Efficient large-scale protein sequence comparison and gene matching to identify orthologs and co-orthologs

Khalid Mahmood1,2, Geoffrey I. Webb3, Jiangning Song1, James C. Whisstock1,2,* and Arun S. Konagurthu3,*

1Department of Biochemistry and Molecular Biology, 2ARC Centre of Excellence in Structural and Functional Microbial Genomics and 3Clayton School of Information Technology, Monash University, VIC 3800, Australia

Received September 17, 2011; Revised December 1, 2011; Accepted December 6, 2011

ABSTRACT

Broadly, computational approaches for ortholog assignment is a three steps process: (i) identify all putative homologs between the genomes, (ii) identify gene anchors and (iii) link anchors to identify best gene matches given their order and context. In this article, we engineer two methods to improve two important aspects of this pipeline [specifically steps (ii) and (iii)]. First, computing sequence similarity data [step (i)] is a computationally intensive task for large sequence sets, creating a bottleneck in the ortholog assignment pipeline. We have designed a fast and highly scalable sort-join method (afree) based on \( k \)-mer counts to rapidly compare all pairs of sequences in a large protein sequence set to identify putative homologs. Second, availability of complex genomes containing large gene families with prevalence of complex evolutionary events, such as duplications, has made the task of assigning orthologs and co-orthologs difficult. Here, we have developed an iterative graph matching strategy where at each iteration the best gene assignments are identified resulting in a set of orthologs and co-orthologs. We find that the afree algorithm is faster than existing methods and maintains high accuracy in identifying similar genes. The iterative graph matching strategy also showed high accuracy in identifying complex gene relationships. Standalone afree available from http://vbc.med.monash.edu.au/~kmahmood/afree. EGM2, complete ortholog assignment pipeline (including afree and the iterative graph matching method) available from http://vbc.med.monash.edu.au/~kmahmood/EGM2.

INTRODUCTION

Identifying orthologous relationships among genes between genomes of various organisms is an essential task in comparative genomics (1). This information is useful to identify shared evolutionary history as well as functional counterparts between shared segments in genomes. Recent advances in sequencing technologies have resulted in proliferation of genome data at unprecedented rates, widening the gap between annotated genes and those without any ascribed functional information. Computational methods for identifying gene orthologs between a pair of genomes usually involves three steps: (i) sequence similarity is calculated between all gene sequences in the genomes being compared; (ii) anchor or similar genomic regions in the form of gene strings are identified; (iii) finally, anchor regions are used to link genes that are potentially functional counterparts. In this article, we focus on two important steps (ii) and (iii) above: we engineer a fast and accurate method for calculating initial sequence similarities and next we develop a method that effectively identifies complex gene–gene relations.

Calculating initial sequence similarity, by all-against-all comparison, is an important foundation of the gene matching process. Techniques for sequence comparison can be broadly categorized into alignment-based and alignment-free methods. Alignment algorithms generally involve a dynamic programming step to produce an optimal match between molecular sequences [for extensive review see Ref. (2)]. Alignment-based algorithms are powerful and have been developed to detect similarity between molecular sequence (genes or proteins) at various levels; global or complete sequence comparison (3) and local or subsequence comparisons (4). Although several efficient implementations have been developed, the computational load for comparing a large number of sequences still poses a challenge for alignment based
and efficiently storing their similarity. The $k$-mer works on the hypothesis that similar sequences share all-against-all similarity search. Essentially, a method termed $afree$.

We engineer a new alignment-free sequence comparison in the context of automated gene matching. This section explains our alignment-free based method for sequence comparison and our graph theoretic approach for matching gene co-orthologs.

Alignment-free sequence comparison, $afree$

Broadly, alignment-free comparison is performed in two steps. In the first step, the number of shared $k$-mers are calculated. This is followed by a step where a statistical measure is employed to quantify the shared $k$-mer count to a similarity measure between each sequence pair. As is the case with gene matching across whole genomes involving a large volume of sequences, it is important to calculate the shared $k$-mer counts in an efficient manner. Our efficient, highly scalable method is explained below.

Definitions. Let $G_1$ and $G_2$ denote two genome sequences represented as a collection of $m$ and $n$ protein sequences,
respectively: \( G_1 = \{p_1, p_2, p_3, \ldots, p_m\} \) and \( G_2 = \{q_1, q_2, q_3, \ldots, q_n\} \). Any protein \( p = \{a_1, a_2, \ldots\} \) is a finite sequence of amino acid letters from the standard 20 letter alphabet. A \( k \)-mers from sequence \( p \) is any (contiguous) substring of size \( k \) that is permissible given the sequence length.

