity of dynamic setting, in Section 4, we adapt EvePPR to solve the dynamic knowledge graph alignment task, where the fully dynamic setting allows more complex evolving scenarios. The corresponding experiments are executed in Section 5, to show the effectiveness and efficiency outperformance of EvePPR. Before we conclude the paper in Section 7, we discuss related work in Section 6.

2 PROBLEM DEFINITION

We use lowercase letters (e.g., α) for scalars, bold lowercase letters for column vectors (e.g., $\boldsymbol{v}$), bold capital letters for matrices (e.g., $\boldsymbol{P}$), parenthesized superscript to denote the temporal or iteration index (e.g., $\boldsymbol{P}^{(t)}$), and unparenthesized superscript to denote the power (e.g., $\boldsymbol{P}^k$). We use graph and network interchangeably. The notation for proposing our EvePPR is summarized in Table 1.

Personalized PageRank. The standard personalized PageRank vector is expressed as follows.

$$\boldsymbol{v} = \alpha \boldsymbol{P}\boldsymbol{v} + (1-\alpha)\boldsymbol{h} \tag{1}$$

where $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is the transition matrix, which is computed as $\boldsymbol{D}^{-1}\boldsymbol{A}$. And $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ are adjacency matrix and degree matrix of the given graph $\mathcal{G}$ with n nodes. $\boldsymbol{h}$ is the stochastic vector (i.e., teleport vector or personalized vector) of random walks, α is the dumping constant factor, and $\boldsymbol{v}$ is the stationary distribution of random walks, i.e., personalized PageRank vector. One way of solving the personalized PageRank is through the power iteration, which iteratively solves $\boldsymbol{v}^{(i+1)}$ as below, until the difference between $\boldsymbol{v}^{(i+1)}$ and $\boldsymbol{v}^{(i)}$ is smaller than a constant tolerance.

$$\boldsymbol{v}^{(i+1)} \leftarrow \alpha \boldsymbol{P}\boldsymbol{v}^{(i)} + (1-\alpha)\boldsymbol{h} \quad /^*\text{power iteration}^*/ \tag{2}$$

Table 1: Table of Notation

Symbol	Definition and Description
$\mathcal{G}$	the underlying graph structure of random walks
$\boldsymbol{P}$	the transition matrix of random walks on graph $\mathcal{G}$
$\boldsymbol{h}$	the stochastic vector (or teleport vector) of random walks
α	damping factor of random walks
$\boldsymbol{v}$	personalized PageRank vector
$\boldsymbol{e}_i$	one-hot vector with the i-th entry equals to 1
$\boldsymbol{q}_i$	single-source PageRank vector correspond to seed vector $\boldsymbol{e}_i$
$\boldsymbol{W}_i$	transition matrix used when calculating $\boldsymbol{q}_i$
ϵ	tolerance hyperparameter
$\otimes$	Kronecker product
$\odot$	Hadamard product

The computation of the power iteration method is costly, whose time complexity is $O(n^3)$ in the worst case.

Source nodes of personalized PageRank. Source nodes (i.e., seed nodes) are non-zero entries in the stochastic vector. If the stochastic vector has only one non-zero entry, we call the PageRank "single-source". If the stochastic vector has multiple non-zero entries, we call the PageRank "having multiple sources".

Fully dynamic setting of personalized PageRank. In our paper, we focus on the fully dynamic setting for personalized PageRank, i.e., each component is dependent on time, as follows.

$$\boldsymbol{v}^{(t)} = \alpha \boldsymbol{P}^{(t)}\boldsymbol{v}^{(t)} + (1-\alpha)\boldsymbol{h}^{(t)} \tag{3}$$

The transition matrix of $\boldsymbol{P}^{(t)}$ is dependent on time because the underlying graph structure is evolving, which can be represented as $\{\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \ldots, \mathcal{G}^{(T)}\}$, $t \in \{1, 2, \ldots, T\}$. The stochastic vector $h^{(t)}$ is dependent on time because we allow end-users to change the interested seed nodes along with the evolving underlying structures. The varying $h^{(t)}$ is untouched by current PageRank tracking methods [10, 24, 25, 27, 42] but has a huge potential to serve real-world downstream tasks such as dynamic knowledge graph alignment. The details are shown in Section 4. With varying $\boldsymbol{P}^{(t)}$ and $h^{(t)}$, we aim to track the exact solution $\boldsymbol{v}^{(t)}$ at each timestamp efficiently instead of solving it from scratch like the power iteration.

For adding and deleting nodes, without loss of generality, we can view them as existing dangling nodes in the previous and future graph structures like [10, 33]. Then we can denote $\boldsymbol{P}^{(t)} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{h}^{(t)} \in \mathbb{R}^{n\times 1}$ for avoiding the dimension inconsistency across timestamps. Note that, we do not limit $\boldsymbol{h}^{(t)}$ to be a one-hot vector, which means the proposed EvePPR can track the multi-source personalized PageRank vector.

Problem. *Personalized PageRank in the Fully Dynamic Setting*

Input: *(i) a sequence of evolving graph structures, represented by transition matrices* $\{\boldsymbol{P}^{(1)}, \boldsymbol{P}^{(2)}, \ldots, \boldsymbol{P}^{(T)}\}$*, (ii) a sequence of varying end-user interests, represented by stochastic vectors* $\{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(T)}\}$*, (iii) constant tolerance* ϵ*.*

Output: *a sequence of personalized PageRank vectors* $\boldsymbol{v}^{(t)}$ *whose error bound is not greater than* ϵ*, for* $t \in \{1, 2, \ldots, T\}$*.*

3 PROPOSED EVEPPR

In this section, we start from introducing the preliminary of tracking personalized PageRank vector in a partially dynamic setting, which paves the way for tracking the personalized PageRank in the fully dynamic setting. Then, we introduce our solutions in the fully dynamic setting, EvePPR and its fast approximation EvePPR-APP.

3.1 Preliminary

Here, we first introduce how to track exact personalized PageRank vector in a partially dynamic setting, i.e., the transition matrix $\boldsymbol{P}^{(t)}$ is evolving, but the stochastic vector $\boldsymbol{h}^{(t)} = \boldsymbol{h}$ is fixed. Targeting this partially dynamic setting, many personalized PageRank tracking methods are proposed [25, 42, 44]. To kick off our fully dynamic tracking method, we first introduce one of these methods, i.e., Offset Score Propagation (OSP) [42], which can track the exact personalized PageRank in this partially dynamic setting.

In the partially dynamic setting, when the graph structure changes from $\mathcal{G}^{(t)}$ to $\mathcal{G}^{(t+1)}$, the PageRank vector $\boldsymbol{v}^{(t)}$ becomes outdated. To get $\boldsymbol{v}^{(t+1)}$, we first need to update the transition matrix from $\boldsymbol{P}^{(t)}$ to $\boldsymbol{P}^{(t+1)}$ by updating adjacency matrix $\boldsymbol{A}^{(t)}$ with newly inserted and deleted edges. With $\boldsymbol{P}^{(t+1)}$, the core idea of OSP [42] is to push out the previous probability distribution score from the changed part to the residual part of the graph [42, 43], and then add the pushed out distribution back to the previous stationary distribution $\boldsymbol{v}^{(t)}$ in order to finally obtain the new stationary distribution $\boldsymbol{v}^{(t+1)}$. The pushed out distribution score is initially stored in the offset vector $\boldsymbol{q}_{\text{offset}}$, and then it will spread over the new graph structure. The detailed tracking process can be described in Algorithm 1.

Theorem 3.1 (Exactness of OSP in Partially Dynamic Setting). *In the partially dynamic setting, OSP($\boldsymbol{v}^{(t-1)}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon$) could output the personalized PageRank vector $\boldsymbol{v}^{(t)}$ satisfying $\boldsymbol{v}^{(t)} = \alpha \boldsymbol{P}^{(t)} \boldsymbol{v}^{(t)} + (1-\alpha)\boldsymbol{h}$. (Proof in Appendix A.1)*

3.2 EvePPR for the Fully Dynamic Setting

In the fully dynamic setting, the major difference from the partially dynamic setting is that we allow the stochastic vector to evolve, i.e., $\boldsymbol{h}^{(t)} \neq \boldsymbol{h}^{(t+1)}$, to reflect the varying interests from end-users when the latent graph structure updates. However, each entry in the stochastic vector is a source node for random walks with restart [32], and we do not limit $\boldsymbol{h}^{(t)}$ to be a one-hot vector. How to track the multi-source personalized PageRank is complex, and how it relates to multiple single-source tracking is not clear until now.

Hence, we propose EvePPR to track the personalized PageRank vector in the fully dynamic setting and allow $\boldsymbol{h}^{(t)}$ to be a multi-source stochastic vector. For EvePPR, we prove that tracking multi-source personalized PageRank can be decomposed into tracking multiple single-source personalized PageRank vectors, by viewing each single-source tracking as an intermediate step of random walks. To be specific, for each single source node i, we store a personalized PageRank vector, i.e., $\boldsymbol{q}_i$, which is initially computed from the first transition matrix $\boldsymbol{P}^{(t=0)}$. Then, given a timestamp t when the single source node i changes, we retrieve $\boldsymbol{q}_i$ and update it (e.g., by OSP), then we involve the updated $\boldsymbol{q}_i$ to the tracking process of $\boldsymbol{v}^{(t)}$. The detailed operations are summarized in Algorithm 2

The Algorithm 2 is easy to understand and has two sequential parts, i.e., pre-computing phase and tracking phase. In Step 2–5, at time $t = 0$, we initialize a dictionary $\mathcal{M}$ for storing the current transition matrix $\boldsymbol{P}^{(t=0)}$ for each node $i \in \{1, 2, \ldots, n\}$. With which transition matrix, the single-source personalized PageRank $\boldsymbol{q}_i$ is computed by the power iteration method by setting each node i as the seed node with the one-hot vector $\boldsymbol{e}_i$ (i.e., the i-th entry equals to 1, others equal to 0). In the future, the transition matrix stored in $\mathcal{M}$ will be updated if the corresponding seed node is updated. In Step 8, we first call OSP to compute a intermediate vector $\boldsymbol{v}_{\text{mid}}$ for time t, which is tracked essentially based on the outdated stochastic vector $\boldsymbol{h}^{(t-1)}$, because OSP is not able to track personalized PageRank vector when $\boldsymbol{h}$ evolves. After that, when we detect the evolving seed nodes in Step 9, we can then track multiple single-source PageRank vectors in Step 12, by retrieving its last active transition matrix in $\mathcal{M}$ and updating it in Step 13. Then, in Step 14, we add each single-source tracked vector to $\boldsymbol{v}_{\text{mid}}$, and finally obtain $\boldsymbol{v}^{(t)}$.

