Hindawi Publishing Corporation
Computational and Mathematical Methods in Medicine
Volume 2014, Article ID 393908, 13 pages
http://dx.doi.org/10.1155/2014/393908

Research Article

Pin-Align: A New Dynamic Programming Approach to Align Protein-Protein Interaction Networks

Farid Amir-Ghiasvand, Abbas Nowzari-Dalini, and Vida Momenzadeh

Department of Computer Science, School of Mathematics, Statistics, and Computer Science, College of Science, University of Tehran, Tehran 1417614411, Iran

Correspondence should be addressed to Abbas Nowzari-Dalini; nowzari@ut.ac.ir

Received 16 June 2014; Accepted 15 October 2014; Published 10 November 2014

Academic Editor: Emil Alexov

Copyright © 2014 Farid Amir-Ghiasvand et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To date, few tools for aligning protein-protein interaction networks have been suggested. These tools typically find conserved interaction patterns using various local or global alignment algorithms. However, the improvement of the speed, scalability, simplification, and accuracy of network alignment tools is still the target of new researches. In this paper, we introduce *Pin-Align*, a new tool for local alignment of protein-protein interaction networks. *Pin-Align* accuracy is tested on protein interaction networks from IntAct, DIP, and the Stanford Network Database and the results are compared with other well-known algorithms. It is shown that *Pin-Align* has higher sensitivity and specificity in terms of KEGG Ortholog groups.

1. Introduction

In the last decade, high throughput techniques in experimental biology produced a large amount of biological data which usually can be modeled by large networks such as metabolic networks, gene regulatory networks, and protein-protein interaction (PPI) networks [1]. Analogous to biological sequence comparison [2–4], comparing large biological networks can improve our biological understanding of them. Comparison of PPI networks is a well-studied area due to the important roles of PPI networks. The main idea behind comparison of PPI networks is to find evolutionary conserved interaction modules which describes functional relevance. Exact comparison of large PPI networks is a NP-hard problem and there is no polynomial time algorithm for it. The NP-hardness of this problem is based on the fact that this problem can be reduced to the subgraph isomorphism problem. In this way, the exact comparison of PPI networks is computationally infeasible, and PPI networks comparison is often addressed by heuristic methods [5–7].

Several algorithms have been proposed for biological network alignment. One of the first proposed network alignment algorithms is *Path-Blast* [8]. This algorithm finds high-score conserved pathways. After that, Sharan et al. designed *Network-Blast* [9] in order to identify conserved protein complexes in multiple species. Koyuturk et al. devised an evolution-based scoring scheme to detect conserved clusters, called *MaWISh* (maximum weight induced subgraph) [10]. Flannick et al. proposed *Graemlin* as the first method to detect conserved subnetworks of arbitrary structures with progressive alignment method [11]. All these mentioned algorithms are called local alignment algorithms owing to the fact that they started to find subnetworks such as pathways and protein complexes and expanded their search results to obtain the feasible alignment. In local network alignment, each protein may be mapped to several proteins. On the other hand, global network alignment algorithm is defined to detect the best overall mapping across all parts of the input networks. *IsoRank* is the first global network alignment algorithm aiming to maximize the overall match between two networks [12]. Flannick et al. extended *Graemlin* to global network alignment by using a training set of known network alignments to learn parameters for the scoring function. This novel algorithm is called *Graemlin 2.0* and is claimed to have linear time complexity with the number of PPIs [13]. The next version of *IsoRank*, *IsoRankN* [14], uses *Nibble-Page-Rank* algorithm [15] to align input networks locally and globally. Tian and Samatova introduced a connected-components

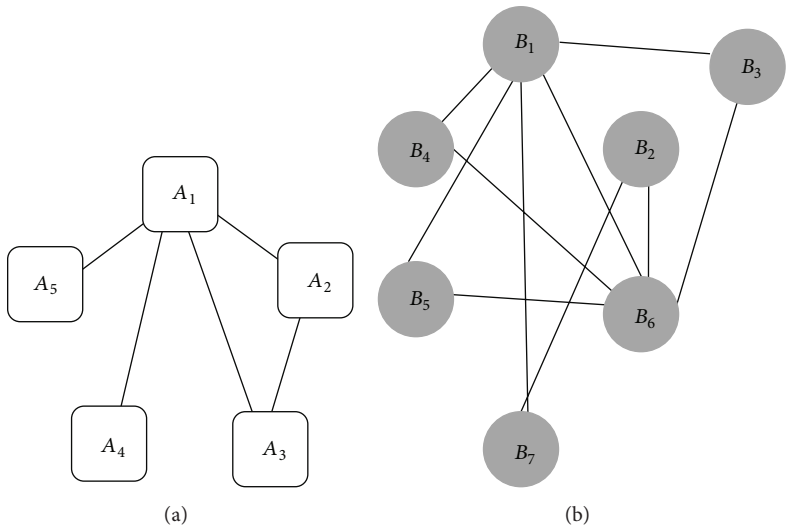


Figure 1: Two sample networks A and B.

based algorithm, called *HopeMap*, for pairwise network alignment with the focus on fast identification of maximal conserved patterns [16]. After that, *GRAAL* [6] and *H-GRAAL* [1] are presented as the global network alignment algorithms based on seed-extend approaches and Hungarian algorithm, respectively. Both of *GRAAL* and *H-GRAAL* algorithms are capable of aligning any kind of network owing to the fact that they are specifically designed based on network structure and do not use sequence similarity data. Observing that the size of true orthologs across species is small comparing to the total number of proteins in all species, Tian and Samatova presented a different approach based on a precompiled list of homologs identified by KO terms [17].

In this paper, *Pin-Align* (protein interaction network alignment) algorithm is introduced as a novel local network alignment algorithm for aligning two protein-protein interaction networks. In *Pin-Align*, Hub Clustering and hierarchical clustering algorithms are applied in the heuristic and dynamic programming phases in order to reduce time and space complexity of the problem. Two different types of scoring, *Node-Scoring* and *Structural-Scoring*, are used to find the best local alignments. *Node-Scoring* is simply based on BLAST bit score [18] and *Structural-Scoring* is based on the importance of *Bridges* (*Bridges* are considerable edges, which take part in most of the paths and consequently conserved paths in the networks) in biological networks.

The *Pin-Align* algorithm is performed on the protein interaction networks of *Escherichia coli (E. coli), Salmonella typhimurium (S. typhimurium), Caulobacter crescentus (C. crescentus), human, mouse, yeast, and Drosophila melanogaste (fly)* extracted from IntAct [19], DIP [20], and Stanford Network database (SNDB) [21]. The obtained results are compared with other well-known local alignment algorithms NetworkBLAST [9], MaWISh [10], Graemlin [11], and Graemlin 2.0 [13], in a way that Graemlin 2.0 compares its results, because it is the only method that compares results of all local alignment tools against KEGG Ortholog (KO) groups [22].

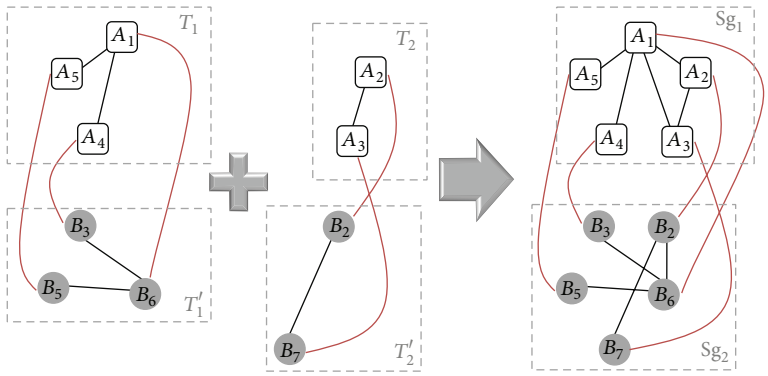
2. Materials and Methods

Protein-protein interaction (PPI) network is defined as the set of relationships among proteins. Here, a PPI network is modeled by an undirected and weighted graph $G = (V, E)$, where nodes correspond to proteins and each weighted edge specifies the probability that two proteins interact. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ (such as two sample networks A and B in Figure 1), the goal of network alignment is to identify possible mappings which map vertices of G_1 into vertices of G_2.