**Algorithm.** To perform an all-against-all comparison, the dataset of sequences in \( G_1 \) and \( G_2 \) are concatenated: \( G_1 \cup G_2 = \{p_1, \ldots, p_m, q_1, \ldots, q_n\} \). A sort-and-join strategy is used to efficiently build a shared \( k \)-mers count for each protein pair in the concatenated set followed by measuring pairwise similarity. This is achieved in three steps described below:

Step 1: assembling sequence words: a list \( L \) of tuples is constructed for \( G_3 \). Each tuple in the list consists of three fields: (i) the \( k \)-mer string, (ii) the protein sequence index it belongs to and (iii) the offset of the string from the start position in the protein sequence. Construction of the list is straightforward as it involves sliding along, with a window of length \( k \), the sequences in \( G_1 \) one by one. (To ensure efficiency in the join phase [see step (ii)] of the algorithm, for each sequence in \( G_3 \), only unique \( k \)-mers in that sequence are recorded in \( L \).) To allow our algorithm to scale to very large number of sequences, we pack each tuple in \( L \) into a 64-bit machine word, i.e. each tuple is encoded as a 64-bit integer. The tuple fields (\( k \)-mer, index and offset) are packed as follows: each amino acid letter in the \( k \)-mer is encoded in \( \lfloor \log_2 20 \rfloor = 5 \) bits. (We use a canonical ordering of amino acids and encode them as integers in the range \([0, 19]\).) The remainder of the 64-bit word is packed with the protein sequence index and the \( k \)-mer word offset.

Specifically in our implementation, we set aside the first \( k \times 5 < 32 \) (most-significant) bits in the 64-bit word to encode each \( k \)-mer. This allows a maximum \( k \)-mer length of 6 (occupying \( 6 \times 5 = 30 \) bits). In the remaining 34 bits, 14 least-significant bits are used to encode the index and the rest the protein sequence index. We note that this configuration allows us to compare, at its maximum, \( 2^{34} \approx 1 \) million protein sequences of length up to \( 2^{14} = 16384 \) amino acids each.

Next, the list \( L \) is sorted purely on the \( k \)-mer as the sort key. We implement a highly efficient least significant digit (LSD) radix sort (see ‘Results’ section). LSD radix sort for fixed-sized keys grows linearly (34). We use a radix size of 8 bits, which allows us to sort \( L \) on the \( k \)-mer key (encoded in the tuple) in a maximum of four linear sweeps through \( L \). (The number of sweeps through the list in sorting is automatically adjusted based on the user defined size of \( k \). For example, when \( k = 3 \), the sorting requires simply two sweeps.) A crucial factor for the efficiency achieved in our sorting phase is due to significant cache locality derived by economically representing \( L \) as an linear array of 64-bit words. (See Figure 1 for an illustration.)

Step 2: counting shared \( k \)-mers: let \( M = (m_{ij})_{0 \leq i < m, 0 \leq j < n} \) be a matrix where \( m_{ij} \) represents the number of \( k \)-mers that overlap between any two proteins \( p_i \) and \( p_j \) in \( G_3 \). The matrix \( M \) can be computed efficiently by performing a relational join of the sorted list \( L \) with itself (i.e. \( L \bowtie L \)).

The operation to compute the counts \( m_{ij} \) proceeds as follows: initially, a pointer \( ptr1 \) is set to point to the first tuple in the sorted list \( L \). Next pointer \( ptr2 = ptr1 + 1 \) is defined and is used to traverse \( L \) while \( L[ptr1].k\text{-mer} = L[ptr2].k\text{-mer} \) i.e. until the \( k \)-mer in the tuples pointed by \( ptr1 \) and \( ptr2 \) match. Together \( ptr1 \) and \( ptr2 \) define the range where equal \( k \)-mers words are found. Next, based on the range \([ptr1, ptr2]\) in the list \( L \), for every pair of tuples in that range containing the corresponding protein indices \((L[ptr1].index \equiv p_i \) and \( L[ptr2].index \equiv p_j)\), the counter \( m_{ij} \) is incremented. We then set \( ptr1 = ptr2 \) and \( ptr2 = ptr1 + 1 \) and repeat the above process until \( ptr1 \) reaches the end of the list \( L \). Figure 2 gives the pseudocode for the relational join operation described above.

