SimFusion+: Extending SimFusion Towards Efficient Estimation on Large and Dynamic Networks

Weiren Yu[†‡], Xuemin Lin[†♯], Wenjie Zhang[†], Ying Zhang[†], Jiajin Le[♮]

[†]The University of New South Wales, Australia [♯]East China Normal University, China

[‡]NICTA, Australia [♮]Donghua University, China

{weirenyu, lxue, zhangw, yingz}@cse.unsw.edu.au lejiajin@dhu.edu.cn

ABSTRACT

SimFusion has become a captivating measure of similarity between objects in a web graph. It is iteratively distilled from the notion that "the similarity between two objects is reinforced by the similarity of their related objects". The existing SimFusion model usually exploits the *Unified Relationship Matrix* (URM) to represent latent relationships among heterogeneous data, and adopts an iterative paradigm for SimFusion computation. However, due to the row normalization of URM, the traditional SimFusion model may produce the trivial solution; worse still, the iterative computation of SimFusion may not ensure the global convergence of the solution. This paper studies the revision of this model, providing a full treatment from complexity to algorithms. (1) We propose SimFusion+ based on a notion of the *Unified Adjacency Matrix* (UAM), a modification of the URM, to prevent the trivial solution and the divergence issue of SimFusion. (2) We show that for any vertex-pair, SimFusion+ can be performed in $O(1)$ time and $O(n)$ space with an $O(km)$-time precomputation done only once, as opposed to the $O(kn^3)$ time and $O(n^2)$ space of its traditional counterpart, where n, m, and k denote the number of vertices, edges, and iterations respectively. (3) We also devise an incremental algorithm for further improving the computation of SimFusion+ when networks are dynamically updated, with performance guarantees for similarity estimation. We experimentally verify that these algorithms scale well, and the revised notion of SimFusion is able to converge to a non-trivial solution, and allows us to identify more sensible structure information in large real-world networks.

Categories and Subject Descriptors

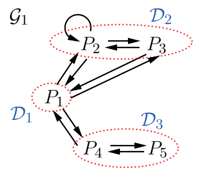
H.3.3 [**Information Search and Retrieval**]: Information Storage and Retrieval; G.2.2 [**Graph Theory**]: Discrete Mathematics

Keywords

Similarity Computation, SimFusion, Web Ranking Algorithm

1. INTRODUCTION

The conundrum of measuring similarity between objects based on hyperlinks in a graph has fueled a growing interest in the fields of information retrieval. This problem is also known as "link-based analysis" or "structural similarity search", and it has been extensively studied by different communities with a proliferation of emerging applications. Examples include collaborative filtering, hyper-text classification, graph clustering and proximity query processing. Recently, while the scale of the Web has dramatically increased our need to produce large graphs, the study of efficiently computing object similarity on such large graphs becomes a desideratum. In practice, an effective similarity measure should not only correlate well with human intuition but also scale well for large amounts of data.



$$\lim_{k\to\infty} \textbf{SimFusion}^{(k)} = \begin{array}{c|ccccc} & \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_2 & \mathcal{D}_3 & \mathcal{D}_3 \\ \hline \mathcal{D}_1 & 1 & .219 & .219 & .219 & .219 \\ \mathcal{D}_2 & .219 & 1 & .219 & .219 & .219 \\ \mathcal{D}_2 & .219 & .219 & 1 & .219 & .219 \\ \mathcal{D}_3 & .219 & .219 & .219 & 1 & .219 \\ \mathcal{D}_3 & .219 & .219 & .219 & .219 & 1 \end{array}$$

$$\lim_{k\to\infty} \textbf{SimFusion+}^{(k)} = \begin{array}{ccccc} \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_2 & \mathcal{D}_3 & \mathcal{D}_3 \\ \hline 1 & .290 & .194 & .139 & .100 \\ .290 & 1 & .304 & .217 & .156 \\ .194 & .304 & 1 & .145 & .105 \\ .139 & .217 & .145 & 1 & .075 \\ .100 & .156 & .105 & .075 & 1 \end{array}$$

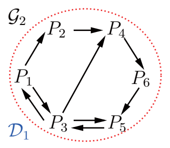
Figure 1: Trivial SimFusion on Heterogeneous Domain

Among the existing metrics, SimFusion [1] can be regarded as one of the attractive ones on account of the following reasons. (i) Similar to PageRank [2] and SimRank [3], SimFusion is based on hyperlinks and follows the reinforcement assumption that "the similarity between objects is reinforced by the similarity of their related objects", which is fairly intuitive and conforms to our basic understandings. (ii) Unlike other measures (*e.g.,* PageRank and SimRank) that explore the linkage patterns merely from a single data space [2–4], SimFusion has the extra benefits of incorporating both inter- and intra-relationships from multiple data spaces in a unified manner to measure the similarity of heterogeneous data objects. (iii) SimFusion offers more intuitive and flexible ways of assigning weighting factors to each data space that reflects their relative importance, as opposed to the PageRank and SimRank measures that need to determine a damping factor. (iv) SimFusion provides a general-purpose framework for measuring structural similarity in a recursive fashion; other well-known measures, such as Co-Citation [5] and Coupling [6] are just special cases of SimFusion.

However, existing work on SimFusion has the following problems. Firstly, although the basic intuition behind the SimFusion model is appealing, it seems inappropriate to use the *Unified Relationship Matrix* (URM) to represent the relationships of heterogeneous objects. The main problem is that, according to the definition of URM $\mathbf{L}$ in [1], the sum of each row of $\mathbf{L}$ is always equal to 1. Since the product of $\mathbf{L}$ and the matrix $\mathbf{1}$ whose entries are all ones is equal to the matrix $\mathbf{1}$ of all ones, there always exists a trivial solution $\mathbf{S} = \mathbf{1}$ to the original SimFusion formula $\mathbf{S} = \mathbf{L} \cdot \mathbf{S} \cdot \mathbf{L}^T$ [1], as illustrated in Example 1. The same phenomena of yielding such a trivial solution may occur in our experimental results in Section 6. To address this issue, we shall revise the original SimFusion model.

EXAMPLE 1 (TRIVIAL SOLUTION). *Figure 1 depicts a graph $\mathcal{G}_1$ partly extracted from Cornell CS Department. Each vertex P_i*

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.



$$\lim_{k\to\infty} \textbf{SimFusion}^{(2k)} \begin{bmatrix} .46 & 0 & .46 & 0 & 0 & .46 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ .46 & 0 & .46 & 0 & 0 & .46 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ .46 & 0 & .46 & 0 & 0 & .46 \end{bmatrix} \neq \lim_{k\to\infty} \textbf{SimFusion}^{(2k+1)} \begin{bmatrix} .38 & 0 & .38 & 0 & 0 & .38 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ .38 & 0 & .38 & 0 & 0 & .38 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ .38 & 0 & .38 & 0 & 0 & .38 \end{bmatrix} \quad \lim_{k\to\infty} \textbf{SimFusion+}^{(k)} \begin{bmatrix} 1 & 0 & .25 & 0 & 0 & .25 \\ 0 & 1 & 0 & .25 & .18 & 0 \\ .25 & 0 & 1 & 0 & 0 & .18 \\ 0 & .25 & 0 & 1 & .25 & 0 \\ 0 & .18 & 0 & .25 & 1 & 0 \\ .25 & 0 & .18 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2: Divergent SimFusion on Homogeneous Domain

denotes a web page, and each edge a hyperlink. There are three categories: $\mathcal{D}_1 = \{P_1\}$ (student), $\mathcal{D}_2 = \{P_2, P_3\}$ (staff), and $\mathcal{D}_3 = \{P_4, P_5\}$ (faculty). We want to retrieve the top-3 similar pairs of web pages in $\mathcal{G}_1$. However, the naive SimFusion fails to correctly find them. We observe that the SimFusion solution is a (trivial) matrix whose entries are all the same. In fact, vertices in $\mathcal{G}_1$ do not have the identical neighbor structures. Hence, the trivial solution is non-semantic in real communities.

Secondly, it is rather expensive to compute SimFusion similarities. The existing approach for SimFusion computation deploys a fixed-point iteration: $\mathbf{S}^{(k+1)} = \mathbf{L} \cdot \mathbf{S}^{(k)} \cdot \mathbf{L}^T$, which requires $O(kn^3)$ time and $O(n^2)$ space [1]. This impedes the scalability of SimFusion on large graphs. Worse still, the iterative computation of SimFusion do not always converge. The convergence of the SimFusion iterations heavily depends on the choice of the initial guess $\mathbf{S}^{(0)}$, as shown in Example 2.

EXAMPLE 2 (DIVERGENCE SIMFUSION). *Consider the disease transmission graph $\mathcal{G}_2$, where each vertex is an organism P_i which can carry the disease, and an edge represents one organism spreading it to another. One wants to find the three most similar organisms to P_2 in $\mathcal{G}_2$. However, the iterative computation of SimFusion does not work properly. We observe the following:*

(i) When $\mathbf{S}^{(0)}$ is set to an $n \times n$ identity matrix (according to [1]) the "even" and "odd" subsequences of $\{\mathbf{S}^{(k)}\}$ are convergent respectively, but they do not converge to the same limit, which makes the full sequence $\{\mathbf{S}^{(k)}\}$ divergent.

(ii) Choosing $\mathbf{S}^{(0)} = \mathbf{1}_n$ (i.e., an $n \times n$ matrix of all 1s) instead, we observe that the full SimFusion sequence $\{\mathbf{S}^{(k)}\}$ is always convergent to $\mathbf{1}_n$ regardless of the graph structure.

This suggests that the original SimFusion iterations may be divergent or converge to a trivial solution, not to mention its scalability. This highlights the need to find a feasible way to guarantee the convergence of the SimFusion iterations, but it is hard to devise an efficient algorithm for the revised SimFusion computation.

Thirdly, it is a big challenge to incrementally compute SimFusion on dynamic graphs. The traditional method [1] has to recompute the similarity from scratch when edges in a graph change over time, which is not adaptive to many evolving networks. Fortunately, we have an observation that the size of the areas affected by the updates is typically small in practice. To this end, we propose an incremental algorithm that fully utilizes these affected areas to compute SimFusion on dynamic graphs.

Contributions. This paper proposes SimFusion+, a revised notion of SimFusion, to provide a full treatment of SimFusion for the convergence issues and to improve its computational efficiency. In summary, we make the following contributions.

1. We formalize the problem of SimFusion+ computation (Section 2). The notion of SimFusion+ revises the divergence and non-semantic convergence worries of the traditional model [1].
2. We present optimization techniques for improving the computation of SimFusion+ to $O(1)$ time and $O(n)$ space for every pair of vertices, plus an $O(km)$-time precomputation run only once (Section 3).
3. We devise an efficient algorithm to compute the SimFusion+ similarity with better accuracy guarantees (Section 4). An error estimate is also given for the SimFusion+ approximation.
4. We devise an incremental algorithm for further optimizing the SimFusion+ computation when edges in networks are dynamically updated (Section 5). We show that the update cost of the incremental algorithm retains $O(\delta n)$ time and $O(n)$ space for handling a sequence of δ edge insertions or deletions.
5. We experimentally verify the effectiveness and scalability of the algorithms, using real and synthetic data (Section 6). The results show that SimFusion+ can govern the convergence towards a meaningful solution, and our algorithms achieve high accuracy and significantly outperform the baseline algorithms.

Related Work. The link-based similarity has become increasingly popular since the famous result of Google PageRank [2] on ranking web pages. Since then, there has been a surge of papers focusing on web link analysis. In particular, a growing interest has been witnessed in the SimFusion model over the past decade [1, 7] as it provides a useful measure of similarity that supports different kinds of intra- and inter-node relations from multiple data spaces.

The iterative computation of SimFusion was proposed in [1] with several problems left open there. In comparison, this work extends [1] by (i) addressing the divergent and trivial solution of the original SimFusion, (ii) optimizing the time and space complexity of similarity computation, and (iii) supporting incremental update on evolving graphs, none of which was considered in [1].

It is worth mentioning that Jeh and Widom have proposed a similar structural measure called SimRank [3], predicated, as SimFusion is, on the idea that vertices are similar if they have similar neighbor structures. The essential difference between the two models is the notion of the convergence principle. SimFusion ensures the existence of the stationary distribution and ergodicity convergence to this distribution, whereas SimRank hinges on a damping factor $0 < c < 1$ to govern the convergence.

