이학박사학위논문

여러 종 사이에서 올소로그 군집기법

Clustering Methods for Finding Orthologs among Multiple Species

충북대학교 대학원

전자계산학과 전자계산학전공

김 선 신

2007 년 8 월

이학박사학위논문

여러 종 사이에서 올소로그 군집기법

Clustering Methods for Finding Orthologs among Multiple Species

지 도 교 수　　이 충 세, 류 근 호

전자계산학과 전자계산학전공

김 선 신

이 논문을 이학박사학위 논문으로 제출함

2007 년 8 월

이 논문을 김선신의 이학박사학위 논문으로 인정함.

심 사 위 원 장 _____

심 사 부위원장 _____

심 사 위 원 _____

심 사 위 원 _____

심 사 위 원 _____

충 북 대 학 교 대 학 원

2007 년 8 월

Clustering Methods for Finding Orthologs among Multiple Species

Sunshin Kim

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science
Chungbuk National University
R.O.Korea

August 2007

Clustering Methods for Finding Orthologs among Multiple Species

Sunshin Kim

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Chungbuk National University
R.O.Korea

August 2007

Clustering Methods for Finding Orthologs among Multiple Species

Sunshin Kim

A dissertation submitted to the Department of Computer Science
and the committee on Graduate School
of Chungbuk National University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Chungbuk National University
Graduate School

This is to certify that I have examined this dissertation and have found that it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Principal Advisor:

_____ _____

Dr. Chung Sei Rhee Dr. Keun Ho Ryu

(School of Electrical & Computer Engineering)

Chair of Supervisory Committee:

_____

Dr. Keun Ho Ryu (School of Electrical & Computer Engineering)

Reading Committee:

_____

Dr. Yong Je Chung, External Examiner (School of Life Science)

_____

Dr. Seung Kee Han, External Examiner (Department of Physics)

_____

Dr. Jaewoo Kang, External Examiner (Korea University)

Table of Contents

List of Figures

List of Tables

List of Algorithms

Clustering Methods for Finding Orthologs among Multiple Species *

Sunshin Kim

Department of Computer Science

Gradudate School, Chungbuk National University

Cheongju, Korea

Supervised by Professors **Chung Sei Rhee and Keun Ho Ryu**

Abstract

Since the early 1980s, molecular sequence data have increased exponentially. However, the functions of most genes (proteins) are unknown. Besides, it takes much more time and efforts to know the functions of genes through biological experiments. Therefore, it is very significant in bioinformatics to predict the unknown functions of the other genes from the known functions of some genes by sequence homology. It is especially desired to cluster orthologs, the genes with preserved function from the common ancestor, for the more accurate annotation of the newly sequence genes.

In order to represent ortholog clusters systematically, we here define a new

* A dissertation for the degree of Doctor in August 2007

mathematical concept, named “setor”, which means a set of the protein sequences with their inherent similarity scores. We also introduce an iterative method that clusters orthologs (joins unit setor pairs with common unit setors) from multiple complete genomes. This algorithm extends InParanoid to cluster orthologs from multiple species. In addition, we introduce a parallel method to cluster orthologs from multiple genomes to improve performance of the iterative method.

On the other hand, the quality of Orthologous Protein Clusters (OPCs) is largely dependent on the results of the reciprocal BLAST hits among genomes. The BLAST algorithm is very efficient and fast, but it is very difficult to get optimal solution among distant phylogenetic species because the genomes with large evolutionary distance typically have low similarity in their protein sequences. In order to reduce the false positives in the OPCs, thresholding is often employed on the BLAST score. However, the thresholding also eliminates large numbers of true positives as the orthologs from distant species likely have low BLAST scores.

In order to solve this problem, we introduce a new automatic method combining the recursive search and MCL algorithms without the use of BLAST thresholding. Especially, we use Markov matrices with self-sequence similarity scores to improve accuracy to detect orthologs. First, we select genomes of our interest and prepare their protein sequence sets. Second, all possible pairwise orthologs are produced by using all possible best reciprocal BLAST hits among *n* genomes. Third, a score matrix is produced by running the recursive search algorithm. Fourth, the score matrix is transformed into the Markov matrix to simulate random walks on a graph. Finally, the Markov clustering is executed to produce the putative orthologs by adjusting the inflation factor. We tested our method using six different species and evaluated the results by comparing against KO OPCs, which are

generated from manually curated known pathways. We evaluated the performance of our method using InParanoid as the baseline approach. We showed that our method produced 54% more numbers of orthologs than InParanoid while keeping the accuracy similar to that of InParanoid. We also introduced a new intuitive similarity measure based on our least-move algorithm for quantifying the similarity between the generated OPCs.

Table for Abbreviations

Symbol	Description
BLAST	Basic Local Alignment Search Tool
BLOSUM	BLOck SUbstitution Matrix
COGs	The database of Clusters of Orthologous Groups of proteins
DNA	DeoxyriboNucleic Acid
FASTA	FAST Alignment
GOLD	Genome OnLine Database
HSP	High-scoring Segment Pair
KO	Kegg Orthology
MCL	Markov CLuster
mRNA	Messenger RiboNucleic Acid
NCBI	National Center for Biotechnology Information
OPCs	Orthologous Protein Clusters
PAM	Point Accepted Mutation
PDB	Protein Data Bank
PSI-BLAST	Position Specific Iterative BLAST
RNA	RiboNucleic Acid

Chapter 1

Introduction

Since the early 1980s, molecular sequence data have increased exponentially. However, the functions of most genes (proteins) have been still unknown. In addition, it takes much more time and efforts to know the functions of genes through biological experiments. Therefore, it is very important in computational biology to predict the unknown functions of the other genes from the known functions of some genes.

We focus on the clustering of orthologous genes from multiple genomes. We first suggest a mathematical framework to describe ortholog clusters systematically. Next, we suggest the iterative and the parallel algorithms to cluster orthologs. We also introduce a new automatic method combining the recursive search and Markov clustering algorithms, named “ReMark”.

This chapter starts with the concepts of genomes and proteomes. We introduce basic problems and concept of orthologs on which this study is focused. We also introduce a

typical approach briefly in contrast with our methods. Finally, we discuss why our methods were proposed.

In the first stage of this dissertation, we introduce the background of our research. In the next stage, our work will be described and discussed.

1.1 From Genomes to Proteomes

The original life on the earth may evolve and be distinguished into two kinds of life, which are prokaryotes and eukaryotes. Prokaryotes do not contain a cell nucleus but eukaryotes do it. Prokaryotes have taxonomically two domains of bacteria and archaea. Eukaryotes consist of animals, plants, fungi, and protests.

The genome of an organism in biology is its whole hereditary information, which is encoded in the DNA (or, for some viruses, RNA). The genome consists of a set of genes, which comprise binary sets of exons and introns. Prokaryotic genes lack introns.

The central dogma of molecular biology represents the transfer of sequence information as shown in Figure 1.1. First, DNA can be replicated into DNA (DNA replication). Second, DNA information can be copied into mRNA (transcription). Finally, using the information in mRNA, proteins can be synthesized (translation).

The proteome is the protein products of genome in a given organism. A cellular proteome is the set of proteins found in a cell. A complete proteome can be used as a complete set of proteins that can be expressed by the genetic material of a cell.

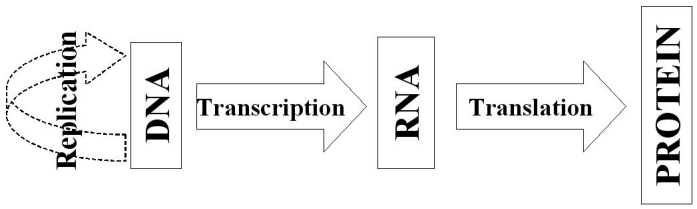


Figure 1.1 The central dogma represents the transfer of sequence information. After the replication of DNA, the information is transcribed into RNA, and proteins are synthesized.

1.2 Explosion of Sequenced Proteins

In the post-sequence era, molecular sequence data have increased exponentially. Until now, about 500 complete genomes are known in GOLD(Genome OnLine Database). In addition, the GenBank sequence database receives sequences produced in laboratories throughout the world from more than 100,000 distinct organisms. Over 61 million sequences have been included in it. On the orther hand, the PDB database contains over 43,000 protein structures. The unknown function of a new protein can be predicted by comparing its sequence or structural homology with the other homology of known function. However, the number of structures is largely limited while the function prediction sometimes can give more exact result by structural homology. Therefore, the functional annotations of most genes are necessarily based on the sequence homology.

1.3 Homology

It is necessary to understand that homology and sequence similarity are completely different. Similarity can arise as a result of convergence without a common evolutionary history. On the other hand, homology is known as descent from a common ancestor. In some cases, homologous genes do not have sequence similarity at all. That is, homologs can evolve so diversely that it is difficult or even impossible to detect similarities. In addition, not all similarities are due to homology [Reec87, Hill94].

There are two types of homologs. One is orthologs that are homologous genes, which have diverged preserving the same function from each other after speciation events. The other is paralogs that are homologous genes, which have different functions each other after gene duplication events. Now, consider an example of Figure 1.2. There is a gene of a genome at the beginning. Later, speciation occurs and evolves. In the next stage, there are two species. One species remains without any event, whereas the other undergoes gene duplication and speciation. During the evolution with time, the function of the original gene is kept without changing, but that of the copied gene is changed into a different function. Therefore, the orthologs with same function are made among three genes (A, B, and D), and between two genes (C and E). The paralogs with different function each other can be made as A-C, A-E, B-E, and C-D.

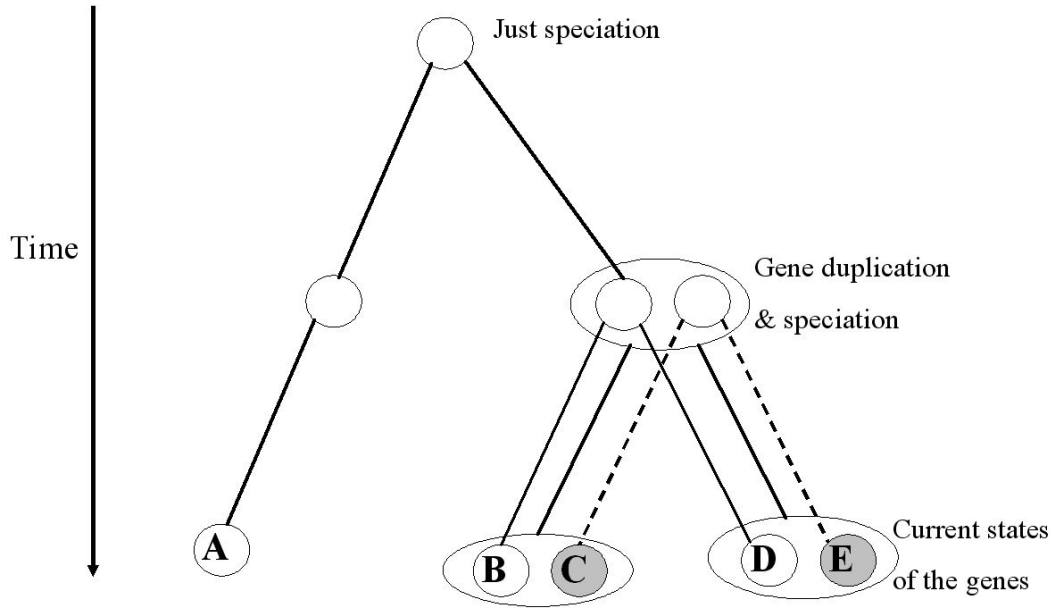


Figure 1.2 The evolutionary process with time. When considering a gene (white circle) of a genome at the beginning time, later speciation occurs and evolves. Next, while one remains without any event, the other undergoes gene duplication and speciation. During the evolution, the function of the original gene remains without changing, but that of the copied gene is changed into a different function (gray circle). As the result, the orthologs with same function are made among three genes (A, B, and D), and between two genes (C and E). The paralogs with different function each other can be made as A-C, A-E, B-E, and C-D.

There are two kinds of methodology for predicting gene function based on sequence homology. One is phylogenomic methods. The other is pairwise similarity-based methods. The phylogenomic method requires many more steps and much more manual work than the latter, but it usually produces more accurate results. In contrast with the former, the pairwise similarity-based methods are very fast and can be automated easily but have less

accuracy. This dissertation is focused on the fast and automatic similarity-based approach of pairwise sequences.

A typical method in phylogenomics is briefly introduced in this section. This method uses evolutionary techniques to improve functional predictions [Eise98]. It is based on reconstructing the evolutionary history of genes to help predict the functions of uncharacterized genes since gene functions change as evolutionary results. It has three steps roughly. First, a phylogenetic tree is generated. The tree represents the evolutionary history of the gene of interest and its homologs. Second, the known functions of the various homologs are overlaid onto the tree. Finally, this method uses the tree structure and the relative phylogenetic positions of genes of different functions, which trace the history of functional changes. The history is used to predict functions of uncharacterized genes.

First of all, identification of homologs is achieved through database searches. In addition, it is benefitial to use more data from database searches to get more accurate result. PSI-BLAST (Position Specific Iterative BLAST) [Alts97] can be useful for automatic iterated searches. The PSI-BLAST produces a position specific scoring matrix from a multiple alignment of the highest scoring hits in an initial BLAST search. The matrix is generated by calculating position-specific scores for each position in the alignment. Highly conserved positions receive high scores and weakly conserved positions receive very low scores. This iterative searching strategy increases sensitivity.

Next, the purpose of sequence alignment is the assignment of positional homology. It is assumed that each column in a multiple sequence alignment contains amino acids which have a common evolutionary history. Each column is treated separately in the phylogenetic analysis. It is, therefore, necessary to exclude ambiguous regions of the assignment of

positional homology [Gate93]. The exclusion of certain alignment positions (masking) can give phylogenetic methods the accurate reflection of the evolution of the genes.

For generating phylogenetic trees from sequence alignments, three methods are commonly used such as parsimony, distance, and maximum likelihood [Swof96]. First, the parsimony method starts with comparing possible trees, each of which is given a score that is a reflection of the minimum number of character state changes. The changes would be required over evolutionary time to fit the sequences into that tree. The tree with the fewest changes is considered to be optimal. Second, the distance method starts with calculating the estimatied evolutionary distance between all pairs of sequences. These distances are then used to generate a tree representing the distance matrix best in its branch patterns and lengths. Finally, maximum likelihood is similar to parsimony methods since possible trees are compared and given a score. If a model of amino acid substitution probabilities are given, the score is based on how likely the given sequences are to have evolved in a particular tree. If the tree has the highest probability, it is considered to be optimal. It is important to check the robustness and accuracy of the phylogenetic patterns before using any specific tree. It is also useful to determine a root for the tree.

Overlaying any known functions onto the tree are required to make functional predictions based on the phylogenetic tree. First, the tree can be used to identify probable gene duplication in the past. The sharing of the genes is made into groups of orthologs and paralogs by this. Uncharacterized genes can be assigned a probable function if the function of any ortholog is known. Second, to predict the probable functions of uncharacterized genes, parsimony reconstruction techniques can be used by identifying the evolutionary history that requires the fewest functional changes over time.

1.4 Clustering Orthologs from Multiple Species

The prediction of protein function is very important in bioinformatics. Especially, it is challengeable to fast and accurately predict gene function based on sequence homology in this explosion age in molecular sequences. For achieving this goal, clustering methods to find orthologs, which are based on sequence similarity, have been proposed in the past decade. The ortholog clusters produced by these methods can be used for functional annotation of newly sequenced genomes.

The first exploration to cluster orthologs by homology started with seven complete genomes [Tatu97]. The database of Clusters of Orthologous Groups of proteins (COGs) is their next work including from 21 to 30 complete genomes [Tatu00, 01]. However, it takes much time and manual work since a case-by-case analysis is used. In addition to that, COGs include many paralogs in some clusters and so it is difficult to predict accurately the function of the unknown gene.

To address these problems, Remm [Remm01] introduced InParanoid, a fully automatic program, to detect orthologous proteins between two species. MultiParanoid [Alex06] published recently has extended InParnoid to cluster proteins from multiple species. On the other hand, Li et al. [Li03] developed OrthoMCL which generates the clusters of orthologs from multiple species using Markov CLuster (MCL) algorithm [Dong00]. Their method shows similar results to the InParanoid. However, both InParanoid and OrthoMCL use thresholds to improve the accuracy of results. So, the large number of true positives is eliminated.

From this motivation, our work focuses on how to cluster orthologs fast and accurately without losing orthologs. We here suggest a mathematical frame for the systemactical

representation of ortholog clusters. We also introduce the iterative and the parallel methods that are automatic and fast for clustering orthologs from multiple complete genomes. On the other hand, we introduce a new automatic method combining the recursive search and MCL algorithms without the use of thresholds. In order to improve accuracy to detect orthologs, we especially use Markov matrices with self-sequence similarity scores.

In the next two chapters, we will review the basic methods for pairwise comparison and pairwise similarity-based methods. The clustering methods that we proposed are introduced in chapter 4. In chapter 5, our experimental results will be introduced in detail. We conclude with summary of results and suggest future work in chapter 6.

Chapter 2

Basic Methods for Sequence Similarity

In this chapter, we introduce the basic methods, which are the bases of pairwise similarity approach. First of all, we introduce the sequence alignment of DNA or proteins [Yona99]. Next, the dynamic programming approach is described and heuristic algorithms [Yona99] are represented. To carry out pairwise comparison of massive sequences, the two main algorithms, FASTA [Pear88] and BLAST [Alts90], were suggested. Although these heuristic algorithms do not give the optimal alignment, they proved to be very effective for sequence comparison and significantly faster than the rigorous dynamic programming algorithm. Finally, we introduce statistical methods for sequence similarity in the last section.

2.1 Sequence Alignment

A sequence alignment is a way of arranging the DNA or protein sequences, the purpose of which is to identify the conserved regions of similarity that may be a result of evolutionary events among the sequences. There are two kinds of alignments. One is the global alignment. The other is the local alignment.

