2012

Fast Parallel Algorithms for Graph Similarity and Matching

Georgios Kollias
Purdue University, gkollias@purdue.edu

Madan Sathe
University of Basel

Olaf Schenk
University of Lugano

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

Report Number:
12-010

Fast Parallel Algorithms for Graph Similarity and Matching

Giorgos Kollias[a], Madan Sathe[b], Olaf Schenk[c], Ananth Grama[a]

[a]Department of Computer Science and Center for Science of Information, Purdue University, USA
[b]Department of Mathematics and Computer Science, University of Basel, Switzerland
[c]Institute of Computational Science, University of Lugano, Switzerland

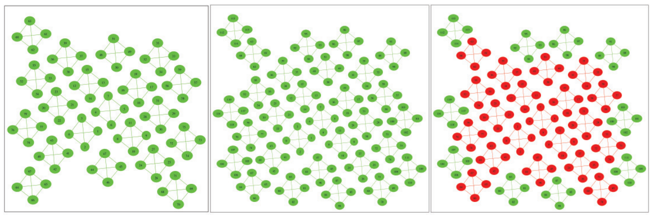
Keywords: graph alignment, vertex similarity, parallel matching, auction algorithm

Abstract

With widespread availability of graph-structured data from sources ranging from social networks to biochemical processes, there is increasing need for efficient and effective graph analyses techniques. Graphs with millions of vertices and beyond are commonplace, necessitating both efficient serial algorithms, as well as scalable parallel formulations. This paper addresses the problem of global graph alignment on supercomputer-class clusters. Given two graphs (or two instances of the same graph), we define graph alignment as a mapping of each vertex in the first graph to a unique vertex in the second graph so as to optimize a given similarity-based cost function[1]. Graph alignment is typically implemented in two steps – in the first step, a similarity matrix is computed. Entries in the matrix quantify similarity of node pairs, one chosen from each graph. In the second step, similar vertices are extracted through a bipartite matching algorithm applied to the similarity matrix. Using a state of the art serial algorithm for similarity matrix computation called Network Similarity Decomposition (NSD), we derive corresponding parallel formulations. Coupling this parallel similarity algorithm with a parallel auction-based bipartite matching technique, we derive a complete graph matching pipeline that is highly efficient and scalable. We validate the performance of our integrated approach on a large, supercomputer-class cluster and diverse graph instances (including Protein Interaction (PPI) networks, Web graphs, and Wikipedia link structures). Experimental results demonstrate that our algorithms scale to large machine configurations and problem instances.

Email addresses: `gkollias@purdue.edu` (Giorgos Kollias), `madan.sathe@unibas.ch` (Madan Sathe), `olaf.schenk@unibas.ch` (Olaf Schenk), `ayg@cs.purdue.edu` (Ananth Grama)

[1]In the sequel, we'll be using the word alignment as a synonym for global graph alignment; this is in contrast to local graph alignment that permits a vertex to have different pairings in feasible local alignments, making it an inherently ambiguous process



(a) (b) (c)

Figure 1: Vertex similarity : The first graph (Figure 1a) when matched against the graph in Figure 1b results in the vertex mapping shown in Figure 1c. In this figure, matching vertices are colored differently (red). This example illustrates the special case of matching a graph (Figure 1b) with its subgraph (Figure 1a).

Specifically, we show that our integrated pipeline enables the alignment of networks of sizes two orders of magnitude larger than currently possible (millions of vertices, tens of millions of edges).

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. Graphs in current databases often scale to millions of vertices and beyond, requiring efficient serial algorithms as well as scalable parallel formulations. Graph kernels such as traversals, centrality computations, and modularity have been studied in both serial and parallel contexts [12, 8, 45, 21]. The problem of matching vertices across graphs based on their topological similarity is more computationally expensive. This follows from the fact that topological similarity of a pair of nodes selected from two graphs, respectively, is determined by their network contexts (broader neighborhood in graphs). Consequently, parallel formulations determine the feasibility envelope for such problems.

The graph alignment problem (please see Fig. 1) can be informally stated as follows: given two graphs, "how similar is each vertex in the first graph to each vertex in the second?" or "what is the best match for each vertex in the first graph to a vertex in the second graph?". A complete solution to the first problem takes the form of a similarity matrix X; its entry x_{ij} corresponds to the similarity of vertex i in the first graph to vertex j in the second. Solution to the second question takes the similarity matrix X and uses bipartite matching to map vertices of the first graph to similar vertices in the second graph, to maximize overall match across the graphs.

To illustrate the problem, we consider the graph in Fig. 2. A similarity matrix (computed using the IsoRank method summarized later in the paper) for

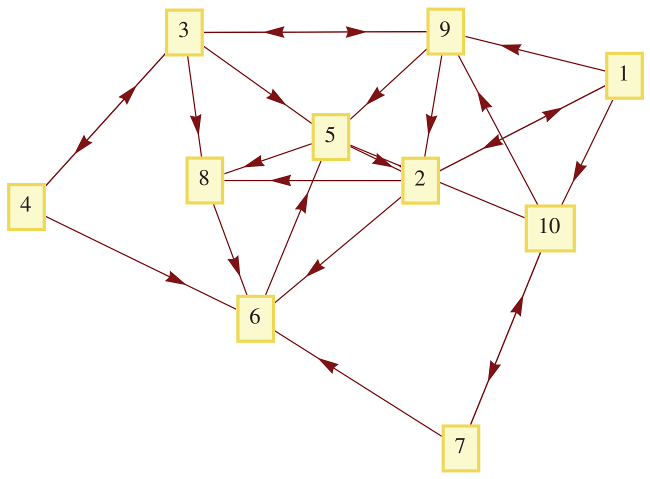


Figure 2: The example graph used for illustrating the alignment process.

this graph aligned to itself (self-similarity) can be computed where a normalization step is also applied after zeroing diagonal elements (in order to preclude the trivial solution of matching each node with itself). More specifically, we then multiply each row by the inverse of the sum of its entries and also with the percentage of this sum over the total matrix sum. The resulting similarity matrix X becomes:

$$X = \begin{pmatrix}
0.00000 & 0.00924 & 0.00371 & 0.00286 & 0.01394 & 0.01312 & 0.00304 & 0.01029 & 0.00432 & 0.00383 \\
0.00924 & 0.00000 & 0.00686 & 0.00396 & 0.03930 & 0.03622 & 0.00454 & 0.02773 & 0.00874 & 0.00715 \\
0.00371 & 0.00686 & 0.00000 & 0.00301 & 0.01004 & 0.00975 & 0.00320 & 0.00714 & 0.00458 & 0.00416 \\
0.00286 & 0.00396 & 0.00301 & 0.00000 & 0.00515 & 0.00491 & 0.00268 & 0.00408 & 0.00314 & 0.00297 \\
0.01394 & 0.03930 & 0.01004 & 0.00515 & 0.00000 & 0.05924 & 0.00621 & 0.04455 & 0.01343 & 0.01041 \\
0.01312 & 0.03622 & 0.00975 & 0.00491 & 0.05924 & 0.00000 & 0.00586 & 0.04096 & 0.01272 & 0.01034 \\
0.00304 & 0.00454 & 0.00320 & 0.00268 & 0.00621 & 0.00586 & 0.00000 & 0.00471 & 0.00339 & 0.00317 \\
0.01029 & 0.02773 & 0.00714 & 0.00408 & 0.04455 & 0.04096 & 0.00471 & 0.00000 & 0.00930 & 0.00750 \\
0.00432 & 0.00874 & 0.00458 & 0.00314 & 0.01343 & 0.01272 & 0.00339 & 0.00930 & 0.00000 & 0.00455 \\
0.00383 & 0.00715 & 0.00416 & 0.00297 & 0.01041 & 0.01034 & 0.00317 & 0.00750 & 0.00455 & 0.00000
\end{pmatrix}$$

Next we apply a matching process and get the following pairs of "similar" vertices: $(5, 6)$, $(2, 8)$, $(3, 9)$, $(1, 10)$ and $(4, 7)$. Quite interestingly the same matching pairs are produced also in the case when no normalization is applied after zeroing the similarity matrix diagonal.

This methodology has several important applications. In analysis of biomolecular networks, nodes represent proteins and edges represent functional association between proteins (binding, co-localization, etc.). A matching computed using the above method reveals proteins that have similar interaction profiles, and consequently are functionally similar. This is a complementary and important similarity measure to traditional sequence-based similarity for proteins.