Optimization techniques have been devised for SimRank computation (*e.g.,* [8–11]). The best-known SimRank algorithm yields $O(k \min\{nm, \frac{n^3}{\log n}\})$ time [8]. The performance gain is mainly achieved by a partial sum function for amortization; as for SimFusion, the conventional matrix multiplication in its iterative formula misled its complexity, which was previously considered $O(kn^3)$ time and $O(n^2)$ space. The idea of the dominant eigenvector in this work significantly improves its computation to $O(km)$ time and $O(kn)$ space, which is more efficient than SimRank [8].

There has also been work on link-based similarity computation. A unified framework of link-based analysis was addressed in [7], which extends PageRank and HITS by (i) considering both inter- and intra-type relationships, and (ii) bringing order to data objects in different data spaces. It differs from this work in that the focus is on finding attribute values of a single object, rather than on improving the complexities for similarity estimation. Extensions of similarity reinforcement assumption were studied in [12], by spreading multiple relationship similarities over interrelated data objects to enhance their mutual reinforcement effects. Nevertheless, neither of these deduces rigorous mathematical formulae, and the rationales behind the integration approaches are different from this work. Recently, a closed-form solution to P-Rank (Penetrating-Rank) formula was addressed in [13]. Cai *et al.* [13] showed that when the damping factor $c = 1$ and weighting factor $\gamma = 0$, P-Rank can be reduced to SimFusion. However, this reasoning is based on the flawed assumption that the diagonal entries diag($\mathbf{S}$) of the P-Rank similarity matrix were not considered. We argue that P-Rank is defined recursively, and hence, the omission of diag($\mathbf{S}$) has an impact on the similarity of a vertex with itself, and recursively, it has an impact on the similarity of different pairs of vertices.

2. SIMFUSION ESTIMATION REVISED

In this section, we first revisit the definition of data space and data relation. We then introduce the notion of the *Unified Adjacency Matrix* (UAM) to revise the SimFusion model.

2.1 Data Space and Data Relation

Graphs studies here are digraphs having no multiple edges.

Data Space. A *data space* is the finite set of all data objects (vertices) with the same data type, denoted by $\mathcal{D} = \{o_1, o_2, \cdots\}$. $|\mathcal{D}|$ denotes the number of data objects in $\mathcal{D}$. Two nonempty data spaces $\mathcal{D}$ and $\mathcal{D}'$ are said to be *disjoint* if $\mathcal{D} \cap \mathcal{D}' = \varnothing$.

Throughout the paper, we shall use the following notations. (i) The *entire space* $\mathcal{D}$ in a network is the union of N disjoint data spaces $\mathcal{D}_1, \cdots, \mathcal{D}_N$ such that $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$ and $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing \quad (i \neq j)$. (ii) The total size $|\mathcal{D}|$ of the entire space, denoted by n, is the sum of the number n_i of the data objects in each data space $\mathcal{D}_i$, *i.e.,* $n = \sum_{i=1}^{N} n_i$ with $n_i = |\mathcal{D}_i| \quad (\forall i = 1, \cdots, N)$.

Intuitively, for heterogeneous networks, the distinct spaces $\mathcal{D}_i$ form a partition of $\mathcal{D}$ into classes. For homogenous networks, the partition of $\mathcal{D}$ is itself.

Data Relation. A *data relation* on $\mathcal{D}$ is defined as $\mathcal{R} \subseteq \mathcal{D} \times \mathcal{D}$, where $(o, o') \in \mathcal{R}$ is a connection (a directed edge) from object o to o'. Data objects in the same data space are related via *intra-type relations* $\mathcal{R}_{i,i} \subseteq \mathcal{D}_i \times \mathcal{D}_i$. Data objects between distinct data spaces are related via *inter-type relations* $\mathcal{R}_{i,j} \subseteq \mathcal{D}_i \times \mathcal{D}_j \quad (i \neq j)$.

Intuitively, the intra-type relation carries connected information in each data space (*e.g.,* co-citation between web pages); the inter-type relation represents interlinked information between different data spaces (*e.g.,* making user requests). As an example, in Figure 1 there are three data spaces : $\mathcal{D} = \{P_1\} \cup \{P_2, P_3\} \cup \{P_4, P_5\}$, where $(P_2, P_2), (P_2, P_3), (P_3, P_2), (P_4, P_5), (P_5, P_4)$ are intra-type relations; $(P_1, P_2), (P_2, P_1), (P_3, P_1), (P_1, P_3), (P_1, P_4), (P_4, P_1)$ are inter-type relations.

2.2 Unified Adjacency Matrix

Let us now introduce the *unified adjacency matrix* (UAM). Consider a graph $\mathcal{G} = (\mathcal{D}, \mathcal{R})$ with data space $\mathcal{D}$ and data relation $\mathcal{R}$.

Unified Adjacency Matrix (UAM). The matrix $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/n^2$ of size $n \times n$ is said to be a *unified adjacency matrix* of the relation $\mathcal{R}$ whenever

$$\tilde{\mathbf{A}} = \begin{pmatrix} \lambda_{1,1}\mathbf{A}_{1,1} & \lambda_{1,2}\mathbf{A}_{1,2} & \cdots & \lambda_{1,N}\mathbf{A}_{1,N} \\ \lambda_{2,1}\mathbf{A}_{2,1} & \lambda_{2,2}\mathbf{A}_{2,2} & \cdots & \lambda_{2,N}\mathbf{A}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N,1}\mathbf{A}_{N,1} & \lambda_{N,2}\mathbf{A}_{N,2} & \cdots & \lambda_{N,N}\mathbf{A}_{N,N} \end{pmatrix},$$

where (i) $\mathbf{1}$ is the $n \times n$ matrix of all ones; (ii) $\mathbf{A}_{i,j}$ is the submatrix of $\mathbf{A}$ whose (o, o')-entry equals 1 if there is an edge from data object o to o', *i.e.,* $\exists\,(o, o') \in \mathcal{R}$, $\frac{1}{n_j}$ if data object o has no neighbors in $\mathcal{D}_j$, or 0 otherwise; and (iii) $\lambda_{i,j}$ is called the *weighting factor* between data space $\mathcal{D}_i$ and $\mathcal{D}_j$ with $0 \leq \lambda_{i,j} \leq 1$ and $\sum_{j=1}^{N} \lambda_{i,j} = 1 \; (\forall i = 1, \cdots, N)$.

Intuitively, $\mathbf{A}_{i,j}$ represents the intra- $(i = j)$ or inter- $(i \neq j)$ relation from data space $\mathcal{D}_i$ to $\mathcal{D}_j$. $\lambda_{i,j}$ reflects the relative importance between data spaces $\mathcal{D}_i$ and $\mathcal{D}_j$.

EXAMPLE 3. *In Figure 1, the relative importance between data space $\mathcal{D}_i$ and $\mathcal{D}_j$ is denoted by a weighting matrix $\mathbf{\Lambda} = (\lambda_{i,j})_{3\times 3}$. Then, the UAM $\mathbf{A}$ of $\mathcal{G}_1$ can be derived from $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/n^2$, where $\tilde{\mathbf{A}}$ is computed from $\mathbf{\Lambda}$ as follows.*

$$\mathbf{\Lambda} = \begin{array}{c|ccc} & \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ \hline \mathcal{D}_1 & \frac{1}{2} & \frac{1}{6} & \frac{1}{3} \\ \mathcal{D}_2 & \frac{1}{6} & \frac{7}{12} & \frac{1}{4} \\ \mathcal{D}_3 & \frac{1}{3} & \frac{1}{4} & \frac{5}{12} \end{array} \Rightarrow \tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2}\begin{bmatrix}1\end{bmatrix} & \frac{1}{6}\begin{bmatrix}1 & 1\end{bmatrix} & \frac{1}{3}\begin{bmatrix}1 & 0\end{bmatrix} \\ \frac{1}{6}\begin{bmatrix}1 \\ 1\end{bmatrix} & \frac{7}{12}\begin{bmatrix}1 & 1 \\ 1 & 0\end{bmatrix} & \frac{1}{4}\begin{bmatrix}\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2}\end{bmatrix} \\ \frac{1}{3}\begin{bmatrix}1 \\ 0\end{bmatrix} & \frac{1}{4}\begin{bmatrix}\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2}\end{bmatrix} & \frac{5}{12}\begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix} \end{bmatrix} = \left[\begin{array}{c|cc|cc} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \hline \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \hline \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{array}\right]$$

SimFusion+ Model. In light of the UAM $\mathbf{A}$, we next propose the revised model of SimFusion, termed SimFusion+, as follows:

$$\mathbf{S} = \frac{\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T\|_2}, \tag{1}$$

where $\mathbf{S}$ is called the *Unified Similarity Matrix* (USM) whose (i, j)-entry represents the similarity score between object i and j.

The uniqueness and existence of the SimFusion+ solution $\mathbf{S}$ to Eq.(1) can be established by *the power iteration* [14, pp.381]. A detailed proof will be shown in Proposition 1 (Section 3).

The revised notion of SimFusion utilizes UAM (rather than URM) to represent data relations because UAM can effectively avoid divergent and trivial similarity solutions while well preserving the intuitive reinforcement assumption of the original model [1]. We observe that the root cause of the flawed solution to the original SimFusion is the "row normalization" of URM. Thus, by using UAM, we have an opportunity to postpone the operation of "row normalization" in a delayed fashion. To this end, we utilize the matrix 2-norm $\|\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T\|_2$ to squeeze similarity scores in $\mathbf{S}$ into $[0, 1]$. The obtained similarity results in USM can not only prevent the divergence issue and the trivial solution but effectively capture the reliability of the similarity evidence between data objects. For instance, the SimFusion+ USMs in Examples 1 and 2 are nontrivial and intuitively explainable.

3. COMPUTING SIMILARITY VIA DOMINANT EIGENVECTOR

A conventional approach for finding the SimFusion+ solution $\mathbf{S}$ to Eq.(1) is to employ the following fixed-point iteration: [1]

$$\mathbf{S}^{(k+1)} = \frac{\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T\|_2}. \tag{2}$$

However, as the matrix multiplication may contain $O(n^3)$ operations, it requires $O(kn^3)$ time and $O(n^2)$ space to compute Eq.(2) for k iterations, which may be quite expensive.

In this section, we study the optimization techniques to improve the computation of SimFusion+. Our key observation is that SimFusion+ computation can be converted into finding the dominant eigenvector of the UAM $\mathbf{A}$. The idea is to calculate the dominant eigenvector of $\mathbf{A}$ once, offline, for the preprocessing, and then it can be effectively memorized to compute similarity at query time.

We first revisit the definition of the dominant eigenvector.

DEFINITION 1 ([14, P.379]). *The dominant eigenvector of the $\mathbf{X}$ is an eigenvector, denoted by $\sigma_{\max}(\mathbf{X})$, corresponding to the eigenvalue λ of the largest absolute value of $\mathbf{X}$ such that*

$$\mathbf{X} \cdot \sigma_{\max}(\mathbf{X}) = \lambda \cdot \sigma_{\max}(\mathbf{X}) \;\; \textit{with} \; \|\sigma_{\max}(\mathbf{X})\|_2 = 1.$$

The dominant eigenvector of the UAM can be utilized for speeding up SimFusion+ computation based on the following proposition.

PROPOSITION 1. *Let $\mathbf{A}$ be the UAM of network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$. The SimFusion+ matrix $\mathbf{S}$ can be computed as*

$$[\mathbf{S}]_{i,j} = [\sigma_{\max}(\mathbf{A})]_i \times [\sigma_{\max}(\mathbf{A})]_j, \tag{3}$$

where $[\star]_{i,j}$ denotes the (i, j)-entry of a matrix, and $[\star]_i$ denotes the i-th entry of a vector.

PROOF. We shall use the knowledge of Kronecker product $(\otimes)$ and vec operator (see [15, p.139] for a detailed description).

(i) We first prove that $vec(\mathbf{S}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A})$.