Two protein sequences are given A = $a_1a_2\ldots a_n$ and B = $b_1b_2\ldots b_m$ where a_i, $b_j \in \wp$, the alphabet of amino acids. A global alignment of these sequences is an alignment where all letters of A and B are considered. We consider that $s(a_i, b_j)$ is the similarity of a_i, b_j and $W > 0$ is the penalty for deleting of one amino acid. The alignment score of M_{ij} matches with a_i and b_j , and G gap deletions is defined as in equation (2.1).

$$\sum_{i,j} M_{ij} \bullet s(a_i, b_j) - G \bullet W. \qquad (2.1)$$

The global similariy of sequences A and B is defined as the largest score of any alignment of sequences A and B,

$$S(\mathrm{A},\mathrm{B}) = \max\{\sum_{i,j} M_{ij} \bullet s(a_i, b_j) - G \bullet W\}. \qquad (2.2)$$

On the other hand, the similarity of two sequences may often be limited to a specific motif or domain. That is, the specific region may contain functional features while the other part may not be related. In these cases, local alignment approach can be effective and fast. A local alignment of A and B is defined as an alignment between a substring of A and a substring of B. The local similarity of sequences A and B is defined as the maximal score over all possible local alignments.

2.2 Smith-Waterman Algorithm

The Smith-Waterman algorithm [Smit81], a dynamic programming approach, performs local sequence alignment for determining reserved and similar regions between two nucleotide or protein sequences. To find pairs of segments with high degrees of similarity, a matrix D is made as the following

$$D_{k0} = D_{0l} = 0 \quad \text{for } 0 \le k \le n \text{ and } 0 \le l \le m. \tag{2.3}$$

M_{ij} is the maximum similarity of two segments ending in a_i and b_j. The values of D are obtained from the following relationship

$$D_{ij} = \max\{D_{i-1,j-1} + s(a_i, b_j), \max_{k \ge 1}\{D_{i-k,j} - W_k\}, \max_{l \ge 1}\{D_{i,j-1} - W_l\}, 0\},$$

$$\text{where } 1 \le i \le n \text{ and } 1 \le j \le m. \tag{2.4}$$

The formula for D_{ij} follows by considering the possibilities of ending the segments at any a_i and b_j.

- If a_i and b_j are joined, the similarity is $D_{i-1,j-1} + s(a_i, b_j)$.
- If a_i is at the end of a deletion of length k, the similarity is $D_{i-k,j} - W_k$.
- If b_j is at the end of a deletion of length l, the similarity is $D_{i-k,j} - W_l$.
- A zero is included to avoid negative similarity indicating no similarity up to a_i and b_j.

2.3 FASTA

This algorithm has two stages in producing a score for pairwise similarity. At the first stage, this algorithm achieves fast performance by using a hash table of all k-tuples in the query

sequence. The *k*-tuple value determines how many successive identities are required in a match. The 10 best diagonal regions are detected using a simple formula based on the number of *k*-tuple matches. Next, these 10 regions are rescanned using a scoring matrix. Each *k*-tuple of the library sequence is searched in the hash table. The appearance of the *k*-tuple in the hash-table makes the offset (the relative displacement of the *k*-tuple) calculated. The offset vector increases at the index corresponding to the offset value. After the library sequence has been scanned, if the diagonal corresponds to the offset with the maximal number of occurrences, it can serve as a seed for the dynamic programming method.

At the second stage, the rescanning is done for the 10 regions which have the highest density of identities. If common *k*-tuples are on the same diagonal and are sufficiently close to an existing run, they are added to the run to form a region. The regions are scored to account for not only the matches but the mismatches. The best region is reported. The nearby high scoring regions are joined even if not on the same diagonal. At last, a dynamic programming is run around the best region to get the optimal score.

2.4 BLAST

This algorithm is based on human intuition as well as mathematical statistics. The reason that the ability of humans is related can be described as the following instance [Kane03]. If we were to compare two sequences by the eye, we would search common patterns shared by two sequences and try to extend these to obtain longer matches rather than examine all possible alignments.

BLAST compares two sequences and seeks all pairs of similar segments with the

similarity score exceeding a threshold. These pairs of segments are called HSPs (High Scoring Segment Pairs). The segment pair is ungapped and it is a pair of segments of the same length. The matching score is to summarize the matches of the amino acides along the segmen pair. The segment pair with the highest score is called MSP (Maximum Segment Pair).

The algorithm is a result drawn from the statistical theory for local alignments without gaps. From the theory, we can get a framework to evaluate the probablility of the similarity between two protein sequences. If the chance is very low, the similarity is statistically significant. The similarity is reported along with its statistical significance.

The BLAST algorithm has three steps:

- Search for exact matches of words of length W, scoring at least T, between the query and sequences in the database. For example, given the sequences GSVEDT and ALVEDN and a word length $W = 3$, BLAST would identify the matching substring VED that shares both sequences.
- Try to extend the match of words that score T or greater in both directions, starting at the seed. The ungapped alignment process extends the initial seed match of length W in each direction in an attempt to find high-scoring ungapped alignment with a score of at least S or and E value lower than the specific threshold. If a high-scoring ungapped alignment is found, the next step takes over the database sequence.
- Perform a gapped alignment between the query sequence and the database sequence using a variation of the Smith-Waterman algorithm. Statistically significant alignments are reported.

2.5 Statistics of Sequence Similarity Scores

In this section, we introduce how to get the scores from sequence similarity statistically [Alts]. The random distribution of optimal global alignment scores is almost unknown under even the simplest random models and scoring systems. The statistics for the scores of local alignments are here introduced. Statistics for the scores of local alignments are well understood. This is especially true for local alignments without gaps. We consider such alignments sought by the original BLAST database search programs. An ungapped local alignment consists of a pair of equal length segments. A modified verstion of the Smith-Waterman algorithm will detect all segment pairs whose scores can not be improved by extension. Those are High-scoring Segment Pairs (HSPs).

We are considering optimal local sequence alignments based on the idea that the maximum of a large number of independent identically distributed random variables tends to an extreme value distribution [Gumb58]. When lengths m and n are sufficiently large, two parameters, κ and λ, characterize the statistics of HSP scores. As desribed simply, the expected number of HSPs with score at least S is given by the following formula

$$E = \kappa mn \quad e^{-\lambda S}. \tag{2.5}$$

This is called the E-value for the score S. E is expected to decrease exponentially with score. The factors, κ and λ, can be considered as natural scales for the search space size and the scoring system respectively.

Raw scores have little meaning like citing a distance without the units, that is, feet, meters. So, by normalizing a raw score using the following formula

$$S' = \frac{\lambda S - \ln \kappa}{\ln 2}, \qquad (2.6)$$

a bit score, *S'*, is obtained. It comprises a standard set of units. The *E*-value corresponding to a given bit score is simply

$$E = mn \quad 2^{-S'}. \qquad (2.7)$$

Bit scores include the statistical essence of the scoring system. So, the size of the search space is necessary to be known to calculate significance.

A Poisson distribution [Karl90, Demb94] describes the number of random HSPs with the score, ζ, equal and larger than *S*. The probability of finding exactly a HSPs with the ζ is given by

$$e^{-E} \frac{E^a}{a!} \qquad (2.8)$$

where *E* is the *E*-value of *S* given by equation (2.5).

In a special case, the chance of finding zero HSPs with the ζ is e^{-E}. Therefore, the probability of finding at least one such HSP is

$$P = 1 - e^{-E} \qquad (2.9)$$

This is the *P*-value associated with the score *S*.

When comparing protein sequences, scoring matrices are used to score for a match and a mismatch. The products of a local alignment program depend strongly on the scores it uses. There is no single scoring scheme for satisfying all purposes. It can improve the sensitivity of one's sequence analysis to understand the basic theory of local alignment scores. The theory for scores, which are used to find ungapped local alignments, is well studied.

A large number of different amino acid substitution scores have been represented based

on a variety of rationales [Heni92, Jone92, Over92]. However, the scores of any substitution matrix, which has negative expected scores, can be written only in the following equation.

$$S_{ij} = (\ln \frac{q_{ij}}{p_i p_j}) \Big/ \lambda \qquad (2.10)$$

In the above equation, the q_{ij}, target frequencies, are positive numbers that sum to 1 and the p_i are background frequencies for the various residues. The positive constant [Karl90, Alts91], λ, is identical to that of equation (2.5).

It does not change the essence of scores to multiply all the ones in a substitution matrix by positive constant. That is, an alignment that was optimal using the original scores remains optimal. Such multiplication changes the factor λ, but it does not change the target frequencies q_{ij}. Therefore, only the target frequencies determine every substitution matrix.

Dayhoff [Dayh78, Schw78] suggested formula (2.10). They made the PAM(Point Accepted Mutation) model of molecular evolution from a study of observed residue replacements in closely related proteins. One PAM corresponds to an average change in 1% of all amino acid positions. This means that not every residue will have changed after 100 PAMs of evolution. That is, some residues will possibly have undergone several mutations, but others will not have mutated at all. At the same time, some will perhaps have returned to their original states. In geneal, the correspondence between PAM distance and evolutionary time is not identified since different protein families evolve at different rates.

Dayhoff's method for calculating target frequencies has been pointed out [Wilb85]. Since her work [Gonn92, Jone92], there have been several efforts to improve her numbers using the massive quantities of derived protein sequence data. Unfortunately, the new PAM

matrices are not improved greatly.

Henikoff [Heni92] has suggested an alternative approach to estimating target frequencies. They examine multiple alignments of distantly related protein regions rather than extrapolate from closely related sequences. An advantage of this approach is to be based on observation. A disadvantage is not to yield any evolutionary model. Some experimental results [Pear95, Heni93] suggest that the BLOSUM matrices produced by this method are usually better than the PAM matrices for detecting biological relationships.

Chapter 3

Pairwise Similarity-based Methods

Clustering is a technique to extract patterns from clusters (groups). Therefore, the patterns of a cluster have their inherent feature. The samples in cluster analysis are unlabled. The clustering, an instance of unsupervised learning, does not assume any type and structure of the model as opposed to supervised learning, where the samples are labled. The supervised learning designs the functional form of the classifier, which assigns each sample to its correct class.

When the data are given as pairwise similarities, pairwise clustering algorithms are very useful, and then we introduce the linkage-based clustering algorithms [Yona99] in this chapter. We specially introduce three typical methods using the reciprocal BLAST best hits. These methods describe the construction of ortholog clusters from completely sequenced genomes.

3.1 Linkage-based Algorithm

Pairwise clustering algorithms can be called graph-based clustering algorithms. The set of a cluster is represented as a weighted directed graph $G = (V, E)$, where the set V are the nodes of the graph and E denotes edges between every pair of nodes. The weight of an edge $w(i, j)$ is a function of the similarity between the nodes i and j. In the pairwise clustering algorithms, there are three types of the linkage approach, which are the single, complete, and average linkage algorithms.

First, the single linkage algorithm uses the distance measure, which is defined as the least of all of the possible pairwise distances between the clusters. If data points are represented as nodes of a graph, the nearest pair of nodes i and j are merged. The similarity of the corresponding pair of nodes is represented as weight $w(i, j)$. This method is effective when the clusters are compact and separated well. The algorithm is, however, very sensitive to noise in the distances between nodes.

Second, the complete linkage algorithm determines the distance measure between two clusters by the most distant pair of nodes, which are opposite to that of the sinlge linkage algorithm. Therefore, if the distance of a pair of points are larger than the other among all other points in the resulting cluster, the pair are merged. In this procedure, all nodes within the cluster are connected by edges. This algorithm is also effective for compact clusters, but if the true clusters are loose, the resulting clusters may be meaningless.

Finally, above two linkage algorithms are very sensitive to noise and outliers. The average linkage method uses the average distance between points of the two clusters, which is a compromise between the two distance measures represented above. As a result, this algorithm is likely to produce more stable clusterings.

3.2 COG Approach

Tatusov (1997) made the COGs bring together the fields of comparative genomes and protein classification. They produced 720 OPCs (Orthologous Protein Clusters) from seven complete genomes. The clusters are made through the reciprocal BLAST best hits among all possible genome pairs. One-third of the total clusters contain one representative of each species. 192 more OPCs include paralogs from only one species. The total clusters include 6814 proteins, which are 37% of the total number of genes in the seven complete genomes. They made it possible to annotate the functions of genes through the COG system.

As their following work, the COG database [Tatu00] was constructed from 21 complete genomes of bacteria, archaea and eukaryotes. The database consists of 2091 OPCs. The total OPCs include 56-83% of the gene products from each of the complete bacterial and archaeal genomes, and about 35% of those from the yeast *Saccharomyces cerevisiae* genome. To fit new proteins into the COGs, the COGNITOR program was developed. The updated COG database [Tatu01] was introduced. The new version comprises 2791 COGs containing 45,350 proteins from 30 genomes of bacteria, archaea and the yeast. In addition, the COGNITOR program was improved with subsequent manual validation.

The basic idea is that any cluster of at least three proteins with reciprocal BLAST best hits are most inclined to belong to an ortholog group. This method tried to cluster orthologs from distant species. Figure 3.1 shows the steps of COG construction [Tatu00]. First, the interesting genomes are selected. Second, perform two-way BLAST best hits and find the orthologs. Third, detect apparent paralogs and remove them. Fourth, detect trigangles, each

of which means a ortholog pair. Fifth, merge triangles with a common side. Finally, make COGs through a case-by-case analysis, which serves to remove false positives and to identify clusters that contain multidomain proteins by investigating the pictorial outputs. The sequences of detected multidomain proteins are split into single-domain segments. Then, steps 1-5 are repeated with these sequences. In addition, some of these groups are split into two or more smaller ones through examination of large COGs.

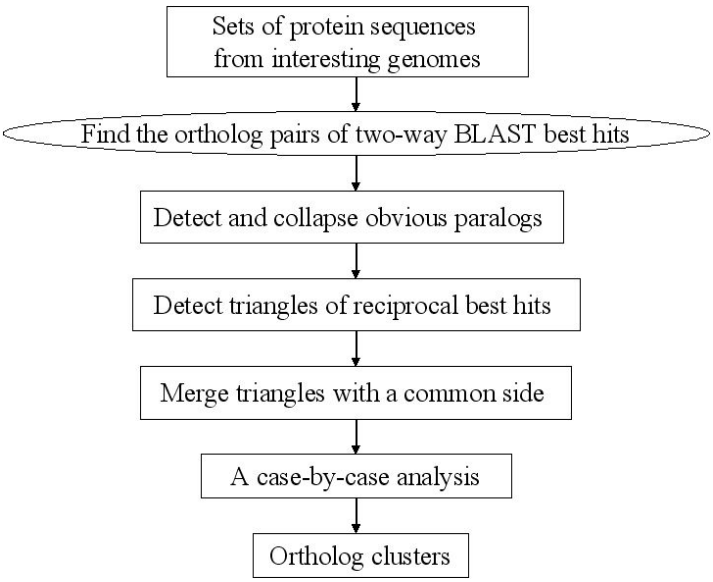


Figure 3.1 The overall steps of the COGs method.

3.3 InParanoid and MultiParanoid

InParanoid [Remm01] explored the challengeable field of clustering orthologs from three points of view as the following description. First, it mostly focuses on eukaryotes unlike the COGs. Second, without time-consuming processes, they introduced the fully automatic program detecting orthologous proteins between only two genomes. Finally, their clusters included inparalogs which are the recent paralogs sharing the same function. For the evaluation of the InParanoid results, OPCs were made from worm and mammalian transmembrane proteins, and were compared to the curated set of OPCs taken by phylogenetic methods. Their result shows 84% true positives, 9% false positives, and 3% false negatives.

On the other hand, two types of user adjustable thresholds are applied to each pairwise match. One is a score cut-off, which is used to separate significant scores from counterfeit matches. They typically use a score cut-off of 50 bits. The cut-off is taken to prevent inclusion of insignificant hits. The other is a overlap cut-off, which is used to prevent shot, domain-level matches. Orthologous sequences are expected to keep the homology over the majority of their length. Therefore, they use an overlap cut-off of 50%. That is, the matching segment of the longer sequence is made to be kept above 50% of its total length. However, owing to this thresholding, many genuine orthologs can not be detected.

Figure 3.2 shows the basic steps to cluster orthlogs and inparalogs between two species. First, it executes all possible reciprocal BLAST best hits between two genomes, including self-reciprocal best hits. The next step is to decide a main protein pair of *a* and *b*, fixed as a center point, from which additional inparalogs are clustered. Figure 3.3 shows clustering of additional orthologs. Both big circles represents species A and B. Each small circle

represents a protein sequence from each species. Main orthologs are represented as *a* and *b*. *S* denotes their similarity score. The score should be considered as reverse distance between *a* and *b*. For clustering of inparalogs, they assume that the inparalogs from the same species are more similar to the main ortholog than to any sequence from orther species.

Recently, Alexeyenko et al. suggested MultiParanoid [Alex06], which extended InParanoid to cluster orthologs from multiple species. This method uses the products from InParanoid, which are multiple ortholog tables for three genomes of human, worm, and fly.

Now let's consider clustering orthologs from three species S1, S2 and S3. From InParanoid, the tables with pairwise ortholog clusters (S1-S2, S2-S3, and S1-S3) are taken. First, this algorithm starts by reading the list of clusters from the table of S1-S2. The clusters in S1-S2 table act as seeds to include sequences in the other proteomes. Second, the method detects the seed orthologs of S1-S2 cluster in the S1-S3 and S2-S3 tables. Third, if detected, the seed cluster includes all the members (inparalogs) in corresponding S1-S3 or S2-S3 clusters. Until all pairwise ortholog clusters are processed, it is rerun. This clustering approach corresponds to a single linkage method.

They used the manually curated reference dataset to estimate the quality of MultiParanoid. Their results are compared to ortholog groups in OrthoMCL, which showed that MultiParanoid produce fewer out-paralogs than these groups.

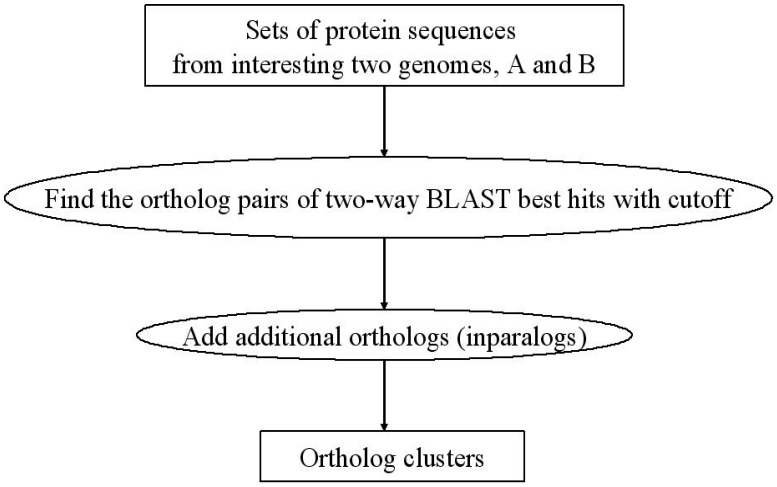


Figure 3.2 The overall steps of the InParanoid method.

