Department of Computer Science Technical Reports

Department of Computer Science

2011

Network Similarity Decomposition (NSD): A Fast and Scalable Approach to Network Alignment

Giorgos Kollias
Purdue University, gkollias@purdue.edu

Ananth Y. Grama
Purdue University, ayg@cs.purdue.edu

Shahin Mohammadi
Purdue University

Report Number:
11-001

Network Similarity Decomposition (NSD): A Fast and Scalable Approach to Network Alignment

Giorgos Kollias, Shahin Mohammadi, and Ananth Grama

Abstract—As graph-structured datasets become commonplace, there is increasing need for efficient ways of analyzing such datasets. These analyses include conservation, alignment, differentiation, and discrimination, among others. When defined on general graphs, these problems are considerably harder than their well-studied counterparts on sets and sequences. This is a direct consequence of the underlying isomorphism associated with many of these problems. In this paper, we study the problem of global alignment of large sparse graphs. Specifically, we investigate efficient methods for computing pairwise topological similarity between nodes in two networks (or within the same network). Pairs of nodes with high similarity can be used to seed global alignments.
We present a novel approach to this computationally expensive problem based on *un-coupling* and *decomposing* ranking calculations associated with computation of similarity scores. *Un-coupling* refers independent pre-processing of each input graph. *Decomposition* implies that pairwise similarity scores can be explicitly broken down into contributions from different link patterns traced back to the initial conditions for the computation. These two concepts result in significant improvements, in terms of computational cost, interpretability of similarity scores, and nature of queries. We show over two orders of magnitude improvement in performance over state-of-the-art IsoRank/ Random Walk formulations, and over an order of magnitude over constrained matrix-triple-product formulations, in the context of real datasets.

Index Terms—H.2.8.d: Data mining; G.1.3.i: Sparse, structured, and very large systems; G.1.3.h: Singular value decomposition

1 Introduction and Motivation

Graph structured datasets are commonly encountered in diverse domains, ranging from biochemical interaction networks, to networks of social and economic transactions. Effective analyses of these datasets hold the potential for significant applications' insight. Compared to data abstracted as sets, sequences, or even graphs with well-defined structure, analyses techniques and software for general large sparse graphs are in relative infancy. Traditional problems relating to conserved components, discriminating components, modularity, clustering, and alignment are more expensive for graph structured data. Many of these problems can be related to (sub)graph isomorphism, which is known to be NP Hard. Consequently, effective techniques must leverage properties of specific datasets to deliver acceptable performance.

Given two graphs, an interesting question one may ask is: "how similar is each node in the first graph to each node in the second?" or "what is the best match for each node in the first graph to nodes in the second graph?". A solution to this problem can be used to align two given networks to identify invariant subgraphs. Note that this is not a solution to the subgraph isomorphism problem (or a homeomorphism) in the general case, since suitable measures of similarity and constraints on mappings must be specified. The construction of a similarity matrix S, where element $s_{i,j}$ denotes the similarity of node i in the first graph to node j in the second graph, depends on the specific measure of node similarity. Similarity measures differ along many dimensions – perhaps, the most relevant here is the topological scope of the measure. Local measures define similarity on the basis of the neighborhood of nodes. Global measures define similarity based on network connectivity patterns over the entire graph. This paper investigates efficient algorithms for computing global similarity measures across two graphs (or a graph with itself). Similarity scores represent the inherent, but latent correspondences between nodes. These scores can serve as seeds for 'growing' conserved subgraphs. In applications such as biochemical pathway analyses, these conserved subgraphs provide important insights into the functional and structural composition of networks.

We initiate our discussion with a formal description of topological similarity of nodes in a network. We draw on an analogy from the problem of identifying "reputed" nodes in a single network – also sometimes called the page-ranking, or node-ranking problem. Perhaps, the most commonly used measure for the rank of a node can be recursively defined as follows: "a node is important if it is linked by other important nodes" [1]. Extending this definition to the node similarity problem, we arrive at the following definition: "two nodes are (topologically) similar if they are linked by other (topologically) similar node pairs" [2], [3]. This analogy can be further extended in application areas like automated image captioning

The authors are with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA
e-mail addresses: gkollias@purdue.edu, mohammas@purdue.edu, ayg@cs.purdue.edu

[4] or synonym extraction [5], where node similarity is modeled as node ranking, inspired by its two dominant models – Page-Rank and HITS [6].

The general notion of node similarities can be extended to account for prior knowledge relating to the networks. To highlight this concept, we introduce notions of elemental similarity and topological similarity. Elemental similarity corresponds to the similarity between two labeled nodes, independent of their link structure. For example, in the context of protein interaction networks, elemental similarity between two nodes may simply correspond to the sequence match between the two proteins. Topological similarity on the other hand combines elemental similarity (where available) with link structure to compute an aggregate measure. We use the term topological similarity and general node similarity interchangeably in this paper.

Elemental similarity can be incorporated into topological similarity scores in different ways. Singh et al. [3] use independent sequence data for protein similarity to augment protein interactions in Protein-Protein-Interaction (PPI) networks. These elemental similarities do not rule out any topological similarity matchings; however, node pairs with higher elemental similarity are favored while computing their topological similarity scores. Alternately, metadata tags on nodes (e.g., the textual content of articles in a corpus), can be used to rule out "irrelevant" node pairs [7]. This in turn can be used to reduce memory and computational overheads significantly. In yet other approaches [5], one cannot enforce exclusion, however, one may specify pair preferences in the initial conditions. In most of these applications, one is not interested in the entire topological similarity matrix, but rather in the best-matching node pairs. This optimization is often formulated as the maximization of some function of the cumulative pairwise topological similarity scores.

To the best of our knowledge, existing approaches to computing node similarity process input graphs simultaneously. A commonly used approach is to compute a sparse (or sparsified) product graph of the input graphs, and to compute node ranks within this product graph. Such approaches, however, have significant computational cost for large graphs, since products of graphs with 10^6 nodes and beyond often become computationally intractable. In this paper, we present a novel approach that allows us to un-couple the processing of input graphs through elemental algebraic manipulations. These manipulations facilitate finer-grained processing to identify similar components across graphs (decomposition) rapidly. In doing so, our approach accelerates node similarity computations by over two orders of magnitude compared to state-of-the-art IsoRank and Random-Walk based approaches. These performance gains are demonstrated on moderate sized graphs derived from real datasets. For larger graphs, these gains can be expected to be significantly higher, as we show using a simple performance model for our algorithm.

The rest of this paper is organized as follows: we introduce basic terminology and background in Section 2.2. We use this background to motivate and describe our proposed algorithm, Network Similarity Decomposition (NSD) in Section 3. We show how our proposed algorithm *un-couples* and *decomposes* the similarity computation kernel of Singh and Berger [3]. Experimental results from our approach on real datasets (biological, social and Web networks) are presented in Section 4. Concluding remarks and avenues for future work are summarized in Section 5.

2 Background and Notations

2.1 Related Results

There are two sets of results on graph analyses that are of particular relevance to our proposed work. The first set targets the ranking of a node in a single network. These results are primarily motivated by applications in information retrieval and web-search. For this reason, nodes are often referred to as "pages" (for web pages), and node ranks often correspond to the "reputation" or "importance" of a page. The second, more recent set of results, targets the correspondence, or similarity, between nodes across multiple networks. These results are motivated by diverse applications, ranging from analyses of biochemical pathways to studies of structure and organization of social networks. Getoor and Diehl [8] provide an excellent survey of early results in this area.