A number of formulations and solutions exist in literature to both the similarity computation and matching problems [59, 7, 35, 51, 5]. An important class of methods relies on the notion that the similarity of two vertices is determined by the similarity of their neighbors. Variants within this class differ on their treatment of dissimilar neighbors (normalization), a-priori vertex similarity (also called elemental similarity), the topological scope (how much of the neighborhood is incorporated into the similarity score), and the iterative procedure to compute similarity. Our recent work in the area has resulted in the development of a serial algorithm called Network Similarity Decomposition (NSD)[30]. NSD can be viewed as an accelerator for a large class of iterative similarity computation algorithms. It has been shown to reduce the computational cost of traditional algorithms by over three orders of magnitude in specific instances.

Given a similarity matrix, algorithms for bipartite matching have a rich history in theoretical computer science. Proposed solutions range from the classic Hungarian and greedy methods to auction-based techniques. Serial and parallel computing tradeoffs of many of these methods have also been studied in prior work. Even with reduced computational cost of NSD and auction-based bipartite matching, for large graphs of interest (10^6 vertices and beyond), it is necessary to exploit scalable parallelism to achieve acceptable performance. This paper focuses on parallel formulations of the graph alignment problem. In particular, it demonstrates that parallel NSD formulations have low communication and synchronization overheads– making them ideal for large-scale parallel platforms. Furthermore, the data decompositions induced by NSD similarity computations flow naturally into our parallel auction-based matching algorithm. This integrated pipeline is shown to have excellent performance and scalability on large-scale parallel platforms and diverse applications. We use this software pipeline to solve some of the largest similarity/ matching problems (over two orders of magnitude larger than those reported earlier) on thousands of processing cores. These results have significant implications for applications ranging from systems biology to social network analysis.

The rest of this paper is organized as follows: we overview related work in Section 2. Section 3 presents a brief description of serial NSD and auction-based matching algorithms. Section 4 presents parallel NSD, the need for sparsification, parallel auction-based matching, and an efficient integration of the two into a single workflow. In Section 5, we present comprehensive performance and scalability results for parallel versions of similarity computations from large-scale experiments (networks with millions of vertices on thousands of processing cores) for the integrated pipeline. Concluding remarks and avenues for future work are presented in Section 6.

2. Related Results

In (serial) graph alignment, matrix similarity computation has traditionally been the computational bottleneck, particularly, when heuristic methods are used for bipartite graph matching to post-process similarity scores. However, as a result of the reduction in time using NSD for similarity computation, bipartite matching represents the dominant computational cost. Consequently, an efficient pipeline must integrate a parallel bipartite weighted graph matching algorithm with a parallel NSD-accelerated similarity computation algorithm.

We highlight here related efforts on the two major components – algorithms for computing similarity matrix X illustrated above, and algorithms that compute pair-wise correspondences across the two graphs using the similarity matrices. To the best of our knowledge, there exist no parallel formulations that integrate these two stages of the pipeline. We provide a brief overview of the serial methods, followed by preliminary efforts at parallelizing these methods.

2.1. Computing Graph Similarities

Graph similarity computations can be broadly classified into two groups. In the first group of methods, the outcome of the computation is a single similarity score $sim(G, G')$, typically normalized in the range $[0, 1]$. This similarity score indicates how similar two graphs G, G' are *in their entirety*. Papadimitriou et al. [47] present an excellent survey of approaches in this group. In this group, the Vertex/edge overlap (VEO) method is based on the principle that two graphs are similar if they share many vertices and/or edges. This can be quantified using a form of edit distance. In vertex ranking (VR), the similarity of two graphs is based on the similarity of the ranking of their vertices, as quantified by a rank correlation method such as Spearman's ρ. Vertex/edge vector similarity (VS) compares two vectors of weights encoding the quality of vertices/ edges in the two graphs. In sequence similarity (SeqS), the similarity of two graphs is determined by the extent to which sequences of short paths of vertices and edges are shared. Signature similarity (SS) translates each graph into a set of features that are randomly projected to lower-dimensional feature space (signatures); the resulting vectors are then compared.

In the second group of methods, the outcome of the computations is a set of numbers x_{ij}, representing the similarity of each vertex i in the first graph to every vertex j in the second graph. This notion of node-wise similarity can be extended to edge-wise similarity, or to similarity of small subgraphs in the two graphs. This second class of methods reflects a finer-grained notion of graph similarity. We can further categorize methods in this group as local or global methods. Local methods attempt to reward similarity among small subgraphs, without penalizing for dissimilarities over non-aligned parts of the graph [27, 32, 56, 36, 62]. A number of local methods have been proposed and used in the context of diverse applications. These methods range from subgraph isomorphism-based methods [36, 62] to greedy methods on sparsified product graphs [32]. Local algorithms, such as PathBlast [27], NetworkBlast [26], MaWISh [32] and Graemlin 1.0 [18], typically allow one vertex to have different pairings in feasible local alignments. Consequently, they are inherently ambiguous and their scoring functions are based on heuristics associated with parameter choice, evolutionary models, or other randomized techniques. Global methods, on the other hand, consider a cost function computed over all vertex alignments [42, 52, 59].

The focus of this paper is on global methods. A large subset of these methods can be viewed as computing the rank of a vertex (in a PageRank [46, 58] sense) in the product graph of the input networks. Of the PageRank-based methods for network similarity, the IsoRank algorithm [59] computes vertex similarity scores by integrating both vertex attributes and topological similarities. The graph kernel approach [54], uses characteristics of networks (bounded degree) to specialize Page-Rank to their target structures. Specifically, in each iteration of similarity update, optimal mappings between neighborhoods of each pair of vertices are computed to determine topological similarity. In the GRAAL family of algorithms - GRAAL [33], H-GRAAL [41], MI-GRAAL [34], C-GRAAL [40] - the "seed and extend" idea is utilized, basically driven by node similarities, as computed by affinities to local connectivity structures. Building on its local counterpart, Graemlin 2.0 [17] integrates a priori known protein sequence similarities (node similarities) and phylogenetic (evolutionary) relations. In NetAlignBP [3], the belief propagation technique is applied, interestingly allowing the consideration of only a subset of potential pairs; Lagrangian, Markov random field and integer quadratic programming have also been proposed [16, 2, 37, 28]. SimRank is a generic method introduced in [24] for computing structural-context similarity between vertices of a single graph.

An excellent survey of early serial results in this area is provided in [19]. Our parallel similarity construction relies on an acceleration scheme, called Network Similarity Decomposition (NSD) [30], which relies on low-rank decompositions of the initial similarity matrix to decouple the matrix construction process. This acceleration has been shown to yield orders of magnitude improvement in serial runtime, based on the size of the networks. We provide an overview of NSD acceleration in Section 3 to motivate our parallel formulations.

Parallel formulations of graph similarity computations have primarily focused on combinatorial methods aimed at identifying subgraph isomorphisms (or correspondingly, connected components in product graphs). Among these, the early efforts [1, 48, 57] utilize different parallel platforms (data parallel and messaging) for extracting isomorphic subgraphs or maximum cliques. For the special case of SimRank, for similarity computations within a single graph instance, [22] discuss an iterative aggregation technique exploiting the inherent parallelism and high memory bandwidth of graphics processing units (GPUs). We are not aware of any parallelizations that utilize the decoupled accelerations in NSD.

2.1.1. Applications of Graph Similarity

There is an expanding body of research in graph alignment methods, especially for biological networks with applications in disease discovery, extending protein functional annotations, identifying conserved modules across species, and studying evolutionary trajectories [43]. Beyond applications in biology, similarity computations can be traced back to [60] for matching chemical structures on a node-by-node basis, or in [61] for comparing electrical circuits. [20] computes similarities between computer programs, analyzing their parse tree representations. In [39], vertex-level granularity scores particularly suited for database schema comparisons are introduced. Graph similarity can also serve as a model for reinterpreting the HITS [29] algorithm for ranking Web pages;

hub and authority scores for a page can be cast as the similarity values of pairs containing this page-node and the nodes of a trivial graph consisting of a single directed edge. This is an interesting observation in [7], which also uses a provably converging iterative procedure for computing similarity scores between vertices with application to synonym extraction. Expanding on these ideas, [63] investigate the integration of edge information to develop coupled node-edge scoring.