Taking $vec(\star)$ on both sides of Eq.(2) and applying Kronecker property $vec(\mathbf{BCD}^T) = (\mathbf{D} \otimes \mathbf{B}) \cdot vec(\mathbf{C})$ [15, p.147] yield

$$vec(\mathbf{S}^{(k+1)}) = \frac{(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})}{\|(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})\|_2}. \tag{4}$$

[1] Note that the uniqueness of the SimFusion+ solution guarantees that $\mathbf{S}$ is insensitive to the initial guess $\mathbf{S}^{(0)}$. For convenience, we choose $\mathbf{S}^{(0)} = \mathbf{1}$ of all 1s, which can be interpreted as "initially, no other vertex-pair is presumably more similar than itself".

Let $\mathbf{x}^{(k)} \triangleq vec(\mathbf{S}^{(k)})$ and $\mathbf{M} \triangleq \mathbf{A} \otimes \mathbf{A}$. Then Eq.(4) having the form $\mathbf{x}^{(k+1)} = \mathbf{M}\mathbf{x}^{(k)}/\|\mathbf{M}\mathbf{x}^{(k)}\|_2$ fits the *power iteration* paradigm [14, pp.381], which follows that the sequence $\{\mathbf{x}^{(k)}\}$ converges to the dominant eigenvector of $\mathbf{M}$. This in turn implies

$$vec(\mathbf{S}) \triangleq \lim_{k\to\infty} vec(\mathbf{S}^{(k)}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A}).$$

One caveat is that the convergence of $vec(\mathbf{S}^{(k)})$ is ensured by the positivity of $\mathbf{A}\otimes\mathbf{A}$ [16, p.508] [2]. This is true because $\mathbf{A}$ is positive and the self-Kronecker product of two positive matrices preserves positivity.

(ii) We next show that $\sigma_{\max}(\mathbf{A} \otimes \mathbf{A}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})$.

Since $\mathbf{A} \cdot \sigma_{\max}(\mathbf{A}) = \lambda \cdot \sigma_{\max}(\mathbf{A})$, it follows that

$$(\mathbf{A} \otimes \mathbf{A}) \cdot (\sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})) = (\mathbf{A}\sigma_{\max}(\mathbf{A})) \otimes (\mathbf{A}\sigma_{\max}(\mathbf{A}))$$
$$= (\lambda \cdot \sigma_{\max}(\mathbf{A})) \otimes (\lambda \cdot \sigma_{\max}(\mathbf{A})) = \lambda^2 \cdot (\sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})).$$

This implies that the dominant eigenvector of $\mathbf{A}\otimes\mathbf{A}$ is actually the self-Kronecker product of the dominant eigenvector of $\mathbf{A}$. Hence,

$$vec(\mathbf{S}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A}).$$

It can be noticed that the (i, j)-entry of the matrix $\mathbf{S}$ (*i.e.,* the $((i-1)\times n+j)$-th entry of the vector $vec(\mathbf{S})$) is exactly the product of the i-th and j-th entries of $\sigma_{\max}(\mathbf{A})$. Thus, Eq.(3) holds. □

Proposition 1 provides the efficient technique for accelerating SimFusion+ computation. The central point in optimizing $\mathbf{S}$ computation is that only *matrix-vector* multiplication is used for computing $\sigma_{\max}(\mathbf{A})$. Once calculated, the vector $\sigma_{\max}(\mathbf{A})$ is memorized and thus will not be recomputed when subsequently required, as opposed to the naive *matrix-matrix* multiplication in Eq.(2).

EXAMPLE 4. *Consider the graph $\mathcal{G}_1$ in Fig.1 with its UAM $\mathbf{A}$ already computed in Example 3. The dominant eigenvector of $\mathbf{A}$ is*

$$\sigma_{\max}(\mathbf{A}) = [.431 \quad .673 \quad .451 \quad .322 \quad .232]^T.$$

Then using Eq.(3) for computing $\mathbf{S}$ yields

$$\mathbf{S} = \begin{bmatrix} .186 & .290 & .194 & .139 & .100 \\ .290 & .453 & .304 & .217 & .156 \\ .194 & .304 & .203 & .145 & .105 \\ .139 & .217 & .145 & .104 & .075 \\ .100 & .156 & .105 & .075 & .054 \end{bmatrix}.$$

Note that $\sigma_{\max}(\mathbf{A})$ is calculated only once for the preprocessing and can be used for computing any entry of $\mathbf{S}$ at query time, e.g.,

$$[\mathbf{S}]_{1,2} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_2 = .431 \times .673 = .290.$$
$$[\mathbf{S}]_{1,3} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_3 = .431 \times .451 = .194.$$

Regarding computational complexity, our approach only needs $O(km)$ preprocessing time and $O(n)$ space to compute $\sigma_{\max}(\mathbf{A})$ by using the following *power iteration* [14, pp.381]:

$$\boldsymbol{\xi}^{(0)} = \mathbf{e}, \quad \boldsymbol{\xi}^{(k+1)} = \frac{\mathbf{A}\boldsymbol{\xi}^{(k)}}{\|\mathbf{A}\boldsymbol{\xi}^{(k)}\|_2} = \frac{\tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)} + \gamma^{(k)}\mathbf{e}}{\|\tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)} + \gamma^{(k)}\mathbf{e}\|_2}, \tag{5}$$

where $\mathbf{e} \triangleq (1, \cdots, 1)^T \in \mathbb{R}^n$ and $\gamma^{(k)} \triangleq \frac{1}{n^2}\sum_{i=1}^{n} [\boldsymbol{\xi}^{(k)}]_i$. [3] The existence and uniqueness of the dominant eigenvector $\sigma_{\max}(\mathbf{A})$ is guaranteed by the combination of Perron-Frobenius theorem [14, p.383] and the positivity of $\mathbf{A}$. Thus, by applying the power method, the sequence $\{\boldsymbol{\xi}^{(k)}\}$ converges to $\sigma_{\max}(\mathbf{A})$. Then, with $\sigma_{\max}(\mathbf{A})$ being memorized, only $O(1)$ time is required at query stage for computing each entry of $\mathbf{S}$ via Eq.(3). Indeed, due to $\mathbf{S}$ symmetry, only $n(n+1)/2$ entries $[\mathbf{S}]_{i,j}$ $(i \le j)$ need to be computed. In contrast to the $O(kn^3)$ time and $O(n^2)$ space of the conventional iterations, our approach is a significant improvement achieved by $\sigma_{\max}(\mathbf{A})$ computation.

[2]According to the Perron-Frobenius theorem [14, p.383], the positivity of $\mathbf{A} \otimes \mathbf{A}$ ensures that there exists a unique dominant eigenvector of $\mathbf{A}\otimes\mathbf{A}$ associated with its eigenvalue being strictly greater in magnitude than its other eigenvalues.

[3]The correctness of Eq.(5) can be proved as follows:
$\mathbf{A}\boldsymbol{\xi}^{(k)} = (\tilde{\mathbf{A}}+\frac{1}{n^2}\mathbf{e}\mathbf{e}^T)\boldsymbol{\xi}^{(k)} = \tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)}+\gamma^{(k)}\mathbf{e}$ with $\gamma^{(k)} = \frac{1}{n^2}\mathbf{e}^T\boldsymbol{\xi}^{(k)}$.

Our method of memorizing $\sigma_{\max}(\mathbf{A})$ can extra accelerate SimFusion+ computation when only a small portion of similarity values of $\mathbf{S}$ need to be computed. Specifically, for certain applications like K-nearest neighbor (KNN) queries, given a vertex i as a query, one needs to retrieve the top-K ($\ll n$) most similar vertices in a graph by computing the i-th row of $\mathbf{S}$. Before Proposition 1 is introduced, computing the similarity of only one vertex-pair still requires $O(kn^3)$ time. In contrast, using the memorized $\sigma_{\max}(\mathbf{A})$, we only need $O(1)$ time for computing a single entry of $\mathbf{S}$ at query time. In fact, for KNN queries, after $\sigma_{\max}(\mathbf{A})$ is memorized with its entries sorted in an descending order for the preprocessing, it only takes constant time to retrieve the top-K results at query stage.

Proposition 1 also gives an interesting characterization of the SimFusion+ matrix.

COROLLARY 1. *The SimFusion+ matrix $\mathbf{S}$ is a rank 1 matrix.*

PROOF. Applying Eq.(3) to $\mathbf{S}$, we obtain that for any two rows of $\mathbf{S}$, $\exists\ \omega = [\sigma_{\max}(\mathbf{A})]_x/[\sigma_{\max}(\mathbf{A})]_y$ *s.t.* $[\mathbf{S}]_{x,*} = \omega \times [\mathbf{S}]_{y,*}$. Hence, the rank of $\mathbf{S}$ is 1. □

4. ESTIMATING SIMFUSION+ WITH BETTER ACCURACY

After the dominant eigenvector $\sigma_{\max}(\mathbf{A})$ has been suggested to speed up SimFusion+ computation, the algorithm presented in this section can guarantee more accurate similarity results.

The main idea of our approach is to leverage an orthogonal subspace for "upper-triangularizing" the UAM $\mathbf{A}$ ($n \times n$ dimension) into a *small* matrix $\mathbf{T}_k$ ($k \times k$ dimension) with $k \ll n$. Due to $\mathbf{T}_k$ small size and almost "upper-triangularity", computing the dominant eigenvector $\sigma_{\max}(\mathbf{T}_k)$ is far less costly than straightforwardly computing $\sigma_{\max}(\mathbf{A})$. We show that the choice of k provides a user-controlled accuracy over the similarity scores. The underlying rationale is that the dominant eigenvector of a matrix can be well-preserved by an orthogonal transformation.

We first use the technique of the Arnoldi decomposition [17] to build an order-k orthogonal subspace for the UAM $\mathbf{A}$.

LEMMA 1 ([17, PP.25-33]). *Let $\mathbf{A}$ be an $n \times n$ matrix. Then, for every $k = 1, 2, \cdots$, we have the following results.*
(a) There exists a unique $k \times k$ almost triangular matrix $\mathbf{T}_k$ s.t.

$$\mathbf{V}_k^T \mathbf{A} \mathbf{V}_k = \mathbf{T}_k, \tag{6}$$

where $\mathbf{V}_k = [\mathbf{v}_1 \,|\, \mathbf{v}_2 \,|\, \cdots \,|\, \mathbf{v}_k]$ is an $n \times k$ matrix consisting of k orthonormal column-vectors $\mathbf{v}_i \in \mathbb{R}^n$ $(i = 1, \cdots, k)$.
(b) The difference between $\mathbf{A}\mathbf{V}_k$ and $\mathbf{V}_k\mathbf{T}_k$ is a zero matrix except the last column. Precisely, there exist a small scalar δ_k and an orthonormal vector $\mathbf{v}_{k+1} \in \mathbb{R}^n$ such that

$$\mathbf{A}\mathbf{V}_k - \mathbf{V}_k\mathbf{T}_k = \delta_k \mathbf{v}_{k+1}\mathbf{e}_k^T, \tag{7}$$

where $\mathbf{e}_k = (0, \cdots, 0, 1)^T \in \mathbb{R}^k$ is a unit vector.

As depicted in Fig.3, by using the upper-triangularization process [4], the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ can be transformed into the small almost triangular $\mathbf{T}_k \in \mathbb{R}^{k\times k}$ by the $n \times k$ orthonormal ma-

[4]From the computational viewpoint, $\mathbf{V}_k, \mathbf{T}_k, \mathbf{v}_k$ and δ_k in Eq.(7) can be obtained by an algorithm in our later developments.