In addition, for each mapping, the corresponding set of conserved edges is also identified. Mappings may be partial; that is, they do not need to be defined for all the nodes of the graphs. Each mapping implies a common subgraph between the two graphs. When protein a_1 from graph G_1 is mapped to protein a_2 from graph G_2, then a_1 and a_2 refer to the same node in the common subgraph; the edges in the common subgraph correspond to the conserved edges [23].

Pin-Align algorithm is a local network alignment algorithm, mainly designed based on the dynamic programming approach. To explain *Pin-Align* in more detail, first a formal definition of local network alignment is presented. Given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, local alignment $\mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$ is a triplet where Sg_1 and Sg_2 are the subgraphs of G_1 and G_2, respectively, and M is defined as the mapping which aligns vertices of Sg_1 into vertices of Sg_2. Two sample local alignments of networks A and B of Figure 1 are shown in Figure 2(a).

Following, the best local network alignment (BLA) is defined as a function to obtain the best local alignment



(a) (b)

Figure 2: (a) Two local alignments of networks A and B of Figure 1. (b) Result of merging them.

Input: Graph $G_1(V_1, E_1)$, Graph $G_2(V_2, E_2)$ such that $|V_1| = n$
Output: Local alignment
(1) Define S as an array of sets with length n
(2) $S[1]$ = Set of all local alignments of size one
(3) **for** $size = 2$ to n **do**
(4) **for** $i = 1$ to $size - 1$ **do**
(5) $j = size - i$
(6) **for** $k = 1$ to $|S[i]|$ **do**
(7) **for** $\ell = 1$ to $|S[j]|$ **do**
(8) **LA** $A = k$'th element of $S[i]$
(9) **LA** $B = \ell$'th element of $S[j]$
(10) $S[size] = S[size] \bigcup Merge(A, B)$
(11) **end for**
(12) **end for**
(13) **end for**
(14) **end for**
(15) **return** The LA with highest score among all LAs in $S[n]$

Algorithm 1: *Simple-Align* algorithm.

for each subgraph of G_1. The best local alignment of Sg_1 as a subgraph of G_1 with a subgraph of G_2 is shown by $BLA_{G_1,G_2}(Sg_1)$, which represents a local alignment with the highest score among all possible alignments between Sg_1 and all possible subgraphs of G_2 that can be mapped to Sg_1.

Now, a naive dynamic programming algorithm for network alignment is demonstrated in the next subsection, and later *Pin-Align* algorithm is presented.

2.1. Dynamic Programming Algorithm to Align PPI Networks. In alignment of two weighted graphs G_1 and G_2 through dynamic programming technique, we generate all LAs of size one (subgraphs with a single vertex) in the first step. For obtaining LAs of larger subgraphs of G_1, we merge all possible pairs of LA of smaller subgraphs. In other words, for each subgraph of G_1 such as Sg_1, there are several pairs of distinct subgraphs (subgraphs with no common vertices) such as (T_1, T_2) where $Sg_1 = T_1 \cup T_2$. Figure 2 demonstrated the merge process of two sample local alignments of networks A and B of Figure 1. We produce all possible local alignments for Sg_1 such as $LA_{G_1,G_2}(Sg_1, Sg_2, M)$ by merging every possible local alignments of T_1 such as $LA_{G_1,G_2}(T_1, T_1', M')$ with every possible local alignment of T_2 such as $LA_{G_1,G_2}(T_2, T_2', M'')$. This process is done iteratively, and finally all local alignments of G_1 including $BLA_{G_1,G_2}(G_1)$ are obtained.

Inspired by dynamic programming technique, we propose *Simple-Align* algorithm in Algorithm 1. This algorithm can find all BLAs; however by considering time and memory complexity, this algorithm is not feasible. To overcome this, we use several heuristics to reduce memory and time complexity and propose a new local alignment algorithm, *Pin-Align*, in next section.

2.2. Pin-Align Algorithm. As explained, *Pin-Align* uses dynamic programming approach to solve the local network alignment problem and overcome the deficiencies of dynamic programming such as time and space complexity by applying heuristic approach. The steps of the *Pin-Align* algorithm are summarized as follows.

(1) Partition the input graph G_1 into smaller clusters. These clusters are dense subgraphs of G_1. The partitioning is done by using a new clustering algorithm named Hub Clustering. Clusters obtained in this step are named *Hub-Clusters*.

(2) For each *Hub-Cluster*$_i$,

(a) create local alignments of size one with BLAST bit score greater than 0, between vertices of *Hub-Cluster*$_i$ and vertices of G_2; local alignments obtained by this approach are named *Candidates* ($C = \mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$ such that Sg_1 is a subgraph of *Hub-Cluster*$_i$ with size one),

(b) collect all *Candidates* of subgraph Sg_1 of G_1 (such as $C = \mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$) in a new set called *Candidate-Collection* and each *Candidate-Collection* is represented by its subgraph of G_1 such as $\mathrm{CC}(\mathrm{Sg}_1)$; *Candidate-Collections* obtained for single nodes of G_1 are used as seeds in Step 2(c),

(c) merge small *Candidate-Collections* (seeds) to gain larger ones. Repeat this process continuously to achieve a *Candidate-Collection* CC(*Hub-Cluster*$_i$) as the final result of merging process. Size of a *Candidate* or a *Candidate-Collection* is defined by the number of vertices of its subgraph of G_1 and cardinality of a *Candidate-Collection* is defined as the number of its *Candidates*. In this step, each *Candidate-Collection* can be merged with several *Candidate-Collections* and create *Candidate-Collections* with larger size. For determining which *Candidate-Collections* should be merged together, a new hierarchical clustering is used as a pattern of merging.

The *Candidate-Collections* of *Hub-Clusters* are named *Final-Candidate-Collections*.

(3) Based on *Final-Candidate-Collections*, find final alignment using similarity graph. This graph is a special weighted bipartite graph where first and second set of vertices of it are shown by V_1 and V_2 which are the set of vertices of G_1 and G_2. The weight of each edge is computed based on *Final-Candidate-Collections*.

Flowchart of *Pin-Align* algorithm is presented in Figure 3 and its details are explained by an example on two networks A and B given in Figure 1.

In Step 1, in order to reduce the search space of the problem, input graph $G_1 = (V_1, E_1)$ is clustered into some dense subgraphs and Step 2 of the algorithm runs for each dense subgraph separately. Generally PPI networks are sparse graphs, but they have dense regions containing high degree vertices named hubs. These regions usually contain protein complexes; therefore conserved modules among different PPIs usually exist in these regions.

To detect dense subgraphs in the graph G_1, a novel clustering algorithm, called *Hub Clustering*, is proposed. The importance of hub nodes and density of subgraphs are two major criteria for this clustering method. First, a portion of highly connected vertices are selected as hub nodes. The vertices degree of these nodes is greater than 95% of other nodes. The hub nodes are considered as the center vertices of dense regions of graph G_1 and are supposed to be initial clusters. For other nodes of G_1, each node is joint to the nearest cluster (the cluster that contains its nearest hub). For this aim, the length of each edge is defined as the negative logarithm of its weight (from adjacency matrix), and then Dijkstra shortest path algorithm [24] is used to find the nearest cluster. The obtained clusters are named *Hub-Clusters*.