Among the efforts targeting node ranking in a (single) network, the two relevant results are the Page-Rank [1], [9] method of Page et al. and Silverstein et al., and the HITS algorithm of Kleinberg et al. [6]. Page-Rank models a web surfer using random walks with occasional new traversals initiated from newly selected pages (nodes). The rank of a page is determined by computing the steady-state distribution of the consequent random process. The HITS model, on the other hand, distinguishes between "hubs" and "authorities", and computes their ranks in a mutually reinforcing manner [10]. The motivation behind HITS is that a good authority should be pointed to by many good hubs, and that a good hub should point to many good authorities. Each page has both a hub score and an authority score. These scores are updated iteratively by the HITS algorithm. The HITS method is closely related to the Page-Rank algorithm – it finds similarity scores using two separate random walks on the corresponding bipartite graph of hubs and authorities, based on two different transition matrices. It follows naturally that the final scores are also the equilibrium distributions of the respective random walks [11]. Building on there methods, Ding et al. [10] propose a framework that unifies the HITS and the Page-Rank methods, and motivates other techniques, *OnormRank*, *InormRank*, and *SnormRank*. A number of modifications have also been proposed to the original Page-Rank and HITS algorithms. Bharat et al. [12] and Chakrabarti et al. [13] extend the HITS method to weighted graphs. Ng et al. [11], [14] analyze the stability of Page-Rank and HITS methods with respect to small perturbations to the network structure. They also present a new variant of HITS method to improve its stability.

Motivated by the Page-Rank and HITS methods, several efforts target computation of similarity of nodes across networks. This problem is sometimes also referred to as network alignment. Blondel et al. [5] generalize the HITS method and introduce a measure of similarity between any pair of vertices in a pair of directed graphs. Their method, however, does not converge to unique odd and even limits in general. Zager et al. [15] propose a modified version of Blondel iterations by adding additional diagonal elements in order to amplify the scores of nodes that are highly connected (have high degree). Their proposed iteration is shown to always converge independent of the initial condition.

Methods based on Page-Rank have also been proposed for finding node similarities in different application domains. Of these, the IsoRank algorithm [3] of Singh et al. is of particular relevance. IsoRank computes vertex similarity scores in protein-protein interaction networks, integrating both vertex attributes (similarity of protein sequences, or elemental similarity) and topological similarities (links to similar nodes). It then uses bipartite matching to align the pair of input networks based on topological similarity scores. Another method, similar in nature to Page-Rank, is the graph kernel proposed by Rupp et al. [16], which uses special characteristics of chemical networks (bounded degree) to specialize Page-Rank to their target structures. Specifically, in each iteration they find the optimal mapping between neighborhoods of each pair of vertices to compute topological similarity. Finding these optimal mappings is feasible, since there are constant numbers of total mappings between neighborhoods of each pair of nodes.

Network Similarity Decomposition (NSD) is a highly efficient algorithm for Page-Rank-based topological similarity computations. It works by *un-coupling* and *decomposing* similarity computations. In addition to significant improvements in computational cost (orders of magnitude), NSD supports a rich-query set efficiently, generates execution traces that can be used to identify patterns that contribute significantly to topological similarity scores, and is highly amenable to parallel implementation [17]. It also supports parameters that can be tuned for desired model, as well as performance requirements.

2.2 Terminology and Preliminaries

We represent a graph $G_A(V_A, E_A)$ by its adjacency matrix A, where $a_{ij} = 1$ iff node i points to node j, indicated by $i \rightarrow j$, and zero otherwise. V_A and E_A denote the vertices and edges of G_A respectively, and n_A denotes the total number of nodes in G_A. Further, $d(i)$ represents the number of links associated with node i (sum of in-links and out-links for directed graphs), also known as the vertex degree of node i. Matrix $\tilde{A}$ is the normalized version of the matrix A^T; formally, $(\tilde{A})_{ij} = a_{ji} / \sum_{i=1}^{n_A} a_{ji}$ for nonzero rows of A and zero otherwise. We denote by 1, the column vector of size n_A consisting of 1's

Using this notation, matrix-vector products have interesting interpretations. If vectors denote score values distributed across the nodes of a directed graph, $y = \tilde{A}x$ or $y^T = x^T \tilde{A}^T$ can be viewed as *score transport* operations: the i^{th} node "pulls" one portion of the current score of each of its neighbors; each portion "pushed" by its j^{th} neighbor is $\frac{x_j}{d(j)}$ in value. This notion of *score transport* helps us interpret more complex expressions. For example, element i, j of matrix $\tilde{A}^n$ is the fraction of the unit score initially at node j, accruing at node i after n steps.

2.3 Network similarity as ranking

Most current algorithms for ranking nodes in a single graph A use either products of A and A^T (the HITS algorithm [6]), or powers of $\tilde{A}$ (the PageRank algorithm [1]). In doing so they capture both in-link and out-link information, since matrix A corresponds to sending on out-links, A^T corresponds to receiving from in-links, and $\tilde{A}$ corresponds to receiving from in-links "contracted" (or normalized) by the number of out-links of the sources.

An interesting extension of this concept relates to the use of link information simultaneously from two graphs in order to derive similarity scores for pairs of nodes, one from each graph. The common basis for node ranking algorithms within a given graph is that "a node has high ranking if its immediate neighbors have high rankings". This recursive definition is used by iterative matrix products to incorporate contributions from distant neighbors. Analogously, a basis for topological [1] similarity is that "a pair of nodes has high similarity if its immediate neighbors have high similarities". This neighbor-relation is defined over a new graph, the tensor product $G_C = G_A \times G_B$, having as its nodes, pairs of nodes from G_A, G_B. Node c_i in this product graph takes the form (a_i, b_i), where $a_i \in V_A$ and $b_i \in V_B$. Edges in this graph, E_C are of the form $(c_i, c_j) \in E_C$ *iff* $(a_i, a_j) \in E_A$ *and* $(b_i, b_j) \in E_B$. Conventional node ranking algorithms can be executed on this product graph G_C. Nodes with high ranking scores in this graph correspond to high-similarity node-pairs from G_A and G_B.

We now consider two examples of ranking nodes in G_C to compute similarity scores between nodes in G_A and G_B.

2.3.1 *HITS Inspired Algorithm*

Blondel et al. [5] construct a matrix X of dimensions $n_B \times n_A$, which is initialized to all 1's. The following

[1] 1. Unweighted directed graphs are networks with weight one on all edges. Their generalization to graphs with arbitrary edge weights is straightforward for much of our discussion here. We use the terms *directed graphs* and *networks* interchangeably

iterative procedure is then applied:

$$X \leftarrow BXA^T + B^T XA. \tag{1}$$

After each step, the matrix X is normalized. Even iterates of this procedure are proved to converge to a matrix X' where x'_{ij} is a measure of the similarity of node $i \in V_B$ and $j \in V_A$.

One may introduce an operator $vec(\cdot)$ for stacking matrix columns into a vector (as well as its associated "inverse" $unvec(\cdot)$ operator for re-assembling the matrix). If $x = vec(X)$, an interesting property of Kronecker products:

$$vec(AXB) = (B^T \otimes A)x, \tag{2}$$

can be written as:

$$AXB = unvec((B^T \otimes A)x) \tag{3}$$

In view of this, we can write the basic step of iterative procedure (1) as:

$$x \leftarrow (A \otimes B + A^T \otimes B^T)x,$$

or, since $C = A \otimes B$, as

$$x \leftarrow (C + C^T)x \tag{4}$$

Consequently, except for the normalization step that can be carried to the end, *these iterations generate products of an arbitrary number of C and C^T factors* applied to some initial vector x.