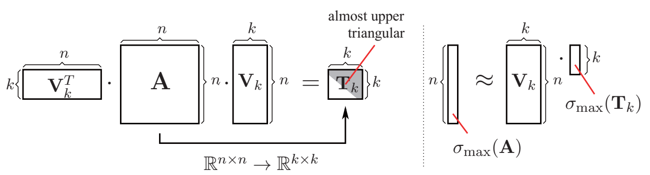

almost upper triangular
n
k
$\mathbf{V}_k^T$
$\mathbf{A}$
$\mathbf{V}_k$
$\mathbf{T}_k$
$\mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{k\times k}$
$\approx$
$\sigma_{\max}(\mathbf{A})$
$\sigma_{\max}(\mathbf{T}_k)$

Figure 3: Upper Triangular Process of UAM

trix $\mathbf{V}_k$ for every iteration. As k increases, $\mathbf{T}_{k+1}$ can be iteratively obtained by bordering the matrix $\mathbf{T}_k$ at the last iteration (*i.e.,* $\mathbf{T}_{k+1} = [\begin{smallmatrix}\mathbf{T}_k & \star \\ \star & \star\end{smallmatrix}]$), and $\mathbf{V}_{k+1}$ by augmenting the matrix $\mathbf{V}_k$ at the last iteration with the vector $\mathbf{v}_{k+1}$ (*i.e.,* $\mathbf{V}_{k+1} = [\mathbf{V}_k \,|\, \mathbf{v}_{k+1}]$). When $k = rank(\mathbf{A})$, it follows that $\delta_k = 0$.

EXAMPLE 5. *Consider the network $\mathcal{G}_1$ in Fig.1 and its UAM $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/5^2$ in Example 3. For $k = 3$, $\exists\, \mathbf{V}_3 = [\mathbf{v}_1 \,|\, \mathbf{v}_2 \,|\, \mathbf{v}_3] \in \mathbb{R}^{5\times 3}$ mapping $\mathbf{A} \in \mathbb{R}^{5\times 5}$ into $\mathbf{T}_3 \in \mathbb{R}^{3\times 3}$ s.t. $\mathbf{T}_3 = \mathbf{V}_3^T \mathbf{A} \mathbf{V}_3$, where*

$$\tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{bmatrix}, \; \mathbf{T}_3 = \begin{bmatrix} 1.08 & .298 & 0 \\ .298 & .190 & .359 \\ 0 & .359 & -.083 \end{bmatrix}, \; \mathbf{V}_3 = \left[\begin{array}{c|c|c} .447 & .125 & -.089 \\ .447 & .750 & .044 \\ .447 & -.125 & .710 \\ .447 & -.125 & -.696 \\ .447 & -.625 & .032 \end{array}\right].$$

$\exists\, \delta_3 = .231, \mathbf{v}_4 = [-.881\ .328\ .137\ .280\ .135]^T$ *s.t. Eq.*(7) *holds. (see Example 7 for a detailed iterative process)*

In light of Lemma 1, we next provide an error estimate for SimFusion+ similarity when using $\sigma_{\max}(\mathbf{T}_k)$ to compute $\sigma_{\max}(\mathbf{A})$.

Error Estimation. We define a k-approximation similarity matrix $\hat{\mathbf{S}}_k$ over a low-order parameter k:

$$[\hat{\mathbf{S}}_k]_{i,j} = [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_i \times [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_j, \tag{8}$$

where $\mathbf{V}_k$ and $\mathbf{T}_k$ can be obtained from Lemma 1.

To differentiate $\hat{\mathbf{S}}_k$ from $\mathbf{S}$, we shall refer to $\mathbf{S}$ as *exact* similarity.

The following estimate for the approximate similarity $\hat{\mathbf{S}}_k$ with respect to the exact $\mathbf{S}$ can be established.

PROPOSITION 2. *For every $k = 1, 2, \cdots$, the following estimate holds:*

$$\|\hat{\mathbf{S}}_k - \mathbf{S}\|_2 \le \epsilon_k, \tag{9}$$

where

$$\epsilon_k = 2 \times |\delta_k \times [\sigma_{\max}(\mathbf{T}_k)]_k|, \tag{10}$$

and δ_k is a small scalar given in Eq.(7)*; $[\sigma_{\max}(\mathbf{T}_k)]_k$ is the k-th entry of the dominant eigenvector of $\mathbf{T}_k$.*

(Please refer to the Appendix for a detailed proof.)

The parameter ϵ_k is intended as a user control over the difference between the approximate and the exact similarity matrices, and hence ϵ_k is generally chosen by a user. Provided that k is selected to satisfy Eq.(10), Proposition 2 states that the gap between the approximate and the exact similarity scores does not exceed ϵ_k.

EXAMPLE 6. *Consider the network $\mathcal{G}_1$ in Fig.1 and the matrix $\mathbf{T}_3, \mathbf{V}_3, \delta_3$ given in Example 5. For $k = 3$, we have*

$$\sigma_{\max}(\mathbf{T}_3) = [.945\ .316\ .089]^T.$$

Therefore, $\mathbf{V}_3 \cdot \sigma_{\max}(\mathbf{T}_3) = [.454\ .663\ .447\ .321\ .228]^T$. *Then applying Eq.*(8) *and the exact* $\mathbf{S}$ *in Example 4 yields*

$$\hat{\mathbf{S}}_3 = \{\textit{Using Eq.}(8)\} = \begin{bmatrix} .206 & .301 & .203 & .146 & .103 \\ .301 & .440 & .296 & .213 & .151 \\ .203 & .296 & .199 & .143 & .102 \\ .146 & .213 & .143 & .102 & .073 \\ .103 & .151 & .102 & .073 & .051 \end{bmatrix}$$

$$\hat{\mathbf{S}}_3 - \mathbf{S} = \{\textit{Using } \mathbf{S} \textit{ in Example 4}\} = .01 \times \begin{bmatrix} 2.02 & 1.09 & .83 & .67 & .34 \\ 1.09 & -1.33 & -.75 & -.41 & -.51 \\ .83 & -.75 & -.40 & -.21 & -.30 \\ .67 & -.41 & -.21 & -.09 & -.17 \\ .34 & -.51 & -.30 & -.17 & -.20 \end{bmatrix}$$

Algorithm 1: SimFusion+ $(\mathcal{G}, \epsilon, (u, v))$

```
   Input  : Network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, accuracy $\epsilon$, vertex pair $(u, v)$.
   Output: Similarity score $s(u, v)$.
1  compute the matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$ of the UAM $\mathbf{A}$ in $\mathcal{G}$ ;
2  initialize $\mathbf{e} \leftarrow (1, 1, \cdots, 1)^T \in \mathbb{R}^n$, and $\mathbf{v}_1 \leftarrow \frac{1}{\sqrt{n}}\mathbf{e}$ ;
3  foreach iteration $k = 1, 2, \cdots$ do
4      initialize the auxiliary vector $\mathbf{w} \leftarrow \tilde{\mathbf{A}}\mathbf{v}_k + \frac{1}{n^2}(\mathbf{e}^T\mathbf{v}_k)\mathbf{e}$ ;
5      for $i = 1, 2, \cdots, k-1$ do
6          compute the almost upper triangular matrix $\mathbf{T}_k \in \mathbb{R}^{k\times k}$ :
             $[\mathbf{T}_k]_{i,k-1} \leftarrow \mathbf{v}_k^T \cdot \mathbf{w}$ ;
7          orthogonalize $\mathbf{w}$ s.t. $\mathbf{w} \perp span\{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ :
             $\mathbf{w} \leftarrow \mathbf{w} - [\mathbf{T}_k]_{i,k-1} \cdot \mathbf{v}_{k-1}$ ;
8      compute the residual scalar $\delta_k \leftarrow \|\mathbf{w}\|_2$ ;
9      find the dominant eigenvector $\sigma_{\max}(\mathbf{T}_k)$ ;
10     estimate the error $\epsilon_k \leftarrow 2 \times |\delta_k \times [\sigma_{\max}(\mathbf{T}_k)]_k|$;
11     if $\epsilon_k \le \epsilon$ then exit for ;
12     compute the residual vector $\mathbf{v}_{k+1}$ :
         $\mathbf{w} \leftarrow \mathbf{w}/\delta_k$ and $\mathbf{v}_{k+1} \leftarrow \mathbf{w}$ ;
13     free $\delta_k, \sigma_{\max}(\mathbf{T}_k)$ ;
14 free $\mathbf{w}, \mathbf{T}_k, \mathbf{v}_{k+1}, \delta_k$ ;
15 compute the approximate dominant eigenvector $\hat{\sigma}_{\max}(\mathbf{A})$
     $\hat{\sigma}_{\max}(\mathbf{A}) \leftarrow [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \cdot \sigma_{\max}(\mathbf{T}_k)$ ;
16 free $\mathbf{v}_1, \cdots, \mathbf{v}_k, \sigma_{\max}(\mathbf{T}_k)$ ;
17 compute the approximate similarity score of $(u, v)$
     $\hat{s}(u, v) \leftarrow [\hat{\sigma}_{\max}(\mathbf{A})]_u \times [\hat{\sigma}_{\max}(\mathbf{A})]_v$ ;
18 return $\hat{s}(u, v)$ ;
```

We note that the gap between $\mathbf{S}$ *and* $\hat{\mathbf{S}}_k$ *for* $k = 3$ *is actually*

$$\|\hat{\mathbf{S}}_3 - \mathbf{S}\|_2 = .0257,$$

which is smaller than ϵ_k (using Eq.(10) *with* $\delta_3 = .231$*)*

$$\epsilon_k = 2 \times |.231 \times .089| = .0411.$$

Notice that if $k = rank(\mathbf{A})$ $(\ll n)$, [5] then $\epsilon_k = 0$ and the k-approximation similarity matrix $\hat{\mathbf{S}}_k$ becomes the conventional exact USM $\mathbf{S}$. From this perspective, the k-approximation similarity can be regarded as a generalization for the conventional similarity.

One of the possible ways of choosing an appropriate low order k for achieving the desired accuracy ϵ is to calculate the estimation error ϵ_k from Eq.(10) in an a-posteriori fashion after each iteration $k = 1, 2, \cdots$. [6] The iterative process stops once $\epsilon_k \le \epsilon$. Due to ϵ_k decreasing monotonicity, such k is the minimum low order *s.t.* $\|\hat{\mathbf{S}}_k - \mathbf{S}\|_2 \le \epsilon$. More concretely, the residual δ_k in Eq.(7) (Lemma 1) approaches 0 as k is increased to n, which implies

$$\epsilon_k = 2 \times |\delta_k| \times |[\sigma_{\max}(\mathbf{T}_k)]_k| \le 2 \times |\delta_k|.$$

Hence, the condition $\epsilon_k \le \epsilon$ (with ϵ_k being obtained from Eq.(10)) can be used as a stopping criterion for determining the minimum low order k needed for the desired accuracy ϵ.

Capitalizing on Eq.(8) and Proposition 2, below we provide an algorithm for SimFusion+ computation with accuracy guarantee.

Algorithm. SimFusion+ is shown in Algorithm 1. It takes as input a network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, a desired accuracy ϵ, and a vertex pair (u, v); it returns the approximate similarity $\hat{s}(u, v)$ such that $|\hat{s}(u, v) - s(u, v)| \le \epsilon$ with $s(u, v)$ being the exact value.

Before illustrating the algorithm, we first present the notations it uses. (a) $[\mathbf{T}_k]_{i,j}$ is the (i, j)-entry of the matrix $\mathbf{T}_k$, and $[\sigma_{\max}(\mathbf{T}_k)]_i$ is the i-th entry of the eigenvector $\sigma_{\max}(\mathbf{T}_k)$. (b)

[5]When k is set to $\text{argmin}_k\{\delta_k = 0\}$ (which is practically much smaller than $rank(\mathbf{A})$), $\epsilon_k = \epsilon_{k+1} = \cdots = \epsilon_n = 0$.

[6]As increased by 1 per iteration, the low order parameter k equals the iteration number.

#-line	time	memory	operation
4	$O(m)$	$O(n)$	sparse matrix-vector multiplication
6	$O(n)$	$O(n)$	vector dot product
7	$O(n)$	$O(n)$	vector addition and scalar multiplication
8	$O(n)$	$O(n)$	computing the 2-norm of a vector
9	$O(k)$	$O(k)$	using the power iteration
10	$O(1)$	$O(k)$	getting the vector component
12	$O(n)$	$O(n)$	scaling the vector

Table 1: Running Time & Memory Space Required per Iteration for Algorithm SimFusion+ in Lines 4-12

$span\{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ is the set of all linear combinations of vectors $\mathbf{v}_1, \cdots, \mathbf{v}_k$. (c) $\hat{\sigma}_{\max}(\mathbf{A})$ denotes the approximation of $\sigma_{\max}(\mathbf{A})$.

