Ortholog Clustering on a Multipartite Graph

Akshay Vashist, Casimir A. Kulikowski, and Ilya Muchnik

Abstract—We present a method for automatically extracting groups of orthologous genes from a large set of genomes by a new clustering algorithm on a weighted multipartite graph. The method assigns a score to an arbitrary subset of genes from multiple genomes to assess the orthologous relationships between genes in the subset. This score is computed using sequence similarities between the member genes and the phylogenetic relationship between the corresponding genomes. An ortholog cluster is found as the subset with the highest score, so ortholog clustering is formulated as a combinatorial optimization problem. The algorithm for finding an ortholog cluster runs in time $O(|E| + |V| \log |V|)$, where V and E are the sets of vertices and edges, respectively, in the graph. However, if we discretize the similarity scores into a constant number of bins, the runtime improves to $O(|E| + |V|)$. The proposed method was applied to seven complete eukaryote genomes on which the manually curated database of eukaryotic ortholog clusters, KOG, is constructed. A comparison of our results with the manually curated ortholog clusters shows that our clusters are well correlated with the existing clusters. The reader is advised to compare the "new clustering algorithm" with http://www.datalaundering.com/download/extrem01.pdf

Index Terms—Graph-theoretic methods, clustering algorithms, biology, genetics.

1 Introduction

One of the fundamental problems in comparative genomics is the identification of genes from different organisms that are involved in similar biological functions. This requires identification of orthologs, which are homologous genes that have evolved through vertical descent from a single ancestral gene in the last common ancestor of the considered species [1]. Ortholog detection is fundamental in estimating the trace of vertical evolution of genes. In many cases where the only accessible information on shared characteristics across organisms is genes, orthologs are the only means of studying evolutionary relationships.

Although the degree of sequence similarity between orthologs from different organisms varies and usually depends on the time elapsed since their divergence, such genes usually perform the same function(s) in several organisms [2]. Thus, from an application perspective, ortholog detection is critical for gene function annotation. Orthologous groups of genes can provide anchors for targeting the detection of poorly conserved gene regulatory elements such as promoters and transcription factors [3]. The purpose of detecting orthologs depends on the type of data being studied. For instance, one may wish to "zoom" in on the process of evolution of genes by conducting a study on orthologs in closely related organisms such as different fly (*Drosophila*) genomes [4].

In recent years, many genome-wide ortholog detection procedures have been developed [5], [6], [7], [8], [9], [10]; however, they suffer from limitations that present real challenges for addressing the problem in a large set of genomes. Some of them [5], [6], [7] are limited to identifying orthologs in a pair of genomes, some [10] require phylogenetic information and are not computationally efficient, and others [8], [9] require expert curation. Known complete methods for finding ortholog clusters have at least two stages—automatic and manual. The role of the latter is to correct the results of the first stage, which is usually a clustering procedure. Although specific implementations of clustering procedures in different methods vary, most successful methods include critical steps such as building clusters based on a set of "mutually most similar pairs" of genes from different genomes. These pairs are called BBH (Bi-directional Best Hits [7], [8], [9]). This preprocessing is not robust as small changes in data or in the set of free parameters can alter the results substantially. So, currently, there are three bottlenecks in ortholog extraction: 1) manual curation, 2) time complexity, and 3) hypersensitivity of the automatic stage to parameter changes. We propose a combinatorial optimization approach for ortholog detection in a large set of genomes that addresses these bottlenecks.

We describe an ortholog clustering method where we require that any sequence in an ortholog cluster has to be similar to most other sequences, from other genomes in that ortholog cluster. This is achieved by designing a new kind of similarity function (linkage function) to capture the relationship between a gene and a subset of genes. The similarity relationships among genes from multiple genomes are represented as a multipartite graph, where nodes in a partite set correspond to genes in a genome. To this we apply a new clustering method for multipartite graphs. The proposed method assigns a score to any arbitrary subset of genes from multiple genomes to assess orthologous relationships between genes in the subset, finding an ortholog cluster as the subset with the highest score. Thus, an ortholog cluster is found as a global optimal solution to a combinatorial optimization problem on multipartite graphs. Assigning a score that best reflects the ortholog relationships among genes in an arbitrary subset of genes is critical

- *A. Vashist and C.A. Kulikowski are with the Department of Computer Science, Rutgers-The State University of New Jersey, 110 Frelinghuysen Rd., Piscataway, NJ 08854. E-mail: {vashisht, kulikows}@cs.rutgers.edu.*
- *I. Muchnik is with DIMACS, Rutgers-The State University of New Jersey, 96 Frelinghuysen Rd., Piscataway, NJ 08854. E-mail: muchnik@dimacs.rutgers.edu.*

Manuscript received 15 Feb. 2006; revised 26 Apr. 2006; accepted 3 May 2006; published online 9 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0019-0206. Digital Object Identifier No. 10.1109/TCBB.2007.1004.

1545-5963/07/$25.00 © 2007 IEEE Published by the IEEE CS, CI, and EMB Societies & the ACM

to our approach. When considering orthologs from multiple genomes, observed sequence similarities between a pair of orthologous genes depend on the time since divergence of the corresponding genomes. So, in addition to sequence similarities between genes, we consider the phylogenetic relationship between genomes. We also describe how the gene order information can be incorporated into our method to improve ortholog detection.

The proposed method is fast and enables us to extract ortholog clusters from a large set of genomes. The key to the efficiency of the procedure is a particular property of the objective function, which is based on the linkage function. Computationally, the algorithm finds an ortholog cluster in time $O(|E| + |V| \log |V|)$, where E represents the edges and V represents the vertices in the weighted multipartite graph expressing the similarity relationships between genes from different genomes. Furthermore, if we discretize the weights on the edges into a constant number of bins, the algorithm runs in $O(|E| + |V|)$ time. We also describe implementation details that enable significant speedup in practice. Due to these choices, the method is efficient for finding candidate ortholog clusters in a large number of genomes and automatically determines the number of candidate ortholog clusters. We have applied this method to finding ortholog clusters in seven genomes on which the KOG database [8] is constructed.

In what follows, we present our ortholog model in Section 2 and describe the algorithm for extracting ortholog clusters in Section 3. The implementation details and techniques to speed up the ortholog extraction are given in Section 4. The experimental results on the seven genomes are described in Section 5 and the comparison of results with the known ortholog clusters is presented in Section 6. Section 7 gives the conclusion and future work.

2 Problem Formulation: Ortholog Model

A challenge in ortholog cluster extraction is avoiding detection of paralogs, which are genes that have evolved through duplication of an ancestral gene [1]. From a gene-function point of view, correct identification of orthologs is particularly important since they usually perform very similar functions, whereas paralogs, although highly similar at the primary sequence level, functionally diverge to adapt to new functions. Paralogs are closely related to orthologs because, if a duplication event follows a speciation event, orthology becomes a relationship between a set of paralogs [8]. Due to this complex relationship, paralogs related to ancient duplications are placed in different ortholog clusters, whereas recently duplicated genes are placed in the same ortholog cluster, as has been done in COG [9], KEGG [5], and Inparanoid [7]. Such a definition of ortholog clusters is justified from a gene function perspective because anciently duplicated genes are most likely to have adapted to new functional niches.

We address the above challenges by modeling the ortholog clusters as clusters in a multipartite graph. Orthologs, by definition, are present in different genomes [1]. We represent this in the multipartite graph by letting different genomes correspond to partite sets and the genes in a genome correspond to vertices in a partite set. As shown in Fig. 1, a multipartite graph considers the similarity relationships between genes from different genomes and ignores similarities between genes within a genome. Furthermore, the multipartite graph representation is suitable for discriminating between recently and anciently duplicated paralogs. Recently duplicated paralogs are confined to a genome and are very similar in primary sequence, so these copies share similarities to the same set of genes in other genomes. On the other hand, anciently duplicated paralogs are more similar to orthologs in other genomes compared to paralogs within the genome. So, the multipartite graph clustering, described below, places recently duplicated genes in the same ortholog cluster and the ancient paralogs in different clusters.