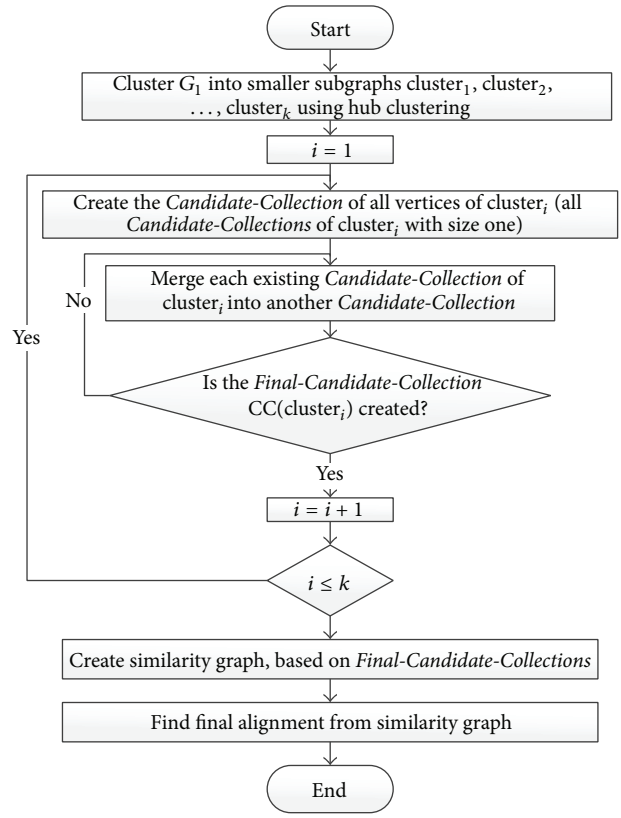


Figure 3: Flowchart of *Pin-Align* algorithm.

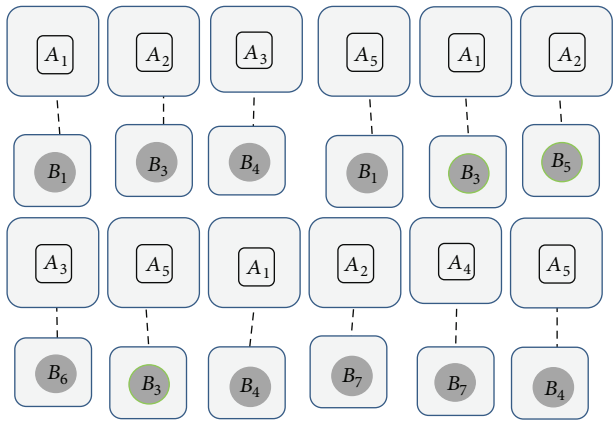
In Step 2, each of *Hub-Clusters* of G_1 (obtained in Step 1) is aligned to G_2. As mentioned, Step 2(a) creates *Candidates* of size one based on BLAST bit score of vertices of two graphs G_1 and G_2 (BLAST bit score is a normalized score which is calculated by basic local alignment search tool, based on sequence similarity [18]). Figure 4(a) shows BLAST bit scores of proteins of networks A and B.

After that, all *Candidates* of each vertex of G_1 are collected into a *Candidate-Collection* with size one as a seed in Step 2(b). In other words, *Candidate-Collection* of each vertex of G_1 such as Sg_1 contains all *Candidates* for Sg_1 from G_2 which can be aligned with Sg_1. For example, *Candidates* of networks A and B of Figure 1 are shown in Figure 4(b) and those *Candidate-Collections* are shown in Figure 4(c).

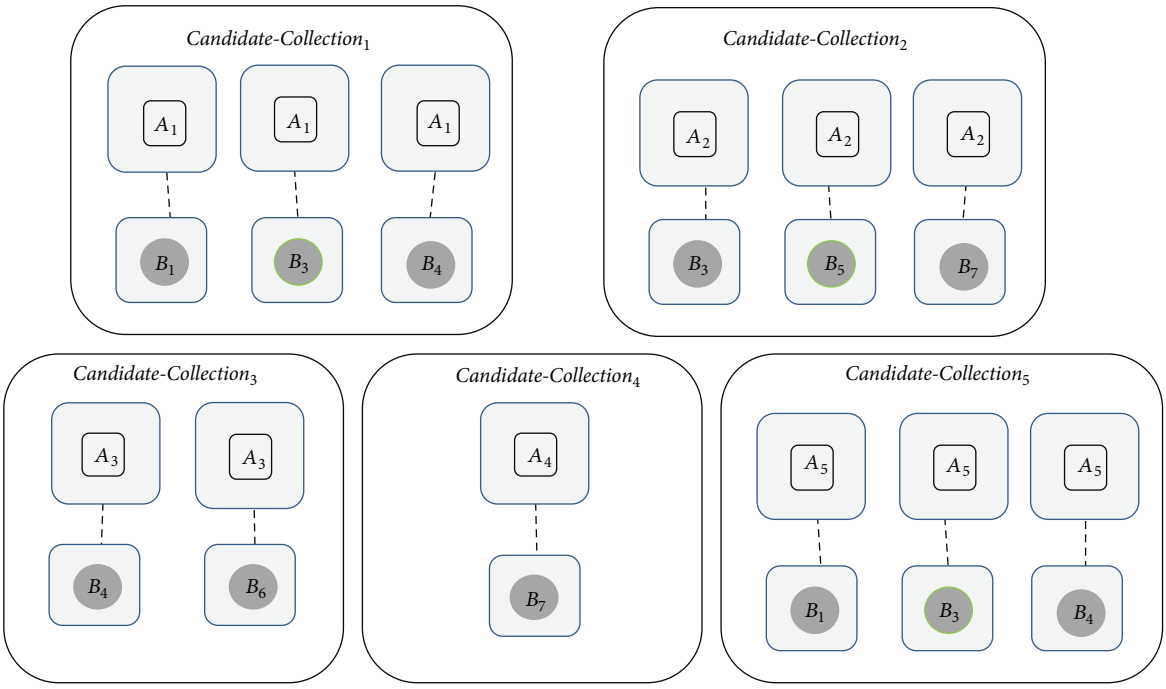
In Step 2(c), two seeds (i.e., *Candidate-Collections* $\mathrm{CC}(\mathrm{Sg}_i)$ and $\mathrm{CC}(\mathrm{Sg}_j)$) are merged in order to produce a

	A_1	A_2	A_3	A_4	A_5
B_1	69.64	0	0	0	49.29
B_2	0	0	0	0	0
B_3	51.83	35.77	0	0	60.08
B_4	66.63	0	4460	0	81.26
B_5	0	1213	0	0	0
B_6	0	0	47.75	0	0
B_7	0	351.6	0	50.45	0

(a)



(b)



(c)

Figure 4: (a) BLAST bit scores for proteins of networks A and B shown in Figure 1. (b) All possible *Candidates* of size one of networks A and B. (c) All possible *Candidate-Collections* of size one for networks A and B.

new seed (new-CC) where its size is sum of sizes of $\mathrm{CC}(\mathrm{Sg}_i)$ and $\mathrm{CC}(\mathrm{Sg}_j)$. Figure 5 illustrates two *Candidate-Collections*, *Candidate-Collection*$_2$ and *Candidate-Collection*$_3$, of Figure 4(c) and the result of merging them.