The algorithm SimFusion+ works as follows. It first computes $\mathbf{A}$ and initializes $\mathbf{v}_1$ (lines 1-2). Using $\mathbf{A}$, it then computes $\mathbf{T}_k$ (lines 4-6), δ_k (lines 7-8) and $\mathbf{v}_{k+1}$ (line 12) by orthonormalizing the vector $\mathbf{A}\mathbf{v}_k$ with respect to $\mathbf{v}_1, \cdots, \mathbf{v}_k$ for every iteration; SimFusion+ also calculates $\sigma_{\max}(\mathbf{T}_k)$ (line 9), and utilizes δ_k and $\sigma_{\max}(\mathbf{T}_k)$ to estimate the error ϵ_k (line 10). The process (lines 3-13) iterates until $\epsilon_k \le \epsilon$, *i.e.,* the minimum low order k is found *s.t.* ϵ_k meets the desired accuracy ϵ (line 11). For such k, the matrix-vector product $[\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \cdot \sigma_{\max}(\mathbf{T}_k)$ is used to approximate the dominant eigenvector of $\mathbf{A}$, and is memorized to compute $\hat{\sigma}_{\max}(\mathbf{A})$ (line 15). The product of the u-th and v-th entries of $\hat{\sigma}_{\max}(\mathbf{A})$ is collected in $\hat{s}(u, v)$, which is returned as the estimated similarity between vertex u and v (lines 17-18).

EXAMPLE 7. *We show how SimFusion+ estimates the similarity in $\mathcal{G}_1$ of Example 1. Given the desired accuracy $\epsilon = 0.05$, SimFusion+ first initializes the UAM $\mathbf{A}$ (in Example 3). It then iteratively computes $\mathbf{T}_k, \mathbf{v}_{k+1}, \delta_k, \sigma_{\max}(\mathbf{T}_k)$ and ϵ_k as follows:*

k	$\mathbf{T}_k$	$\mathbf{v}_{k+1}$	δ_k	$\sigma_{\max}(\mathbf{T}_k)$	ϵ_k
0	–	$[.447\ .447\ .447\ .447\ .447]^T$	–	–	–
1	$[1.08]$	$[.125\ .750\ -.125\ -.125\ -.625]^T$	*.298*	$[1]$	*.596*
2	$\begin{bmatrix} 1.08 & .298 \\ .298 & .190 \end{bmatrix}$	$[-.089\ .044\ .710\ -.697\ .032]^T$	*.359*	$\begin{bmatrix} .957 \\ .290 \end{bmatrix}$	*.208*
3	$\begin{bmatrix} 1.08 & .298 & 0 \\ .298 & .190 & .359 \\ 0 & .359 & -.083 \end{bmatrix}$	$[-.881\ .329\ .137\ .280\ .135\ .231]^T$	*.231*	$\begin{bmatrix} .945 \\ .316 \\ .090 \end{bmatrix}$	*.041*

The iteration terminates at $k = 3$ because the estimation error $\epsilon_3 = .041 \le \epsilon\,(= .05)$. SimFusion+ then memorizes $\hat{\sigma}_{\max}(\mathbf{A}) = [\mathbf{v}_1|\mathbf{v}_2|\mathbf{v}_3] \cdot \sigma_{\max}(\mathbf{T}_3)$ and returns the similarity $\hat{s}(u, v)$, i.e., the (u, v) entry of $\hat{\mathbf{S}}_3$, as shown in Example 6.

We next analyze the time and space complexity of SimFusion+ .

Running Time. The algorithm consists of two phases: preprocessing (lines 1-16), and on-line query (lines 17-18).

(i) For the preprocessing, (a) it takes $O(m)$ time to compute $\tilde{\mathbf{A}}$ (line 1) and $O(n)$ time to initialize $\mathbf{v}_1$ (line 2). (b) The total time of the **for** loop is analyzed in Table 1 (line 3-13), which is bounded by $O(m + 4n + k + 1)$ for each iteration. (c) It takes $O(kn)$ time to compute $\hat{\sigma}_{\max}(\mathbf{A})$ (line 15). Hence, the total time in this phase is $O(m+k(m+4n+k+1)+kn)$, which is bounded by $O(km)$.

(ii) The on-line query phase (lines 17-18) can be done in constant time for each query by virtue of $\hat{\sigma}_{\max}(\mathbf{A})$ memorization.

Combining (i) and (ii), the query time of SimFusion+ is in $O(1)$, plus an $O(km)$-time precomputation.

Memory Space. (i) In the precomputation, (a) initializing $\tilde{\mathbf{A}}$ and $\mathbf{v}_1$ takes $O(n)$ space (lines 1-2). (b) For each iteration k, the space complexity is analyzed in Table 1, which is bounded by $O(n)$ (line 3-13). (c) As the **for** loop terminates, only $\mathbf{v}_1, \cdots, \mathbf{v}_k$ and $\sigma_{\max}(\mathbf{T}_k)$ are kept in memory, yielding $O(kn + k)$ space; the other intermediate results can be freed (line 14). (d) Computing $\hat{\sigma}_{\max}(\mathbf{A})$ takes $O(k)$ space (line 15). Once computed, $\hat{\sigma}_{\max}(\mathbf{A})$ is memorized, yielding $O(n)$ space; $\mathbf{v}_1, \cdots, \mathbf{v}_k$ and $\sigma_{\max}(\mathbf{T}_k)$ are not used subsequently and thus can be freed (line 16).

(ii) For the on-line query (lines 17-18), $\hat{s}(u, v)$ can be computed in $O(n)$ space with $\hat{\sigma}_{\max}(\mathbf{A})$ memorized.

Taking (i) and (ii) together, the total space is bounded by $O(kn)$.

5. INCREMENTAL SIMFUSION+

For certain applications like social networks, graphs are frequently modified [9]. It is too costly to recalculate similarities every time when edges in the graphs are updated. This motivates us to study the following *incremental SimFusion+ estimating problem*.

Given a network $\mathcal{G}$, the eigen-information in $\mathcal{G}$, and a list $\bar{\mathcal{G}}$ of updates (edge deletions and insertions) to $\mathcal{G}$, it is to compute the new USM $\mathbf{S}'$ in $\mathcal{G}'$. Here $\mathcal{G}'$ is the updated $\mathcal{G}$, denoted by $\mathcal{G} + \bar{\mathcal{G}}$.

The idea is to maximally reuse the eigen-information in $\mathcal{G}$ when computing $\mathbf{S}'$. The observation is that $\bar{\mathcal{G}}$ is often small in practice; hence, $\mathbf{S}'(= \mathbf{S} + \bar{\mathbf{S}})$ is slightly different from $\mathbf{S}$. It is far less costly to find the change $\bar{\mathbf{S}}$ to the old $\mathbf{S}$ than to recalculate the new $\mathbf{S}'$ from scratch. The main result in this section is the following.

THEOREM 1. *The incremental SimFusion+ estimating problem is solvable in $O(\delta n)$ time and $O(n)$ space for every vertex pair, where δ is the number of edges affected by the update $\bar{\mathcal{G}}$.*

As we shall see later, δ captures the size of areas in a graph $\mathcal{G}$ that is affected by updates $\bar{\mathcal{G}}$; hence δ is much smaller than n when $\bar{\mathcal{G}}$ is small. That is, the incremental SimFusion+ can be performed more efficiently than computing similarities in $\mathcal{G}'$. This suggests that we compute the eigenvector of $\mathbf{A}$ in $\mathcal{G}$ once, and then incrementally compute SimFusion+ when $\mathcal{G}$ is updated.

To prove Theorem 1, we first introduce a notion of incremental UAM. We then devise an incremental algorithm for handling batch edge updates with the desired bound.

5.1 Incremental Unified Adjacency Matrix

Consider an old network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$ and a new $\mathcal{G}' = (\mathcal{D}, \mathcal{R}')$.

Incremental UAM. The matrix $\bar{\mathbf{A}}$ is said to be the *incremental UAM* of the update $\bar{\mathcal{G}}$ $(= \mathcal{G}' - \mathcal{G}$, *i.e.,* a list of edge insertions and deletions) *iff* $\bar{\mathbf{A}} = \mathbf{A}' - \mathbf{A}$, where $\mathbf{A}$ and $\mathbf{A}'$ are the UAMs of the old network $\mathcal{G}$ and the new $\mathcal{G}'$, respectively.

Intuitively, the nonzero entries of $\bar{\mathbf{A}}$ can identify the edges in $\mathcal{G}$ that is affected by updates $\bar{\mathcal{G}}$. Typically, $\bar{\mathbf{A}}$ is a sparse matrix when δ is small. Indeed, the number of nonzero entries in $\bar{\mathbf{A}}$ is bounded by $O(\delta n)$, which represents the costs that are inherent to the incremental problem itself, *i.e.,* the amount of work absolutely necessary to be performed for the problem.

Using $\bar{\mathbf{A}}$, we next provide a strategy for incrementally computing SimFusion+ similarity.

PROPOSITION 3. *Given a network $\mathcal{G}$ and an update $\bar{\mathcal{G}}$ to $\mathcal{G}$, let $\mathbf{A}$ be the UAM of $\mathcal{G}$, and $\bar{\mathbf{A}}$ the incremental UAM of $\bar{\mathcal{G}}$. Then the new USM $\mathbf{S}'$ of the new network $\mathcal{G}'$ $(= \mathcal{G} + \bar{\mathcal{G}})$ can be computed as*

$$[\mathbf{S}']_{i,j} = [\boldsymbol{\xi}']_i \cdot [\boldsymbol{\xi}']_j \text{ with } [\boldsymbol{\xi}']_i = [\boldsymbol{\xi}_1]_i + \textstyle\sum_{p=2}^{n} c_p \times [\boldsymbol{\xi}_p]_i$$

$$c_p = \frac{\boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}}{\alpha_p - \alpha_1} \text{ and } \boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1. \tag{11}$$

where $\boldsymbol{\xi}_p$ is the eigenvector of $\mathbf{A}$ corresponding to the eigenvalue α_p with $\|\boldsymbol{\xi}_p\|_2 = 1$, and $\boldsymbol{\xi}_1$ is the dominant eigenvector of $\mathbf{A}$.

(Please refer to the Appendix for a detailed proof.)

The main idea in incrementally computing $\mathbf{S}'$ is to reuse $\bar{\mathbf{A}}$ and the eigen-pair $(\alpha_p, \boldsymbol{\xi}_p)$ of the original $\mathbf{A}$. From the computational perspective, memorization techniques can be applied to Eq.(11) for an extra speed-up in computing $[\boldsymbol{\xi}']_i$. Once $\boldsymbol{\eta}$ is computed, it can be memorized for computing $c_2, \cdots, c_n$. When $c_2, \cdots, c_n$ are calculated, they can be memorized for computing $[\boldsymbol{\xi}']_i$ and $[\boldsymbol{\xi}']_j$.

Algorithm 2: IncSimFusion+ $(\mathcal{G}, \mathbf{A}, (\alpha_p, \boldsymbol{\xi}_p), \bar{\mathcal{G}}, (u, v))$

```
Input  : Network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, the old UAM $\mathbf{A}$ of $\mathcal{G}$,
         eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$, the update $\bar{\mathcal{G}}$ to $\mathcal{G}$, query $(u, v)$.
Output: New similarity score $s'(u, v)$.
1  compute the incremental UAM $\bar{\mathbf{A}}$ for the update $\bar{\mathcal{G}}$ :
     $\bar{\mathbf{A}} \leftarrow$ UpdateA $(\mathcal{G}, \mathbf{A}, \bar{\mathcal{G}})$ ;
2  initialize $a \leftarrow [\boldsymbol{\xi}_1]_u, \quad b \leftarrow [\boldsymbol{\xi}_1]_v$ ;
3  compute $\boldsymbol{\eta} \leftarrow \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1$ ;
4  free $\bar{\mathbf{A}}, \ \boldsymbol{\xi}_1$;
5  for $p \leftarrow 2, \cdots, n$ do
6      compute $t \leftarrow \boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}, \quad c_p \leftarrow t/(\alpha_p - \alpha_1)$ ;
7      compute $a \leftarrow a + c_p \times [\boldsymbol{\xi}_p]_u, \quad b \leftarrow b + c_p \times [\boldsymbol{\xi}_p]_v$ ;
8      free $\alpha_p, \ \boldsymbol{\xi}_p$ ;
9  free $\boldsymbol{\eta}, \ t$ ;
10 compute $s'(u, v) \leftarrow a \times b$ ;
11 return $s'(u, v)$ ;
```

5.2 An Incremental Algorithm for SimFusion+

We next prove Theorem 1 by providing an incremental algorithm, referred to as IncSimFusion+, for handling δ edge updates.