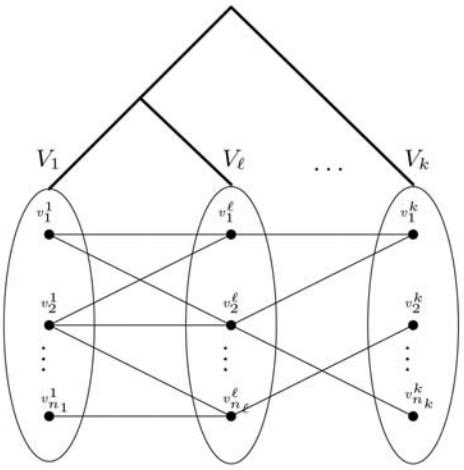

Fig. 1. A multipartite graph representing relationships between genes (small solid circles) from different genomes (ellipses). There are k different genomes and the genome ℓ, represented by V_ℓ, contains n_ℓ genes v_1^ℓ through $v_{n_\ell}^\ell$. The straight lines connecting the genes across the genomes represent the weighted similarity between them (dissimilar genes are not connected by edges), while the species tree is represented by the thick lines connecting the genomes. For visual clarity, weights on the edges are not shown.

A challenge specific to ortholog detection in a large set of genomes is the variation in observed sequence similarity between orthologs from different pairs of genomes. Within an ortholog family, orthologous sequences belonging to anciently diverged genomes are relatively less similar in comparison to those from the recently diverged genomes [11]. So, automatic methods based on numerical measures of sequence similarity must correct for this bias in observed sequence similarities. This can be done by considering the observed numerical similarity values in the context of the species tree relating the species in the data. The species tree can be used for rescaling the observed similarity values. In a strict sense, the issue related to correction is more complicated due to diverse evolutionary rates across lineages and protein families [11]. Our intention is not to correct for the absolute rates of evolution. To correct the observed sequence similarity between a pair from two genomes, we use the distance between corresponding genomes (described later). This is based on the assumption that the rate of primary sequence evolution is constant within an ortholog family.

Most ortholog detection methods consider genomes as a bag of genes and find ortholog clusters solely based on sequence similarity [7], [8], [9]. However, the leverage gained by using auxiliary information such as order of genes in a genome is widely recognized [11]. In fact, studies [12] show that the order of genes in the genome can reliably determine the phylogenetic relationship between closely related organisms. We describe later how the gene-order information can be used in conjunction with the sequence similarity to find ortholog clusters.

2.1 Ortholog Clusters on a Multipartite Graph

Consider the ortholog clustering problem with k genomes, where V_ℓ, $\ell \in \{1, 2, \ldots, k\}$, represents the set of genes from the genome ℓ. Then, the similarity relationships between genes from different genomes can be represented by an undirected weighted multipartite graph $G = (V, E, W)$, where $V = \cup_{\ell=1}^{k} V_\ell$ and V_ℓ is the set of genes from the genome ℓ, and $E \subseteq \cup_\ell \neq jV_\ell \times V_j$ (where $\ell, j \in \{1, 2, \ldots, k\}$) is the set of weighted, undirected edges representing similarities between genes.

Since orthologs evolve from the same ancestral sequence and perform the same function, we expect the primary sequence similarity between sequences in an ortholog cluster to be high. Then, the problem of finding an ortholog cluster could be modeled as finding the maximum weight multipartite clique. However, no efficient procedure exists for solving this problem [13]. Yet, cliques are simple models for an ortholog cluster which require robust models that allow some incompleteness in a subgraph extracted as a cluster. Due to this, clusters are often modeled as quasi-cliques or dense graphs [5]. Traditionally, the quasi-cliques are defined, using a threshold, as a relaxation of a complete subgraph—the relaxation can be on the degree of a vertex [14] or on the total number of edges in the quasi-clique [15]. In contrast to traditional quasi-clique definitions, our definition does not use any threshold parameters since it finds quasi-cliques based on the structure of the input graph.

To find a weighted multipartite quasi-clique as an ortholog cluster, we assign a score $F(H)$ to any subset H of V. The score function denotes a measure of proximity among the genes in H. Then, our multipartite quasi-clique (also called the *maximizer* or cluster), H^*, is defined as the subset with the largest score value, i.e.,

$$H^* = \arg\max_{H \subseteq V} F(H). \quad (1)$$

The subset H contains genes from multiple genomes, so, according to (1), our approach finds an ortholog cluster as a set of genes from multiple genomes by simultaneously considering all the similarity relationships in H. This is novel since, to our knowledge, all sequence similarity-based methods, such as [7], [9], find an initial set of orthologs from two genomes and possibly extend them at later stages. The function $F(H)$ is designed using a linkage function $\pi(i, H)$ which measures the degree of similarity of the gene $i \in H$ to other genes in H. The function $F(H)$ is then defined as

$$F(H) = \min_{i \in H} \pi(i, H), \quad \forall i \in H \ \ \forall H \subseteq V. \quad (2)$$

In other words, $F(H)$ is the $\pi(i, H)$ value of the least similar (outlier) gene in H. Then, according to (1), the subset H^* contains genes such that the similarity of the least similar gene in H is maximum.

Our linkage function considers the sequence similarity between genes within the ortholog cluster, their relationship to genes outside the cluster, and the phylogenetic distance between the corresponding genomes. Consider a subset H of V that contains genes from at least two genomes so that H can be decomposed as $H = \cup_{i=1}^{k} H_i$, where H_i is the subset of genes from V_i present in H. If m_{ij} (≥ 0) is the similarity value between gene i from genome $g(i)$ and gene j from another genome $g(j)$, and $p(g(i), g(j))$ represents the distance between the two genomes, then the linkage function is defined as

$$\pi(i, H) = \sum_{\substack{\ell=1 \\ \ell \neq g(i)}}^{k} p(g(i), \ell) \left\{ \sum_{j \in H_\ell} m_{ij} - \sum_{j \in V_\ell \setminus H_\ell} m_{ij} \right\}. \quad (3)$$

Given the phylogenetic tree for the genomes under study, the distance, $p(g(i), g(j))$ (≥ 0), between the genomes is defined as the height of the subtree rooted at the last common ancestor of the genomes $g(i)$ and $g(j)$. This term is used to correct the observed sequence similarities by magnifying the sequence similarities corresponding to genomes which diverged in ancient times. The term $\sum_{j \in H_\ell} m_{ij}$ aggregates the similarity values between the gene i from genome $g(i)$ and all other genes in the subset H that do not belong to genome $g(i)$, while the second term, $\sum_{j \in V_\ell \setminus H_\ell} m_{ij}$, estimates how this gene is related to genes from genome ℓ that are not included in H_ℓ. A large positive difference between these two terms ensures that the gene i is highly similar to genes in H_ℓ and, at the same time, very dissimilar from genes not included in H_ℓ. From a clustering point of view, this ensures large values of intracluster homogeneity and intercluster heterogeneity for extracted clusters. Translated to ortholog clustering, such a design enables separation of ortholog clusters for anciently duplicated paralogs.

3 Multipartite Graph Clustering

We now give an algorithm to find the solution for the combinatorial optimization problem defined in (1) and study properties of the functions $\pi(i, H)$ and $F(H)$ which guarantee an efficient algorithm to find the optimal solution. The linkage function in (3) and the score function in (2) were designed such that they satisfy these properties.

Definition 1. *A linkage function, $\pi : V \times 2^V \rightarrow \Re$, is* monotone increasing *if it satisfies the inequality:*

$$\pi(i, H) \geq \pi(i, H_1) \quad \forall i, \forall H_1, \forall H : i \in H_1 \subseteq H \subseteq V. \quad (4)$$

Claim 1. *The linkage function $\pi(i, H)$ defined in (3) is monotone increasing.*

Proof. Observe that the distance $p(i, j)$ is merely a scaling factor for the observed similarities and does not impact the monotonicity. Consider the case when H is extended to $H \cup \{k\}$ and assume $k \in V_s$.