2.2. Weighted Matching Algorithms in Bipartite Graphs

Weighted graph matching algorithms extract a matching M of similar vertices subject to the constraint that a vertex is an endpoint of at most one matching edge. A typical objective of matching algorithms is to find a matching where the weight of the matching, i.e., the sum over the matched edges, is maximized.

There are two broad classes of algorithms that achieve a matching with a maximum weight:

Approximate weighted matching algorithms compute a maximal matching, i.e., no edge can be added to M without violating the matching property, with a maximum weight. A well-known representative of this class is a simple greedy heuristic that proceeds as follows: store the edge weights in a list, sort the weights in a decreasing order, and insert edges in the matching set starting from the largest to the smallest entry conserving the feasibility of the matching. There exists a linear-time implementation of this $^1/_2$-approximation algorithm [51]. Sophisticated approaches such as $^2/_3$- or $^3/_4$-approximation have been published by several authors (see e.g., [14, 50]). Attempts to parallelize these methods have been reported in [11, 23, 38, 49].

Exact weighted matching algorithms obtain a maximum matching, i.e., a matching with the largest possible number of edges, with a maximum weight. The maximum weighted matching problem can be optimally solved in polynomial time using the idea of the augmenting path. An augmenting path is a path that has odd length, its ends are not in M, and its edges are alternatively out of and in M. Implementations based on the concept are, for instance, the Hungarian method and its variants [15, 25, 35, 44], or auction-based matching algorithms [5]. In a different approach, the matching problem can be formulated as a linear program, the well-known *linear sum assignment problem*, and solution techniques like the simplex algorithm or interior-point methods can be applied [9].

One of the interesting candidates for massively parallel platforms is the auction algorithm. Previous efforts to develop a parallel auction algorithm have resulted in formulations for both shared- and messaging platforms [4, 10, 53]. Recently, a highly scalable distributed auction algorithm has been developed that computes weighted matchings on sparse and dense bipartite graphs running on hundreds of compute nodes, while efficiently using multi-cores on each compute node [55]. This formulation provides the basis for the matching component of our algorithmic workflow.

3. Serial Algorithm for Similarity Matrix Computation and Auction-Based Matching

We first provide necessary background on the serial algorithms for constructing the similarity matrix, and the auction-based scheme for bipartite matching. Please note that this description is not meant to be comprehensive, rather, we provide sufficient details to motivate our parallel formulations. We refer the readers to [30, 55] for more details on these methods.

3.1. Terminology and Preliminaries

We represent a graph $G_A = (V_A, E_A)$ by its adjacency matrix A, where $a_{ij} = 1$ iff vertex i points to vertex j, indicated by $i \to j$, and zero otherwise. V_A and E_A denote the vertices and edges of G_A respectively, and $n_A = |V_A|$. 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 also introduce operator $vec(\cdot)$ for stacking matrix columns into a vector, as well as its associated "inverse" $unvec(\cdot)$ operator for re-assembling the matrix.

3.2. Network Similarity Decomposition (NSD)

In [59], an iterative procedure of the following form is proposed:

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

Here $\otimes$ denotes the Kronecker product of matrices, $x = vec(X)$, with x_{ij} as previously defined, and $h = vec(H)$, with element h_{ij} of matrix H corresponding to the elemental similarity score between vertex $i \in V_B$ and $j \in V_A$ (matrix H codes a-priori similarity of vertices). The vector h is normalized to unity. Successive iterates scale topological similarity and elemental similarity of vertices by factors $\alpha \leq 1$ and $1 - \alpha$, respectively. Utilizing the property of Kronecker products, $AXB = unvec((B^T \otimes A)x)$, for "unvec"ing, Equation 1 can also be expressed in terms of a triple-matrix kernel as

$$X \leftarrow \alpha \tilde{B} X \tilde{A}^T + (1-\alpha) H. \tag{2}$$

NSD acceleration of IsoRank relies on low-rank representations of the H matrix. We start 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), is a well established method that can be used for this purpose, enabling us to write:

$$H = \sum_{i=1}^{r} \sigma_i u_i {v_i}^T, \tag{3}$$

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_i vectors ($v_i v_j^T = \delta_{ij}$).

Using (2) and (3), after suitable manipulations, we get:

$$X^{(n)} = \sum_{i=1}^{r} \sigma_i \left[(1-\alpha) \sum_{k=0}^{n-1} \alpha^k \tilde{B}^k u_i v_i^T (\tilde{A}^T)^k + \alpha^n \tilde{B}^n u_i v_i^T (\tilde{A}^T)^n \right], \tag{4}$$

where parenthesized superscripts denote the iteration step. Setting $u_i^{(k)} = \tilde{B}^k u_i$ and $v_i^{(k)} = \tilde{A}^k v_i$, we obtain

$$X^{(n)} = \sum_{i=1}^{r} \sigma_i \left[(1-\alpha) \sum_{k=0}^{n-1} \alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T \right] \tag{5}$$

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

$$X^{(n)} = \sum_{i=1}^{r} X_i^{(n)} \tag{6}$$

where $X_i^{(n)}$ are computed separately separately, from each SVD triplet (σ_i, u_i, v_i)

$$X_i^{(n)} = \sigma_i \left[(1-\alpha) \sum_{k=0}^{n-1} \alpha^k u_i^{(k)} {v_i^{(k)}}^T + \alpha^n u_i^{(n)} {v_i^{(n)}}^T \right] \tag{7}$$

We refer readers to [30] for details of these derivations and their associated performance improvements in serial runtime. 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:

$$H = \sum_{i=1}^{s} w_i z_i^T, \tag{8}$$

where the number of components s does not necessarily coincide with r, the rank of H ($s \geq r$ for exact decompositions). Any decomposition into outer products (including non-orthogonal decompositions such as those from clustering or Non-negative Matrix Factorization (NMF)) can be used for this purpose. NSD accelerated IsoRank is summarized in Algorithm 1.

3.3. Auction-Based Bipartite Weighted Matching

Starting from a similarity matrix, efficiently identifying vertex correspondences requires a high quality and fast bipartite graph matching procedure. This graph matching algorithm views the similarity matrix as a bipartite graph, and consequently in the rest of the paper, we use the terms graph matching and bipartite graph matching interchangeably. More specifically, an n_A-by-n_B similarity matrix X, where $n_A \leq n_B$, can be transformed into a bipartite graph $G = (V_A, V_B, E)$, where $E \subseteq V_A \times V_B$. Each row i represents a vertex in V_A, and each column j a vertex in V_B. A nonzero entry x_{ij} in the matrix represents a weight of the edge $(i, j) \in E$. A subset $M \subseteq E$ in a bipartite graph is called a matching if no pair of edges of M are incident to the same vertex.

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

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

Auction algorithms find the maximum weighted matching via an *auction*: V_A and V_B represent the set of buyers and objects, respectively. A weighted edge x_{ij} is the benefit that buyer i obtains by acquiring object j. The auction-based algorithm (see Algorithm 2) consists of three phases: the *initialization phase* (lines 1–4), the *bidding phase* (lines 6–9), and the *assignment phase* (lines 10–11). Each object j has an associated price p_j, which is initially set to zero. In an auction iteration, the bidding and assignment phase, and the update of the price and of the increment ε are performed until every buyer is assigned to an object. We will discuss the initialization and update of the crucial term ε (lines 4, 12) in the next Section 4.

3.4. Quality Measures for Matching

Given two graphs G_A and G_B, the quality of the computed matchings m_i, m_j is computed from the *alignment graph*: If $m_i = (v_i^A, v_i^B)$ and $m_j = (v_j^A, v_j^B)$ in $G_A \times G_B$ are two matches, then $(m_i, m_j) \in E_{A \times B} \Leftrightarrow (v_i^A, v_j^A) \in E_A$ and $(v_i^B, v_j^B) \in E_B$.

When analyzing the alignment graph of two networks, a measure for the topological evaluation of the computed matching is the number of *conserved edges* across the two networks. This corresponds to the number of edges in the alignment graph. Each conserved edge implies matching of the corresponding edges connecting the elements of the endpoints in the input networks. Consequently, vertex matching naturally follows from edge matching and vice-versa. An alternate measure called *similarity rate* is defined as the ratio of conserved edges over the minimum of the edges in the two networks. For a more comprehensive discussion of qualitative assessment of graph matching, we refer readers to [30].

