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

Giorgos Kollias, Shahin Mohammadi, and Ananth Grama

Abstract—As graph-structured data sets become commonplace, there is increasing need for efficient ways of analyzing such data sets. 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. In this paper, we study the problem of global alignment of large sparse graphs. Specifically, we investigate efficient methods for computing approximations to the state-of-the-art IsoRank solution for finding 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 *uncoupling* and *decomposing* ranking calculations associated with the computation of similarity scores. *Uncoupling* refers to independent preprocessing of each input graph. *Decomposition* implies that pairwise similarity scores can be explicitly broken down into contributions from different link patterns traced back to a low-rank approximation of 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 supported queries. We show over two orders of magnitude improvement in performance over IsoRank/Random Walk formulations, and over an order of magnitude improvement over constrained matrix-triple-product formulations, in the context of real data sets.

Index Terms—Data mining, sparse, structured, and very large systems, singular value decomposition

1 Introduction and Motivation

Graph-structured data sets are commonly encountered in diverse domains, ranging from biochemical interaction networks, to networks of social and economic transactions. Effective analyses of these data sets hold the potential for significant applications' insight. 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 [4] or synonym extraction [5], where node similarity is modeled as node ranking, inspired by its two dominant models—Page-Rank and HITS [6]. In other application scenarios, [7], [8], the topological similarity of two nodes is determined by the overlap in their incidence pattern upon a predetermined set of graphlets, (in the citations, they consider 29 graphlets—all possible subgraphs consisting of up to five nodes). In [7], the distribution of nodes over the incidence patterns is considered, while in [8], each node is explicitly labeled by a vector of coordinates indicating the patterns the node participates in.

• *The authors are with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907.*
E-mail: {gkollias, mohammas}@purdue.edu, ayg@cs.purdue.edu.

Manuscript received 13 Jan. 2011; revised 15 June 2011; accepted 20 June 2011; published online 29 July 2012.
Recommended for acceptance by T. Grust.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-01-0018.
Digital Object Identifier no. 10.1109/TKDE.2011.174.

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 node label similarity and topological similarity. Node label similarity corresponds to the similarity between two labeled nodes, independent of their link structure. For example, in the context of protein interaction networks, node label similarity between two nodes may simply correspond to the sequence match score between the two proteins. Topological similarity, on the other hand, combines node label 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.

Node label 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 node label similarities do not rule out any topological similarity matchings; however, node pairs with higher node label 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 [9]. 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 uncouple 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 data sets. For larger graphs, these gains can be expected to be significantly higher, as we show using a simple performance model for our algorithm.

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 [10] 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], [11] method of Page et al. and Silverstein et al., and the HITS algorithm of Kleinberg [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 [12]. 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 [13]. Building on these methods, Ding et al. [12] 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 and Henzinger [14] and Chakrabarti et al. [15] extend the HITS method to weighted graphs. Ng et al. [13], Zheng et al. [16] 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 and Verghese [17] 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 node label 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. [18], which uses special characteristics of molecular structure graphs (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 *uncoupling* 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 [19]. It also supports parameters that can be tuned for desired model, as well as performance requirements.

NSD operates on a low-rank approximation of node label similarities. In [20] a related theoretical approach is developed within the context of solving the Sylvester equation. However, the applicability of their results leverages the specific spectral content or hierarchical structure for the input matrices. Please note that the idea of low-rank approximation has previously been used for finding similar nodes in a single network. However, in these formulations it is the graph structure, and not the node label similarity scores that are approximated. In [21], this is the case for acceleration of SimRank [2] computations. Liben-Nowell and Kleinberg [22] also use a low-rank approximation of the network for computing a related similarity measure called Katz scores.

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, $(\tilde{A})_{ij} = a_{ji} / \sum_{k=1}^{n_A} a_{jk}$ for nonzero rows of A and zero otherwise. We denote by $\mathbf{1}$ 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 ith node "pulls" one portion of the current score of each of its neighbors; each portion "pushed" by its jth 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 1s. The following 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 $j \in V_A$. Introducing operator $vec(\cdot)$ for stacking matrix columns into a vector (as well as its associated "inverse" $unvec(\cdot)$ operator for reassembling the matrix), and writing $x = vec(X)$, we can derive the following iterative procedure (1):

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

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

$$x \leftarrow (C + C^T)x. \tag{2}$$

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.

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.