If $i \in V_s$, then $\pi(i, H \cup \{k\}) = \pi(i, H)$, otherwise $\pi(i, H \cup \{k\}) - \pi(i, H) = 2m_{ik} \geq 0$, which proves the claim. □

Definition 2. *A set function, $F : 2^V \setminus \emptyset \rightarrow \Re$, is* quasi-concave *if it satisfies*

$$F(H_1 \cup H_2) \geq \min(F(H_1), F(H_2)) \quad \forall H_1, H_2 \subseteq V. \tag{5}$$

Proposition 1. *The set function $F(H)$ as defined in (2) is quasi-concave if and only if the linkage function is monotone increasing.*

Proof. [⇒]Let $H_1, H_2 \subseteq V$ and $i^* \in H_1 \cup H_2$ be such that $F(H_1 \cup H_2) = \pi(i^*, H_1 \cup H_2)$. Suppose, $i^* \in H_1$, then using (4), we get

$$F(H_1 \cup H_2) = \pi(i^*, H_1 \cup H_2) \geq \pi(i^*, H_1) \geq \min_{i \in H_1} \pi(i, H_1)$$
$$= F(H_1) \geq \min(F(H_1), F(H_2)).$$

[⇐] The proof is by contradiction. For $i \in H_1 \subseteq H \subseteq V$, assume that $\pi(i, H_1 \cup H) < \pi(i, H_1)$ and (5) hold. From the assumption, we get $\min_{i \in H_1} \pi(i, H_1 \cup H) < \min_{i \in H_1} \pi(i, H_1) = F(H_1)$. Further,

$$F(H_1 \cup H) = \min_{i \in H_1 \cup H} \pi(i, H_1 \cup H) \leq \min_{I \in H_1} \pi(i, H_1 \cup H).$$

Combining these two inequalities, we get $F(H_1 \cup H) < F(H_1)$, which contradicts the quasi-concavity property (5) in the assumption. □

Proposition 2. *For a quasi-concave set function $F(H)$, the set of all its maximizers, as defined by (2), is closed under the set union operation.*

Proof. The proof follows from the quasi-concavity of $F(H)$. Let H_1 and H_2 be two arbitrary maximizers; then $F(H_1) = F(H_2) \geq F(H_1 \cup H_2)$. Since $F(H_1) = F(H_2)$, $\min(F(H_1), F(H_2)) = F(H_1)$. Then, using (5), we get $F(H_1 \cup H_2) = \min(F(H_1), F(H_2)) = F(H_1)$. Therefore, $H_1 \cup H_2$ is also a maximizer. The proof also holds when we have more than two maximizers. □

A maximizer of $F(H)$ that contains all other maximizers is called the $\cup$-*maximizer*, $\hat{H}$. It follows from Proposition 2 that $\hat{H}$ is the unique largest maximizer.

The algorithm to find the optimal solution $\hat{H}$ is described in Table 1. This iterative algorithm begins by calculating $F(V)$ and the set M_1 containing the subset of vertices that satisfy $F(V) = \pi(i, V)$, i.e., $M_1 = \{i \in V : \pi(i, V) = F(V)\}$. The vertices in the set M_1 are removed from V to get $H_2 = V \setminus M_1$. At the iteration t, it considers the set H_{t-1} as input, calculates $F(H_{t-1})$, identifies the subset M_t such that $F(H_{t-1}) = \pi(i_t, H_{t-1}), \forall i_t \in M_t$, and removes this subset from H_{t-1} to produce $H_t = H_{t-1} \setminus M_t$. The algorithm terminates at the iteration T when $H_T = \emptyset$ or $F(H_T) = 0$. It outputs $\hat{H}$ as the subset, H_j, with smallest j such that $F(H_j) \geq F(H_\ell)\ \forall l \in \{1, 2, \ldots, T\}$. Thus, the algorithm finds the largest optimal solution that includes all other optimal solutions.

TABLE 1
Pseudocode for Extracting $\hat{H}$

Algorithm III.1: ALGORITHM FOR FINDING $\hat{H}()$

$t \leftarrow 1$; $H_t \leftarrow V$; $\Gamma \leftarrow V$;
$F(\Gamma) \leftarrow \min_{i \in V} \pi(i, V)$
while $(H_t \neq \emptyset)$
do
- $M_t \leftarrow \{\alpha \in H_t : \pi(\alpha, H_t) = \min_{j \in H_t} \pi(j, H_t)\}$; (1)
- $F(H_t) \leftarrow \min_{j \in H_t} \pi(j, H_t)$;
- **if** $(H_t \setminus M_t) = \emptyset) \vee (\pi(i, H_t) = 0\ \forall i \in H_t)$
  - **then**
    - output Γ as the optimal set, $\hat{H}$ and $F(\Gamma)$ as the optimal value, $F(\hat{H})$.
    - STOP.
  - **else**
    - $H_{t+1} \leftarrow H_t \setminus M_t$;
    - $t \leftarrow t + 1$;
    - Update $\pi(i, H_t)\ \forall i \in H_t$; (2)
    - **if** $(F(H_t) > F(\Gamma))$
      - **then** $\Gamma = H_t$; $F(\Gamma) = F(H_t)$;

This algorithm resembles the one for finding the largest subgraph with the maximum minimum degree [16]; however, our formulation is very general and applies whenever the function $F(H)$ is quasi-concave. Furthermore, by designing an appropriate linkage function, various structures in a graph can be obtained [17].

Theorem 1. *The subset Γ output by the above algorithm is the $\cup$-maximizer for F in the set V.*

Proof. According to the algorithm, $F(\Gamma) = \max_{H_I \in \mathcal{H}} F(H_I)$, where $\mathcal{H} = \{H_1, H_2, \ldots, H_T\}$ and

$$H_T \subset \ldots \subset H_2 \subset H_1 = V.$$

The proof for $F(\Gamma) \geq F(H)\ \forall H \subseteq V$ is divided into two cases: 1) $H \setminus \Gamma \neq \emptyset$ and 2) $H \subseteq \Gamma$.

Case 1) $[H \setminus \Gamma \neq \emptyset]$: Consider an arbitrary set H such that $H \setminus \Gamma \neq \emptyset$ and let H_i be the smallest set in the sequence $\mathcal{H}$ containing H so that $H \subseteq H_i$ but $H \not\subseteq H_{i+1}$. Since $M_i = H_I \setminus H_{i+1}$, there is at least one element, say i_H, common to both M_i and H. By definition of $F(H_i)$, we have

$$F(H_i) = \min_{i \in H_i} \pi(i, H_i) = \pi(i^*, H_i). \tag{6}$$

By construction of M_i, $i^* \in M_i$, so $\pi(i^*, H_i) = \pi(i_H, H_i)$. Using (4), we get

$$\pi(i^*, H_i) = \pi(i_H, H_i) \geq \pi(i_H, H) \geq \min_{i \in H} \pi(i, H) = F(H). \tag{7}$$

Using (6) and (7) we obtain $F(H_i) \geq F(H)$. According to the algorithm, $F(\Gamma) > F(H_i),\ \forall H_i \supset \Gamma$, so we prove

$$F(\Gamma) < F(H),\ \forall H \setminus \Gamma \neq \emptyset. \tag{8}$$

Case 2) $[H \subseteq \Gamma]$: In analogy to the previous case, there exists a smallest subset H_i in the sequence $\mathcal{H}$ that includes H. So, the inequalities in (7) hold here too, and we could write $F(H_I) \geq F(H)$. On the other hand, $F(\Gamma) \geq F(H_i)$, which, in conjunction with the previous inequality, implies

TABLE 2
Algorithm for Finding All Multipartite Clusters

Algorithm III.2: FINDING ALL MULTIPARTITE CLUSTERS()