Algorithm 2 Sequential Auction Algorithm for Maximum Weighted Matching

```
Input: Bipartite graph $G = (V_A, V_B, E, w)$
Output: Matching $M$
 1: $M \leftarrow \emptyset$                                   ▷ current matching
 2: $I \leftarrow \{i : 1 \leq i \leq n_A\}$                  ▷ set of unassigned buyers
 3: $p_j \leftarrow 0$ for $j = 1, \ldots, n_B$               ▷ initialize prices for objects
 4: initialize($\varepsilon$)                                  ▷ initialize $\varepsilon$
 5: while $I \neq \emptyset$ do                                ▷ auction iteration
 6:     $j_i \leftarrow \arg\max_j \{x_{ij} - p_j\}$          ▷ find best object of buyer $i$
 7:     $u_i \leftarrow x_{ij_i} - p_{j_i}$                   ▷ store profit of the most valuable object
 8:     $v_i \leftarrow \max_{j \neq j_i} \{x_{ij} - p_j\}$   ▷ store second-best profit
 9:     $p_{j_i} \leftarrow p_{j_i} + u_i - v_i + \varepsilon$ ▷ update price with the bid $u_i - v_i$ and $\varepsilon$
10:     $M \leftarrow M \cup \{i, j_i\}$; $I \leftarrow I \setminus \{i\}$   ▷ assign buyer to the desired object
11:     $M \leftarrow M \setminus \{k, j_i\}$; $I \leftarrow I \cup \{k\}$   ▷ free previous owner $k$ if available
12:     update($\varepsilon$)                                  ▷ increment/decrement $\varepsilon$
13: end while
```

4. Building an Integrated Parallel Graph Matching Formulation

Using the NSD-accelerated similarity construction and the auction-based bipartite matching as our serial bases, we propose highly efficient and scalable parallel formulations. Specifically, we show that both phases of the alignment process lend themselves naturally to parallel implementations and that the output from the first phase flows naturally into the second phase without introducing significant copying overheads.

4.1. Parallelizing NSD

NSD-based similarity matrix construction consists of two parts:

- Computing iterates of $\tilde{A}$ and $\tilde{B}$ applied over each of the corresponding $z_i^{(0)}$ and $w_i^{(0)}$ vectors (Algorithm 1, lines 3–8).
- Computing outer products of the iterates and sum (Algorithm 1, lines 9–13, 15).

In the rest of this section we describe two possible approaches to NSD parallelization. The first approach is generic, not customized for integration with a subsequent matching extraction stage. It has been used in preliminary standalone experiments of NSD parallelization over heterogeneous platform testbeds. More specifically, in Algorithm 3 the iterates are computed by the root process and are consequently partitioned and distributed to a $p \times q$ process grid (lines 2–14). Outer products are then independently calculated, and the final, naturally distributed, similarity matrix is synthesized by worker processes (lines 15–23).

This approach, in general, induces a 2-D block decomposition of the resulting matrix. However by setting $p = 1$ ($q = 1$) it reduces to a 1-D row-wise (column-wise) formulation. In this scenario, choosing to parallelize only the second part

Algorithm 3 Parallel NSD (generic)

```
1: Root (lines 2–14) and $(r,u)$ worker process in the $p \times q$ grid (lines 15–23).
2: compute $\tilde{A}$, $\tilde{B}$
3: for $i = 1$ to $s$ do
4:     $w_i^{(0)} \leftarrow w_i$, $z_i^{(0)} \leftarrow z_i$
5:     for $k = 0$ to $n$ do
6:         $w_i^{(k)} \leftarrow \tilde{B} w_i^{(k-1)}$
7:         $z_i^{(k)} \leftarrow \tilde{A} z_i^{(k-1)}$
8:     end for
9: end for
10: for $i = 1, \ldots s, k = 0, \ldots ,n$ do
11:     Partition $w_i^{(k)}$ in $p$ fragments, $w_{i,1}^{(k)}, \ldots ,w_{i,p}^{(k)}$
12:     Partition $z_i^{(k)}$ in $q$ fragments, $z_{i,1}^{(k)}, \ldots ,z_{i,q}^{(k)}$
13: end for
14: Send to every process $(r,u)$ in the process grid $p \times q$ its corresponding $w_{i,r}^{(k)}$,
    $z_{i,u}^{(k)}$ fragments, $\forall i = 1, \ldots s, k = 0, \ldots ,n$ $(r = 1, \ldots ,p,\ u = 1, \ldots ,q)$
15: Receive corresponding $w_{i,r}^{(k)}$, $z_{i,u}^{(k)}$ fragments, $\forall i = 1, \ldots s, k = 0, \ldots ,n$ from
    the root process
16: for $i = 1$ to $s$ do
17:     zero $X_{i,ru}^{(n)}$
18:     for $k = 0$ to $n-1$ do
19:         $X_{i,ru}^{(n)} \leftarrow X_{i,ru}^{(n)} + \alpha^k w_{i,r}^{(k)} {z_{i,u}^{(k)}}^T$
20:     end for
21:     $X_{i,ru}^{(n)} \leftarrow (1-\alpha) X_{i,ru}^{(n)} + \alpha^n w_{i,r}^{(n)} {z_{i,u}^{(n)}}^T$
22: end for
23: $X_{ru}^{(n)} \leftarrow \sum_{i=1}^{s} X_{i,ru}^{(n)}$
```

of NSD can be justified on the grounds of its quadratic complexity (in the number of vertices) compared to the linear complexity (in the number of edges) of the first part.

The second approach is specifically targeted towards integration with the parallel auction algorithm for matching, and is the one adopted for the large-scale experiments reported in Section 5 (Algorithm 6, lines 2–9). Auction-based algorithms introduce the metaphors of *buyers* and *objects*, respectively mapped to row and column indices of the similarity matrix. Consequently, 2-D block decompositions partition both the buyers and objects sets, thus resulting in excessive communication and synchronization costs. Consequently, we restrict ourselves to a partitioning of the "buyers" only. This translates to a 1-D distribution of row blocks. To further increase concurrency, $\tilde{B}$-generated vector iterates are computed using a parallel sparse matrix-vector multiplication kernel (Algorithm 6, line 8).

4.2. Parallel Auction-based Weighted Matching

Algorithm 2 corresponds primarily of the bidding and assignment phase. The bidding phase contains the bid computation of a free buyer, and the assignment phase includes the matching of the buyer to the object and the price update of the object. In a parallel version of the algorithm (see Algorithm 4), bids of free buyers can be simultaneously computed. Each free buyer computes a bid for the most-valuable object according to the current price of the object. The buyer with the highest bid for an object is determined and is assigned to the object. The prices of the objects are updated according to the highest bids. The parallel bidding phase starts again with the free buyers.

The parallel auction algorithm is based on a 1D row-wise distribution of the entire matrix. Each process procures a set of buyers and performs the auction iterations until locally free buyers are assigned in the global matching. The bid computation on each process can be further accelerated using existing shared memory parallelization strategies that differ in how the number of threads are involved in the bid calculation for a buyer. We map, block-wise, the number of available threads to unassigned buyers. The communication cost of the parallel auction algorithm corresponds to the exchange of local prices for the objects among the processes to determine the winner for the object. This communication cost can be reduced by exchanging only locally altered prices, and by bundling messages into a single message. Additionally, every process submits only the locally highest price for the objects. The auction algorithm also has excellent memory scalability. If the graph is distributed a-priori, a price vector $p \in \mathbb{R}^{n_B}$ is stored at each process.

4.2.1. ε-scaling

ε-scaling is an important aspect of auction-based bipartite matching described in Algorithms 2 and 4. Consider line 9 in the Algorithm 2. Here, a new price for an object is computed by adding the bid and a small increment ε to the old value of the price. To understand the importance of ε in the price update, assume that ε is set to zero. Furthermore, imagine that two buyers are bargaining for the same valuable object, while the best and second-best profits are of the same value. In this case, the updated price remains unchanged. In such a scenario, neither buyer will be satisfied with the current assignment, and the process ends in a *price war*, where a small number of buyers are competing for equally desirable objects. In order to ensure that the price for an object is raised after each iteration, a small increment ε is introduced.