2.3.2 PageRank Inspired Algorithm

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

$$x \leftarrow \alpha \tilde{A} \otimes \tilde{B} x + (1-\alpha) h. \qquad (3)$$

Here $x = vec(X)$, with x_{ij} as above, and $h = vec(H)$, with element $h_{i,j}$ of matrix H corresponding to the node label similarity score between node $i \in V_B$ and $j \in V_A$. The vector h is normalized to have unit norm. Successive iterates scale topological similarity and node label 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. Iteration (3), essentially a power method, is known to converge in $O(log(1/(1-\alpha))$ steps for arbitrary $x^{(0)}$ [3]. Furthermore, for $\alpha = 1$, this reduces to a general Markov chain system governed by its corresponding convergence properties. Note also that $\tilde{C} = \widetilde{A \otimes B} = \tilde{A} \otimes \tilde{B}$ ($\tilde{\Box}$ distributes over $\Box \otimes \Box$), and thus iteration step (3) can also be written as

$$x \leftarrow \alpha \tilde{C} x + (1-\alpha) h. \qquad (4)$$

Equation (4) 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 node label similarities). By rewriting (4) in matrix notation using "unvec" ("unvec"ing), we obtain

$$X \leftarrow \alpha \tilde{B} X \tilde{A}^T + (1-\alpha) H. \qquad (5)$$

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 Fig. 1 example). Note that since similarity score transport is inherently bidirectional—a node imports the similarity scores from its neighbors, and also serves as a similarity score source for each of them—this method is better suited for undirected graphs.

3 Network Similarity Decomposition

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 (4), and use h for the initial condition ($x^{(0)} = h$), successively yielding

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

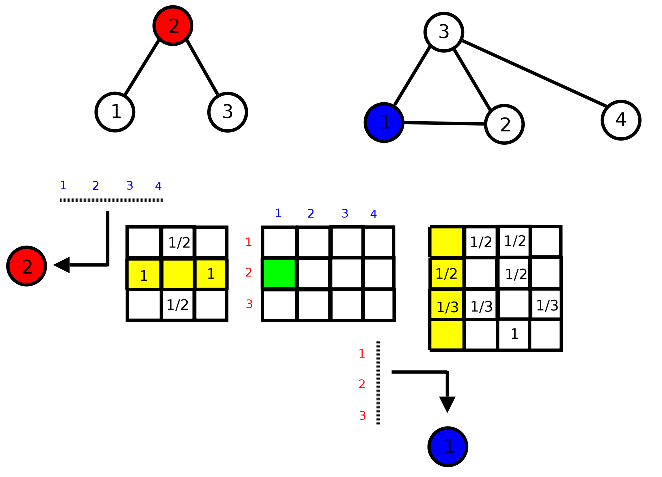


Fig. 1. An example 3×4 similarity matrix X (bottom center) between $\tilde{B}$ and $\tilde{A}^T$—under their respective graphs G_B and G_A (at top left and top right, respectively)—exactly as in $\tilde{B} X \tilde{A}^T$. Node 2 "pulls" the total score of its two neighbors; however there are four ways of doing this (second row of $\tilde{B}$ multiplied with each of the four columns in X gives a row vector y() with four elements). This corresponds to the fact that node 2's neighbors are implicitly linked with each of the four 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 be $\frac{1}{2}$ and $\frac{1}{3}$, because in turn nodes 2 and 3 have 2 and 3 neighbors, respectively.

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

$$x^{(n)} = (1-\alpha) \sum_{k=0}^{n-1} \alpha^k \tilde{C}^k h + \alpha^n \tilde{C}^n h, \qquad (6)$$

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

$$x^{(\infty)} = (1-\alpha) \sum_{k=0}^{\infty} \alpha^k \tilde{C}^k h. \qquad (7)$$

Such expansions have been studied in [23], albeit in the context of PageRank calculations. Expansion (6) 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 (6). 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 the property $AXB = unvec((B^T \otimes A)x)$ of Kronecker products for "unvec"ing, this yields

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

Similarly, the "unvec"ed kth 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 (6) for $X^{(n)}$ can be written as

$$X^{(n)} = (1-\alpha)\sum_{k=0}^{n-1}\alpha^k \tilde{B}^k H(\tilde{A}^T)^k + \alpha^n \tilde{B}^n H(\tilde{A}^T)^n. \qquad (8)$$

This expansion could alternatively be generated by iterating for n steps the "unvec"ed version of (4) 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

$$H = \sum_{i=1}^{r}\sigma_i u_i {v_i}^T, \qquad (9)$$

where $r \leq \min(n_A, n_B)$ is the rank of H, and $\sigma_i > 0$, u_i, v_i for $i = 1, \dots, r$ 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 (9) into (8), we get

$$X^{(n)} = \sum_{i=1}^{r}\sigma_i\left[(1-\alpha)\sum_{k=0}^{n-1}\alpha^k \tilde{B}^k u_i v_i^T (\tilde{A}^T)^k + \alpha^n \tilde{B}^n u_i v_i^T (\tilde{A}^T)^n\right]. \qquad (10)$$

