Published online 30 December 2011

Nucleic Acids Research, 2012, Vol. 40, No. 6 ***e44***
doi:10.1093/nar/gkr1261

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

Khalid Mahmood[1,2], Geoffrey I. Webb[3], Jiangning Song[1], James C. Whisstock[1,2,*] and Arun S. Konagurthu[3,*]

[1]Department of Biochemistry and Molecular Biology, [2]ARC Centre of Excellence in Structural and Functional Microbial Genomics and [3]Clayton 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 [(i) 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

*To whom correspondence should be addressed. Tel: +61 3 9905 3227; Fax: +61 3 9905 5400; Email: arun.konagurthu@monash.edu
Correspondence may also be addressed to James C. Whisstock. Email: james.whisstock@monash.edu

approaches (5). Later, heuristics-based alignment approaches [such as FASTA (6) and BLAST (7,8)] were proposed to overcome the computational costs. In general, these methods start by compiling a list of substrings (also termed *k*–mers) of certain length. The sequence database is then scanned to identify sequences that contain these words. Finally, matched words are extended to maximize the match length until the score falls below the threshold. Heuristic alignment methods are efficient when searching a large database of sequences. However, in the context of methods for calculating orthologs, it is only required to identify all sequence that share a prescribed similarity [step (i) above] and not the sequence alignment itself. Further, in the context of comparative genomics, tools used to perform this task are usually very laborious both in terms of time and manual intervention (for example, data formatting). For example, a typical sequence set of human (~22 000) and mouse (~23 000) genes would result in millions of hits using the BLAST tool (taking ~10 h on a desktop computer). Therefore, it is desirable for gene matching pipelines in comparative genomics to provide a fast and efficient alternative to external software. Alignment-free methods for sequence comparison provide an efficient alternative. Recently, such techniques have gained popularity for large-scale comparison of biological data. Alignment-free methodologies have been applied to traditional sequence comparison and clustering of molecular sequences, searching regulatory and transcription factor binding sequence motifs in genome sequences, and large-scale phylogeny studies such as that involving 884 prokaryote organisms (9–14). Recently, alignment-free methodologies have also shown promise in searching for local and global structural similarities in the growing protein structure databases (15). Alignment-free methods are based on the underlying hypothesis that two similar sequences share a prescribed proportion of *k*–mers (16). These methods generally work by calculating the number of shared *k*–mers between a sequence pair, followed by calculating a statistical similarity measure based on these words. Several alignment-free sequence comparison methods have been been proposed. [See Ref. (5) for an extensive review].

In this work, first, we explore alignment-free sequence comparison in the context of automated gene matching. We engineer a new alignment-free sequence comparison method termed `afree` that can rapidly perform all-against-all similarity search. Essentially, `afree` works on the hypothesis that similar sequences share *k*–mers, or in other words, the higher the number of common *k*–mers between two sequences, the higher is their similarity. The `afree` method works by calculating and efficiently storing *k*–mers for every sequence in the dataset. This is followed by a series of calculations and heuristics that, in a single process, calculates the similarity between every sequence pair in the dataset. This is different from current methods that perform large scale comparisons based on a series of pairwise sequence comparisons (9,10).

The next focus of this study is to develop a method that can help understand complex gene relationships. Several computational algorithms have been developed to match genes between a pair of genomes in an effort to identify orthologs including MAGIC (17), FISH (18), DAGchainer (19), ADHoRe (20), OSfinder (21) and EGM (22). As more and more complex genomes are being sequenced, especially eukaryotics genomes, identifying relationships between genes has become more complex. Identifying one-to-one orthologous relationships had generally been seen to be sufficient for guiding information about gene function and evolution for smaller related species (23,24). However, the task of assigning gene orthologs has become more complex with the availability of large genomes where evolutionary events such as segmental duplications as well as whole genome duplications following speciation are not uncommon. Such evolutionary scenarios often lead to two or more genes orthologous to one or more genes, known as *co-orthologous* genes (24–29). Therefore, identifying co-orthologs is an important comparative genomics task and is commonly used to discover and transfer experimental gene function information between mammals and model experimental systems. Example cases include Drosophila discs large (dlg) gene (30,31), *hox* cluster genes (32) and Fugu genes (29) like synapsin (SYN) (33). To achieve this, we develop an iterative graph matching approach which at each iteration identifies gene matches such that the sum total of similarity scores for all gene matches is maximized.

