J. Parallel Distrib. Comput. 74 (2014) 2400–2410

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

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, Universita della Svizzera italiana, Switzerland

HIGHLIGHTS

- Global graph alignment on supercomputer-class clusters.
- Experimentally we demonstrate the scalability of our algorithm.
- Alignment of networks of sizes two orders of magnitude larger than currently possible.
- Serial NSD description heavily revised; introduction condensed or otherwise updated.
- New section comparing our method to a recent multithreaded approach.

ARTICLE INFO

Article history:
Received 24 October 2012
Received in revised form
19 August 2013
Accepted 25 December 2013
Available online 6 January 2014

Keywords:
Graph alignment
Vertex similarity
Parallel matching
Auction algorithm

ABSTRACT

This paper addresses the problem of global graph alignment on supercomputer-class clusters. We define the alignment of two graphs, 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] Using a state of the art serial algorithm for the computation of vertex similarity scores called Network Similarity Decomposition (NSD), we derive corresponding parallel formulations. Coupling this parallel similarity algorithm with a parallel auction-based bipartite matching technique, we obtain a highly efficient and scalable graph matching pipeline. We validate the performance of our integrated approach on a large parallel platform and on diverse graph instances (including Protein Interaction, Wikipedia and Web networks). Experimental results demonstrate that our algorithms scale to large machine configurations (thousands of cores) and problem instances, enabling the alignment of networks of sizes two orders of magnitude larger than reported in the current literature.

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' insights. 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 [9,7,35,16]. 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, efficient parallel formulations determine the feasibility envelope for such problems.

The graph alignment problem 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 problem takes the similarity matrix X and uses bipartite matching to map each vertex in the first graph to its most similar vertex in the second graph, to maximize the overall similarity across the graphs.

To illustrate the problem, we consider the graph in Fig. 1 and compute its similarity matrix using the IsoRank method (summarized later in the paper) for this graph aligned to itself

* Corresponding author.
E-mail addresses: gkollias@purdue.edu, giorgos@purdue.edu, gidiko.purdue@gmail.com (G. Kollias), madan.sathe@unibas.ch (M. Sathe), olaf.schenk@usi.ch (O. Schenk), ayg@cs.purdue.edu (A. Grama).

[1] We use the terms alignment and global graph alignment interchangeably; this is in contrast to local graph alignment, which permits a vertex to have different pairings in feasible local alignments, making it an inherently ambiguous process.

0743-7315/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.12.010

$$X = \begin{pmatrix} 0.000 & 0.924 & 0.371 & 0.286 & 1.394 & 1.312 & 0.304 & 1.029 & 0.432 & 0.383 \\ 0.924 & 0.000 & 0.686 & 0.396 & 3.93 & 3.622 & 0.454 & 2.773 & 0.874 & 0.715 \\ 0.371 & 0.686 & 0.000 & 0.301 & 1.004 & 0.975 & 0.320 & 0.714 & 0.458 & 0.416 \\ 0.286 & 0.396 & 0.301 & 0.000 & 0.515 & 0.491 & 0.268 & 0.408 & 0.314 & 0.297 \\ 1.394 & 3.930 & 1.004 & 0.515 & 0.000 & 5.924 & 0.621 & 4.455 & 1.343 & 1.041 \\ 1.312 & 3.622 & 0.975 & 0.491 & 5.924 & 0.000 & 0.586 & 4.096 & 1.272 & 1.034 \\ 0.304 & 0.454 & 0.320 & 0.268 & 0.621 & 0.586 & 0.000 & 0.471 & 0.339 & 0.317 \\ 1.029 & 2.773 & 0.714 & 0.408 & 4.455 & 4.096 & 0.471 & 0.000 & 0.930 & 0.750 \\ 0.432 & 0.874 & 0.458 & 0.314 & 1.343 & 1.272 & 0.339 & 0.930 & 0.000 & 0.455 \\ 0.383 & 0.715 & 0.416 & 0.297 & 1.041 & 1.034 & 0.317 & 0.750 & 0.455 & 0.000 \end{pmatrix}.$$

Box I.

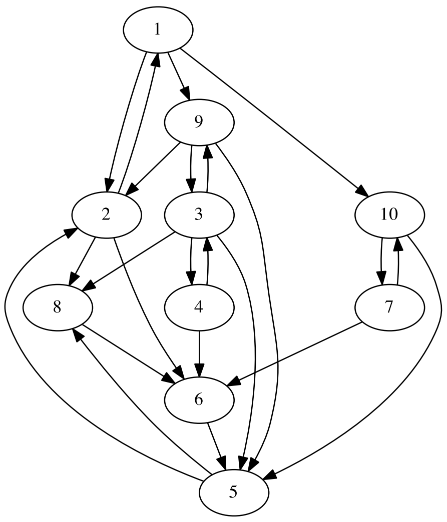


Fig. 1. The example graph used for illustrating the alignment process.