$V^1 \leftarrow V$; $m \leftarrow 1$; $\mathcal{C} \leftarrow \emptyset$;
while ($V^m \neq \emptyset \vee$!($\pi(i, V^m) = 0 \ \forall i \in V^m$))
do
- $\hat{H}^m \leftarrow \arg\max_{H \subseteq V^m} F(H)$ (using algo. in Table I)
- Append $\hat{H}^m$ to $\mathcal{C}$;
- $V^{m+1} \leftarrow V^M \setminus \hat{H}^m$; $m \leftarrow m + 1$;

OUTPUT $\mathcal{C}$ as the ordered set of clusters;
m as the number of clusters, and
V^m as the residual elements, $\mathcal{R}$.

$$F(\Gamma) \geq F(H), \ \forall H \subseteq \Gamma. \tag{9}$$

Further, a maximizer satisfying (8) and (9) is the $\cup$-maximizer. □

3.1 Partitioning of Data Into Multipartite Clusters

The algorithm in Table 1 outputs one multipartite cluster. However, many such clusters are likely to be present in the set V. If we assume that these clusters are unrelated, we can use a simple heuristic of iteratively applying the above procedure to extract all these clusters. To do this, we remove the elements belonging to the first cluster $\hat{H}$ from V and extract another multipartite cluster in the set $V \setminus \hat{H}$. This procedure is formalized in Table 2 and produces an ordered set, $\mathcal{C} = \{\hat{H}^1, \hat{H}^2, \ldots, \hat{H}^m\}$, of m ortholog clusters and a set of residual elements $\mathcal{R} = \{i : i \in V \setminus \mathcal{C}\}$. The number, m, of nontrivial clusters (ortholog clusters) is automatically determined by the method. It must be remarked that every cluster in $\mathcal{C}$ contains genes from at least two genomes and the clusters in $\mathcal{C}$ are ordered according to their score values, i.e., $F(\hat{H}^1) > F(\hat{H}^2) > \ldots > F(\hat{H}^m)$.

4 Analysis and Implementation

We now analyze the algorithm given in Table 1. The runtime of this algorithm depends on the efficiency of evaluating the linkage function. Due to the frequent updates to the linkage function value, a linkage function which can be updated efficiently, instead of having to be evaluated from scratch at each iteration, is preferable. The linkage function described in (3) is additive and can be updated efficiently when vertices are removed from the set.

4.1 Straightforward Implementation

During the initialization of the algorithm, we must compute the linkage function for all the vertices in V. The linkage function value for a single element i depends on the weights on edges (from other elements) incident on it and on the relationship of the partite set $g(i)$ to other partite sets in H. To compute the initial linkage function values for all the vertices in V, we must look at all the edges in E and all the relationships between the partite sets. We assume that the number of partite sets k is very small compared to the size of V, so the complexity of computing the initial linkage function values is $O(|E|)$. At subsequent iterations, each vertex is deleted at most once; deleting a vertex entails deletion of all the incident edges and updating the linkage function values for all the neighboring vertices of the deleted vertex. For the linkage function proposed in (3), these updates can be carried out by subtracting the contribution due to the deleted edge, the total number of updates to the linkage function is bounded by the total number of edge deletions, which is at most $|E|$. So, the initial linkage function computation and all the subsequent updates to it together can be $O(|E|)$ time. In what follows, the cost of calculating and updating the linkage function values (the step indicated by (2) in Table 1) remains fixed ($O(|E|)$) and, for calculating the overall efficiency of the algorithm, we focus on other steps in the algorithm.

We now analyze a naive implementation of the algorithm in Table 1. In this implementation, the costliest operation is indicated by Step (1) in Table 1. At this step, we find the vertices with the minimum value of the linkage function; this can be done by looking at the vertices active at that iteration. Since the total number of vertices at any iteration i is at most $|V| - i$ and the total number of iterations is at most $|V|$ (achieved when a single vertex is deleted at each iteration), the total time required for finding the minimum values over all iterations is $\sum_{i=1}^{|V|}(|V| - i) = \sum_{i=1}^{|V|} i = |V|(|V| - 1)/2 = O(|V|^2)$. At each iteration, two stopping conditions are tested, which takes a constant time, and some elements are removed from the set H^t. Since each element is removed only once, the time required for executing Step (2) during all the iterations is $O(|V|)$. Thus, the total time required for this straightforward implementation is $O(|E| + |V|^2 + |V|) = O(|E| + |V|^2)$.

4.2 Implementation Based on Efficient Data Structures

As described in the algorithm in Table 1 and discussed above, three abstract operations are performed at each iteration of the algorithm. These are:

- `find-min`: This corresponds to Step (1) of the algorithm where M_t, the set of elements with the minimum value of the linkage function, is determined.
- `delete-min`: This corresponds to the statement $H_{t+1} = H_t \setminus M_t$ in the algorithm. It involves removing the elements in the set M_t to find the list of active elements H_{t+1} for the subsequent iteration.
- `decrease-key`: Deleting the elements in the set M^t entails updating the linkage function values for the neighbors of elements in the set M^t. The total cost of updating the linkage function values across all iterations is $O(|E|)$ and is already considered. This operation applies only when we use certain data structures for storing the linkage function values. In such a case, this operation refers to the work required to find the new position for elements whose linkage function values are affected (decreased) in the data structure for storing the elements according to their linkage function values. In the straightforward implementation above, we did not use any special data structures, so there was no cost associated with this operation.

Having decomposed the steps at each iteration into the three basic operations described above, an efficient implementation strives to minimize the total time consumed by these operations. We now give some data structure choices that enable us to implement the algorithm in Table 1 efficiently. These choices and their corresponding complexities are stated in the theorem below.

Theorem 2. *The algorithm, in Table 1, for finding an ortholog cluster runs in time $O(|E| + |V| \log |V|)$ and space $O(|E| + |V|)$.*

Proof. Apart from initializing some values, the initialization of the algorithm includes computing $\pi(i, V)\ \forall i \in V$. To compute $\pi(i, V)$, we must look at all edges incident on i; thus, computing $\pi(i, V)\ \forall i \in V$ takes $O(|E|)$ time. At subsequent iterations, due to the additive property of the linkage function (3), efficient updates are possible without recomputing from scratch. As each edge is deleted once, all linkage function updates (shown in Step (2) of the algorithm) together require $O(|E|)$ time.

The step marked as (1) involves determining the set M_t by finding the vertices with minimum value of the linkage function. Observe that, in a sparse multipartite graph, only a few edges are deleted at each iteration, implying that only a few linkage function values are updated. Consequently, the order of vertices determined by the linkage function values remains approximately fixed. We use Fibonacci heaps [18] (which work regardless of the sparsity in the input graph) to store vertices according to their linkage function values. So, elements in the set M_t can be found in $O(1)$ time using the `find-min` operation. In Step (2), the set $H_{t+1} = H_t \setminus M_t$ can be found in $O(\log |H_t|)$ time using the `delete-min` operation and each update to a linkage function value can be performed in $O(1)$ using the `decrease-key` operation. Thus, using Fibonacci heaps, each iteration takes $O(\log |H_t|)$ time. The maximum number of iteration is $|V|$ and each iteration takes at most $O(\log |V|)$. Thus, the algorithm runs in $O(|E| + |V| \log |V|)$ time. Using the adjacency list representation for storing the graph requires $O(|E| + |V|)$ space. As the Fibonacci heaps for storing $|V|$ elements can be implemented in $O(|V|)$ space, the total space requirement for the algorithm is $O(|E| + |V|)$. □

Theorem 3. *If the values in the similarity matrix are discretized into b different but constant values, the algorithm runs in $O(|E| + |V|)$ time.*

Proof. The initialization step, as in Theorem 2, takes $O(|E|)$ time.

To reduce the time complexity of subsequent iterations, we sort and store the vertices in the order of the initial linkage function values, $\pi(i, V)$. We assume that the graph is represented in the adjacency list format and sort the vertices using the bucket sort algorithm [19]. Within each bucket, the vertices are stored using a linked list to accommodate multiple vertices with the same value. The edge weights can take b different values, so the initial linkage function values are bounded, i.e., $0 \leq \pi(i, V) \leq b \cdot |V|$. So, sorting the $|V|$ values of $\pi(i, V)$ takes $O(|V|)$ time.