For the parallel auction-based matching algorithm the following ε-scaling strategies for sparse and dense graphs are proposed. For sparse graphs $\varepsilon_{\text{local}}$ is initialized to $\varepsilon_{\text{local}} = \frac{n+1}{\theta}$, and decremented slightly by $\varepsilon_{\text{local}} = \max\{\epsilon, \varepsilon_{\text{local}} - \epsilon\}$, where $\theta = 16$ and $\epsilon = \frac{1}{n+1}$.

Unfortunately, in the dense case, using this choice of variables, the parallel algorithm often runs into a price war scenario, the number of iterations may dramatically rise, and the algorithm does not scale well. Consequently, an alternate scaling strategy is used. The value of $\varepsilon_{\text{local}}$ is initialized to a small

Algorithm 4 Parallel Auction Algorithm for Weighted Matchings

```
Input: Bipartite graph $G = (V_A, V_B, E, w)$
Output: Matching $M$
1:  $M_{\text{local}} \leftarrow \emptyset$                                  ▷ set of locally matched buyers
2:  $I_{\text{local}} \leftarrow \{i : 1 \leq i \leq \frac{n_A}{P}\}$         ▷ reindexing set of locally free buyers
3:  $I_{\text{global}} \leftarrow \text{allgather}(I_{\text{local}})$          ▷ globally free buyers
4:  $p_j \leftarrow 0$ for $j = 1, \ldots, n_B$                             ▷ global price vector for the objects
5:  initialize($\varepsilon_{\text{local}}$)
6:  while $I_{\text{global}} \neq \emptyset$ do
7:      $j_i \leftarrow \arg\max_j \{w_{ij} - p_j\}$                        ▷ computation phase via threading
8:      $u_i \leftarrow w_{ij_i} - p_{j_i}$
9:      $v_i \leftarrow \max_{j \neq j_i} \{w_{ij} - p_j\}$
10:     $p_{j_i} \leftarrow p_{j_i} + u_i - v_i + \varepsilon_{\text{local}}$   ▷ update prices with bid $u_i - v_i$ and $\varepsilon_{local}$
11:     $M_{\text{local}} \leftarrow M_{\text{local}} \cup \{i, j_i\}$        ▷ locally assign buyer $i$ to desired object
12:     gather_changed_prices($p_{j_i}$)                                   ▷ communication phase
13:     check_winner($j_i$)                                                ▷ if overbidded update local price
14:     $I_{\text{local}} \leftarrow I_{\text{local}} \setminus \{i\}$ or $M_{\text{local}} \leftarrow M_{\text{local}} \setminus \{i, j_i\}$   ▷ update sets
15:     $I_{\text{global}} \leftarrow \text{allgather}(I_{\text{local}})$     ▷ update global free buyers
16:     update($\varepsilon_{\text{local}}$)
17: end while
```

value and adaptively increased relatively to the overall progress (see Algorithm 5). The basic idea behind this heuristic is that in the inner iteration at least δ buyers get assigned to an object while $\varepsilon_{\text{local}}$ converges faster to a large value (line 6). In the outer loop (line 5), $\varepsilon_{\text{local}}$ will be reset again to a small value if the threshold has been exceeded. Thus, the approximate variant forces the auction algorithm to match a buyer faster in the early stage of the algorithm. This aggressive ε-scaling strategy is embedded in the main routines in Algorithm 4. The algorithm delivers a maximal matching with maximum weight, but the quality of the match is adequate in the context of graph similarity. For a detailed discussion of these issues (not directly related to parallel processing issues, which form the focus of this paper), we refer the readers to [31]. The proposed heuristic terminates if every buyer is matched, or the prices for the objects are too expensive, so the bids for unassigned buyers are negative.

4.3. A Parallel Sparsification Strategy

While our similarity computation routines are capable of analyzing large graphs 10^6 vertices and beyond, they generate similarity matrices in outer product forms. To the best of our knowledge there are currently no matching algorithms that can be applied directly on such low-rank matrix representations. Therefore, it becomes imperative to explicitly compute the similarity matrix from these outer product forms. This task poses constraints in terms of storage requirements for the similarity matrix, which is quadratic in the number of vertices in the graphs. As an example, the similarity matrix for two graphs of

Algorithm 5 Adaptive Parallel Auction Algorithm

```
1: Perform the initialization phase of algorithm 4 (lines 1–4)
2: $\xi \leftarrow 2; \theta \leftarrow 16; \gamma \leftarrow \frac{n+1}{\theta}$
3: $\delta \leftarrow \left\lfloor \min\left\{\frac{|I_{\text{global}}|}{\xi}, \frac{n}{\theta}\right\}\right\rfloor$          ▷ initialize threshold $\delta$
4: while $I_{\text{global}} \neq \emptyset$ do
5:     $\varepsilon_{\text{local}} \leftarrow \frac{\theta}{n+1}$          ▷ reset $\varepsilon_{local}$ to small value
6:     while $|I_{\text{global}}| > \delta$ do
7:         Perform bidding and assignment phase of algorithm 4 (lines 7–15)
8:         if $\gamma > \varepsilon_{\text{local}}$ then
9:             $\varepsilon_{\text{local}} \leftarrow \varepsilon_{\text{local}} \cdot \xi$
10:        else
11:            $\varepsilon_{\text{local}} \leftarrow \gamma$
12:        end if
13:        $\gamma \leftarrow \gamma/\xi$
14:    end while
15:    $\delta \leftarrow \delta/\xi$; $\theta \leftarrow \theta \cdot \xi$          ▷ update $\delta$ and $\theta$
16: end while
```

k	**#conserved edges**
5 (0.07%)	784 (53.88%)
10 (0.13%)	913 (62.75%)
100 (1.33%)	1263 (86.80%)
200 (2.66%)	1317 (90.52%)
500 (6.65%)	1413 (97.11%)
1000 (13.30%)	1442 (99.11%)

Table 1: For various k we compute the number of conserved edges resulting from a sparsified similarity matrix instance (for two PPI networks). Percentages are computed based on the fact that the number of columns is 7518 (k column) and the number of conserved edges from the *dense* similarity matrix is 1455.

10^6 vertices each, is a dense matrix of 10^{12} entries. This requires a distributed memory of in the order of a few terabytes, simply for storing the similarity scores.

To address this storage requirement, we propose a sparsification scheme that is integrated into the assembly process for the similarity matrix from the outer products. In addition to reducing storage, while not significantly impacting the match quality, the result of the sparsification scheme must be in a form that can be directly used by the parallel matching algorithm (i.e., in a row-wise block partitioned form). We use the following strategy:

- Use the j^{th} element of each of the $w_{i,r}^{(n)}$ vectors (in lines 19 and 21 of Algorithm 3 ; $q = 1$) to scale the ${z_i^{(k)}}^T$ vectors consecutively for all local

Algorithm 6 NSD-based Parallel Graph Matching

```
1: □ = root process, no labels = all processes r
2: □ load adjacency matrices A, B and component vectors $w_i$, $z_i$;
3: □ compute $\tilde{A}$, $\tilde{B}$;
4: broadcast $\tilde{A}$, $w_i$, $z_i$;
5: distribute $\tilde{B}$ by row blocks          ▷ each process r gets its $\tilde{B}_r$ part;
6: for all components $i$ and steps $k$ ($z_i^{(0)} = z_i$, $w_i^{(0)} = w_i$)
7: compute vector iterates $z_i^{(k)} \leftarrow \tilde{A} z_i^{(k-1)}$
8: compute vector iterates $w_{i,r}^{(k)} \leftarrow \tilde{B}_r w_i^{(k-1)}$, gather $w_i^{(k)}$ (// matvec);
9: compute row-wise the local similarity matrix $X_r$ (embarrassingly //)
10:         ▷ NSD-based, sparsify if needed (sort row entries, keep largest ones);
11: compute weighted matchings by // auction
12:                                      ▷ matching permutation lands on root;
13: □ compute number of conserved edges, similarity rate;
```

row indices j.

- Once a row of the similarity matrix is constructed, retain only the k largest values in the row (with $k << n$) before advancing to the next row.