Hence, this article reports two advancements to the gene matching pipeline:

(1) A highly scalable alignment-free sequence comparison method for efficient detection of gene similarities.
(2) A new method to identify co-orthologous genes between genomes has been designed that will provide more insights to infer gene function and evolution.

The methods have been incorporated within the Encapsulated Gene-by-gene Matching (EGM) pipeline (22), resulting in a new, more efficient software, EGM2.

MATERIALS AND METHODS

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–mer 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: $G_3 = G_1 \cup G_2 = \{p_1, \ldots p_n, q_1, \ldots, q_n\}$. A sort-and-join strategy is used to efficiently build a shared k–mer 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_3 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 $\lceil \log_2 20 \rceil = 5$ 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 offset and the rest the protein sequence index. We note that this configuration allows us to compare, at its maximum, $2^{20} \approx 1$ million protein sequences of length up to $2^{14} = 16\,384$ 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 $M = (m_{ij})_{1 \le i,j \le m+n}$ 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. $L \bowtie L$)).

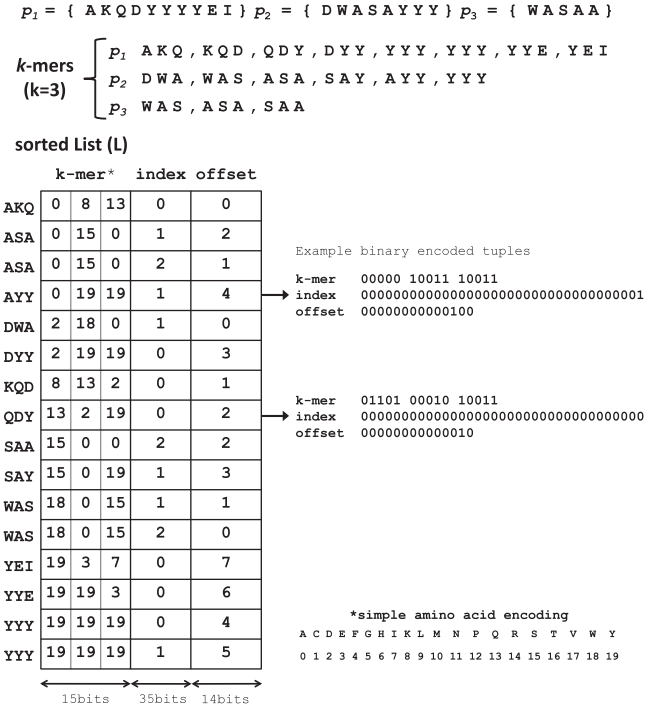

Figure 1. Constructing the sorted list of tuples. The figure illustrates an example construction of the sorted list of tuples L. Each tuple in L is composed of three fields: the k–mer, the index of the protein in the genome data and the offset from the start of the sequence. With $k = 3$, proteins p_1, p_2 and p_3 form 8, 6 and 3 k–mers, respectively. Binary form of the tuples AYY and QDY is depicted on the right. Also note that only unique k–mers are recorded in L to improve efficiency in the join phase of the algorithm [see step (ii)].