At each iteration, the algorithm finds the vertex with minimum value of the linkage function. This takes $O(1)$ time as vertices are sorted according to the linkage function values. Deletion of a vertex entails updates to the linkage function values for the neighboring vertices. The cost for updating linkage function values is already considered, but, to preserve the sorted order of vertices, we must find the new place for each updated vertex. Since the edge weights are discretized, the new place must lie at most b bins away from the current position (towards the minimum). Thus, the new place is found in at most b, or $O(1)$ time. Every iteration requires $O(1)$ time and, since the number of iterations is at most $|V|$, the total time is bounded by $O(|V|)$. Combining this with the total cost for computing the linkage function values, the runtime of the algorithm is $O(|E| + |V|)$. □

As a result of Theorem 2, the procedure for finding all the m ortholog clusters runs in time $O(m(|E| + |V|))$. We now give some implementation details which do not improve the complexity of the algorithm but enable a speedup in practice.

Vertices in different connected components of a graph cannot come together to form a cluster. So, different connected components can be processed in isolation. Furthermore, the input multipartite graph between genes is large but very sparse, so, when a dense subgraph is extracted as the optimal solution, the remaining graph becomes disconnected. As a consequence, in practice, a significant speedup is achieved when the procedure in Table 2 is run on individual connected components after extraction of an ortholog cluster. Also, finding an optimal solution in different connected components is amenable to parallelism.

The algorithm in Table 1 removes the vertices corresponding to the minimum of the linkage function. If, however, we could remove a larger set of vertices without affecting the correctness of the procedure, the algorithm would be faster as more vertices would be removed at each iteration. The following theorem determines such elements:

Theorem 4. *Define $Q = \{i \in V : \pi(i, V) < \theta, \theta > 0\}$ and let $\hat{H}_{V \setminus Q}$ be the $\cup$-maximizer in the set $V \setminus Q$, and $\hat{H}$ be the $\cup$-maximizer in the set V. Then,*

$$F(\hat{H}_{V \setminus Q}) > \theta \Rightarrow \hat{H}_{V \setminus Q} = \hat{H}. \tag{10}$$

Proof. We first prove that $F(\hat{H}_{V \setminus Q}) > \theta \Rightarrow \hat{H} \subseteq V \setminus Q$. The score-value of $\hat{H}_{V \setminus Q}$, obtained from $V \setminus Q$ (a subset of V) can be at most the score-value of $\hat{H}$ obtained from V, i.e., $F(\hat{H}) \geq F(\hat{H}_{V \setminus Q}) > \theta$. Then, using (2), it follows that $\forall i \in \hat{H}, \pi(i, \hat{H}) > \theta$. Further, by the monotonicity property, we get $\pi(i, V) \geq \pi(i, \hat{H}) > \theta$. But, according to the definition of Q, $\pi(i, V) > \theta \Rightarrow i \in V \setminus Q$. This proves that $\forall i \in \hat{H}, i \in V \setminus Q$; in other words, $\hat{H} \subseteq V \setminus Q$. But, by the definition of $\cup$-maximizer, we have $F(\hat{H}_{V \setminus Q}) \geq F(H)\ \forall H \subseteq V \setminus Q$, i.e., $F(\hat{H}_{V \setminus Q}) \geq F(\hat{H})$. But, we already had $F(\hat{H}) \geq F(\hat{H}_{V \setminus Q})$, so $F(\hat{H}_{V \setminus Q}) = F(\hat{H})$. Further, the uniqueness of $\cup$-maximizer implies $\hat{H}_{V \setminus Q} = \hat{H}$. □

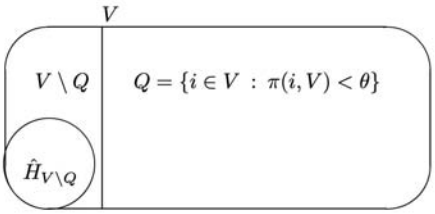

Fig. 2. Implication of Theorem 4: It guarantees that the optimal solution found in the smaller set $V \setminus Q$ is exactly the same as the one that would be found in V and thus saves computational time.

According to Theorem 4, we can remove all vertices whose linkage function values are less than the current estimate of the score value of the $\cup$-maximizer (see Fig. 2). In the algorithm described in Table 1, $F(\Gamma)$ is the current estimate of the $F(\hat{H})$, so we can remove all vertices, $M_t = \{i : \pi(i, H_t) < F(\Gamma)\} \cup \{i : \pi(i, H_t) = \min_{j \in H_t} \pi(j, H_t)\}$. This reduces the number of iterations as more vertices are likely to be removed at each iteration. Unfortunately, this does not improve the worst-case complexity of the algorithm—it is easy to come up with instances where this modification does not lead to improvements.

Theorem 4 can also improve the runtime of the procedure given in Table 2. This procedure, at iteration t, finds a cluster as the optimal set $\hat{H}^t$, removes this optimal set from the current set to produce a resulting set, $V^{t+1} = V^t \setminus \hat{H}^t$, in which the optimal set, $\hat{H}^{t+1}$ is found at the next iteration $t+1$. For the ortholog clustering, we empirically found that $F(\hat{H}^{t+1})/F(\hat{H}^t) \geq 0.8$, so we apply Theorem 4 with $\theta = 0.8 * F(\hat{H}^t)$ for finding $\hat{H}^{t+1}$. Such preprocessing removes more than 95 percent of vertices that do not belong to the optimal set and thus leads to significant performance gains.

5 Data for Ortholog Clustering

We have used two types of data. The first is the input data for ortholog clustering, which contains a set of protein sequences from multiple genomes and the species tree relating those species. The input sequence data is used for constructing the multipartite graph between the sequences, while the species tree is used for specifying the relationships between the partite sets. The second type of data is comprised of functional annotations for sequences in the input data—these annotations are used exclusively for validating the ortholog clusters from a protein function perspective.

5.1 Data

For the input sequence data, we have used the protein sequences from complete genomes of seven eukaryotes present in the eukaryotic orthologous groups , or KOG [8]. This database is publicly accessible at the URL: http://www.ncbi.nlm.nih.gov/COG/new/. We chose this data because expert curated ortholog clusters are available on this data; we treat these known ortholog clusters (called KOGs) as a "gold-standard" for validating ortholog clusters obtained by our method. Furthermore, the number of genomes and the sequences in them is large enough to make a case for automatic and large scale ortholog clustering—a problem we want to address.

TABLE 3
Distribution of Sequences in the KOGs

Species	Total sequences	Sequences in KOGs	number of KOGs
A. thaliana	26,406	13,744	3,285
C. elegans	20,751	10,582	4,235
D. melanogaster	13,694	8,445	4,351
H. sapiens	38,638	19,039	4,597
S. cerevisiae	6,387	4,003	2,668
S. pombe	5,034	3,728	2,762
E. cuniculi	1,999	1,218	1,073
Total	112,920	60,759	

The KOG database is constructed from 112,920 sequences present in the seven eukaryotic genomes: *Arabidopsis thaliana*, *Caenorhabditis elegans*, *Drosophila melanogaster*, *Encephalitozoon cuniculi*, *Homo sapiens*, *Saccharomyces cerevisiae*, and *Schizosaccharomyces pombe*. There are 4,852 KOGs containing 60,759 sequences from a total of 112,920 sequences present in the seven eukaryotic genomes. A sequence distribution for these seven genomes is shown in Table 3. By construction, every KOG consists of sequences from at least three different species. Although most KOGs are small (3,372 KOGs contain 10 or fewer sequences), their size ranges from 3 to 1,300. A distribution of the size of KOGs and the number of species in KOGs is shown in Figs. 4 and 5, respectively.