2.3.2 PageRank Inspired Algorithm

Singh et al. [3] propose an iterative procedure of the form:

$$x \leftarrow \alpha \tilde{A} \otimes \tilde{B} x + (1-\alpha)h. \tag{5}$$

Here $x = vec(X)$, with x_{ij} as above, and $h = vec(H)$, with element $h_{i,j}$ of matrix H corresponding to the elemental similarity score between node $i \in V_B$ and $j \in V_A$. The vector h is normalized to unity. Successive iterates scale topological similarity and elemental similarity of nodes by factors $\alpha \leq 1$ and $1-\alpha$, respectively. In the specific application context of Singh et al., h encodes protein sequence similarity scores, and protein interaction networks G_A and G_B are undirected. Note also that $\tilde{C} = A \tilde{\otimes} B = \tilde{A} \otimes \tilde{B}$ ($\tilde{\square}$ distributes over $\square \otimes \square$), and thus iteration step (6) can also be written as:

$$x \leftarrow \alpha \tilde{C} x + (1-\alpha)h \tag{6}$$

Equation (6) is effectively a Page-Rank calculation, $x \leftarrow \alpha G x + (1-\alpha)v$, where the Google matrix G [1] for *one* graph is replaced by the $\tilde{C}$ matrix (the column stochastic form of the Kronecker product of the adjacency matrices of *two* graphs) and the *personalization* vector v (user preferences in browsing) is replaced by h (precomputed matching preferences, or elemental similarities). By "unvec"ing (6), we obtain:

$$X \leftarrow \alpha \tilde{B} X \tilde{A}^T + (1-\alpha)H \tag{7}$$

The topological similarity component in this equation $\alpha \tilde{B} X \tilde{A}^T$ has an interesting interpretation along the lines of the *score transport* metaphor (see Figure 1 example).

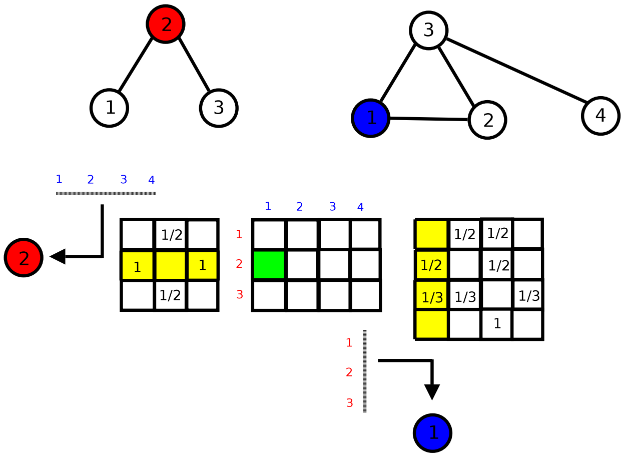


Fig. 1: An example 3×4 similarity matrix X (bottom center) between $\tilde{B}$ and $\tilde{A}^T$ -under their respective graphs G_B and G_A (at top left and top right respectively)- exactly as in $\tilde{B}X\tilde{A}^T$. Node 2 "pulls" the total score of its two neighbors; however there are 4 ways of doing this (2nd row of $\tilde{B}$ multiplied with each of the 4 columns in X gives a row vector y() with 4 elements). This corresponds to the fact that node 2's neighbors are implicitly linked with each of the 4 nodes in the other graph (i.e., for possible matchings). However, the X_{21} entry will finally be updated only by y(2) and y(3) weighted contributions, because only nodes 2 and 3 happen to be neighbors of node $1 \in V_A$ ("pushing" their contributions to it). The weights will are $\frac{1}{2}$ and $\frac{1}{3}$, because in turn nodes 2 and 3 have 2 and 3 neighbors, respectively.

3 Network Similarity Decomposition (NSD)

We present our method for decomposing the computation of the similarity scores for a given pair of networks. The starting point for our discussion is the approach of Singh et al. [3]. Our method dramatically reduces the cost of these computations. Furthermore, it provides execution traces that can be used to interpret topological similarity scores by identifying link patterns that significantly contribute to topological similarity.

We start by expanding the iteration (6), and without loss of generality use h for the initial condition ($x^{(0)} = h$), successively yielding:

$$\begin{aligned} x^{(1)} &= \alpha \tilde{C} h + (1-\alpha)h, \\ x^{(2)} &= \alpha^2 \tilde{C}^2 h + (1-\alpha)\alpha \tilde{C} h + (1-\alpha)h, \\ &\ldots \end{aligned}$$

resulting, after n steps, in the following expression for iterate $x^{(n)}$:

$$x^{(n)} = (1-\alpha) \sum_{k=0}^{n-1} \alpha^k \tilde{C}^k h + \alpha^n \tilde{C}^n h, \tag{8}$$

which in the limit $n \to \infty$ for $\alpha < 1$ simplifies to

$$x^{(\infty)} = (1-\alpha)\sum_{k=0}^{\infty} \alpha^k \tilde{C}^k h \tag{9}$$

Such expansions have been studied in [18], albeit in the context of PageRank calculations. Expansion (8) states that similarity scores ("vec"ed into iterate $x^{(n)}$) can be expressed as power series in $\tilde{C}$ (capturing the network structure of the graphs under comparison) applied to vector h (capturing prescribed information or user preferences), also weighted by parameter α (determining the relative contribution of these two factors in the computed similarity scores).

We now focus on expanding and "unvec"ing power terms in Expression (8). In view of the property $(\tilde{A} \otimes \tilde{B})(\tilde{A} \otimes \tilde{B}) = \tilde{A}^2 \otimes \tilde{B}^2$ for multiplying Kronecker products, for the second power term we have:

$$\tilde{C}^2 h = (\tilde{A} \otimes \tilde{B})(\tilde{A} \otimes \tilde{B})h = (\tilde{A}^2 \otimes \tilde{B}^2)h$$

Utilizing Property (3) for "unvec"ing, this yields:

$$\tilde{B}^2 H (\tilde{A}^T)^2$$

Similarly, the "unvec"ed k^{th} power term corresponding to $\tilde{C}^k h$ for any k, expands to $\tilde{B}^k H (\tilde{A}^T)^k$, and so the "unvec"ed version of Expression(8) for $X^{(n)}$ can be written as:

$$X^{(n)} = (1-\alpha)\sum_{k=0}^{n-1} \alpha^k \tilde{B}^k H (\tilde{A}^T)^k + \alpha^n \tilde{B}^n H (\tilde{A}^T)^n \tag{10}$$

This expansion could alternatively be generated by iterating for n steps the "unvec"ed version of expression (6) and setting $X^{(0)} = H$.

We proceed by decomposing H (with the dual purpose of encoding preferences and serving as the initial condition for our iterations), into a sum of outer products of vectors. Singular Value Decomposition (SVD), a well established method for this purpose, enables us to write:

$$H = \sum_{i=1}^{r} \sigma_i u_i {v_i}^T, \tag{11}$$

where $r \leq \min(n_A, n_B)$ is the rank of H, and $\sigma_i > 0$, u_i, v_i for $i = 1, \ldots, r$ are, respectively, the singular values, the left singular vectors and the right singular vectors of H. Note that σ_i are implied sorted (σ_1 is its largest singular value); additionally vectors u_i constitute an orthonormal basis ($u_i u_j^T = \delta_{ij}$); similarly for vectors v_j vectors ($v_i v_j^T = \delta_{ij}$).