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–mer 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

Data: list L, matrix M (initialized to 0)
Result: M(i,j) → number of shared k–mers between any pair of sequences in G
Comment: L.size gives the size of list L. $L[x].k$-mer gives the k–mer field of the tuple at $L[x]$. Similarly $L[x].index$ gives the protein sequence index (in G) of that tuple.;

$Lsize \leftarrow L.size$;
$ptr1, ptr2 \leftarrow 0$;
while $ptr1 < Lsize$ $ptr2 < Lsize$ **do**
  $ptr2 \leftarrow ptr1 + 1$;
  while $ptr2 < Lsize$ $L[ptr2].k\text{-}mer = L[ptr1].k\text{-}mer$ **do**
    $ptr2 \leftarrow ptr2 + 1$;
  end
  for $x = ptr1$ **to** $ptr2$ **do**
    for $y = ptr1$ **to** $ptr2$ **do**
      $i \leftarrow L[x].index$;
      $j \leftarrow L[y].index$;
      $M(i,j) \leftarrow M(i,j) + 1$;
    end
  end
  $ptr1 \leftarrow ptr2$;
end

Figure 2. Join phase. Pseudocode for the list join [Step (ii)] to count the number of shared k–mers.

clock times of the join step across several genome-wide comparisons are plotted.)

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 *Sørensen-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:

$$\mathrm{SD} = \frac{200\mathrm{m_{ij}}}{K_\mathrm{i} + K_\mathrm{j}}$$

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.

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.

Define G_1' and G_2' as the encapsulated forms of the genome G_1 and G_2 . In short, encapsulation is a transformation of a genome from a set of gene sequences to a set of integers where each integer identifies its respective gene family. These gene families are computed using single-linkage clustering, where the measure of similarity between nodes (genes) is given in the matrix M. Further, define a weighted bipartite graph $G = (V \equiv G_1' \cup G_2', E)$, where V is the vertex set of the two disjoint encapsulated genomes (G_1' and G_2'), and E is the set of all edges between every node in the encapsulated genomes. To reduce the number of spurious and possibly non-orthologous matches mainly rising from non-homologous genomic segments, the matrix M is not used directly to extract edge weights for E . Instead, we define an *ad hoc* weight matrix $W = (w_{ij})_{1 \le i,j \le m+n}$, where $w(i, j)$ corresponds to an edge in E. The matrix W is computed using a seed-and-extend strategy that reinforces homologous gene segments (strings of genes) i.e. longer stretches of homologous genes (nodes) will have a larger weight. A more detailed description of the approach is presented in Ref. (22).

Given G and W, we compute gene matches using the Hungarian method (39) for maximum weight bipartite graph matching approach [see Ref. (40) for implementation details]. This produces the maximum weight

'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 $C = (p_a, q_b), \ldots, (p_i, q_j)$ 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 ∞. 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 of 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 term 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` approach 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 λ is an empirically determined scaling parameter:

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

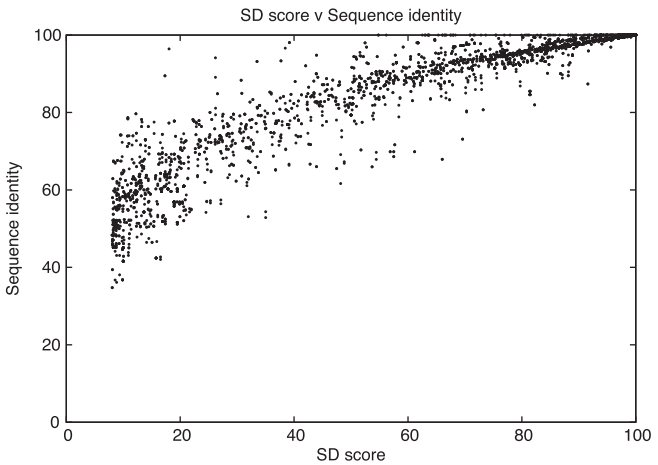

Figure 3. SD similarity measure. This figure illustrates the relationship between the SD score and the corresponding pairwise sequence identity (attained from BLAST). The Human and Mouse chromosome X comparison was used to perform this analysis with $k = 5$ and $SD > 15$. It is clear that the SD statistic is correlated with the percentage sequence identity. Therefore, it is a reliable measure to quantify similarity based on shared k–`mers`.

