PROTEINS
STRUCTURE ■ FUNCTION ■ BIOINFORMATICS

Clustering orthologous proteins across phylogenetically distant species

Sunshin Kim,[1] Jaewoo Kang,[2,3]* Yong Je Chung,[4] Jinyan Li,[5] and Keun Ho Ryu[1]

[1] School of Electrical and Computer Engineering, Chungbuk National University, Cheongju, Korea
[2] Department of Computer Science and Engineering, College of Information and Communication, Korea University, Seoul, Korea
[3] Department of Biostatistics, College of Medicine, Korea University, Seoul, Korea
[4] School of Life Sciences, Chungbuk National University, Cheongju, Korea
[5] School of Computer Engineering, Nanyang Technological University, Nanyang, Singapore 639798

ABSTRACT

The quality of orthologous protein clusters (OPCs) is largely dependent on the results of the reciprocal BLAST (basic local alignment search tool) hits among genomes. The BLAST algorithm is very efficient and fast, but it is very difficult to get optimal solution among phylogenetically distant species because the genomes with large evolutionary distance typically have low similarity in their protein sequences. To reduce the false positives in the OPCs, thresholding is often employed on the BLAST scores. However, the thresholding also eliminates large numbers of true positives as the orthologs from distant species likely have low BLAST scores. To rectify this problem, we introduce a new hybrid method combining the Recursive and the Markov CLuster (MCL) algorithms without using the BLAST thresholding. In the first step, we use InParanoid to produce n(n−1)/2 ortholog tables from n genomes. After combining all the tables into one, our clustering algorithm clusters ortholog pairs recursively in the table. Then, our method employs MCL algorithm to compute the clusters and refines the clusters by adjusting the inflation factor. We tested our method using six different genomes and evaluated the results by comparing against Kegg Orthology (KO) OPCs, which are generated from manually curated pathways. To quantify the accuracy of the results, we introduced a new intuitive similarity measure based on our Least-move algorithm that computes the consistency between two OPCs. We compared the resulting OPCs with the KO OPCs using this measure. We also evaluated the performance of our method using InParanoid as the baseline approach. The experimental results show that, at the inflation factor 1.3, we produced 54% more orthologs than InParanoid sacrificing a little less accuracy (1.7% less) than InParanoid, and at the factor 1.4, produced not only 15% more orthologs than InParanoid but also a higher accuracy (1.4% more) than InParanoid.

Proteins 2008; 71:1113–1122.

Key words: orthologs; species; genomes; clustering; BLAST; proteins; paralogs; homologs; database; threshold.

INTRODUCTION

Since orthologs were first suggested by Fitch,[1] several ortholog clusters have been introduced independently and used successfully for comparative genomics and genome annotation. One of such examples is the COG (clusters of orthologous group) database[2–4] in NCBI (National Center for Biotechnology Information), which is constructed from 66 complete genomes using the gapped BLAST program.[5] BLAST is a very efficient and fast heuristic method,[6–10] but it has been known that it is often ineffective to detect the homology between evolutionarily distant species.[11–14] Genes with homology are called homologs. There are two types of homologs. One is orthologs that are genes speciated from the same ancestor gene preserving the same function. The other is paralogs, genes related by duplication but having different functions.[1,15]

The COG database released in 2000[3] were constructed from 21 complete genomes. It first identifies groups of three proteins with best reciprocal BLAST hits, and then performs case-by-case manual analysis to eliminate false-positives. Unfortunately, some of the resulting clusters include large numbers of paralogs from the same lineage. Such clusters are not very useful for predicting functions for newly sequenced genes because the clusters' functional coherence is low as many genes from the same genome are included.

Grant sponsors: the Korea Research Foundation, Korean Government (MOEHRD/BITRC); Korea University; MS Bioinformatics; the Korea Science and Engineering Foundation (KOSEF); the Korean government (MOST); Grant number: R01-2007-000-10926-0.
*Correspondence to: Jaewoo Kang, Department of Computer Science and Engineering, College of Information and Communication, Korea University, Seoul, Korea.
E-mail: kangj@korea.ac.kr
Received 14 April 2007; Revised 27 July 2007; Accepted 22 August 2007
Published online 14 November 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/prot.21792

Another example of OPCs is KO, originally introduced in 1998[16] and recently updated.[17,18] It is a database with the ortholog group tables, which contains orthologous genes extracted from metabolic and regulatory pathways. The ortholog groups are a curated reference data set of orthologous relations. The clusters in KO are with high accuracy, as they were classified according to the known functions of proteins and they were manually edited from the pathways that clearly show the functional relations of proteins involved. However, KO contains only limited sets of orthologs because of the small number of known pathways.