(self-similarity). A normalization step is applied to the similarity matrix after zeroing diagonal elements (to preclude the trivial solution of matching each node with itself). The resulting similarity matrix X (entries scaled by a factor of 100) is given in Box I.

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

This methodology has several important applications. In the analysis of biomolecular networks, nodes represent proteins and edges represent the 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 the traditional sequence-based similarity for proteins.

A number of other formulations and solutions exist in the literature to both the similarity computation and matching problems [47,6,27,41,4]. 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) [23]. 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 the prior work. Even with the 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 the 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, 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 in the context of 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 the 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 the 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. As a consequence, integrating a parallel bipartite weighted graph matching algorithm with a parallel NSD-accelerated similarity computation algorithm should result in a highly efficient pipeline. The main theme of this paper is the detailed presentation of this scheme and the experimental verification of its performance characteristics.

We highlight here related efforts on the two major components—algorithms for computing the similarity matrix X

illustrated above, and algorithms that compute pair-wise correspondences across the two graphs using the similarity matrix. 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. [37] present an excellent survey of approaches in this group.

In the second group of methods, the outcome of the computations is a matrix of elements 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 the 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 [20,24,45,28,48]. Global methods, on the other hand, consider a cost function computed over all vertex alignments [33,42,47].

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 [36,46] sense) in the product graph of the input networks. Of the PageRank-based methods for network similarity, the IsoRank algorithm [47] computes vertex similarity scores by integrating both vertex attributes and topological similarities. The graph kernel approach [43], 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 the topological similarity. In the GRAAL family of algorithms – GRAAL [25], H-GRAAL [32], MI-GRAAL [26], C-GRAAL [31] – 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 [14] integrates a priori known node (protein sequence) similarities and phylogenetic (evolutionary) relations. In NetAlignBP [2], a belief propagation approach is proposed, interestingly allowing the consideration of only a subset of potential pairs; Lagrangian, Markov random field and integer quadratic programming formulations have also been proposed [13,1,29,22]. SimRank is a generic method introduced in [18] for computing the structural-context similarity between vertices of a single graph.

An excellent survey of early serial results in this area is provided in [15]. Our parallel similarity construction relies on an acceleration scheme, called Network Similarity Decomposition (NSD) [23], 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, depending on the size of the networks. We provide an overview of NSD acceleration in Section 3 to motivate our parallel formulations.

There have been some recent efforts aimed at parallel graph alignment. GHOST [38] computes histograms of the spectra of the weighted Laplacians of the subgraphs induced around each vertex up to a number of shortest path lengths. These serve as vertex signatures that can drive matching at the next stage—implementing a seed-and-extend strategy which is sequential; however computing signatures is parallelizable. In its current version though, GHOST does not support traditional high-performance computing environments. In [21], two methods, Matching Relaxation (MR) and Belief Propagation (BP) are used to compute the similarity matrix, followed by 1/2-approximate matching, are parallelized for a shared memory platform. Section 5.3.1 presents a comparative study of our method with the work of Khan et al. [21].

We are not aware of any parallelizations that utilize the decoupled accelerations in NSD.

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 whose weight 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, the 1/2-approximation algorithm [41] with a linear-time implementation. Sophisticated approaches such as 2/3-or 3/4-approximation have also been proposed (please see e.g., [11,40]). Attempts to parallelize these methods have been reported in [8,17,30,39].

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 [12,19,27,34] – not amenable to massive parallelization – or auction-based matching algorithms [4].

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 multiple cores at each compute node [44]. 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 the 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 [23,44] for more details on these methods.

3.1. Terminology and preliminaries

We represent a directed 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 $i \rightarrow j$, 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 j of A and zero otherwise. 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.

3.2. Network Similarity Decomposition (NSD)

Singh et al. [47] propose IsoRank, a two-step approach to computing pair-wise similarity of vertices in two graphs. The first step in their method computes a similarity matrix X, of two graphs G_B and G_A with m and n vertices, respectively ($m \leq n$). Matrix X is computed through an iterative procedure that accrues similarities diffused over successively larger neighborhoods. The resulting matrix $X_{m\times n}$ is normalized – its elements sum to unity – and elements x_{ij} represent the similarity score of the vertices $i \in V_B$ and $j \in V_A$. The second step of the algorithm uses a maximum-weight bipartite matching algorithm to find the best matching pairs of vertices in graphs G_A and G_B based on the similarity scores X computed in the first step.

The IsoRank iteration kernel is of the form:

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

where H is a known elemental similarity score matrix that quantifies the similarity of vertex pairs based on a-priori knowledge. This knowledge may be in the form of vertex label distances, distances derived from other vertex characteristics, etc. Note that the topological connectivity of vertices does not figure in this elemental similarity score. The computation in Eq. (1) – $\alpha\tilde{B}X\tilde{A}^T$ (a triple-matrix product) – implements the recursive intuition behind this similarity computation approach. The factor $\alpha \in [0, 1]$ is the damping factor that denotes the relative contribution of the topological component to matrix X; the remaining $1-\alpha$ portion is injected at each iterative step by the independent similarity information represented in matrix H.