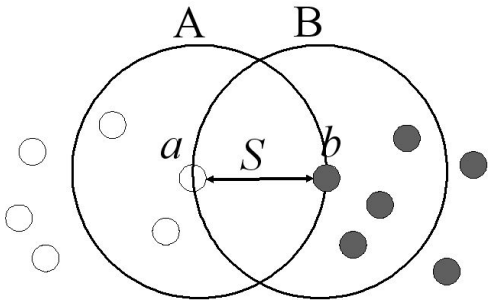


Figure 3.3 A diagram for clustering additional orthologs. Both big circles represent species, A and B. Each small circle represents a protein sequence from each species. Main orthologs are represented as *a* and *b*. *S* denotes their similarity score. The score should be considered as reverse distance between *a* and *b*. For clustering of inparalogs, they assume that the inparalogs from the same species are more similar to the main ortholog than to any sequence from orther species.

3.4 OrthoMCL

Li et al. (2003) developed OrthoMCL which generates OPCs from multiple species using MCL algorithm [Dong00]. Their study focuses on an approach to identify automatic eukaryotic orthologs. They used the proteome data set from seven genomes (human, fly, worm, yeast, *Arabidopsis*, *Plasmodium falciparum*, and *Escherichia coli*). This approach is similar to InParanoid but can be extended to cluster orthologs from multiple genomes. This method shows very similar performance to the InParanoid as compared with their result using the concept of coherence, which means that the members in a group generated by one method are a subset of members in a group generated by the other. However, it has some problems to compare two groups using the degree of coherence. We will discuss about it in detail in section 5.1.

Chen et al. (2006) constructed the OrthoMCL database, which houses ortholog cluster predictions for 55 species. The database includes 16 bacterial and 4 archaeal genomes, and 24 unikonts(12 animals, 9fungi, microsporidium, *Dictypostelium*, *Entamoeba*), 4 plants/algae and 7 apicomplexan parasites. A number of 511,797 proteins (81.6% of the total dataset) were grouped into 70,388 ortholog clusters. They made the OrthoMCL-DB web interface that gives a convenient way to search for sequences and their corresponding ortholog clusters, which are based on protein accession number or text keywords. The interface also provides a BLAST-based sequence similarity search function, which allows users to find their interesting sequences or identify homologs that have been grouped into ortholog clusters.

The overall steps of OrthoMCL are shown in Figure 3.4. First, select the interesting genomes. Second, find the orthologs and inparalogs of two-way BLAST best hits. Third,

generate similarity matrix. Finally, apply MCL (Markov CLustering algorithm) [Dong00] to the matrix. For applying MCL, the construction of matrices is required. Figure 3.5 shows an example of sequence relationships and similarity matrix construction. Dotted arrows represent inparalogs (recent paralogs). Solid arrows represent orthologs. The lower left contains normalized weights to minimize the impact of inparalogs on the clustering of cross-species orthologs.

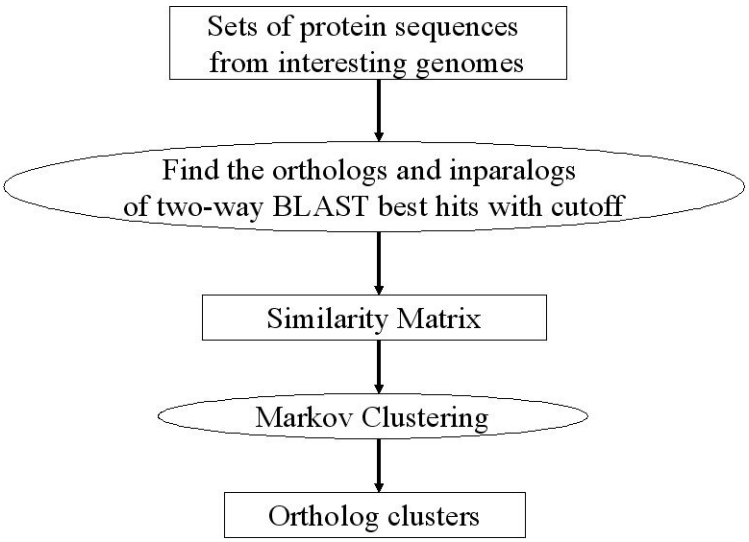
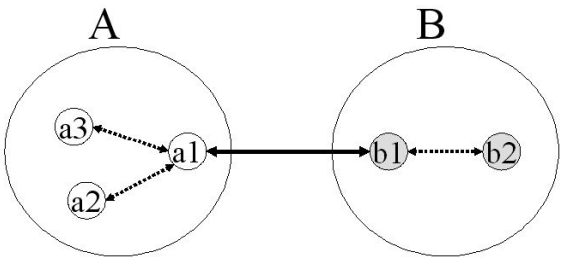


Figure 3.4 The overall steps of the OthoMCL method.



	a1	a2	a3	b1	b2
a1					
a2	233				
a3	118	117			
b1	69	68	45		
b2	33	33	34	100	

Figure 3.5 The diagram of orthologs and their similarity matrix. Dotted arrows represent inparalogs (recent paralogs). Solid arrows represent orthologs. The lower left contains normalized weights to minimize the impact of inparalogs on the clustering of cross-species orthologs.

The MCL algorithm is based on probability and graph flow theory. MCL simulates random walks on a graph using Markov matrices to determine the transition probabilities among nodes of the graph. Therefore, the random walk in the graph that visits a dense cluster will likely stay in the cluster until many of its nodes have been visited. The main idea [Dong00] of MCL algorithm is like simulating flow within a graph, which means that the flow is encouraged where the current is strong and discouraged where the current is

weak. There are two operators, inflation and expansion, working alternately. The expansion operator is responsible for allowing flow to connect different regions of the graph. That is, the flow expansion is represented by the usual matrix product. The inflation operator is responsible for both strengthening and weakening of current. If the both operators are sufficiently iterated close to an idempotent matrix, the process stops. Especially, the power coefficients of the inflation operator influence the granularity of the resulting partition. In addition, the node with a high return weight will likely be an attractor that has positive return probability. This has less effect than the inflation factor for increasing cluster granularity, but needs to be considered as an important feature according to the characteristics of data.

The Markov matrix is, in general, represented as in equation (3.1). Each element in the matrix represents the probability of random walk in the graph.

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{pmatrix} \tag{3.1}$$

The *j*th column of above matrix is represented as equation (3.2). The *j*th node in a graph has the transition rates with each of *n* nodes. The total sum of all the transition rates in the *j*th column is one. Now we can represent the *i*th node of the *j*th column as in equation (3.3).

$$\vec{M}_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ a_{3j} \\ \vdots \\ a_{nj} \end{pmatrix} \tag{3.2}$$

$$M_{ij} = a_{ij} \tag{3.3}$$

The expansion operator is simply represented by multiplying matrix as the following.

$$Ep(M) = M^2 \tag{3.4}$$

The inflation operator is described by the following equation. If the transition rate is high, this operation makes it higher, while lower if the transition rate is low.

$$Inf(M_{ij}) = \frac{(M_{ij})^r}{\sum_{i=1}^{n}(M_{ij})^r} \tag{3.5}$$

The idempotent matrix has the characteristics represented in equation (3.6). In this case, the expansion operation does not make the matrix changed. This means that the members of each cluster are kept without alteration.

$$M = M^2 \tag{3.6}$$

The MCL algorithm starts with the Markov matrix. Until the matrix is idempotent, both operations of expansion and inflation are calculated alternately as shown in Algorithm 3.1.

Algorithm 3.1 MCL algorithm with inflation and expansion operators

```
Procedure Markov_Clustering ( )
Input : Markov Matrix
Output : Idempotent Matrix
1.  for i = 1  to  ∞  do
2.      calculate Expansion
3.      calculate Inflation
4.      if  matrix is idempotent  then
5.          break
6.      end if
7.  end for
```

Chapter 4

Our Clustering Algorithms

We propose automatic methods that cluster orthologs from multiple species in this chapter. We first suggest a mathematical framework and define the concept of “setor” to describe ortholog clusters systematically. Next, we introduce three kinds of our work for the construction of OPCs. Our first method, the iterative algorithm, extends InParanoid to cluster orthologs from multiple complete genomes. The second method, the parallel algorithm, is to improve the time complexity of the first method. Finally, we introduce the ReMark method that employs the recursive search and the MCL algorithms. This final method was designed to produce much more orthologs (with keeping better or similar accuracy) than the existing methods.

4.1 Mathematical Methods

We introduce a new mathematical concept, named "setor", to describe ortholog clusters clearly and simply. A cluster can usually be represented as a set or a graph. However, the frames of those do not give the concept of self-similarity of a node. Therefore, we define the concept of the setor to use sets and self-similarities of nodes together.

If we suppose that each gene (protein) of a genome has fundamentally the unique and inherent self-sequence similarity score, $\widehat{G}_i$ can be defined as a genome setor with d_i different genes and self-sequence similarity scores as in equation 4.1. Another genome setor is represented as equation 4.2. The jth gene sequence of the ith genome is represented as g_{ij}. Then, the self-sequence similarity score of a gene, g_{ij}, is outputted by the function, f_{BLAST}, which means the reciprocal BLAST best hits as in equation 4.3. The sequence similarity score between g_{ij} of $\widehat{G}_i$ and g_{kj} of $\widehat{G}_k$ is also similarly described as in equation 4.4. The jth setor of the ith genome setor is represented as a unit setor in equation 4.5. As shown in Figure 4.1, the unit setor, $\widehat{g}_{ij}$, is the gene sequence with its inherent self-sequence similarity score produced by the reciprocal BLAST best hits.

$$\widehat{G}_i = (\widehat{g}_{i1}, \widehat{g}_{i2}, \widehat{g}_{i3}, \ldots, \widehat{g}_{ij}, \ldots, \widehat{g}_{id_i}) \tag{4.1}$$

$$\widehat{G}_k = (\widehat{g}_{k1}, \widehat{g}_{k2}, \widehat{g}_{k3}, \ldots, \widehat{g}_{kj}, \ldots, \widehat{g}_{kd_i}) \tag{4.2}$$

$$s_{(ij,ij)} = f_{BLAST}(g_{ij}, g_{ij}) \tag{4.3}$$

$$s_{(ij,kj)} = f_{BLAST}(g_{ij}, g_{kj}) \tag{4.4}$$

$$\widehat{g}_{ij} = (g_{ij}, s_{(ij,ij)}) \tag{4.5}$$

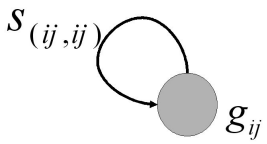


Figure 4.1 A gene of a unit setor $\hat{g}_{ij}$, which contains a sequence g_{ij} with a self-sequence similarity score $s_{(ij,ij)}$.

Now we define the joining operator, ⊗, to cluster putative orthologs between two setors in equation 4.6. We can get a pairwise setor, $\hat{G}_{ik}$ between two genome setors, $\hat{G}_i$ and $\hat{G}_k$. Let's consider an example of Figure 4.2. A putative unit setor pair, $\hat{g}_{i1}\hat{g}_{k1}$, is produced by the reciprocal BLAST best hits between two setors, $\hat{G}_i$ and $\hat{G}_k$. The putative setor pair is represented as shown in Figure 4.3. The $\hat{g}_{i1}\hat{g}_{k1}$ is represented as $\hat{g}_{i1}\hat{g}_{k1} = (g_{i1}g_{k1}, s_{(i1,i1)}s_{(k1,k1)}s_{(i1,k1)})$, where g_{i1} and g_{k1} are sequences, s_{i1} and s_{k1} are self-sequence similarity scores, and $s_{(i1,k1)}$ is the sequence similarity score between two sequences of g_{i1} and g_{k1}.

$$\hat{G}_i \otimes \hat{G}_k = \hat{G}_{ik} \tag{4.6}$$

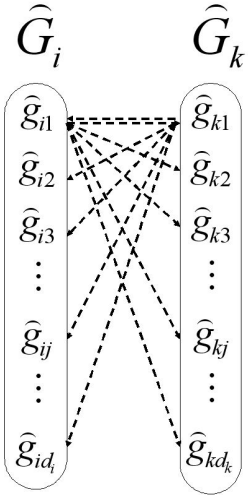


Figure 4.2 An instance of a putative setor pair $\widehat{g}_{i1}\widehat{g}_{k1}$ produced by the reciprocal BLAST best hits.

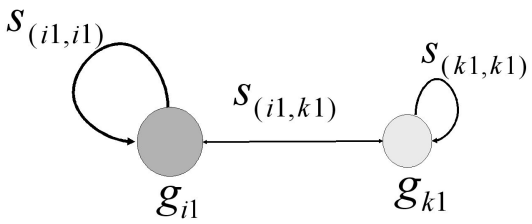


Figure 4.3 A diagram of a putative setor pair produced in Figure 4.2, which is represented as $\widehat{g}_{i1}\widehat{g}_{k1} = (g_{i1}g_{k1}, s_{(i1,i1)}s_{(k1,k1)}s_{(i1,k1)})$ where g_{i1} and g_{k1} are sequences, $s_{(i1,i1)}$ and $s_{(k1,k1)}$ are self-sequence similarity scores, and $s_{(i1,k1)}$ is a sequence similarity score between two sequences of g_{i1} and g_{k1}.

We also consider the joining operations among three genome setors as shown in Figure 4.4 to help understand clustering orthologs among multiple genomes. We can practically identify orthologs by the reciprocal BLAST best hits between two genomes. Figure 4.4 shows, for instance, three setors with reciprocal best hits among three genomes. We can get three joining pairwise setors as equation 4.7 through 4.9 among $\hat{G}_1$, $\hat{G}_2$, and $\hat{G}_3$. A total setor is clustered through common genes among three setors (equation 4.7-4.9) as equation 4.10.

$$\hat{G}_{12} = (\hat{g}_{11}\hat{g}_{22}, \hat{g}_{12}\hat{g}_{21}, \hat{g}_{13}\hat{g}_{23}, \hat{g}_{1d_1}\hat{g}_{2d_2}) \quad (4.7)$$

$$\hat{G}_{13} = (\hat{g}_{12}\hat{g}_{31}, \hat{g}_{1d_1}\hat{g}_{3d_3}) \quad (4.8)$$

$$\hat{G}_{23} = (\hat{g}_{21}\hat{g}_{31}, \hat{g}_{23}\hat{g}_{3d_3}) \quad (4.9)$$

$$\hat{\Gamma} = (\hat{g}_{11}\hat{g}_{22}, \hat{g}_{12}\hat{g}_{21}\hat{g}_{31}, \hat{g}_{13}\hat{g}_{23}\hat{g}_{3d_3}, \hat{g}_{1d_1}\hat{g}_{2d_2}\hat{g}_{3d_3}) \quad (4.10)$$

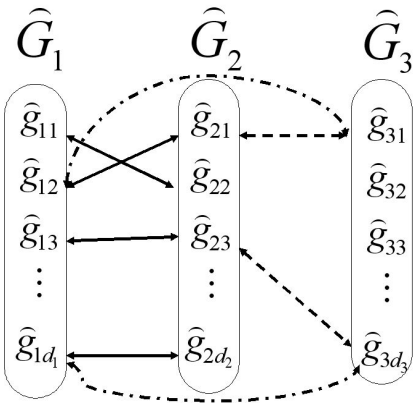


Figure 4.4 An example of the joining operations among three genome setors. The solid lines represent the result (clustering orthologs) of the joining operations between the two genome setors $\hat{G}_1$ and $\hat{G}_2$, the dashed lines between $\hat{G}_2$ and $\hat{G}_3$, and the dot-dashed lines between $\hat{G}_1$ and $\hat{G}_3$.

$$
\begin{array}{ccccc}
\widehat{G}_{12} & & & & \\
\widehat{G}_{13} & \widehat{G}_{23} & & & \\
\widehat{G}_{14} & \widehat{G}_{24} & \widehat{G}_{34} & & \\
\vdots & \vdots & \vdots & \ddots & \\
\widehat{G}_{1n} & \widehat{G}_{2n} & \widehat{G}_{3n} & \cdots & \widehat{G}_{n-1n}
\end{array}
$$

Figure 4.5 All possible $n(n-1)/2$ pairwise genome setors from n genome setors.

In general, we can get $n(n-1)/2$ setors with putative orthologs, as shown in Figure 4.5, between two genome setors when considering n complete genomes. We also define the joining operator, $\Im$, to cluster putative orthologs from multiple setors in equation 4.11. The quality of the total setor $\widehat{\Gamma}$ depends on the pairwise setor $\widehat{G}_{ij}$, which can not get exact results since it uses pairwise comparisons by heuristic BLAST algorithm.

$$\Im_{i=1}^{n-1}\Im_{j=2}^{n}\widehat{G}_{ij} = \widehat{\Gamma} \quad where \quad i < j. \tag{4.11}$$

4.2 Iterative Algorithm

We here introduce an automatic method that cluster orthologs from multiple complete genomes. This algorithm extends InParanoid to cluster orthologs from multiple species. It starts with the products of the InParanoid, which are $n(n-1)/2$ tables of orthologs from n genomes. This method comprises two stages roughly. The first stage is to detect and save all possible putative pairwise setors with the reciprocal BLAST best hits from n genomes, and join all possible unit setor pairs with common unit setors. The final stage is to add new unit setor pairs to the existing setors. It detects and saves all possible unit setor pairs between the new setor and n setors, and joins all possible unit setor pairs with common unit setors.

The algorithm has two procedures. The first procedure is to cluster all possible orthologs (putative unit setor pairs) from n genomes. The second procedure is to cluster additional orthologs into the already grouped clusters.

First stage : Detect and save all possible unit setor pairs with the best reciprocal BLAST hits among n genome setors, and merge all possible unit setor pairs with common unit setors.

Second stage : To add new unit setors of a new genome setor to the existing clusters, detect and save all possible unit setor pairs with the best reciprocal BLAST hits between the new genome setor and n genome setors, and merge all possible unit setor pairs with common unit setors.