To address these problems, Remm *et al.*[19] introduced InParanoid, a fully automatic program, to detect orthologs and inparalogs between two species. It mostly focuses on eukaryotes, and unlike the previous approaches, the resulting clusters include inparalogs that are the recent paralogs sharing the same function because of the duplication happened after speciation. To reduce the number of paralogs in the result, it uses two types of thresholds, *score cut-off* and *overlap cut-off*, which we will explain in detail later. MultiParanoid[20] published recently have extended InParanoid to cluster proteins from multiple proteomes.

Meanwhile, Li *et al.* developed OrthoMCL,[21] which generates OPCs from multiple species using MCL algorithm.[22] The MCL algorithm is known to be effective for detecting protein families especially with complicated domain structures.[23] Very recently, Chen *et al.* constructed an OPC database called OrthoMCL-DB[24] using the 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 50 bits and an overlap cut-off of 50%. OrthoMCL chooses a *P*-value cut-off of 1e-5. Because of this thresholding, however, many legitimate but evolutionarily distant orthologs (hence with low similarity) 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 method for constructing OPCs. In the first step, our method generates $n(n-1)/2$ initial ortholog tables of reciprocal best hits by running InParanoid program for each pair of n genomes. We employ InParanoid to get the initial ortholog tables because it is a fully automated, fast running ortholog detection program. It then generates an augmented table combining the initial ortholog tables and computes a set of initial clusters by merging the gene pairs in the augmented table according to their identifications(IDs). In this step, our method uses a recursive algorithm to cluster orthologs from the augmented tables across distant genomes. It combines all the two-genome ortholog pairs by recursively linking them by their protein IDs, which leads to a set of initial clusters of multi-genome orthologs. In the second step, for each initial cluster, it runs the MCL algorithm to further refine it and obtain the final clusters. Our hybrid method, combining the recursive and the MCL algorithms, 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. Instead, it refines the initial clusters from the recursive algorithm using MCL through adjusting the inflation factor.

A challenging problem 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. MultiParanoid used a small set of manually curated orthologs including species such as human, worm, and fly. OrthoMCL simply compared their results against those of InParanoid and showed that the two results are similar. Recently, Chen *et al.*[25] proposed a new method for evaluating the performance of different orthology detection strategies. To assess the accuracy of strategies, they used a statistical technique called latent class analysis (LCA)[26] to assess the agreement and disagreement of the results obtained from different strategies and use them to determine the statistical confidence of the results. They showed that both OrthoMCL and InParanoid exhibit the best overall performance. Especially, they showed the clusters of OrthoMCL are more consistent with the enzyme commission (EC) assignments, concerning protein function and domain architectures, than KOG (euKaryotic Orthologous Groups).[4] However, the quality of results estimated by LCA is dependent on a specific collection of methods inspected.

To assess the accuracy of our results, we investigated how much our results are consistent with KO that includes diverse ortholog clusters across various distant species. To quantify the similarity between the two groups of OPCs, we introduced a distance measure called *least-move distance*, which computes the number of moves that genes in one OPCs have to make before the two OPCs become identical.

METHODS

Our approach works in two steps. One is to cluster orthologous proteins recursively using our recursive clustering algorithm from the augmented ortholog table consisting of the all possible $n(n-1)/2$ tables with reciprocal best hits derived by the InParanoid program. The second stage is to split the initial clusters using the MCL algorithm into more refined and tighter clusters. Figure 1 shows the overview of our approach.

In the first stage, we first select n genomes of our interest and prepare their protein sequence sets. Then, the $n(n-1)/2$ tables are drawn from the InParanoid using all possible best reciprocal BLAST hits between each pair of proteomes. Note that in doing so we do not use any threshold in InParanoid. These ortholog tables are combined into one augmented table and initial clustering is

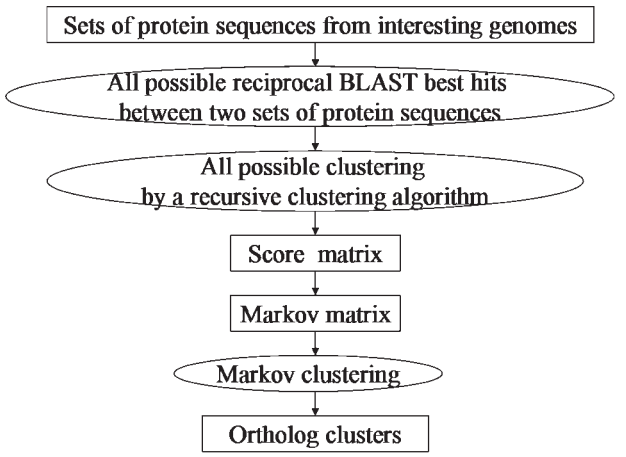


Figure 1
The overall steps of our approach.

performed by running the recursive clustering algorithm over the augmented table as illustrated in Figure 2.