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

$$X^{(n)} = \sum_{i=1}^{r}\sigma_i\left[(1-\alpha)\sum_{k=0}^{n-1}\alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T\right]. \qquad (11)$$

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

$$X_i^{(n)} = \sigma_i\left[(1-\alpha)\sum_{k=0}^{n-1}\alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T\right]. \qquad (12)$$

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

$$X^{(n)} = \sum_{i=1}^{r} X_i^{(n)}. \qquad (13)$$

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

$$H = \sum_{i=1}^{s} w_i z_i^T, \qquad (14)$$

and, (12) and (13) can be modified, respectively, to

$$X_i^{(n)} = (1-\alpha)\sum_{k=0}^{n-1}\alpha^k w_i^{(k)} {z_i^{(k)}}^T + \alpha^n w_i^{(n)} {z_i^{(n)}}^T, \qquad (15)$$

and

$$X^{(n)} = \sum_{i=1}^{s} X_i^{(n)}. \qquad (16)$$

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 Algorithm 1.

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

```
1: compute $\tilde{A}$, $\tilde{B}$
2: zero $X^{(n)}$
3: for $i = 1$ to $s$ do
4:   $w_i^{(0)} \leftarrow w_i$
5:   $z_i^{(0)} \leftarrow z_i$
6:   for $k = 1$ to $n$ do
7:     $w_i^{(k)} \leftarrow \tilde{B} w_i^{(k-1)}$
8:     $z_i^{(k)} \leftarrow \tilde{A} z_i^{(k-1)}$
9:   end for
10:  for $k = 0$ to $n-1$ do
11:    $X^{(n)} \leftarrow X^{(n)} + (1-\alpha)\alpha^k w_i^{(k)} {z_i^{(k)}}^T$
12:  end for
13:  $X^{(n)} \leftarrow X^{(n)} + \alpha^n w_i^{(n)} {z_i^{(n)}}^T$
14: end for
```

3.1 Complexity Considerations

A straightforward, although naive, implementation of the similarity calculation for dense X would proceed along the lines of (4). We denote the number of nonzero 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 on the order of $nnz_A \times nnz_B$ floating point operations.

If we use the iteration kernel of (5) (triple-matrix products) as in [9], 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 over the naive implementation by a factor of the order of this ratio. The space complexity is dominated by an $O(n_A \times n_B)$ term—for storing the similarity matrix X—that is also an improvement over the naive implementation ($O(nnz_A \times nnz_B)$).

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

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

dominated by an $O(n_A \times n_B)$ term per iteration for each component. 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. Note that the memory requirements are of order $O(n_A \times n_B)$.

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 7, 8 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 presupposition of vanishing similarity scores for a large fraction of potential matches. For instance, in [9], 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 (Algorithm 2).

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).

Algorithm 2. NSD_Sparse

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

However, if no such candidate pairs are available an alternative, heuristic, sparsification scheme could be used: X is generated in row-wise fashion, and only a prespecified fraction of the largest entries in each row is retained, depending on the memory available.[2]

2. Preliminary numerical experiments that are part of a separate publication on parallelizing NSD [24] show that retaining, say, $< 3\%$ of such entries in each row of X yields up to 90 percent of the number of conserved edges for typical PPI networks (fly/yeast).

TABLE 1
Species for Which PPI Network Data (Undirected) is Used in Our Experiments

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

3.3 Remarks on the Use of Singular Value Decomposition

We note some important aspects of SVD, used in the previous section: the similarity matrix H is a nonnegative matrix. In the general case, some entries in vectors u_i, and v_i will be negative, due to orthonormality. Perron-Frobenius theorem ensures nonnegativity only of elements of vectors u_1, v_1. This implies that some component score matrices $X_i^{(n)}$ will have negative entries (a negative entry at the p,q position in matrix $X_i^{(n)}$ implies that the ith SVD triplet *penalizes* the matching of nodes $p \in V_B$ and $q \in V_A$). However, the sum of these entries, is guaranteed to be positive. Another inconvenience from negative entries is the difficulty in bounding the error in $X^{(n)}$, if using less than r terms in the SVD expansion. These difficulties can be overcome through the use of nonnegative factorizations [25], [26], or by treating the positive and negative elements in the factors separately [27]. However, these methods introduce additional considerations in terms of performance and cost.