This sparsification procedure decreases the storage requirement associated with the similarity matrix by a factor $\frac{k}{n}$. It can be adaptively tuned as a trade-off between available memory and input network sizes. Our intention here is to provide a practical and intuitive aproximation strategy, rather than a formally quantified pruning solution. We empirically note that this strategy works well for our test cases. Table 1 demonstrates that the proposed sparsification preserves, to a large extent, the "quality" of the similarity matrix output: with 5% of similarity matrix entries we can match almost 95% of the conserved edges for two PPI networks.

We collect all stages in our workflow (parallel similarity matrix construction, sparsification, parallel auction-based matching, computation of quality indices) into an integrated procedure for parallel graph matching. This is described in Algorithm 6, the basis for the implementation running on a large, supercomputer-class cluster. Note that a subset of sparse matrix-vector products is also parallelized. Some of the steps in this skeleton algorithm have already been described as parts of Algorithm 3 (computation of the local similarity matrix, specifically for a $p \times 1$ process grid), in Subsection 4.3 (sparsification), in Algorithm 4 (parallel matching) and in Subsection 3.4 (quality measures).

4.4. Complexity of the Integrated Approach

The sparsification procedure requires *sorting* (per row), and this introduces an extra average complexity term of $O(n^2 \log n)$, for networks of size n. This is in addition to the standard $O(n^2)$ complexity of matrix similarity construction (per component, without sparsification). The sorting procedure can be

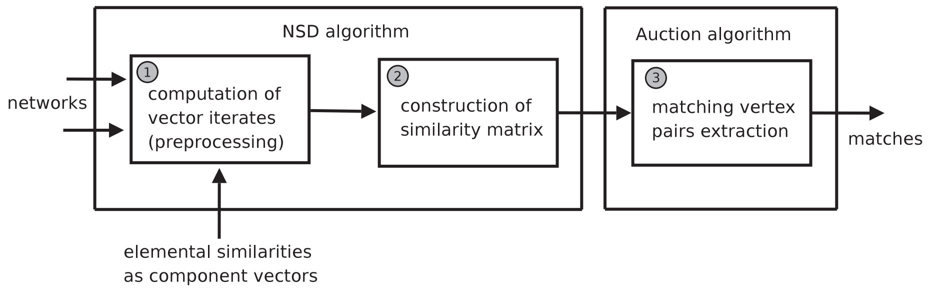

NSD algorithm
Auction algorithm
networks
1
computation of vector iterates (preprocessing)
2
construction of similarity matrix
3
matching vertex pairs extraction
matches
elemental similarities as component vectors

Figure 3: The NSD-based graph matching pipeline: NSD outputs a similarity matrix and the auction matching algorithm generates pairs of vertices from the two networks that match; indices can characterize the quality of these matches and can be computed at the right end of the pipeline.

the dominant part of the computation for a small number of components (e.g. $s = 1$). However, this cost is amortized for larger values of s. Furthermore, other hash-based approaches can be used to approximate these ranges. Note also that auction matching stage, that follows this stem in the integrated pipeline of Figure 3, has a worst case complexity of $O(nm \log(nC))$; n and m are respectively the size and the number of nonzero values of the (sparsified) similarity matrix and $C = \max_{ij} |x_{ij}|$.

5. Experimental Results

We provide results from a detailed set of experiments to quantify the performance of our methods on large-scale parallel platforms for diverse sets of input networks. Results are presented for two variants of the method: with and without sparsification of the similarity matrix. Performance results are complemented by quality measures, computed as conserved edges from matching results.

5.1. Experimental Environment and Setup

The code is implemented in C using a "hybrid" parallel programming model (MPI and OpenMP). This model efficiently utilizes both shared address space models supported by multiple cores and messaging across nodes.

In all cases $\alpha = 0.8$ (recall that α is the fraction of the similarity score that comes from topological similarity; the rest comes from elemental similarity), the number of iterations is fixed. Also $s = 10$ randomly generated components were input in all runs; this choice reflects the fact that no specific, a-priori matching preferences are available in general. Uniform H scores encode the base truth that, initially, any vertex in one graph could match any vertex in the other with equal likelihood.

Pair	Graph	#Vertices	#Edges
protein-protein	yeast	5,499	31,898
	fruitfly	7,518	25,830
net/pfinan	net4-1	88,343	1,265,035
	pfinan512	74,752	335,872
snapA	soc-slashdot090221	82,144	549,202
	soc-slashdot090216	81,871	545,671
snapB	soc-slashdot0902	82,168	948,464
	soc-slashdot0811	77,360	905,468
usroads	usroads	129,164	165,435
	usroads-48	126,146	161,950
dnvs	halfb	224,617	6,306,219
	fullb	199,187	5,953,632
b3	m133-b3	200,200	800,800
	shar_te2-b3	200,200	800,800
coAuthors	coAuthorsDBLP	299,067	977,676
	coAuthorsCiteseer	227,320	814,134
notreDame	NotreDame_www	325,729	929,849
	web-NotreDame	325,729	1,497,134
stanford	Stanford	281,903	2,312,497
	web-Stanford	281,903	2,312,497

(a)

Pair	Graph	#Vertices	#Edges
amazon	amazon0505	410,236	3,356,824
	amazon0601	403,394	3,387,388
delaunay	delaunay_n19	524,288	1,572,823
	delaunay_n18	262,144	786,396
authorsSelf	coAuthorsCiteseer	227,320	814,134
	coAuthorsCiteseer	227,320	814,134
coPapers	coPapersDBLP	540,486	15,245,729
	coPapersCiteseer	434,102	16,036,720
papersSelf	coPapersCiteseer	434,102	16,036,720
	coPapersCiteseer	434,102	16,036,720
dbpedia1	dbpedia-3.0_300k	300,000	1,320,138
	dbpedia-3.5.1_500k	500,000	10,546,881
eu/in	eu-2005_300k	300,000	10,835,193
	in-2004_500k	500,000	8,506,508
dbpedia2	dbpedia-3.0_500k	500,000	2,680,807
	dbpedia-3.5.1_1500k	1,500,000	26,794,451
euSelf	eu-2005	862,664	19,235,140
	eu-2005	862,664	19,235,140

(b)

Table 2: Characteristics of networks (organized in pairs) used in experiments. Note the extra spacings defining the 7 graph pair sets.

Pair	Total time (s)	#Cores
protein-protein	75	1
net/pfinan	796	48
snapA	2,688	48
snapB	1,497	48
usroads	281	384
dnvs	880	384
b3	1,593	384
coAuthors	659	768
notreDame	764	768
stanford	615	768

(a)

Pair	Total time (s)	#Cores
amazon	558	3,072
delaunay	938	3,072
authorsSelf	226	3,072
coPapers	2,167	3,072
papersSelf	1,630	3,072
dbpedia1	17,382	128
eu/in	18,122	128
dbpedia2	16,838	256
euSelf	10,939	256

(b)

Table 3: Networks (organized in pairs) used in experiments, together with base timings recorded at corresponding compute core counts. Note the extra spacings defining the 7 graph pair sets.

Our experiments are performed on the Cray XE6 at the Swiss National Supercomputing Centre in Manno, Switzerland. The Cray XE6 has 176 dual-socket compute nodes, each socket is a 12-core AMD Opteron (aka Magny-Cours), connected through a Gemini communication interface. We map each

Pair	eu/in				dbpedia1			
Cores	**128**	**256**	**512**	**1024**	**128**	**256**	**512**	**1024**
t_generateIterates	5.07	5.04	5.33	6.13	11.00	11.19	11.53	12.36
t_generateRow	16,451	8,152	4,030	1,225	15,704	7,475	3,254	1,229
t_sort	1,578	788	395	197	1,606	802	401	201
t_similarityMatrix	18,046	8,949	4,429	1,424	17,327	8,287	3,660	1,431
t_parallelAuction	55.16	28.82	16.32	11.90	31.97	19.78	14.37	14.93
t_total	18,122	8,999	4,467	1,458	17,382	8,329	3,697	1,471

Table 4: Timing results (in secs) from various phases of the similarity analysis process for the eu/in and dbpedia1 datasets. With reference to Algorithm 6, t_generateIterates corresponds to lines 7-8, t_generateRow and t_sort are subparts of t_similarityMatrix (lines 9-10), t_parallelAuction corresponds to lines 11-12 and t_total is the total time elapsed.