Algorithm 1 Offset Score Propagation (OSP)

Input:
previous personalized PageRank vector $\boldsymbol{v}^{(t-1)}$, previous transition matrix $\boldsymbol{P}^{(t-1)}$, new transition matrix $\boldsymbol{P}^{(t)}$, damping factor α, error tolerance ϵ

Output:
updated personalized PageRank vector $\boldsymbol{v}^{(t)}$

1: Obtain offset vector $\boldsymbol{q}_{\text{offset}} = \alpha(\boldsymbol{P}^{(t)} - \boldsymbol{P}^{(t-1)})\boldsymbol{v}^{(t-1)}$
2: Set $i = 0$, $\boldsymbol{v}_{\text{offset}} = \boldsymbol{q}_{\text{offset}}$, $\boldsymbol{x}^{(i)}_{\text{offset}} = \boldsymbol{q}_{\text{offset}}$
3: **for** $i = 1$; $\|\boldsymbol{x}^{(i-1)}_{\text{offset}}\|_1 > \epsilon$; i++ **do**
4: $\quad \boldsymbol{x}^{(i)}_{\text{offset}} = \alpha \boldsymbol{P}^{(t)} \boldsymbol{x}^{(i-1)}_{\text{offset}}$
5: $\quad \boldsymbol{v}_{\text{offset}} += \boldsymbol{x}^{(i)}_{\text{offset}}$
6: **end for**
7: **Return:** $\boldsymbol{v}^{(t)} = \boldsymbol{v}^{(t-1)} + \boldsymbol{v}_{\text{offset}}$

Algorithm 2 EvePPR

Input: a sequence of transition matrices $\{\boldsymbol{P}^{(0)}, \boldsymbol{P}^{(1)}, \ldots, \boldsymbol{P}^{(T)}\}$, a sequence of stochastic vectors $\{\boldsymbol{h}^{(0)}, \boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(T)}\}$, damping factor α; error tolerance ϵ

Output: a sequence of personalized PageRank vectors $\{\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(T)}\}$

/* Pre-computing Phase */
1: Compute $\boldsymbol{v}^{(0)} = PowerIteration(\boldsymbol{P}^{(0)}, \boldsymbol{h}^{(0)}, \alpha, \epsilon)$
2: Construct dictionary $\mathcal{M}$ with n keys $\{1, 2, ..., n\}$
3: **for** iteration $i = 1$; $i \leq n$; i++ **do**
4: $\quad \boldsymbol{q}_i = PowerIteration(\boldsymbol{P}^{(0)}, \boldsymbol{e}_i, \alpha, \epsilon)$
5: $\quad$ Set $\mathcal{M}[i] = \boldsymbol{P}^{(0)}$
6: **end for**
/* Tracking Phase */
7: **for** timestamp $t = 1$; $t \leq T$; t++ **do**
8: $\quad \boldsymbol{v}_{\text{mid}} = OSP(\boldsymbol{v}^{(t-1)}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon)$
9: $\quad \Delta \boldsymbol{h} = \boldsymbol{h}^{(t)} - \boldsymbol{h}^{(t-1)}$
10: $\quad$ **for** iteration $i = 1$; $i \leq n$; i++ **do**
11: $\quad\quad$ **if** $\Delta \mathbf{h}(i) \neq 0$ **then**
12: $\quad\quad\quad \boldsymbol{q}_i = OSP(\boldsymbol{q}_i, \mathcal{M}[i], \boldsymbol{P}^{(t)}, \alpha, \epsilon)$
13: $\quad\quad\quad \mathcal{M}[i] = \boldsymbol{P}^{(t)}$
14: $\quad\quad\quad \boldsymbol{v}_{\text{mid}} += \Delta \boldsymbol{h}(i) \boldsymbol{q}_i$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: $\quad \boldsymbol{v}^{(t)} = \boldsymbol{v}_{\text{mid}}$
18: **end for**
19: **Return:** $\boldsymbol{v}^{(t)}$ for $t \in \{0, 1, \ldots, T\}$

The theoretical analysis of exactness, time complexity, and error bound of the proposed EvePPR are expressed as follows.

Theorem 3.2 (Exactness of EvePPR). *At each timestamp t of the fully dynamic setting, EvePPR outputs the exact personalized PageRank vector $\boldsymbol{v}^{(t)}$ that satisfies $\boldsymbol{v}^{(t)} = \alpha \boldsymbol{P}^{(t)}\boldsymbol{v}^{(t)} + (1-\alpha)\boldsymbol{h}^{(t)}$.*

Proof. Firstly, according to the power iteration for solving $\boldsymbol{q}_i$ in the pre-computing phase,

$$\boldsymbol{q}_i = \alpha \boldsymbol{P}^{(0)}\boldsymbol{q}_i + (1-\alpha)\boldsymbol{e}_i \iff \boldsymbol{q}_i = (1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(0)})^k \boldsymbol{e}_i \quad (4)$$

Then, in the tracking phase for timestamp t, $\boldsymbol{v}_{\text{mid}}$ is first figured out as below,

$$\boldsymbol{v}_{\text{mid}} = OSP(\mathbf{v}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon) = (1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k \boldsymbol{h}^{(t-1)} \quad (5)$$

and when a certain $\boldsymbol{q}_i$ is called, it will be updated as follows.

$$\boldsymbol{q}_i = OSP(\boldsymbol{q}_i, \mathcal{M}[i], \boldsymbol{P}^{(t)}, \alpha, \epsilon) = (1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k \boldsymbol{e}_i \quad (6)$$

Assume l seed nodes evolve, then the one-hot decomposition of $\Delta\boldsymbol{h} = \boldsymbol{h}^{(t)} - \boldsymbol{h}^{(t-1)}$ is rewritten as follows.

$$\Delta\boldsymbol{h} = \sum_{j=1}^{l}\Delta\boldsymbol{h}(j)\boldsymbol{e}_j \quad (7)$$

Finally, $\boldsymbol{v}^{(t)}$ calculated by EvePPR is

$$\begin{aligned}
\boldsymbol{v}^{(t)} &= \boldsymbol{v}_{\text{mid}} + \sum_{j=1}^{l}\Delta\boldsymbol{h}(j)\boldsymbol{q}_j \\
&= (1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k \boldsymbol{h}^{(t-1)} + \sum_{j=1}^{l}\Delta\boldsymbol{h}(j)((1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k \boldsymbol{e}_j) \\
&= ((1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k)(\boldsymbol{h}^{(t-1)} + \sum_{j=1}^{l}\Delta\boldsymbol{h}(j)\boldsymbol{e}_j) \\
&= (1-\alpha)\sum_{k=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^k \boldsymbol{h}^{(t)} \\
&= (1-\alpha)(\boldsymbol{I} - \alpha \boldsymbol{P}^{(t)})^{-1}\boldsymbol{h}^{(t)}
\end{aligned} \quad (8)$$

Hence, $\boldsymbol{v}^{(t)} = \alpha \boldsymbol{P}^{(t)}\boldsymbol{v}^{(t)} + (1-\alpha)\boldsymbol{h}^{(t)}$ gets proved for EvePPR. □

Theorem 3.3 (Time Complexity of EvePPR). *At timestamp t of the fully dynamic setting, EvePPR outputting personalized PageRank $\boldsymbol{v}^{(t)}$ costs $O(m(l+1)log_{\alpha}\epsilon)$, where m is the number of non-zero entries of $\boldsymbol{P}^{(t)}$, l is the number of non-zero entries of $\boldsymbol{h}^{(t)}$, ϵ is the tolerance.*

Proof. In a single OSP calculation, when $\boldsymbol{P}^{(t-1)}$ is updated to $\boldsymbol{P}^{(t)}$, $\boldsymbol{x}^{(i)}_{\text{offset}}$ will be computed iteratively through $\boldsymbol{x}^{(i)}_{\text{offset}} = \alpha \boldsymbol{P}^{(t)}\boldsymbol{x}^{(i-1)}_{\text{offset}}$, which takes $O(m)$, where m is the number of non-zero entries in $\boldsymbol{P}^{(t)}$. OSP stops when the error tolerance is achieved, i.e., $\|\boldsymbol{x}^{(i)}_{\text{offset}}\|_1 = \|(\alpha \boldsymbol{P}^{(t)})^i \boldsymbol{x}^{(0)}_{\text{offset}}\|_1 = \alpha^i \|(\boldsymbol{P}^{(t)})^i \boldsymbol{x}^{(0)}_{\text{offset}}\|_1 \le \epsilon$. In PageRank setting, we have $\|\boldsymbol{P}^{(t)}\|_1 \le 1$, then $i = \log_{\alpha}\frac{\epsilon}{\|\boldsymbol{x}^{(0)}_{\text{offset}}\|_1}$ suffices. $\|\boldsymbol{x}^{(0)}_{\text{offset}}\|_1 = \|\alpha(\boldsymbol{P}^{(t)} - \boldsymbol{P}^{(t-1)})\boldsymbol{v}^{(t)}\|_1 \le 2\alpha\|\boldsymbol{v}^{(t)}\|_1$ Therefore, the number of iterations in one OSP is in $O(\log_{\alpha}\epsilon)$, the time complexity of a single OSP is in $O(m\log_{\alpha}\epsilon)$. EvePPR calls OSP $l+1$ times, where l is the number of non-zero entries in $\Delta\boldsymbol{h} = \boldsymbol{h}^{(t)} - \boldsymbol{h}^{(t-1)}$. Therefore, the number of iterations in one EvePPR is in $O((l+1)\log_{\alpha}\epsilon)$, and the time complexity of EvePPR is in $O(m(l+1)\log_{\alpha}\epsilon)$. □

