Parallel Implementation of GRAph Aligner (GRAAL) Algorithm for Network Alignment

Si Li, Shengai Jin, Jonathan Z. Sun, Chaoyang Zhang

School of Computing, University of Southern Mississippi, Hattiesburg, MS, 39406, USA

Abstract - Network alignment is one of the most commonly used biological network comparison methods since determining protein functions shifted the focus from targeting specific proteins based solely on sequence homology to analyses of the whole proteome based on protein-protein interactions (PPI). Aligning PPI networks of different species is of great importance when detecting evolutionary conserved pathways, or protein complexes. However, when it comes to large biological network data, the improved serial algorithms still take a long time. In this paper, a parallel algorithm for network alignment, which is based on the serial implementation of the GRAph Aligner Algorithm (GRAAL), is designed to improve the efficiency of network alignment. This algorithm is implemented in parallel with C++ and the Message Passing Interface (MPI) library. The results show that the parallel implementation of GRAph Aligner improves significantly in efficiency without losing accuracy, compared to the serial GRAAligner algorithm.

Keywords: GRAAL, network alignment, PPI, parallel implementation

1 Introduction

1.1 Network alignment

Network alignment is considered to be one of the most common methods to analyze and compare biological networks. It is mainly about finding structure or topology similarities between two or more networks. Similar to sequence alignments, network alignments have two main instances: local network alignment and global network alignment. Based on the hypothesis that aligned sub-graphs are conserved through evolution, the goal of local alignment is to search for evolutionary conserved building blocks of the cellular machinery, disregarding the overall similarity between networks. A global network alignment gives a unique and one-to-one alignment from every node in a smaller network to exactly one node in the other network. Hence, the goal of global network alignment is to search the maximal overall match between two or more networks.

1.2 PPI datasets

PPI networks are usually obtained by two high-throughput experimental bio-techniques. They are yeast two-hybrid screening, resulting in binary interaction data and protein complex purification methods using mass-spectrometry, resulting in co-complex data. Many databases containing PPI networks are also available online. These include Biological General Repository for Interaction Datasets (BioGRID), IntAct, Database of Interacting Proteins (DIP), Mammalian Protein-Protein Interaction Database (MIPS), and many others. The datasets that we use in this thesis are mainly from BioGRID and DIP databases.

1.3 Background

Recently, large amounts of experimental biological network data are becoming available due to the advanced techniques used in the biological field. These biological networks include protein-protein interaction (PPI) networks, transcriptional-regulation networks, brain functional networks, and metabolic networks. We mainly focus on analyzing PPI networks, which are probably the most commonly studied type of biological networks. In PPI networks, nodes represent proteins and edges among nodes stand for the interactions between proteins. It is generally represented as an undirected graph with no self-loops. PPI networks are of particular importance because proteins play a crucial role in all cell functions. Instead of acting in isolation, proteins always cooperate with other proteins to perform many biological functions and create large complicated networks. We learn protein-protein interaction (PPI) networks and apply network alignment as the biological network comparison method to focus on the analysis of the entire proteome. Comparative analyzing of PPI networks can give valuable insight into biological mechanisms, evolutionary changes, and provide deeper understanding of complex diseases. Exploring the inner
relationship of PPI network topology and biological functions are a big challenge given the scale of biological network data. Therefore, an efficient algorithm is essential to be implemented.

The GRAph Aligner Algorithm (GRAAL), which is a global network alignment method, works well in network alignment. However, when it comes to large scale biological network data, the computational costs are increased significantly. In this work, we develop a parallel implementation based on the GRAph Aligner Algorithm (GRAAL).

In this paper, we first give a brief introduction to the serial GRAAL algorithm in Section 2. Then, the parallel GRAAL algorithm including design and implementation are presented in Section 3. In the Section 4, we introduce the implementation environment. Section 5 analyzes the performance of the parallel algorithm. Finally, we conclude the work in section 6.

2 Serial GRAAL algorithm

Kuchaiev et al. recently proposed a topological method of the global alignment of biological networks based on graphlet degree signatures, GRAph Aligner (GRAAL) [1]. Since this method does not use protein information, it can be used to align any two networks, not just biological ones. It mainly contains four steps given as follows:

Step 1: Compute the vector of graphlet degrees of a node and signature similarities, which provide a novel method to measure the local topology in a node’s vicinity and similarity between nodes from two networks (this vector is a matrix, also called a signature, that describes the node’s neighborhood topology).

\[ D_j(u,v) = \frac{\sum \log(u_i+1) - \log(v_i+1)}{\log(\max\{u_i,v_i\}+2)} \]  

(1)

Step 2: Compute the cost matrix of aligning each node of the first network with each node in the second network.

\[ C(u,v) = 2 - ((1 - \alpha) \times \frac{\deg(u)\deg(v)}{\max(\deg(G)) + \max(\deg(H))} + \alpha \times S(u,v)) \]  

(2)

Step 3: Choose a pair of nodes (u, v) from the two networks respectively as an initial seed with minimal cost, then build “spheres” of all possible radii around nodes u and v. Spheres of the same radius in the two networks are then greedily aligned.