Although semantically appealing, this iteration is hard to apply to graphs with hundreds of thousands of vertices and beyond, since the storage requirements for (dense) matrix X outgrow the physical memory of typical computing platforms. Parallelizing, and thus distributing the storage, across a compute cluster (or cloud) is a potential solution to this problem. However, the triple-matrix-product introduces significant compute overheads.

To address the high serial complexity of IsoRank, we recently proposed an acceleration technique called Network Similarity Decomposition (NSD), which drastically reduces the computation and memory requirements of the algorithm in specific (frequently-encountered) cases. NSD adopts the approach of IsoRank, but models the similarity computation as a series of matvecs, instead, to avoid the costs of a triple-matrix-product. The series-of-matvecs formulation is briefly described as follows:

without loss of generality, we use H as the initial condition $X^{(0)}$, and after t iterations, $X^{(t)}$ takes the form

$$X^{(t)} = (1-\alpha)\sum_{k=0}^{t-1}\alpha^k\tilde{B}^k H(\tilde{A}^T)^k + \alpha^t\tilde{B}^t H(\tilde{A}^T)^t. \tag{2}$$

However, matrix H can be decomposed into a set of s vector pairs (components) that can generally be expressed as:

$$H = \sum_{i=1}^{s} w_i z_i^T. \tag{3}$$

Substituting decomposition (3) in Eq. (2) yields

$$X^{(t)} = \sum_{i=1}^{s} X_i^{(t)}, \tag{4}$$

where

$$X_i^{(t)} = (1-\alpha)\sum_{k=0}^{t-1}\alpha^k w_i^{(k)} {z_i^{(k)}}^T + \alpha^t w_i^{(t)} {z_i^{(t)}}^T, \tag{5}$$

and $w_i^{(k)} = \tilde{B}^k w_i$, $z_i^{(k)} = \tilde{A}^k z_i$. This formulation provides the basis for the NSD method. The method is presented in Algorithm 1. For more details on NSD, including the derivation of the associated expressions, please see [23].

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: $X^{(n)} \leftarrow \sum_{i=1}^{s} X_i^{(n)}$
```

3.3. Auction-based bipartite weighted matching

Auction algorithms find the maximum weighted matching via an *auction*: V_A and V_B represent the set of buyers and objects, respectively. This metaphor naturally implies the concurrent activity reflecting on the amenability of these algorithms to parallel implementations. 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 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, the vertex matching naturally follows from the 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 the qualitative assessment of graph matching, we refer readers to [23].

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

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: $I \leftarrow \{i : 1 \le i \le n_A\}$                    ▷ set of unassigned buyers
 3: $p_j \leftarrow 0$ for $j = 1, \dots, n_B$                ▷ initialize prices for objects
 4: initialize($\varepsilon$)                                 ▷ initialize $\varepsilon$
 5: while $I \neq \emptyset$ do                               ▷ auction iteration
 6:     $j_i \leftarrow \arg\max_j \{w_{ij} - p_j\}$          ▷ find best object of buyer i
 7:     $u_i \leftarrow w_{ij_i} - p_{j_i}$                   ▷ store profit of the best object
 8:     $v_i \leftarrow \max_{j \neq j_i} \{w_{ij} - p_j\}$   ▷ 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
```

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 $k = 1$ 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: end for
10: for $i = 1, \dots s, k = 0, \dots, n$ do
11:     Partition $w_i^{(k)}$ in $p$ fragments, $w_{i,1}^{(k)}, \dots, w_{i,p}^{(k)}$
12:     Partition $z_i^{(k)}$ in $q$ fragments, $z_{i,1}^{(k)}, \dots, z_{i,q}^{(k)}$
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, $\forall i = 1, \dots s, k = 0, \dots, n$ $(r = 1, \dots, p, u = 1, \dots, q)$
15: Receive corresponding $w_{i,r}^{(k)}, z_{i,u}^{(k)}$ fragments, $\forall i = 1, \dots, s, k = 0, \dots, n$ 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: $X_{ru}^{(n)} \leftarrow \sum_{i=1}^{s} X_{i,ru}^{(n)}$
```

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 their 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 parallel testbeds. More specifically, in Algorithm 3 the iterates are computed by the root process and are subsequently partitioned and distributed to a $p \times q$ process grid (lines 2–14). Outer products are then independently computed, and the final, naturally distributed, similarity matrix is synthesized by worker processes (lines 15–23).

Setting $p = 1$ (or $q = 1$), we reduce this to a 1-D formulation. In this scenario, choosing to parallelize only the second part 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. We partition the set of “buyers” only; this means a buyer has access to all objects within the boundaries of a single process saving in communication and synchronization. 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 the 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 together. 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 the auction-based bipartite matching described in Algorithms 2 and 4. Consider line 9 in 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

Algorithm 4 Parallel Auction Algorithm for Weighted Matchings