A significant advantage of this join strategy on a sorted list is that the entire set of sequences in \( G_1 \) and \( G_2 \) are compared in a single process. Further, the join operation on sorted lists benefits from spatial locality of reference and remains mostly linear as its complexity depends quadratically on the size of the largest range of common \( k \)-mers in \( L \). Recall that for each sequence in \( G_3 \) only unique \( k \)-mers are used to populate the tuples in \( L \). That is, if a sequence contains a repetitive \( k \)-mer then only the first occurrence is recorded in the list. This significantly reduces the chances of ‘blowups’ in the join operation. (This can be inferred from Figure 4b where the wall
Therefore, sample sets. For two proteins index is a simple statistic to evaluate the similarity between (SD) (35,36) as a measure of similarity. The SD similarity similarity score based on the counts (37,38). Finally, the matrix similarity between two sequences on this idea (see earlier the underlying notion is that proteins with large SD similarity index has commonly been used in molecular comparisons among other complex evolutionary rearrangements, one-to-one relationships are not sufficient to fully map gene relationships between genomes. Here we describe a simple iterative matching strategy that identifies all gene matches or co-orthologs between genomes.

**Determining Co-orthologs**

The gene matching pipeline described here builds on the EGM method (22). EGM models the comparison between two genomes as a bipartite graph with weighted edges (each node is a protein sequence in the genome). Each edge between proteins is weighted on their sequence similarity and reinforced when a continuous stretch of proteins in the two genomes share a prescribed level of similarity (gene segments or strings are more likely to be conserved). Taken together, EGM uses a segment length-dependant edge weighting scheme (derived from sequence similarity and gene context) to calculate gene matching by transforming the task to a linear assignment problem. EGM then employs the Hungarian method for weighted bipartite graph matching to identify best gene matches i.e. genes are matched that maximize the synteny given the weights. However, this method has a limitation, where gene matches follow the simple one-to-one relationship. In the case of large and complex genomes with segmental duplications among other complex evolutionary rearrangements, one-to-one relationships are not sufficient to fully map gene relationships between genomes. Here we describe a simple iterative matching strategy that identifies all gene matches or co-orthologs between genomes.

**Step 3: similarity measures:** A similarity measure is defined as a function that quantifies the k-mer counts between any pair of protein sequences. As mentioned earlier the underlying notion is that proteins with large shared k-mer counts are likely to be similar. A number of methods have been previously proposed to calculate the similarity between two sequences on this idea (see ‘Introduction’ section).

Here we implement the Sorensen-Dice similarity index (SD) (35,36) as a measure of similarity. The SD similarity index is a simple statistic to evaluate the similarity between sample sets. For two proteins $p_i$ and $p_j$ of lengths $l_i$ and $l_j$, respectively, the total number of k-mers in the two sequences are $K_i = l_i - k + 1$ and $K_j = l_j - k + 1$, respectively. Therefore, $K_i$ and $K_j$ form two samples of k-mers. While $m_{ij}$ is the number of k-mers shared between the two sample sets. Given $K_i$, $K_j$ and $m_{ij}$, the SD similarity index is calculated as:

$$SD = \frac{200m_{ij}}{K_i + K_j}$$

We note that SD takes the value in the range [0, 100], where a score of 100 indicates identical sequences. The SD similarity index has commonly been used in molecular sequence comparison and retrieval programs such as Refs (37,38). Finally, the matrix $M$ is updated to store SD similarity score based on the counts $m_{ij}$.

We additionally implement several strategies to improve the accuracy of the similarity score. For instance, two sequences are only compared if the length of the shorter sequences is above a prescribed proportion of the longer sequence. This reduces the chances of potentially misleading matches based on small motif or domain level similarity. From the perspective of efficiency, a large number of comparisons are filtered and only comparisons of interest (in terms of gene matching) are performed.

**Clock times of the join step across several genome-wide comparisons are plotted.**

**Figure 2.** Join phase. Pseudocode for the list join [Step (ii)] to count the number of shared k-mers.
‘one-to-one’ assignment in the bipartite graph i.e. one gene is matched with one other gene. In order to identify co-orthologs, however, an iterative Hungarian algorithm strategy provided a good solution (41). Briefly, the iterative approach works in the following manner.