Algorithm 4.1 Algorithm to cluster orthologs from *n* genome setors

Procedure Clustering_Orthologs ( )
Input: All genome setors $\forall_i \hat{G}_i$
Output: The putative setor $\hat{\Gamma}$

1. Clustering_Pair_Orthologs ( )
2. Initial_Clustering_Orthologs ( )
3. **for** $m = 4$ to n
4. Subclustering_Orthologs ( )
5. **end for**

Algorithm 4.2 Algorithm to cluster new orthologs from a new genome setor and *n* genome setors

Procedure Clustering_New_Orthologs ( )
Input: a new genome $\exists_k \hat{G}_k$ and $\forall_i \hat{G}_i$
Output: The putative setor $\hat{\Gamma}$

1. Subclustering_New_Orthologs ( )
2. $m \leftarrow n + 1$
3. Subclustering_Orthologs ( )

Algorithm 4.3 Algorithm for producing orthologs (putative setors) from *n* genome setors

Subprocedure Clustering_Pair_Orthologs ( )
Input: All genome setors $\forall_i \hat{G}_i$
Output: The putative setors $\forall_i \forall_j \hat{G}_{ij}$

1. **for** $i = 1$ to $i = n - 1$ do
2. **for** $j = i + 1$ to $j = n$ do
3. $\hat{G}_{ij} \leftarrow \hat{G}_i \otimes \hat{G}_j$
4. **end for**
5. **end for**

Algorithm 4.4 Algorithm for producing new orthologs (putative unit setor pairs) from a new genome setor and *n* genome setors

Subprocedure Subclustering_New_Orthologs ( )
Input: a new genome setor $\exists_k \hat{G}_k$ and $\forall_i \hat{G}_i$
Output: The putative setors $\forall_i \exists_k \hat{G}_{ik}$

1. $k \leftarrow n + 1$
2. **for** $i = 1$ to $i = n$ do
3. $\hat{G}_{ik} \leftarrow \hat{G}_i \otimes \hat{G}_k$
4. **end for**

Algorithm 4.1 shows the processes of the first stage to detect and cluster putative unit setor pairs among *n* genome setors and Algorithm 4.2 shows the second stage. Algorithms 4.3 and 4.4 show, respectively, the steps to get all possible unit setor pairs with the best reciprocal BLAST hits among *n* selected genome setors, and between a

new genome setor and *n* genome setors. The results of both Algorithms 4.3 and 4.4 can be drawn from InParanoid program. Algorithm 4.5, from three pairwise genome setors, shows all possible steps to search all possible unit setor pairs with common unit setors, and join the unit setor pairs into the total setor $\hat{\Gamma}$. Next, Algorithm 4.6 presents all possible steps to search all unit setor pairs with common unit setors from the pairwise genome setors gotten due to adding the setors 4-*n*th, and join the detected unit setor pairs into $\hat{\Gamma}$.

Algorithm 4.5 Algorithm to cluster orthologs (unit setor pairs with common unit setors) from three pairwise genome setors

Subprocedure Initial_Clustering_Orthologs ( )

Input: The putative setors $\hat{G}_{12}$, $\hat{G}_{13}$, and $\hat{G}_{23}$

Output: The putative setor $\hat{\Gamma}$

1. $\hat{\Gamma}_1 \leftarrow \hat{\Gamma}_1 \cup (\hat{G}_{(1)2} \cap \hat{G}_{(1)3})$
2. $\hat{\Gamma}_2 \leftarrow \hat{\Gamma}_2 \cup (\hat{\Gamma}_2 \cap \hat{G}_{(2)3})$
3. $\hat{\Gamma}_3 \leftarrow \hat{\Gamma}_3 \cup (\hat{\Gamma}_3 \cap \hat{G}_{2(3)})$
4. $\hat{\Gamma}_2 \leftarrow \hat{\Gamma}_2 \cup (\hat{G}_{1(2)} \cap \hat{G}_{(2)3})$
5. $\hat{\Gamma}_3 \leftarrow \hat{\Gamma}_3 \cup (\hat{\Gamma}_3 \cap \hat{G}_{1(3)})$
6. $\hat{\Gamma}_3 \leftarrow \hat{\Gamma}_3 \cup (\hat{G}_{1(3)} \cap \hat{G}_{2(3)})$

Algorithm 4.6 Algorithm to cluster orthologs of pairwise genome setors due to the 4-*n*th setors

Subprocedure Subclustering_Orthologs ( )

Input: The existing setor $\hat{\Gamma}$ and $\forall_i \exists_m \hat{G}_{im}$

Output: The updated setor $\hat{\Gamma}$

1. **for** $i = 1$ to $i = m - 1$ **do**
2. $\quad \hat{\Gamma}_i \leftarrow \hat{\Gamma}_i \cup (\hat{\Gamma}_i \cap \hat{G}_{(i)m})$
3. **end for**
4. **for** $i = 1$ to $i = m - 1$ **do**
5. $\quad \hat{\Gamma}_m \leftarrow \hat{\Gamma}_m \cup (\hat{\Gamma}_m \cap \hat{G}_{i(m)})$
6. **end for**
7. **for** $i = 1$ to $i = m - 2$ **do**
8. $\quad$ **for** $j = i + 1$ to $j = m - 1$ **do**
9. $\quad\quad \hat{\Gamma}_i \leftarrow \hat{\Gamma}_i \cup (\hat{G}_{(i)j} \cap \hat{G}_{(i)m})$
10. $\quad\quad$ **for** $k = 1$ to $k = m - 1$ **do**
11. $\quad\quad\quad$ **if** i != k **then**
12. $\quad\quad\quad\quad \hat{\Gamma}_m \leftarrow \hat{\Gamma}_m \cup (\hat{\Gamma}_m \cap \hat{G}_{k(m)})$
13. $\quad\quad\quad\quad$ **if** $j = k$ **then**
14. $\quad\quad\quad\quad\quad \hat{\Gamma}_k \leftarrow \hat{\Gamma}_k \cup (\hat{\Gamma}_k \cap \hat{G}_{(k)m})$
15. $\quad\quad\quad\quad$ **end if**
16. $\quad\quad\quad$ **end if**
17. $\quad\quad$ **end for**
18. $\quad$ **end for**
19. **end for**
20. **for** $i = 1$ to $i = m - 2$ **do**
21. $\quad$ **for** $j = i + 1$ to $j = m - 1$ **do**
22. $\quad\quad \hat{\Gamma}_j \leftarrow \hat{\Gamma}_j \cup (\hat{G}_{i(j)} \cap \hat{G}_{(j)m})$
23. $\quad\quad$ **for** $k = 1$ to $k = m - 1$ **do**
24. $\quad\quad\quad$ **if** j != k **then**
25. $\quad\quad\quad\quad \hat{\Gamma}_m \leftarrow \hat{\Gamma}_m \cup (\hat{\Gamma}_m \cap \hat{G}_{k(m)})$
26. $\quad\quad\quad\quad$ **if** $i = k$ **then**
27. $\quad\quad\quad\quad\quad \hat{\Gamma}_k \leftarrow \hat{\Gamma}_k \cup (\hat{\Gamma}_k \cap \hat{G}_{(k)m})$
28. $\quad\quad\quad\quad$ **end if**
29. $\quad\quad\quad$ **end if**
30. $\quad\quad$ **end for**
31. $\quad$ **end for**
32. **end for**
33. **for** $i = 1$ to $i = m - 2$ **do**
34. $\quad$ **for** $j = i + 1$ to $j = m - 1$ **do**
35. $\quad\quad \hat{\Gamma}_m \leftarrow \hat{\Gamma}_m \cup (\hat{G}_{i(m)} \cap \hat{G}_{j(m)})$
36. $\quad\quad$ **for** $k = j + 1$ to $k = m - 1$ **do**
37. $\quad\quad\quad \hat{\Gamma}_m \leftarrow \hat{\Gamma}_m \cup (\hat{\Gamma}_m \cap \hat{G}_{k(m)})$
38. $\quad\quad$ **end for**
39. $\quad$ **end for**
40. **end for**

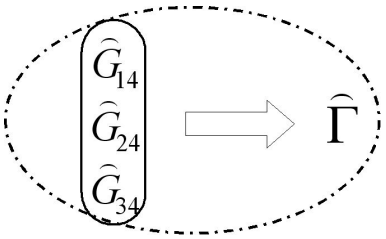


Figure 4.6 The initial relationship (dot-dashed oval) between each of the left pairwise genome setors, which are produced from 1st-3rd genome setors and 4th genome setor, and the total setor.

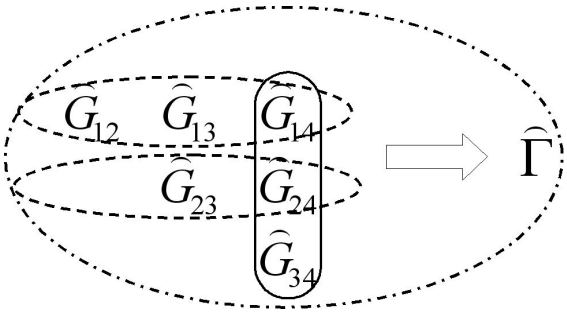


Figure 4.7 The horizontal relationship (dashed oval) between each pairwise genome setor from 1st-3rd genome setor (the unit setor pairs, orthologs, of the pairwise genome setor are not joined into the total setor in Algorithm 4.5) and each pairwise genome setor with orthologs not joined into the total setor in the processes of Figure 4.6. Owing to the prior step, the relationship (dot-dashed oval) occurred between each of the pairwise genome setors and the total setor.

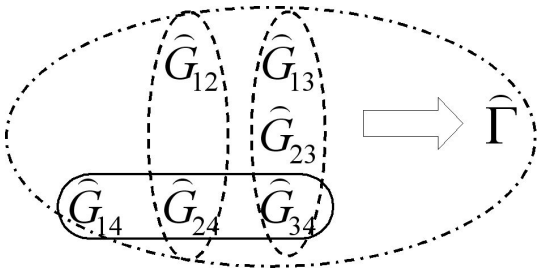


Figure 4.8 The vertical relationship (dashed oval) between the pairwise genome setors (orthologs of that are not joined into the total setor in Figure 4.7), and owing to the prior step the relationship (dot-dashed oval) between each of the pairwsie genome setors and the total setor.

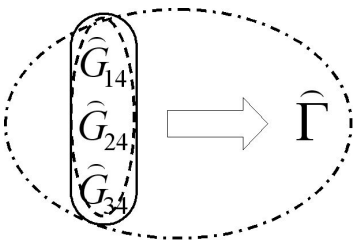


Figure 4.9 The final relationship (dashed oval) between the pairwise genome setors, unit setor pairs of which are not joined into the total setor in Figure 4.8. Owing to the prior step, the dot-dashed oval, as the next step, represents the relationship between each of the pairwise genome setors and the total setor.

We, here, explain all the steps in detail to cluster the 4th genome setor into the total setor from the three genomes. All the unit setor pairs with common unit setors among three setors are represented as in Algorithm 4.5. In pairwise genome setors and the total setor, the common genome are represented as the subscript numbers in parentheses.

From the new three pairwise genome setors due to the additional 4th genome setor, all possible steps to search and save the unit setor pairs with common unit setors, when compared with the total setor, are represented as Figure 4.6 and the following expressions in general, which have the same meaning as the statements of the lines 1-6 in Algorithm 4.6. All the steps can be generalized as case (a) and case (b) of formula (4.12).

$$
\begin{aligned}
&Case\ \ (a): \hat{\Gamma}_i^{n+1} = \hat{\Gamma}_i^{n} \cup (\hat{\Gamma}_i^{n} \cap \hat{G}_{(i)m}) \\
&where \qquad (i): 1 \le i \le m-1 \\
&Case\ \ (b): \hat{\Gamma}_m^{n+1} = \hat{\Gamma}_m^{n} \cup (\hat{\Gamma}_m^{n} \cap \hat{G}_{i(m)}) \\
&where \qquad (m): 1 \le i \le m-1 .
\end{aligned}
\tag{4.12}
$$

After the above phase, all possible steps to search and store the unit setor pairs with common unit setors from the tables are considered as Figure 4.7 in case of the 4th genome setor and the following expressions in general. In this case the lines 7-19 of Algorithm 4.6 have the same steps as those. All the steps are generalized as case (a) and case (b) of formula (4.13).

$$
\begin{aligned}
&Case\ (a): \quad \widehat{\Gamma}_i^{n+1} = \widehat{\Gamma}_i^n \cup (\widehat{G}_{(i)j} \cap \widehat{G}_{(i)m}) \\
&where \quad (i): 1 \le i \le m-2, \quad i+1 \le j \le m-1. \\
&Case\ (b): \begin{cases} \widehat{\Gamma}_m^{n+1} = \widehat{\Gamma}_m^n \cup (\widehat{\Gamma}_m^n \cap \widehat{G}_{k(m)}) \quad when \quad (m): \quad if \quad i \ne k \\ \widehat{\Gamma}_k^{n+1} = \widehat{\Gamma}_k^n \cup (\widehat{\Gamma}_k^n \cap \widehat{G}_{(k)m}) \quad when \quad (k): \quad if \quad j = k \end{cases} \\
&where \quad 1 \le i \le m-2, \quad i+1 \le j \le m-1, \quad and \quad 1 \le k \le m-1.
\end{aligned}
\tag{4.13}
$$

After finishing the above phase, Figure 4.8 in case of the 4th genome setor and in general formula (4.14) follows, which are also expressed by the lines 20-32 of Algorithm 4.6.

$$
\begin{aligned}
&Case\ (a): \widehat{\Gamma}_j^{n+1} = \widehat{\Gamma}_j^n \cup (\widehat{G}_{i(j)} \cap \widehat{G}_{(j)m}) \\
&where \quad (j): 1 \le i \le m-2, \quad i+1 \le j \le m-1. \\
&Case\ (b): \begin{cases} \widehat{\Gamma}_m^{n+1} = \widehat{\Gamma}_m^n \cup (\widehat{\Gamma}_m^n \cap \widehat{G}_{k(m)}) \quad when \quad (m): \quad if \quad j \ne k \\ \widehat{\Gamma}_k^{n+1} = \widehat{\Gamma}_k^n \cup (\widehat{\Gamma}_k^n \cap \widehat{G}_{(k)m}) \quad when \quad (k): \quad if \quad i = k \end{cases} \\
&where \quad 1 \le i \le m-2, \quad i+1 \le j \le m-1, \quad and \quad 1 \le k \le m-1.
\end{aligned}
\tag{4.14}
$$

Finally, all possible steps among just the pairwise genome setors due to the fourth genome setor are represented as Figure 4.9 in case of the 4th genome setor, and the following expressions in general, which have the same steps as the lines 33-40 of Algrorithm 4.6. All the steps are expressed as case (a) and case (b) of formula (4.15).

$$Case\,(a): \hat{\Gamma}_m^{n+1} = \hat{\Gamma}_m^n \bigcup (\hat{G}_{i(m)} \bigcap \hat{G}_{j(m)})$$
$$where\,(m): 1 \le i \le m-2, \quad i+1 \le j \le m-1.$$

$$Case\,(b): \hat{\Gamma}_m^{n+1} = \hat{\Gamma}_m^n \bigcup (\hat{\Gamma}_m^n \bigcap \hat{G}_{k(m)})$$
$$where\,(m): 1 \le i \le m-2, \quad i+1 \le j \le m-1, \quad and \quad j+1 \le k \le m-1.$$

(4.15)

We suggested an automatic method to reduce time and efforts, and extend InParanoid for clustering orthologs. The time complexity of this algorithm is $O(lmn^4)$ when l is the number of the unit setor pairs in the total table, m is the number of the unit setor pairs in the normal tables, and n is the number of genomes. The complexity of $O(lmn^4)$ is equal to $O(ln^2\ mn^2)$. Then it is approximately equal to $O(N^2)$ when $N \approx ln^2 \approx mn^2$ where N is the number of the unit setor pairs in the total table. We can also conclude that the quality of this method settles on the products of orthologs detected by the InParanoid.

4.3 Parallel Algorithm

The algorithm introduced in section 4.2 has unreasonable time complexity of $O(N^2)$ for the large number of unit setor pairs, N, in the total table. We also introduce a parallel method to cluster orthologs from multiple genomes to improve performance of the iterative method.

We use a hypercube network model as shown in Figure 4.10, which is very useful to present an efficient algorithm for some basic communication operations. A d-dimensional hypercube consists of $p = 2^d$ processors. The time complexity for

communication on the hypercube is $O(\log p)$ [Kuma94].

Algorithm 4.7 has an overview of the suggested parallel algorithm. The host program runs on the master processor P_0. First, Step 1 broadcasts all genomes to every processor. Second, step 2 runs Algorithm 4.8. The lines 3-5 combine each table into the total table. Step 6 broadcast the total table to the all the worker processors. The step 7 runs Algorithm 4.9. Last, the lines 8-10 join the total tables together. As a result of all the processes, the total putative setor is produced.

In Algorithm 4.8, the k_{th} processor runs a sequential process to produce pairwise genome setors and add them to T_k, independently, $i = (k-1) \cdot \lfloor n/p \rfloor + 1$ through $1 \le (\alpha \cdot \lfloor k \cdot n/p \rfloor \le (n-1)$, where α is used to share the load evenly even if a little bit rough. Step 7 broadcasts T_k, produced in k_{th} processor, to the master processor P_0.

In Algorithm 4.9, the k_{th} processor runs a sequential process to cluster orthologs (unit setor pairs) of the table Γ and broadcast and check the gene ID of unit setor pairs, independently, $i = (k-1) \cdot \lceil l/p \rceil + 1$ through $k \cdot \lceil l/p \rceil \le l$. Step 6 broadcasts Γ_k to the master processor P_0. In addition, we define the symbol $\oplus$ to be represented as $g_i \oplus \forall_j g_j$ as a meaning to search and merge the unit setor pairs with the identification of g_i in all genes of Γ when given a common gene g_i.

Here, the time complexity of Algorithm 4.8 is $O(S_i S_j U_k U_l n^2/P)$, where S_i and S_j are the length of sequences of each protein, U_k and U_l are the number of proteins of each genome, and n is the number of genomes. The time complexity of Algorithm 4.9 is $O(N^2/P)$, where N is the number of clusters of the total table Γ.

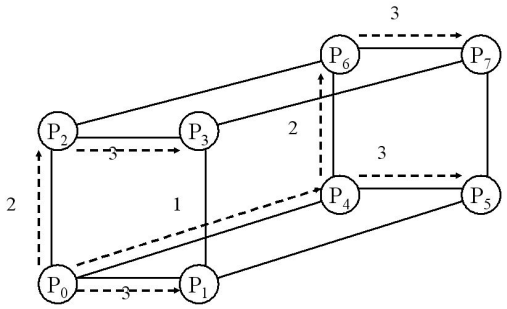


Figure 4.10 An example of one-to-all broadcast on a three dimensional hypercube. The number of processors is 8 and the information in P_0 (master processor) can be sent to the other processors (worker processors) within three steps.