```
Input: Bipartite graph $G = (V_A, V_B, E, w)$
Output: Matching $M_{\text{global}}$
 1: $M_{\text{local}} \leftarrow \emptyset$    ▷ set of locally matched buyers
 2: $I_{\text{local}} \leftarrow \{i : 1 \le i \le \frac{n_A}{P}\}$    ▷ 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 $j = 1, \dots, n_B$    ▷ 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()    ▷ communication phase
13:     check_winner($i$)    ▷ 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
18: $M_{\text{global}} \leftarrow \text{gather}(M_{\text{local}})$
```

Algorithm 5 Adaptive Parallel Auction Algorithm

```
 1: Perform the initialization phase of algorithm 4 (lines 1–4)
 2: $\theta \leftarrow 16$; $\gamma \leftarrow \frac{n+1}{\theta}$
 3: $\delta \leftarrow \left\lfloor \min\left\{\frac{|I_{\text{global}}|}{2}, \frac{n}{\theta}\right\}\right\rfloor$    ▷ 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:             $\varepsilon_{\text{local}} \leftarrow \varepsilon_{\text{local}} \cdot 2$
10:         else
11:             $\varepsilon_{\text{local}} \leftarrow \gamma$
12:         end if
13:         $\gamma \leftarrow \gamma/2$
14:     end while
15:     $\delta \leftarrow \delta/2$; $\theta \leftarrow \theta \cdot 2$    ▷ update $\delta$ and $\theta$
16: end while
```

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. A practical way of looking at ε is as an indicator of the aggressiveness of the auction. Large values of ε, although increasing the risk for a buyer to pay an unnecessarily high price for an object, accelerate assignments, thus shortening the time for the auction to come to an end.

We have evaluated various ε-scaling strategies in order to identify the one resulting in best performance (in terms of the number of auction rounds and the sum of weights for the computed matchings). Please refer to [44] for a comprehensive presentation of available options; note that two of the authors of our current work are also the first two authors in the cited publication. We use the following strategy here: the value of $\varepsilon_{\text{local}}$ is initialized, to a small value and adaptively increased relatively to the overall progress (see Algorithm 5). More specifically, at each round, while the number of free buyers in the auction exceeds a threshold value δ – that decreases dynamically – we get more aggressive by increasing $\varepsilon_{\text{local}}$, however constrained by a "ceiling" value of γ, which also changes in the inner loop of Algorithm 5. In practice, this approach matches a buyer faster in the early stage of the algorithm. This aggressive ε-scaling strategy is embedded in the main routines in Algorithm 4 and delivers a maximal matching with the maximum weight; however the quality of the match is adequate in the context of graph similarity. 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 with 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 10^6 vertices each, is a dense matrix of 10^{12} entries. This requires a distributed memory of 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 jth 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 row indices j;
- once a row of the similarity matrix is constructed, retain only the k largest values in the row (with $k \ll n$) 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 to trade-off available memory and input network sizes. Our intention here is to provide a practical and intuitive approximation strategy, rather than a formally quantified pruning solution. We empirically note that this strategy works remarkably well for our protein–protein network alignment, which was the most convenient to check for a range of sparsification factors: Table 1 shows 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 the two PPI networks.

Table 1
Let k be the number of the largest values retained from each row of the similarity matrix we generate. For various values of k we compute the number of conserved edges resulting from this 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.

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%)

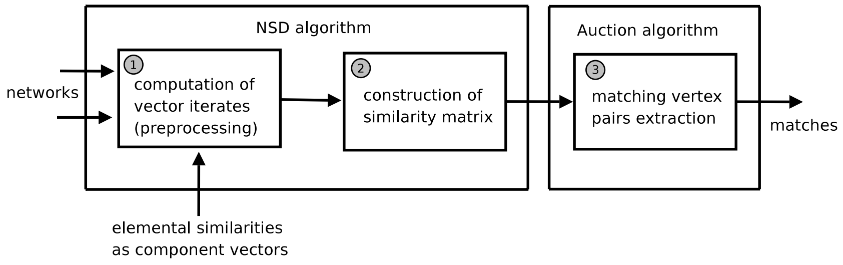

NSD algorithm
Auction algorithm
networks
1 computation of vector iterates (preprocessing)
2 construction of similarity matrix
3 matching vertex pairs extraction
matches
elemental similarities as component vectors

Fig. 2. 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.

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 all steps k ($z_i^{(0)} = z_i$, $w_i^{(0)} = w_i$) do
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: end for
10: compute row-wise the local similarity matrix $X_r$ (embarrassingly //)
11:  ▷ NSD-based, sparsify if needed (sort row entries, keep largest ones);
12: compute weighted matchings using the parallel auction algorithm
13:                          ▷ matching permutation lands on root;
14: □ compute number of conserved edges, similarity rate;
```

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 Section 4.3 (sparsification), in Algorithm 4 (parallel matching) and in Section 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 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 the auction matching stage that follows this step in the integrated pipeline of Fig. 2 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 $C = \max_{ij} |x_{ij}|$ [3].