Theorem 3.4 (Error Bound of EvePPR in Practice). *At timestamp t in the fully dynamic setting, if OSP only works k iterations, the PageRank tracking error of EvePPR is bounded by $\frac{\alpha\epsilon}{1-\alpha}(1+\|\Delta\boldsymbol{h}\|_1)$, with $\Delta\boldsymbol{h} = \boldsymbol{h}^{(t)} - \boldsymbol{h}^{(t-1)}$ and ϵ is the tolerance.*

Proof. Assume the OSP terminates at the k-th iteration, then the remaining error $O(error\ of\ OSP)$ is expressed as follows.

$$\begin{aligned}
O(error\ of\ OSP) &\le \|\sum_{i=k+1}^{\infty}\boldsymbol{x}^{(i)}_{\text{offset}}\|_1 \le \sum_{i=k+1}^{\infty}\alpha^{i-k}\|\boldsymbol{x}^{(k)}_{\text{offset}}\| \\
&\le \sum_{i=1}^{\infty}\alpha^i\epsilon = \frac{\alpha\epsilon}{1-\alpha}
\end{aligned} \quad (9)$$

Then, the error bound of EvePPR is

$$\begin{aligned}
O(error) &\le O(error\ of\ OSP) + \sum_{j=1}^{l}|\Delta\boldsymbol{h}(j)|\,O(error\ of\ OSP) \\
&= \frac{\alpha\epsilon}{1-\alpha}(1+\|\Delta\boldsymbol{h}\|_1)
\end{aligned} \quad (10)$$

where j denotes the index of non-zero entries in $\Delta\boldsymbol{h}$. □

3.3 EvePPR Approximation: EvePPR-APP

EvePPR can track the personalized PageRank vector in the fully dynamic setting with bounded effectiveness and time complexity. However, EvePPR leverages OSP to do the intermediate tracking, whose summation of matrices multiplication (i.e., Steps 4–5 in Algorithm 1) is still time-consuming in practice, because we need to call OSP multiple times for each varying seed node in each timestamp (i.e., Step 12 in Algorithm 2). Therefore, we aim to find an approximation method of EvePPR, which can avoid calling OSP for each varying seed node but still can track the multi-source personalized PageRank vector in the fully dynamic setting.

Algorithm 3 EvePPR-APP

Input: previous and current transition matrix $\boldsymbol{P}^{(t-1)}$ and $\boldsymbol{P}^{(t)}$, previous and current stochastic vector $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{h}^{(t)}$, damping factor α, error tolerance ϵ, previous PageRank vector $\boldsymbol{v}^{(t-1)}$
Output: approximated current PageRank vector $\boldsymbol{v}^{(t)}$

1: $\boldsymbol{v}_{\text{mid}} \leftarrow OSP(\boldsymbol{v}^{(t-1)}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon)$
2: Set $k = 0$, $\boldsymbol{x}^{(k)} = \boldsymbol{v}_{\text{mid}}$, $\boldsymbol{r}^{(k)} = (1-\alpha)\boldsymbol{h}^{(t)} - (\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\boldsymbol{v}_{\text{mid}}$
3: **while** $\|\boldsymbol{r}^{(k)}\|_{\infty} > \epsilon$ **do**
4: Denote the largest absolute value in $\boldsymbol{r}^{(k)}$ as $r_i^{(k)}$
5: $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + r_i^{(k)}\boldsymbol{e}_i$
6: $\boldsymbol{r}^{(k+1)} = (1-\alpha)\boldsymbol{h}^{(t)} - (\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\boldsymbol{x}^{(k+1)}$
7: $k += 1$
8: **end while**
9: **Return:** $\boldsymbol{v}^{(t)} = \boldsymbol{x}^{(k)}$

To this end, we develop EvePPR-APP that utilizes the logic of Gaussian-Southwell method [25]. To be specific, for each unstable personalized PageRank vector $\boldsymbol{v}'$, we will keep a residual vector $\boldsymbol{r}$ that records the divergence between $\boldsymbol{v}'$ and the stably distributed PageRank vector $\boldsymbol{v}$. By pushing out the largest entry of $\boldsymbol{r}$ to $\boldsymbol{v}'$

iteratively, $\boldsymbol{v}'$ approaches $\boldsymbol{v}$. In general, this push operation avoids the continuous multiplication of matrices but only needs the multiplication between the matrix and vector. Thus, the computational complexity is saved. The details are illustrated in Algorithm 3.

Theorem 3.5 (Time Complexity of EvePPR-APP). *At timestamp t of the fully dynamic setting, EvePPR-APP outputting the approximated personalized PageRank vector $\boldsymbol{v}^{(t)}$ costs $O(m\log_\alpha \epsilon + n(\frac{(1+\alpha)\alpha}{(1-\alpha)^2} + \frac{\|\Delta \boldsymbol{h}\|_1}{\epsilon}))$, where m is the number of non-zero entries of $\boldsymbol{P}^{(t)}$, $\Delta \boldsymbol{h} = \boldsymbol{h}^{(t)} - \boldsymbol{h}^{(t-1)}$, and ϵ is the tolerance.*

Proof. Calling OSP takes $O(m\log_\alpha \epsilon)$, as analysed in Theorem 3.3, where m is the number of non-zero entries in $\boldsymbol{P}^{(t)}$. Then EvePPR decreases the residual vector $\boldsymbol{r}$ for the varying stochastic vector $\boldsymbol{h}$, which can be proven as a form of Gauss-Southwell [25].

$$\begin{aligned}\boldsymbol{r}^{(k+1)} &= (1-\alpha)\boldsymbol{h}^{(t)} - (\boldsymbol{I} - \alpha \boldsymbol{P}^{(t)})\boldsymbol{x}^{(k+1)} \\ &= (1-\alpha)\boldsymbol{h}^{(t)} - (\boldsymbol{I} - \alpha \boldsymbol{P}^{(t)})(\boldsymbol{x}^{(k)} + r_i^k \boldsymbol{e}_i) \\ &= \boldsymbol{r}^{(k)} - r_i^{(k)}\boldsymbol{e}_i + \alpha r_i^k \boldsymbol{P}^{(t)}\boldsymbol{e}_i \\ &= \boldsymbol{r}^{(k)} - r_i^{(k)}\boldsymbol{e}_i + \alpha r_i^k \boldsymbol{P}_i^{(t)}\end{aligned} \tag{11}$$

where $\boldsymbol{P}_i^{(t)}$ is the i-th column of $\boldsymbol{P}^{(t)}$.

In each iteration k, the i-th entry of $\boldsymbol{r}^{(k)}$ is set to be 0 at first, and then adds $\alpha r_i^k \boldsymbol{P}_i^{(t)}$. Since we have $\|\boldsymbol{P}_i^{(t)}\|_1 \le 1$, then we have

$$\begin{aligned}\|\boldsymbol{r}^{(k)}\|_1 - \|\boldsymbol{r}^{(k+1)}\|_1 &\ge (1-\alpha\|\boldsymbol{P}_i^{(t)}\|_1) r_i^{(k)} \\ &\ge (1-\alpha)\epsilon\end{aligned} \tag{12}$$

Therefore, the while-loop in Algorithm 3 will terminate within $\frac{\|\boldsymbol{r}^{(0)}\|_1}{(1-\alpha)\epsilon}$ number of iterations.

$$\begin{aligned}\frac{\|\boldsymbol{r}^{(0)}\|_1}{(1-\alpha)\epsilon} &= \frac{\|(1-\alpha)\boldsymbol{h}^{(t)} - (\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\boldsymbol{v}_{\text{mid}}\|_1}{(1-\alpha)\epsilon} \\ &= \frac{\|(1-\alpha)(\boldsymbol{h}^{(t-1)} + \Delta\boldsymbol{h}) - (\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\boldsymbol{v}_{\text{mid}}\|_1}{(1-\alpha)\epsilon} \\ &\le \frac{\|(1-\alpha)\boldsymbol{h}^{(t-1)} - (\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\boldsymbol{v}_{\text{mid}}\|_1 + \|(1-\alpha)\Delta\boldsymbol{h}\|_1}{(1-\alpha)\epsilon}\end{aligned} \tag{13}$$

According to the error bound of OSP analysed in Theorem 3.4, $\boldsymbol{v}_{\text{mid}}$ differs from $\boldsymbol{v}^{(t)}$ for at most $\frac{\alpha\epsilon}{1-\alpha}$, then we have the bound for $\frac{\|\boldsymbol{r}^{(0)}\|_1}{(1-\alpha)\epsilon}$ as follows.