Algorithm 4.7 Parallel algorithm to cluster orthologs to the master processor P_0

Procedure Parallel_Clustering_Orthologs ( )

Input : All genome setors $\forall_i \widehat{G}_i$

Output : The putative setor $\widehat{\Gamma}'$

1. Broadcast $\forall_i \widehat{G}_i$ to all the worker processors
2. Parallel_Clustering_Pair_Orthologs ( ) in each worker processor
3. **For** $k = 1$ **to** p **do**
4. Add T_k to Γ
5. **end for**
6. Broadcast Γ to all the worker processors
7. Parallel_Subclustering_Orthologs ( ) in each worker processor
8. **For** $k = 1$ **to** p **do**
9. $\Gamma' \leftarrow \Gamma' \bigcup \Gamma_k$
10. **end for**

Algorithm 4.8 Parallel algorithm to produce ortholgs (unit setor pairs) between two complete genomes to each worker processor

Subprocedure Parallel_Clustering_Pair_Orthologs ( )

Input : All genome setors $\forall_i \widehat{G}_i$

Output : Each table T_k

1. **for** $i = ((k-1)\cdot\lfloor n/p \rfloor + 1)$ to $1 \le (\alpha \cdot \lfloor k \cdot n/p \rfloor) \le (n-1)$ **do**
2. **for** $j = i + 1$ to n **do**
3. $\widehat{G}_{ij} \leftarrow \widehat{G}_i \otimes \widehat{G}_j$
4. Add $\widehat{G}_{ij}$ to T_k
5. **end for**
6. **end for**
7. Broadcast T_k to the P_0 processor

Algorithm 4.9 Parallel algorithm to cluster orthologs (joining unit setor pairs with common unit setors) in the total table to each worker processor

Subprocedure Parallel_Subclustering_Orthologs ( )

Input : The table Γ

Output : Each table Γ_k

1. $l \leftarrow$ the total number of rows in Γ
2. **for** $i = ((k-1)\cdot\lceil l/p \rceil + 1)$ to $(k \cdot \lceil l/p \rceil) \le l$ **do**
3. $\Gamma_k \leftarrow g_i \oplus \forall_j g_j$
4. Broadcast ID and check Flag to all the worker processors
5. **end for**
6. Broadcast Γ_k to the P_0 processor

We used a hypercube network which is efficient for communication. Therefore, the time complexity of the parallel algorithm is $O(N^2/P)$, where N is the number of the unit setor pairs in the total table and P is the number of processors.

4.4 ReMark Clustering Approach

The previous two methods we introduced in Section 4.2 and 4.3 focus on the work to extend InParanoid and improve time efficiency. In this Section, we introduce an automatic ReMark method that employs the **Re**cursive search and the **Mark**ov algorithms. This method is designed to produce much more orthologs (with keeping better or similar accuracy) than the existing methods.

Our approach has two stages roughly. One is to produce a score matrix from all possible pairwise genome setors of n genome setors. The other is to produce a putative setor by employing the MCL algorithm. Figure 4.11 shows the overview of our approach. All the processes to produce a putative setor are shown in Algorithm 4.10. First, we select genomes of our interest and prepare their protein sequence sets. Second, all possible pairwise genome setors are produced in Algorithm 4.11 by using all possible best reciprocal BLAST hits between two genome setors. Third, a score matrix is produced by

running the recursive search algorithm as illustrated in Algorithm 4.12. Fourth, the score matrix is transformed into the Markov matrix to simulate random walks on a graph. Finally, the Markov clustering is executed to produce the putative setor by the MCL algorithm.

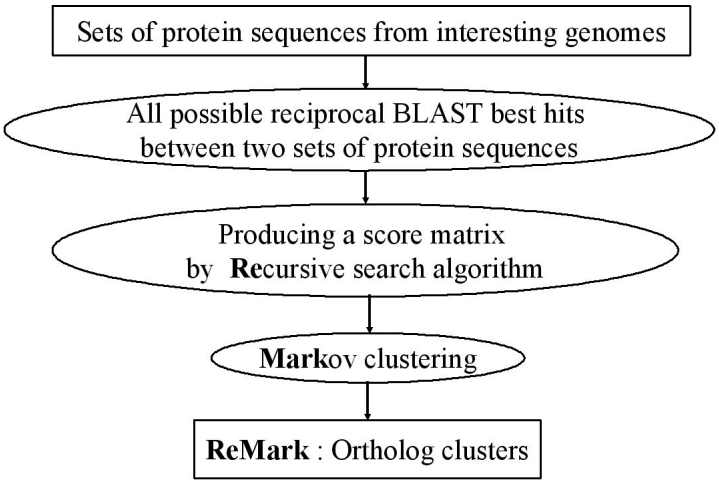


Figure 4.11 The overall steps of our approach.

Algorithm 4.10 The ReMark clustering algorithm

Procedure ReMark_Clustering_Orthologs ( )
Input : All genome setors $\forall_i \widehat{G}_i$
Output : The putative setor $\widehat{\Gamma}$

1. New_Clustering_Pair_Orthologs ( )
2. **for** $i = 1$ **to** the total number of lines **do**
3. Recursive_Search ( ) for a gene of i line in T
4. **if** the size of the score matrix ≥ 2 **then**
5. Change the score matrix into the Markov matrix
6. Markov_Clustering ( )
7. Save each cluster in a file respectively
8. **end if**
9. **end for**

Algorithm 4.11 Algorithm for clustering orthologs from all pair genomes

Subprocedure New_Clustering_Pair_Orthologs ( )
Input : All genome setors $\forall_i \widehat{G}_i$
Output : The combined table T

1. **for** $i = 1$ to $n - 1$ **do**
2. **for** $j = i + 1$ to n **do**
3. $\widehat{G}_{ij} = \widehat{G}_i \otimes \widehat{G}_j$
4. Add $\widehat{G}_{ij}$ to T
5. **end for**
6. **end for**

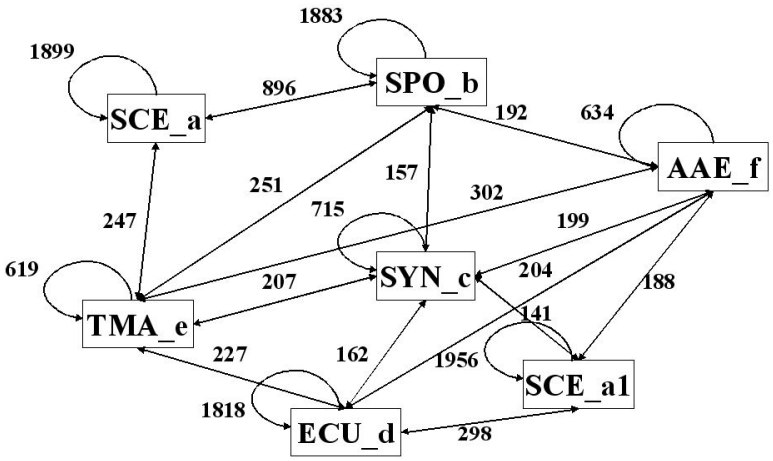
Algorithm 4.12 Our recursive search algorithm

```
Subprocedure Recursive_Search ( )
Input : The gene and the combined table T
Output : A score matrix
1.  Read the combined table
2.  for i = 1 to the total number of lines do
3.      if (The first gene ID = The gene ID) and (Gene Pairs Flag = 0) then
4.          Save the gene pair and score in the score matrix
5.          Check the Flag of Gene Pairs and Genes
6.          if The second gene Flag = 0 then
7.              Recursive_Search(the second gene)
8.          end if
9.      end if
10.     if (The second gene ID = The gene ID) and (Gene Pairs Flag = 0) then
11.         Save the gene pair and score in the score matrix
12.         Check the Flag of Gene Pairs and Genes
13.         if The first gene Flag = 0 then
14.             Recursive_Search(the first gene)
15.         end if
16.     end if
17. end for
```

In the previous work [Kim06], we proposed an automatic method that clusters orthologous proteins from $n(n-1)/2$ ortholog tables taken from InParanoid for n genomes. But the algorithm is somewhat complex and difficult to understand. A simple and clear method is proposed here. This algorithm starts with the combined table of $n(n-1)/2$ pairwise genome setors, and recursively detects unit setor pairs with identical unit setors in the table and writes the gene ID and score in a score matrix as shown in Algorithm 4.12. The score matrix represents the initial clusters as shown in the graph on the top in Figure 4.12.

Note that we didn't use the thresholds in the first step to generate the initial clusters. Our recursive search algorithm starts from all reciprocal best hits without pruning. We could perhaps have improved the accuracy of the initial clusters by enforcing the thresholds. However, it could generate a lot of false negatives. In order to avoid this problem, we include all results in the first step and refine the results in the second step using the MCL algorithm through adjusting its inflation factor.

MCL algorithm is a network flow based graph partitioning algorithm. It can generate clusters of different levels of granularity. Cluster granularity can be controlled by inflation factor. Large inflation factor generates large numbers of small clusters while small inflation factor generates small numbers of large clusters.



	SCE_a	SPO_b	SYN_c	SCE_a1	ECU_d	TMA_e	AAE_f
SCE_a	1899	896	0	0	0	249	0
SPO_b	896	1883	157	0	0	251	192
SYN_c	0	157	715	141	162	207	199
SCE_a1	0	0	141	1956	298	0	188
ECU_d	0	0	162	298	1818	227	204
TMA_e	247	251	207	0	227	619	302
AAE_f	0	192	199	188	204	302	634

Figure 4.12 The diagram of orthologs and score matrix. The graph on the top represents putative orthologs (joining unit setor pairs) with the reciprocal BLAST best hits. The table below shows the corresponding score matrix.

Unlike OrthoMCL [Li03], we make use of diagonal terms that are self-reciprocal best hit scores computed against themselves. We observed in our experiment that more reliable clusters were generated with diagonal scores than without. In our experiment later, we will show evidence that the diagonal scores are critical to improve the accuracy of our results. OrthoMCL also normalized the weights to minimize the impact of inparalogs. We did not

consider normalization since we do not include inparalogs in our result.

We obtain a group of protein clusters after running MCL with our score matrix. As was done in COG, we only include the clusters with 3 or more unit setors in the final result.

We used two main algorithms, the recursive search and MCL methods, in the ReMark clustering method. The time complexity of the recursive search algorithm is $O(N^2)$, where N is the number of unit setor pairs in the total table. The time complexity of the MCL algorithm is $O(n^3)$, where n is the number of nodes of the input graph.

Chapter 5

ReMark: Evaluation and Results

Since orthologs were first suggested by Fitch (1970), several ortholog clusters have been introduced and used successfully for comparative genomics and genome annotation. The first explorer for this challengeable work started with comparison of proteins encoded in seven complete genomes using the gapped BLAST program [Tatu97]. Their work has been extended continuously so that the COG (Clusters of Orthologous Group) database in NCBI includes OPCs (Orthologous Protein Clusters) from 66 complete genomes.

Unfortunately, some of COGs include large numbers of paralogs from one lineage. Such clusters are, accordingly, not very useful for predicting functions for newly sequenced genes. In addition to that, it takes time consuming processes due to manual work.

To address these problems, Remm [Remm01] introduced InParanoid, a fully

automatic program, to detect orthologous proteins between two species. In order to reduce the number of paralogs in the result, it uses two types of thresholds, score cut-off and overlap cut-off. MutiParanoid [Alex06] published recently has extended InParanoid to cluster proteins from multiple proteomes.

Meanwhile, Li et al. developed OrthoMCL [Li03], which generates OPCs from multiple species using Markov Cluster (MCL) algorithm [Dong00]. Chen et al. later constructed an OPC database called OrthoMCL-DB [Chen06] using OrthoMCL algorithm. It contains OPCs of 55 species.

Both InParanoid and OrthoMCL use thresholds to improve the accuracy of result. InParanoid uses a score cut-off of 50bits and an overlap cut-off of 50%. OthoMCL chooses a P-value cut-off of 1e-5. Because of this thresholding, however, many orthologs are not detected. Nonetheless, without the thresholding, it is very difficult to ensure the accuracy of the clusters as false positives are increasing.

In this work, we propose a new automatic method introduced in Section 4.4 for constructing OPCs. Our ReMark clustering method can produce much more orthologs than previous methods, because it does not use BLAST thresholding that can prematurely eliminate true positives in the early stage of the clustering process, while it refines the initial clusters from the recursive search algorithm using MCL by adjusting the inflation factor. Especially, we have considerably improved the accuracy to detect orthologs by adding the self-sequence similarity scores.

To assess the accuracy of our results, we compared our OPCs against that of KO. In order to quantify the similarity between the two groups of OPCs, we introduced a distance measure called ‘least moving distance,’ which computes the number of moves that genes in one OPCs have to make before the two OPCs become identical.

Another challenge in developing OPC algorithms is that it is difficult to evaluate the quality of clusters as there is no large enough curated reference ortholog data set. InParanoid used a small set of manually curated ortholog set including species such as human, worm and fly. OrthoMCL simply compared their results against that of InParanoid and showed that the results are similar. We compared our results with KO, which has diverse ortholog clusters across various distant species.

5.1 Concept of Similarity

In order to compare two different clustering methods, we need to be able to quantify the similarity between the two clustering results. OrthoMCL used the number of "coherent" clusters between the two results. They define the coherence as one cluster in one result set is a subset of a cluster in the other result set.

Let us consider the cases shown in Figure 5.1. Figure 5.1(a) shows two clustering results, K and G. K consists of four clusters, K1 to K4, while G has only one Cluster, G1. According to OrthoMCL, the number of the coherent clusters between the two groups is four because every cluster in K is a subset of G1. On the other hand, the two groups in Figure 5.1(b) represent zero coherence as no cluster is subset of any other. Intuitively, however, both Figure 5.1(a) and (b) seem to have low similarity and we cannot say clusters in Figure 5.1(a) is much more coherent than those in Figure 5.1(b).

Owing to these problems, we, therefore, propose the new concept of similarity between two groups of clusters. The idea is that we view the difference between two groups as the least number of proteins that need to move from one cluster to another within one group to

make the group identical to the other. For instance, in Figure 5.1(a), three genes need to move to make the two groups identical, e.g., genes 2, 3, and 4 to cluster K1. The similarity can be computed as $S = 1 - M / P$, where M is the least number of proteins moved and P is the total number of proteins. According to this formula, the similarities for Figure 5.1(a) and (b) are 1/4 and 1/2, respectively. A detailed explanation of the algorithm to compute the least number of moves, is given in the following section.

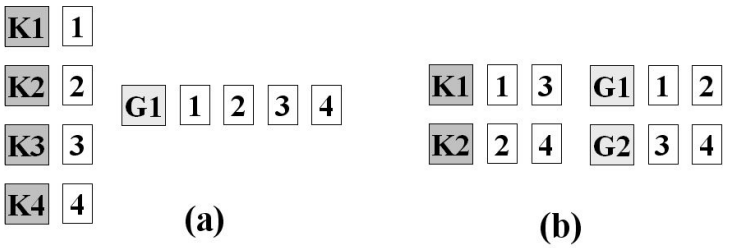


Figure 5.1 Comparing clustering results. Two groups of clusters, Ks and Gs, are obtained from different OPC algorithms. According to the concept of coherence in the OrthoMCL, the number of the coherent clusters between the two groups in (a) is four, and zero in (b). However, using our similarity measure, the values of (a) and (b) are 1/4 and 1/2 respectively. Our results, intuitively, seem more reasonable for comparing the similarity of two groups.

5.2 Least-move Algorithm

Consider the instance in Figure 5.2. Using this example, we will briefly explain how our algorithm finds the least number of proteins to be moved. First, the algorithm finds the cluster pair that shares the most number of proteins. In our example, it is K1 and G1 pair as they share three proteins. Second, it identifies the disjoint members between the two clusters, K1 and G1. Protein 8 in K1 and protein 3 in G1 are identified. We check the two proteins as the ones to be moved. The algorithm then moves to the next pair that shares the next most proteins (in our example, K2 and G2), and repeats the previous two steps with them. Protein 4 is the only disjoint member between the two. The algorithm checks it as the third protein to be moved and completes as there is no more clusters to be processed left in G. As the result, the least number of proteins to be moved is three and the similarity score for Figure 5.2 is 5/8. Algorithm 5.1 shows the details of this algorithm.

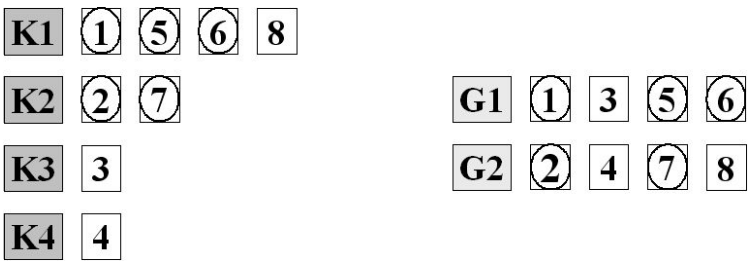


Figure 5.2 An example for illustrating the least-move algorithm. The first step finds the cluster pair (K1 and G1) that shares the most number of proteins (shown in the circles). The second step finds and checks the disjoint members (8 and 3) between the two clusters, K1 and G1. This procedure is repeated until all the members of clusters are checked.

Algorithm 5.1 Least-move algorithm to calculate similarity between two groups

Procedure Cal_Similariy ( )

Input: Two Groups with same genes

Output: The rate of similarity between two groups, K group and G group

1. Initialize similarity, moves, and genes the
2. Create dictionary data structure of K group and G group
3. Initialize the Flag_T1 of K group and the Flag_T2 of G group
4. Cluster all the genes with K group
5. Cluster all the genes with G group
6. Create dictionary data structure for the key flags of K group and G group
7. Create dictionary data structure of the genes for finding the largest cluster
8. Create dictionary data structure of the genes for the list moving of K group
9. Create dictionary data structure of the genes for the list moving of G group
10. Initialize the key flags, Flag_kg and Flag_g, of K group and G group
11. Initialize the gene flags, Flag_g1 and Flag_g2, for the list moving of K group and G group
12. **while** (Flag_T1 = 0 or Flag_T2 = 0) **do**
13. **for** all genes in all clusters **do**
14. Initialize the Flag_gene of the genes for finding the cluster with largest genes
15. Find the cluster pair with the largest genes detected by K group and G group
16. Check the Flag_gene of the genes
17. **end for**
18. **for** all the genes in the cluster with the largest genes in K group **do**
19. Find the identical genes in the clusters of G group with the genes of K group,
20. Check the Flag_g1 for the list moving of K group
21. Check the moves
22. Check the Flag_g of G group
23. **end for**
24. **for** all the genes in the cluster with the largest genes in Group **do**
25. Find the identical genes in the clusters of K group with the genes of G group,
26. Check the Flag_g2 for the list moving of G group
27. Check the moves
28. Check the Flag_kg of K group
29. **end for**
30. Check the Flag_T1 of K group and the Flag_T2 of G group
31. **end while**
32. Similarity ← 1 – moves/genes

5.3 Experimental Results

We used six different genomes in our test, three eukaryotes and three bacteria, as shown in Table 5.1.

