Genome analysis

Efficient estimation of pairwise distances between genomes

Mirjana Domazet-Lošo1,2 and Bernhard Haubold1,*

1Department of Evolutionary Genetics, Max-Planck-Institute for Evolutionary Biology, 24306 Plön, Germany and 2Faculty of Electrical Engineering and Computing, Department of Applied Computing, University of Zagreb, Zagreb, Croatia

Received on August 4, 2009; revised on September 25, 2009; accepted on October 10, 2009
Advance Access publication October 13, 2009
Associate Editor: Martin Bishop

ABSTRACT

Motivation: Genome comparison is central to contemporary genomics and typically relies on sequence alignment. However, genome-wide alignments are difficult to compute. We have, therefore, recently developed an accurate alignment-free estimator of the number of substitutions per site based on the lengths of exact matches between pairs of sequences. The previous implementation of this measure requires $O(n^2)$ suffix tree constructions and traversals, where $n$ is the number of sequences analyzed. This does not scale well for large $n$.

Results: We present an algorithm to extract $O(n)$ pairwise distances in a single traversal of a single suffix tree containing $n$ sequences. As a result, the run time of the suffix tree construction phase of our algorithm is reduced from $O(n^2)$ to $O(n \log n)$, where $L$ is the length of each sequence. We implement this algorithm in the program kr version 2 and apply it to 825 HIV genomes, 13 genomes of enterobacteria and the complete genomes of 12 Drosophila species.

We show that, depending on the input dataset, the new program is at least 10 times faster than its predecessor.

Availability: Version 2 of kr can be tested via a web interface at http://guanine.evolbio.mpg.de/kr2/. It is written in standard C and its source code is available under the GNU General Public License from the same web site.

Contact: haubold@evolbio.mpg.de

Supplementary information: Supplementary data are available at Bioinformatics online.

1 INTRODUCTION

Evolutionary distances between homologous DNA sequences are traditionally computed from multiple sequence alignments. However, multiple sequence alignment belongs to the class of non-deterministic polynomial-time complete (NP-complete) problems (Aho et al., 1974, Ch. 10; Wang and Jiang, 1994). As a consequence, its solution becomes computationally unfeasible even for relatively small sets of short sequences (Carrillo and Lipman, 1988). This has motivated the heuristic solutions implemented in widely used programs including clustalw (Larkin et al., 2007), muscle (Edgar, 2004), MAFFT (Katoh et al., 2005) and MAVID (Bray and Pachter, 2004). In spite of much progress, the unabating growth of sequence data ensures that multiple genome alignment remains computationally challenging (Dewey and Pachter, 2006).

Depending on the desired degree of accuracy, it may be possible to estimate the evolutionary distances between sequences without a multiple alignment. Pairwise alignments such as those returned efficiently by the program MUltimer (Kurtz et al., 2004) might be sufficient. Alignment-free pairwise distances represent even more radical departure from the paradigm of sequence comparison based on multiple alignment.

Alignment-free distances were first described over two decades ago (Blaidsell, 1986; Vinga and Almeida, 2003). Today, with several 1000 genome projects under way, there is renewed interest in their superior efficiency. In a recent comprehensive study of distance-based phylogeny reconstruction, Höhl and Ragan (2007) compared 10 alignment-free and one alignment-based approach. Their first result was that for syntenic sequences the slow alignment-based metric was more accurate than any of the alignment-free metrics tested. Among the latter, approaches based on pattern matching gave the best results (Höhl and Ragan, 2007).

The pattern matching of the winning algorithms was based on the TEIRESIAS algorithm (Rigoutsos and Floratos, 1998), which runs in time proportional to the number of matches. Since this number grows rapidly with the lengths of the input sequences, pattern-based distances are unsuitable for genome comparison.

In the absence of high among-site rate variation, the best alignment-free method not based on pattern detection is the average common substring metric was more accurate than any of the alignment-free metrics tested. Among the latter, approaches based on pattern matching gave the best results (Höhl and Ragan, 2007).

Algorithmically simpler are traditional approaches based on word frequency counts, which can be computed very rapidly (Vinga and Almeida, 2003). Compared with the average common substring approach, word frequency methods have the disadvantage that the optimal word length needs to be determined a priori. However, if this choice is made carefully, such methods can also be applied to compute useful phylogenies for distantly related genomes (Sims et al., 2009).

Unfortunately, the branch lengths of phylogenies based on measures like the average common substring length or word frequencies are difficult to interpret as evolutionary distances, since these usually take the form of substitutions per homologous site. Because the alignment-free measures are not based on homology, they are generally not convertible to evolutionary distances.
One exception to this rule is pattern-based approaches, albeit at the cost of much increased run times (Höh and Ragan, 2007).