5. Experimental results

We present 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 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, the parameter α used in similarity matrix construction is set to 0.8 (recall that α is the fraction of the similarity score that comes from the topological similarity; the rest comes from the elemental similarity), the number of iterations is fixed. Further, $s = 10$ randomly generated components were input in all runs; this choice reflects the fact that no specific, a-priori matching preferences are available in general.

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 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 in 3072 compute cores, at maximum. The PathScale programming environment (version 3.1.61) is used with its accompanying compiler.

As datasets, 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 [10], the Wikipedia datasets containing its inter-article link structure [5], and well-known protein–protein interaction networks [47]. We also report 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 speedup plots.

5.2. *Results with sparsification*

In this set of experiments, we construct a sparsified version of the similarity matrix. 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 1024 cores. We note that our algorithm can extract matching pairs for half-a-million vertex networks in less than 30 min using our cluster (see Table 4).

Table 2
Characteristics of networks (organized in pairs) used in experiments. Note the extra separators defining the 7 graph pair sets, directly correlated to the containing graph size ranges; it is the size that drives the different experimental configurations (baseline and increments in core counts).

Pair	Graph	#Vertices	#Edges
(a)			
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
(b)			
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

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

Pair	Total time (s)	#Cores
(a)		
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
(b)		
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

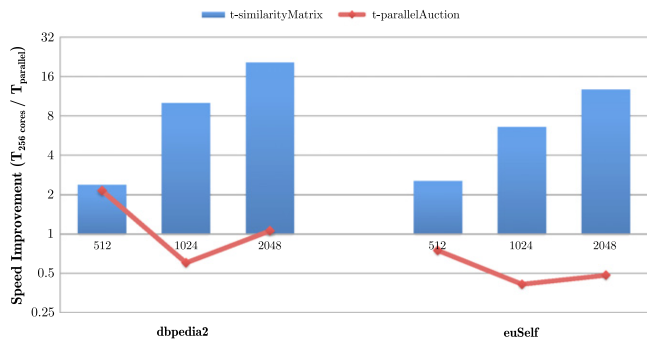


Fig. 3. Speed improvement of the similarity matrix construction in the strong scaling sense, and the parallel auction in the weak scaling sense by using up to 2048 compute cores.

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 (which also implies a decrease in the number of rows locally assigned), so as to utilize the full 16 GB memory available for each socket of the 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 results for auction matching and strong scaling for similarity matrix construction are shown in Fig. 3 for up to 2048 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

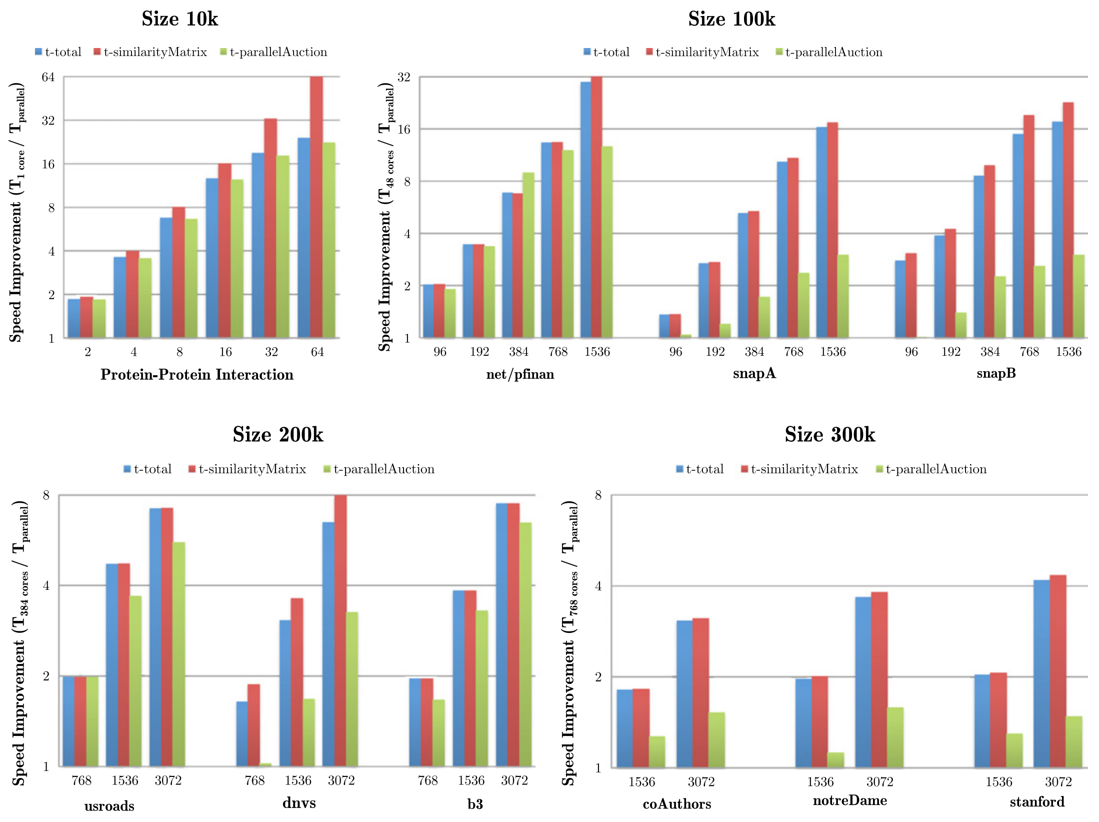
The scalability of our approach is also demonstrated for datasets and configurations without using sparsification. In Fig. 4 near linear speedup is reported for parallel similarity matrix construction and also for the overall time. This follows from the fact that parallel auction matching scales reasonably 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 achieved through parallelization: extracting matching pairs for two typical networks using 64 cores takes only about three seconds. Using sequential state-of-the-art approaches like IsoRank [47] these matchings require about 1.5 h for a solution of comparable quality. Fig. 4 presents timing results for the largest instances tested. Similarity computation requires 256 sockets (3072 cores) to store the entire data.