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 data sets.

4.1 Timing Results (PPI Networks)

We first examine the computation time for our proposed technique. For this purpose, we use PPI and the IsoRank approach of Singh et al. for comparison (a native binary for IsoRank available is available with [28]). Details of data for seven species (networks) used in the experiment are provided in Table 1.

We also use Matlab codes from netalign [29], specifically their IsoRank and maximum weight bipartite matching implementations; these were originally written to support [9]. 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 [9]). 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

TABLE 2
Timing Results from Running NSD (Column 2)—Implemented in Matlab—against Reference IsoRank Implementation from [28] (Column 4), for All Pairs of Species (Column 1)

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

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 3rd column for cases with larger instance pairs and thus the most time consuming ones (not shown)). $a = 1.0$, *20 iterations.*

conditions (outer product of suitably normalized 1's for each pair) and execute 20 iterations in all runs. For undirected networks, setting $\alpha = 1.0$ in (5) yields $X \leftarrow \tilde{B}X\tilde{A}^T$. This iteration is known to converge to $B1(A1)^T$ (modulo a suitable scaling factor). This steady-state solution corresponds to a similarity matrix with x_{ij} entries analogous to the products of the degrees of nodes $b_i \in V_B$ and $a_j \in V_A$. Consequently, high degree nodes are matched first. We report on the state of the similarity diffusion process for uniform initial similarities for varying number of hops. Note also 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.

The choice of α parameter does not have any performance implications. For example, setting $\alpha = 1$ is the simplest choice when no node label similarity information is available. In [30], the authors advocate the use of $\alpha = 0.6$ in order to reduce the effect of noisy network data. We choose $\alpha = 0.8$ in most experiments, based on their findings for the parameter value maximizing the number of conserved edges, and also because PageRank calculations—with IsoRank being essentially PageRank over a product graph—typically choose α's in this range (e.g., Google uses $\alpha = 0.85$).

The selection of a fixed number of iterations (20) in our experiments does not guarantee convergence. For example, in the $\|\cdot\|_1$ norm (norm-1) of the difference of successive iterates for all α—for larger α, more iterations are needed. A more direct stopping criterion would be the maximization of a quality measure such as the number of conserved edges. However, this is an expensive computation in itself. For $\alpha = 1.0$, 20 iterations result in a norm-1 of 10^{-2}; exact convergence on the other hand produces 410 conserved edges, although 21 iterations result in 439 conserved edges. For $\alpha = 0.8$ at convergence (needing 32 iterations, with norm-1 of 5×10^{-6}), we get 495 conserved edges and for 19 iterations we get an improved "solution" (524 conserved edges). For $\alpha = 0.6$ convergence can be reached in only 15 iterations (429 edges), however, with only five steps we can get 465 conserved edges. This phenomenon of reaching satisfactory quality results with fewer iterations—making convergence in the strict norm-1 sense less important—is expected due to the low diameter of the PPI networks (in [31] average diameters of 2.29 to 6.23 are reported).

Unfortunately the native code from Singh and Berger [28] 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 Fig. 2.

Each native IsoRank run computes the similarity matrix and calculates two alternate best pair matchings (fourth column in Table 2). The corresponding timings for computing the similarity matrix alone with our IsoRank NSD Matlab codes are presented in the second 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 third 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 third 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 Fig. 3 illustrates the respective NSD

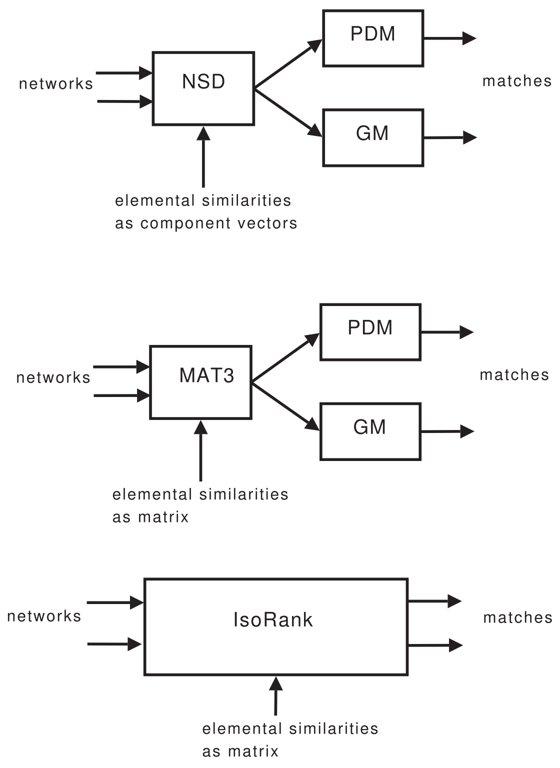

networks
NSD
PDM
GM
matches
elemental similarities as component vectors
networks
MAT3
PDM
GM
matches
elemental similarities as matrix
networks
IsoRank
matches
elemental similarities as matrix