$$\begin{aligned}\frac{\|\boldsymbol{r}^{(0)}\|_1}{(1-\alpha)\epsilon} &\le \frac{\|(\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})\frac{\alpha\epsilon}{1-\alpha}\|_1 + \|(1-\alpha)\Delta\boldsymbol{h}\|_1}{(1-\alpha)\epsilon} \\ &\le \frac{(1+\alpha)\frac{\alpha}{1-\alpha}}{(1-\alpha)} + \frac{\|\Delta\boldsymbol{h}\|_1}{\epsilon} \\ &= \frac{(1+\alpha)\alpha}{(1-\alpha)^2} + \frac{\|\Delta\boldsymbol{h}\|_1}{\epsilon}\end{aligned} \tag{14}$$

For each iteration, the time complexity is $O(n)$. So the total time complexity of EvePPR-APP is below.

$$\begin{aligned}O(\text{EvePPR-APP}) &= O(m\log_\alpha \epsilon) + O(n(\frac{(1+\alpha)\alpha}{(1-\alpha)^2} + \frac{\|\Delta\boldsymbol{h}\|_1}{\epsilon})) \\ &= O(m\log_\alpha \epsilon + n(\frac{(1+\alpha)\alpha}{(1-\alpha)^2} + \frac{\|\Delta\boldsymbol{h}\|_1}{\epsilon}))\end{aligned} \tag{15}$$

□

Theorem 3.6 (Error Bound of EvePPR-APP in Practice). *At timestamp t in the fully dynamic setting, the PageRank tracking error of EvePPR-APP is bounded by $\frac{n\epsilon}{1-\alpha}$, where ϵ is the tolerance.*

Proof. At timestamp t, let $\boldsymbol{v}^{(k)}$ denote the tracked result from EvePPR-APP, $\boldsymbol{v}$ denote the exact PageRank vector, $\boldsymbol{r}^{(k)}$ and $\boldsymbol{r}$ denote their residues respectively. When EvePPR-APP terminates, $\left|r_i^{(k)}\right| \le \epsilon, \forall\, i \in \{1,\dots,n\}$. Therefore, $\|\boldsymbol{r} - \boldsymbol{r}^{(k)}\|_1 \le n\epsilon$, and we can have

$$\begin{aligned}\|\boldsymbol{v} - \boldsymbol{v}^{(k)}\|_1 &\le \|(\boldsymbol{I} - \alpha\boldsymbol{P}^{(t)})^{-1}\|_1 \, \|\boldsymbol{r} - \boldsymbol{r}^{(k)}\|_1 \\ &= \|\sum_{i=0}^{\infty}(\alpha\boldsymbol{P}^{(t)})^i\|_1 \, \|\boldsymbol{r} - \boldsymbol{r}^{(k)}\|_1 \\ &\le \sum_{i=0}^{\infty}\|(\alpha\boldsymbol{P}^{(t)})^i\|_1 \, \|\boldsymbol{r} - \boldsymbol{r}^{(k)}\|_1 \\ &\le \sum_{i=0}^{\infty}\alpha^i\|\boldsymbol{r} - \boldsymbol{r}^{(k)}\|_1 \le \sum_{i=0}^{\infty}\alpha^i n\epsilon = \frac{n\epsilon}{1-\alpha}\end{aligned} \tag{16}$$

□

Theoretically, as the cost of speeding up the computation, the one-norm error bound of EvePPR-APP is larger than that of EvePPR. EvePPR-APP shares the same error scale with the strong baseline Gauss-Southwell [25], and empirically we can assign ϵ a small scalar to reduce the side-effect brought by n.

3.4 Empirical Evaluation of EvePPR

After proving EvePPR theoretically, we now empirically verify both tracking effectiveness and efficiency with previous PageRank methods in a real-world dataset MathOverflow [26]. This widely-used network dataset is dynamics and has 24,818 nodes and 506,550 edges. We then choose OSP [42] and Gauss-Southwell [25] as two state-of-the-art baseline methods to track PageRank dynamically.

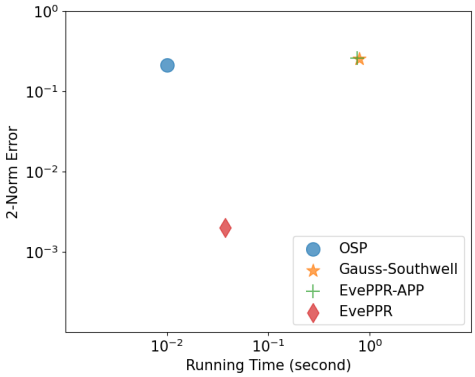


Figure 2: Tracking Error and Running Time of Different PageRank Algorithms in MathOverflow Network.

Scenario: Starting from the 100,000-th edge[1], we aggregate 20,000 edges for each timestamp. We calculate an exact single-source PageRank vector when the graph has the first 100,000 edges. Then for each following timestamp, we update the transition matrix according to the newly added edges. As for the stochastic vector, we let the seed node pass sequentially (i.e., seed node is i at t and $i+1$ at $t+1$). We evaluate the L2-norm tracking error and running time of each method on each timestamp. After tracking 5

[1]In MathOverflow, edges from approximately the 300,000-th are just identically repeating with different timestamps

timestamps, the average tracking error and running time are shown in Figure 2. The lower L2-norm error stands for the higher tracking accuracy between two PageRank vectors (i.e., the tracked PageRank v.s. the ground-truth PageRank by power iterations), and the lower running time stands for higher computational efficiency. It can be seen that EvePPR outperforms Gauss-Southwell on both accuracy and efficiency, while EvePPR-APP can achieve similar performance to Gauss-Southwell. Compared to OSP, EvePPR outperforms OSP on the accuracy, with a slight sacrifice on running time.

4 KNOWLEDGE GRAPH ALIGNMENT

Knowledge graph alignment, i.e., searching node correspondence (or similarity) across knowledge graphs, has a wide application on web mining, social network analysis, question answering, protein-protein interaction prediction, etc [11, 23, 36, 37, 46, 50]. To be specific, given the adjacency matrices, node attributes, and edge attributes of two knowledge graphs, retrieving one-to-one node pair between two knowledge graphs is the main goal [46].

4.1 Static Knowledge Graph Alignment

Static Alignment Solution using Personalized PageRank. Given two graphs $\mathcal{G}_1 = \{\boldsymbol{A}_1, \boldsymbol{N}_1, \boldsymbol{E}_1\}$ and $\mathcal{G}_2 = \{\boldsymbol{A}_2, \boldsymbol{N}_2, \boldsymbol{E}_2\}$, where $\boldsymbol{A}$, $\boldsymbol{N}$, $\boldsymbol{E}$ stands for adjacency matrix, node attribute matrix (i.e., $\boldsymbol{N}(a, a)$ is attribute of node a), and edge attribute matrix (i.e., $\boldsymbol{E}(a, b)$ is attribute of edge ab), respectively. n_1 (or n_2) is the number of nodes in graph $\mathcal{G}_1$ (or $\mathcal{G}_2$). The similarity-based alignment solution $\boldsymbol{s} \in \mathbb{R}^{n_1 n_2}$ [46] can be rewritten in the following form.

$$\boldsymbol{s} = \alpha \widetilde{\boldsymbol{W}} \boldsymbol{s} + (1 - \alpha)\boldsymbol{h} \tag{17}$$

where $\widetilde{\boldsymbol{W}} = \boldsymbol{D}^{-\frac{1}{2}} \boldsymbol{N}(\boldsymbol{E} \odot (\boldsymbol{A_1} \otimes \boldsymbol{A_2}))\boldsymbol{N}\boldsymbol{D}^{-\frac{1}{2}}$. Suppose we have K node attributes and L edge attributes in total (i.e., $\boldsymbol{N}_i^k(a, a) = 1$ denotes that node a in $\mathcal{G}_i$ has the attribute k, and $\boldsymbol{E}_i^l(a, b) = 1$ denotes edge ab in $\mathcal{G}_i$ has attribute l), then the corresponding matrices are $\boldsymbol{N} = \sum_{k=1}^{K} \boldsymbol{N}_1^k \otimes \boldsymbol{N}_2^k$ and $\boldsymbol{E} = \sum_{l=1}^{L} \boldsymbol{E}_1^l \otimes \boldsymbol{E}_2^l$. Matrix $\boldsymbol{D} = diag(\sum_{k,k'=1}^{K} \sum_{l=1}^{L} (\boldsymbol{N}_1^k (\boldsymbol{E}_1^l \odot \boldsymbol{A}_1) \boldsymbol{N}_1^{k'} \boldsymbol{1}) \otimes (\boldsymbol{N}_2^k (\boldsymbol{E}_2^l \odot \boldsymbol{A}_2) \boldsymbol{N}_2^{k'} \boldsymbol{1}))$ is the diagonal degree matrix of $\boldsymbol{W} = \boldsymbol{N}(\boldsymbol{E} \odot (\boldsymbol{A}_1 \otimes \boldsymbol{A}_2))\boldsymbol{N}$.

Eq. 17 is in the same form of Eq. 1, if we concatenate two knowledge graphs together. Each entry of $\boldsymbol{s} \in \mathbb{R}^{n_1 n_2}$ represents the similarity score of a node pair across the two graphs. Then we apply a heuristic greedy match algorithm [17] to obtain the final alignment based on the ranking, i.e., similarity vector $\boldsymbol{s}$.

4.2 Fully Dynamic Setting and Solution

Evolving Elements. In real world, one challenge on knowledge graph alignment is that, the graphs are changing from time to time, but recomputing the alignment solution from scratch is time consuming. There are many kinds of changes that can occur, making the graph alignment task even more complex. For instance, in the LinkedIn knowledge graph, a user can start following someone (edge insertion), change his or her occupation (node attribute change), become CEO of a company from employee (edge attribute change), or change his or her username (prior knowledge change). A good prior knowledge (or pre-knowledge) about pre-aligned anchors across knowledge graphs is important for the alignment task, because it reduces the alignment search space for improbable nodes and strengthens the alignment between the nodes that seem to be from the same entity. For dynamic networks, the pre-knowledge is important to be dynamic as well, since the networks are able to change so many times that any starting static pre-knowledge will become distorted.