MPI process to a socket, fix the number of OMP threads either to 8 or 12, and test scalability for up to 256 MPI processes, resulting to 3,072 compute cores, at maximum. The PathScale programming environment (version 3.1.61) is used with its accompanying compiler.

As dataset a diverse set of networks (see Table 2) available in the form of their adjacency matrices are taken from the University of Florida sparse matrix collection [13], the Wikipedia datasets containing its inter-article link structure [6], and well-known protein-protein interaction networks [59]. We also report about timings and number of cores to run the full integrated pipeline on diverse adjacency pairs (see Table 3). These are the baseline computations for the speed improvement plots.

5.2. Results with Sparsification

In this set of experiments we construct a sparsified version of the similarity matrix and sparsification can be driven by two different objectives as we increase the number of cores. In the first approach, the total number of nonzero entries of the *global* (sparsified) similarity matrix is kept constant. This can be enforced by using a constant value for k (the number of values per row we keep). It follows that the number of nonzero entries for the local part of the resulting similarity matrix will decrease for larger configurations, since the number of rows locally assigned is also reduced in this case. This corresponds to the *strong scaling case for auction matching*. In this case, near-linear speedup is observed for all compute-intensive intermediate steps for up to 1,024 cores. We note that our algorithm can extract matching pairs for half-a-million vertex networks in less than 30 minutes using our cluster (see Table 4).

In the second approach, the total number of nonzero entries in the *local part* of the similarity matrix is kept constant. We implement this in our code by adaptively increasing k with the number of cores (that also implies a decrease in the number of rows locally assigned), so as to utilize the full 16 GB memory

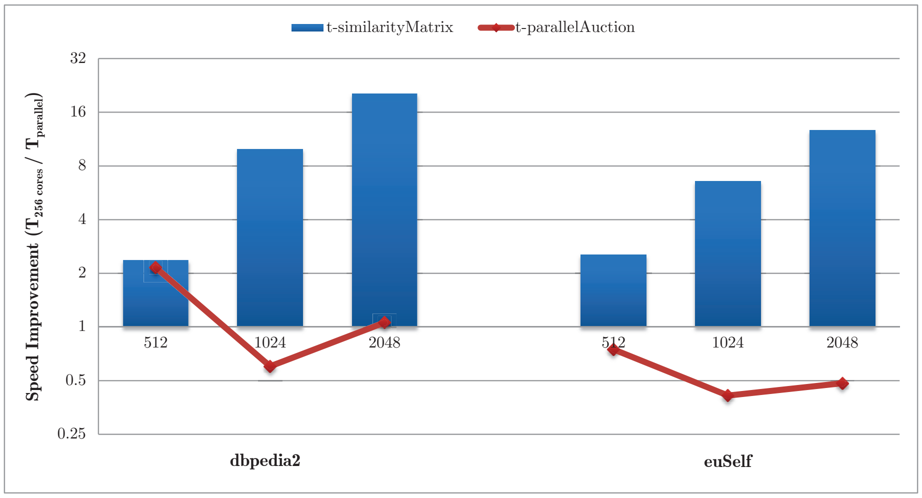


Figure 4: Speed improvement of the similarity matrix construction in the strong scaling sense, and the parallel auction in the weak scaling sense by using up to 2,048 compute cores.

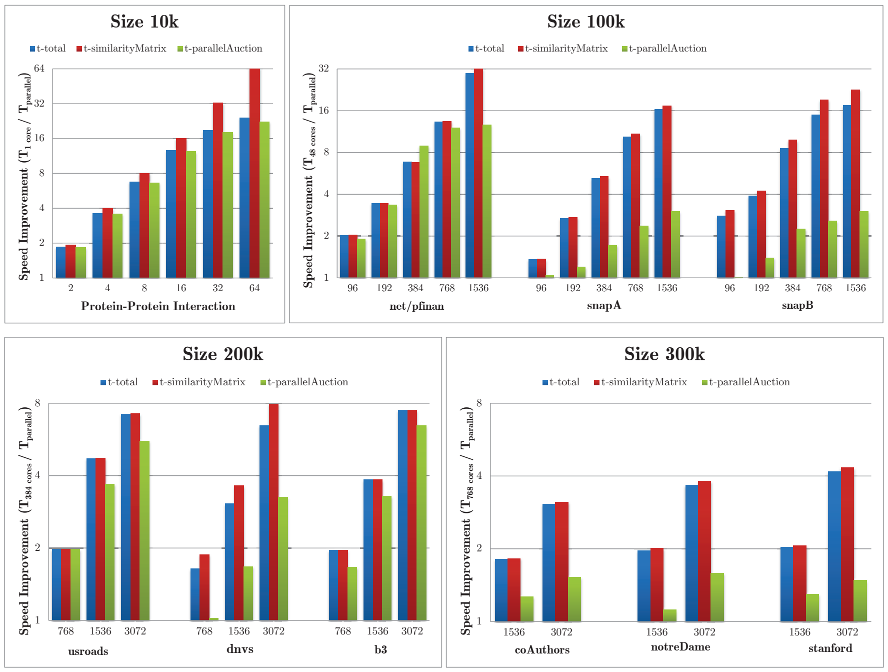
available for each socket of Cray XE6. This is the *weak scaling case for auction matching*. Note that in this scenario, the auction matching algorithm is effectively applied to different (successively denser) similarity matrices as the number of cores is increased, and this impacts the matching pairs returned. Weak scaling experiments for auction matching and strong scaling for similarity matrix construction are illustrated in Figure 4 using up to 2,048 cores. We can process pairs of million-vertex networks and extract similar vertex pairs in a couple of hours on such configurations.

5.3. Results without Sparsification

The scalability of our approach is also demonstrated for datasets and configurations without using sparsification. In Figure 5 near linear speedup is reported for parallel similarity matrix construction and also for the overall time. This follows from the fact that parallel auction matching scales reasonably well and takes only a small fraction of the overall time. Particularly, for protein-protein interaction networks, a dramatic time reduction for the full pipeline is gained from parallelization: extracting matching pairs for two typical networks using 64 cores takes about three seconds. Using sequential state-of-the-art approaches like IsoRank [59] these matching requires about 1.5 hours for a solution of comparable quality. Table 5 presents timing results for the largest instances tested. Similarity computation requires 256 sockets (3072 cores) to store the entire data.

5.4. Quality Measurement

Up to 3,072 cores are used for matching up to approximately 500k-vertex networks. We report on the similarity rate, that is a “normalized” version of



Pair	amazon	delaunay	coPapers	papersSelf	authorsSelf
t_similarityMatrix	481.73	935.04	2,156.20	1,620.30	222.56
t_parallelAuction	76.18	2.61	10.37	9.47	3.01
t_total	557.91	937.65	2,166.57	1,629.77	225.57

Figure 5: Speed improvement and timing results (in secs) from the major steps in our integrated approach for variable-sized dataset using up to 3,072 compute cores.

the number of conserved edges. Our intention here is to assess the robustness of our approach for the case of self-similarity (matching a graph with itself): since no approximation is introduced (e.g. by sparsification) it is expected to obtain a number of conserved edges equal to the number of edges in the graph in the optimal case. This is indeed the case for authorsSelf and papersSelf pairs (Table 5).

6. Conclusions and Future Work

We address the problem of matching similar vertices of graph pairs in parallel. Our approach consists of two basic components: *parallel NSD*, a highly efficient and scalable parallel formulation based on a recently introduced serial algorithm for similarity matrix computation and *parallel auction-based bipartite matching*. We validate the performance of our integrated pipeline on a large,

Pair	#CE	Rate
protein-protein	745	0.03
net/pfinan	74,778	0.22
snapA	14,296	0.02
snapB	77,617	0.09
usroads	2,666	0.02
dnvs	1,750,799	0.29
b3	29,217	0.15
coAuthors	85,437	0.11
notreDame	113,992	0.12
stanford	107,968	0.05

(a)

Pair	#CE	Rate
amazon	46,278	0.01
delaunay	112,152	0.14
authorsSelf	814,134	1.00
coPapers	3,520,545	0.23
papersSelf	16,036,720	1.00
dbpedia1	1,100	0.004
eu/in	80,884	0.04
dbpedia2	2,082	0.007
euSelf	219,759	0.26

(b)

Table 5: Quality measurement indices from experiments with various network pairs: number of conserved edges (CE) and the similarity rate (rate). The extra spacings define the 7 graph pair sets (in the sense of the caption in Table 2.

supercomputer-class cluster and diverse graph instances. We provide experimental results demonstrating that our algorithms scale to large machine configurations and problem instances. In particular, we show that our integrated pipeline enables alignment of networks of sizes two orders of magnitude larger than currently possible (millions of vertices, tens of millions of edges).