In the second stage, score matrices are first constructed from the initial clusters produced in the previous step as explained in Clusters and score matrices Section. The score matrices are transformed later into Markov matrices to simulate random walks on a graph. Then, the Markov clustering is executed to split the initial clusters into more consistent ones using the MCL algorithm. Finally, the terminal ortholog clusters are produced.

A recursive clustering algorithm

The COG method[2] detects triangles formed from protein lines (pairs) with reciprocal best hits among genomes and merges triangles with a common side of a protein line (pair) through biological analysis without an arbitrary threshold. The InParanoid algorithm,[19] with a score cut-off of 50 bits and an overlap cut-off of 50%, decides a main protein pair of *a* and *b*, fixed as a center point, from which additional inparalogs are clustered. MultiParanoid[20] searches and merges gene pairs with an identical gene in ortholog tables drawn from three genomes of human, worm, and fly. The MultiParanoid extends InParanoid to multiple genomes by combining the results from InParanoid over multiple pairs of genomes and thus the overall quality of the result is dependent on that of InParanoid.[20]

In the previous work,[27] we proposed an automatic method that clusters orthologous proteins from $n(n-1)/2$ ortholog tables generated from n genomes using InParanoid. But the algorithm is somewhat complex and difficult to understand. A simple and clear method is proposed here. This algorithm starts with the augmented table combininig $n(n-1)/2$ ortholog tables, and recursively detects and merges gene pairs with identical genes. Both the algorithms correspond to a single-linkage method. Figure 2 shows the algorithm.

Clusters and score matrices

Recall that we did not use the thresholds in the first step to generate the initial clusters. Our recursive 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. 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. The key idea of it[22] is to simulate flow within a graph such that the flow is encouraged when the current is strong, but discouraged when the current is weak. The inflation factor is used for both strengthening and weakening the current, and thus the cluster granularity can be controlled by the inflation factor. Large inflation factor generates large numbers of small clusters while small inflation factor generates small numbers of large clusters.

To apply the MCL algorithm, the initial clusters are transformed into score matrix as shown in Figure 3. Unlike OrthoMCL,[21] we make use of diagonal scores that are self-reciprocal best hit scores computed against themselves. A node with a high return weight (i.e., a large diagonal score) will likely be an attractor with a positive return probability. It has less effect on cluster granularity than the inflation factor, but needs also to be considered as it influences the quality of the final clusters generated. We observed in our experiment that more reliable clusters were generated with diagonal scores than without. OrthoMCL also normalized the weights to minimize the impact of inparalogs. We do not consider normalization since we do not include inparalogs in our result.