1. Apply the Hungarian method to \( W \) with respect to the current matching \( C \) to produce a set of matching. In other words, the first iteration of the Hungarian algorithms would results in a current gene match set \( C \subseteq E \).

2. For each matched edge in \( C = (p_{i1}, q_{j1}), \ldots, (p_{ik}, q_{jk}) \) where \( p \in G'_{1} \) and \( q \in G'_{2} \), the edge weights \( (w_{a,b}, \ldots, w_{i,j}) \) in \( W \) are set to \( \infty \). This ensures that the current gene matching is not the ‘best’ available match for the next iteration.

3. The modified weight matrix \( W \) is then used as input for the next graph matching iteration [steps (i) and (ii)] until a specified number of iterations are performed or no more matches are found.

This results in the identification of multiple gene matches that collectively form a comprehensive set of one-to-many or many-to-many gene relationships, thus identifying putative co-orthologs.

RESULTS AND DISCUSSION

Two main algorithmic enhancements to the EGM gene matching pipeline have been described here; (i) an automated alignment-free sequence comparison method and (ii) a method to identify co-orthologs. The aim of these enhancements is to fully automate the task of matching gene orthologs between genomes. To evaluate the effectiveness and performance of our methods, we have performed several experiments. Comparisons were performed at two levels. On a relatively smaller scale, we compared the gene set of Human chromosome X versus Mouse chromosome X, Human chromosome 20 versus Mouse chromosome 2, *Mycobacterium Tuberculosis* versus *Mycobacterium Leprae*. On a larger scale, we compared the entire gene set in the Human, Mouse and Rat genomes. Protein sequences for the Human, Mouse and Rat genomes were obtained from the Integr8 database (42) and the *Mycobacterium* datasets were obtained from the NCBI Genbank database.

The performance of *afree* was evaluated against BLAST as well as the recently published UBLAST (10) tools.

Evaluating *afree*

The BLAST tool is a standard method used for calculating sequence similarity between protein sequences and is commonly employed in gene matching to calculate an initial sequence similarity matrix (22). Therefore, we used data generated with BLAST for evaluating *afree* against the state of the art UBLAST (10). The UBLAST tool performs rapid sequence search and works on the hypothesis that most sequence comparisons are not essential and only a few top hits (sequences with high number of common \( k \)-mers) can be compared to improve performance. This feature is described by the maximum targets feature in the tool. The performance is evaluated in terms of precision and recall values. Sequence matches identified by BLAST are considered true. Precision is defined as the fraction of matches identified that are true.

\[
\text{precision} = \frac{\text{true positive matches}}{\text{total reported matches}}
\]

While, recall is defined as the fraction of all true matches that are identified.

\[
\text{recall} = \frac{\text{true positive matches}}{\text{total blast matches}}
\]

The BLAST program was executed using default parameters and the E-value threshold set at 0.001. Both *afree* and UBLAST were executed using the \( k = 5 \). For a more comprehensive analysis, UBLAST searches were performed using both the complete all-against-all search (from now on denoted as UBLAST) as well as the maximum hits set at 50 (from now on denoted as UBLAST50) and similarly, the E-value threshold set at 0.001. The *afree* tool uses a threshold on SD score to determine sequence similarity. First we performed analysis to understand the relationship between SD scores and percentage identity from sequence alignments. Next, we analyzed the relationship between the SD score and pairwise sequence identity as reported by BLAST. Figure 3 shows the relationship between the SD score and percentage sequence identity. It is evident that sequence identity and the SD distance score are related and their relation can be approximately formalized by the following function, where \( \lambda \) is an empirically determined scaling parameter:

\[
f(SD) = \begin{cases} 
\exp(-\lambda/SD) \times 100 & \text{if SD > 0} \\
0 & \text{if SD = 0}
\end{cases}
\]