Table 1. Comparison between BLAST, UBLAST and afree

Comparisons	Method	SD score>	Time (min)	Matches	TP	Precision	Recall	Avg. id
Human Chr. X versus Mouse Chr. X	BLAST	–	1.8	6254	6254	1.000	1.000	88.1
	UBLAST	–	0.31	4984	4979	0.999	0.796	92.7
	UBLAST50	–	0.12	4984	4978	0.999	0.796	92.7
	afree	15	0.01	5364	5362	0.999	0.857	92.8
Human Chr. 20 versus Mouse Chr. 2	BLAST	–	5.9	21 441	21 441	1.000	1.000	64.9
	UBLAST	–	0.54	14 358	14 347	0.999	0.669	68.3
	UBLAST50	–	0.19	11 839	11 828	0.999	0.552	71.6
	afree	9	0.05	14 848	14 851	0.999	0.693	66.5
Mycobacterium tuberculosis versus *M. leprae*	BLAST	–	8.7	9955	9955	1.000	1.000	90.6
	UBLAST	–	1.73	8070	8044	0.997	0.808	94.4
	UBLAST50	–	0.4	8061	8039	0.997	0.808	94.5
	afree	10	0.05	9330	9310	0.998	0.935	93.1
Human versus Mouse	BLAST	–	600	352 209	352 209	1.000	1.000	67.7
	UBLAST	–	39.45	176 666	171 967	0.973	0.488	76.5
	UBLAST50	–	4.53	147 537	145 542	0.986	0.413	80.9
	afree	10	4.17	202 364	198 449	0.981	0.563	79.3
Mouse versus Rat	BLAST	–	560	573 787	573 787	1.000	1.000	66.4
	UBLAST	–	42.04	285 901	278 269	0.973	0.485	74.9
	UBLAST50	–	4.5	223 845	219 675	0.981	0.382	80.4
	afree	10	4.32	337 092	332 559	0.987	0.579	76.4

This table shows performance comparison between BLAST, UBLAST (all matches), UBLAST50 (maximum target = 50) and 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 afree is the fastest. We also assessed the accuracy of UBLAST and afree by comparing homologous pairs against BLAST output that was considered to be true positive homologs. The precision value shows that both UBLAST and afree maintain high precision. However, afree shows better ability in identifying a larger fraction of true positive homologs as indicated by the recall values. Overall, 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 *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, *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 afree) were used to calculate all-against-all sequence comparison. The results show that 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 afree implementation is the fastest of the methods compared and both UBLAST and afree maintain similarly high precision. Further, in all comparisons our 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 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 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.

afree performance. Next, we analyze the efficiency of the afree algorithm by examining individual components of the algorithm. In the first case, we assess the scalability of the sort phase. We randomly generate *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 *k*–mers ($k = 5$) was sorted in < 12 s (note that the list for the Human versus Mouse comparison contains 21 million tuples).