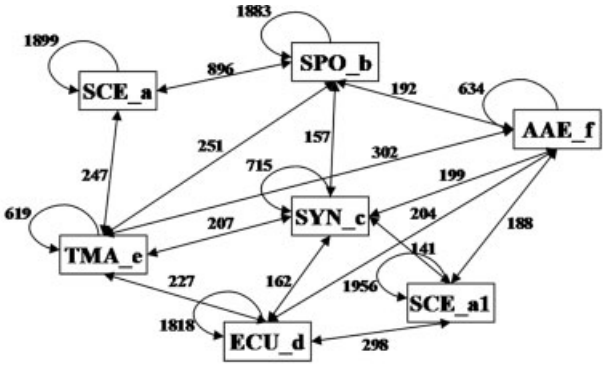
We obtain a group of protein clusters after running MCL with our score matrix. As done in COG, we only include the clusters having three or more proteins in the final result. Now, let us consider an instance to show how the graph in Figure 3 is partitioned after applying MCL. As shown in Table I, 11 OPC groups are produced according to varying inflation factors from 1.0 to 2.0, which are represented as OPC10 through OPC20. Each OPC group has clusters such as if10oc1 in case of OPC10, and as if14oc1 and if14oc2 in case of OPC14. At inflation factor 1.4, the graph is split into two clusters. The if14oc1 cluster includes three proteins denoted SCE_a, SPO_b, and TMA_e, where the capital letters represent the names of species and the lowercase letters indicate proteins. At inflation factor 1.8, another split occurs resulting three clusters in OPC18 as shown in Table I.

```
Procedure Clustering Orthologous Proteins
Input: The augmented table with Gene Pairs /* Gene pairs are located in two columns
       /* of the augmented table
Output: The initial OPCs(Orthologous Protein Clusters)
1. Create dictionary data structure of Gene Pairs and Genes /* For arranging the data
   /* according to alphabetical order of data
2. Create array data structure of Genes /* For storing the data of genes
3. Open the augmented table
4. Initialize the Flags of Gene Pairs and Genes /* For checking whether the data are
   /* clustered or not
5. for i = 1 to the total number of lines do
6.     Call ID_SEARCH(the gene) for a gene of i line
7. end for
```

```
Sub_procedure ID_SEARCH(n)
1. Open the augmented table
2. for i = 1 to the total number of lines do /* There are two parts for searching
     /* the gene ID of n, one of which is looking for the gene equal to n
     /* in the first column with the genes and the other is looking for
     /* the gene equal to n in the second column with the genes in the table
3.   Initialize the array of Genes
4.   if The first gene ID = n then /* Start searching the gene equal to n
       /* in the first column with genes in the table
5.     Save the gene pair in the array of Genes
6.     if Gene Pairs Flag = 0 then
7.       Print the gene pair
8.       Check the Flag of Gene Pairs and Genes
9.       if The second gene Flag = 0 then
10.        ID_SEARCH(the second gene) /* Start searching the gene
           /* in the second column with genes in the table
11.      end if
12.    end if
13.  end if
14.  if The second gene ID = n then /* Start searching the gene equal to n
       /* in the second column with genes in the table
15.    Save the gene pair in the array of Genes
16.    if Gene Pairs Flag = 0 then
17.      Print the gene pair
18.      Check the Flag of Gene Pairs and Genes
19.      if The first gene Flag = 0 then
20.        ID_SEARCH(the first gene) /* Start searching the gene
           /* in the first column with genes in the table
21.      end if
22.    end if
23.  end if
24. end for
```

Figure 2

Our recursive clustering algorithm to cluster orthologous proteins from the augmented table.



	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 3
The graph on the top represents the reciprocal best hits among proteins and the table below shows the corresponding score matrix.

The concept of similarity

To be able 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 4. Figure 4(a) shows two clustering results, K and G. K consists of four clusters, K1–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 4(b) represent zero coherence as no cluster is subset of any other. Intuitively, however, both

Table I
An Example to Show How the Graph in Figure 3 is Partitioned After Applying the MCL Algorithm

OPC10	OPC11	OPC12	OPC13	OPC14	OPC15	OPC16	OPC17	OPC18	OPC19	OPC20
if10oc1	if11oc1	if12oc1	if13oc1	if14oc1	if15oc1	if16oc1	if17oc1	if18oc1	if19oc1	if20oc1
SCE_a SPO_b SYN_c SCE_a1	SCE_a SPO_b SYN_c SCE_a1	SCE_a SPO_b SYN_c SCE_a1	SCE_a SPO_b SYN_c SCE_a1	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e	SCE_a SPO_b TMA_e
ECU_d	ECU_d	ECU_d	ECU_d	if14oc2	if15oc2	if16oc2	if17oc2	if18oc2	if19oc2	if20oc2
TMA_e AAE_f	TMA_e AAE_f	TMA_e AAE_f	TMA_e AAE_f	SYN_c SCE_a1 ECU_d AAE_f	SYN_c SCE_a1 ECU_d AAE_f	SYN_c SCE_a1 ECU_d AAE_f	SYN_c SCE_a1 ECU_d AAE_f	SYN_c ECU_d AAE_f	SYN_c ECU_d AAE_f	SYN_c ECU_d AAE_f
								if18oc3	if19oc3	if20oc3
								SCE_a1	SCE_a1	SCE_a1

Figure 4(a,b) seem to have low similarity and we cannot say clusters in Figure 4(a) is much more coherent than those in Figure 4(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 4(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 4(a,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.

The least-move algorithm

Consider the instance in Figure 5. 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 cluster 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 is 5/8. Figure 6 shows the details of this algorithm.

RESULTS

We used six different genomes in our test, three eukaryotes and three bacteria, as shown in Table II. Figure 7 compares the number of orthologs detected by InParanoid (IP) and our method with varying inflation factors. As explained in Clusters and score matrices Section, 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 means 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 compared our clustering results against the KO clusters.* 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 8 shows the number of proteins that overlap with KO OPCs for each case shown in Figure 7. 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 9 shows the accuracy of the clustering results compared against KO OPCs. We used the similarity measure introduced in The Concept of Similarity Section 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

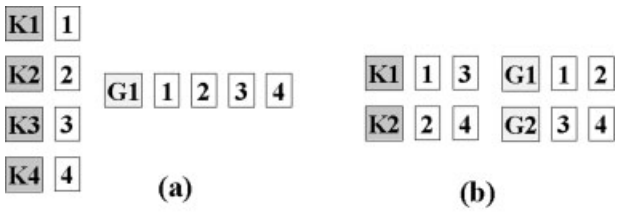


Figure 4
Comparing clustering results: Two groups of clusters, Ks and Gs, are obtained from different OPCs.