![Figure 3. SD similarity measure. This figure illustrates the relationship between the SD score and the corresponding pairwise sequence identity (attained from BLAST). The Human and Mouse chromosome X comparison was used to perform this analysis with \( k = 5 \) and \( SD > 15 \). It is clear that the SD statistic is correlated with the percentage sequence identity. Therefore, it is a reliable measure to quantify similarity based on shared \( k \)-mers.](https://academic.oup.com/nar/article-abstract/40/6/e44/2409483)
This table shows performance comparison between BLAST, UBLAST and \textit{afree} (see details in text). Several all-against-all sequence comparison were performed ranging from chromosome scale data to large whole genome scale data. Expectedly, BLAST proved to be the most time intensive. UBLAST and UBLAST50 provide a significant speed-up to BLAST, but our method \textit{afree} is the fastest. We also assessed the accuracy of UBLAST and \textit{afree} by comparing homologous pairs against BLAST output that was considered to be true positive homologs. The precision value shows that both UBLAST and \textit{afree} maintain high precision. However, \textit{afree} shows better ability in identifying a larger fraction of true positive homologs as indicated by the recall values. Overall, \textit{afree} provides a significant speed up to BLAST while maintaining a very high precision and recall.

For the purpose of gene matching, the aim is to identify closely related sequences. It is clear that \textit{k–mer} based distance measures are suitable; however, for more diverged sequences it is still not well understood if such measures are reliable, at least to the level of low sequence identities (43). Therefore, in all cases two protein sequences are considered similar if their similarity measure (SD score, \textit{E}-value, sequence identity) is within a prescribed threshold and the length of the shorter of the two protein sequences is at least 50\% of the other (the proportionality threshold reduces the chance of inferring similarity based on short motifs). The SD score thresholds are determined empirically for each experiment.

Table 1 shows the results from five different comparisons between varying sizes of datasets. In all cases, four methods (BLAST, UBLAST, UBLAST50—with maximum targets set at 50 and \textit{afree}) were used to calculate all-against-all sequence comparison. The results show that \textit{afree} and UBLAST are significantly faster than BLAST. Expectedly, in comparison to UBLAST, the UBLAST50 showed a significant speed-up as fewer comparisons are performed. The UBLAST50 speed-up did not significantly affect the number of observed similar sequences (recall) in smaller datasets (chromosome level) while maintaining a high precision value (see Matches, precision and recall values in Table 1). However, in the case of larger datasets (Human, Mouse and Rat), we observed that UBLAST was able to identify considerably larger number of similar sequences in comparison to UBLAST50, again shown by the higher recall values, with a minor decrease in precision. Overall, however, our \textit{afree} implementation is the fastest of the methods compared and both UBLAST and \textit{afree} maintain similarly high precision. Further, in all comparisons our \textit{afree} tool was able to identify a larger number of true matches as represented by high recall values. This is important as the aim of such comparison is to identify maximum number of putative homology relationships providing a larger cover on the genomes being compared. Our \textit{afree} approach performs better than both the complete and maximum target versions of UBLAST, mainly because UBLAST performs the sequence comparison on task-by-task basis i.e. given a dataset of sequences, each sequence is iteratively used as a query and searched against the database. While our \textit{afree} approach achieves higher performance by calculating the comparison in a single bulk operation i.e. in a single iteration, all sequences are compared with every other sequence in the dataset.

\textit{afree} performance. Next, we analyze the efficiency of the \textit{afree} algorithm by examining individual components of the algorithm. In the first case, we assess the scalability of the sort phase. We randomly generate \textit{k–mer} list of size ranging from 1000 to 100 million tuples. Figure 4a shows the results of the experiments where it is evident that the sort time grows linearly with the list size, a highly desirable feature for the algorithm to be highly efficient. The largest list containing 100 million randomly generated \textit{k–mers} \((k = 5)\) was sorted in \(< 12\) s (note that the list for the Human versus Mouse comparison contains 21 million tuples).
Figure 4. afree performance. This figure illustrates the performances over time of the afree method. (a) Shows the efficiency and scalability of our sorting implementation list containing 100 million tuples is sorted in < 12 s (and in the Human versus Mouse comparison, list contains 21 million tuples). (b) Shows that the join step takes the largest proportion of time in the program execution.

Figure 4b shows the time taken by various phases in the algorithm on real data. The figure shows that the majority of the time is spent in the join phase whose complexity grows quadratically with to the size of the largest run (block) of k-mers that are common. However, even for the large genome scale comparisons such as Human versus Mouse, afree remains the fastest of the methods compared.