Inserting (11) into (10), we get

$$X^{(n)} = \sum_{i=1}^{r} \sigma_i \left[(1-\alpha)\sum_{k=0}^{n-1} \alpha^k \tilde{B}^k u_i v_i^T (\tilde{A}^T)^k + \alpha^n \tilde{B}^n u_i v_i^T (\tilde{A}^T)^n \right] \tag{12}$$

Setting $u_i^{(k)} = \tilde{B}^k u_i$ and $v_i^{(k)} = \tilde{A}^k v_i$, we obtain

$$X^{(n)} = \sum_{i=1}^{r} \sigma_i \left[(1-\alpha)\sum_{k=0}^{n-1} \alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T \right] \tag{13}$$

Or, more compactly, as a sum of component score contributions $X_i^{(n)}$ with:

$$X_i^{(n)} = \sigma_i \left[(1-\alpha)\sum_{k=0}^{n-1} \alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T \right] \tag{14}$$

Separately, from each SVD triplet (σ_i, u_i, v_i):

$$X^{(n)} = \sum_{i=1}^{r} X_i^{(n)} \tag{15}$$

We stress the fact that SVD is only one of the alternatives for decomposing H into a sum of outer products for a given number, s, of vector pairs. Such a decomposition can generally be expressed as:

$$H = \sum_{i=1}^{s} w_i z_i^T, \tag{16}$$

and, Expressions (14) and (15) can be modified, respectively, to:

$$X_i^{(n)} = (1-\alpha)\sum_{k=0}^{n-1} \alpha^k w_i^{(k)} {z_i^{(k)}}^T + \alpha^n w_i^{(n)} {z_i^{(n)}}^T, \tag{17}$$

and:

$$X^{(n)} = \sum_{i=1}^{s} X_i^{(n)} \tag{18}$$

Note that the decomposition happens along sets of paths of successively larger length k. However, their contributions are damped because of the $(1-\alpha)a^k$ factor with $a \in [0,1]$). In this context, $w_i^{(k)} = \tilde{B}^k w_i$, and $z_i^{(k)} = \tilde{A}^k z_i$. Here, s is simply the number of components used, and does not necessarily coincide with r, the rank of H ($s \geq r$ for exact decompositions). Consequently, SVD corresponds to the special case: $w_i \leftarrow \sigma_i u_i$, $z_i \leftarrow v_i$ (with additional orthonormality conditions, not necessarily required, $s = r$). The aforementioned procedure is summarized in Alg. 1.

Algorithm 1 NSD: Calculate $X^{(n)}$ given A, B, $\{w_i, z_i | i = 1, \ldots, s\}$, α and n

```
compute $\tilde{A}$, $\tilde{B}$
for $i = 0$ to $s$ do
    $w_i^{(0)} \leftarrow w_i$
    $z_i^{(0)} \leftarrow z_i$
    for $k = 0$ to $n$ do
        $w_i^{(k)} \leftarrow \tilde{B} w_i^{(k-1)}$
        $z_i^{(k)} \leftarrow \tilde{A} z_i^{(k-1)}$
    end for
    zero $X_i^{(n)}$
    for $k = 0$ to $n-1$ do
        $X_i^{(n)} \leftarrow X_i^{(n)} + \alpha^k w_i^{(k)} {z_i^{(k)}}^T$
    end for
    $X_i^{(n)} \leftarrow (1-\alpha) X_i^{(n)} + \alpha^n w_i^{(n)} {z_i^{(n)}}^T$
end for
$X^{(n)} \leftarrow \sum_{i=1}^{s} X_i^{(n)}$
```

3.1 Complexity considerations

A straightforward, although naive, implementation of the similarity calculation for dense X would proceed along the lines of Equation (6). We denote the number of non-zero elements in adjacency matrix A (the number of edges in graph G_A), by nnz_A. The adjacency matrix C of the product graph of A and B has $nnz_C = nnz_A \times nnz_B$ nonzero entries. Consequently, each iteration takes an order of $nnz_A \times nnz_B$ floating point operations.

If we use the iteration kernel of (7) (triple-matrix products) as in [7], the computational complexity per step is of the order of $n_B \times nnz_A + n_A \times nnz_B$. Since in our applications, $\frac{nnz_{A,B}}{n_{A,B}} > 1$, this is an improvement by an identical factor.

By using our proposed method, NSD (Algorithm (1)), each iteration step (for a number of s components) costs:

$$s \times (nnz_A + nnz_B + n_A \times n_B).$$

Therefore, for $s < \frac{nnz_{A,B}}{n_{A,B}}$ (practically when the number of components is less than the average degree in our graphs, which is the case in our applications) significant improvements are possible. This is confirmed by our numerical experiments.

What we highlight above is only one aspect of the improvement. Our decomposition approach is flexible enough to address specialized queries without constructing the full similarity matrix X, since it effectively operates in a node pair-by-pair fashion. These specialized queries include:

- What is the similarity score of nodes $i \in V_B$ and $j \in V_A$? NSD can compute this query in $O(s \times n)$ floating point operations. This value is also the respective *relative* similarity score of the two nodes.
- Find an upper or lower bound of final X entries. NSD computes this in $O(s \times n \times \max\{n_A, n_B\})$ floating point operations.

In these cases, we assume that the vectors from the power iterations (steps 6, 7 in Algorithm 1) are available. The cost of computing these vectors is $O(s \times n \times \max\{nnz_A, nnz_B\})$. However, this cost can be amortized since they can be reused.

3.2 Extension to Sparse Networks

Our algorithm outputs a dense similarity matrix X, i.e., it produces similarity scores for each pair of nodes ($i \in V_B, j \in V_A$). This corresponds to a dense bipartite graph between V_A and V_B. Such a dense similarity matrix becomes prohibitively large for pairs of graphs of the order of only a few tens of thousands of nodes each. This drastically constrains similarity explorations of a variety of other networks under the "dense X" assumption. Several applications, though, permit the pre-supposition of vanishing similarity scores for a large fraction of potential matches. For instance, in [7], where topic/ subject networks are compared, the similarity matrix X is "sparsified" down to a density of 10^{-4} (sparse bipartite graph) by eliminating, from the start, node matches with poor affinity in their textual labels (i.e., using metadata tags).

Given a bipartite graph L with nodes in $V_B \cup V_A$ and edge set (as a set of candidate node pairs) $E_L = \{(p,q)|p \in V_B, q \in V_A\}$ that is sparse ($|V_B| \times |V_A| \gg |E_L|$), we can "sparsify" our NSD algorithm as follows:

Algorithm 2 NSD_Sparse

```
Enforce sparse $X_i^{(n)}$ matrices with non-zero entries at positions indexed by $E_L$.
Replace occurrences of outer vector products $P_i^{(k)} = w_i^{(k)} {z_i^{(k)}}^T$, $k = 0, \ldots, n$, in steps 11, 13 in Algorithm 1 with their "sparsified" versions:
for $(p,q) \in E_L$ do
    $(P_i^{(k)})_{pq} = (w_i^{(k)})_p ({z_i^{(k)}}^T)_q$
end for
```

Algorithm 2 iterates over only the candidate pairs (even though power iterations $w_i^{(k)} \leftarrow \tilde{B} w_i^{(k-1)}$, $z_i^{(k)} \leftarrow \tilde{A} z_i^{(k-1)}$ include all nodes, albeit separately for each graph). This strategy may be further optimized if there are unpaired nodes in either graph (smaller node set for L).

3.3 Remarks on the use of Singular Value Decomposition

We note some important aspects of SVD, used in the previous section:

- Similarity matrix H is a non-negative matrix. In the general case, some entries in vectors u_i, and v_i will be negative, due to orthonormality. Perron-Frobenius theorem ensures non-negativity only of elements of vectors u_1, v_1. This implies that some component score matrices $X_i^{(n)}$ will have negative entries. This might seem unnatural at first glance, because we are used to non-negative similarity scores. However, note that these scores are added together, the corresponding summation in the final $X^{(n)}$ is non-negative by construction (see Eq (10)). A negative entry at the p, q position in matrix $X_i^{(n)}$ implies that the i^{th} SVD triplet *penalizes* the matching of nodes $p \in V_B$ and $q \in V_A$.
- Another inconvenience from nonnegative entries is the difficulty in bounding the error in $X^{(n)}$, if using less than r terms in the SVD expansion. However, if we decompose H as in (16) using vectors with non-negative entries $w_i \geq 0, z_i \geq 0, i = 1, \ldots, s$ then the matrix sequence of partial sums $f_j = \sum_{i=1}^{j} X_i^{(n)}$, $j = 1, \ldots, s$ would be element-wise increasing, and since $f_s = X^{(n)}$ is normalized to unity, this would give us an indication of the proportion of the target score matrix reached by utilizing less than s components. In this respect, Non-negative Matrix Factorization (NMF) [19] could be advantageous compared to SVD.