The important problem in this step is discovering the feasible combining pattern to find pairs of *Candidate-Collections* where merging them generates high score *Candidate-Collections*. In order to find high score *Candidates* and avoid complex searches, combining pattern for merging *Candidate-Collections* should produce *Candidates* with more conserved edges. Hence we use hierarchical clustering as combining patterns [25]. Hierarchical clusters are generally constructed by generating a sequence of partitions where each subcluster belongs to one supercluster in its entity.

In hierarchical clustering we should determine how the distance between two clusters is computed. If $S_i = (V_i', E_i')$ and $S_j = (V_j', E_j')$ are subgraphs of $G_1 = (V_1, E_1)$, the distance between two *Candidate-Collections* $\mathrm{CC}(S_i)$ and $\mathrm{CC}(S_j)$ is computed, using the following formula:

$$\mathrm{dis}\left(\mathrm{CC}\left(S_i\right), \mathrm{CC}\left(S_j\right)\right) = \frac{\left|V_i'\right| + \left|V_j'\right|}{\sum_{v \in V_i'} \sum_{u \in V_j'} \left(a_{v,u}/e_{v,u}\right)}. \quad (1)$$

In this equation, $a_{v,u}$ is equal to one if there is an edge between vertices u and v; otherwise it is zero, and $e_{v,u}$ is the negative logarithm of probability of interaction between two proteins v and u in the graph G_1. We use the hierarchical clustering obtained based on formula, as the combining

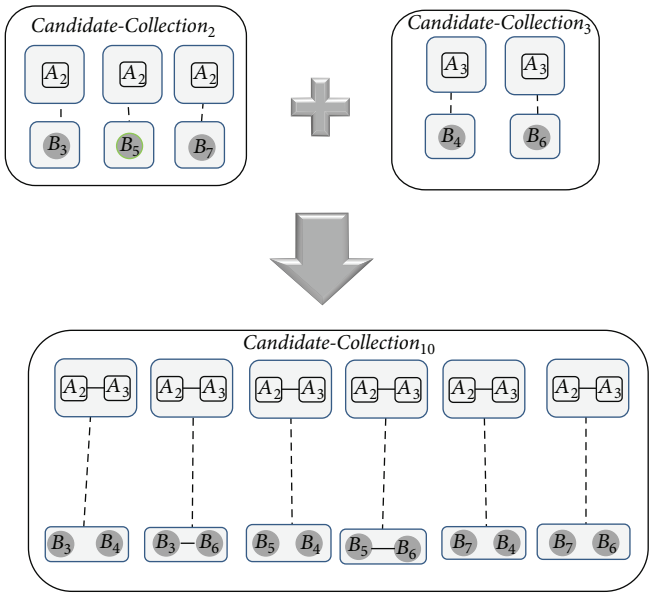


Figure 5: Two *Candidate-Collections* and the result of merging them.

pattern for merging *Candidate-Collections*. Cardinality of new *Candidate-Collections* obtained in higher levels of hierarchical clustering tree is increased exponentially as a result of merging smaller *Candidate-Collections*, so we have to save the *Candidates* with highest score and discard *Candidates* with low scores. In Step 2(c) we use a multiobjective optimization technique for finding the highest score *Candidates* [26]. The highest score *Candidates* are obtained by sorting *Candidates* based on two criteria, first, *Nodes-Scores*; next, *Structural-Scores*. Finally, the *Candidates* with the highest scores are chosen in each step. Following, we describe this scoring schema.

Node-Score. *Node-score* of each *Candidate* $C = \mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$ is obtained by using the following equation:

$$\mathrm{NodeScore}(C) = \sum_{i=1}^{|\mathrm{Sg}_1|} \mathrm{BLASTBitScore}\left(\mathrm{Sg}_1^i, \mathrm{MV}\left(\mathrm{Sg}_1^i, M\right)\right). \tag{2}$$

In this equation, i is an iterator on vertices of Sg_1 and Sg_1^i is the ith vertex of Sg_1; $\mathrm{MV}(v, M)$ indicates a vertex of Sg_2 mapped into $v \in \mathrm{Sg}_1$ by the mapping M.

Structural-Score. Due to the existence of hubs and small words in biological networks, certainly the network contains bridges. Bridges are considerable edges, which take part in most of the paths (specifically conserved paths) in the networks. Because of the importance of bridges, if $\mathrm{Sg}_1 = (V_1', E_1')$ is a subgraph of $G_1 = (V_1, E_1)$ and $\mathrm{Sg}_2 = (V_2', E_2')$ is a subgraph of $G_2 = (V_2, E_2)$, the structural scoring function for each *Candidate* $C = \mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$ is defined as follows:

$$\begin{aligned}\mathrm{StructuralScore}(C) &= \sum_{v \in V_1'} \sum_{u \in V_1'} \left(f(v) \times f(u)\right)^{\alpha} \\ &\quad \times \left(P_{G_1}(v,u)\, P_{G_2}(\mathrm{MV}(v,M), \mathrm{MV}(u,M))\right)^{1-\alpha}.\end{aligned} \tag{3}$$

In this equation, $f(i)$ shows the number of conserved edges at vertex i in the subgraph Sg_1; α is a constant value between 0 and 1 and shows the importance of adjacent vertices degrees; $P_G(x, y)$ is the probability of interaction between protein x and protein y in the network G.

As mentioned before, to control the exponential growth of the cardinality of *Candidate-Collections* in combining process, we use a multiobjective optimization method on node score and structural score, as optimization objectives. In this way, after creating each *Candidate-Collection* among all *Candidates* obtained in that, we select only a limited number of *Candidates* (based on upon scoring functions) and discard others. Obtained *Candidate-Collections* of *Hub-Clusters* are named *Final-Candidate-Collections* which contain *Final-Candidates* (*Candidates* of *Hub-Clusters*).

After creating all *Final-Candidate-Collections*, in Step 3 of *Pin-Align* Algorithm, an *Initial Similarity Bipartite Graph* $\mathrm{ISBG}(G_1, G_2) = (V_1, V_2, E)$ is created which consists of vertices of $G_1 = (V_1, E_1)$ as its first part and vertices of $G_2 = (V_2, E_2)$ as its second part, and E is a set of weighted edges between V_1 and V_2. The weight of each edge represents the similarity between its incident vertices. Let $C(v_i, u_j)$ be a set of all *Final-Candidates* which maps v_i into u_j. In $\mathrm{ISBG}(G_1, G_2)$,

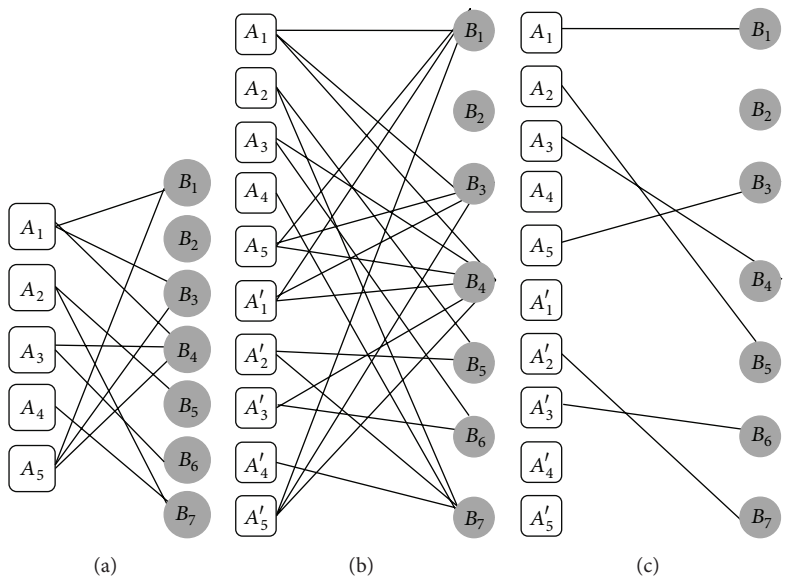


Figure 6: (a) The *Initial Similarity Bipartite Graph* (ISBG) between networks A and B in Figure 1 (the weights of edges are hidden for simplicity). (b) The ISBG, with duplication of first part. (c) Graph M_1, obtained from first part of duplicated ISBG by performing Hungarian algorithm.