In an attempt to unite the efficiency of alignment-free approaches with the biological relevance of traditional distance measures, Haubold et al. (2009) have recently developed an alignment-free estimator of the number of nucleotide substitutions per site, \( K_r \). This is based on a quantity very similar to the average common substring length, the average shortest unique substring (shustring) length. Using a mathematical model of DNA sequence evolution, the average shustring length is converted to \( K_r \). Like the classical estimators of nucleotide substitution rates (Felsenstein, 2004, Ch. 13), the \( K_r \) has the disadvantage of being restricted to DNA sequences, but in return it is more accurate than model-free approaches including the average common substring metric (Haubold et al., 2009).

The original implementation of \( K_r \) in the program kr version 1 was based on \( n(n-1) \) suffix trees for the computation of all pairwise distances from \( n \) input sequences. This becomes slow for large \( n \). Here, we describe an algorithm for extracting all distances from a single suffix tree containing \( n \) sequences. We have implemented this approach in kr version 2 and demonstrate the superior speed of this program by comparing it with version 1 in applications ranging in size from 825 genomes of human immunodeficiency virus (HIV) strains to 12 complete genomes of Drosophila species.

\section*{2 SYSTEM AND METHODS}

\subsection*{2.1 Shustrings}

The substitution rate estimator \( K_r \) is based on the idea of pairwise shustrings, which were originally introduced to compare genomes (Haubold and Wehe, 2006a; Haubold et al., 2005). Consider two sequences, \( S_1 = \text{ACGTS} \) and \( S_2 = \text{ACGTS} \), which we can think of as \textit{query} and \textit{subject}, respectively. At every position \( i \) in \( S_1 \), we determine the shortest substring \( S_1[i..j] \) that is absent from \( S_2 \). For example, \( S_1[1..3] = \text{ACC} \) is the shortest substring starting at the first position in \( S_1 \) that is absent from \( S_2 \). The length of this shustring is 3 and in a similar way we look up the lengths of the shustrings at every position in \( S_1 \). \( K_r \) is a function of the average of these shustring lengths (Haubold et al., 2009).

Notice the sentinel character, \( \$ \), at the end of \( S_1 \). This character differs from every other character including itself and its addition guarantees that a unique suffix and, by implication, a shustring is defined at every position in \( S_1 \). However, the query/subject labeling influences the average shustring length, \( L_r \), in general \( L_r(S_1[j..i]) \neq L_r(S_2[j..i]) \), where \( \ell_1 \) is the sum of shustring lengths when \( S_1 \) is query and \( \ell_2 \) is the sum of shustring lengths when \( S_2 \) is query. Because of this asymmetry, we compute for each pair of sequences both possible average shustring lengths. As explained by Haubold et al. (2009), the final distance is the greater of the two possible \( K_r \) values, as this is conservative with respect to the effect of variation in the copy number of elements shared between query and subject.

The computation of shustring lengths constitutes a central part of the \( K_r \) calculation. These lengths are best looked up using a suffix tree (Gusfield, 1997).

\subsection*{2.2 Suffix tree}

Our input dataset consists of \( n \) sequences labeled by sequence identifiers \( 1,2,\ldots,n \). Each sequence comprises its forward and reverse strands. The \( n \) sequences are indexed using a generalized suffix tree, which represents every suffix contained in the dataset once. Figure 1 shows a generalized suffix tree of four sequences. In our particular example, the suffix tree indexes sequences where—apart from the sentinel—\( A \) is a prefix of \( S_4 \); \( S_1 = \text{AS} \); \( S_2 = \text{ACS} \); \( S_3 = \text{ACC} \) and \( S_4 = \text{ACCC} \). This enables us to demonstrate certain features of our traversal algorithm later on. Of course, in real applications there is no restriction on the sequence of \( S_i \) with respect to \( S_j \).

A given suffix \( S_j[i..j] \) is looked up in the suffix tree by concatenating the edge labels from the root to a terminal node (leaf) designated \( i_j \) (see black nodes in Fig. 1A). More generally, the concatenated edge label along the path leading from the root to any node \( x \) is called the \textit{path label} of \( x \). For example, ACC is the path label of \( x_1 \) (Fig. 1A).

Suffix trees have the useful property that all prefixes of path labels of branch nodes (gray nodes in Fig. 1A) are repeated somewhere in the dataset. Conversely, unique substrings run from the root to a terminal edge. A shustring starting at \( S_i[j] \) is therefore found by looking up the terminal node designated \( i_j \). The path label of its parent extended by a single nucleotide is the desired shustring. For example, in Figure 1A the substring ACC is a shustring with respect to the total dataset, while ACC is not.