Species	Dataset name	#Nodes	#Edges
nematode worm(*C. elegans*)	celeg	2805	4572
fruitfly(*D. melanogaster*)	dmela	7518	25830
bacterium(*E. coli*)	ecoli	1821	6849
bacterium(*H. pylori*)	hpylo	706	1414
human(*H. sapiens*)	hsapi	9633	36386
mouse(*M. musculus*)	mmusc	290	254
yeast(*S. cerevisiae*)	scere	5499	31898

TABLE 1: Species for which PPI network data (undirected) is used in our experiments.

- Regarding the mixing of negative and non-negative entries in the same $X_i^{(n)}$ component, one may utilize the method of Boutsidis [20] for writing each SVD term as a difference of two separate, *non-negative* sub-terms:

$$H = \sum_{i=1}^{r} \sigma_i(u_{i+}v_{i+}^T + u_{i-}v_{i-}^T) - \sigma_i(u_{i+}v_{i-}^T + u_{i-}v_{i+}^T), \tag{19}$$

  where an extra $+$ subindex denotes the respective vector with its negative entries zeroed, while an extra $-$ subindex denotes the vector with its non-negative entries zeroed and its negative ones substituted with their absolute values. On the downside, the run time is expected to increase by a factor of four, in the worst case.

4 Numerical experiments

We now present detailed experimental evaluation of our proposed algorithm and compare it to a number of existing approaches in the context of diverse network-structured datasets.

4.1 Timing results (PPI Networks)

We first examine the computation time for our proposed technique. For this purpose, we use protein interaction data (PPI) and the IsoRank approach of Singh et al. for comparison (native binary for IsoRank available at [21]). Details of data for seven species (networks) used in the experiment are provided in (Table 4.1).

We also use Matlab codes from netalign [22], specifically their IsoRank and maximum weight bipartite matching implementations; these were originally written to support [7]. Note that this IsoRank implementation does not explicitly construct the Kronecker product $\tilde{A}\otimes\tilde{B}$ to apply to $vec(X)$ at each step (referred as SpaIsoRank in [7]). This is prohibitive for our test data. However, this implementation uses the equivalent triple matrix product kernel of $\tilde{B}X\tilde{A}^T$ instead (MAT3 code).

We compute the similarity matrices X for all possible pairs (first column in Table 2) of species using only PPI data (network data). We set $\alpha = 1.0$, use uniform initial conditions (outer product of suitably normalized 1's for each pair) and execute 20 iterations in all runs. Note that $\alpha = 1.0$ is not treated as a special case in our Matlab code. This would yield additional speedups. Instead, all terms are generated.

Unfortunately the native code from Singh et al. [21] does not provide an option for generating either the similarity matrix X, or the timings for its computation. It internally uses the result of this (first) phase, to extract the best matching node pairs (second phase). The total timing results - for both phases - are reported together with the extracted matching pairs. Specifically, results from two matching algorithms are reported [3], where X is interpreted as encoding a weighted bipartite graph:

- A generic maximum-weight bipartite matching algorithm is applied to X.
- A heuristic iterative approach is applied to X, where the highest "score" x_{ij} is located, the pairing (i, j) is recorded and all "scores" involving either i or j are deleted until one of the graphs gets all its nodes paired (also referred as GM-Greedy Match hereafter).

The focus of our approach is the computation of X in a fast, decomposable manner. It follows that if similarity matrices agree and the same matching process is applied to them, then the same best pair matches will be generated. Experiments show that the similarity matrices we compute are identical to within machine precision to those computed by netalign. Block diagrams of the codes used in our experiments are summarized in Figure 3.

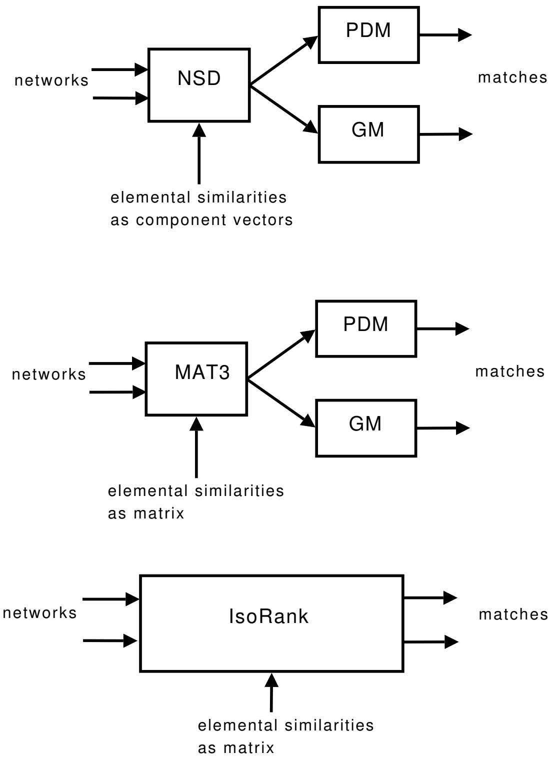


Fig. 2: NSD and MAT3 are codes for computing the similarity matrix, PDM, GM are codes for extracting matchings; IsoRank binary includes codes for computing the similarity matrix and extracting two matchings.

Each native IsoRank run computes the similarity matrix and calculates two alternate bast pair matchings (4th column in Table 2). The corresponding timings for computing the similarity matrix alone with our IsoRank NSD Matlab codes are presented in the 2nd column of Table 2. For a fair timing comparison we independently run the Primal-Dual Matching (PDM) implementation from netalign on our computed similarity matrix X to get best pairs matchings. These timings are reported in the 3rd column of Table 2); Here, we assume that the generic matching algorithm in Singh et al. [3] will have comparable execution time and, in any case, it will not dominate the main computation of the similarity matrix. We also implement the GM heuristic (Java and C codes) and achieve timings much smaller than those reported in the 3rd column. We conclude from these results that in all cases, NSD significantly outperforms the original IsoRank implementation for computing topological similarity matrix X (after taking into account the overhead of matching in tabulated IsoRank timings).

NSD also outperforms the netalign Matlab codes (MAT3) for computing similarity scores alone. Here, the comparisons are direct and Figure 3 illustrates the respective NSD speedups for all species combinations. Speedup factors of roughly five to nine are observed. We subsequently implement NSD in Java (for additional gains) and run a series of timing experiments for various $\alpha < 1.0$, the same number of iterations (20) as before, $s = 1$, and uniform initial conditions, also serving as elemental similarities (column 2) against MAT3 (column3) and IsoRank (column 6); see Table 3 ($a = 0.80$). We also include timings for both match extraction algorithms: PMD (column 4 in Matlab) and GM (column 5 in Java). This data clearly shows that NSD outperforms the similarity matrix implementations in MAT3 by a of factor roughly 25! Gains are even more dramatic with respect to the IsoRank implementation: NSD is observed to be up to three orders of magnitude faster for the larger PPI networks (after subtracting the "overhead" of two matching algorithm - PDM and GM - execution times).

These reported timings are for the case $s = 1$. For larger values of s, one may expect a drop in the performance improvement (a factor linear in s). In other words, it would take a few tens or a few hundreds of component similarity matrices $X_i^{(n)}$ for our method to have performance characteristics respectively similar to those of existing MAT3 or IsoRank implementations. Even in such cases, NSD provides significant additional information (not available in the alternative computational approaches) of exactly how initial condition components contribute to the final similarity scores.