Algorithm. The algorithm accepts as input a network $\mathcal{G}$, the UAM $\mathbf{A}$ of $\mathcal{G}$, the eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$, an update $\bar{\mathcal{G}}$ (a list of edge insertions and deletions) to $\mathcal{G}$, and a vertex pair (u, v).

It works as follows. (a) IncSimFusion+ first computes the incremental UAM $\bar{\mathbf{A}}$ for the update $\bar{\mathcal{G}}$ by using procedure UpdateA (line 1). UpdateA incrementally finds all the changes to the old UAM $\mathbf{A}$ in the presence of a list of edge updates to $\mathcal{G}$. (b) For the given vertex pair (u, v), IncSimFusion+ initializes a and b based on the dominant eigenvector $\boldsymbol{\xi}_1$ of $\mathbf{A}$ (line 2) ; it computes $\boldsymbol{\eta}$ once and memorizes $\boldsymbol{\eta}$ for computing $c_2, \cdots, c_n$ (line 3). (c) Once computed, $c_2, \cdots, c_n$ are memorized for calculating the u-th and v-th entries of the dominant eigenvector $\boldsymbol{\xi}'$ of the new UAM,which is collected in a and b, respectively (lines 4-9). IncSimFusion+ returns $a \times b$ as the similarity $\hat{s}(u, v)$ (lines 10-11).

Edge Update. The procedure UpdateA is used for incrementally updating the UAM $\bar{\mathbf{A}}$ by virtue of $\bar{\mathcal{G}}$. An update $\bar{\mathcal{G}}$ is represented as a sequence of 2-tuples $(\mathcal{D} \times \mathcal{D}, \mathsf{op})$ that records every single action of the edge update, in which $\mathcal{D} \times \mathcal{D}$ is a set of δ edges to be inserted or deleted, and op is either "+" (edge insertion) or "−" (edge deletion). For instance, after the edge (P_3, P_5) is added and (P_1, P_2) is removed from $\mathcal{G}_1$ in Fig.1, the update $\bar{\mathcal{G}}$ is denoted by

$$\bar{\mathcal{G}} = \{(P_3, P_5, +), (P_1, P_2, -)\}.$$

UpdateA identifies the incremental $\bar{\mathbf{A}}$ in two phases. (i) It first finds the affected nodes and the data spaces for each edge update in $\bar{\mathcal{G}}$ using a breadth-first search. (ii) It then updates the corresponding entries of $\bar{\mathbf{A}}$ based on the following. We abuse the notation $\mathcal{N}_{\mathcal{D}}(u)$ to denote all the neighbors of object u in the data space $\mathcal{D}$, *i.e.,*

$$\mathcal{N}_{\mathcal{D}}(u) = \{v \in \mathcal{D} | \ (u, v) \in \mathcal{R}\}.$$

Based on the partition of the entire data space $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$ with $n_i = |\mathcal{D}_i|$, the incremental UAM $\bar{\mathbf{A}}$ can be accordingly partitioned into N^2 submatrices $\bar{\mathbf{A}}_{i,j}$.

- For each edge insertion $(u, v, +) \in \bar{\mathcal{G}}$ with $u \in \mathcal{D}_i$ and $v \in \mathcal{D}_j$, (i) we set all entries of $[\bar{\mathbf{A}}_{i,j}]_{u,\star}$ to $-\frac{\lambda_{i,j}}{n_j}$ except for their v-th entries to 0 if $\mathcal{N}_{\mathcal{D}_j}(u) = \varnothing$; (ii) we set all entries of $[\bar{\mathbf{A}}_{j,i}]_{\star,v}$ to $-\frac{\lambda_{j,i}}{n_i}$ except for their u-th entries to 0 if $\mathcal{N}_{\mathcal{D}_i}(v) = \varnothing$; (iii) we set $[\bar{\mathbf{A}}_{i,j}]_{u,v} = \lambda_{i,j}$ otherwise.
- For each edge deletion $(u, v, -) \in \bar{\mathcal{G}}$ with $u \in \mathcal{D}_i$ and $v \in \mathcal{D}_j$, (i) we set all entries of $[\bar{\mathbf{A}}_{i,j}]_{u,\star}$ to $\frac{\lambda_{i,j}}{n_j}$ except for their v-th entries to 0 if $|\mathcal{N}_{\mathcal{D}_j}(u)| = 1$; (ii) we set all entries of $[\bar{\mathbf{A}}_{j,i}]_{\star,v}$ to $\frac{\lambda_{j,i}}{n_i}$ except for their u-th entries to 0 if $|\mathcal{N}_{\mathcal{D}_i}(v)| = 1$; (iii) we set $[\bar{\mathbf{A}}_{i,j}]_{u,v} = -\lambda_{i,j}$ otherwise.

Complexity. The algorithm IncSimFusion+ is in $O(\delta n)$ time and $O(n)$ space for handling δ edge updates in $\bar{\mathcal{G}}$. (i) The procedure UpdateA can be bounded by $O(\delta n)$ time and $O(n)$ space (line 1). For each edge update in $\bar{\mathcal{G}}$, it is in at most $O(n)$ time and $O(n)$ intermediate space to update the corresponding entries of $\bar{\mathbf{A}}$. (ii) Computing $\boldsymbol{\eta}$ requires an $O(\delta)$-time and $O(n)$-space sparse matrix-vector multiplication $\bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1$ (line 3). (iii) For every c_p, it takes $O(\delta)$ time and $O(n)$ space to calculate $\boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}$ (line 6) since $\boldsymbol{\eta}$ is a sparse vector with only $O(\delta)$ nonzeros; and $c_2, \cdots, c_p$ are memorized for computing $[\boldsymbol{\xi}']_i$, which requires $O(\delta n)$ time and $O(n)$ space in total. (iv) Computing a, b and $s'(u, v)$ needs constant time and space (lines 7 and 10). Thus, combining (i)-(iv) , the total complexity is bounded by $O(\delta n)$ time and $O(n)$ space.

EXAMPLE 8. *We show how IncSimFusion+ works. Consider the graph $\mathcal{G}_1$ in Fig.1 with its UAM $\mathbf{A}$ in Example 3. Suppose two edges (P_1, P_2) and (P_2, P_1) are removed from $\mathcal{G}_1$. The new USM $\mathbf{S}'$ is updated as follows.*

First, UpdateA is invoked for precomputing the incremental $\bar{\mathbf{A}}$ and the eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$ in an off-line fashion:

$$\bar{\mathbf{A}} = \begin{bmatrix} 0 & -\frac{1}{6} & 0 & 0 & 0 \\ -\frac{1}{6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

p	α_p	$\boldsymbol{\xi}_p$	c_p
1	*1.184*	$[.431\ .673\ .451\ .322\ .232]^T$	–
2	*.503*	$[.708\ -.522\ -.242\ .388\ .132]^T$	*.062*
3	*-.480*	$[-.256\ -.020\ .095\ .716\ -.641]^T$	*-.018*
4	*-.366*	$[-.021\ -.507\ .853\ -.119\ .017]^T$	*-.025*
5	*.242*	$[.497\ .127\ .037\ -.467\ -.719]^T$	*.069*

IncSimFusion+ next computes $\boldsymbol{\eta}$ from $\bar{\mathbf{A}}$ and $\boldsymbol{\xi}_1$ (line 3):

$$\boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1 = [-.112 \quad -.072 \quad 0 \quad 0 \quad 0]^T.$$

Then, c_p can be derived from the memorized $\boldsymbol{\eta}$ and $(\alpha_p, \boldsymbol{\xi}_p)$, e.g.,

$$\begin{aligned} c_2 &= \boldsymbol{\xi}_2^T \cdot \boldsymbol{\eta}/(\alpha_2 - \alpha_1) = -.0419/(.503 - 1.184) = .062, \\ c_3 &= \boldsymbol{\xi}_3^T \cdot \boldsymbol{\eta}/(\alpha_3 - \alpha_1) = .030/(-.480 - 1.184) = -.018. \end{aligned}$$

Once computed, $c_2, \cdots, c_5$ are memorized for calculating $[\boldsymbol{\xi}']_\star$:

$$\boldsymbol{\xi}' = \boldsymbol{\xi}_1 + \sum\nolimits_{p=2}^{5} c_p \times \boldsymbol{\xi}_p = [.327\ .703\ .485\ .326\ .266]^T.$$

Hence, applying $[\boldsymbol{\xi}']_\star$ to the new USM $\mathbf{S}'$ (line 10) yields

$$\mathbf{S}' = \begin{bmatrix} .107 & .230 & .159 & .107 & .087 \\ .230 & .494 & .341 & .230 & .187 \\ .159 & .341 & .235 & .158 & .129 \\ .107 & .230 & .158 & .107 & .087 \\ .087 & .187 & .129 & .087 & .071 \end{bmatrix}.$$

6. EXPERIMENTAL EVALUATION

In this section, a comprehensive empirical study of the proposed similarity estimating methods is presented.

6.1 Experimental Setting

Datasets. We used three real-life datasets and a synthetic dataset.

(1) MSN *Data.* [7] The MSN search log data were taken from "Microsoft Live Labs: Accelerating Search in Academic Research". This dataset was also used in the prior work [1]. It contains about 15M user queries from the United States in May 2006 and the corresponding clickthrough URLs. The dataset was formatted by showing each query, the URLs of the associated web pages, and the number of clickthroughs by query, as depicted below.

[7]http://research.microsoft.com/ur/us/fundingopps/RFPs/Search_2006_RFP.aspx

Query	URL	Clicks	URL	Clicks
Shopping	shopping.yahoo.com	2,375	www.ebay.com	1,859

The 15K most common queries in the search log were chosen, and the hyperlinks from the contents of the top 32K popular web pages were parsed. We built a network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, which consists of a web page space $\mathcal{D}_w$ and a query space $\mathcal{D}_q$.

(2) DBLP *Data.* [8] This dataset was derived from a snapshot of the computer science bibliography (from 2001 to 2010). We selected the research papers published in the following conference proceedings: "SIGIR", "KDD", "VLDB", "ICDE", "SIGMOD" and "WWW". Choosing a time step of two years, we built 5 DBLP web graphs $\mathcal{G}_i$ $(i = 1, \cdots, 5)$ with the sizes listed below:

	$\mathcal{G}_1$: 01-02	$\mathcal{G}_2$: 01-04	$\mathcal{G}_3$: 01-06	$\mathcal{G}_4$: 01-08	$\mathcal{G}_5$: 01-10
$\lvert\mathcal{D}\rvert$	1,838	3,723	5,772	9,567	12,276
$\lvert\mathcal{R}\rvert$	7,103	14,419	29,054	45,310	64,208

For each graph $\mathcal{G}_i = (\mathcal{D}_i, \mathcal{R}_i)$, two data spaces were used: paper space $\mathcal{D}_p^i$ and author space $\mathcal{D}_a^i$.

(3) WebKB *Data.* [9] This dataset collects web pages from the computer science departments of four universities: Cornell (CO), Texas (TE), Washington (WA) and Wisconsin (WI). It was also used in the previous work [13] for link-based similarity estimation. For each university, a network $\mathcal{G}_{U_i} = (\mathcal{D}_i, \mathcal{R}_i)$ was built, in which (a) the web pages in $\mathcal{D}_i$ were classified into 7 categories (data spaces): student, faculty, staff, department, course, project and others, and (b) the UAM of $\mathcal{R}_i$ represented the hyperlink adjacency matrix. The sizes of these networks are as follows:

	U_1: CO	U_2: TE	U_3: WA	U_4: WI
$\lvert\mathcal{D}\rvert$	867	827	1,263	1,205
$\lvert\mathcal{R}\rvert$	1,496	1,428	2,969	1,805

(4) Synthetic Data. The data were produced by the C++ boost graph generator, with 2 parameters: the number of vertices and the number of edges. Varying the graph parameters, we used this dataset to represent homogenous networks for an in-depth analysis.