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

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 node label similarities (column 2) against MAT3 (column 3) and IsoRank (column 6); see Table 3 ($a = 0.80$). We also include timings for both match

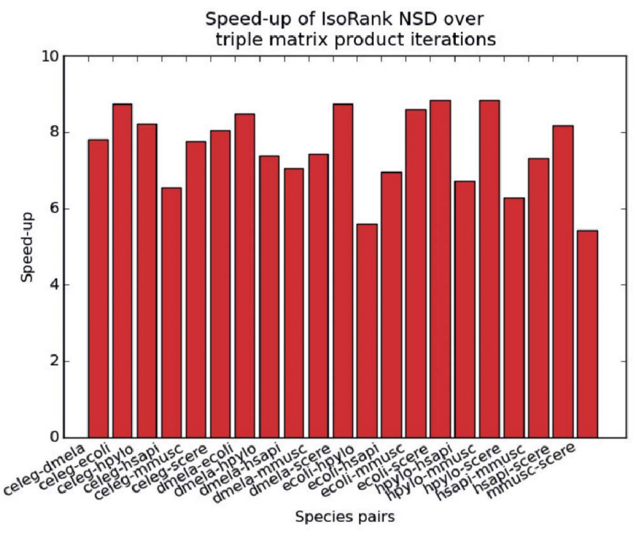


Fig. 3. Speedups in computing similarity matrices for all species pairs with NSD against MAT3 (from [29]). Here $a = 1.0$ and NSD is implemented in Matlab. Note that by porting it to Java, 25 fold speedups are observed (for $a = 0.80$, see columns 2 and 3 in Table 3).

TABLE 3
Timing Results (in Seconds) from Running NSD (Column 2)—Implemented in Java—against Reference IsoRank Implementation from [28] (Column 6) and the (5) Approach (MAT3, Column 3), for All Pairs of Species (Column 1)

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

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).

extraction algorithms: PMD (column 4 in Matlab) and GM (column 5 in Java). This data clearly show that NSD outperforms the similarity matrix implementations in MAT3 by a factor of 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. Further discussion on the $s > 1$ case can be found in Section 4.3.

4.2 Self-Similarity in Networks

We also use the Facebook networks referenced in [32], each describing the corresponding USA university Facebook community (Table 4[3]). 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

3. Although not large, these test data sets are practical since they contain links to internal nodes only; trying to enforce this property to other larger Facebook data sets 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.

TABLE 4
Facebook Networks Used in the Experiments

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

Number of nodes and edges are given in each case.

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.

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 [33]) 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

TABLE 5
Maximum Common Connected Subgraphs (MCCS) Characteristics (Columns 2, 3, 4) Discovered After Postprocessing NSD Results from Each Network (Column 1) Paired with Itself

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

Average degree of each node in the original graph is also given (column 5).

TABLE 6
Characteristics of Web Graphs Used in Experiments

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

Particularly eu-2005 data set was gathered by UbiCrawler [36].

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 data sets are available in an efficient compressed format utilizing the special techniques described in [34] and their manipulation was made possible by adapting our NSD method also to interface their provided software framework, WebGraph [35].

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 (12) 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 Fig. 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 percent, respectively, for cnr-2000 and eu-2005)—also implying the complementary percentages of matched nodes participate in isolated triangles, edges, or nodes in the alignment graphs.

The matching procedure rules out pairing of a node with itself. However, it permits matching nodes from distant regions in the graph. The development of common connected subgraphs, also including this pair node, would reveal repeated substructures—subgraph "copies"—in these distant regions; some of which appear to be extensive. This follows from the fact that *one* common connected subgraph in the alignment graph essentially represents *two* subgraphs in the original graph with a satisfactory edge overlap for the underlying matching of their nodes. Therefore, repeating