the weight of an edge $e = \{v_i, u_j\}$, where $v_i \in V_1$ and $u_j \in V_2$, is equal to zero if $|C(v_i, u_j)| = 0$; otherwise this weight is defined by the following formula:

$$W\left(v_i, u_j\right) = \left(\mathrm{Node}_{\mathrm{Sim}}\left(v_i, u_j\right) + \mathrm{Neighborhoods}_{\mathrm{Sim}}\left(v_i, u_j\right)\right) \times \frac{1}{\left|C\left(v_i, u_j\right)\right|}, \tag{4}$$

where $\mathrm{Node}_{\mathrm{Sim}}(v_i, u_j)$ is the similarity between two proteins v_i and u_j and it is calculated by using the following formula:

$$\mathrm{Node}_{\mathrm{Sim}}\left(v_i, u_j\right) = \mathrm{BLASTBitScore}\left(v_i, u_j\right) \times \left|C\left(v_i, u_j\right)\right|, \tag{5}$$

$\mathrm{Neighborhoods}_{\mathrm{Sim}}(v_i, u_j)$ is the similarity between conserved neighborhoods of v_i and u_j and is calculated as follows:

$$\mathrm{Neighborhoods}_{\mathrm{Sim}}\left(v_i, u_j\right) = \sum_{c \in C(v_i, u_j)} \sum_{(v'_i, u'_j) \in \mathrm{CN}_c(v_i, u_j)} \mathrm{BLASTBitScore}\left(v'_i, u'_j\right). \tag{6}$$

Let $N_G(v)$ be the set of all neighbors of v ($N_G(v) = \{u \in V(G) \mid uv \in E(G)\}$); then $\mathrm{CN}_c(v_i, u_j)$ for *Candidate* $c = \mathrm{LA}_{G_1,G_2}(\mathrm{Sg}_1, \mathrm{Sg}_2, M)$ is

$$\mathrm{CN}_c\left(v_i, u_j\right) = \left\{\left(v'_i, \mathrm{MV}\left(v'_i, M\right)\right) \mid v'_i \in N_{G_1}\left(v_i\right)\right\}. \tag{7}$$

An *Initial Similarity Bipartite Graph* for networks A and B in Figure 1 is shown in Figure 6(a). After creating the *Initial Similarity Bipartite Graph* (ISBG), we can find just a single mapping for each vertex by finding maximum matching in this graph. The maximum matching can be found by applying the Hungarian algorithm (with polynomial time complexity) on the ISBG [27].

As mentioned, this approach can find just a single mapping for each vertex, although it cannot support some evolutionary functions such as duplication and diversion. To support duplication and diversion, we should consider matching of a single vertex from one part into two vertices of other parts in the ISBG. This type of matching is named *DMatch*. In other words, *DMatchs* are paths with length 2 such as $P(v'_i, v_j, v'_k)$ in the ISBG which aligns v_j from one graph into both v'_i and v'_i from another graph. Here the problem is to find a set of *Matchs* and *DMatchs* from the ISBG where sum of their weights is maximum. This problem is named *Maximum Weighted DMatching*. To decrease the size of the solution space of this problem, we decrease the size of edge set of the ISBG by deleting some edges. The above procedure is performed by Algorithms 2 and 3.

Algorithm 2 gives the ISBG as input and creates the *Final Similarity Bipartite Graph*. In Steps 1–4 of Algorithm 2 all vertices of one part of the ISBG are duplicated and each duplicated vertex is connected to all vertices conjunct to the main vertex with the same weight as shown in Figure 6(b); then the Hungarian algorithm is performed on this graph in Step 5 of Algorithm 2. Obtained bipartite graph is named M_1. The result of these steps for ISBG of Figure 6(a) is shown in Figure 6(c). All duplicated vertices of M_1 are merged together in Step 6 of Algorithm 2. In Steps 7–12 of Algorithm 2, the same process is implemented on the other part of the obtained ISBG and the matching M_2 is obtained. Figures 7(a) and 7(b) illustrate these steps for ISBG of Figure 6(a). Step 13 of Algorithm 2 combines M_1 and M_2 to create Sim_G as the *Final Similarity Bipartite Braph* as shown in Figure 7(c).

Input: *Initial-similarity-bipartite-graph* $S = (S_1, S_2, E_S)$
Output: Graph Sim_G
(1) *Bipartite-graph* $SD_1 = S$
(2) **for all** vertex $v \in S_1$ **do**
(3) Duplicate v in SD_1
(4) **end for**
(5) *Bipartite-graph* $M_1 = \textit{Maximum-Weight-Matching}(SD_1)$
(6) Merge duplicated vertices of SD_1 in M_1
(7) *Bipartite-graph* $SD_2 = S$
(8) **for all** vertex $v \in S_2$ **do**
(9) Duplicate v in SD_2
(10) **end for**
(11) *Bipartite-graph* $M_2 = \textit{Maximum-Weight-Matching}(SD_2)$
(12) Merge duplicated vertices of SD_2 in M_2
(13) *Bipartite-graph* $\text{Sim}_G = M_1 \cup M_2$
(14) **return** Sim_G

Algorithm 2: Find Sim_G Graph.

Input: Sim_G Graph M
Output: Final Alignment
(1) **for all** edge $\{v_1, v_2\} \in M$ **do**
(2) **if** $deg(v_1) = 1$ **and** $deg(v_2) = 1$ **then**
(3) $Result = Result \cup \{v_1, v_2\}$
(4) $M = M \setminus \{v_1, v_2\}$
(5) **end if**
(6) **end for**
(7) $DMatch = \phi$
(8) P_1 = first part of M
(9) P_2 = second part of M
(10) **for all** vertex $v \in P_1$ **do**
(11) e_1 = first edge connected to v
(12) e_2 = second edge connected to v **if** $deg(v) > 1$
(13) e_3 = third edge connected to v **if** $deg(v) > 2$
(14) **if** $deg(v) = 2$ **then**
(15) $DMatchs = DMatchs \bigcup (v, e_1, e_2, \text{weight}(e_1) + \text{weight}(e_2))$
(16) **end if**
(17) **if** $deg(v) = 3$ **then**
(18) $DMatchs = DMatchs \bigcup (v, e_1, e_2, \text{weight}(e_1) + \text{weight}(e_2))$
(19) $DMatchs = DMatchs \bigcup (v, e_1, e_3, \text{weight}(e_1) + \text{weight}(e_3))$
(20) $DMatchs = DMatchs \bigcup (v, e_2, e_3, \text{weight}(e_2) + \text{weight}(e_3))$
(21) **end if**
(22) **end for**
(23) SortByWeight ($DMatchs$)
(24) **for all** $dm \in DMatchs$ **do**
(25) **if** $(dm \cap Result = \phi)$ **then**
(26) $Result = Result \cup dm$
(27) **end if**
(28) **end for**
(29) **return** $Result$

Algorithm 3: Find final alignment.

It can be proven that *Maximum Weighted Dmatchings* of graph Sim_G are exactly similar to *Maximum Weighted Dmatchings* of ISBG. So Sim_G contains all edges of *Maximum Weighted Dmatching* and the degree of its vertices is at most 3. For this reason obviously the solution space of the *Maximum Weighted DMatching* problem on Sim_G is considerably decreased in comparison with the ISBG.