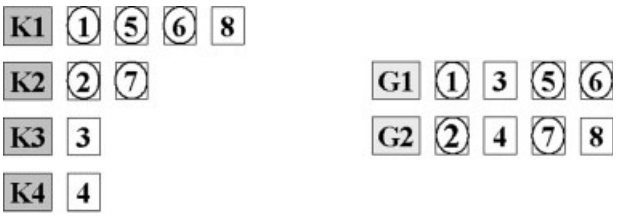


Figure 5
An example for illustrating the least-move algorithm.

*The KO dataset (genes_ko.list) was obtained from ftp://ftp.genome.jp/pub/kegg/linkdb/genes/.

```
Procedure Cal_Similariy /* For calculating similarity between both groups (K group and G group)
Input: Both Groups with same genes /* One group has the same genes (proteins) as the other
Output: The rate of similarity between two groups, K group and G group
1.  Initialize similarity, moves, and genes
2.  Create dictionary data structure of K group and G group /* For arranging genes according to
    /* alphabet order of those
3.  Initialize the Flag_T1 of K group and the Flag_T2 of G group /* For checking whether all
    /* the clusters of both groups are considered
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 /* For arranging clusters
    /* according to alphabetical order or numerical order of those
7.  Create dictionary data structure of the genes for finding the largest cluster /* Through counting
    /* the number of genes of the largest cluster
8.  Create dictionary data structure of the genes for the list moving of K group /* Through counting
    /* the number of moving genes from K group
9.  Create dictionary data structure of the genes for the list moving of G group /* Through counting
    /* the number of moving genes from 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 /* There are three steps for finding the least number of
        /* genes to be moved
13.     for all genes in all clusters do /* The first step starts, in this loop, to find the cluster pair that
            /* shares the most number of genes
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 /* The second step starts
            /* to find the disjoint members of the cluster, belonging to K group,  between two clusters
            /* of K group and G group in the clusters of G group
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 G group do /* The third step starts
            /* to find the disjoint members of the cluster, belonging to G group, between two clusters
            /* of K group and G group in the clusters of K group
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. Print similarity := 1 – moves/genes
```

Figure 6

The least-move procedure to calculate the similarity between two groups of clusters.

and our method are disjoint clusters. To be able to compare them, we transformed KO OPCs to a disjoint one by removing duplicate proteins. As shown in Figure 9, 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

Table II

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

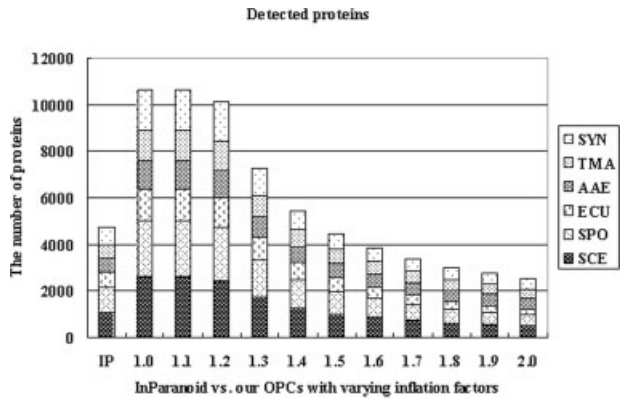


Figure 7

The number of orthologs detected by InParanoid (IP) and our method with varying inflation factors. 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).

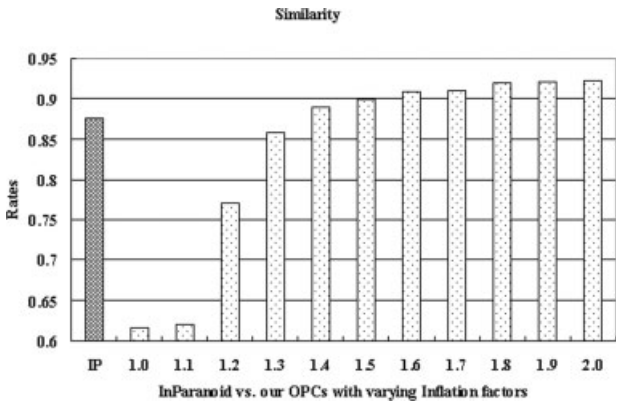


Figure 9

The similarity between KO and InParanoid (IP) and our OPCs with varying inflation factors.