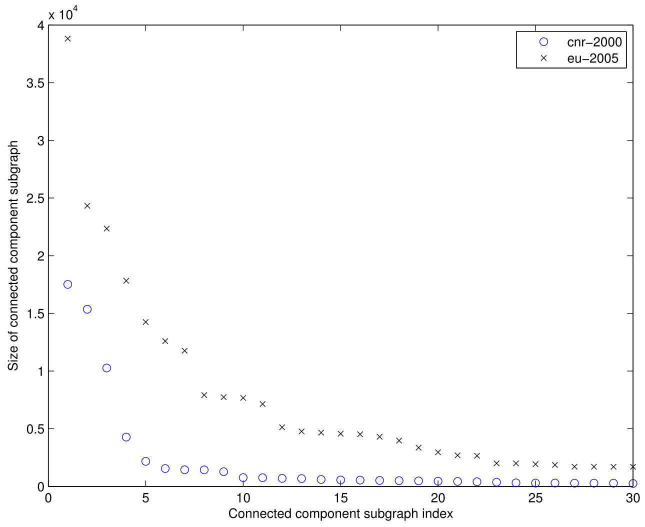
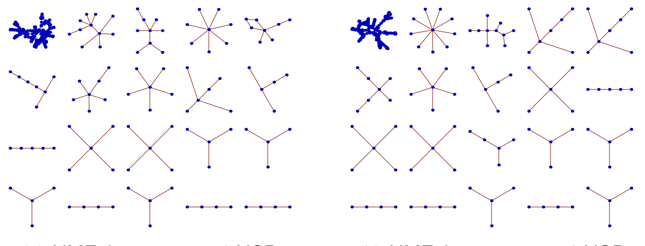
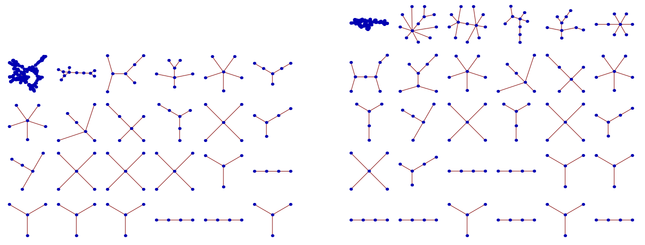


Fig. 4. The sizes of the 30 of the largest common connected subgraphs from self-similarity runs of NSD on Webgraphs. There were detected subgraphs of sizes up to 17,520 nodes (for cnr-2000) and 38,818 nodes (for eu-2005).



(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).

linkage patterns within each graph can be found (self-similarity).

It should be noted that NSD is applied to directed graphs, and therefore the scores are constrained to similarities in the structure of incoming links only.

4.3 The Effect of Multiple Components ($s > 1$)

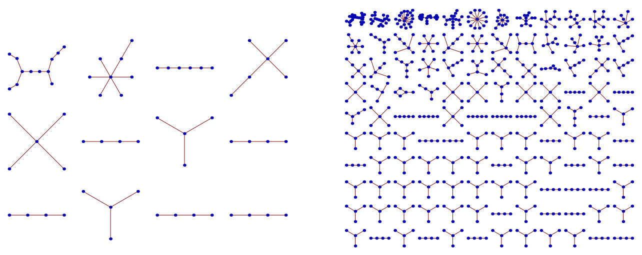
We comment on the effect of using multiple components ($s > 1$) on the structure of common connected subgraphs extracted. Using BLAST (sequence) similarity scores as the matrix H of node label similarities for the fly/yeast PPI networks we computed all common connected subgraphs of ≥ 4 nodes ($\alpha = 0.80$, 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. (Fig. 5).
- Decomposing node label 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. (Fig. 6).
- Decomposing node label similarity into the dominant $s = 500$ and $s = 1{,}000$ components by SVD and then successively running NSD code for computing the similarity scores and GM for extracting the matches. (Fig. 7).

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

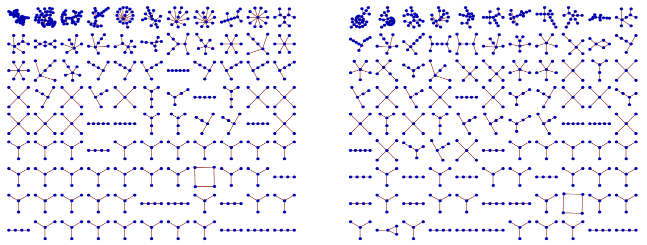
- Count the number of edges in the alignment graph (conserved edges). Each conserved edge implies matching the corresponding edges connecting the elements of the matching pairs at its endpoints in the input networks. So node matching naturally leads to edge matching.
- Compute the size of the connected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

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.



(a) Node label similarities H (b) MAT3

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



(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 $s = 1{,}000$ components of H as computed by SVD.