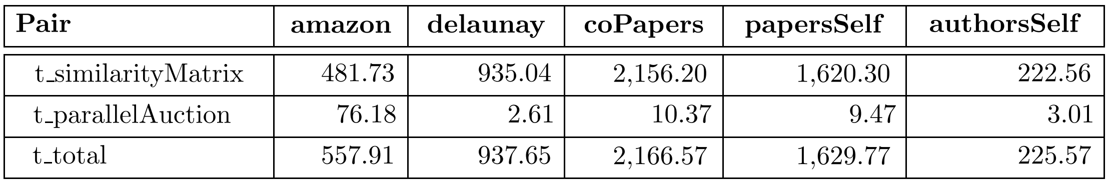
5.3.1. Comparison with the multithreaded network alignment of Khan et al. [21]

There are a number of fundamental differences between our approach and the parallelization reported by Khan et al. [21], apart from their target platform of shared-memory machines. These differences relate to the encoding of prior preferences—their method uses a sparse matrix representation of similarities (matrix L), whereas our method relies on a sum of outer vector products (matrix H is typically dense). Their matchings are constrained to non-zero pairs in matrix L. It is not therefore straightforward to encode, in L, the absence of prior similarities—the case where all node alignments are equally likely, a-priori. This is in fact

Table 4
Timing results (in s) 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 10–11), t_parallelAuction corresponds to lines 12–13 and t_total is the total time elapsed.

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





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

Fig. 4. Speed improvement and timing results (in s) from the major steps in our integrated approach for variable-sized dataset using up to 3072 compute cores.

the more frequently encountered case in real applications since information for constructing node similarity priors for large graphs is rarely available. Even in cases where this information is available, constructing and storing such a matrix may itself be prohibitively expensive. In contrast, our formulation is well-suited to this case. Conversely, our method cannot effectively integrate multiple vertex–vertex priors which is the preferred format for L. In our experiments, input matrices H are dense—though not explicitly formed. They are composed of $s = 10$ randomly generated rank-1 components. Obviously the construction of the corresponding matrix L proved difficult even for the smallest network pair (protein–protein), well exceeding 1 GB of storage. Consequently, we generated sparse random matrices L with nonzero densities of 1% (`ppi1`) and 10% (`ppi10`), with dimensions commensurate to our fly/yeast datasets.

For 400 iterations, batch size $r = 10$ and the default parameters in the code provided by Khan et al. (compiled with Gnu C++ Compiler, version 4.5.3) we run experiments for up to 64 cores in 8-processor/80-core system based on Intel Xeon 2.40 GHz CPU. Pure solution times lie in the range of 25.8 s (1 core) down to 5.8 s (16 cores) a maximum speedup approximately of 4×. Although these numbers seem competitive with ours as absolute numbers for the same dataset pair (but with dense preferences)—about 3 s for 64 cores and 75 s for the single core (see Table 3 and Fig. 4, top left plot), this is not the case:

- scalability is generally poor in [21], only up to ×15 with saturation at around 40 cores; we include computations scaling up to 3072 cores;

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

Pair	#CE	Rate
(a)		
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
(b)		
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

- experimenting with `ppi10` (which is still $\times 10$ less dense than our H) absolute solution times increase considerably (e.g., 55.4 s for the 32-core run).

Perhaps, most interestingly, the topological quality of our matching is highly competitive: we conserve 745 edges in the mapping while our L for `ppi1` only reports 89 edge "overlaps". Although partly expected, since the mappings of Khan et al. are a subset of nonzero entries injecting the biologically motivated priors gives a maximum of 381 edge overlaps (see Fig. 3 in [21]).