Identifying co-orthologs using iterative EGM

Next we performed experiments to evaluate the performance of various initial sequence similarity data inputs to our EGM ortholog identification pipeline (see 'Determining co-orthologs' section). Five comparisons, as described in the previous paragraph, were performed with initial sequence similarity data attained from BLAST (denoted as EGM_B), UBLAST (denoted as EGM_U) and the afree (denoted as EGM_Af) approach. Note that UBLAST50 was not used as the aim of this experiment is to identify the maximum number of orthologs and further, UBLAST50 ⊆ UBLAST. We used the EnsEMBL Compara (44) dataset of orthologs to assess the approximate accuracy of identifying orthologs and co-orthologs using the various similarity data sources (no Compara data available for the Mycobacterium spp). The number of identified orthologs overlapping with the Compara data were considered true positives. We empirically determined that in most cases four iterations were sufficient to identity orthologs (see Figure 5, where each iteration reveals unique pairs of putative orthologs represented by different colors).

Table 2 summarizes the results. The results show that EGM_B identified the largest number of orthologs among the three sources. Further, the average sequence identity of the EGM_B orthologs is 79.73%, lower than both EGM_U (82.4%) and EGM_Af (83.38%) based orthologs,
Table 2. Iterative EGM for identifying complex orthologous relationships

<table>
<thead>
<tr>
<th>Comparisons</th>
<th>Method</th>
<th>Matches per iteration</th>
<th>Total</th>
<th>TP</th>
<th>Precision</th>
<th>Recall</th>
<th>Avg. id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human Chr. X versus Mouse Chr. X</td>
<td>EGM_B</td>
<td>448</td>
<td>90</td>
<td>28</td>
<td>22</td>
<td>588</td>
<td>418</td>
</tr>
<tr>
<td></td>
<td>EGM_U</td>
<td>358</td>
<td>64</td>
<td>23</td>
<td>15</td>
<td>460</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>EGM_AF</td>
<td>395</td>
<td>43</td>
<td>5</td>
<td>3</td>
<td>446</td>
<td>373</td>
</tr>
<tr>
<td>Human Chr. 20 versus Mouse Chr. 2</td>
<td>EGM_B</td>
<td>365</td>
<td>14</td>
<td>3</td>
<td>1</td>
<td>383</td>
<td>344</td>
</tr>
<tr>
<td></td>
<td>EGM_U</td>
<td>295</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>305</td>
<td>276</td>
</tr>
<tr>
<td></td>
<td>EGM_AF</td>
<td>347</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>358</td>
<td>324</td>
</tr>
<tr>
<td>Mycobacterium tuberculosis versus M. leprae</td>
<td>EGM_B</td>
<td>1390</td>
<td>92</td>
<td>31</td>
<td>15</td>
<td>1528</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>EGM_U</td>
<td>1273</td>
<td>65</td>
<td>20</td>
<td>11</td>
<td>1369</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>EGM_AF</td>
<td>1358</td>
<td>71</td>
<td>18</td>
<td>9</td>
<td>1456</td>
<td>–</td>
</tr>
<tr>
<td>Human versus Mouse</td>
<td>EGM_B</td>
<td>15155</td>
<td>6107</td>
<td>3436</td>
<td>2218</td>
<td>26916</td>
<td>14485</td>
</tr>
<tr>
<td></td>
<td>EGM_U</td>
<td>12303</td>
<td>3877</td>
<td>2222</td>
<td>1651</td>
<td>20553</td>
<td>11658</td>
</tr>
<tr>
<td></td>
<td>EGM_AF</td>
<td>14018</td>
<td>5003</td>
<td>2645</td>
<td>1543</td>
<td>23209</td>
<td>13505</td>
</tr>
<tr>
<td>Mouse versus Rat</td>
<td>EGM_B</td>
<td>17230</td>
<td>7618</td>
<td>4877</td>
<td>3610</td>
<td>33333</td>
<td>17187</td>
</tr>
<tr>
<td></td>
<td>EGM_U</td>
<td>15443</td>
<td>5262</td>
<td>3490</td>
<td>2801</td>
<td>26996</td>
<td>15199</td>
</tr>
<tr>
<td></td>
<td>EGM_AF</td>
<td>16298</td>
<td>6665</td>
<td>4158</td>
<td>2933</td>
<td>29994</td>
<td>16046</td>
</tr>
</tbody>
</table>

This table shows the performance of the iterative EGM pipeline using different input homology data sources (BLAST → EGM_B, UBLAST → EGM_U, afree → EGM_AF). Various gene matching comparisons were performed with four graph matching iterations. Identified orthologs that overlapped with Compara dataset of orthologs were considered true positives (TP). In all comparisons, EGM_B identified the most true orthologs, however, showed the lowest precision especially for large genome-scale comparisons. EGM_U was next best, followed by EGM_AF, which resulted in higher precision, respectively. Together with the recall values, the data suggest that the iterative EGM pipeline is a reliable and effective method for identifying complex ortholog and co-ortholog relationships. Further, the data shows that afree and UBLAST provide reliable alternatives to BLAST based input data to gene matching pipelines. (No compara orthologs are available for the Mycobacterium genomes above).