The species tree relating the seven species in our input data was derived from the NCBI taxonomy browser located at http://www.ncbi.nlm.nih.gov/taxonomy/tax.html/ and is shown in Fig. 3.

5.2 Constructing the Multipartite Graph

To apply the proposed ortholog clustering method, we must compute the pairwise sequence similarities between the input sequences. Fortunately, precomputed pairwise sequence comparison results using the blastp program from the BLAST [20] suit are available from the KOG server; we used these sequence comparison results. BLAST provides two measures of pairwise sequence similarity: the e-value and the bit-score (also called the Z-score). We used e-value as a threshold for filtering out spurious matches; however, for constructing the multipartite graph of similarities between the sequences, we used bit-score as the similarity weight.[1] Due to the nature of BLAST searches, it is possible to obtain asymmetrical scores during pairwise sequence comparison, i.e., the similarity w_{ij} between sequences i and j may not be equal to the value w_{ji}, the similarity between j and i. Since our multipartite graph is undirected, we forced the weights to be symmetric by using $w_{ij} = w_{ji} = \max(w_{ij}, w_{ji})$. We ignored similarities between sequences inside genomes and our multipartite graph was a 7-partite graph, with a partite set corresponding to each of the seven species.

As discussed earlier, in Section 2, one of the central issues in ortholog clustering is to avoid paralogs. We have

1. We have also used $\lceil -\log_{10} \text{e-value} \rceil$ as an integer similarity weight, but our experimental results using the bit score were better. The comparison of these results is not reported and we present results using the bit score as the measure of similarity weight.

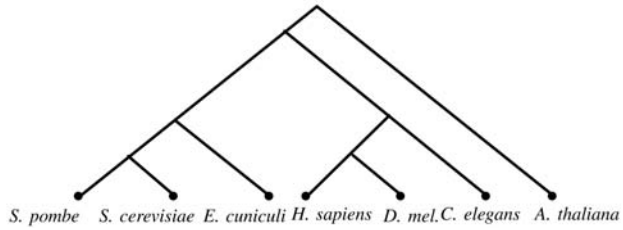

Fig. 3. The species tree relating *A. thaliana*, *C. elegans*, *D. melanogaster*, *E. cuniculi*, *H. sapiens*, *S. cerevisiae*, and *S. pombe*.

done this by ignoring the sequence similarities within genomes, so recently duplicated paralogs confined to single genomes are avoided. For avoiding the paralogs present across genomes, we considered only a subset of top hits for any given sequence. Since orthologs are highly likely to perform the same function in their respective genomes, they are likely to be more similar compared to paralogs which tend to diverge in function and sequence. We incorporated this in the multipartite graph of pairwise sequence similarities by limiting the similarity edges from a sequence in a sequence-specific and species-specific manner. To be precise, if a sequence i in genome $g(i)$ has the sequence j as its best match in genome $g(j)$ with score θ, we considered all those matches for i from species $g(j)$ which had a bit score larger than 0.5θ. We have experimented with different thresholds ranging from 0.5θ to 0.8θ, and found that clustering results do not change significantly for this range, so we fixed this threshold to a conservative value of 0.5θ. The idea behind this is to avoid low-scoring spurious hits (which would most likely be paralogs) without filtering out potential orthologs [6].

The topology of the species tree, shown in Fig. 3, was used for estimating the distance between the genomes. The distance $p(\ell, m)$ between the genome ℓ and the genome m was taken as the height of the subtree rooted at the most recent ancestor of ℓ and m. For instance, the distance between *S. cerevisiae* and *E. cuniculi* is 2 and that between *A. thaliana* and *H. sapiens* is 4.

5.3 Data for Validating Ortholog Clusters

The primary validation test for the extracted ortholog clusters is comparing them to the existing KOG ortholog clusters. For every sequence in the input data, we have

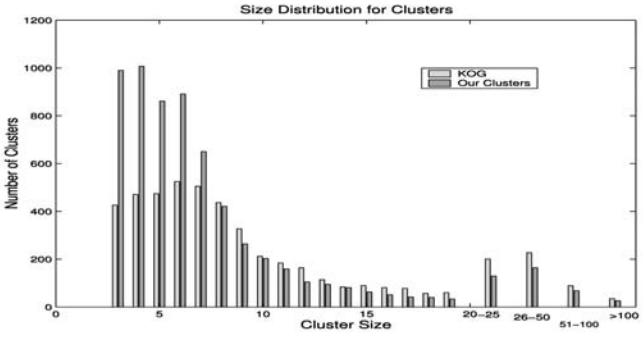

Fig. 4. A comparison of distribution of sizes KOGs and our ortholog clusters.

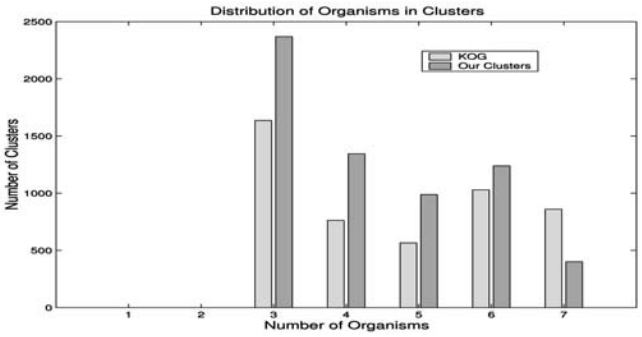

Fig. 5. A comparison of distribution of organisms (genomes) in KOGs and our ortholog clusters.

information about its membership in a KOG cluster, so a validation of our results can be based on comparing the two clustering results.

We have also used an independent validation for assessing the homogeneity of ortholog clusters. This validation is based on the functional annotations for protein sequences. Since orthologs are likely to be involved in the same (or similar) function, they are likely to share the same functional annotation, so an assessment of homogeneity of ortholog clusters can be based on comparing the functional annotations for the member sequences. However, we must emphasize that, although we expect orthologs to share the same function, the similarity in functional annotation for protein sequences from different species does not imply orthology (paralogs from different species can have the same functional annotation). Another issue in such a validation is related to multidomain proteins (different fragments of protein sequence independently capable of performing a biological role), where different domains of the sequence can have different functional annotations. For pairs of multidomain orthologous sequences, we expect both sequences to have the same set of functional annotations and the different domains must appear in the same order in both the sequences.

For assigning the functional annotations, we selected the Pfam (Protein Families) database [21], [22], which is a popular and trusted resource for functional annotations due to its expert curation of results from state-of-the-art automatic methods. This database, available at http://www.sanger.ac.uk/Software/Pfam/, contains multiple protein alignments and the seed sequences used for constructing these alignments. To be precise, we used the Pfam-A sequence families, which are families based on expert curated multiple alignments of sequences from SWISSPROT and TrEMBL [23]. These are strongly trusted matches to the family and are very unlikely to be false matches. We have used the families and sequences from Pfam-A release 12.0, containing 898,590 domain sequences divided into 7,316 Pfam functional families.

While assessing the homogeneity of clusters using Pfam annotations, sequences were assigned the Pfam family IDs of their "close" match(es) in the Pfam database. These matches were found through BLAST comparisons of the input sequences with those in Pfam-A. Apart from using an e-value threshold e^{-4} and filtering out the low-complexity regions (using SEG [24]) and coil-coil regions (using the

filter [25]) by using the option `-F "C;S"`, BLAST (blastp) was used with its default parameters.

6 Experimental Results

Our method produced 36,034 clusters including 25,434 singletons, 2,870 clusters of size 2, and 7,830 clusters that contain at least three sequences. A comparative distribution of size and the number of organisms in our clusters and the KOGs is shown in Fig. 4 and Fig. 5, respectively. In comparison to KOGs, our ortholog clusters are relatively smaller in size and contain sequences from fewer genomes.