Table 5.1 The species used in our experiments.

SCE	*Saccharomyces cervisiae*(baker's yeast)	Eukaryota
SPO	*Schizosaccharomyces pombe*(fission yeast)	Eukaryota
ECU	*Encephalitozoon cuniculi*	Eukaryota
AAE	*Aquifex aeolicus* VF5(NC_000918)	Bacteria
TMA	*Thermotoga maritime* MSB8(NC_000853)	Bacteria
SYN	*Synechocystis* PCC6803(NC_000911)	Bacteria

Figure 5.3 compares the number of orthologs detected by InParanoid (IP) and our method with varying inflation factors. As explained in Section 2.3, we only considered the clusters containing three or more proteins; that is, we disregarded the clusters with only one or two proteins. The result from InParanoid is shown in the far left. InParanoid found 4706 orthologs from the six genomes in total. Our method found far more numbers of orthologs especially when small inflation factors are used. As the inflation factor increased, the number of proteins gradually decreased. Our method found more than twice as many proteins as InParanoid when the inflation factor of 1.2 or less is used. With 1.3, it found 7248 proteins and with 1.4, 5415 proteins, which represent roughly 54% and 15% increase from InParanoid. The total number of proteins in the six genomes is 19,468 (5,869 SCE; 5,045 SPO; 1,996 ECU; 1,529 AAE; 1,858 TMA; 3,171 SYN).

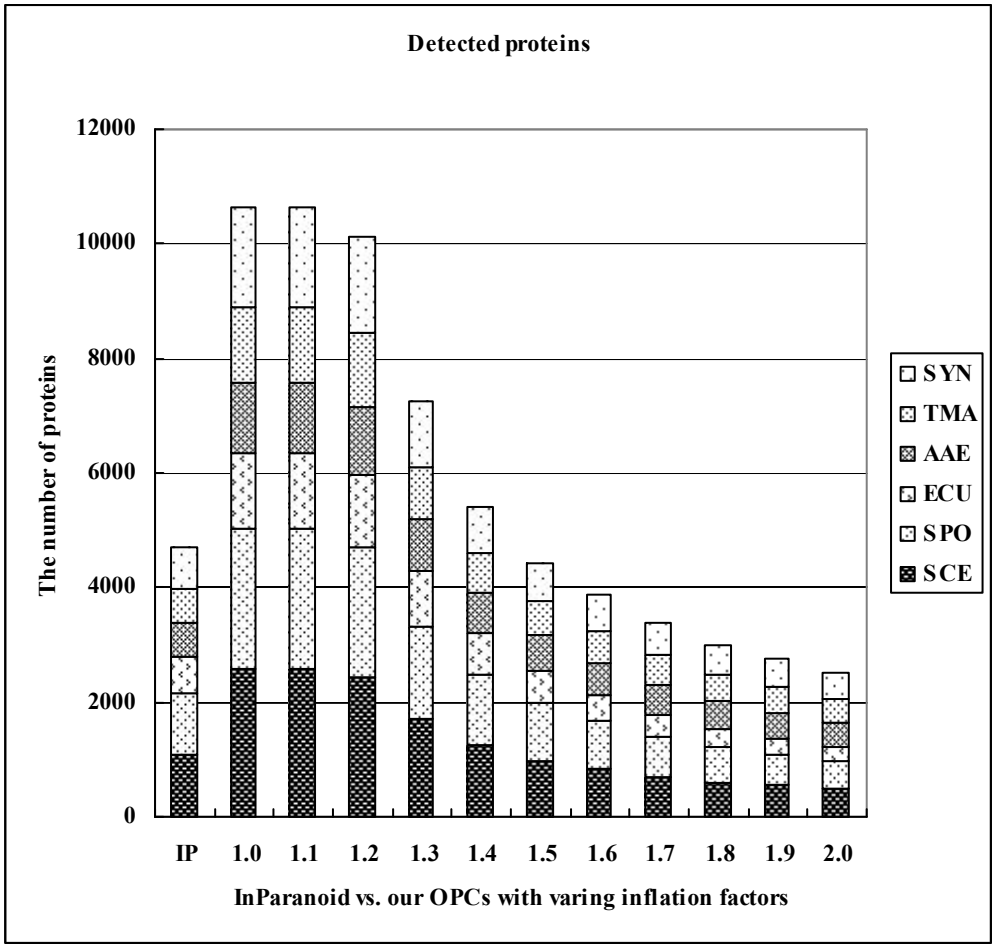


Figure 5.3 The number of orthologs detected by InParanoid (IP) and our method with varing inflation factors. InParanoid found 4706 orthologs from the six genomes in total. As the inflation factor increased, the number of proteins gradually decreased. Our method found more than twice as many proteins as InParanoid when the inflation factor of 1.2 or less is used. With 1.3, it found 7248 proteins and with 1.4, 5415 proteins, which represent roughly 54% and 15% increase from InParanoid. The total number of proteins in the six genomes is 19,468 (5,869 SCE; 5,045 SPO; 1,996 ECU; 1,529 AAE; 1,858 TMA; 3,171 SYN).

For estimating the accuracy of the OPC results, we used OPCs published by KO as the gold standard. KO clusters are constructed manually from known pathways and contain only experimentally validated orthologs. The number of orthologs identified in KO OPCs is small as the number of known pathways is limited. In order to be able to compare the results from InParanoid and our method, we extract from the results only the proteins that overlap with KO and use them to compute the accuracy. Figure 5.4 shows the number of proteins that overlap with KO OPCs for each case shown in Figure 5.3. InParanoid produced 2153 orthologs overlapping with KO while our method produced 2321 and 2674 orthologs with inflation factors 1.4 and 1.3 respectively. Note that orthologs from ECU and SCE appear not significant in the graph because KO OPCs do not contain many proteins from the two genomes.

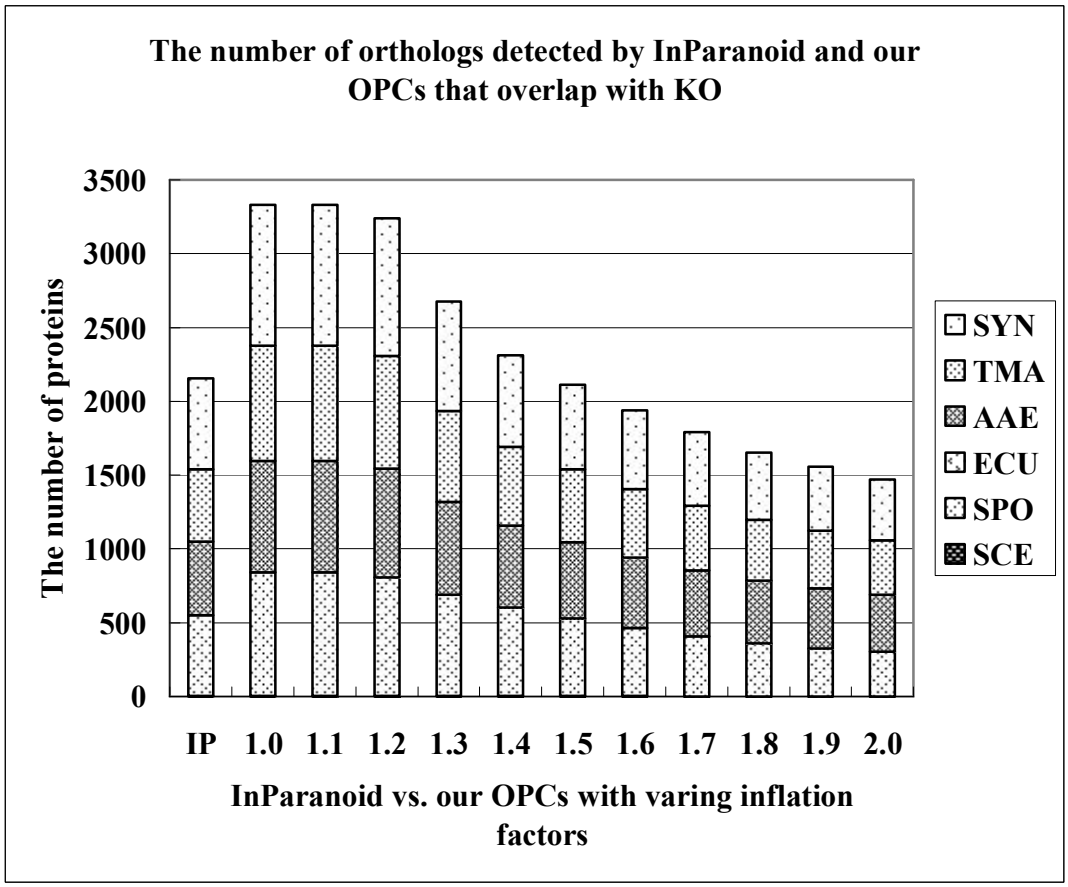


Figure 5.4 The number of proteins that overlap with KO OPCs for each case shown in Figure 5.3. This graph shows the number of proteins that overlap with KO OPCs for each case shown in Figure 5.3. InParanoid produced 2153 orthologs overlapping with KO while our method produced 2321 and 2674 orthologs with inflation factors 1.4 and 1.3 respectively. Note that orthologs from ECU and SCE appear not significant in the graph because KO OPCs do not contain many proteins from the two genomes.

Figure 5.5 shows the accuracy of the clustering results compared against KO OPCs. We used the similarity measure introduced in Section 2.3 to show how well the resulting OPCs match to KO OPCs. KO OPCs are overlapping clusters (i.e., one protein can belong to more than one clusters) while OPCs from InParanoid and our method are disjoint clusters. In order to be able to compare them, we transformed KO OPCs to a disjoint one by removing duplicate proteins. As shown in Fig 5.5, InParanoid produced the result about 88% matching with KO OPCs. On the other hand, our method produced results that are not quite similar to KO OPCs when small inflation factors are used. However, as the inflation factor increases, the accuracy improves significantly. In fact, our OPCs, at inflation factor 1.3, include 54% more orthologs than InParanoid (i.e., 7248 vs. 4706) while keeping a little less accuracy (1.7%) than InParanoid. In case of the factor = 1.4, our OPCs produce 15% more orthologs than InParanoid (i.e., 5415 vs. 4706) while keeping a little more accuracy (1.4%) than InParanoid.

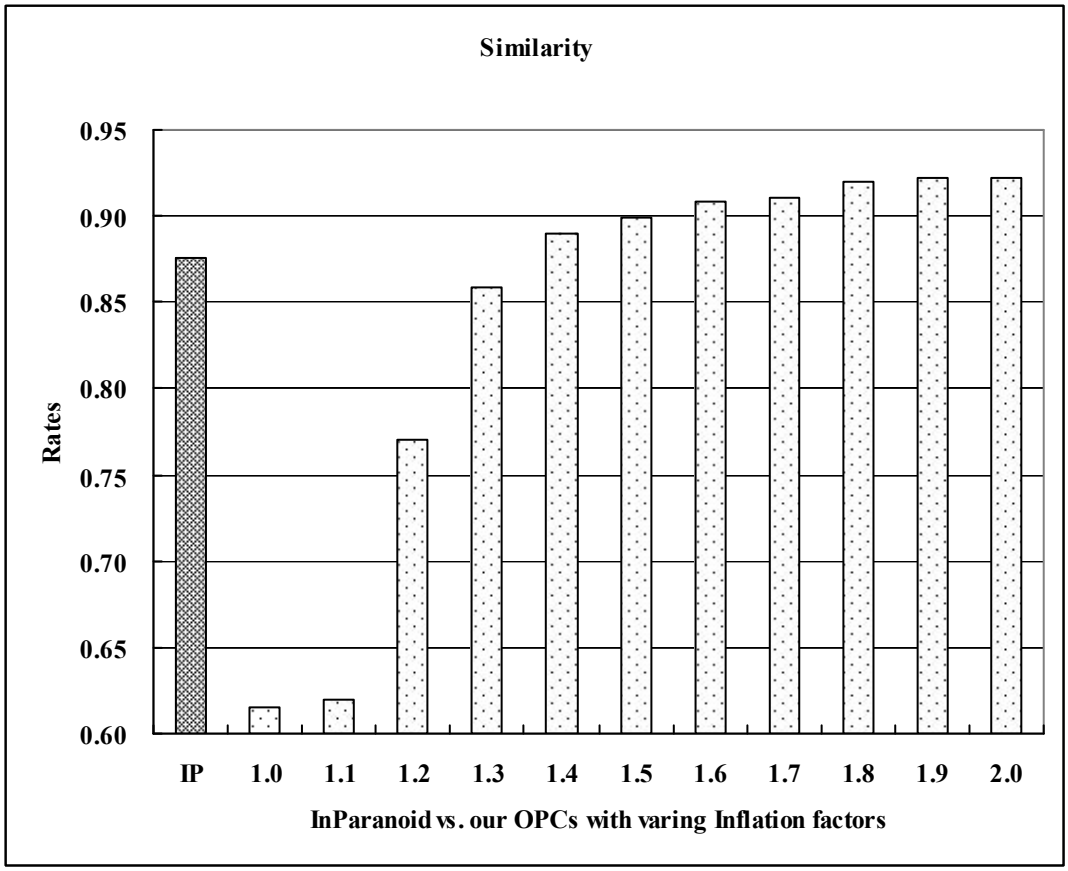


Figure 5.5 The similarity between KO and InParanoid(IP) and our OPCs with varing inflation factors. InParanoid produced the result about 88% matching with KO OPCs. On the other hand, our method produced results that are not quite similar to KO OPCs when small inflation factors are used. However, as the inflation factor increases, the accuracy improves significantly. In fact, our OPCs, at inflation factor 1.3, include 54% more orthologs than InParanoid (i.e., 7248 vs. 4706) while keeping a little less accuracy (1.7%) than InParanoid. In case of the factor = 1.4, our OPCs produce 15% more orthologs than InParanoid (i.e., 5415 vs. 4706) while keeping a little more accuracy (1.4%) than InParanoid.

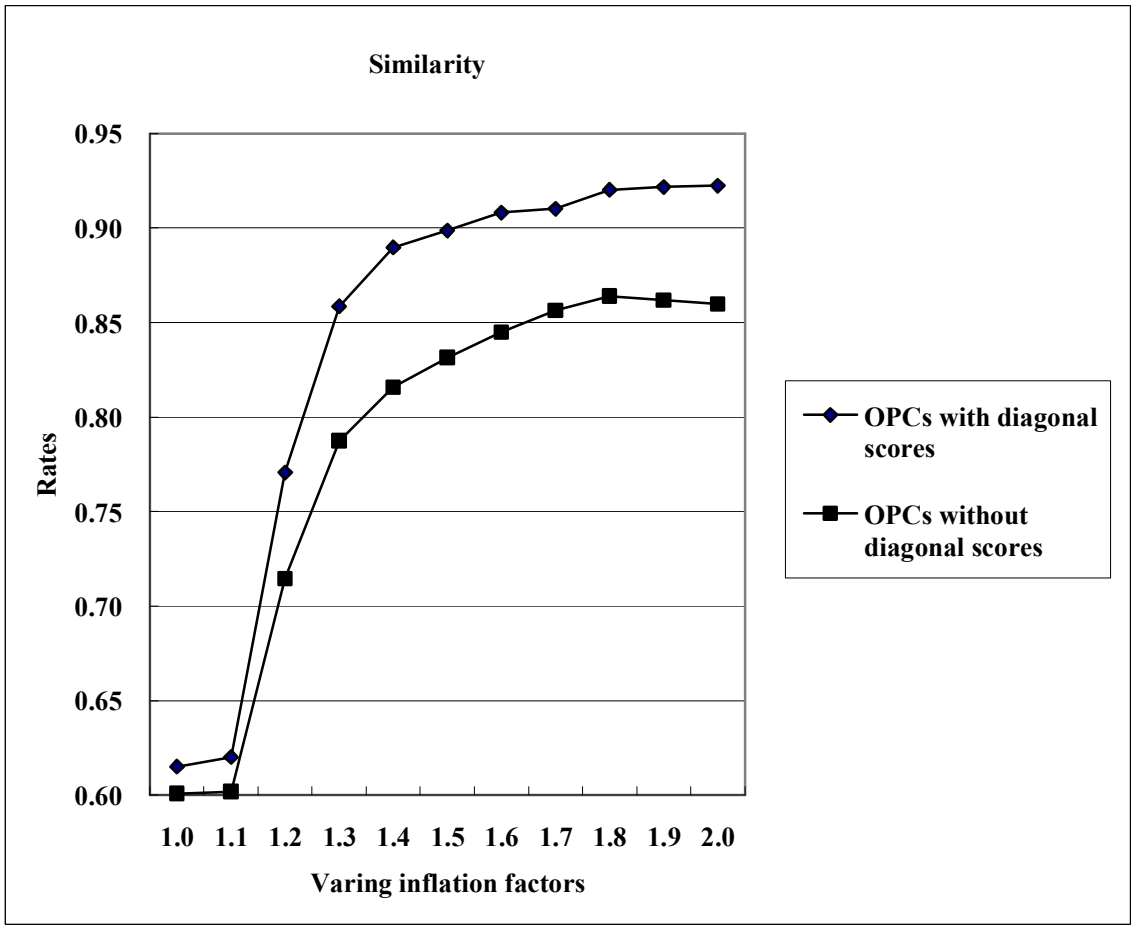


Figure 5.6 The similarity between our OPCs with and without diagonal scores. About 6~7% of the difference is kept when the inflation factor increases from 1.2. Therefore, we can notice that diagonal scores play an important role for constructing OPCs.

Figure 5.6 shows the difference of the accuracy between our OPCs with and without diagonal scores against KO OPCs. About 6~7% of the difference is kept when the inflation factor increases from 1.2. Therefore, we can notice that diagonal scores play an important role for constructing OPCs.