To find the final alignment using the graph Sim_G, we use a greedy algorithm with polynomial time complexity as shown in Algorithm 3. Algorithm 3 gives the *Final Similarity*

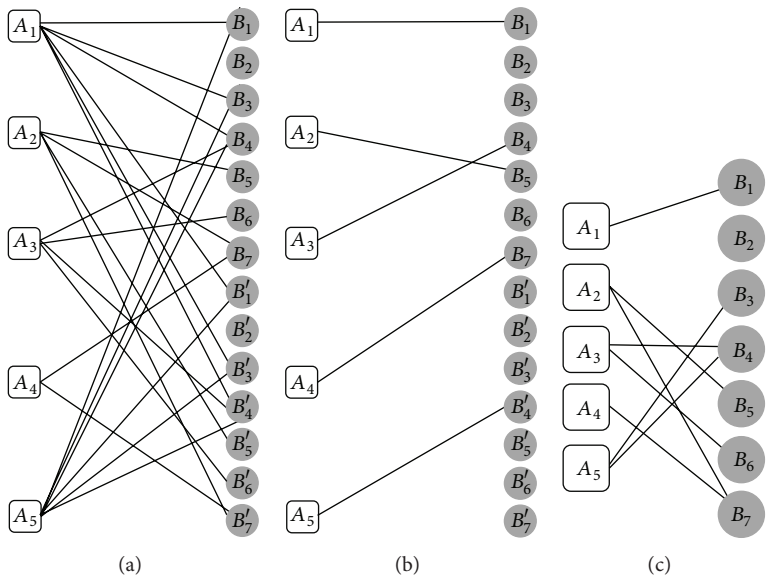


Figure 7: (a) The ISBG between networks A and B in Figure 1, with duplication of second part. (b) Graph M_2, obtained from second part duplicated ISBG by performing Hungarian algorithm. (c) The *Final Similarity Bipartite Graph* (sim_G) for networks A and B.

Bipartite Graph Sim_G (obtained by Algorithm 2) as input and matches all pairs of adjacent vertices where degree of both of them is one. For other vertices, Algorithm 3 creates all possible *Dmatchs* and selects them in a greedy manner. Algorithm 3 utilizes the following steps for all vertices of graph Sim_G.

(1) If the degree of the vertex v is one and degree of adjacent vertex is also one, these two vertices are assigned to each other.

(2) If the degree of the vertex v is two and v is adjacent to v'_i and v'_j, the path $P(v'_i, v, v'_j)$ is created as a potential *Dmatch*.

(3) If the degree of the vertex v is three and v is adjacent to v'_i, v'_j, and v'_k, potential *Dmatchs* $P(v'_i, v, v'_j)$, $P(v'_i, v, v'_k)$, and $P(v'_j, v, v'_k)$ are created.

The weight of a given *DMatch* can be calculated by the following formula:

$$W\left(P\left(v'_i, v, v'_j\right)\right) = W\left(v_i, v\right) + W\left(v, v_j\right). \quad (8)$$

After creating all potential *DMatchs*, we sort them based on their weights; then in a successive rounds we choose the highest weight *DMatchs* $P(v'_i, v, v'_j)$ and delete all other edges connected to v'_i, v, or v'_j. By this way all assignments are found and finally the local alignment of two graphs is obtained in Steps 24–27 of Algorithm 3. Therefore the output of this algorithm is the final alignment between two given graphs G_1 and G_2. The final alignment between networks A and B in Figure 1 is shown in Figure 8.

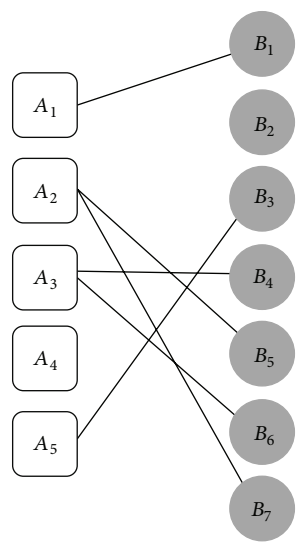


Figure 8: Final alignment between networks A and B.

3. Results

In this section, we intend to compare our algorithm to other well-known local alignment algorithms (NetworkBLAST [9], MaWISh [10], Graemlin [11], and Graemlin 2.0 [13]), in a way that Graemlin 2.0 compares its results, because it is the only method that compares results of all local alignment tools against KEGG Ortholog (KO) groups [22] on different data sets. The data sets contain the PPI networks data from SNDB [21] for the organism pairs (*E. coli*, *S. typhimurium*) and (*E. coli*, *C. crescentus*), PPI networks data from DIP [20] for the organism pair (*human, mouse*), and data from IntAct [19] database for the organism pair (*yeast, fly*). The comparison is proceeded based on specificity and sensitivity in terms

Table 1: Results of *Pin-Align* algorithm on four pairs of PPI networks, using different percentages in hub clustering step (Step 1).

	Percentage	C_{eq}	C_{node}	C_{or}	Tot
(*E. coli*, *S. typhimurium*)	60%	0.95	0.93	1651	1779
	70%	0.95	0.94	1723	1835
	80%	0.95	0.94	1773	1890
	90%	0.95	0.94	1782	1905
(*E. coli*, *C. crescentus*)	60%	0.87	0.84	666	794
	70%	0.89	0.86	665	772
	80%	0.88	0.85	693	811
	90%	0.89	0.87	764	876
(*Human*, *mouse*)	60%	0.78	0.72	217	299
	70%	0.82	0.78	248	316
	80%	0.78	0.73	226	309
	90%	0.82	0.78	258	329
(*Yeast*, *fly*)	60%	0.87	0.85	258	303
	70%	0.89	0.87	280	323
	80%	0.91	0.88	350	396
	90%	0.91	0.89	416	466

Table 2: PPI networks from SNDB for the organism pairs (*E. coli*, *S. typhimurium*) and (*E. coli*, *C. crescentus*).

	(*E. coli*, *S. typhimurium*)				(*E. coli*, *C. crescentus*)			
	C_{eq}	C_{node}	C_{or}	Tot	C_{eq}	C_{node}	C_{or}	Tot
NB	0.77	0.49	457	1016	0.78	0.50	346	697
Gr2.0	0.95	0.94	627	667	0.79	0.78	447	573
MW	0.84	0.64	1309	2050	0.77	0.54	458	841
Gr2.0	0.97	0.96	1611	1678	0.77	0.76	553	727
Gr	0.80	0.77	985	1286	0.69	0.64	546	847
Gr2.0	0.96	0.95	1157	1217	0.82	0.81	608	750
Pin-Align	**0.95**	**0.94**	**1782**	**1905**	**0.89**	**0.87**	**764**	**876**

of KO groups introduced in Graemlin 2.0 [13]. In short—as Graemlin 2.0 defined—to uniquely specify an alignment, the mapping should be transitive; that is, if protein A is aligned with proteins B and C, then protein B must also be aligned with protein C. Mathematically, this means that the mapping is an equivalency relation, so groups of aligned proteins are referred to as equivalence classes. An equivalent class is defined as correct if all protein members in the class are in the same KO group. To measure specificity, Graemlin 2.0 computed two metrics:

(1) the fraction of equivalence classes that were correct (C_{eq}),

(2) the fraction of nodes that were in correct equivalence classes (C_{node}),

and to measure sensitivity, it computed two metrics:

(1) the total number of nodes that were in correct equivalence classes (C_{or}),

(2) the number of equivalence classes that contained 2 species (Tot).