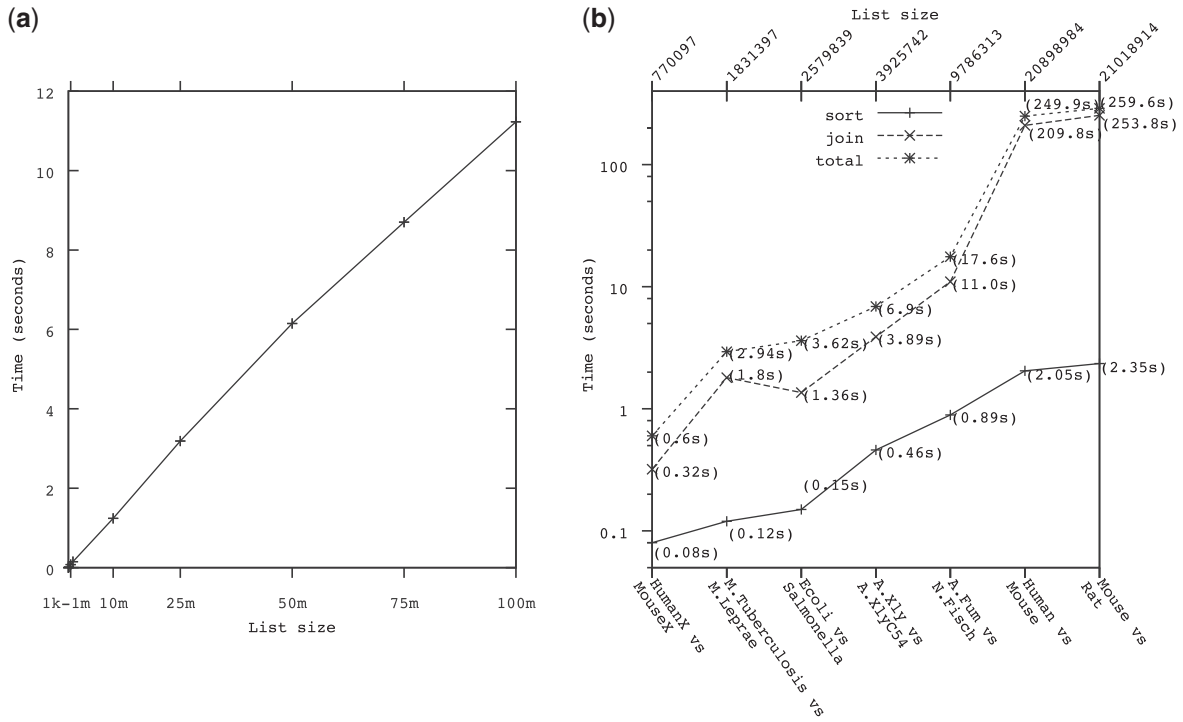

Figure 4. `afree` performance. This figure illustrates the performances over time of the `afree` method. (**a**) Shows the efficiency and scalability of our sorting implementation list containing 100 million tuples is sorted in < 12 s (and in the Human versus Mouse comparison, list contains 21 million 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 $\subseteq$ 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).

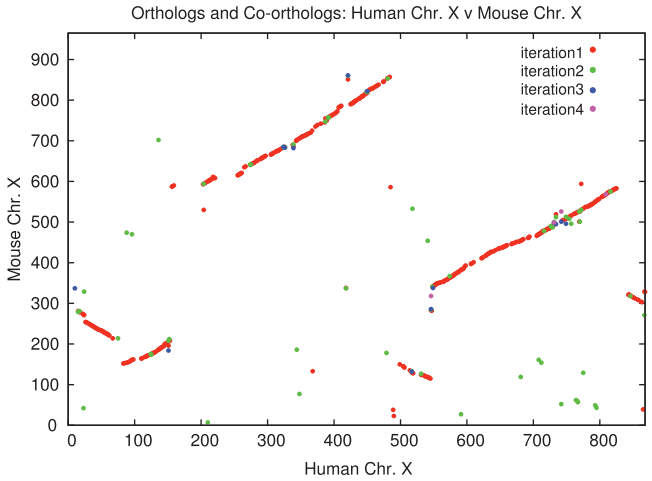

Figure 5. Iterative EGM. This figure illustrates the gene matching calculated by the iterative EGM pipeline. The Human and Mouse X chromosomes orthologs and co-orthologs are presented as a dotplot where each dot represents a gene match and colors represent the iteration (Iteration1-red: 424 orthologs with average sequence identity of 85.1%, similarly, iteration2-green: 71,69,7%, iteration3-blue: 14, 63.2% and iteration4-purple: 5, 74.2%).