Step 4: Calculate the edge correctness (EC) which is the percentage of edges in the first graph that are aligned to edges in the second graph.

\[ EC = \frac{|\{(u,v)\in E_1 | (f(u),f(v))\in E_2\}|}{|E_1|} \times 100\% \]  

(3)

3 Parallel GRAAL algorithm

Based on the analysis of the four steps of the serial algorithm introduced in the Section 2, we see that there are two main parts in this algorithm that can be implemented in parallel. The first part is the vector matrix calculation which describes the node’s neighborhood topology. The second part is the cost matrix calculation which is the foundation to choose the initial seeds of two networks. Considering cost matrix calculation is just a formula based on previous result, it costs little time which we can skip. So we are focus on the parallel implementation of vector matrix calculation. The Implementation in [1] uses 73 different orbits across all graphlets of size 2 to 5. The vector of 73 coordinates is the signature of a node that describes the topology of its neighborhood and captures its interconnectivities.

Therefore, the vector of 73 coordinates must be calculated for each node in the network. Note that the calculation of the vector matrix of a large network is a time consuming task. In order to reduce the running time and improve the performance, we can calculate the vector matrix and cost matrix using parallel computing technology. In order to do matrix calculations, we implement a block-row decomposition technology to partition both the data and the computational operations.

3.1 Data decomposition

The vector calculation contains a matrix-vector multiplication operation. So, we use a block-row decomposition technology to partition data into equal-size sub-matrix, which are sent to each processor. Figure 1 shows the data decomposition.

![Figure 1: Data block-row decomposition](image-url)
3.2 Task decomposition

Step 1: We use one processor to read adjacency matrix from two network files.

Step 2: Scatter the adjacency matrix to each processor to compute local vector matrix.

Step 3: Gather those local vector matrix from each processor.

3.3 Parallel algorithm implementation

3.3.1 Degree matrix calculation

For each node in each network, its degree must be calculated. We use several processors to calculate the fixed number of many different nodes’ degrees simultaneously, and then combine these results to get the final degree matrix at a single destination process.

3.3.2 Vector matrix calculation

According to the number of graphlets we choose, there are related numbers of orbits among those graphlets. For every node in the two networks, we count the number of graphlets connecting to a node for all graphlets through different orbits. This is the node’s signature vector. For example, if there are 73 different orbits between 30 graphlets consisting of 2 to 5 nodes, then the signature vector of a node has 73 coordinates. Calculating the matrix of this vector can also be parallelized in the same way as in Step 1.

3.3.3 Distance matrix calculation

\( D_i(u, v) \) denotes the distance between the \( i^{th} \) orbits of nodes \( u \) and \( v \) from the two networks. \( D_i(u, v) \) for all node pairs from a matrix. The total distance \( D(u, v) \) between nodes \( u \) and \( v \) is calculated as

\[
D(u, v) = \frac{\sum_{i=1}^{73} D_i(u, v)}{\sum_{i=0}^{73} w_i}
\]  

(4)

and \( D(u, v) \) for all node pairs from a matrix, too. We can calculate both matrices in parallel.

3.3.4 Construction of spheres and alignment of spheres

Once the seed is found, GRAAL builds ‘spheres’ of all possible radii around nodes \( u \) and \( v \). Spheres of the same radius in two networks are then greedily aligned. These two tasks are independent on different nodes and can be easily parallelized.

4 Implementation Environment

The parallel algorithm was implemented using C++ and MPI. It was tested on the platform of Albacore Linux cluster in the School of Computing at the University of Southern Mississippi [8]. The cluster consists of 256 processor cores, 300GB of RAM and a 1-Gigabit Ethernet interconnects. Albacore is a hybrid, distributed-shared memory cluster, consisting primarily of Intel Xeon 56xx processors and Intel Xeon 55xx processors.

5 Results and Performance Analysis

The parallel GRAAL algorithm was compiled and executed on the Albacore cluster. Both the serial and the parallel GRAAL algorithms resulted in the same aligned networks, which verify the correctness of the parallel implementation.

We downloaded four datasets from the Database of Interacting Proteins (DIP) online database [9] and BioGRID websites [10], and created four synthetic networks. Then we aligned these synthetic networks using both serial and parallel codes to verify the results and analyze the performance of the parallel algorithm. The parallel code is executed with different numbers of processors, e.g. 2, 4, 8 and 16 in the Albacore Linux cluster.