InParanoid (i.e., 7248 vs. 4706) while sacrificing a little less accuracy (1.7% less) than InParanoid. In case of the factor 1.4, our OPCs not only produced 15% more orthologs than InParanoid (i.e., 5415 vs. 4706) but also achieved a better accuracy (1.4% better) than InParanoid.

Figures 10 and 11 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 in Figure 10 has 70% of clusters (826 out of 1187) including only one SCE protein in them. Whereas our method with inflation factor 1.3 and 1.4 has 77 and 78% of clusters respectively as shown in Figure 11. 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. As the inflation factor increased, our method produced increasingly better clusters in terms of the number of clusters having one protein from one genome.

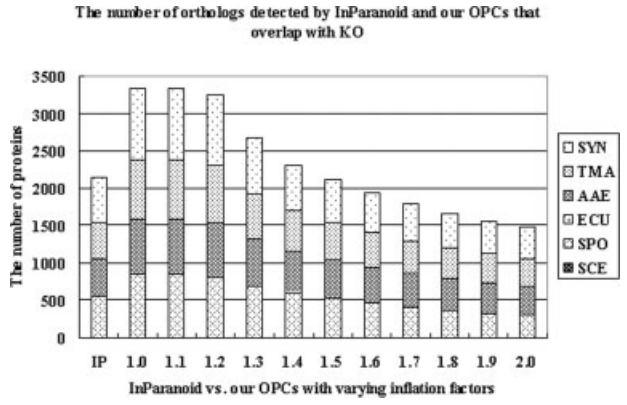


Figure 8

The number of proteins that overlap with KO OPCs for each case shown in Figure 7.

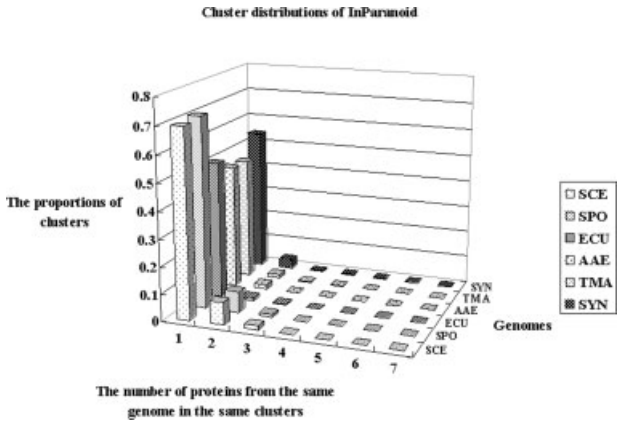


Figure 10

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. The graph shows the distribution of clusters from InParanoid.

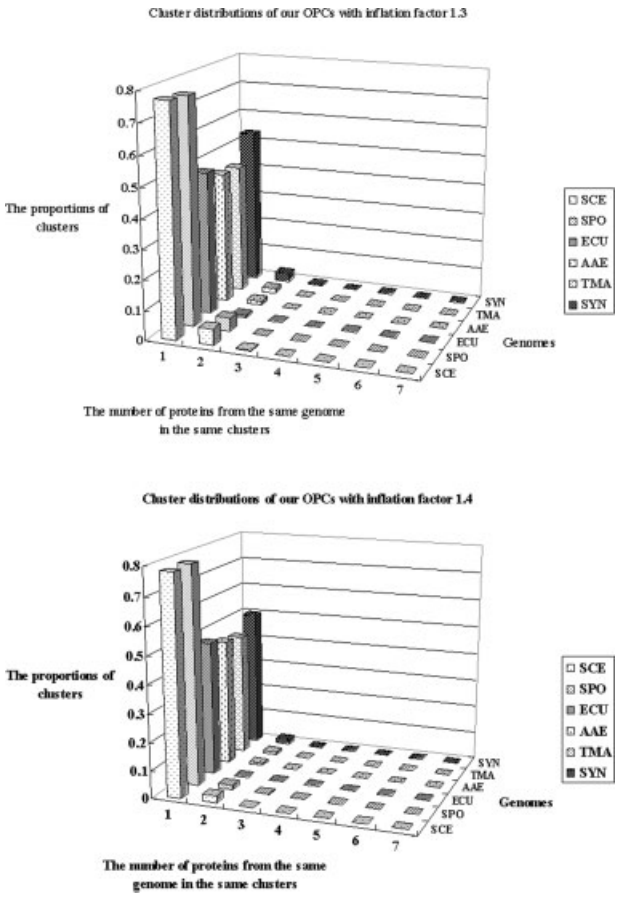


Figure 11

The graphs on the top and bottom show the distribution of clusters from our OPCs with inflation factors of 1.3 and 1.4, respectively.