Algorithm 4 Construct Prior Knowledge

Input: the candidate set information of all the nodes in $\mathcal{G}_1$
Output: the prior knowledge vector $\boldsymbol{h}$ for graph alignment

```
set h = 0
for i = 0, i < n1; i++ do
    weight = 1 / (n1 |candidate(i)|)
    index = i * n2
    for j = 0, j < n2, j++ do
        if j ∈ candidate(i) then
            h(index + j) = weight
        end if
    end for
end for
Return: h
```

Prior Knowledge Encoding and Tracking. We use filtering methods to dynamically update the pre-knowledge encoded in the stochastic vector $\boldsymbol{h}^{(t)}$. We propose two specific filters to update the pre-knowledge after any changes mentioned above happen in the two knowledge graphs. The purpose of such filtering is to assign higher prior weights to the pairs of nodes (that seem similar to those that seem not, based on direct information given by graphs) to reduce the search space. We set *candidate* to record the possible candidates of alignment. For example, $candidate(a)$ returns the list of all possible nodes in $\mathcal{G}_2$ that are alignment candidates of node a in $\mathcal{G}_1$. At the initial timestamp, we set $candidate(a)$ to be all nodes in $\mathcal{G}_2$ for any node a in $\mathcal{G}_1$, then use the following two existing filters **sequentially** to renew *candidate*:

- **Node Attribute Filter:** This filter comes from the intuition that, if two nodes, say node a in graph $\mathcal{G}_1$ and node x in graph $\mathcal{G}_2$ are similar, they must at least have the same node attributes. In other words, $x \notin candidate(a) \Leftarrow \boldsymbol{N}_1(a, a) \neq \boldsymbol{N}_2(x, x)$. An example of this would be, the same user's LinkedIn account and Facebook account should show the same occupation.
- **One-hop Filter:** This filter comes from the intuition that, if two nodes are similar, they should have similar neighbors. We measure this using the (neighbor attribute, connecting edge attribute) pairs. For node a in $\mathcal{G}_1$ and node x in $\mathcal{G}_2$, the trigger condition of filtering x out of $candidate(a)$ is, if a has a neighbor $b \in neighbor(a)$ with an edge ab between them, but x does not have such a neighbor y in $\mathcal{G}_2$ that satisfies: the attribute of y is the same as b, and the attribute of edge xy is the same as edge ab. In other words, $x \notin candidate(a) \Leftarrow \exists b \in \mathcal{G}_1$ s.t. $\forall y \in \mathcal{G}_2, (\boldsymbol{N}_1(b, b), \boldsymbol{E}_1(a, b)) \neq (\boldsymbol{N}_2(y, y), \boldsymbol{E}_2(x, y))$.

An advantage of such a design is, *candidate* can be quickly updated when an edge is inserted/deleted, a node attribute is changed, or an edge attribute is changed. Due to the limitation of space, we discuss fast updating of *candidate* in Appendix D.

After getting the correct *candidate* for each timestamp, we distribute the alignment probability uniformly among the candidates of a node. Algorithm 4 shows how to compute the prior knowledge from *candidate*, assuming $\mathcal{G}_1$ and $\mathcal{G}_2$ have n_1 and n_2 nodes respectively and the node index are from 0 to $n_1 - 1$ or $n_2 - 1$.

5 EXPERIMENTS

5.1 Datasets

The real-world temporal networks (or graphs) [2] are summarized in Table 2, which are used to compare the effectiveness and efficiency of the proposed EvePPR and EvePPR-APP with other state-of-the-art baseline methods. We employ three real-world temporal networks after necessary preprocessing. MovieLens-1M [20] is a bipartite network containing one million movie ratings, where an edge between a user and a movie shows which level the user rates the movie. Bitcoin Alpha [19, 20] is a user-user trust/distrust network from the Bitcoin Alpha platform, where each edge is labeled from -10 to +10, indicating the trust level between users. WikiLens [5, 20] is also a bipartite rating network, where an edge represents a rating. To demonstrate the effectiveness of EvePPR and EvePPR-APP on node attribute changes and edge attribute changes, we extract synthetic temporal networks with such changes on each of the three real-world datasets mentioned.

Table 2: Statistics of Real-World Graphs

Graphs	Format	|V|	|E|	Time Span
MovieLens-1M	Bipartite	9,746	1,000,209	35 months
Bitcoin Alpha	Unipartite	3,783	24,186	64 months
WikiLens	Bipartite	5,437	26,937	46 months

5.2 Baselines

We choose several state-of-the-art methods for graph alignment as the baselines of our experiments. We compare static attributed network alignment algorithm (FINAL [46]), dynamic attributed subgraph matching algorithm based on eigenvalue decomposition (FIRST [4]), dynamic knowledge graph alignment based on graph neural networks (DINGAL [36]), dynamic random walk with restart based on offset score propagation (OSP [42]), together with our EvePPR and EvePPR-APP. Moreover, we provide two self-ablation comparisons to validate the effectiveness of our methods and the necessity of allowing stochastic vector to change in Appendix C.

5.3 Experimental Setting

Before doing the experiments, we notice the following issues and remedy by preprocessing and extracting. (1) MovieLens-1M and WikiLens have only 2 node attributes: user and ratee. Changing a node from a user to a ratee or vice versa will destroy the bipartite structure of the network. As for Bitcoin Alpha, it has only one node attribute, hence the node attribute change is impossible to occur. (2) There is only one network in each dataset, but we need two networks to do graph alignment.

Preprocessing. We process all time-evolving networks as undirected. To assign the node attribute, we further classify the nodes in each dataset based on their degrees. Additionally, for the WikiLens dataset, whose edge label can be an integer plus a half, we multiply all the edge labels by two to avoid non-integer edge attribute.

Extract the query graph. For each dataset, we extract an amount of nodes and the edges between them, i.e., a temporal subgraph of the original temporal network, to form the query network. The size of the extracted query graph of each dataset can be found in Table 3. Since we are actually tracking similarities between node pairs, the scale of personalized PageRank tracking is the multiplication of numbers of nodes in the original graph and the query graph, i.e., 877,140, 378,300, 815,550 for MovieLens-1M, Bitcoin Alpha, WikiLens respectively.

Table 3: Statistics of Extracted Query Graphs

Extracted Subgraphs	Format	|V|	|E|
MovieLens-1M	Bipartite	90	375
Bitcoin Alpha	Unipartite	100	423
WikiLens	Bipartite	150	553

Knowledge Graph Alignment Evaluation Metric. The network alignment task is, for a certain dataset, to find the best alignment of the extracted query graph in the original network. The **alignment accuracy** is defined to be the portion of nodes in the query graph whose top k alignment of its ranking list (i.e., a list consisting of nodes in the original graph, sorted by the similarity) contains the ground-truth [46]. In experiments, we set $k = 1$. Also, we set three dynamic network alignment scenarios as follows. In each scenario, we allow the query graph and the original graph to evolve simultaneously.

- Edge Insertion [3]: For MovieLens-1M and Bitcoin Alpha datasets, we take the first-occurred 90% of edges as the initial network to start tracking. For WikiLens, this ratio is 20%. When an edge is inserted in the original network and the insertion position is in the scope of the extracted query network, we also add that edge into the query network.
- Node Attribute Change: For each dataset, we randomly extract a sequence of nodes and change their attributes. We take the last timestamp networks in the Edge Insertion scenario as the starting networks, then apply node attribute changes with other properties (i.e., graph structures, edge attributes) unchanged.
- Edge Attribute Change: For each dataset, we randomly extract a sequence of edges and change their attributes. Same to the Node Attribute Change scenario, we take the last time networks in Edge Insertion scenario as the starting networks. If an edge to be changed is in both original and query networks, we change the attribute of that edge in both networks.

When a change occurs in the extracted query network, we mark a timestamp and activate the tracking methods. The numbers of timestamps for the three datasets in the edge insertion scenario are 187, 39, 33, respectively. For each timestamp, we first update the transition matrix and stochastic (prior knowledge) vector, and then apply the tracking methods. All the experiments run on a single Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz [4].

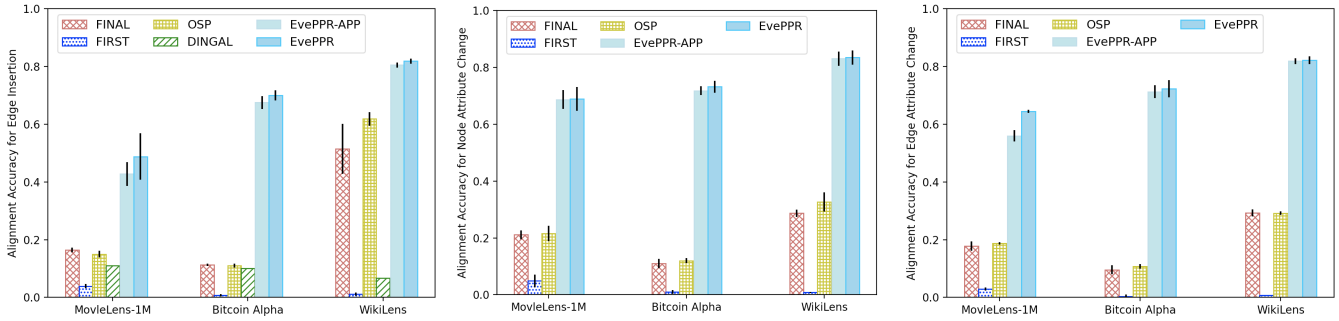
5.4 Effectiveness Comparison

We evaluate the performance of EvePPR and EvePPR-APP by comparing their alignment accuracy with other baselines. We uniformly choose some timestamps and calculate the mean and standard deviation of the accuracy at those timestamps. The experiment results are illustrated in Figure 3. First, both EvePPR and EvePPR-APP outperform the existing baseline methods on accuracy in all scenarios. Second, EvePPR-APP can achieve closely but not as accurate

[2] In experiments, we call "network" and "graph" interchangeably.