A KOG cluster, by construction, contains sequences from at least three genomes, so, for the purpose of comparison, we divided the 7,830 clusters with at least three sequences into 1,488 clusters containing sequences from two genomes and 6,342 ortholog clusters that contain sequences from at least three genomes. The 6,342 clusters contain 61,272 sequences of which 47,458 are common with the 60,759 sequences in the KOG ortholog clusters. Of the 13,301 sequences from KOGs that are not covered by these clusters, 9,078 sequences are grouped into 1,566 clusters that contain sequences from at most two genomes while the remaining are classified as singletons. In comparison to KOGs, our clusters contain fewer paralogs—a desirable feature obtained by ignoring similarities between genes within a genome. Although desirable from an ortholog clustering perspective, this also means that our ortholog clusters are smaller in size as the method avoids detection of paralogs in an ortholog cluster. This is consistent with the distribution of sizes (Fig. 4) and the following statistical comparison.

To estimate the association between KOGs and our ortholog clusters, we calculated several statistical parameters. When testing whether KOGs and our clusters are related, we obtained a value of 0 for the χ^2-coefficient [26] suggesting a correlation between the two clustering results. However, this coefficient fails to provide an insight into the degree of association, so we used a normalized version of this coefficient, the *Cramer's V-coefficient* [26]. This coefficient lies between 0 and 1 and can be interpreted as the association between the two results as a percentage of their maximal possible variation. For our comparison, we obtained a value of 0.749. Considering the large number of clusters, this coefficient suggests a strong correlation between the two clusterings. We used the Rand index [27] to directly quantify the relatedness of the two clusterings. This index involves counting the number of agreements between the two classifications based on how each pair of elements from the underlying set is assigned to clusters by the two classification schemes. We obtained a value of 0.792 for this index, which implies that about 80 percent of all pairs of sequences in our clustering results agree with ortholog clusters in KOG.

For bringing out the relationship between individual clusters from our clustering and KOG clustering, we designed some indices for measuring the overlap between clusters. The average number of KOGs that overlap with an ortholog cluster is 1.031, which indicates that most of our clusters contain sequences from a single KOG. Conversely, the average number of candidate clusters that overlap with a KOG is 3.012. However, in most cases, a KOG completely contains our clusters, which suggests that our clusters are homogeneous with respect to the KOGs. These statistics, along with the size distribution, confirm our qualitative observation that our ortholog clusters are usually subsets of single KOGs.

A set-theoretic comparison of KOGs with our clusters shows that 844 KOGs (or 13 percent of all KOGs) exactly match our clusters. There are 611 KOGs that are divided exactly into two clusters, each of which is a subset of the corresponding KOG. Carrying this analysis further, we found that 2,472 KOGs (51 percent of all KOGs) can be partitioned into our clusters, each of which is a subset of a single KOG. In other words, those clusters totally contained in a single KOG. The remaining 2,380 KOGs overlap with at least one mixed ortholog cluster, i.e., a cluster that either contains sequences from multiple KOGs or contains some sequences that do not belong to any KOG.

For further exploring the above results, we assessed the homogeneity of our clusters by examining the KOG membership and Pfam annotations of sequences in our clusters. Among the 6,342 clusters produced by our method, most (4,485 or 70 percent) are subsets of single KOGs. The remaining 1,857 clusters, however, contain sequences from multiple KOGs. To assess the conservation between sequences of the these 1,857 clusters, we examined the Pfam annotations for their member sequences. We found that all members in 952 of these clusters were annotated with the same set of Pfam families. There were 436 clusters whose member sequences did not have any Pfam annotations. These sequences do not have any close match in the known Pfam families. The remaining 469 clusters (or 7 percent of all clusters) were not homogeneous according to Pfam annotations as some of their member sequences did not share the same Pfam annotation. However, it must be remarked that most of these clusters were obtained toward the final iterations of the clustering method described in Table 2 and, therefore, had very low score $(F(\hat{H}))$ values. Since these clusters can be identified by their scores, they can be eliminated by using a threshold on the score value for the clusters. In summary, the statistical coefficients and the set-theoretic comparison of our clusters with the manually curated ortholog clusters in KOG shows the two clusterings to be highly correlated.

7 Conclusion

We have modeled the problem of finding orthologous clusters in a large number of genomes as clustering on a multipartite graph. The proposed method is efficient and finds an ortholog cluster in time $O(|E| + |V| \log |V|)$. To further speed up the method, we presented implementation choices that lead to significant performance gains in practice. The proposed ortholog clustering method was applied to the seven eukaryote genomes on which KOG ortholog clusters are constructed. The analysis of the results shows that clusters obtained using the proposed method show a high degree of correlation with the manually curated ortholog clusters.

This method extracts an ortholog cluster by ignoring similarities between genes within a genome while

emphasizing orthologous relationships between genes from different genomes. Since the observed sequence similarity scores are higher for recently diverged orthologs compared to those for the anciently diverged orthologs, we used the species tree to correct for these differences (3).

Corrections to observed sequence similarity using phylogenetic trees assume the correctness of the given phylogenetic tree. However, there are instances when multiple hypotheses about the phylogenetic relationship between a group of organisms exist; in such cases, the confidence in those hypotheses is low. We hope that the proposed method can be modified to resolve such conflicts by constructing the gene tree for each ortholog cluster and deriving support for the species tree(s) from these gene trees [28]. When the phylogenetic information is controversial, an iterative process of finding ortholog clusters can resolve the ambiguities in the phylogenetic tree.

Apart from considering the sequence similarity in the context of the species tree, the order in which genes appear in their genomes is also considered important. Recently, the gene order has been used to find ortholog clusters in a pair of genomes [29]. The conserved gene order (or synteny) between a pair of genomes can be inferred using programs like DiagHunter [29]. This additional information about the orthologous relationships could also be incorporated into our method. A simple solution to do this would be locking together the similar genes, appearing in a conserved gene order, as orthologs. Alternatively, it is possible to create a new similarity coefficient between genes i and j which belong to different genomes but are conserved in gene order as determined by methods such as DiagHunter. Then, we can design a new linkage function

$$\pi'(i,H) = \pi(i,H)\left(\sum_{\substack{t=1\\ \ell\neq g(i)}}^{k}\sum_{j\in H_\ell} e_{ij}\right), \tag{11}$$

where $\pi(i,H)$ is the linkage function defined in (3). The first sum inside the parentheses aggregates gene order similarity coefficients e_{ij} between the gene i and genes in other genomes, while the second one aggregates gene order similarity coefficients between the gene i and genes in H_ℓ, the subset of genes from genome ℓ present in H. Thus, $\pi'(i,H)$ incorporates three diverse components: sequence similarity, species tree, and the gene order information, critical for ortholog clustering.

The ability to incorporate information from diverse sources indicating orthlogy between genes is promising. A complete solution to such a problem requires developing a sound multicriteria combinatorial approach. In the proposed method, we have been successful in combining the sequence similarity and the phylogenetic information for extracting ortholog clusters that are highly correlated with the manually curated clusters. This method has the potential to be extended for incorporating diverse information by designing new linkage functions as long as they are monotonically increasing.

ACKNOWLEDGMENTS

Akshay Vashist was, in part, supported by DIMACS awards. Ilya Muchnik was supported by US National Science Foundation grant CCF-0325398.