Figure 12 shows the distribution of orthologs found in the generated OPCs considering only the clusters that are identical to that of KO OPCs (top) and the numbers of such clusters produced (bottom). With the inflation factor of 1.3, our method produced 1817 orthologs in 896 clusters that are identical to KO OPCs while InParanoid produced 1555 proteins in 681 such clusters. All the datasets used in our experiments and the ortholog clusters our method produced are available in http://dblab.chungbuk.ac.kr/~sskim04/.

CONCLUSION

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. 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.

To rectify this problem, we introduced a new OPC method that does not use BLAST thresholding. Our method first constructs an augmented table combining all possible pairwise InParanoid results and build an initial clusters by running our recursive clustering algorithm. It then employs MCL algorithm to further refine the clusters through adjusting the inflation factor. Our aim is to find more numbers of orthologs from distant species while not sacrificing the accuracy of the result.

We tested our method using six different genomes and compared the result to KO OPCs. KO OPCs are generated from manually curated known pathways. We also compared our results with InParanoid. In the previous section, we showed that our method performs significantly better than InParanoid especially in terms of the number of orthologs produced while maintaining the accuracy in the comparable level. In addition, we introduced a new intuitive similarity measure based on our least-move algorithm for quantifying the similarity

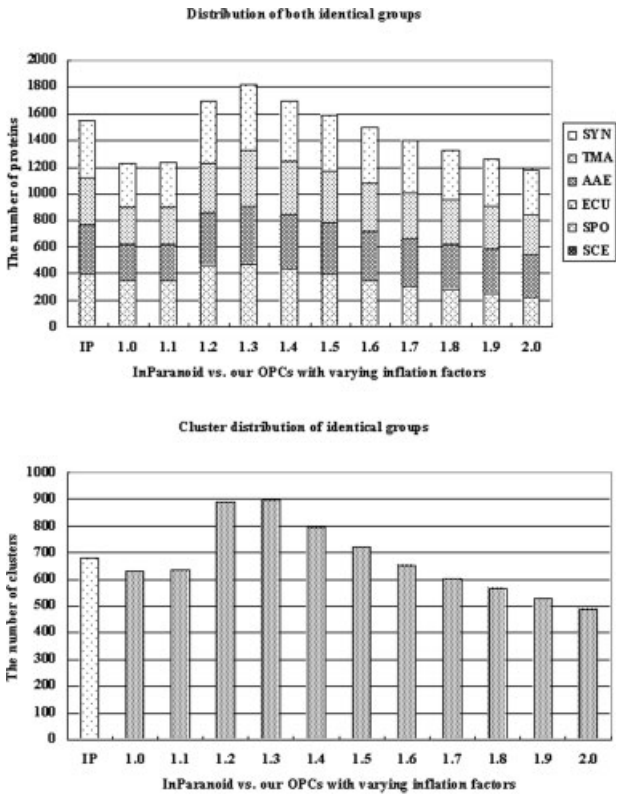


Figure 12

The graph on the top shows the distribution of orthologs considering only the clusters that are identical to KO OPCs. The bottom shows the numbers of such clusters.

between the OPCs. In the future, we plan to build an OPC database extending the clusters generated from our method to include more diverse species.

ACKNOWLEDGMENTS

Some part of the research was done, supported by Chungbuk National University, while Dr. Sunshin Kim was visiting at NC State University.