Compared Algorithms. The following algorithms were implemented in C++: (1) SimFusion+ and IncSimFusion+ ; (2) SF, a SimFusion algorithm via matrix iteration [1]; (3) CSF, a variant of SF, which leverages PageRank stationary distribution [13]; (4) SR, a SimRank algorithm via partial sums function [8]; (5) PR, a Penetrating-Rank algorithm encoding both in- and out-links [4].

Evaluation Metrics. For evaluating the performance of the algorithms, we used *Normalized Discounted Cumulative Gain* (NDCG) metrics [13]. The NDCG at a rank position p is defined as $\text{NDCG}_p = \frac{1}{\text{IDCG}_p} \sum_{i=1}^{p} \frac{2^{\text{rank}_i} - 1}{\log_2(1+i)}$, where rank_i is the graded relevance of the similarity result at rank position i, and IDCG_p is the normalization factor to guarantee that NDCG of a perfect ranking at position p equals 1.

Twelve IT experts were hired to judge the similarity of the five algorithms. The final judgment was rendered by a majority vote.

All experiments were run on a machine with a Pentium(R) Dual-Core (2.00GHz) CPU and 4GB RAM, using Windows Vista. The algorithms were implemented in Visual C++. Each experiment was repeated over 5 times, and the average is reported here.

6.2 Experimental Results

6.2.1 Accuracy

We first evaluated the accuracy of SimFusion+ *vs.* SF, CSF, SR and PR in estimating the similarity, using real-life data.

We randomly chose 50 queries and 40 pages from the MSN query log, and compared the average NDCG_{10} (and NDCG_{30}) of the five algorithms. The results are shown in Figure 4, in which the x-axis categorizes the objects according to query and web page. We find the following. (i) In most cases, SF seems hardly to get sensible similarities because with the increasing number of iterations, all the similarities of SF will asymptotically approach the same value. This verifies the convergence issue of the original model [1]. (ii) When SF did not fail, SimFusion+ always gave more accurate estimation on average than the other algorithms. For instance, for the top 10 queries, the average NDCG_{10} of SimFusion+ (0.79) is 10x better than SF (0.07), 39% better than CSF (0.57), 58% better than SR (0.50), and 15% better than PR (0.69), whereas for the top 30 web pages, the average NDCG_{30} of SimFusion+ (0.64) is 12x better than SF (0.05), 45% better than CSF (0.44), 33% better than SR (0.48), and 21% better than PR (0.53). This is because substituting UAM for URM effectively avoids divergent or trivial solutions, thus improving the quality and reliability of SimFusion+ similarity, as expected.

[8]http://www.informatik.uni-trier.de/~ley/db/

[9]http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

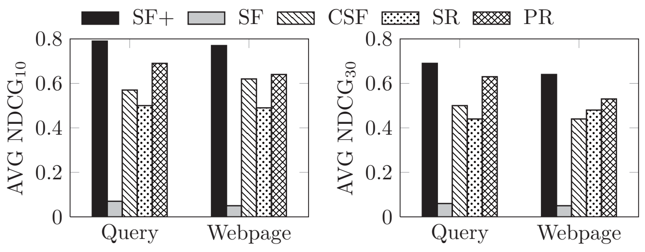


Figure 4: Comparing SimFusion+ with other ranking algorithms for the average NDCG_{10} and NDCG_{30} on MSN data

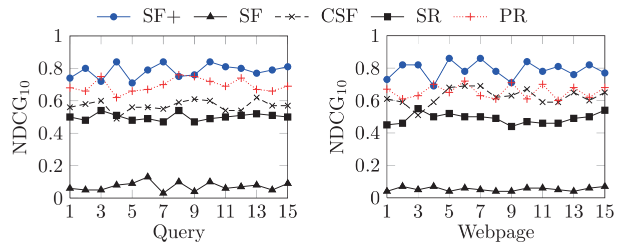


Figure 5: The detailed query-by-query and page-by-page comparisons for NDCG_{10} on MSN data

To further verify the accuracy, we randomly selected another 15 queries and 15 web pages from MSN data. In Figure 5, the query-by-query and page-by-page comparisons are shown for NDCG_{10} of the five algorithms. We observe that (i) for 12 out of 15 queries, SimFusion+ achieved highest accuracy of the five algorithms;for 13 out of 15 web pages, SimFusion+ outperformed the other algorithms in its accuracy, and was slightly less accurate than PR for only 2 pages. (ii) For all the queries and web pages, SimFusion+ showed the best accuracy performance on average, PR the second, and SF the worst. This is because SimFusion+ uses UAM to encode the intra- and inter-relations in a comprehensive way, thus making the results unbiased.

We also evaluated the performance of SimFusion+ on DBLP and WebKB datasets. In DBLP experiments, 20 authors were randomly chosen from $\mathcal{G}_1$:01-02, $\mathcal{G}_3$:01-06 and $\mathcal{G}_5$:01-10 data, respectively. We compared the similarity of the top 10 authors in $\mathcal{G}_i$ $(i = 1, 3, 5)$ estimated by the five algorithms. The results of the average NDCG_{10} are depicted in Figure 6. It can be seen that SimFusion+ again achieved better accuracy on DBLP data. For instance, SimFusion+ (0.88) on $\mathcal{G}_3$:01-06 was 13x better than SF (0.06), 95% better than CSF (0.45), 26% better than SR (0.7), and 19% better than PR (0.74). In WebKB experiments, we computed NDCG within 10 web pages for each object in each university data (CO,TE,WI,WA) and evaluated the average scores. Figure

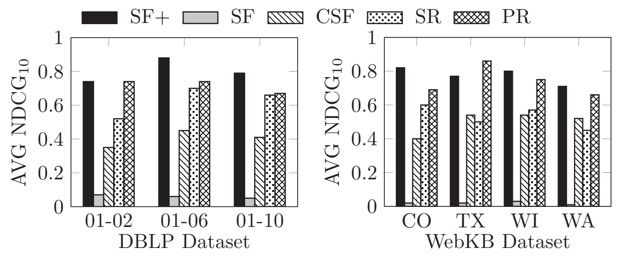


Figure 6: Comparing SimFusion+ with other ranking algorithms for the average NDCG$_{10}$ on DBLP and WEBKB data

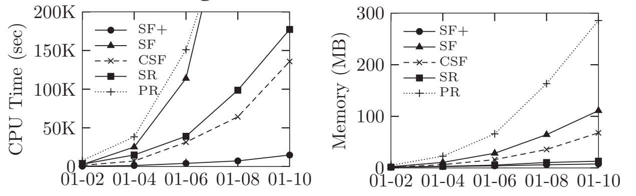


Figure 7: Comparing the CPU time and memory of the ranking algorithms on DBLP data

6 shows that SimFusion+ outperformed the other 4 algorithms on CO, WI and WA data, except that PR (0.8) did 6% better than SimFusion+ (0.75) on TX data. This tells that SimFusion+ accuracy performance is consistently stable on different experimental datasets.

6.2.2 *CPU Time & Memory Space*

We then evaluated the running time and memory space efficiency of SimFusion+ , SF, CSF, SR and PR using real datasets.

Figure 7 shows the CPU time and memory consumption for the five algorithms on DBLP. The total time and memory for each algorithm showed an increasing tendency with the growing size of DBLP. It can be noted that the time for SimFusion+ was at least one order of magnitude faster than CSF and SR on average, and more than 20x faster than PR and SF, whereas the space for SimFusion+ and SR increased linearly with the size of DBLP, in contrast with the quadratic increase in memory for SF, CSF, PR, as expected. This drastic speedup and decrease in RAM is due to the memorization of $\sigma_{\max}(\mathbf{T}_k)$ for computing USM, thus saving much time and space for repetitive matrix multiplications.

To further evaluate the efficiency, we compare the time and memory of the five ranking algorithms on WEBKB. In Figure 8, the results indicate that SimFusion+ took about 10x less time than SF and PR, and 6x less time than CSF and SR on average. The memory space for SimFusion+ was also efficient and scaled well with the size of WEBKB. It can be seen that SF also took small memory space (approx. 1M) when the data size was small (*e.g.,* CO and TX). However, when the data size increased (*e.g.,* WI and WA), SF was less useful since large memory storage (about 2.5M) was required to keep the intermediate result of the k-th iterative USM. In all the cases, SimFusion+ performed the best.

On large datasets, the effect of SimFusion+ is even more pronounced. Figure 9 reports the average time and memory of the five algorithms on MSN data, in which the y-axis is log-scale. We chose 50 queries and 40 web pages from $\mathcal{D}_q$ and $\mathcal{D}_w$, respectively. For each $o_q \in \mathcal{D}_q$ (*resp.* $o_w \in \mathcal{D}_w$), we estimated the similarity between o_q (*resp.* o_w) and other object $o \in \mathcal{D}_q \cup \mathcal{D}_w$. We see that SimFusion+ was highly efficient on large datasets (nearly 2 orders of magnitude than SF and PR, and 1 order of magnitude than CSF in both time and space). This validates that the performance of SimFusion+ is fairly stable among different datasets.

6.2.3 *Incremental Performance*

We next evaluated the incremental performance of IncSimFusion+ on real datasets. Given a sequence of edge updates (δ edge insertions and deletions in $\bar{\mathcal{G}}$), we compared the running time of IncSimFusion+ with that of SimFusion+ ; the latter had to recalculate the UAM when edges were updated, and the computational cost was counted. In these experiments, the eigen information of the old UAM can be preconditioned in an off-line fashion and shared by all the updated graphs for incremental computation, and hence their costs were not counted at the query stage. Due to space limitations, below we only reported the results on MSN dataset.

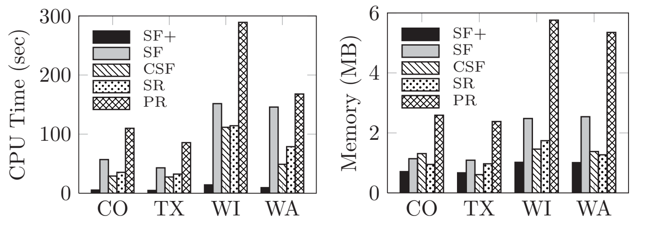


Figure 8: Comparing the CPU time and memory of the ranking algorithms on WEBKB data

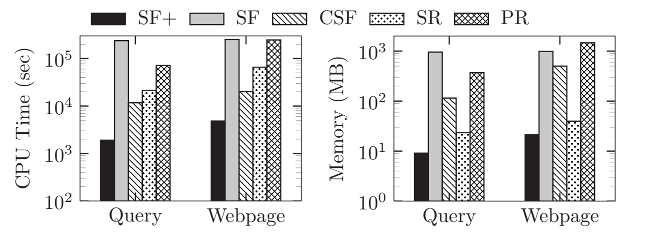


Figure 9: Comparing the CPU time and memory of the ranking algorithms for the given query and web page on MSN data

Varying δ (the number of the edges to be updated) from 600 to 3000, we first evaluated the running time of IncSimFusion+ and SimFusion+ over MSN data, respectively, for estimating all the similarities between the query objects in $\mathcal{D}_q$. Figure 10 shows that IncSimFusion+ outperformed SimFusion+ when $\delta < 2800$, but SimFusion+ performed better for larger δ, as expected. This is because the small value of δ often preserves the sparseness of the incremental UAM, and hence reduces the computational cost of $\boldsymbol{\eta}$ when the USM was incrementally updated.

To further validate the performance of IncSimFusion+ , we estimated the similarity among the web page in $\mathcal{D}_w$ and tested its CPU time on MSN data. In Figure 10, the result shows that IncSimFusion+ was highly efficient for the small number of edge updates. When $\delta > 7700$, SimFusion+ did better than IncSimFusion+ . This tells that increasing the number of updated edges induces more nonzeros in $\bar{\mathbf{A}}$, thus increasing the difficulty of incremental computation. We also noticed that the SimFusion+ time was less sensitive to the small number of updated edges, whereas the IncSimFusion+ time was linearly increased with δ, as expected. This is because once the edges are changed, SimFusion+ has to recompute all the similarities from scratch. In contrast, IncSimFusion+ only computes the similarities from the affected area of edge updates. In light of this, IncSimFusion+ scales well with δ.