Figure 5.7-9 show the distribution of clusters for each of the six genomes with respect to the numbers of proteins from the same genome in the same clusters. We compared InParanoid and our method with the inflation factors of 1.3 and 1.4. For example, the tall bar in the front left corner (for SCE genome) represents the proportions of clusters where only one protein from SCE is included. Similarly, the next bar at x = 2, y = SCE, represents the proportion of clusters where two SCE proteins are included. For example, InParanoid has 70% of clusters, 826 out of 1187 clusters, that inlcude one SCE protein in them. Whereas our method with inflation factor 1.3 and 1.4 has 77% and 78% clusters respectively. For the case where two proteins from the SCE genome are clustered together, InParanoid has 8.4% of clusters and our method with 1.3 and 1.4 has 5.4% and 2.5% respectively. It is desirable to have smaller numbers of proteins from the same genome in each cluster as possible because multiple proteins from the same genome that are clustered together can be paralogs. With that respect, our method produced in a sense purer OPCs than InParanoid.

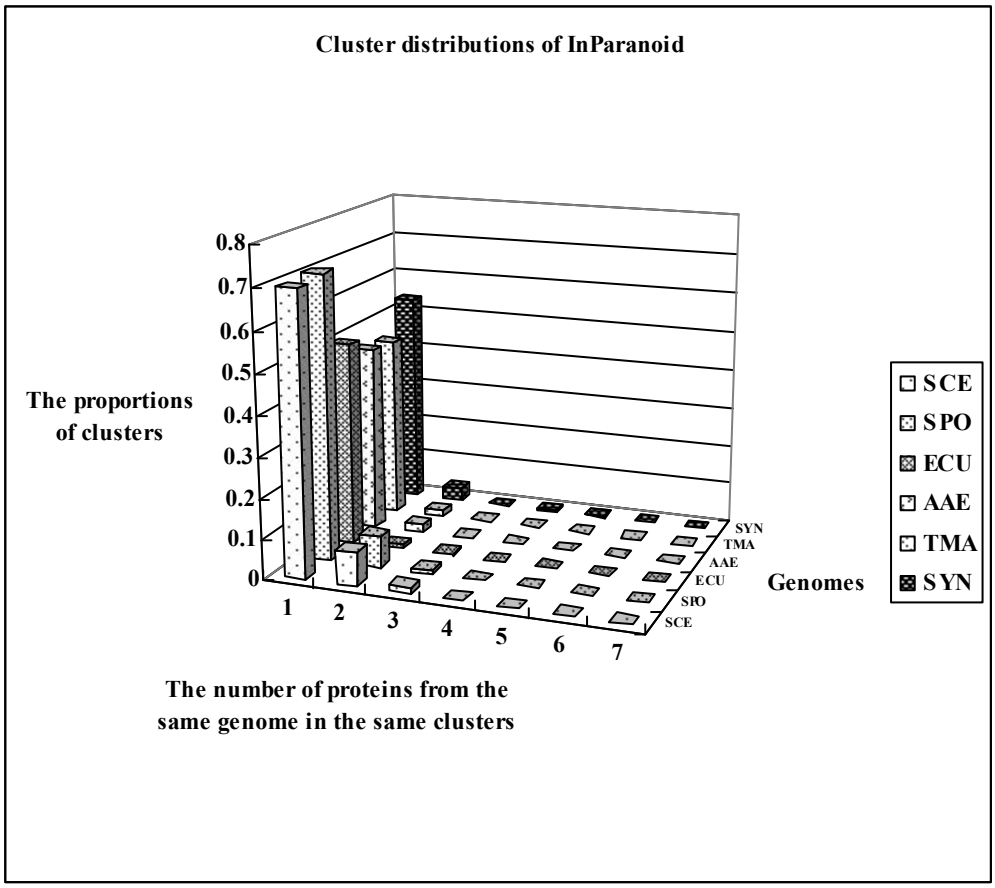


Figure 5.7 The distribution of ortholog clusters for each of six genomes with respect to the number of proteins that are coming from the same genome in the same clusters. This graph shows the distribution of clusters from InParanoid. The tall bar in the front left corner (for SCE genome) represents the proportions of clusters where only one protein from SCE is included. Similarly, the next bar at x = 2, y = SCE, represents the proportion of clusters where two SCE proteins are included. InParanoid has 70% of clusters, 826 out of 1187 clusters, which inlcude one SCE protein in them. In the case of clusters with two SCE proteins, InParanoid has 8.4% of clusters.

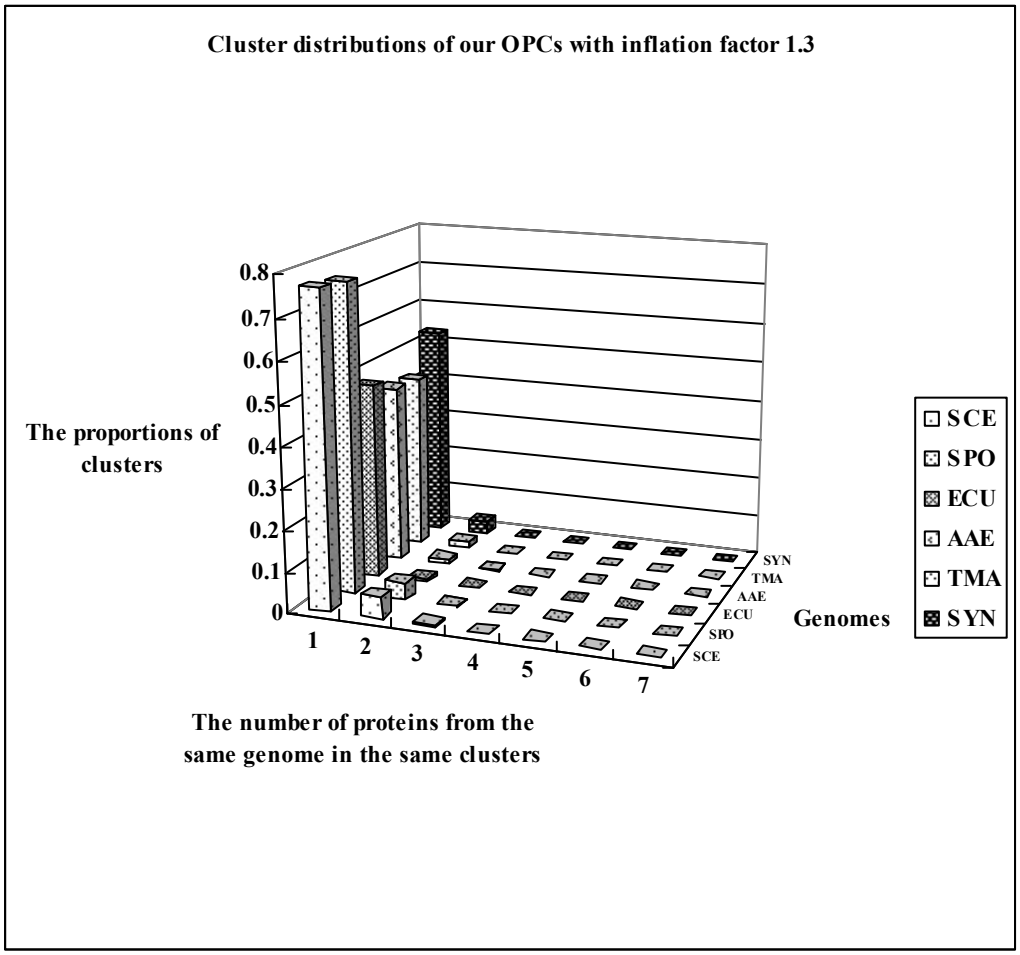


Figure 5.8 The distribution of clusters from our OPCs with inflation factor 1.3. In case of clusters with one SCE protein, our method has 77% clusters. In the clusters with two SCE proteins, it has 5.4% of clusters. It is desirable to have smaller numbers of proteins from the same genome in each cluster as possible because multiple proteins from the same genome that are clustered together can be paralogs. With that respect, our method produced in a sense purer OPCs than InParanoid.

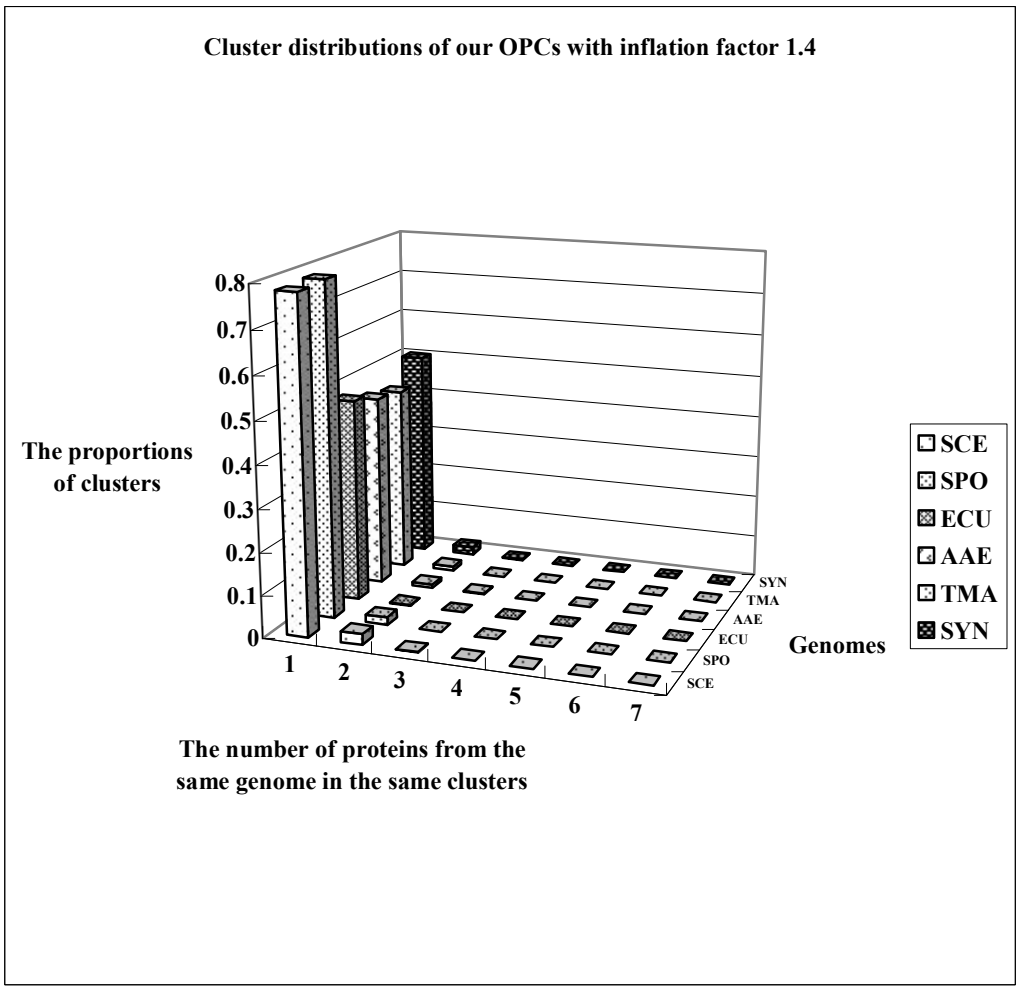


Figure 5.9 The distribution of clusters from our OPCs with inflation factor 1.4. In case of clusters with one SCE protein, our method has 78% clusters. In the clusters with two SCE proteins, it has 2.5% of clusters.

Figure 5.10-11 show the distribution of orthologs found in the generated OPCs considering only the clusters that are identical to that of KO OPCs and the numbers of such clusters produced. With the inflation factor of 1.3, our method produced 1817 orthologs in 896 clusters identical to KO OPCs while InParanoid produced 1555 proteins and 681 clusters, respectively.

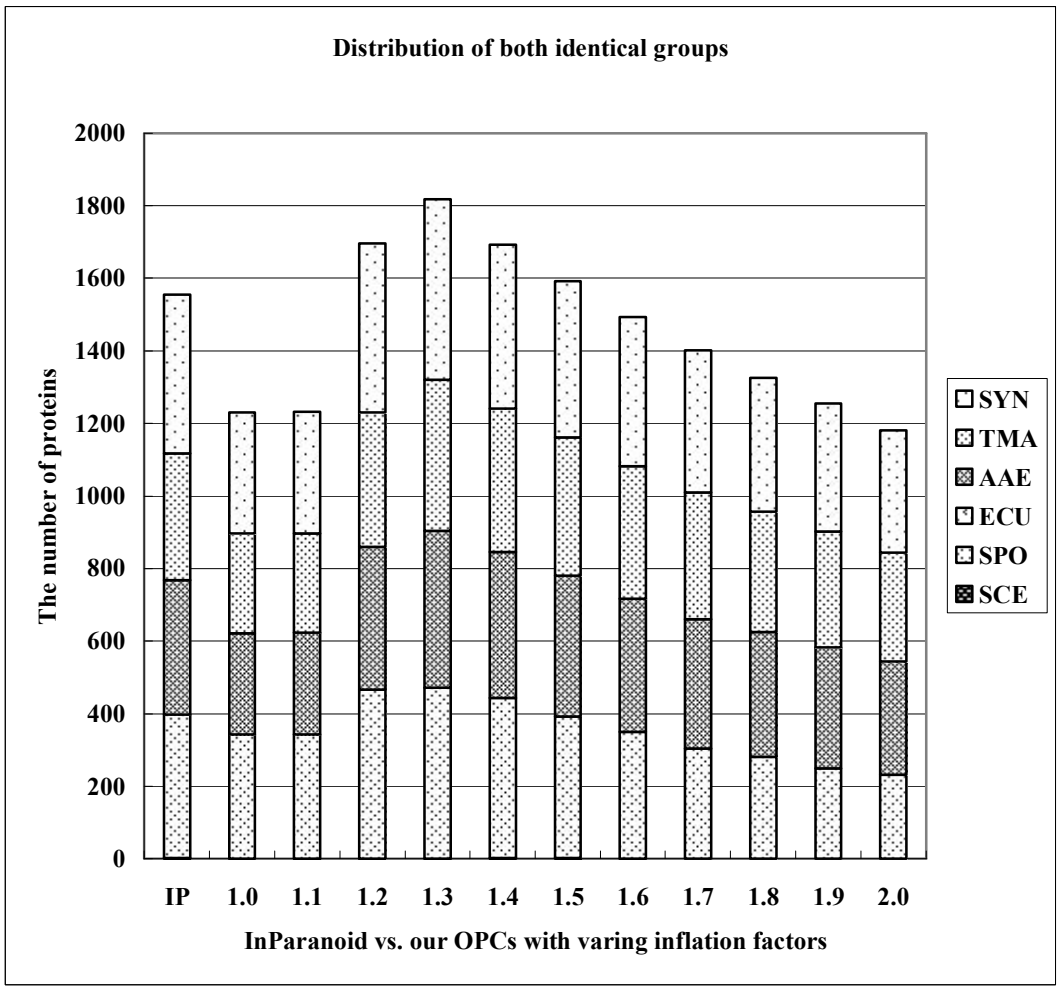


Figure 5.10 The distribution of the orthologs of clusters that are identical to KO OPCs. This graph shows the distribution of orthologs found in the generated OPCs considering only the clusters that are identical to that of KO OPCs. With the inflation factor of 1.3, our method produced 1817 orthologs identical to KO OPCs while InParanoid produced 1555 proteins.

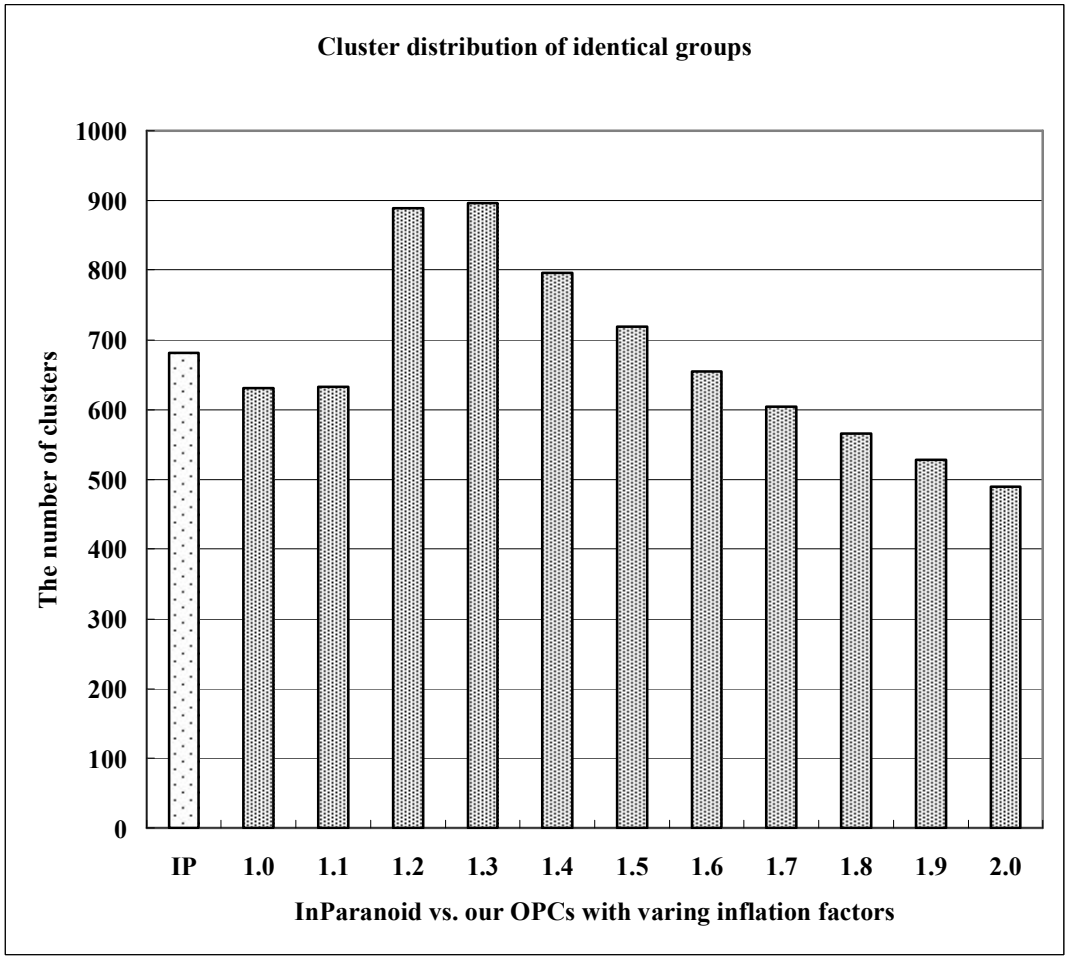


Figure 5.11 The distribution of ortholog clusters that are identical to KO OPCs. This graph shows the distribution of the clusters that are identical to that of KO OPCs. With the inflation factor of 1.3, our method produced 896 clusters identical to KO OPCs while InParanoid produced 681 clusters.