[3] Moreover, an edge-deletion scenario is illustrated in Appendix B plus graph alignment baseline algorithms.

[4] https://github.com/DongqiFu/EvePPR



(a) Edge Insertion (b) Node Attribute Change (c) Edge Attribute Change

Figure 3: Alignment Accuracy under Three Dynamic Scenarios

as EvePPR, validating that EvePPR can track exact personalized PageRank vector when the transition matrix and stochastic vector change simultaneously. In the following section, we will compare the efficiency of methods and show that EvePPR-APP is a good trade-off between accuracy and efficiency. Third, the accuracy on node attribute change scenario and edge attribute change scenario is higher than edge insertion scenario. This is because the attribute change scenarios start from the final network of edge insertion scenario, and hence they have more information to use.

5.5 Efficiency Analysis

We measure the time consumption of each method computing the tracking solution in the edge insertion scenario. We record the mean and standard deviation of sampled timestamps in Table 4. EvePPR-APP and EvePPR outperform FINAL and FIRST, but is not as fast as OSP, because OSP does not take any consideration on stochastic vector change. EvePPR-APP is about 4 times slower than OSP, but it can handle more complex dynamic changes. The reason why the executing time of EvePPR has a considerable standard deviation is that, the numbers of changed entries in the stochastic (prior knowledge) vector vary largely among timestamps, and theoretically the tracking time of EvePPR is proportional to the number of changed entries.

Table 4: Efficiency Comparison (unit: seconds)

Methods	MovieLens-1M	Bitcoin Alpha	WikiLens
FINAL	41.677±0.703	11.904±0.075	20.171±1.199
FIRST	29.537±5.355	157.581±22.596	369.695±320.471
OSP	0.0212±0.002	0.0108±0.002	0.0201±0.003
EvePPR-APP	0.0737±0.013	0.0410±0.010	0.0920±0.043
EvePPR	11.9296±34.967	7.8602±12.965	0.8973±2.230

6 RELATED WORK

PageRank Tracking. The PageRank algorithm has wide applications in many domains such as web mining [30], anomaly detection [39, 41], and urban network ranking [2, 16]. Many variants of PageRank have been proposed. For example, in [31] the authors develop PageRank for temporal graphs based on temporal random walk; in [14] the authors provide a way to audit element importance based on their influence; in [35] the authors design vertex-diminished random walk for imbalanced networks. Mathematically, an exact tracking for PageRank can be done based on tracking the inverse matrix from Shermon-Morrison Lemma [28], but the computation is costly unaffordable in practice. To trade-off between accuracy and efficiency, some methods have been proposed to approximate the partial dynamic PageRank vector of temporal graph [3, 25, 42, 44]. **Graph Alignment**. Much work has been devoted to the graph alignment problem, including incomplete graph alignment [47], bipartite graph alignment [18], dynamic graph alignment [22], multiple network alignment [34, 45] and so on. Besides the methods used in our experiment, COSNET [48] considers both local and global consistency among multiple networks while aligning; SimRank [12] finds node similarities which can be used for graph alignment, via graph-theory approach, and later a fast algorithm for computing SimRank is proposed in [21]. There are also neural-network-based solutions such as [11, 36, 40]. To the best of our knowledge, we first propose personalized PageRank tracking in the fully dynamic setting, which allows the transition matrix and stochastic vector to evolve simultaneously. We demonstrate that the fully dynamic setting can contribute to the dynamic knowledge graph alignment task by simulating the more complex evolving real-world environment. According to [29], the dynamics in real-world graphs or networks are usually complex and complicated. In our experiments, based on real-world dynamic datasets, we build and simulate more dynamic scenarios like node attribute changes and edge attribute changes. In the meantime, we look forward to and would like to devote ourselves to discovering real-world datasets that support complex dynamic scenarios.

7 CONCLUSION

In this paper, we first propose the personalized PageRank tracking method in the fully dynamic setting (i.e., both transition matrix and stochastic vector evolve), named EvePPR. We also prove the tracking accuracy, time complexity of EvePPR and its approximation method EvePPR-APP. Then we find that using PageRank in the fully dynamic setting can well address the dynamic knowledge graph alignment problem, and we design extensive experiments to show the outperformance of EvePPR with baseline methods.

ACKNOWLEDGEMENT

This work is supported by National Science Foundation under Award No. IIS-1947203, IIS-2117902, and IIS-2137468. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

REFERENCES

[1] Charu C. Aggarwal and Karthik Subbian. 2014. Evolutionary Network Analysis: A Survey. *ACM Comput. Surv.* (2014).

[2] Taras Agryzkov, José Luis Oliver, Leandro Tortosa, and José-Francisco Vicent. 2012. An algorithm for ranking the nodes of an urban network based on the concept of PageRank vector. *Appl. Math. Comput.* (2012).

[3] Esteban Bautista and Matthieu Latapy. 2022. A local updating algorithm for personalized PageRank via Chebyshev polynomials. *Soc. Netw. Anal. Min.* (2022).

[4] Boxin Du, Si Zhang, Nan Cao, and Hanghang Tong. 2017. FIRST: Fast Interactive Attributed Subgraph Matching. In *KDD 2017*.

[5] Dan Frankowski, Shyong K. Lam, Shilad Sen, F. Maxwell Harper, Scott Yilek, Michael Cassano, and John Riedl. 2007. Recommenders everywhere: : the WikiLens community-maintained recommender system. In *WIKISYM 2007*.

[6] Dongqi Fu, Yikun Ban, Hanghang Tong, Ross Maciejewski, and Jingrui He. 2022. DISCO: Comprehensive and Explainable Disinformation Detection. In *CIKM 2022*.

[7] Dongqi Fu, Liri Fang, Ross Maciejewski, Vetle I. Torvik, and Jingrui He. 2022. Meta-Learned Metrics over Multi-Evolution Temporal Graphs. In *KDD 2022*.

[8] Dongqi Fu and Jingrui He. 2021. SDG: A Simplified and Dynamic Graph Neural Network. In *SIGIR 2021*.

[9] Dongqi Fu and Jingrui He. 2022. Natural and Artificial Dynamics in Graphs: Concept, Progress, and Future. *Frontiers in Big Data* (2022).

[10] Dongqi Fu, Dawei Zhou, and Jingrui He. 2020. Local Motif Clustering on Time-Evolving Graphs. In *KDD 2020*.

[11] Ji Gao, Xiao Huang, and Jundong Li. 2021. Unsupervised Graph Alignment with Wasserstein Distance Discriminator. In *KDD 2021*.

[12] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *KDD 2002*.

[13] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. 2003. Extrapolation methods for accelerating PageRank computations. In *WWW 2003*.

[14] Jian Kang, Meijia Wang, Nan Cao, Yinglong Xia, Wei Fan, and Hanghang Tong. 2018. AURORA: Auditing PageRank on Large Graphs. In *IEEE BigData 2018*.

[15] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *J. Mach. Learn. Res.* (2020).

[16] Yongyeon Kim, Hyeon-A. Kim, Chul-Ho Shin, Kyung-Hee Lee, Chi-Hwan Choi, and Wan-Sup Cho. 2015. Analysis on the Transportation Point in Cheongju City Using Pagerank Algorithm. In *BigDAS 2015*.

[17] Giorgios Kollias, Shahin Mohammadi, and Ananth Grama. 2012. Network Similarity Decomposition (NSD): A Fast and Scalable Approach to Network Alignment. *IEEE Trans. Knowl. Data Eng.* (2012).

[18] Danai Koutra, Hanghang Tong, and David M. Lubensky. 2013. BIG-ALIGN: Fast Bipartite Graph Alignment. In *ICDM 2013*.

[19] Srijan Kumar, Francesca Spezzano, V. S. Subrahmanian, and Christos Faloutsos. 2016. Edge Weight Prediction in Weighted Signed Networks. In *ICDM 2016*.

[20] Jérôme Kunegis. 2013. KONECT: the Koblenz network collection. In *WWW 2013, Companion Volume*.

[21] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. 2010. Fast computation of SimRank for static and dynamic information networks. In *EDBT 2010*.

[22] Liangyue Li, Hanghang Tong, Yanghua Xiao, and Wei Fan. 2015. *Cheetah*: Fast Graph Kernel Tracking on Dynamic Graphs. In *SDM 2015*.

[23] Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. 2021. Neural-Answering Logical Queries on Knowledge Graphs. In *KDD 2021*.

[24] Dingheng Mo and Siqiang Luo. 2021. Agenda: Robust Personalized PageRanks in Evolving Graphs. In *CIKM 2021*.

[25] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient PageRank Tracking in Evolving Networks. In *KDD 2015*.

[26] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *WSDM 2017*.

[27] Sungchan Park, Wonseok Lee, Byeongseo Choe, and Sang-Goo Lee. 2019. A Survey on Personalized PageRank Computation Algorithms. *IEEE Access* (2019).

[28] Walter W. Piegorsch and George Casella. 1990. Erratum: Inverting a Sum of Matrices. *SIAM Rev.* (1990).