REFERENCES

1. Fitch WM. Distinguishing homologous from analogous proteins. Syst Zool 1970;19:99–113.
2. Tatusov RL, Koonin EV, Lipman DJ. A genomic perspective on protein families. Science 1997;278:631–637.
3. Tatusov RL, Galperin MY, Natale DA, Koonin EV. The COG database: a tool for genome-scale analysis of protein functions and evolution. Nucleic Acids Res 2000;28:33–36.
4. Tatusov RL, Fedorava ND, Jackson JD, Jacobs AR, Kiryutin B, Koonin EV, Krylov DM, Mazumder R, Mekhedov SL, Nikolskaya AN, Rao BS, Smirnov S, Sverdlov AV, Vasudevan S, Wolf YI, Yin JJ, Natale DA. The COG database: an updated version includes eukaryotes. BMC Bioinformatics 2003;4:41.
5. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol 1990;215:403–410.
6. Chervitz SA, Aravind L, Sherlock G, Ball CA, Koonin EV, Dwight SS, Harris MA, Dolinski K, Mohr S, Smith T, Weng S, Cherry JM, Botstein D. Comparison of the complete protein set of worm and yeast:-orthology and divergence. Science 1998;282:2022–2028.
7. Kanehisa M, Peer B. Bioinformatics in the post-sequences era. Nat Genet Suppl 2003;33:305–310.
8. Mushegian AR, Garey JR, Martin J, Xiu LX. Large-scale taxonomic profiling of eukaryotic model organisms: a comparison of orthologous proteins enclosed by the human, fly, nematode, and yeast genomes. Genome Res 1998;8:590–598.
9. Rubin GM, Yandell MD, Wortman JR, Gabor Miklos GL, Nelson CR, Hariharan IK, Fortini ME, Li PW, Apweiler R, Fleischmann W, Cherry JM, Henikoff S, Skupski MP, Misra S, Ashburner M, Birney E, Boguski MS, Brody T, Brokstein P, Celniker SE, Chervitz SA, Coates D, Cravchik A, Gabrielian A, Galle RF, Gelbart WM, George RA, Goldstein LS, Gong F, Guan P, Harris NL, Hay BA, Hoskins RA, Li J, Li Z, Hynes RO, Jones SJ, Kuehl PM, Lemaitre B, Littleton JT, Morrison DK, Mungall C, O'Farrell PH, Pickeral OK, Shue C, Vosshall LB, Zhang J, Zhao Q, Zheng XH, Lewis S. Comparative genomics of the eukaryotes. Science 2000;287:2204–2215.
10. Wheelan SJ, Boguski MS, Duret L, Makałowski W. Human and nematode orthologs—lessons from the analysis of 1800 human genes and the proteome of Caenorhabditis elegans. Gene 1999;238:163–170.
11. Bork P, Koonin EV. Predicting functions from protein sequence-where are the bottlenecks? Nat Genet 1998;18:313–318.
12. Eisen JA. Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis. Genome Res 1998;8:163–167.
13. Galperin MY, Koonin EV. Source of systematic error in functional annotation of genomes: domain rearrangement, nonorthologous gene displacement and operon disruption. In Silico Biol 1998;1:55–67.
14. Kimmen S. Phylogenomic inference of protein molecular function: advances and challenges. Bioinformatics 2004;20:170–179.
15. Fitch WM. Homology, a personal view on some of the problems. Trends Genet 2000;16:227–231.
16. Bono H, Goto S, Fujibuchi W, Ogata H, Kanehisa M. Systematic prediction of orthologous units of genes in the complete genomes. Genome Inform Ser Workshop Genome Inform 1998;9:32–40.
17. Kanehisa M, Goto S, Kawashima S, Okuno Y, Hattori M. The KEGG resource for deciphering the genome. Nucleic Acids Res 2004;32:D277–D280.
18. Kanehisa M, Goto S, Hattori M, Aoki-Kinoshita KF, Itoh M, Kawashima S, Katayama T, Araki M, Hirakawa M. From genomics to chemical genomics: new developments in KEGG. Nucleic Acids Res 2006;34:D354–D357.
19. Remm M, Storm CE, Sonnhammer EL. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. J Mol Biol 2001;314:1041–1052.
20. Alexeyenko A, Tamas I, Liu G, Sonnhammer EL. Automatic clustering of orthologs and inparalogs shared by multiple proteomes. Bioinformatics 2006;22:e9–15.
21. Li L, Stoeckert CJ, Roos DS. OrthoMCL: identification of ortholog groups for eukaryotic genomes. Genome Res 2003;13:2178–2189.
22. Van Dongen S. Graph clustering by flow simulation. Ph.D thesis 2000, University of Utrecht, The Netherlands.
23. Enright AJ, Van Dongen S, Ouzounis CA. An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res 2002;30:1575–1584.
24. Chen F, Mackey AJ, Stoeckert CJ, Roos DS. OrthoMCL-DB: querying a comprehensive multi-species collection of ortholog groups. Nucleic Acids Res 2006;34:D363–D368.
25. Chen F, Mackey AJ, Vermunt JK, Roos DS. Assessing performance of orthology detection strategies applied to eukaryotic genomes. PLoS ONE 2007;2:e383.
26. Hui SL, Zhou XH. Evaluation of diagnostic tests without gold standards. Stat. Methods Med. Res 1998;7:354–370.
27. Kim S, Jung KS, Ryu KH. Automatic orthologous-protein-clustering from multiple complete-genomes by the best reciprocal BLAST Hits. In Proc PAKDD 2006 Workshop, BioDM 2006 2006;3916:60–70.