As we described in Step 1 of the *Pin-Align* algorithm, *Pin-Align* chooses some vertices as hubs whose degrees are greater than about 95% of total vertices. For networks *S. typhimurium, C. crescentus, human,* and *yeast*, *Pin-Align* chooses vertices as hubs in a way that their degrees are greater than 264, 126, 16, and 23, respectively. The reason why these degrees are greater than 95 percent of total vertices degrees is clear from cumulative frequency of their vertices degrees which is shown in charts of Figure 9. For example, in the PPI network of *S. typhimurium* the degrees of about 95% of vertices are less than 264, so in this network we choose vertices with degrees greater than 264 as hubs.

Table 1 shows *Pin-Align* results with different hub degrees. Column two of this table (percentage column) contains different percentages which are used for hub clustering step (Step 1).

The results of the algorithms are demonstrated in Tables 2 and 3. In these tables NB stands for NetworkBlast, MW for MaWISh, Gr for Graemlin, and Gr 2.0 for Graemlin 2.0. They show that *Pin-Align* is the most specific and sensitive aligner in comparison with other alignment tools like NetworkBlast, MaWISh, Graemlin, and Graemlin 2.0. Because Graemlin 2.0

Table 3: PPI networks from DIP and IntAct for the organism pairs (*human*, *mouse*) and (*yeast*, *fly*).

	(*Human*, *mouse*)				(*Yeast*, *fly*)			
	C_{eq}	C_{node}	C_{or}	Tot	C_{eq}	C_{node}	C_{or}	Tot
NB	0.33	0.06	65	1010	0.39	0.14	43	306
Gr2.0	0.83	0.81	228	281	0.58	0.58	155	267
MW	0.59	0.36	87	241	0.45	0.37	10	27
Gr2.0	0.88	0.86	181	210	0.90	0.91	20	30
Gr	0.59	0.53	108	203	0.33	0.29	35	122
Gr2.0	0.86	0.84	151	179	0.61	0.61	57	93
Pin-Align	**0.82**	**0.78**	**258**	**329**	**0.91**	**0.89**	**416**	**466**

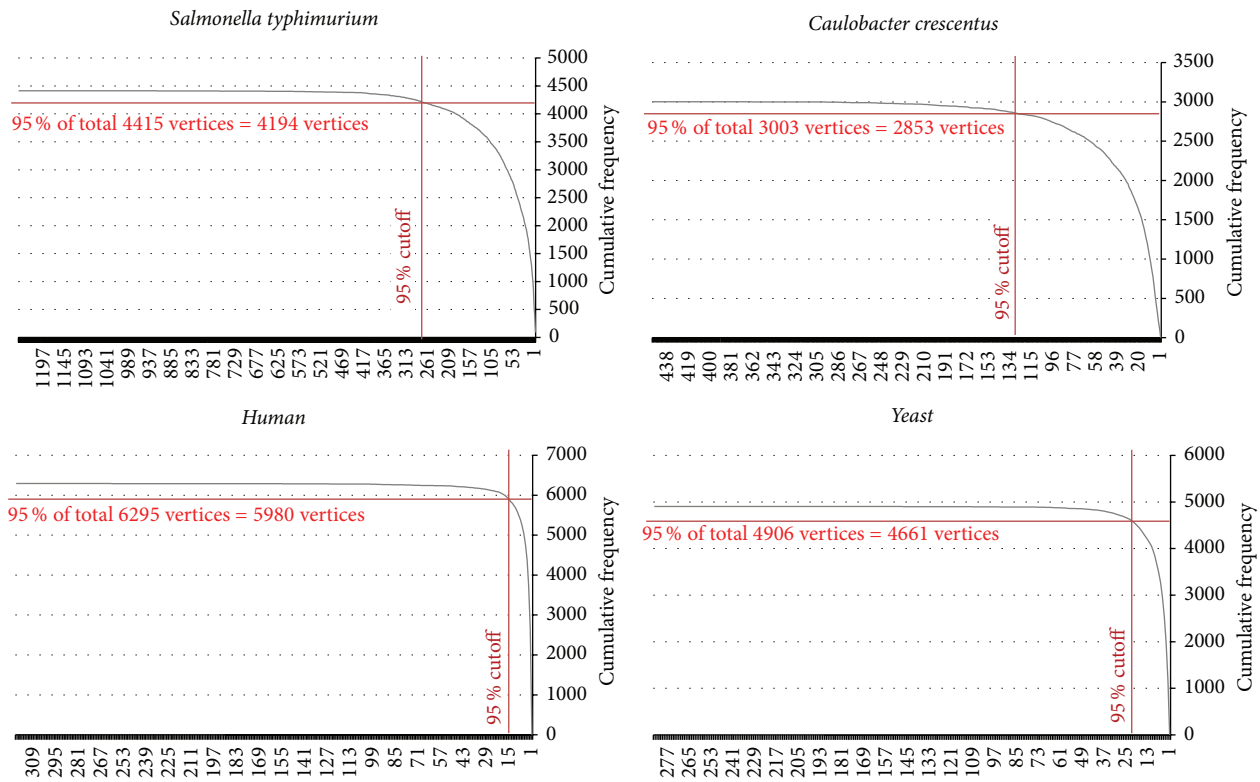


Figure 9: Cumulative frequency of vertices degrees in networks of *S. typhimurium*, *C. crescentus*, *human*, and *yeast*.

for each of its comparisons to other local aligner removed equivalence classes of its output that did not have a node in common with output of another local aligner, we have different results of Graemlin 2.0. Graemlin 2.0 did this due to considerations of local aligners to nodes in conserved modules and global aligners to all nodes.

In Table 2, we use the PPI network data from SNDB for the organism pairs (*E. coli*, *S. typhimurium*) and (*E. coli*, *C. crescentus*). In Table 3 we use the PPI networks data from DIP and IntAct for the organism pairs (*human*, *mouse*) and (*yeast*, *fly*).

Figure 10 shows that *Pin-Align* also finds more correct equivalence classes than NetworkBlast, MaWISh, Graemlin, and Graemlin 2.0. By considering all the above results and comparing *Pin-Align* to the other algorithms, it is evident that *Pin-Align* is more accurate in most cases based on the C_{eq}, C_{node}, C_{or}, and Tot measures.

4. Conclusions

In this paper, we presented *Pin-Align*, a pairwise local network alignment to improve accuracy of the alignment. *Pin-Align* algorithm is mainly designed based on the dynamic programming approach and has specificity and sensitivity comparable with existing tools such as NetworkBlast, Graemlin, Graemlin 2.0, and MaWISh. Our novel scoring system is based on bridges which are considerable edges due to the