4.2 Self-similarity in Networks

We also use the Facebook networks referenced in [23], each describing the corresponding USA university Face-

Species pair	NSD (secs)	PDM (secs)	IsoRank (secs)
celeg-dmela	7.92	166.79	833.97
celeg-ecoli	1.84	44.40	137.08
celeg-hpylo	0.74	19.48	35.61
celeg-hsapi	12.68	232.15	1094.36
celeg-mmusc	0.35	12.71	20.66
celeg-scere	6.40	150.34	798.06
dmela-ecoli	5.08	91.89	785.21
dmela-hpylo	1.88	25.62	216.66
dmela-hsapi	32.53	479.88	N/A
dmela-mmusc	1.09	15.86	55.06
dmela-scere	15.01	191.18	5306.00
ecoli-hpylo	0.61	12.78	55.10
ecoli-hsapi	7.70	83.14	2022.19
ecoli-mmusc	0.20	11.58	10.38
ecoli-scere	4.08	79.85	1297.77
hpylo-hsapi	2.70	15.66	286.01
hpylo-mmusc	0.06	.47	6.40
hpylo-scere	1.79	6.77	227.80
hsapi-mmusc	1.27	1.59	87.82
hsapi-scere	20.35	26.68	7691.00
mmusc-scere	0.81	3.22	43.23

TABLE 2: Timing results from running NSD (column 2) - implemented in Matlab- against reference IsoRank implementation from [21] (column 4), for various pairs of species (column 1). Reference IsoRank implementation also computes matchings with two different algorithms and column 3 contains timings for Primal-Dual Matching (PDM in Matlab) to find out its overhead (the second - heuristic- matching algorithm (Greedy Matching - GM) is also implemented (in C and Java) only to find times smaller than those in the 2nd column in sample cases (not depicted). $a = 1.0$, 20 iterations.

Species pair	NSD	MAT3	PDM	GM	IsoRank
celeg-dmela	3.15	64.20	152.12	7.29	783.48
celeg-ecoli	0.79	16.26	43.40	1.37	158.93
celeg-hpylo	0.43	5.73	14.01	0.56	37.34
celeg-hsapi	3.28	69.74	163.05	9.54	1209.28
celeg-mmusc	0.26	2.12	8.02	0.24	17.68
celeg-scere	1.97	44.61	127.70	4.16	949.58
dmela-ecoli	1.86	37.79	86.80	4.78	807.93
dmela-hpylo	0.85	16.91	23.04	1.90	204.02
dmela-hsapi	8.61	211.19	590.16	28.10	7840.00
dmela-mmusc	0.51	6.02	11.98	0.73	51.12
dmela-scere	4.79	131.22	182.91	12.97	4905.00
ecoli-hpylo	0.33	3.33	8.63	0.36	41.68
ecoli-hsapi	2.41	47.48	79.23	4.76	2029.56
ecoli-mmusc	0.26	1.77	8.24	0.20	10.03
ecoli-scere	1.49	35.86	69.88	2.60	1264.24
hpylo-hsapi	1.18	18.92	13.29	1.83	316.90
hpylo-mmusc	0.17	0.52	3.02	0.09	6.13
hpylo-scere	0.68	14.01	11.76	1.05	220.82
hsapi-mmusc	0.74	8.74	18.85	0.93	70.24
hsapi-scere	6.09	152.02	181.17	15.56	6714.00
mmusc-scere	0.46	4.35	2.71	0.47	40.86

TABLE 3: Timing results (in seconds) from running NSD (column 2) - implemented in Java- against reference IsoRank implementation from [21] (column 6) and the Equation (7) approach (MAT3, column 3), for various pairs of species (column 1). There are also timings for two matching algorithms: PDM (in Matlab) and GM (in Java). $a = 0.8$, 20 iterations, $s = 1$ (uniform initial conditions).

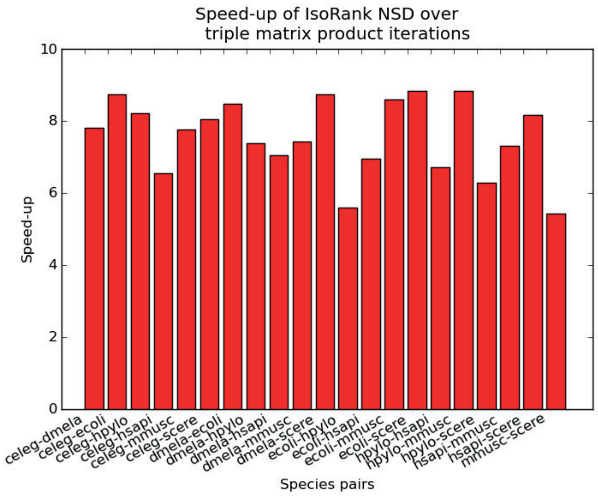


Fig. 3: Speedups in computing similarity matrices for various species-pairs with NSD against MAT3 (from [22]). Here $a = 1.0$ and NSD is implemented in Matlab. Note that by porting it to Java, 25 fold speedups are observed (for $a = 0.80$, see columns 2 and 3 in Table 3).

book community (Table 4 [2]). A Facebook network is undirected since both endpoint profiles (nodes) should agree to a "friend" relation (edge).

Since a pairwise comparison is hard to justify in this context, we compute similarity scores between nodes belonging to the same network as a first step. This approach is also applied for PPI networks of comparable sizes (number of nodes). We use $\alpha = 0.80$ and perform 20 iterations in each case for $s = 1$ (one pair of random initial conditions - breaking possible ties - with the same seed for all experiments).

As expected, our algorithm correctly matches each node with itself. However we exclude these matches by zeroing such similarity scores and then extract the matches from the modified similarity matrix using GM. The reason is that we want to identify common connected subgraphs based on the matches. Generally, for two graphs G_A, G_B, these are identified after the construction of their *alignment graph* as follows: If $m_1 = (i_1, j_1)$ and $m_2 = (i_2, j_2)$ in $V_B \times V_A$ are two matches then m_1 and m_2 are connected by an edge iff i_1 is connected with i_2 and j_1 with j_2 in the original graphs. Common connected subgraphs are identified as the connected subgraphs in the alignment graph, (i.e. "clusters" of *pairs* of matching nodes from the original graphs also conserving their link patterns). Therefore, in our case, with $G_A \equiv G_B$ this would mean that, failing to exclude the aforementioned generic matches, the common connected subgraphs would simply replicate the original single graph structure, which surely would not add to our knowledge.

Facebook Network	#Nodes	#Edges
Caltech36	769	33312
Princeton12	6596	586640
Georgetown15	9414	851276

TABLE 4: Facebook networks used in the experiments. Number of nodes and edges are given in each case.

Allowing for the symmetries in the matching pairs in the single graph case ($(i_1, i_2) \equiv (i_2, i_1)$ and $m_1 = (i_1, i_2)$, $m_2 = (i_3, i_4)$ are connected in the common connected subgraphs iff i_1, i_3 and i_2, i_4 (*or* i_1, i_4 and i_2, i_3) are connected)), we identify the *maximum* such Common Connected Subgraph (MCCS) (using routines from the networkx package [24]) for each case. Our results are shown in Table 5. It is clear that the MCCS for Facebook networks are an order of magnitude larger than PPI networks of comparable sizes; in Facebook networks, a large fraction of the identified pairs (column 5) participate in the MCCS. This could be attributed partly to the fact that in Facebook networks a user often connects with friends of his/her "friends" since these are readily accessible, thus inheriting part of their neighborhood structure and strengthening the corresponding similarity scores. On the other hand, there seems to be strong correlation with the average degree in the original network: a large degree will generally increase the probability of connecting two matches (i.e., node pairs), since this relates to the connectivity "options" of their contained nodes.