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

Comparisons	Method	Matches per iteration				Total	TP	Precision	Recall	Avg. id
		1	2	3	4					
Human Chr. X versus Mouse Chr. X	EGM_B	448	90	28	22	588	418	0.711	0.60	78.9
	EGM_U	358	64	23	15	460	335	0.728	0.48	81.1
	EGM_{AF}	395	43	5	3	446	373	0.836	0.54	85.9
Human Chr. 20 versus Mouse Chr. 2	EGM_B	365	14	3	1	383	344	0.898	0.75	83.2
	EGM_U	295	8	1	1	305	276	0.905	0.61	85.4
	EGM_{AF}	347	11	0	0	358	324	0.905	0.71	84.7
Mycobacterium tuberculosis versus *M. leprae*	EGM_B	1390	92	31	15	1528	–	–	–	79.7
	EGM_U	1273	65	20	11	1369	–	–	–	80.8
	EGM_{AF}	1358	71	18	9	1456	–	–	–	80.5
Human versus Mouse	EGM_B	15 155	6107	3436	2218	26 916	14 485	0.538	0.74	76
	EGM_U	12 303	3877	2222	1651	20 053	11 658	0.581	0.60	79.5
	EGM_{AF}	14 018	5003	2645	1543	23 209	13 505	0.582	0.69	79.4
Mouse versus Rat	EGM_B	17 230	7618	4877	3610	33 335	17 187	0.516	0.82	80.8
	EGM_U	15 443	5262	3490	2801	26 996	15 199	0.563	0.73	83.9
	EGM_{AF}	16 298	6605	4158	2933	29 994	16 406	0.542	0.78	83.5

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 whole genome-scale comparisons. EGM_{AF} was the next best in terms of the number of true orthologs while maintaining high precision. Both EGM_{AF} and EGM_U maintained similar high precision, except in the Human Chr. X versus Mouse Chr. X and Mouse versus Rat comparisons where EGM_{AF} and EGM_U 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)

Iteration	Co-orthologs		
	EGM_B	EGM_U	EGM_{AF}
1	836	697	744
2	612	501	552
3	261	234	220
4	175	159	137
Total	1884	1591	1653

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. Each gene mapping iteration presents the number oftrue positive co-orthologs (one-to-many or many-to-many relationships identified by Ensembl Compara). Expectedly, EGM_B identifies the highest number of co-orthologs while EGM_{AF} is next best followed by EGM_U. These data suggest that BLAST based data maybe able to identify distant homology relationships more than the UBLAST and `afree` approaches, where the role of shared *k*–`mers` in identifying distant homology is not well understood.

suggesting 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), interestingly, 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 comparison 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 (21 461 and 23 202 genes, respectively). The new iterative EGM pipeline identified a total of 26 916 orthologs based on the BLAST similarity data (Table 2), with 14 485 of these gene matches overlapping with the Compara orthologs. Similarly, EGM_U and EGM_{AF} lead to 11 658 and 13 505 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.

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_{AF}	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_{AF} in the Human versus Mouse and Mouse versus Rat comparisons. TP orthologs are those matches 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_{AF} is able to identify the highest proportion of true positive co-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_{AF} (initial input from `afree`) showing similar precision (avg. 0.57). The data suggest that `afree` together with the iterative EGM strategy is an highly efficient and accurate method for identifying co-orthologous gene relationships.

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 -Z 6 -A 3 -g 50000`. Similarly, ADHoRe experiments were carried out using parameters `r2-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 DAGc hainer 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 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 algorithms 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 `afree` provides a robust and rapid tool for fast sequence comparisons. The results show that `afree` is fast and accurate for smaller chromosome to larger genome scale data. We compared `afree` to the traditionally used BLAST application to assess the accuracy and saw that `afree` maintains high precision while providing a significant speed-up. Further comparisons between `afree` and UBLAST (a state of the art sequence search tool) shows that although the two maintain similar high precision, `afree` correctly calculates a higher fraction of true positive matches (recall) with better performance (Table 1). `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 identity 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).

Summarizing, we have devised new methodologies in order to fully automate the task of rapid gene matching between genomes. The algorithms are shown to be efficient and accurate. The methods have been implemented into the new EGM2 pipeline that provides significant performance gains to the previous version. In addition, our alignment free sequence comparison tool is incorporated into the EGM pipeline, making it, to the best of our knowledge, the first fully independent and automated tool for large-scale gene matching for ortholog identification. As a result, we believe that the methods and tools described here will significantly improve the efficiency in the way biologists study how genes evolve and eventually their function.