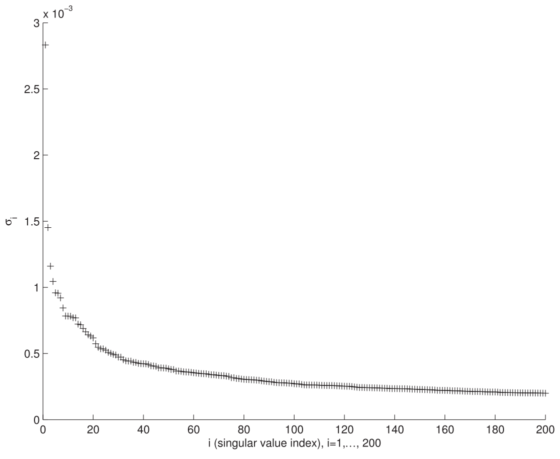


Fig. 8. Plot of the largest 200 singular values σ_i of H, the matrix of sequence similarities for the fly/yeast networks.

In our experiments SVD decompositions for $s = 5$, 10, 500, 1,000 components coupled with NSD, respectively, give rise to 514, 537, 1,143, 1,236 conserved edges compared to 1,455 edges from MAT3. For the fly/yeast networks, a low-rank approximation for the sequence similarities in the respective H matrix ($s = 5$ components) captures about 35 percent of the conserved edges produced by the exact H. This is due to the application-specific structure of the SVD spectrum for this matrix H: if sorted, its singular values σ_is do not decay fast enough. There is a persisting background plateau of similar magnitude contributions after the rapid decrease for $i < 20$ (please see Fig. 8). However, these decompositions 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, 1,000) 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 node label 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 node label similarity scores used for small s) gives a "smooth" succession of results in the SVD case. Also using NMF that—by definition—generates nonnegative node label 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. The decomposition happens with respect to 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. The biggest advantages in speed can be gained when preferences (that also serve as the initial conditions in the proposed iteration) can be written as sums of small number 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 preprocessing/input parameters (α, preferences) and the appropriate postprocessing 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 "uncoupling" 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.

Acknowledgments

This work was supported by the US National Science Foundation grants DBI-0835677, DMS-0800568 and through the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370. The authors would like to sincerely acknowledge the comments and concerns of the reviewers, which served to enhance the quality of the manuscript significantly.

References

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

[2] G. Jeh and J. Widom, "SimRank: A Measure of Structural-Context Similarity," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 538-543, http://portal.acm.org/citation.cfm?id=775126, 2002,

[3] R. Singh, J. Xu, and B. Berger, "Global Alignment of Multiple Protein Interaction Networks with Application to Functional Orthology Detection," *Proc. Nat'l Academy of Sciences USA,* vol. 105, no. 35, pp. 12763-12768, 2008.

[4] J.Y. Pan, H.J. Yang, C. Faloutsos, and P. Duygulu, "Automatic Multimedia Cross-Modal Correlation Discovery," *Proc. 10th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 653-658, 2004.

[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] N. Pržulj, "Biological Network Comparison Using Graphlet Degree Distribution," *Bioinformatics,* vol. 23, no. 2, pp. e177-e183, http://bioinformatics.oxfordjournals.org/content/23/2/e177.abstract, Jan. 2007.

[8] T. Milenković and N. Pržulj, "Uncovering Biological Network Function via Graphlet Degree Signatures," *Cancer Informatics,* vol. 6, pp. 257-273, http://www.ncbi.nlm.nih.gov/pubmed/19259413, 2008.

[9] M. Bayati, M. Gerritsen, D.F. Gleich, A. Saberi, and Y. Wang, "Algorithms for Large, Sparse Network Alignment Problems," *Proc. IEEE Ninth Int'l Conf. Data Mining (ICDM '09),* pp. 705-710, 2009.

[10] L. Getoor and C. Diehl, "Link Mining: A Survey," *SigKDD Explorations,* special issue on link mining, vol. 7, no. 2, pp. 3-12, Dec. 2005.

[11] C. Silverstein, S. Brin, and R. Motwani, "Beyond Market Baskets: Generalizing Association Rules to Dependence Rules," *Data Mining Knowledge and Discovery,* vol. 2, no. 1, pp. 39-68, 1998.

[12] C.H.Q. Ding, X. He, P. Husbands, H. Zha, and H.D. Simon, "PageRank, HITS and a Unified Framework for Link Analysis," *Proc. 25th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '02),* pp. 353-354, 2002.

[13] A.Y. Ng, A.X. Zheng, and M.I. Jordan, "Link Analysis, Eigenvectors and Stability," *Proc. 17th Int'l Joint Conf. Artificial Intelligence (IJCAI '01),* pp. 903-910, 2001.