As part of future work, we investigate the feasibility of a bipartite matching algorithm accepting as input the vectors of low-rank approximations of the similarity matrix, rather than the fully assembled similarity matrix. We will explore the possibility of substituting a formal pruning strategy for the optional sparsification stage.

References

[1] R. Allen, L. Cinque, S. Tanimoto, L. Shapiro, and D. Yasuda. A parallel algorithm for graph matching and its MasPar implementation. *Parallel and Distributed Systems, IEEE Transactions on*, 8(5):490–501, 1997.

[2] S. Bandyopadhyay, R. Sharan, and T. Ideker. Systematic identification of functional orthologs based on protein network comparison. *Genome research*, 16(3):428–435, 2006.

[3] M. Bayati, M. Gerritsen, D.F. Gleich, A. Saberi, and Y. Wang. Algorithms for large, sparse network alignment problems. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 705–710. IEEE, 2009.

[4] D. P. Bertsekas and D. A. Castañon. Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing*, 17(707–732), 1991.

[5] D. P. Bertsekas and D. A. Castañon. A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications*, 1:277–297, 1993.

[6] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - A crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2009.

[7] V.D. 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.*, 46(4):647–666, 2004.

[8] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[9] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[10] L. Buš and P. Tvrdik. Towards auction algorithms for large dense assignment problems. *Computational Optimization and Application*, 43(3):411–436, 2009.

[11] Ü. V. Çatalyürek, F. Dobrian, A. H. Gebremedhin, M. Halappanavar, and A. Pothen. Distributed-memory parallel algorithms for matching and coloring. In *2011 International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Workshop on Parallel Computing and Optimization (PCO'11)*, pages 1966–1975. IEEE Press, 2011.

[12] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[13] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011.

[14] D. E. Drake and S. Hougardy. Linear time local improvements for weighted matchings in graphs. In *WEA'03 Proceedings of the 2nd international conference on Experimental and efficient algorithms*, pages 107–119, 2003.

[15] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22:973–996, 1999.

[16] M. El-Kebir, J. Heringa, and G. Klau. Lagrangian relaxation applied to sparse global network alignment. *Pattern Recognition in Bioinformatics*, pages 225–236, 2011.

[17] J. Flannick, A. Novak, C.B. Do, B.S. Srinivasan, and S. Batzoglou. Automatic parameter learning for multiple network alignment. In *Proceedings of the 12th annual international conference on Research in computational molecular biology*, pages 214–231. Springer-Verlag, 2008.

[18] J. Flannick, A. Novak, B.S. Srinivasan, H.H. McAdams, and S. Batzoglou. Graemlin: general and robust alignment of multiple large interaction networks. *Genome research*, 16(9):1169–1181, 2006.

[19] L. Getoor and C.P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[20] D. Gitchell and N. Tran. Sim: a utility for detecting similarity in computer programs. In *ACM SIGCSE Bulletin*, volume 31, pages 266–270. ACM, 1999.

[21] D. Gregor and A. Lumsdaine. The Parallel BGL: A generic library for distributed graph computations. In *In Parallel Object-Oriented Scientific Computing (POOSC*. Citeseer, 2005.

[22] Guoming He, Haijun Feng, Cuiping Li, and Hong Chen. Parallel SimRank computation on large graphs with iterative aggregation. In Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang, editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 543–552. ACM, 2010.

[23] S. Hougardy and D. E. Vinkemeier. Approximating weighted matchings in parallel. *Information Processing Letters*, 99(3):119–123, 2006.

[24] 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*, pages 538–543. ACM, 2002.

[25] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[26] M. Kalaev, M. Smoot, T. Ideker, and R. Sharan. NetworkBLAST: comparative analysis of protein networks. *Bioinformatics*, 24(4):594–596, 2008.

[27] B.P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B.R. Stockwell, and T. Ideker. PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Research*, 32(suppl 2):W83, 2004.

[28] G.W. Klau. A new graph-based method for pairwise global network alignment. *BMC bioinformatics*, 10(Suppl 1):S59, 2009.

[29] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[30] G. Kollias, S. Mohammadi, and A. Grama. Network Similarity Decomposition (NSD): A fast and scalable approach to network alignment. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints):1–13, 2011.

[31] G. Kollias, M. Sathe, S. Mohammadi, and A. Grama. A fast approach for the global alignment of protein-protein interaction networks. 2012 (submitted).

[32] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama. Pairwise alignment of protein interaction networks. *Journal of Computational Biology*, 13(2):182–199, 2006.

[33] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface*, 7(50):1341–1354, 2010.

[34] Oleksii Kuchaiev and Natasa Przulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics*, 27(10):1390–1396, 2011.

[35] H. W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[36] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320. IEEE Computer Society, 2001.

[37] Z. Li, S. Zhang, Y. Wang, X.S. Zhang, and L. Chen. Alignment of molecular networks by integer quadratic programming. *Bioinformatics*, 23(13):1631–1639, 2007.

[38] F. Manne and R. H. Bisseling. A parallel approximation algorithm for the weighted maximum matching problem. In *Proceedings Seventh International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*, volume 4967 of *Lecture Notes in Computer Science*, pages 708–717, 2008.

[39] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002.

[40] V. Memišević and N. Pržulj. C-GRAAL: Common-neighbors-based global GRAph ALignment of biological networks. *Integr. Biol.*, 2012.

[41] T. Milenković, W.L. Ng, W. Hayes, and N. Pržulj. Optimal network alignment with graphlet degree vectors. *Cancer informatics*, 9:121, 2010.

[42] T. Milenković and N. Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer Informatics*, 6:257–273, 2008. PMID: 19259413.

[43] S. Mohammadi and A. Grama. Biological network alignment. *Functional Coherence of Molecular Networks in Bioinformatics*, pages 97–136, 2012.

[44] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[45] M.E.J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[46] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford University, 1998.

[47] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[48] P. M. Pardalos, J. Rappe, and M.G.C. Resende. An exact parallel algorithm for the maximum clique problem. In *In High Performance and Software in Nonlinear Optimization*, pages 279–300. Kluwer Academic Publishers, 1998.

[49] M. A. Patwary, R. H. Bisseling, and F. Manne. Parallel greedy graph matching using an edge partitioning approach. In *Proceedings of the Fourth ACM SIGPLAN Workshop on High-level Parallel Programming and Applications (HLPP 2010)*, pages 45–54, 2010.

[50] S. Pettie and P. Sanders. A simpler linear time $2/3$-ϵ approximation for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.

[51] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS 99*, pages 259–269. Springer, 1999.

[52] N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177 –e183, January 2007.

[53] J. Riedy. *Making static pivoting scalable and dependable*. PhD thesis, EECS Department, University of California, Berkeley, 2010.

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

[55] M. Sathe, O. Schenk, and H. Burkhart. An auction-based weighted matching implementation on massively parallel architectures. 2011 (submitted).

[56] R. Sharan, S. Suthram, R.M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R.M. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *Proceedings of the National Academy of Sciences of the United States of America*, 102(6):1974, 2005.

[57] Y. Shinano, T. Fujie, Y. Ikebe, and R. Hirabayashi. Solving the maximum clique problem using PUBB. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 326–332, 1998.

[58] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Min. Knowl. Discov.*, 2(1):39–68, 1998.

[59] 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*, 105(35):12763–12768, 2008.

[60] E.H. Sussenguth Jr. A graph-theoretic algorithm for matching chemical structures. *Journal of Chemical Documentation*, 5(1):36–43, 1965.

[61] M. Takashima, A. Ikeuchi, S. Kojima, T. Tanaka, T. Saitou, and J. Sakata. A circuit comparison system with rule-based functional isomorphism checking. In *Design Automation Conference, 1988. Proceedings., 25th ACM/IEEE*, pages 512–516. IEEE, 1988.

[62] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[63] Laura A. Zager and George C. Verghese. Graph similarity scoring and matching. *Appl. Math. Lett*, 21(1):86–94, 2008.