We also use two large Web networks (directed graphs); details are provided in Table 6. Note that these datasets are available in an efficient compressed format utilizing

[2] Although not large, these test datasets are practical since they contain links to internal nodes only; trying to enforce this property to other larger Facebook datasets by pruning, unfortunately removed a substantial percentage of the initial links. Note that in general the availability of Facebook networks is quite limited due to access restrictions during their collection.

Network	#Nodes	#Edges	% of Nodes	Average degree
hpylo	7	6	1.98	2.00
dmela	84	92	2.23	3.44
scere	239	288	8.69	5.80
hsapi	189	245	3.92	3.78
Caltech36	268	1297	69.79	43.32
Princeton12	1986	7334	60.22	88.94
Georgetown15	2654	7651	56.38	90.43

TABLE 5: Maximum Common Connected Subgraphs(MCCS) characteristics (columns 2,3,4) discovered after post-processing NSD results from each network (column 1) paired with itself. Average degree of each node in the original graph is also given (column 5).

the special techniques described in [25] and their manipulation was made possible by adapting our NSD method also to interface their provided software framework, WebGraph [26].

We compute topological-only self-similarity scores ($\alpha = 1$, 20 iterations) allowing us to get a rank-one representation of self-similarity scores if choosing, without loss of generality, rank-one initial conditions; then essentially the sum in Equation (14) vanishes and the two vectors in the outer product remaining are identical (i.e. the ranking vector).

This space-efficient representation of self-similarity scores produced by NSD is critical: The scores matrix X is dense in general and for these Webgraph instances this would raise the need for storing a dense matrix with dimensions within a few hundreds of thousands range (a few TBs) if methods like IsoRank were to be used, which is well beyond current memory capacities.

This representation also facilitates the matching extraction process: We sort the ranking vector values, then partition them into pairs starting from large entries and finally match the nodes corresponding to these pairs (indexed as in the original ranking vector). This procedure is really the GM method idea applied to a matrix that is the outer product of identical vectors under the constraint not to match a node with itself; this is exactly our case for self-similarity.

These matchings are then used to identify common connected subgraphs. $1,912$ common connected subgraphs containing 4 or more nodes are revealed for cnr-2000 - there are $5,178$ such subgraphs for eu-2005 - see Figure 4 for the distribution of sizes of the largest of them. These cover a large percentage of the total number of nodes in the two graphs (28.29% and 37.31% respectively for cnr-2000 and eu-2005). So NSD applied to a directed graph (where accordingly the scores capture similarities in the structure of incoming links only) can produce matchings revealing extensive repeated substructures in it (self-similarity).

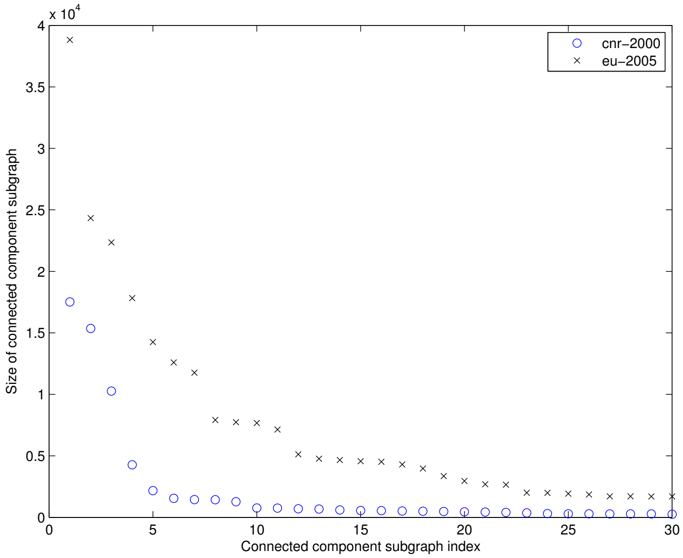


Fig. 4: The sizes of the 30 of the largest common connected subgraphs from self-similarity runs of NSD on Webgraphs: There were detected subgraphs of sizes up to $17,520$ nodes (for cnr-2000) and $38,818$ nodes (for eu-2005).

Graph	Crawl date	Domain crawled	#Nodes	#Edges
cnr-2000	2000	cnr	325,557	3,216,152
eu-2005	2005	eu	862,664	19,235,140

TABLE 6: Characteristics of Web graphs used in experiments. Particularly eu-2005 dataset was gathered by UbiCrawler [27].

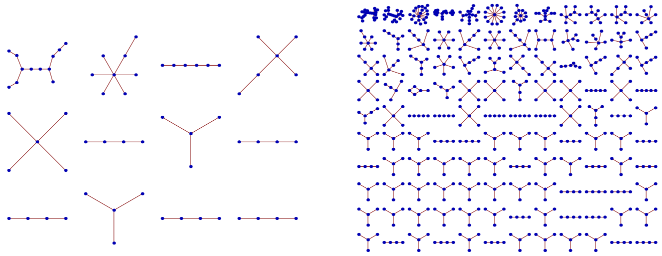
4.3 The effect of multiple components ($s > 1$)

We comment on the effect of using multiple components ($s > 1$) on the structure of common connected subgraphs extracted. Using BLAST (sequence) similarity scores as the matrix H of elemental similarities for the fly/yeast PPI networks we computed all common connected subgraphs of ≥ 4 nodes ($\alpha = 0.80$, 20 iterations) for various cases:

- Assuming H as provider of similarity scores (i.e.

no similarity matrix computation) and also running MAT3 code with H as one of the input arguments. Both were followed by a matching extraction phase using GM. (Figure 5).

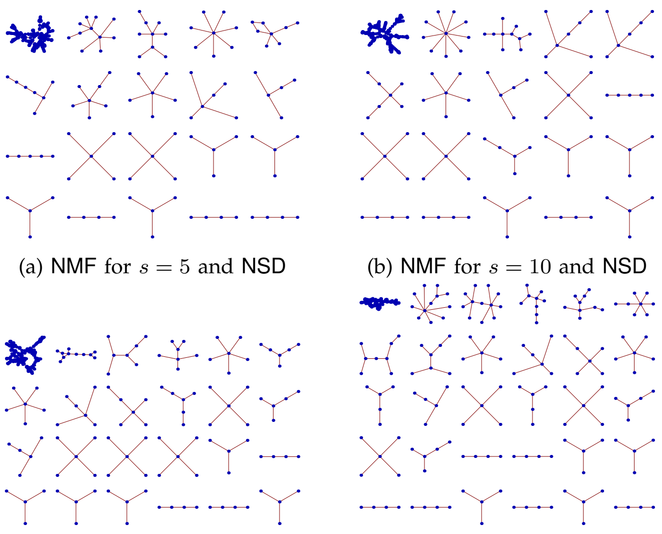
- Decomposing elemental similarity into the dominant $s = 5$ and $s = 10$ components by both NMF and SVD and then successively running NSD code for computing the similarity scores and GM for extracting the matches. (Figure 6).
- Decomposing elemental similarity into the dominant $s = 500$ and $s = 1000$ components by SVD and then successively running NSD code for computing the similarity scores and GM for extracting the matches. (Figure 7).



(a) Elemental similarities H

(b) MAT3

Fig. 5: Common connected subgraphs in the alignment graph produced by generating matches from elemental similarities only (5a) and similarities computed by MAT3 (5b).



(a) NMF for $s = 5$ and NSD

(b) NMF for $s = 10$ and NSD

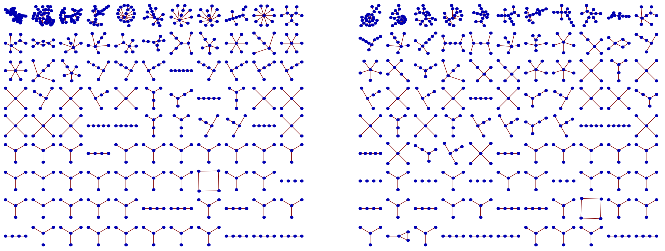
(c) SVD for $s = 5$ and NSD

(d) SVD for $s = 10$ and NSD

Fig. 6: Common connected subgraphs in the alignment graph produced by running NSD on the dominant $s = 5$ and $s = 10$ components of H as computed by NMF (upper row) and SVD (lower row).