Chapter 6

Conclusion

In this thesis, we have studied how to construct OPCs from multiple species. We here suggested a mathematical framework to represent ortholog clusters systematically. We also introduced an iterative method that cluster orthologs from multiple complete genomes. In addition, we introduced a parallel method to cluster orthologs from multiple genomes to improve performance of the iterative method. We, especially, suggested a new automatic clustering method to cluster orthologs without thresholding, and improved the accuracy to detect orthologs by Markov matrices with self-sequence similarity scores. We also proposed a new intuitive similarity measure to compare two groups. In section 6.1, we summarize and discuss our results. We discuss our future work in section 6.2.

6.1 Summary of Results

The iterative algorithm has the time complexity of $O(lmn^4)$ for genomes n, and the numbers of unit setor pairs, l and m, in the total table and the normal table, respectively. The time complexity of the parallel algorithm is $O(N^2/P)$, where N is the number of unit setor pairs in the total table and P is the number of processors. The recursive search algorithm also shows similar performace to the iterative method.

On the orther hand, the quality of ortholog clusters is largely dependent on the results of the reciprocal BLAST hits among genomes. The BLAST algorithm is very efficient and fast, but it is very difficult to get optimal solution among distant phylogenetic species because the genomes with large evolutionary distance typically have low similarity in their protein sequences. In order to reduce the false positives in the OPCs, thresholding is often employed on the BLAST score. However, the thresholding eliminates large numbers of true positives as the orthologs from distant species likely have low BLAST scores.

In order to rectify this problem, we introduced a new OPC method that does not use BLAST thresholding. Our method employs MCL algorithm to compute the clusters and refines the clusters by adjusting the inflation factor. Our goal is to find more numbers of orthologs from distant species while not sacrificing the accuracy of the result. In fact, it is possible to achieve the goal since the ability to detect orthologs is quite well improved by using the Markov matrices including self-sequence similarity scores.

We tested our method using six different genomes and compared the result to KO OPCs. KO OPCs are generated from manually curated known pathways. Its coverage is small but is highly accurate. We used it as gold standard to measure the accuracy of the results from our method as well as the results from InParanoid. We showed that our method produced

54% more numbers of orthologs than InParanoid while keeping the accuracy similar to that of InParanoid. We also introduced a new intuitive similarity measure based on our least-move algorithm for quantifying the similarity between the OPCs.

6.2 Future Directions

We will address the following work. First, we will construct our ReMark database for storing and managing OPCs extracted from many diverse species. We will also make the ReMark-DB web interface which provides the list of member proteins and a multiple sequence alignment, a statistical summary and graphical view of similarities, and a graphical representation of domain architecture. Second, we will develop a method to include additional genome sequences. When considering the processes for constructing OPCs, we can see that it is not trivial. Finally, we will improve the accuracy of our OPCs as considering the evolutionary distance of each species.

References

[Alex06] A. Alexeyenko, I. Tamas, G. Liu and E. L. Sonnhammer, "Automatic clustering of orthologs and inparalogs shared by multiple proteomes," Bioinformatics, Vol. 22, pp. e9-15.

[Alts] http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html

[Alts90] S. F. Altchul, R. J. Carrol and D. J. Lipman, "Basic local alignment search tool," J. Mol. Biol., Vol. 215, pp. 403-410, 1990.

[Alts91] S. F. Altschul, "Amino acid substitution matrices from an information theoretic perspective," J. Mol. Biol., Vol. 219, pp. 555-565, 1991.

[Alts97] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller and D. J. Lipman, "Gapped Blast and PSI-Blast: a new generation of protein database search programs," Nucleic Acids Research, Vol. 25, pp. 3389-3402, 1997.

[Bono98] H. Bono, S. Goto, W. Fujibuchi, H. Ogata and M. Kanehisa, "Systematic Prediction of Orthologous Units of Genes in the complete Genomes," Genome Inform Ser Workshop Genome Inform., Vol. 9, pp. 32-40, 1998.

[Bork98] P. Bork and E. V. Koonin, "Predicting functions from protein sequence: Where are the bottlenecks?" Nat. Genet., Vol. 18, pp. 313-318, 1998.

[Chen06] F. Chen, A. J. Mackey, C. J. Stoeckert and D. S. Roos, "OrthoMCL-DB: querying a comprehensive multi-species collection of ortholog groups," Nucleic Acids Res., Vol. 34, pp. D363-8, 2006.

[Cher98] S. A. Chervitz, L. Aravind, G. Sherlock, C. A. Ball, E. V. Koonin, S. S. Dwight, M. A. Harris, K. Dolinski, S. Mohr, T. Smith, S. Weng, J. M. Cherry and D. Botstein, "Comparison of the complete protein set of worm and yeast:orthology and divergence," Science, Vol. 282 , pp. 2022-2028, 1998.

[Dayh78] M. O. Dayhoff, R. M. Schwartz and B. C. Orcutt, "A model of evolutionary change in proteins," In "Atlas of Protein Sequence and Structure," Vol. 5, Suppl. 3 (ed. M.O. Dayhoff), pp. 345-352, Natl. Biomed. Res. Found., Washington, DC, 1978.

[Demb94] A. Dembo, S. Karlin and O. Zeitouni, "Limit distribution of maximal non-aligned two-sequence segmental score," Ann. Prob., Vol. 22, pp. 2022-2039, 1994.

[Eise98] J. A. Eisen, "Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis," Genome Res., Vol. 8, pp. 163-167, 1998.

[Enri02] A. J. Enright, S. Van Dongen, C. A. Ouzounis, “An efficient algorithm for large-scale detection of protein families,” Nucleic Acids Res., Vol. 30, pp. 1575-1584, 2002.

[Fitc00] W. M. Fitch, “Homology, a personal view on some of the problems,” Trends Genet., Vol. 16, pp. 227-231, 2000.

[Fitc70] W. M. Fitch, “Distinguishing homologous from analogous proteins,” Syst. Zool., Vol. 19, pp. 99-113, 1970.

[Galp98] M. Y. Galperin and E. V. Koonin, “Source of systematic error in functional annotation of genomes: domain rearrangement, nonorthologous gene displacement and operon disruption,” In Silico Biol., Vol. 1, pp. 55-67, 1998.

[Gate93] J. Gatesy, R. DeSalle and W. Wheeler, “Alignment-ambiguous nucleotide sites and the exclusion of systematic data,” Molecular Phylogenetics and Evolution, Vol. 2, pp. 152-157, 1993.

[Gonn92] G. H. Gonnet, M. A. Cohen and S. A. Benner, “Exhaustive matching of the entire protein sequence database,” Science, Vol. 256, pp. 1443-1445, 1992.

[Gumb58] E. J. Gumbel, “Statistics of extremes,” Columbia Universtiy Press, New York, NY, 1958.

[Heni92] S. Henikoff and J. G. Henikoff, “Amino acid substitution matrices from protein blocks,” Proc. Natl. Acad. Sci. USA, Vol. 89, pp. 10915-10919, 1992.

[Heni93] S. Henikoff and J. G. Henikoff, “Performance evaluation of amino acid substitution matrices,” Proteins, Vol. 17, pp. 49-61, 1993.

[Hill94] D. M. Hillis, “In Homology: The hierarchical basis of comparative biology,” Academic Press, San Diego, CA, pp. 339-368, 1994.

[Jone92] D. T. Jones, W. R. Taylor and J. M. Thornton, “The rapid generation of mutation data matrices from protein sequences,” Comput. Appl. Biosci., Vol. 8, pp. 275-282, 1992.

[Kane03] M. Kanehisa, B. Peer, “Bioinformatics in the post-sequences era,” nature genetics supplement, Vol. 33, pp. 305-310, 2003.

[Karl90] S. Karlin, and S. F. Altschul, “Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes,” Proc. Natl. Acad. Sci. USA, Vol. 87, pp. 2264-2268, 1990.

[Kim06] S. Kim, K. S. Jung and K. H. Ryu, “Automatic Orthologous-Protein-Clustering from Multiple Complete-Genomes by the Best Reciprocal BLAST Hits,” In Proc. of PAKDD 2006 Workshop, BioDM 2006, Vol. 3916, pp. 60-70, 2006.

[Kimm04] S. Kimmen, “Phylogenomic inference of protein molecular function: advances and challenges,” Bioinformatics, Vol. 20, pp. 170-179, 2004.

[Kuma94] V. Kumar, A. Grama, A. Gupta and G. Karypis, “Introduction to Parallel computing,” The Benjamin/Cummings Publishing Company, Inc., 1994.

[Li03] L. Li, C. J. Stoeckert and D. S. Roos, “OrthoMCL: Identification of Ortholog Groups for Eukaryotic Genomes,” Genome Res. Vol. 13, pp. 2178-89, 2003.

[Mush98] A. R. Mushegian, J. R. Garey, J. Martin and L. X. Xiu, “Large-scale taxonomic profiling of eukaryotic model organisms: a comparison of orthologous proteins enclosed by the human, fly, nematode, and yeast genomes,” Genome Res., Vol. 8, pp. 590-598, 1998.

[Over92] J. Overington, D. Donnelly, M. S. Johnson, A. Sali and T. L. Blundell, “Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds,” Prot. Sci., Vol. 1, pp. 216-226, 1992.

[Pear88] W. R. Pearson and D. J. Lipman, ”Improved tools for biological sequence comparison,” Proc. Natl. Acad. Sci. USA, Vol. 85, pp. 2444-2448, 1988.

[Pear95] W. R. Pearson, “Comparison of methods for searching protein sequence databases,” Prot. Sci., Vol. 4, pp. 1145-1160, 1995.

[Reec87] G. R. Reeck, C. de Haen, D. C. Teller, R. E Doolittle, W. M. Fitch, R. E. Dickerson, P Chambon, A. D. McLachlan, E. Margoliash, T H. Jukes and E. Zukerkandl, “Homology in proteins and nucleic acids: A terminology muddle and a way out of it,” Cell, Vol. 50, 667, 1987.

[Remm01] M. Remm, C. E. Storm, E. L. Sonnhammer, “Automatic Clustering of Orthologs and in-paralogs from Pairwise Species Comparisons,” J. Mol. Biol., Vol. 314, pp. 1041-1052, 2001.

[Rubi00] G. M. Rubin, M. D. Yandell, J. R. Wortman, G. L. Gabor Miklos, C. R. Nelson, I. K. Hariharan, M. E. Fortini, P. W. Li, R. Apweiler, W. Fleischmann, J. M. Cherry, S. Henikoff, M. P. Skupski, S. Misra, M. Ashburner, E. Birney, M. S. Boguski, T. Brody, P. Brokstein, S. E. Celniker, S. A. Chervitz, D. Coates, A. Cravchik, A. Gabrielian, R. F. Galle, W. M. Gelbart, R. A. George, L. S. Goldstein, F. Gong, P. Guan, N. L. Harris, B. A. Hay, R. A. Hoskins, J. Li, Z. Li, R. O. Hynes, S. J. Jones, P. M. Kuehl, B. Lemaitre, J. T. Littleton, D. K. Morrison, C. Mungall, P. H. O'Farrell, O. K. Pickeral, C. Shue, L. B. Vosshall, J. Zhang, Q. Zhao, X. H. Zheng and S. Lewis, “Comparative genomics of the eukaryotes,” Science, Vol. 287, pp. 2204-2215, 2000.

[Schw78] R. M. Schwartz and M. O. Dayhoff, “Matrices for detecting distant relationships,” In “Atlas of Protein Sequence and Structure,” Vol. 5, Suppl. 3 (ed. M.O. Dayhoff), pp. 353-358, Natl. Biomed. Res. Found., Washington, DC, 1978.

[Smit81] T. F. Smith and M. S. Waterman, “Identification of Common Molecular Subsequences,” Journal of Molecular Biology, Vol. 147, pp. 195-197, 1981.

[Swof96] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis, “Phylogenetic inference,” Molecular Systematics (2nd ed.), Sinauer Associates, Sunderland, Massachusetts, pp. 407-514, 1996.

[Tatu00] R. L. Tatusov, M. Y. Galperin, D. A. Natale and E. V. Koonin, “The COG database: a tool for genome-scale analysis of protein functions and evolution,” Nucleic Acids Res., Vol. 28, pp. 33-36, 2000.

[Tatu97] R. L. Tatusov, E. V. Koonin and D. J. Lipman, “A genomic perspective on protein families,” Science, Vol. 278, pp. 631-637, 1997.

[Whee99] S. J. Wheelan, M. S. Boguski, L. Duret and W. Makałowski, “Human and nematode orthologs – lessons from the analysis of 1800 human genes and the proteome of Caenorhabditis elegans,” Gene, Vol. 238, pp. 163-170, 1999.

[Wilb85] W. J. Wilbur, “On the PAM matrix model of protein evolution,” Mol. Biol. Evol., Vol. 2, pp. 434-447, 1985.

[Yona99] G. Yona, “Methods for global organization of the protein sequence space,” Ph. D. thesis, 1999.

http://biozon.org/people/golan/

여러 종 사이에서 올소로그 군집기법

김 선 신

충북대학교 대학원 전자계산학과 전자계산학 전공
(지도교수 이 충 세, 류 근 호)

요 약

1980 년대 초반부터 분자서열 데이터는 기하급수적으로 증가하여 왔다. 그러나 대부분의 단백질의 기능은 밝혀지지 못하고 있다. 더욱이 생물학 실험을 통해서 단백질의 기능을 알아내는 것은 많은 시간과 노력을 필요로한다. 따라서 이미 기능을 알고 있는 단백질로부터, 서열상동성에 기반하여, 기능을 모르는 단백질의 기능을 예측하는 일은 생명정보학에서 매우 중요하다.

이 논문에서 우선 올소로그군집을 체계적으로 표현하기 위해 수학적 구조를 제안한다. 여기서 “setor”라는 수학적 개념을 정의한다. 이 개념은 단백질의 고유한 서열유사도 스코어를 함유한 단백질의 집합을 의미한다. 다음으로는 여러 유전체로부터 동일한 기능을 하는 올소로그를 묶어주는 반복 알고리즘을 소개한다. 이 알고리즘은 단지 두개의 유전체로부터 올소로그를 묶어주는 InParanoid 를 여러유전체로 확장한다. 우리는 또한 이 반복기법의 수행속도를 증가시키기 위한 병렬기법을 소개한다.

한편, 올소로그 군집의 질은 유전체사이의 BLAST 알고리즘의 상호최대 히트

에 대개 의존한다. 이 BLAST 알고리즘은 매우 빠르고 효율적이지만, 진화적거리가 먼 종사이에서는 최적의 결과를 생산하기가 매우 어렵다. 왜냐하면 진화적거리가 먼 유전체는 단백질 서열의 유사도가 낮기 때문이다. 이로인해서, 올소로그군집에서는 거짓-긍정(False Positive)를 줄이기 위해 종종 한계값(thresholding)이 사용된다. 하지만, 이 한계값의 사용은 많은 수의 참-긍정(True Positive)인 올소로그를 제거하게 된다.

따라서 BLAST 한계값을 사용하지 않고, 이문제를 해결하기 위해 순환기법과 MCL 알고리즘을 결합한 새로운 기법을 도입한다. 특히, 자신의 서열 유서도 점수(self-sequence similarity scores)를 Markov 매트릭스에 추가하여, 올소로그를 탐색하는 정확도를 증진 시킨다. 제시한 방법은 우선 n 개의 관심있는 유전체를 선정한다. 다음으로는 n 개의 유전체사이에 상호최대 BLAST 히트를 찾아서 모든 가능한 올소로그쌍을 생성한다. 이제 우리의 순환 알고리즘을 사용하여 스코어 매트릭스를 생성한다. 다음 단계에서는 MCL 알고리즘을 사용하여 묶여진 군집을, 팽창인자(Inflation factor)를 조절하여, 더 세분하여 정제한다. 우리의 방법은 여섯개의 서로 다른 종을 사용하여 실험하였다. 실험결과는 KO 군집 (정확하게 기능별로 분류된 군집)과 비교하여 평가하였다. 또한 InParanoid 결과를 기준선으로 두고 우리의 기법을 평가하였다. 우리의 결과는 InParanoid 결과와 정확도가 비슷한 수준을 유지하면서 InParanoid 보다 54% 더 많은 올소로그를 생산하였다. 한편, 올소로그 군집사이의 유사도를 양적으로 비교하기 위해, 새로운 직관적인 유사도 비교 방법을 소개하였다.

Acknowledgements

First of all, I would like to thank my advisors, Professor Chung Sei Rhee and Professor Keun Ho Ryu, of computer science. Prof. Rhee has helped me to study fault-tolerant system and parallel computing during mostly the first stage of the Ph.D. program in this school. Prof. Ryu has supported me to not only prepare this dissertation but also explore diverse fields in bioinformatics during the second stage.

I would next like to thank Professor Seung Kee Han of physics. He helped me to study various fields in physics, the experience of which assists me to look into the other fields. We have sometimes taken discussion about my work together.

I would also like to thank Professor Yong Je Chung and Professor Young Chang Kim of life science. Prof. Chung has helped me to understand biochemical or biological meanings of terminology and issues in bioinformatics, and has discussed the several problems of the field with me. Prof. Kim guided me into the first step of this thesis.

I am thankful to Professor Jaewoo Kang of computer science at the Korea University. We discussed about the problems of my work and his comments helped me to advance my study.

I am grateful to Professor Jinyan Li in the Institute for Infocomm Research in Singapore. He reviewed my paper and gave good comments to improve the quality of the paper.

I would like to thank all of my colleagues, Kwang Su Jung, Namsrai Oyun-Erdene, Ho Sun Son, Ki Jin Yu, Maya Sharma, Heon Gyu Lee, Hong Kyu Park, Bum Ju Lee, and Daesung Kim. We have discussed about diverse issues in bioinformatics, and they have helped me to realize what problems are there.

I would like to thank my parents and parents-in-law for their support for so long time. I

also would like to thank my wife, Jeong bin Choi, for her patience, and thank my daughter, Yearim, for giving bright sunshine in my life.

Specially, I am deeply grateful to all the people that have helped and taught me until now although do not mention the names of those.

August 2007

Sunshin Kim