Table 3. Ensembl Compara co-orthologs (Human versus Mouse)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>EGM_B</th>
<th>EGM_U</th>
<th>EGM_AF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>836</td>
<td>697</td>
<td>744</td>
</tr>
<tr>
<td>2</td>
<td>612</td>
<td>501</td>
<td>552</td>
</tr>
<tr>
<td>3</td>
<td>261</td>
<td>234</td>
<td>220</td>
</tr>
<tr>
<td>4</td>
<td>175</td>
<td>159</td>
<td>137</td>
</tr>
<tr>
<td>Total</td>
<td>1884</td>
<td>1591</td>
<td>1653</td>
</tr>
</tbody>
</table>

The table shows the performance of different input similarity data (BLAST → EGM_B, UBLAST → EGM_U, afree → EGM_AF) in determining Human and Mouse co-orthologs. The data suggests that more distantly related proteins are matched. In comparison to EGM_U, the EGM_AF approach leads to a higher number of ortholog matches. The higher count means that a larger proportion of the genome is matched which is important especially in genome annotation. In terms of precision (fraction of true identified orthologs), we observed that the EGM_AF and EGM_U approaches lead to higher precision compared with BLAST. These data suggest that k-mer-based approaches are robust enough to accurately determine orthologs at varying genome sizes. Among EGM_U and EGM_AF, the precision values were comparable, except the Human Chr. X versus Mouse Chr. X and Mouse versus Rat comparisons, where EGM_AF and EGM_U performed better, respectively. However, the EGM_AF approach maintained a higher orthologs count among all comparisons. The precision with the Compara dataset revealed that the iterative Hungarian method strategy (see ‘Determining co-orthologs’ section) is able to accurately and robustly match gene orthologs and co-orthologs.

We further analyze the orthologs identified in the Human versus Mouse comparison (21461 and 23202 genes, respectively). The new iterative EGM pipeline identified a total of 26916 orthologs based on the BLAST similarity data (Table 2), with 14485 of these gene matches overlapping with the Compara orthologs. Similarly, EGM_U and EGM_AF lead to 11658 and 13505 true positive orthologs. Next we analyze these true orthologs to assess EGM’s ability to identify co-orthologs. The Compara datasets represents the cardinality of orthologs relationships as being ‘one2one’, ‘one2many’ or even ‘many2many’, with the later two representing co-ortholog relationships. Table 3 lists the number of co-orthologs correctly identified by EGM at each iteration. Expectedly, EGM_B identified the most co-orthologous relationships, ahead of EGM_AF and EGM_U, respectively.

We also evaluate the overall performance of the iterative EGM by comparing the orthologs identified by gene matching approaches DAGchainer (19) and ADHoRe (20). Table 4 shows the results for the Human versus Mouse and Mouse versus Rat comparisons performed using EGM_AF, DAGchainer and ADHoRe.
Both DAGchainer and ADHoRe were run with several parameters to achieve the highest count of gene matches. The DAGchainer comparisons were carried out using parameters: \(-s -I -E 6 -A 3 -g 50000\). Similarly, ADHoRe experiments were carried out using parameters \(r2\text{-cutoff} = 0.9\ max\_dist = 20\). Both DAGchainer and ADHoRe compute all significant matching genomic segments between genomes. We extract all gene matches within these segments resulting in a list of gene matches. In all cases, the iterative EGM performed better than DAGchainer and ADHoRe in terms of the number of computed orthologs. However, ADHoRe maintained higher precision with fewer orthologs. Next we analyzed the ability of three methods to identify co-orthologs (by comparing against the Ensembl Compara data, see above). In both the Human versus Mouse and Mouse versus Rat comparisons, EGM identified the highest number of true positive co-orthologs.