We further tested the parallel code using three different datasets: the Fruit fly PPI network with 96 nodes from DIP database, the Human Herpes Virus PPI network with 152 nodes, and the Human Immunodeficiency Virus PPI network with 321 nodes from BioGRID website. As described in the Section 3, the vector matrix calculation is the most time consuming part in the parallel GRAAL algorithm, so we only analyze the performance of the vector matrix calculation in this paper. Table 1, Table 2, and Table 3 give the computation time \( T \) for calculating the vector matrix in the serial and parallel GRAAL algorithms for Fruit fly PPI network dataset, the Human Herpes Virus PPI network dataset and the Human Immunodeficiency Virus PPI network dataset with respect to different number of processors \( P \). Table 4 shows the corresponding speedup that is defined as the ratio of serial computation time to the parallel computation time (We denote the serial and parallel execution times as \( T_s \) and \( T_p \), speedup is defined as \( S = T_s / T_p \)). Table 5 shows the corresponding efficiency that is defined as the ratio of speedup to the number of processors (Efficiency is defined as \( E = S / p \), which \( S \) is the speedup and \( p \) is the number of processors). The reasons why the speedup and efficiency are chosen are that, for speedup, it delineates how much performance gain is achieved via a parallel design over serial design; and for efficiency, it describes how much time is spent on the computation. For convenience of performance analysis, the speedup and efficiency are shown in Figure 5 and 6.
From Table 1, through Table 5, P is the number of processors and T is the average execution time (in seconds). For Table 4 and Table 5, H.Immu is short for Human Immunodeficiency Virus PPI network, H.Herp is short for Human Herpes Virus PPI network, F.Fly is short for Fruit Fly PPI network.

Table 1: The execution times with different number of processors for the Fruit Fly dataset

<table>
<thead>
<tr>
<th>P</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1.41</td>
<td>0.75</td>
<td>0.39</td>
<td>0.21</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table 2: The execution times with different number of processors for the Human Herpes Virus dataset

<table>
<thead>
<tr>
<th>P</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>39.64</td>
<td>22.26</td>
<td>11.97</td>
<td>6.58</td>
<td>3.97</td>
</tr>
</tbody>
</table>

Table 3: The execution times with different number of processors for the Human Immunodeficiency Virus dataset

<table>
<thead>
<tr>
<th>P</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>219.69</td>
<td>126.53</td>
<td>77.52</td>
<td>50.63</td>
<td>33.29</td>
</tr>
</tbody>
</table>

Table 4: The speedup (S) of the three datasets

<table>
<thead>
<tr>
<th></th>
<th>P=1</th>
<th>P=2</th>
<th>P=4</th>
<th>P=8</th>
<th>P=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.Immu</td>
<td>1.00</td>
<td>1.74</td>
<td>2.83</td>
<td>4.34</td>
<td>6.60</td>
</tr>
<tr>
<td>H.Herp</td>
<td>1.00</td>
<td>1.78</td>
<td>3.31</td>
<td>6.02</td>
<td>9.98</td>
</tr>
<tr>
<td>F.Fly</td>
<td>1.00</td>
<td>1.88</td>
<td>3.62</td>
<td>6.71</td>
<td>11.75</td>
</tr>
</tbody>
</table>

Table 5: The efficiency (E) of the three datasets

<table>
<thead>
<tr>
<th></th>
<th>P=1</th>
<th>P=2</th>
<th>P=4</th>
<th>P=8</th>
<th>P=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.Immu</td>
<td>1.00</td>
<td>0.87</td>
<td>0.71</td>
<td>0.54</td>
<td>0.41</td>
</tr>
<tr>
<td>H.Herp</td>
<td>1.00</td>
<td>0.89</td>
<td>0.83</td>
<td>0.75</td>
<td>0.62</td>
</tr>
<tr>
<td>F.Fly</td>
<td>1.00</td>
<td>0.94</td>
<td>0.91</td>
<td>0.84</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Figure 2: Execution time for Fruit Fly dataset

Figure 3: Execution time for Human Herpes virus dataset

Figure 4: Execution time for Human Immunodeficiency virus dataset

Figure 5: The speedup with different processors on Albacore
The results show that the computation time decreases significantly and the speedup increases when the number of processors increases. It is noted that the speedup curve is below the ideal speedup denoted by the dashed line in the Figure 5 because the parallel GRAAL algorithm needs to complete extra tasks and communication between processors. Since most parallel tasks are independent in the parallel algorithm, the communication cost is not significant. In summary, the parallel implementation of the GRAAL algorithm can significantly improve the computational performance of network alignment.

6 Conclusion

A parallel implementation of vector matrix and cost matrix calculations in the GRAAL algorithm is presented, which uses a block-row decomposition technique. The adjacency matrices of two networks are distributed onto \( p \) processes, with each process solving a sub-problem. This preliminary study shows that the parallel GRAAL implementation can significantly improve the computational performance, which provides an efficient way for aligning large networks, especially biological networks.

7 Acknowledgement

This work was partially supported by the NSF EPSCoR project “Modeling and Simulation of Complex Systems” (NSF #EPS-0903787). School of Computing at the University of Southern Mississippi provided high performance computing facilities, and Dr. Glover George provided technical support.

8 References