[29] Giulio Rossetti and Rémy Cazabet. 2018. Community Discovery in Dynamic Networks: A Survey. *ACM Comput. Surv.* 51, 2 (2018), 35:1–35:37. https://doi.org/10.1145/3172867

[30] Rajendra Kumar Roul and Jajati Keshari Sahoo. 2021. A novel approach for ranking web documents based on query-optimized personalized pagerank. *Int. J. Data Sci. Anal.* (2021).

[31] Polina Rozenshtein and Aristides Gionis. 2016. Temporal PageRank. In *ECML-PKDD 2016*.

[32] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM 2006*.

[33] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. 2008. Proximity Tracking on Time-Evolving Bipartite Graphs. In *SDM 2008*.

[34] Vipin Vijayan and Tijana Milenkovic. 2018. Multiple Network Alignment via MultiMAGNA++. *IEEE ACM Trans. Comput. Biol. Bioinform.* (2018).

[35] Jun Wu, Jingrui He, and Yongming Liu. 2018. ImVerde: Vertex-Diminished Random Walk for Learning Imbalanced Network Representation. In *IEEE BigData 2018*.

[36] Yuchen Yan, Lihui Liu, Yikun Ban, Baoyu Jing, and Hanghang Tong. 2021. Dynamic Knowledge Graph Alignment. In *AAAI 2021*.

[37] Yuchen Yan, Si Zhang, and Hanghang Tong. 2021. BRIGHT: A Bridging Algorithm for Network Alignment. In *WWW 2021*.

[38] Yuchen Yan, Qinghai Zhou, Jinning Li, Tarek F. Abdelzaher, and Hanghang Tong. 2022. Dissecting Cross-Layer Dependency Inference on Multi-Layered Inter-Dependent Networks. In *CIKM 2022*.

[39] Zhe Yao, Philip Mark, and Michael G. Rabbat. 2012. Anomaly Detection Using Proximity Graph and PageRank Algorithm. *IEEE Trans. Inf. Forensics Secur.* (2012).

[40] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *KDD 2017*.

[41] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. 2019. Fast and Accurate Anomaly Detection in Dynamic Graphs with a Two-Pronged Approach. In *KDD 2019*.

[42] Minji Yoon, Woojeong Jin, and U Kang. 2018. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. In *WWW 2018*.

[43] Minji Yoon, Jinhong Jung, and U Kang. 2018. TPA: Fast, Scalable, and Accurate Method for Approximate Random Walk with Restart on Billion Scale Graphs. In *ICDE 2018*.

[44] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. In *KDD 2016*.

[45] Jiawei Zhang and Philip S. Yu. 2015. Multiple Anonymized Social Networks Alignment. In *ICDM 2015*.

[46] Si Zhang and Hanghang Tong. 2016. FINAL: Fast Attributed Network Alignment. In *KDD 2016*.

[47] Si Zhang, Hanghang Tong, Jie Tang, Jiejun Xu, and Wei Fan. 2020. Incomplete Network Alignment: Problem Definitions and Fast Solutions. *ACM Trans. Knowl. Discov. Data* (2020).

[48] Yutao Zhang, Jie Tang, Zhilin Yang, Jian Pei, and Philip S. Yu. 2015. COSNET: Connecting Heterogeneous Social Networks with Local and Global Consistency. In *KDD 2015*.

[49] Dawei Zhou, Si Zhang, Mehmet Yigit Yildirim, Scott Alcorn, Hanghang Tong, Hasan Davulcu, and Jingrui He. 2017. A Local Algorithm for Structure-Preserving Graph Cut. In *KDD 2017*.

[50] Qinghai Zhou, Liangyue Li, Xintao Wu, Nan Cao, Lei Ying, and Hanghang Tong. 2021. Attent: Active Attributed Network Alignment. In *WWW 2021*.

A PROOF OF THEOREM 3.1

According to [43], the personalized PageRank can be rewritten in the form of infinite summation as follows.

$$\boldsymbol{v} = \alpha \boldsymbol{P}\boldsymbol{v} + (1-\alpha)\boldsymbol{h} \iff \boldsymbol{v} = (1-\alpha)(\boldsymbol{I} - \alpha \boldsymbol{P})^{-1}\boldsymbol{h}$$

$$\iff \boldsymbol{v} = (1-\alpha)\sum_{i=0}^{\infty}(\alpha \boldsymbol{P})^i \boldsymbol{h}$$

According to [42], $OSP(\boldsymbol{v}^{(t-1)}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon)$ is as follows.

$$\begin{aligned}
&OSP(\boldsymbol{v}^{(t-1)}, \boldsymbol{P}^{(t-1)}, \boldsymbol{P}^{(t)}, \alpha, \epsilon) \\
&= \sum_{j=0}^{\infty}((\alpha \boldsymbol{P}^{(t)})^j \alpha(\boldsymbol{P}^{(t)} - \boldsymbol{P}^{(t-1)})(1-\alpha)\sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i \boldsymbol{h}) + \boldsymbol{v}^{(t-1)} \\
&= (1-\alpha)\sum_{j=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j(\alpha \boldsymbol{P}^{(t)} - \alpha \boldsymbol{P}^{(t-1)})\sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i \boldsymbol{h} + \boldsymbol{v}^{(t-1)} \\
&= (1-\alpha)(\sum_{j=1}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j \sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i - \sum_{j=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j \sum_{i=1}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i)\boldsymbol{h} + \boldsymbol{v}^{(t-1)} \\
&= (1-\alpha)(\sum_{j=1}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j(\boldsymbol{I} + \sum_{i=1}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i) - (\boldsymbol{I} + \sum_{j=1}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j)\sum_{i=1}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i)\boldsymbol{h} + \boldsymbol{v}^{(t-1)} \\
&= (1-\alpha)(\sum_{j=1}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j - \sum_{i=1}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i)\boldsymbol{h} + \boldsymbol{v}^{(t-1)} \\
&= (1-\alpha)(\sum_{j=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^j - \sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i)\boldsymbol{h} + (1-\alpha)\sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t-1)})^i \boldsymbol{h} \\
&= (1-\alpha)\sum_{i=0}^{\infty}(\alpha \boldsymbol{P}^{(t)})^i \boldsymbol{h} \\
&= (1-\alpha)(\boldsymbol{I} - \alpha \boldsymbol{P}^{(t)})^{-1}\boldsymbol{h} \\
&= \boldsymbol{v}^{(t)}
\end{aligned}$$

B EDGE DELETION EXPERIMENT

The edge deletion experiment is executed in Bitcoin Alpha dataset.

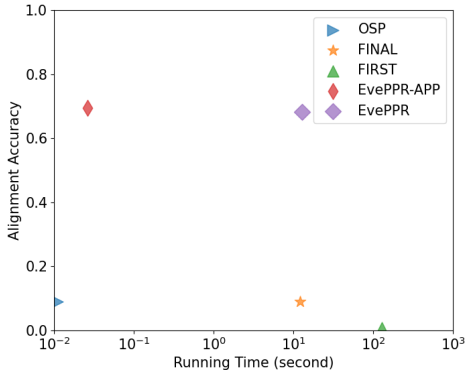
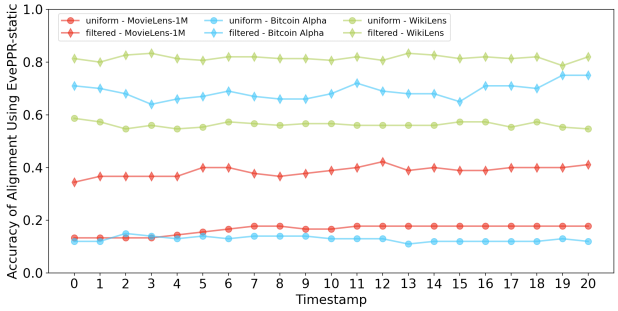


Figure 4: Alignment Accuracy and Running Time.

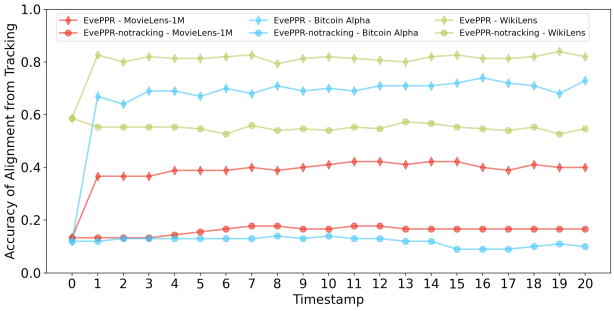
Edge Deletion: We take the last timestamp networks in Edge Insertion scenario as the starting networks. Then we delete 10% of all edges appeared in the original graph from all previous 32 timestamps. When an edge is deleted in the original network and the deletion position is in the extracted query network, we also delete that edge in the query network. Following the same experimental setting in Section 5, the performance of alignment accuracy w.r.t running time is shown in Figure 4.

C ABLATION STUDY

Here, we design two ablation studies. Ablation study I (shown in Figure 5(a)) compares performance between the given uniform prior knowledge and our filtered prior knowledge at each timestamp, which shows the effectiveness of our proposed filters. Ablation study II (shown in Figure 5(b)) compares EvePPR with EvePPR-notracking, where EvePPR-notracking removes the functionality of updating prior knowledge changes in $\boldsymbol{h}$, i.e., identical to OSP which only allows transition matrix to change.