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

1. Kriventseva,E.V., Rahman,N., Espinosa,O. and Zdobnov,E.M. (2008) OrthoDB: the hierarchical catalog of eukaryotic orthologs. *Nucleic Acids Res*., **36**, D271–D275.
2. Vingron,M. and Waterman,M.S. (1994) Sequence alignment and penalty choice: review of concepts, case studies and implications. *J. Mol. Biol*., **235**, 1–12.
3. Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol*., **48**, 443–453.
4. Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol*., **147**, 195–197.
5. Vinga,S. and Almeida,J. (2003) Alignment-free sequence comparison-a review. *Bioinformatics*, **19**, 513–523.
6. Pearson,W.R. (1990) Rapid and sensitive sequence comparison with fastp and fasta. *Methods Enzymol*, **183**, 63–98.
7. Altschul,S.F., Madden,T.L., Schaffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*., **25**, 3389–3402.
8. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol*., **215**, 403–410.
9. Li,W. and Godzik,A. (2006) CD-HIT: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**, 1658–1659.
10. Edgar,R.C. (2010) Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, **26**, 2460–2461.
11. Koohy,H., Dyer,N.P., Reid,J.E., Koentges,G. and Ott,S. (2010) An alignment-free model for comparison of regulatory sequences. *Bioinformatics*, **26**, 2391–2397.
12. Gordân,R., Narlikar,L. and Hartemink,A.J. (2010) Finding regulatory dna motifs using alignment-free evolutionary conservation information. *Nucleic Acids Res*., **38**, e90.
13. Xu,M. and Su,Z. (2010) A novel alignment-free method for comparing transcription factor binding site motifs. *PLoS One*, **5**, e8797.
14. Jun,S.R., Sims,G.E., Wu,G.A. and Kim,S.H. (2010) Whole-proteome phylogeny of prokaryotes by feature frequency profiles: An alignment-free method with optimal feature resolution. *Proc. Natl Acad. Sci. USA*, **107**, 133–138.
15. Zhi,D., Shatsky,M. and Brenner,S.E. (2010) Alignment-free local structural search by writhe decomposition. *Bioinformatics*, **26**, 1176–1184.
16. Arunachalam,M., Jayasurya,K., Tomancak,P. and Ohler,U. (2010) An alignment-free method to identify candidate orthologous enhancers in multiple drosophila genomes. *Bioinformatics*, **26**, 2109–2115.
17. Swidan,F., Rocha,E.P., Shmoish,M. and Pinter,R.Y. (2006) An integrative method for accurate comparative genome mapping. *PLoS Comput. Biol*., **2**, e75.
18. Calabrese,P.P., Chakravarty,S. and Vision,T.J. (2003) Fast identification and statistical evaluation of segmental homologies in comparative maps. *Bioinformatics*, **19(Suppl. 1)**, i74–i80.
19. Haas,B.J., Delcher,A.L., Wortman,J.R. and Salzberg,S.L. (2004) Dagchainer: a tool for mining segmental genome duplications and synteny. *Bioinformatics*, **20**, 3643–3646.
20. Vandepoele,K., Saeys,Y., Simillion,C., Raes,J. and Van De Peer,Y. (2002) The automatic detection of homologous regions (ADHoRe) and its application to microcolinearity between arabidopsis and rice. *Genome Res*., **12**, 1792–1801.
21. Hachiya,T., Osana,Y., Popendorf,K. and Sakakibara,Y. (2009) Accurate identification of orthologous segments among multiple genomes. *Bioinformatics*, **25**, 853–860.
22. Mahmood,K., Konagurthu,A.S., Song,J., Buckle,A.M., Webb,G.I. and Whisstock,J.C. (2010) EGM: encapsulated gene-by-gene matching to identify gene orthologs and homologous segments in genomes. *Bioinformatics*, **26**, 2076–2084.