5.4. *Quality measurement*

Up to 3072 cores are used for matching up to approximately 500k-vertex networks. We report on the similarity rate, that is a "normalized" version of 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, 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.

Acknowledgments

The authors wish to thank the Swiss National Supercomputing Centre CSCS in Manno, Switzerland for generously providing resources to perform the experimental part of this work. This work is supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370, and by NSF grants DBI 0835677 and 0641037.

References

[1] S. Bandyopadhyay, R. Sharan, T. Ideker, Systematic identification of functional orthologs based on protein network comparison, Genome Res. 16 (3) (2006) 428–435.
[2] M. Bayati, M. Gerritsen, D.F. Gleich, A. Saberi, Y. Wang, Algorithms for large, sparse network alignment problems, in: Ninth IEEE International Conference on Data Mining, 2009, ICDM'09, IEEE, 2009, pp. 705–710.
[3] D.P. Bertsekas, The auction algorithm: a distributed relaxation method for the assignment problem, Ann. Oper. Res. 14 (1) (1988) 105–123.
[4] D.P. Bertsekas, D.A. Castañon, A forward/reverse auction algorithm for asymmetric assignment problems, Comput. Optim. Appl. 1 (1993) 277–297.
[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, Dbpedia—a crystallization point for the web of data, in: Web Semantics: Science, Services and Agents on the World Wide Web, 2009.
[6] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, P. Van Dooren, A measure of similarity between graph vertices: applications to synonym extraction and Web searching, SIAM Rev. 46 (4) (2004) 647–666.
[7] U. Brandes, A faster algorithm for betweenness centrality, J. Math. Sociol. 25 (2) (2001) 163–177.
[8] Ü.V. Çatalyürek, F. Dobrian, A.H. Gebremedhin, M. Halappanavar, 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, IEEE Press, 2011, pp. 1966–1975.
[9] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, 1990.
[10] T.A. Davis, Y. Hu, The university of Florida sparse matrix collection, ACM Trans. Math. Software 38 (1) (2011).
[11] D.E. Drake, 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, 2003, pp. 107–119.
[12] I.S. Duff, J. Koster, On algorithms for permuting large entries to the diagonal of a sparse matrix, SIAM J. Matrix Anal. Appl. 22 (1999) 973–996.
[13] M. El-Kebir, J. Heringa, G. Klau, Lagrangian relaxation applied to sparse global network alignment, Pattern Recognit. Bioinform. (2011) 225–236.
[14] J. Flannick, A. Novak, C.B. Do, B.S. Srinivasan, S. Batzoglou, Automatic parameter learning for multiple network alignment, in: Proceedings of the 12th Annual International Conference on Research in Computational Molecular Biology, Springer-Verlag, 2008, pp. 214–231.
[15] L. Getoor, C.P. Diehl, Link mining: a survey, ACM SIGKDD Explor. Newsl. 7 (2) (2005) 3–12.
[16] D. Gregor, A. Lumsdaine, The parallel BGL: a generic library for distributed graph computations, in: Parallel Object-Oriented Scientific Computing, POOSC, 2005.
[17] S. Hougardy, D.E. Vinkemeier, Approximating weighted matchings in parallel, Inform. Process. Lett. 99 (3) (2006) 119–123.
[18] G. Jeh, J. Widom, SimRank: a measure of structural-context similarity, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2002, pp. 538–543.
[19] R. Jonker, A. Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, Computing 38 (4) (1987) 325–340.
[20] B.P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B.R. Stockwell, T. Ideker, PathBLAST: a tool for alignment of protein interaction networks, Nucleic Acids Res. 32 (Suppl. 2) (2004) W83.
[21] Arif M. Khan, David F. Gleich, Alex Pothen, Mahantesh Halappanavar, A multithreaded algorithm for network alignment via approximate matching, in: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), IEEE, 2012, pp. 1–11.
[22] G.W. Klau, A new graph-based method for pairwise global network alignment, BMC Bioinform. 10 (Suppl. 1) (2009) S59.
[23] G. Kollias, S. Mohammadi, A. Grama, Network similarity decomposition (NSD): a fast and scalable approach to network alignment, IEEE Trans. Knowl. Data Eng. 24 (2012) 2232–2243.
[24] M. Koyutürk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, A. Grama, Pairwise alignment of protein interaction networks, J. Comput. Biol. 13 (2) (2006) 182–199.