[14] K. Bharat and M.R. Henzinger, "Improved Algorithms for Topic Distillation in a Hyperlinked Environment," *Proc. 21st Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '98),* pp. 104-111, 1998.

[15] 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, nos. 1-7, pp. 65-74, 1998.

[16] A.X. Zheng, A.Y. Ng, and M.I. Jordan, "Stable Algorithms for Link Analysis," *Proc. 24th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '01),* pp. 258-266, 2001.

[17] L.A. Zager and G.C. Verghese, "Graph Similarity Scoring and Matching," *Applied Math. Letter,* vol. 21, no. 1, pp. 86-94, 2008.

[18] M. Rupp, E. Proschak, and G. Schneider, "Kernel Approach to Molecular Similarity Based on Iterative Graph Similarity," *J. Chemical Information and Modeling,* vol. 47, pp. 2280-2286, 2007.

[19] G. Kollias and A. Grama, "Parallel Network Similarity Decomposition," *Parallel Matrix Algorithms and Applications,* 2010.

[20] L. Grasedyck, "Existence of a Low Rank or H-Matrix Approximant to the Solution of a Sylvester Equation," *Numerical Linear Algebra with Applications,* vol. 11, no. 4, pp. 371-389, http://onlinelibrary.wiley.com.login.ezproxy.lib.purdue.edu/doi/10.1002/nla.366/pdf, 2004.

[21] 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," *Proc. 13th Int'l Conf. Extending Database Technology,* pp. 465-476, 2010.

[22] D. Liben-Nowell and J. Kleinberg, "The Link-Prediction Problem for Social Networks," *J. Am. Soc. for Information Science and Technology,* vol. 58, no. 7, pp. 1019-1031, http://onlinelibrary.wiley.com.login.ezproxy.lib.purdue.edu/doi/10.1002/asi.20591/full, 2007.

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

[24] G. Kollias, M. Sathe, O. Schenk, and A. Grama, "Fast Parallel Algorithms for Graph Similarity and Matching," technical report, Purdue Univ., 2012.

[25] 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, http://dx.doi.org/10.1038/44565, Oct. 1999.

[26] S.A. Vavasis, "On the Complexity of Nonnegative Matrix Factorization," *SIAM J. Optimization,* vol. 20, pp. 1364-1377, http://dx.doi.org/10.1137/070709967, Oct. 2009.

[27] 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.

[28] R. Singh and B. Berger, "IsoRank and IsoRankN," http://groups.csail.mit.edu/cb/mna/, 2012.

[29] D.F. Gleich, "Netalign: Network Alignment Codes," http://www.stanford.edu/dgleich/publications/2009/netalign/, 2012.

[30] R. Singh, J. Xu, and B. Berger, "Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology," *Proc. 11th Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB '07),* vol. 4453, pp. 16-31, http://dx.doi.org/10.1007/978-3-540-71681-5_2, 2007.

[31] K. lok Ng and C. Huang, "A Cross-Species Study of the Protein-Protein Interaction Networks via the Random Graph Approach," *Proc. IEEE Fourth Symp. Bioinformatics and Bioeng. (BIBE),* pp. 561-567, 2004.

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

[33] NetworkX Project Website, https://networkx.lanl.gov/, 2012.

[34] P. Boldi and S. Vigna, "The WebGraph Framework I: Compression Techniques," *Proc. 13th Int'l World Wide Web Conf. (WWW '04),* pp. 595-601, 2004.

[35] WebGraph Project Website," http://webgraph.dsi.unimi.it/, 2012.

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

Giorgos Kollias received the BSc degree in physics in 2000, the MSc degree in computational science in 2002 from the University of Athens, Greece, and the PhD degree in computer science from the University of Patras, Greece, in 2009. He is currently a postdoctoral researcher in the Center for Science of Information and the Computer Science Department at Purdue University. His research interests include dynamical systems, problem solving environments, graph analysis, and parallel computing.

Shahin Mohammadi received the BSc degree in computer science in 2008 and joined Purdue University in 2010 as a graduate student. His research interests include parallel computing and computational biology. His current work is focused on development of efficient algorithms and models for problems in computational biology.

Ananth Grama received the PhD degree from the University of Minnesota in 1996. He is currently a professor of computer sciences and an associate director of the Center for Science of Information at Purdue University. His research interests include areas of parallel and distributed computing architectures, algorithms, and applications. On these topics, he has authored several papers and texts. He is a member of the American Association for Advancement of Sciences.