\section*{3 ALGORITHM}

\subsection*{3.1 Description}

We start with a generalized suffix tree, \( T \), of \( S_1,S_2,\ldots,S_n \). From this we wish to compute for every pair of sequences, \( (S_i,S_j) \), the number of substitutions/site, \( K_r^{ij} \), as a function of the corresponding average
Efficient estimation of pairwise distances

Shustring length, $t_j/S_i$. For this purpose we assume that every branch node, $x$, of $T$ has the following six fields:

1. seqIds[x]: the set of sequence identifiers referred to by the terminal nodes in the subtree rooted on $x$;
2. termSeqIds[x]: the set of sequence identifiers referred to by terminal children of $x$;
3. branchChildren[x]: the set of children of $x$ that are branch nodes;
4. countTermSubtree[i, x]: the number of terminal nodes that refer to $S_i$ in the subtree rooted on $x$;
5. countTermChildren[x]: the number of terminal children of $x$ that refer to $S_i$;
6. stringDepth[x]: the length of the path label of $x$.

At every branch node $x$, we examine the child nodes. Among these, we first scan and count the terminal nodes that may be present (black nodes in Fig. 1A). Given such a terminal node $y$, the sequence $S_j$ it refers to serves as query when compared with all other distinct sequences $S_i$ in the subtree rooted on $x$. Hence, we add the string depth of $x$ plus one to the appropriate $t_j$.

Figure 1 demonstrates this procedure as part of the application of Algorithm 1 to our example dataset. The first node encountered by the algorithm is $x_1$. Its path label ACC can be extended in two ways to form the shustrings ACCCC in the comparison $(S_1, S_2)$ and ACCCA in the reverse comparison $(S_2, S_1)$. We therefore add 4 to $t_{k_2}$ and $t_{k_4}$ (Fig. 1B). Function scanTerm in Algorithm 1 automates this procedure.

The next node encountered in our tree traversal is $x_2$. Application of scanTerm leads to the addition of 3 to $t_{k_3}$ and $t_{k_2}$. Apart from the terminal child, we also need to examine the branch child of $x_2$, $x_1$. The extended path label of $x_2$, ACC, is a shustring in the comparisons $(S_1, S_2)$ and $(S_2, S_1)$. Hence, we add 3 to $t_{k_3}$ and $t_{k_4}$ (Fig. 1B). This procedure is automated in function scanBran of Algorithm 1.

Notice that both scanBran and scanTerm focus on terminal nodes. The crucial difference between the functions is that scanTerm(x, $\ell$) looks up the terminal nodes directly attached to $x$, whereas scanBran(x, $\ell$) looks up the terminal nodes in the entire subtree rooted on the branch children of $x$.

Upon completion of function traverse, the $n \times n$ matrix $\ell$ contains the sum of shustring lengths for all pairs of sequences (Fig. 1B). The final step in Algorithm 1 is the conversion of this matrix to an $n \times n$ matrix of $K_i$ values (Haubold et al., 2009).

### 3.2 Run time analysis

The computation of pairwise $K_i$ values is divided into three distinct phases: suffix tree construction, suffix tree traversal, and $K_i$ computation. Assume for the purpose of this analysis that suffix tree construction is linear in the length of the input string and that each of the $n$ input sequences is of length $L$ at long. In $\ell$ version 1, a suffix tree was constructed for each pair of sequences separately leading to a construction time $O(n^2L)$. In version 2, a single suffix tree is constructed for all sequences, which takes time $O(nL)$.

In contrast with the difference in construction time, the two approaches have the same run time bound for tree traversal: for each pair of input sequences, $L$ shustring lengths need to be summed. Hence, the traversal phase of $K_i$ always runs $O(n^2L)$.

**Algorithm 1** Estimate substitution rate

Require: $T$ [suffix tree of DNA sequences $S_1, S_2, ..., S_n$]
Require: $L$ [$L_i = |S_i|$]
Ensure: $K_i$ [$n \times n$ matrix of substitution rates]