23. Remm,M., Storm,C.E. and Sonnhammer,E.L. (2001) Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J. Mol. Biol*., **314**, 1041–1052.
24. Sonnhammer,E.L.L. and Koonin,E.V. (2002) Orthology, paralogy and proposed classification for paralog subtypes. *Trends Genet*., **18**, 619–620.
25. Koonin,E.V. (2005) Orthologs, paralogs, and evolutionary genomics. *Ann. Rev. Genet*., **39**, 309–338.
26. Bandyopadhyay,S., Sharan,R. and Ideker,T. (2006) Systematic identification of functional orthologs based on protein network comparison. *Genome Res*., **16**, 428–435.
27. Dehal,P. and Boore,J.L. (2005) Two rounds of whole genome duplication in the ancestral vertebrate. *PLoS Biol*., **3**, e314.
28. Sjolander,K. (2004) Phylogenomic inference of protein molecular function: advances and challenges. *Bioinformatics*, **20**, 170–179.
29. Woolfe,A. and Elgar,G. (2007) Comparative genomics using fugu reveals insights into regulatory subfunctionalization. *Genome Biol*., **8**, R53.
30. Fukuhara,H., Masuda,M., Yageta,M., Fukami,T., Kuramochi,M., Maruyama,T., Kitamura,T., Murakami,Y. and Masvuda,M. (2003) Association of a lung tumor suppressor tslc1 with mpp3, a human homologue of drosophila tumor suppressor dlg. *Oncogene*, **22**, 6160–6165.
31. Sakarya,O., Armstrong,K.A., Adamska,M., Adamski,M., Wang,I.F., Tidor,B., Degnan,B.M., Oakley,T.H. and Kosik,K.S. (2007) A post-synaptic scaffold at the origin of the animal kingdom. *PLoS One*, **2**, e506.
32. Santini,S., Boore,J.L. and Meyer,A. (2003) Evolutionary conservation of regulatory elements in vertebrate hox gene clusters. *Genome Res*., **13**, 1111–1122.
33. Yu,W.P., Brenner,S. and Venkatesh,B. (2003) Duplication, degeneration and subfunctionalization of the nested synapsin-timp genes in fugu. *Trends Genet*., **19**, 180–183.
34. Kärkkäinen,J. and Rantala,T. (2009) Engineering radix sort for strings. In: Amir,A., Turpin,A. and Moffat,A. (eds), *String Processing and Information Retrieval*, Vol. 5280, of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, pp. 3–14.
35. Dice,L. (1945) Measure of the amount of ecologic association between species. *Ecology*, **26**, 297–302.
36. Sorensen,T. (1948) A method for establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Videnski Selskab Biologiske Skrifter*, **5**, 1–34.
37. Widmann,J., Hamady,M. and Knight,R. (2006) Divergentset, a tool for picking non-redundant sequences from large sequence collections. *Mol. Cell Proteomics*, **5**, 1520–1532.
38. Smith,L.H. and Wilbur,W.J. (2010) Finding related sentence pairs in medline. *Informat. Retr*., **13**, 601–617.
39. Kuhn,H.W. (1955) The hungarian method for the assignment problem. *Naval Res. Logist. Quart*., **2**, 83–97.

40. Papadimitriou,C.H. and Steiglitz,K. (1998) *Combinatorial Optimization: Algorithms and Complexity*. Courier Dover Publications, Mineola New York.
41. Bansal,A.K., Bork,P. and Stuckey,P. (1998) Automated pair-wise comparisons of microbial genomes. *Math. Model. Sci. Comput.*, **19**, 1–23.
42. Pruess,M., Kersey,P. and Apweiler,R. (2005) The integr8 project-a resource for genomic and proteomic data. *In Silico Biol.*, **5**, 179–185.
43. Edgar,R.C. (2004) Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Res.*, **32**, 380–385.
44. Flicek,P., Aken,B.L., Beal,K., Ballester,B., Caccamo,M., Chen,Y., Clarke,L., Coates,G., Cunningham,F., Cutts,T. *et al.* (2008) Ensembl 2008. *Nucleic Acids Res.*, **36**, D707–D714.
45. Sankoff,D. (1999) Genome rearrangement with gene families. *Bioinformatics*, **15**, 909–917.