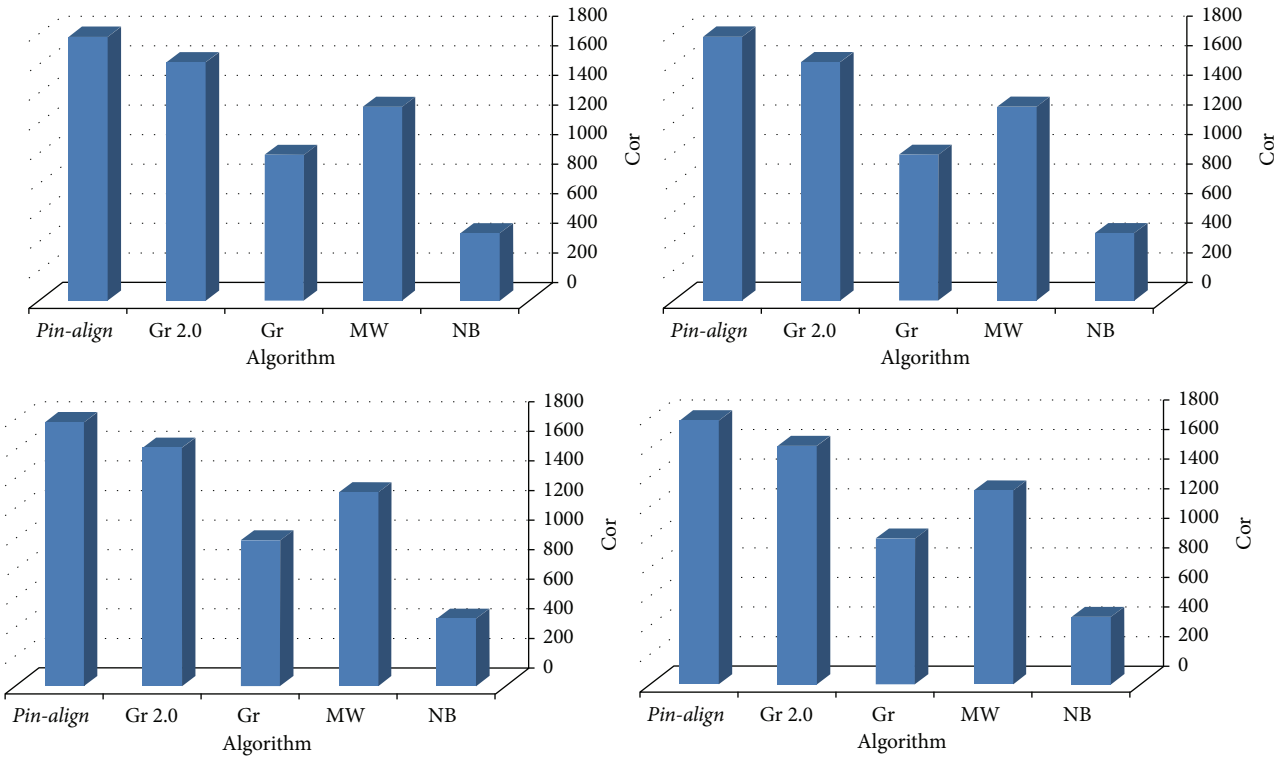


Figure 10: The number of correct equivalence classes that are found in alignment of organism pairs (*E. coli*, *S. typhimurium*), (*E. coli*, *C. crescentus*), (*human*, *mouse*), and (*yeast*, *fly*) by *Pin-Align*, in comparison with Graemlin 2.0, Graemlin, MaWISh, and NetworkBlast.

existence of hubs and small words in biological networks which certainly contain bridges. Bridges take part in most of the paths and consequently conserved paths in the networks. In future work, we plan to extend this approach to multiple network alignment.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank Mehdi Sadeghi for valuable comments and suggestions and Graemlin team [13] from Stanford University for sharing the datasets and helpful communication. The authors would like to acknowledge the financial support of University of Tehran for this research under Grant no. 6151184/01/04.

References

[1] 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*, vol. 7, no. 50, pp. 1341–1354, 2010.

[2] Y. Wang, T. Hindemitt, and K. F. X. Mayer, "Significant sequence similarities in promoters and precursors of *Arabidopsis thaliana* non-conserved microRNAs," *Bioinformatics*, vol. 22, no. 21, pp. 2585–2589, 2006.

[3] P.-A. He, X.-F. Li, J.-L. Yang, and J. Wang, "A novel descriptor for protein similarity analysis," *MATCH Communications in Mathematical and in Computer Chemistry*, vol. 65, no. 2, pp. 445–458, 2011.

[4] Y. Zhang and W. Chen, "A new measure for similarity searching in DNA sequences," *MATCH Communications in Mathematical and in Computer Chemistry*, vol. 65, no. 2, pp. 477–488, 2011.

[5] T. Milenković and N. Pržulj, "Uncovering biological network function via graphlet degree signatures," *Cancer Informatics*, vol. 6, pp. 257–273, 2008.

[6] T. Milenković, W. L. Ng, W. Hayes, and N. Pržulj, "Optimal network alignment with graphlet degree vectors," *Cancer Informatics*, vol. 9, pp. 121–137, 2010.

[7] B. Neyshabur, A. Khadem, S. Hashemifar, and S. S. Arab, "NETAL: a new graph-based method for global alignment of protein-protein interaction networks," *Bioinformatics*, vol. 29, no. 13, pp. 1654–1662, 2013.

[8] 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*, vol. 32, pp. 83–88, 2004.

[9] R. Sharan, S. Suthram, R. M. Kelley et al., "Conserved patterns of protein interaction in multiple species," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 6, pp. 1974–1979, 2005.

[10] M. Koyuturk, Y. Kim, U. Topkara, S. Subramaniam, W. Szpankowski, and A. Grama, "Pairwise alignment of protein

interaction networks," *Journal of Computational Biology*, vol. 13, no. 2, pp. 182–199, 2006.

[11] J. Flannick, A. Novak, B. S. Srinivasan, H. H. McAdams, and S. Batzoglou, "Græmlin: general and robust alignment of multiple large interaction networks," *Genome Research*, vol. 16, no. 9, pp. 1169–1181, 2006.

[12] R. Singh, J. Xu, and B. Berger, "Global alignment of multiple protein interaction networks," in *Proceedings of the 13th Pacific Symposium on Biocomputing (PSB '08)*, pp. 303–314, Kohala Coast, Hawaii, USA, January 2008.

[13] J. Flannick, A. Novak, C. B. Do, B. S. Srinivasan, and S. Batzoglou, "Automatic parameter learning for multiple local network alignment," *Journal of Computational Biology*, vol. 16, no. 8, pp. 1001–1022, 2009.

[14] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger, "IsoRankN: spectral methods for global alignment of multiple protein networks," *Bioinformatics*, vol. 25, no. 12, pp. i253–i258, 2009.

[15] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using PageRank vectors," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06)*, pp. 475–486, IEEE Computer Society, Los Alamitos, Calif, USA, October 2006.

[16] W. Tian and N. F. Samatova, "Pairwise alignment of interaction networks by fast identification of maximal conserved patterns," *Pacific Symposium on Biocomputing*, vol. 14, pp. 99–110, 2009.

[17] W. Tian and N. F. Samatova, "Global alignment of pairwise protein interaction networks for maximal common conserved patterns," *International Journal of Genomics*, vol. 2013, Article ID 670623, 11 pages, 2013.

[18] S. F. Altschul, T. L. Madden, A. A. Schäffer et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[19] S. Kerrien, Y. Alam-Faruque, B. Aranda et al., "IntAct-Open source resource for molecular interaction data," *Nucleic Acids Research*, vol. 35, no. 1, pp. D561–D565, 2007.

[20] I. Xenarios, Ł. Salwínski, X. J. Duan, P. Higney, S.-M. Kim, and D. Eisenberg, "DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions," *Nucleic Acids Research*, vol. 30, no. 1, pp. 303–305, 2002.

[21] B. S. Srinivasan, A. F. Novak, J. A. Flannick, S. Batzoglou, and H. H. McAdams, "Integrated protein interaction networks for 11 microbes," in *Research in Computational Molecular Biology*, vol. 3909 of *Lecture Notes in Computer Science*, pp. 1–14, 2006.

[22] M. Kanehisa, S. Goto, Y. Sato, M. Kawashima, M. Furumichi, and M. Tanabe, "Data, information, knowledge and principle: back to metabolism in KEGG," *Nucleic Acids Research*, vol. 42, no. 1, pp. D199–D205, 2014.

[23] R. Singh, J. Xu, and B. Berger, "Pairwise global alignment of protein interaction networks by matching neighborhood topology," in *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology (RECOMB '07)*, pp. 16–31, April 2007.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and S. Clifford, *Introduction to Algorithms*, MIT Press, Cambridge, UK, 3rd edition, 2009.

[25] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.

[26] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: a tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[27] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955.