### CONCLUSION

We have developed and tested methodologies that significantly help fully automate the non-trivial tasks of gene matching and identification of homologous genomic segments. Previously, methods for gene matching such as DAGchainer (19), ADHoRe (20) and EGM (22) rely on external applications such as BLAST to provide input (initial all-against-all similarity) data. However, the use of external software such as BLAST is resource intensive and the computational time requirements creates a bottleneck in the ortholog identification pipeline, especially for large-scale datasets. In addition, external applications pose technical difficulties from various, often complicated, software parameters to manual post-processing to extract the required data (22). To this end, our alignment-free approach \texttt{afree} provides a robust and rapid tool for fast sequence comparisons. The results show that \texttt{afree} is fast and accurate for smaller chromosome to larger genome scale data. We compared \texttt{afree} to the traditionally used BLAST application to assess the accuracy and saw that \texttt{afree} maintains high precision while providing a significant speed-up. Further comparisons between \texttt{afree} and UBLAST (a state of the art sequence search tool) shows that although the two maintain similar high precision, \texttt{afree} correctly calculates a higher fraction of true positive matches (recall) with better performance (Table 1). \texttt{afree} achieves high performance using a novel fast bulk comparison-based method that calculates similarity between every sequence pair in the data set. Current techniques, including UBLAST, perform sequence comparison in an iterative manner i.e. each sequence is iteratively used as a query to search against the reference dataset. Further, the goal of performing an all-against-all sequence search is to build a matrix of similarities where the actual pairwise sequence alignments are not required. Previously, the EGM pipeline (22) employed the Hungarian method for graph matching to calculate gene matching on a one-to-one basis (i.e. one gene is matched to only one other gene from the corresponding genomes). Earlier, such relationships were considered sufficient for identifying common gene ancestors (45). However, as mentioned earlier, sequence data from more complex genomes is fast becoming available making it increasingly desirable that more complex gene matches be used than simple one-to-one relationships (26–28). Larger complex genomes have higher tendency to contain large protein families and further, evolutionary events such as speciation and duplications especially in eukaryotes add to the complexity in correctly matching orthologous genes. In essence, now the task is not only to match ‘best’ gene matches (orthologs), instead the task is to identify all ‘best’ matches (co-orthologs) given the gene context and order information. To this end, our iterative EGM strategy based on the Hungarian algorithm has shown great potential. Results from our experiments show that the iterative EGM is effective and accurate in identifying true orthologs and co-orthologs (Tables 2 and 3). Comparison with DAGchainer and ADHoRe have also shown that our iterative algorithm is more effective in identifying co-orthologous relationships as defined by the compara ortholog data sets (Table 4).

| Table 4. Comparison between DAGchainer, ADHoRe and iterative EGM |
|----------------|----------------|--------------|
| Human versus Mouse | Mouse versus Rat |
| TP CO Recall | TP CO Recall |
| DAGchainer 9338 1017 0.23 | 11 169 1781 0.26 |
| ADHoRe 11 727 1249 0.28 | 11 716 1498 0.22 |
| EGM 13 505 1653 0.38 | 16 406 3574 0.53 |

The table presents the number of co-orthologs (CO) derived from the set of true positive (TP) matches identified by DAGchainer, ADHoRe and EGM in the Human versus Mouse and Mouse versus Rat comparisons. TP orthologs are those that overlap with the Ensembl Compara dataset and the number of co-orthologs are those that present one-to-many or many-to-many relationships. It is evident from the recall values that EGM is able to identify the highest proportion of true positive orthologs. In terms of total gene matches, ADHoRe (initial input from BLAST) results show high precision (avg. 0.68) with DAGchainer (initial input from BLAST) and EGM showing similar precision (avg. 0.57). The data suggest that \texttt{afree} together with the iterative EGM strategy is an highly efficient and accurate method for identifying co-orthologous gene relationships.
ACKNOWLEDGEMENTS

The authors acknowledge: Australian Research Council (ARC) Centre of Excellence in Structural Functional Microbial Genomics for support; Monash e-Research Centre and the Victorian Bioinformatics Consortium for computational resources.

FUNDING

K.M. is a PhD Student supported by ARC scholarship. J.S.’s research is funded by National Health and Medical Research Council (NHMRC) Peter Doherty Fellowship. J.C.W. is an ARC Federation Fellow and Honorary NHMRC Principle Research Fellow. A.S.K’s research is supported by Monash Larkins Fellowship. Funding for open access charge: Australian Research Council.

Conflict of interest statement. None declared.

REFERENCES