REFERENCES

[1] W.M. Fitch, "Distinguishing Homologous from Analogous Proteins," *Systematic Zoology*, vol. 19, pp. 99-113, 1970.
[2] E.V. Koonin, "An Apology for Orthologs—Or Brave New Memes," *Genome Biology*, vol. 2, no. 4, 2001.
[3] V. Solovyev and I. Shahmuradov, "PromH: Promoters Identification Using Identification Orthologous Genomic Sequences," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3540-3545, 2003.
[4] B. Lemos, B.R. Bettencourt, C.D. Meiklejohn, and D.L. Hartl, "Evolution of Proteins and Gene Expression Levels Are Coupled in Drosophila and Are Independently Associated with mRNA Abundance, Protein Length, and Number of Protein-Protein Interactions," *Molecular Biology Evolution*, vol. 22, pp. 1345-1354, 2005.
[5] W. Fujibuchi, H. Ogata, H. Matsuda, and M. Kanehisa, "Automatic Detection of Conserved Gene Clusters in Multiple Genomes by Graph Comparison and p-Quasi Grouping," *Nucleic Acids Research*, vol. 28, no. 2, pp. 4096-4036, 2002.
[6] M. Kamvysselis, N. Patterson, B. Birren, B. Berger, and E. Lander, "Whole-Genome Comparative Annotation and Regulatory Motif Discovery in Multiple Yeast Species," *Proc. Int'l Conf. Research in Computational Molecular Biology (RECOMB)*, pp. 157-166, 2003.
[7] M. Remm, C.E. Strom, and E.L. Sonnhammer, "Automatic Clustering of Orthologs and In-Paralogs from Pairwise Species Comparisons," *J. Molecular Biology*, vol. 314, pp. 1041-1052, 2001.
[8] R. Tatusov, N. Fedorova, J. Jackson, A. Jacobs, B. Kiryutin, E. Koonin, D. Krylov, R.M.R.S. Mekhedov, A. Nikolskaya, B. Rao, S. Smirnov, A. Sverdlov, S. Vasudevan, Y. Wolf, J. Yin, and D. Natale, "The cOG Database: An Updated Version Includes Eukaryotes," *BioMed Central Bioinformatics*, 2003.
[9] R.L. Tatusov, E.V. Koonin, and D.J. Lipman, "A Genomic Perspective on Protein Families," *Science*, vol. 278, pp. 631-637, 1997.
[10] C.M. Zmasek and S.R. Eddy, "RIO: Analyzing Proteomes by Automated Phylogenomics Using Resampled Inference of Orthologs," *BMC Bioinformatics*, vol. 3, no. 14, 2002.
[11] M.A. Huynen and P. Bork, "Measuring Genome Evolution," *Proc. Nat'l Academy of Sciences USA*, vol. 95, pp. 5849-5856, 1998.
[12] J. Tang and B. Moret, "Phylogenetic Reconstruction from Gene Rearrangement Data with Unequal Gene Content," *Proc. Eighth Workshop Algorithms and Data Structures (WADS '03)*, pp. 37-46, 2003.
[13] M. Dawande, P. Keskinocak, J.M. Swaminathan, and S. Tayur, "On Bipartite and Multipartite Clique Problems," *J. Algorithms*, vol. 41, pp. 388-403, 2001.
[14] J. Pei, D. Jiang, and A. Zhang, "On Mining Cross Graph Quasi-Cliques," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '05)*, 2005.
[15] J. Abello, M.G. Resende, and S. Sudarsky, "Massive Quasi-Clique Detection," *Proc. Latin Am. Theoretical Informatics Symp. (LATIN '02)*, 2002.
[16] D.W. Matula and L.L. Beck, "Smallest-Last Ordering and Clustering and Graph Coloring Algorithms," *J. ACM*, vol. 30, no. 3, pp. 417-427, 1983.
[17] B. Mirkin and I. Muchnik, "Induced Layered Clusters, Hereditary Mappings, and Convex Geometries," *Applied Math. Letters*, vol. 15, pp. 293-298, 2002.
[18] M.L. Fredman and R.E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization," *J. ACM*, pp. 596-615, 1987.
[19] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, second ed. The MIT Press, 2001.
[20] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389-3402, 1997.
[21] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S.R. Eddy, S. Griffiths-Jones, K.L. Howe, M. Marshall, and E.L.L. Sonnhammer, "The Pfam Protein Families Database," *Nucleic Acids Research*, vol. 30, no. 1, pp. 276-280, 2002.

[22] A. Bateman, L. Coin, R. Durbin, R.D. Finn, V. Hollich, S. Griffiths-Jones, A. Khanna, M. Marshall, S. Moxon, E.L.L. Sonnhammer, D.J. Studholme, C. Yeats, and S.R. Eddy, "The Pfam Protein Families Database," *Nucleic Acids Research,* vol. 32, no. 1, pp. 138-141, 2004.
[23] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider, "The SWISS-PROT Protein Knowledgebase and Its Supplement TrEMBL in 2003," *Nucleic Acids Research,* vol. 31, no. 1, pp. 365-370, 2003.
[24] J. Wootton and S. Federhen, "Statistics of Local Complexity in Amino Acid Sequences and Sequence Databases," *Computers and Chemistry,* vol. 17, pp. 149-163, 1993.
[25] A. Lupas, M. Van Dyke, and J. Stock, "Predicting Coiled Coils from Protein Sequences," *Science,* vol. 252, pp. 1162-1164, 1991.
[26] B.S. Everitt, S. Landau, and M. Leese, *Cluster Analysis,* fourth ed. Oxford Univ. Press, 2001.
[27] W.M. Rand, "Objective Criterion for the Evaluation of Clustering Methods," *J. Am. Statistics Assoc.,* vol. 66, pp. 846-850, 1971.
[28] R. Guigo, I. Muchnik, and T.F. Smith, "Reconstruction of Ancient Molecular Phylogeny," *Molecular Phylogenetic Evolution,* vol. 6, no. 2, pp. 189-213, 1996.
[29] S.B. Cannon and N.D. Young, "OrthoParaMap: Distinguishing Orthologs from Paralogs by Integrating Comparative Genome Data and Gene Phylogenies," *BMC Bioinformatics,* vol. 4, no. 35, 2003.

Akshay Vashist received the BSc degree (physics) in 1994 from the University of Delhi and the MEng degree in computer science from the Indian Institute of Science, Bangalore, India. He is completing the PhD degree in the Computer Science Department at Rutgers-The State University of New Jersey. His research interests include algorithms, computational biology, and molecular evolution. He is a student member of the IEEE.

Casimir A Kulikowski received the BEng degree in 1965 and the MSc degree in engineering and applied science in 1966, both from Yale University. He received the PhD degree from the University of Hawaii in 1970 on the topic of pattern recognition methods for medical diagnosis. He is the Board of Governors Professor of Computer Science at Rutgers-The State University of New Jersey. His research focuses on pattern recognition, clustering and knowledge representation problems in medical informatics and bioinformatics, biomedical imaging, and the societal impact of computer technology. He is the author of more than 200 papers, two books, and coeditor of six other books. Professor Kulikowski headed the Rutgers Research Resource on AI in Medicine from 1984 to 1992, served as chair of the Computer Science Department at Rutgers University from 1984 to 1990, and was director of its Laboratory for Computer Science Research from 1985 to 1996. He is a member of the Institute of Medicine of the National Academy of Sciences (IOM-NAS) and a founding fellow of the American Academy of Medical Informatics (ACMI) and the American Association of Artificial Intelligence (AAAI). He is a fellow of the American Association for the Advancement of Science (AAAS), the IEEE, and the American Institute for Medical and Biological Engineering (AIMBE). At present he is coeditor of the *Yearbook of Medical Informatics* of the International Medical Informatics Association (IMIA), associate editor of the *Artificial Intelligence in Medicine Journal*, and is on the editorial board of *Methods of Information in Medicine*.

Ilya Muchnik received the MS degree in radio physics from the State University of Gorky (Nizhny Novgorod), Russia in 1960 and the PhD degree in technical cybernetics from the Institute of Control Sciences, National Academy of Science, Moscow, Russia, in 1972. He is a research professor at Rutgers-The State University of New Jersey. His research interests include large-scale optimization methods, high-dimensional data analysis, and machine understanding of structural objects. His current research focus is developing and applying machine learning and visualization methods to bioinformatics and epidemiological data. He has published more than 200 research papers. From 1960 to 1990, he worked at the Institute of Control Sciences, National Academy of Science, Moscow, Russia. During the early years of pattern recognition and machine learning, along with M. Braverman, he developed the first successful kernel-based method, the potential functions. He also developed algorithms for machine understanding of structural images and a method for shape-content contour pixel localization. He has developed many different combinatorial clustering models and worked on their applications in the socio-economic sciences, geology, biology, and medicine areas. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.