6.2.4 *Effect of* ϵ

We used 9 web graphs with the number of vertices increased from 600K to 1.4M. We varied ϵ from 0.01 to 0.0001 and ran SimFusion+ on each graph. The results are reported in Figure 11. It can be seen that the computational time and memory consumption for SimFusion+ was sensitive to ϵ. The smaller the ϵ is, the larger amounts of the CPU time and memory space are, as expected. These confirmed our observation in Section 4, where we envisage that the small choice of ϵ imposes more iterations on computing $\mathbf{T}_k$ and $\mathbf{v}_k$, and hence increases the estimation costs.

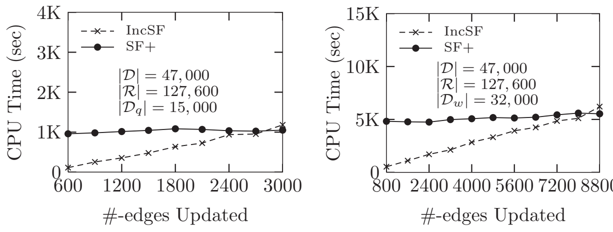


Figure 10: Comparing the CPU time of IncSimFusion+ with that of SimFusion+ on MSN data

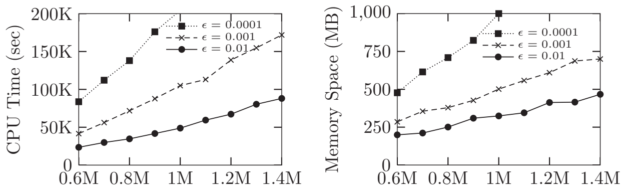


Figure 11: Effect of ϵ for SimFusion+ on synthetic data

7. CONCLUSIONS

We present SimFusion+, a revision of SimFusion, for preventing the trivial solution and the divergence issue of the SimFusion model. We propose efficient techniques to improve the time and space complexity of SimFusion+ computation with accuracy guarantees. We also devise an incremental algorithm to compute SimFusion+ similarity on dynamic graphs when edges are frequently updated. The empirical results on both real and synthetic datasets show that our methods achieve high performance and result quality.

We are currently studying the vertex-updating methods for incrementally computing SimFusion+. We are also to extend our techniques to parallel SimFusion+ computing on GPU.

8. REFERENCES

[1] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and D. Zhuang, "SimFusion: Measuring similarity using unified relationship matrix," in *SIGIR*, pp. 130–137, 2005.
[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," technical report, Stanford InfoLab, November 1999.
[3] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *KDD*, pp. 538–543, 2002.
[4] P. Zhao, J. Han, and Y. Sun, "P-Rank: A comprehensive structural similarity measure over information networks," in *CIKM*, pp. 553–562, 2009.
[5] H. Small, "Co-citation in the scientific literature: A new measure of the relationship between two documents," *J. Am. Soc. Inf. Sci.*, vol. 24, no. 4, pp. 265–269, 1973.
[6] B. Jarneving, "Bibliographic coupling and its application to research-front and other core documents," *J. Informetrics*, vol. 1, no. 4, pp. 287–307, 2007.
[7] W. Xi, B. Zhang, Z. Chen, Y. Lu, S. Yan, W. Ma, and E. A. Fox, "Link Fusion: A unified link analysis framework for multi-type interrelated data objects," in *WWW*, pp. 319–327, 2004.
[8] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for SimRank computation," *VLDB J.*, vol. 19, no. 1, pp. 45–66, 2010.
[9] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu, "Fast computation of SimRank for static and dynamic information networks," in *EDBT*, pp. 465–476, 2010.
[10] G. He, H. Feng, C. Li, and H. Chen, "Parallel SimRank computation on large graphs with iterative aggregation," in *KDD*, pp. 543–552, 2010.
[11] W. Yu, W. Zhang, X. Lin, Q. Zhang, and J. Le, "A space and time efficient algorithm for SimRank computation," *World Wide Web*, vol. 15, no. 3, pp. 327–353, 2012.
[12] G. Xue, H. Zeng, Z. Chen, Y. Yu, W. Ma, W. Xi, and E. A. Fox, "MRSSA: An iterative algorithm for similarity spreading over interrelated objects," in *CIKM*, pp. 240–241, 2004.
[13] Y. Cai, M. Zhang, C. H. Q. Ding, and S. Chakravarthy, "Closed form solution of similarity algorithms," in *SIGIR*, pp. 709–710, 2010.
[14] G. Williams, *Linear Algebra with Applications*. Jones and Bartlett Publishers, 2007.
[15] A. J. Laub, *Matrix Analysis for Scientists and Engineers*. SIAM: Society for Industrial and Applied Mathematics, 2004.
[16] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, February 1990.
[17] Y. Saad, *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, 2 ed., April 2003.

Appendix: Proofs

Proof of Proposition 2.

PROOF. Let $\boldsymbol{\psi}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \alpha(\mathbf{x}) \cdot \mathbf{x}$ be a vector function of $\mathbf{x} \in \mathbb{R}^n$, with $\alpha(\mathbf{x})$ being a real function of $\mathbf{x}$. To simplify notations, we shall denote by α_k the dominant eigenvalue of $\mathbf{T}_k$, and

$$\boldsymbol{\eta}_k = \sigma_{\max}(\mathbf{T}_k), \;\; \boldsymbol{\xi}_k = \mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k), \;\; \boldsymbol{\xi} = \sigma_{\max}(\mathbf{A}).$$

Using $\mathbf{T}_k\boldsymbol{\eta}_k = \alpha_k\boldsymbol{\eta}_k$ and Eq.(7) in Lemma 1, we have

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{\xi}_k) &= \mathbf{V}_k\mathbf{T}_k\boldsymbol{\eta}_k + \delta_k\mathbf{v}_{k+1}\mathbf{e}_k^T\boldsymbol{\eta}_k - \alpha_k\mathbf{V}_k\boldsymbol{\eta}_k \\ &= \delta_k\mathbf{v}_{k+1}(\mathbf{e}_k^T\boldsymbol{\eta}_k) = \delta_k[\boldsymbol{\eta}_k]_k\mathbf{v}_{k+1},\end{aligned}$$

where $[\boldsymbol{\eta}_k]_k$ denotes the k-th entry of $\boldsymbol{\eta}_k$. Hence,

$$\|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \le \|\boldsymbol{\psi}(\boldsymbol{\xi}_k)\|_2 = \left|\delta_k[\boldsymbol{\eta}_k]_k\right|\|\mathbf{v}_{k+1}\|_2 = \left|\delta_k[\boldsymbol{\eta}_k]_k\right|.$$

Since $vec(\mathbf{S}_k) = \boldsymbol{\xi}_k \otimes \boldsymbol{\xi}_k$ and $vec(\mathbf{S}) = \boldsymbol{\xi} \otimes \boldsymbol{\xi}$, we have

$$\begin{aligned}\|\mathbf{S}_k - \mathbf{S}\|_2 &= \|vec(\mathbf{S}_k) - vec(\mathbf{S})\|_2 = \|\boldsymbol{\xi}_k \otimes \boldsymbol{\xi}_k - \boldsymbol{\xi} \otimes \boldsymbol{\xi}\|_2 \\ &= \|\boldsymbol{\xi}_k \otimes (\boldsymbol{\xi}_k - \boldsymbol{\xi}) + (\boldsymbol{\xi}_k - \boldsymbol{\xi}) \otimes \boldsymbol{\xi}\|_2 \\ &\le \underbrace{\|\boldsymbol{\xi}_k\|_2}_{\le 1} \cdot \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 + \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \cdot \underbrace{\|\boldsymbol{\xi}\|_2}_{\le 1} \\ &= 2 \times \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \le 2 \times |\delta_k \times [\boldsymbol{\eta}_k]_k|,\end{aligned}$$

which completes the proof. $\square$

Proof of Proposition 3.

PROOF. For the new graph $\mathcal{G}'$, let $\boldsymbol{\xi}'$ be the dominant eigenvector of $\mathbf{A}'$ with its eigenvalue α', and $\bar{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}' - \boldsymbol{\xi}_1$, $\bar{\alpha}_1 = \alpha' - \alpha_1$, $\bar{\mathbf{A}} = \mathbf{A}' - \mathbf{A}$. Then, $\mathbf{A}'\boldsymbol{\xi}' = \alpha'\boldsymbol{\xi}'$ can be rewritten as

$$(\mathbf{A} + \bar{\mathbf{A}})(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1) = (\alpha_1 + \bar{\alpha})(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1).$$

Expanding the above equation, eliminating $\bar{\mathbf{A}}\bar{\boldsymbol{\xi}}_1$ and $\bar{\alpha}_1\bar{\boldsymbol{\xi}}_1$ (the high-order infinitesimals of $\bar{\boldsymbol{\xi}}_1$), and using $\mathbf{A}\boldsymbol{\xi}_1 = \alpha_1\boldsymbol{\xi}_1$, we have

$$\mathbf{A}\bar{\boldsymbol{\xi}}_1 + \boldsymbol{\eta} = \alpha_1\bar{\boldsymbol{\xi}}_1 + \bar{\alpha}_1\boldsymbol{\xi}_1 \text{ with } \boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1. \qquad (12)$$

Since $\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_n$ of $\mathbf{A}$ constitute a basis for $\mathbb{R}^n$, there exist scalars $c_1, \cdots, c_n$ *s.t.* $\bar{\boldsymbol{\xi}}_1 = c_1\boldsymbol{\xi}_1 + \cdots + c_n\boldsymbol{\xi}_n$. Substituting this into Eq.(12) and pre-multiplying both sides by $\boldsymbol{\xi}_j^T$ produce

$$\boldsymbol{\xi}_j^T \sum_{i=1}^n c_i\alpha_i\boldsymbol{\xi}_i + \boldsymbol{\xi}_j^T\boldsymbol{\eta} = \alpha_1\boldsymbol{\xi}_j^T \sum_{i=1}^n c_i\boldsymbol{\xi}_i + \bar{\alpha}_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_1. \quad (j = 2, \cdots, n)$$

Due to $\boldsymbol{\xi}_i$ orthonormality $(i = 1, \cdots, n)$, we have

$$\underbrace{c_j\alpha_j\boldsymbol{\xi}_j^T\boldsymbol{\xi}_j}_{=c_j\alpha_j} + \boldsymbol{\xi}_j^T\boldsymbol{\eta} = \underbrace{c_j\alpha_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_j}_{=c_j\alpha_1} + \underbrace{\bar{\alpha}_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_1}_{=0}.$$

Hence, $c_j = (\boldsymbol{\xi}_j^T\boldsymbol{\eta})/(\alpha_j - \alpha_1)$ with $\boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1 \quad (j = 2, \cdots, n)$.

To determine c_1, we use the identity $(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1)^T(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1) = 1$. Expanding the left-hand side, eliminating the high-order infinitesimal $\bar{\boldsymbol{\xi}}_1^T\bar{\boldsymbol{\xi}}_1$, and replacing $\bar{\boldsymbol{\xi}}_1$ with $c_1\boldsymbol{\xi}_1 + \cdots + c_n\boldsymbol{\xi}_n$, we have

$$\underbrace{\boldsymbol{\xi}_1^T\boldsymbol{\xi}_1}_{=1} + \underbrace{\boldsymbol{\xi}_1^T\sum_{i=1}^n c_i\boldsymbol{\xi}_i}_{=c_1} + \underbrace{\sum_{i=1}^n c_i\boldsymbol{\xi}_i^T\boldsymbol{\xi}_1}_{=c_1} = 1.$$

Due to $\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_n$ orthonormality, it follows that $c_1 = 0$. Thus,

$$vec(\mathbf{S}') = \boldsymbol{\xi}' \otimes \boldsymbol{\xi}' \text{ with } \boldsymbol{\xi}' = \boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1 + \sum_{p=2}^n c_p \cdot \boldsymbol{\xi}_p,$$

$$c_p = \frac{\boldsymbol{\xi}_p^T\boldsymbol{\eta}}{\alpha_p - \alpha_1}, \text{ and } \boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1.$$

$\square$