When analyzing the alignment graph of two networks, two measures for the topological only evaluation of the computed matching could be used:

- Count the number of edges in the alignment graph (conserved edges). Each conserved edge implies matching the corresponding edges connecting the elements of the matching pairs at its endpoints in the input networks. So node matching naturally leads to edge matching.
- Compute the size of the connected components in the alignment graph (common connected subgraphs). These subgraphs are essentially matchings of substructures in the input networks.



(a) SVD for $s = 500$ and NSD

(b) SVD for $s = 1000$ and NSD

Fig. 7: Common connected subgraphs in the alignment graph produced by running NSD on the dominant $s = 500$ and $s = 1000$ components of H as computed by SVD.

Definitely the existence of many conserved edges increases the probability of them being part of extensive connected subgraphs. However it could also be the case that they are parts of more connected subgraphs all of moderate sizes.

In our experiments SVD decompositions for $s = 5$, $10, 500, 1000$ components coupled with NSD respectively give rise to $514, 537, 1143, 1236$ conserved edges compared to 1455 edges from MAT3. However they identify a large connected subgraph of 239 nodes for $s = 5$ when the largest such subgraph from MAT3 consists of only 22 nodes.

As the number of components is increased ($s = 10$, $500, 1000$) the largest subgraph size drops (respectively 204, 55 and 25 nodes) for SVD. This follows from the fact that the use of more components should yield better approximations of elemental similarities in H and thus results closer to the ones from MAT3 code are expected.

Note also that the use of the GM for matching extraction (which is resilient to possibly negative values in the elemental similarity scores used for small s) gives a "smooth" succession of results in the SVD case. Also using NMF that - by definition - generates nonnegative elemental scores, leads to results qualitatively similar to those from SVD (e.g. 521 conserved edges and a largest common connected subgraph of 266 for $s = 10$ components).

5 Concluding Remarks and Future Work

We have presented NSD, a novel approach for uncoupling and decomposing ranking-inspired methods for computing similarity scores between graph nodes. Decomposition happens with respect to the the rank-one terms building up the initial condition (the preferences).

The uncoupled (per-graph) computations involve sparse matrix-vector products, "merged" only at the end. Timing data for NSD, show that our approach is up to three orders of magnitude faster than state-of-the-art methods, like IsoRank. This is especially true when preferences (that also serve as the initial conditions in the proposed iteration) can be written as sums of small number of of outer products. It follows that NSD is a *fast* method for computing similarities (or self-similarities) for nodes of general types of (undirected) networks (PPI and Facebook networks are only two examples); we also apply it to large (directed) Web graphs. This node similarity information provides input for subsequently identifying common link structures in the graphs, thus suggesting a form of latent structure and function. In some cases, the quality of the results (matching pairs) may depend on the choice of suitable pre-processing/input parameters (α, preferences) and the appropriate post-processing of score values (e.g., filtering based on thresholds). This implied "calibration" procedure in turn, demands domain knowledge. Consequently, while NSD is a *general* tool, its parameters and output processing can be *specialized*.

The "un-coupling" and "decomposing" properties of NSD offer natural avenues for parallelization. Furthermore, the concept of decomposition for *directed* graphs offers novel insights into network comparison. Both of these research directions are currently under investigation, and will be addressed in separate reports.

Acknowledgment

Acknowledgments to be added to camera ready version upon acceptance.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," Stanford University, Tech. Rep., 1998.

[2] G. Jeh and J. Widom, "SimRank: a measure of structural-context similarity," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton, Alberta, Canada: ACM, 2002, pp. 538–543. [Online]. Available: http://portal.acm.org/citation.cfm?id=775126

[3] R. Singh, J. Xu, and B. Berger, "Global alignment of multiple protein interaction networks with application to functional orthology detection," *Proceedings of the National Academy of Sciences*, vol. 105, no. 35, p. 12763, 2008.

[4] J. Y. Pan, H. J. Yang, C. Faloutsos, and P. Duygulu, "Automatic Multimedia Cross-modal Correlation discovery," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM New York, NY, USA, 2004, pp. 653–658.

[5] V. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, "A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching," *SIAM Rev.*, vol. 46, no. 4, pp. 647–666, 2004.

[6] J. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM*, vol. 46, pp. 604–632, 1999.

[7] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang, "Algorithms for Large, Sparse Network Alignment Problems," *Data Mining, IEEE International Conference on*, vol. 0, pp. 705–710, 2009.

[8] L. Getoor and C. Diehl, "Link Mining: A Survey," *SigKDD Explorations Special Issue on Link Mining*, vol. 7, no. 2, december 2005.

[9] C. Silverstein, S. Brin, and R. Motwani, "Beyond Market Baskets: Generalizing Association Rules to Dependence Rules," *Data Min. Knowl. Discov.*, vol. 2, no. 1, pp. 39–68, 1998.

[10] C. H. Q. Ding, X. He, P. Husbands, H. Zha, and H. D. Simon, "PageRank, HITS and a unified framework for link analysis," in *SIGIR*, 2002, pp. 353–354.

[11] A. Y. Ng, A. X. Zheng, and M. I. Jordan, "Link Analysis, Eigenvectors and Stability," in *IJCAI*, 2001, pp. 903–910.

[12] K. Bharat and M. R. Henzinger, "Improved Algorithms for Topic Distillation in a Hyperlinked Environment," in *SIGIR*, 1998, pp. 104–111.

[13] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. M. Kleinberg, "Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text," *Computer Networks*, vol. 30, no. 1-7, pp. 65–74, 1998.

[14] A. X. Zheng, A. Y. Ng, and M. I. Jordan, "Stable Algorithms for Link Analysis," in *SIGIR*, 2001, pp. 258–266.

[15] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Appl. Math. Lett.*, vol. 21, no. 1, pp. 86–94, 2008.

[16] M. Rupp, E. Proschak, and G. Schneider, "Kernel approach to molecular similarity based on iterative graph similarity," *Journal of chemical information and modeling*, vol. 47, 2007.

[17] G. Kollias and A. Grama, "Parallel Network Similarity Decomposition," in *Parallel Matrix Algorithms and Applications*, Basel, Switzerland, 2010.

[18] C. Brezinski and M. Redivo-Zaglia, "The PageRank Vector: Properties, Computation, Approximation, and Acceleration," *SIAM J. Matrix Anal. Appl*, vol. 28, pp. 551–575, 2006.

[19] D. D. Lee and S. H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, October 1999. [Online]. Available: http://dx.doi.org/10.1038/44565

[20] C. Boutsidis and E. Gallopoulos, "SVD based initialization: A head start for nonnegative matrix factorization," *Pattern Recognition*, vol. 41, no. 4, pp. 1350–1362, 2008.

[21] Rohit Singh and Bonnie Berger, "IsoRank and IsoRankN," http://groups.csail.mit.edu/cb/mna/.

[22] David F. Gleich, "netalign: Network Alignment codes," http://www.stanford.edu/~dgleich/publications/2009/netalign/.

[23] A. L. Traud, E. D. Kelsic, P. J. Mucha, and M. A. Porter, "Community Structure in Online Collegiate Social Networks," 2008, arXiv:0809.0960.

[24] "NetworkX project website," https://networkx.lanl.gov/.

[25] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. Manhattan, USA: ACM Press, 2004, pp. 595–601.

[26] "WebGraph project website," http://webgraph.dsi.unimi.it/.

[27] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.