(a) Uniform Prior Knowledge and Filtered Prior Knowledge



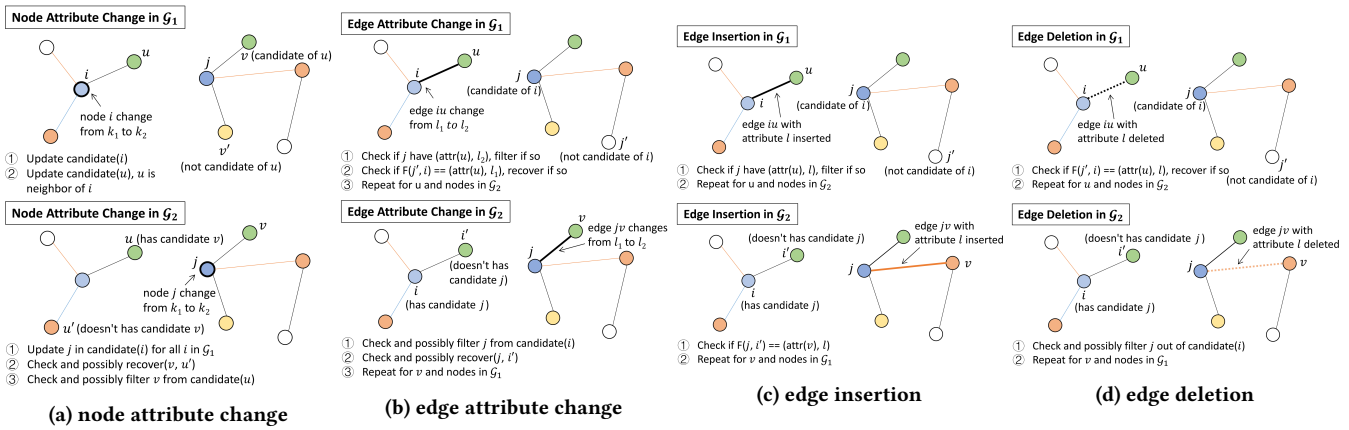
(b) Tracking Prior Knowledge and Not Tracking Prior Knowledge

Figure 5: Ablation Study

For both ablation studies, we take the initial and first 20 timestamps in edge insertion scenario as the dynamic setting. EvePPR-static computes the transition matrix and initial PageRank vector in the same way as EvePPR. Changing the prior knowledge from the filtered knowledge into a uniform distribution, an obvious drop occur on each dataset. This shows that our proposed filter can give much better prior knowledge than uniform distribution. In ablation study II, we give the uniform prior knowledge at timestamp 0, and give the filtered prior knowledge later on. It turns out that EvePPR-notracking cannot take advantage of the better input pre-knowledge from timestamp 1, but EvePPR can "recover" from the initial low accuracy very quickly.

D FAST UPDATE OF PRIOR KNOWLEDGE

In order to achieve fast update of the filtering result, we need a matrix to record the filtering information. Let $\boldsymbol{F}$ to be an $n_2 \times n_1$ matrix, where $\boldsymbol{F}(j,i)$ will be used to record which (NodeAttribute, EdgeAttribute) filtered node j in $\mathcal{G}_2$ away from $candidate(i)$, the candidate set of i in $\mathcal{G}_1$. If $j \in candidate(i)$, $\boldsymbol{F}(j,i)$ should be $none$. We first introduce the recovering methods, which is an inverse process of filtering: for $x \notin candidate(a)$, recovering method will check if x can be a candidate of a again. We configure the one-hot filter function to return the filtering pair if one_hop_filter(x,a) filters x out of $candidate(a)$, or $none$ otherwise. This ensures that $\boldsymbol{F}(x,a)$ will be updated after executing $recover(x,a)$. When a change occur,



(a) node attribute change **(b) edge attribute change** **(c) edge insertion** **(d) edge deletion**

Figure 6: Fast Update of Prior Knowledge

react according to which type the change is and where the change occurs, as described as follows.

D.1 Change of Node Attribute

Change in $\mathcal{G}_1$. When an attribute of a node in $\mathcal{G}_1$ is changed, say node i in $\mathcal{G}_1$ is changed from attribute k_1 to k_2. As an immediate consequence, we need to udpate $candidate(i)$ using node attribute filter and $recover$ function. Then, as for the one-hop neighbor condition, we need to consider the neighbors of i. Let u to be a neighbor of i, then for the nodes in $\mathcal{G}_2$ that are candidates of u, we need to check if they have the neighbor pair $(\boldsymbol{N}_1(i,i), \boldsymbol{E}_1(i,u))$.

For a node v in $\mathcal{G}_2$ that is not a candidate of u, if $(k_1, \boldsymbol{E}_1(i,u)) \neq \boldsymbol{F}(v,u)$, this means the previous filtering pair of one-hop filter is not changed and still works on v. Then v still should be filtered from $candidate(u)$. Nothing should be changed in this case. If $(k_1, \boldsymbol{E}_1(i,u)) == \boldsymbol{F}(v,u)$, then it is possible that $v \in candidate(u)$, hence it is necessary to check this by the $recover(v,u)$.

For a node v in $\mathcal{G}$ that is a candidate of u, the additional work would be to check that v has neighbor pair $(k_2, \boldsymbol{E}_1(i,u))$. if v has, then v can still be a candidate of u. But if v does not, it needs to be removed from $candidate(u)$.

Change in $\mathcal{G}_2$. When an attribute of node in $\mathcal{G}_2$ is changed, say node j in $\mathcal{G}_2$ is changed from attribute k_1 to k_2. Similar to the above analysis, we need to first update the candidate information on all the nodes in $\mathcal{G}_1$ with j. Then, we need to reconsider the neighbors of j. Take $v \in neighbor(j)$ as an example:

For a node u in $\mathcal{G}_1$ such that $v \notin candidate(u)$, check whether $(k_2, \boldsymbol{E}_2(j,v)) == \boldsymbol{F}(v,u)$. If so, then it is possible that $v \in candidate(u)$ now, hence we should check this by $recover(v,u)$.

For a node u in $\mathcal{G}_1$ such that $v \in candidate(u)$, check if "v does not have another neighbor pair $(k_1, \boldsymbol{E}_2(j,v))$ & u has neighbor pair $(k_1, \boldsymbol{E}_2(j,v))$". If so, filter v out of $candidate(u)$.

D.2 Change of Edge Attribute

A change that only refers to edge attribute will not trigger the recovery condition about the node attribute filter, but just the one-hop filter.

Change in $\mathcal{G}_1$. If edge iu in $\mathcal{G}_1$ is changed from attribute l_1 to l_2, then we only need to reconsider nodes in $\mathcal{G}_2$ with node i and node u, since other nodes in $\mathcal{G}_1$ are not related to this edge attribute change according to the definition of the filters we are using. For nodes in $\mathcal{G}_2$ that are candidates of i, check if they have neighbor pair $(\boldsymbol{N}_1(u,u), l_2)$. If any node does not have, filter it out of $candidate(i)$. For node j in $\mathcal{G}_2$ that are not candidates of i, if $(\boldsymbol{N}_1(u,u), l_1) == \boldsymbol{F}(j,i)$, then it is possible that, after this change, $j \in candidate(i)$, hence we need to check this by $recover(j,i)$. After the operations related to node i, do similar things to update $candidate(u)$.

Change in $\mathcal{G}_2$. If edge jv in $\mathcal{G}_2$ is changed from attribute l_1 to l_2, we only need to consider nodes in $\mathcal{G}_1$ with j and v, since other nodes in $\mathcal{G}_2$ will not be affected. We take node j first for example. For a node i in $\mathcal{G}_1$, if $j \in candidate(i)$, check if "j does not have another neighbor pair $(\boldsymbol{N}_2(v,v), l_1)$ & i has neighbor pair $(\boldsymbol{N}_2(v,v), l_1)$". If so, filter j out of $candidate(i)$; if not, remain $j \in candidate(i)$. If $j \notin candidate(i)$, check if $(\boldsymbol{N}_2(v,v), l_2) == \boldsymbol{F}(j,i)$. If not, keep $j \notin candidate(i)$. But if so, check by the $recover(j,i)$.

Algorithm 5 recover(x, a)

Input: a node a in $\mathcal{G}_1$, a node x in $\mathcal{G}_2$
Output: check if x should be recover back into $candidate(a)$. If so, recover x back to $candidate(a)$

```
if x ∉ candidate(a) & N1(a,a) = N2(x,x) then
    candidate(a).add(x)
    F(x,a) ← one_hop_filter(x,a)
end if
```

D.3 Insertion or Deletion of An Edge

Change in $\mathcal{G}_1$. When an edge iu with attribute l is inserted in $\mathcal{G}_1$, check the candidates of i and u. Take i as an example. For $j \in candidate(i)$, check if j has a neighbor pair $(\boldsymbol{N}_1(u,u), l)$. If not, filter j out of $candidate(i)$. When an edge iu with attribute l is deleted in $\mathcal{G}_1$, check the nodes that are not candidates of i or u. Still take i as an example. For $j \notin candidate(i)$, check if $(\boldsymbol{N}_1(u,u), l) == \boldsymbol{F}(j,i)$. If so, check if j can be a candidate of i by recover(j, i).

Change in $\mathcal{G}_2$. When an edge jv with attribute l is inserted in $\mathcal{G}_2$, check whether j or v can be candidates of nodes in $\mathcal{G}_1$. Take j as an example. For i in $\mathcal{G}_1$ such that $j \notin candidate(i)$, check if $(\boldsymbol{N}_2(v,v), l) == \boldsymbol{F}(j,i)$. If so, check if j can be a candidate of i by $recover(j,i)$. When an edge jv with attribute is deleted in $\mathcal{G}_2$, check whether j or v should be filtered out of candidates of node in $\mathcal{G}_1$. Still take j as an example. For i in $\mathcal{G}_1$ such that $j \in candidate(i)$, check if "j does not have another neighbor pair $(\boldsymbol{N}_2(v,v), l)$ & i has neighbor pair $(\boldsymbol{N}_2(v,v), l)$". If so, filter j out of $candidate(i)$.