[25] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, N. Pržulj, Topological network alignment uncovers biological function and phylogeny, J. R. Soc. Interface 7 (50) (2010) 1341–1354.
[26] Oleksii Kuchaiev, Natasa Przulj, Integrative network alignment reveals large regions of global network similarity in yeast and human, Bioinformatics 27 (10) (2011) 1390–1396.
[27] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1955) 83–97.
[28] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proceedings of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society, 2001, pp. 313–320.
[29] Z. Li, S. Zhang, Y. Wang, X.S. Zhang, L. Chen, Alignment of molecular networks by integer quadratic programming, Bioinformatics 23 (13) (2007) 1631–1639.
[30] F. Manne, 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, in: Lecture Notes in Computer Science, vol. 4967, 2008, pp. 708–717.
[31] V. Memišević, N. Pržulj, C-GRAAL: Common-neighbors-based global GRAph ALignment of biological networks, Integr. Biol. (2012).
[32] T. Milenković, W.L. Ng, W. Hayes, N. Pržulj, Optimal network alignment with graphlet degree vectors, Cancer Inform. 9 (2010) 121.
[33] T. Milenković, N. Pržulj, Uncovering biological network function via graphlet degree signatures, Cancer Inform. 6 (2008) 257–273. PMID: 19259413.
[34] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. 5 (1) (1957) 32–38.
[35] M.E.J. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. 103 (23) (2006) 8577–8582.
[36] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, Stanford University, 1998.
[37] P. Papadimitriou, A. Dasdan, H. Garcia-Molina, Web graph similarity for anomaly detection, J. Internet Serv. Appl. 1 (1) (2010) 19–30.
[38] Rob Patro, Carl Kingsford, Global network alignment using multiscale spectral signatures, Bioinformatics 28 (23) (2012) 3105–3114.
[39] M.A. Patwary, R.H. Bisseling, 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, 2010, pp. 45–54.
[40] S. Pettie, P. Sanders, A simpler linear time $2/3-\epsilon$ approximation for maximum weight matching, Inform. Process. Lett. 91 (6) (2004) 271–276.
[41] R. Preis, Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs, in: STACS 99, Springer, 1999, pp. 259–269.
[42] N. Pržulj, Biological network comparison using graphlet degree distribution, Bioinformatics 23 (2) (2007) e177–e183.
[43] M. Rupp, E. Proschak, G. Schneider, Kernel approach to molecular similarity based on iterative graph similarity, J. Chem. Inf. Model. 47 (2007).
[44] Madan Sathe, Olaf Schenk, Helmar Burkhart, An auction-based weighted matching implementation on massively parallel architectures, Parallel Comput. 38 (12) (2012) 595–614.
[45] R. Sharan, S. Suthram, R.M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R.M. Karp, T. Ideker, Conserved patterns of protein interaction in multiple species, Proc. Natl. Acad. Sci. USA 102 (6) (2005) 1974.
[46] C. Silverstein, S. Brin, R. Motwani, Beyond market baskets: generalizing association rules to dependence rules, Data Min. Knowl. Discov. 2 (1) (1998) 39–68.
[47] R. Singh, J. Xu, B. Berger, Global alignment of multiple protein interaction networks with application to functional orthology detection, Proc. Natl. Acad. Sci. 105 (35) (2008) 12763–12768.
[48] X. Yan, J. Han, gSpan: Graph-based substructure pattern mining, in: Proceedings. 2002 IEEE International Conference on Data Mining, 2002, ICDM 2002, IEEE, 2002, pp. 721–724.

Giorgos Kollias received the B.Sc. in Physics in 2000 and the M.Sc. in Computational Science in 2002 from the University of Athens, Greece, and the Ph.D. in Computer Science from the University of Patras, Greece, in 2009. He is currently a Postdoctoral Researcher at the Center for Science of Information and the Computer Science Department at Purdue University, USA. His research interests include dynamical systems, Problem Solving Environments, graph analysis and parallel computing.

Madan Sathe studied computer science at the TU Dortmund University, Germany. He graduated in 2007 with a diploma thesis in the field of multiobjective optimization in collaboration with the Indian Institute of Technology Kanpur, India. During his doctoral studies in computer science at the University of Basel, Switzerland, he worked as a research assistant in the field of graph algorithms, scientific computing, and high performance computing. During this time he also went on a research visit to Purdue University, USA. He was awarded the Ph.D. from the University of Basel in 2012.

Olaf Schenk is currently an associate professor of computational science at the Universita della Svizzera italiana in Switzerland. He received the diploma degree in mathematics from the Karlsruher Institute of Technology (formerly, University of Karlsruhe), Germany, and the Ph.D. degree from Swiss Federal Institute of Technology, Zurich in Switzerland. Schenk's current research interests include algorithmic and architectural problems in computational mathematics, scientific computing, and high-performance computing. He has authored over 80 research articles, and has coedited or coauthored 4 books including the widely used text book e.g. on ‘‘Combinatorial Scientific Computing’’ published by Chapman and Hall/CRC. In 2008, he received an IBM Faculty Award on Cell Processors for Biomedical Hyperthermia Applications. His research has also resulted in the development of highly efficient parallel algorithms and software for sparse matrix factorization (PARDISO). He is on the editorial board of the SIAM Journal on Scientific Computing.

Ananth Grama received the Ph.D. degree from the University of Minnesota in 1996. He is currently a Professor of Computer Sciences and Associate Director of the Center for Science of Information at Purdue University. His research interests span 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.