1. for all $1 \leq j \leq n$ do
2. $t_j \leftarrow 0$ [initialize pairwise sums of shustring lengths, $\ell$]
3. traverse(root($T_i$, $\ell$))
4. for all $2 \leq i \leq n$ do
5. for all $1 \leq j < i$ do
6. $K^j_i \leftarrow K^j_i + \min(\ell_{j,i}/L_i, \ell_{j,i}/L_j)$
7. function traverse($x$, $\ell$)
8. for all $y \in$ branchChildren[x] do
9. traverse($y$, $\ell$)
10. countTermChildren[i, $x$]
11. scanBran($x$, $\ell$)
12. end function
13. function scanTerm($x$, $\ell$)
14. for all $i$ $\in$ termSeqIds[x] do
15. for all $j$ $\in$ seqIds[x]\[\ell i\] do
16. $\ell_{ij} \leftarrow \ell_{ij} + (\text{stringDepth}[x] + 1) \times \text{countTermChildren}[i, x]$
17. end function
18. function scanBran($x$, $\ell$)
19. for all $y$ $\in$ branchChildren[x] do
20. for all $i$ $\in$ seqIds[y]\[\ell j\] do
21. for all $j$ $\in$ seqIds[y] do
22. $\ell_{ij} \leftarrow \ell_{ij} + (\text{stringDepth}[x] + 1) \times \text{countTermSubtree}[i, y]$
23. end function

Finally, the computation of $K_i$ also runs in $O(n^2L)$ (Haubold et al., 2009). However, in version 2, we have introduced a number of computational shortcuts that improve the execution time of this part of the code. In summary, both versions of $K_i$ run $O(n^2L)$ but with different constant factors.

### 3.3 Auxiliary software

Sequence samples were simulated using the coalescent simulation program ms (Hudson, 2002) in conjunction with madd, which is freely available from the ms web site. Jukes–Cantor distances were estimated from the delta files generated by DnaSP using our program parseDelta, which is also freely available from the ms web site. Distances were clustered using the neighbor joining algorithm (Saitou and Nei, 1987) as implemented in the program neighbor, which is part of the PHYLIP package (Felsenstein, 1989). The resulting phylogenetic trees were drawn using MEGA (Kumar et al., 2008) or drawgram (PHYLIP package). Differences in tree shape were quantified using distedist (PHYLIP package). Correlations between distance matrices were computed using the program rcorr (Bonnet and Van de Peer, 2002).

### 4 IMPLEMENTATION

We have implemented Algorithm 1 in our program $K_i$ version 2, which replaces its predecessor (Haubold et al., 2009). However, apart from Algorithm 1 and the concomitant decrease in run
time, the two versions are similar and produce identical results: both implement the underlying suffix tree as an enhanced suffix array (Abouelhoda et al., 2002) using the suffix array library by Manzini and Ferragina (2002). The resulting program is written in standard C and designed to run under the UNIX command line. It can be tested on the k; r web site via a simple web interface that returns a phylogeny based on pairwise Kr values. Its sources and documentation are also freely available from the same web site.

5 APPLICATION

In the following applications of k; r, we investigate whether Kr is a consistent statistic, quantify its accuracy in the face of horizontal gene transfer and gene duplication, compare the speed of versions 1 and 2 and use the program to compute evolutionary distances from genomes.

5.1 Consistency of Kr

Haubold et al. (2009) already used simulations to demonstrate the accuracy of Kr for substitution rates below ~0.5. We complemented this result by investigating the behavior of Kr as the size of the dataset grows. Of the two parameters that determine dataset size, n and L, only the latter is relevant here, since Kr is a pairwise measure. We simulated samples of 10 homologous sequences of different lengths affected by a variable number of single nucleotide polymorphisms (SNPs) per site, s. For a given sample, we computed all pairwise Kr values and correlated these with the true substitution rates. Figure 2A shows the correlation between true and estimated distance matrices as a function of sequence length. As expected for a consistent statistic, the graph approaches perfect correlation for long sequences. Notice also that a sample of length 10 kb with s = 0.01 segregating sites per site contains 100 SNPs, and so does a sample of 1 kb sequences with s = 0.1. These two configurations also have a very similar average correlation (Fig. 2A). In fact, when we replotti
t the average correlation as a function of the total number of SNPs in Figure 2B, we see more clearly that it is the raw number of SNPs that determines the accuracy of Kr.

5.2 Horizontal gene transfer and gene duplication

Real genomes are not only affected by the single nucleotide substitutions modeled so far, but also by other forces of evolution, including horizontal gene transfer and gene duplication.

We modeled horizontal gene transfer by simulating samples of 10 sequences 100 kb long with s = 0.1. Each member of the sample received 1–10 chunks of 100 bp or 1 kb from random members of the sample that are distinct from the recipient. Figure 3 shows that the transfer of 100 bp chunks hardly affects the accuracy of distance estimation. In contrast, transfer of 1 kb chunks, i.e. 1% of the genome, significantly degrades distance estimation, while still leaving a strong average correlation between true and estimated distances of at least 0.91.

Gene duplication was modeled by carrying out the transfers just described within rather than between members of the sample. Each sequence was affected by 1–10 duplications spanning 1 kb, which spanned between 1 and 5 copies. While increases in duplication or copy number lead to less accurate distances, the effect is much weaker than that of horizontal transfer of equally long segments (Supplementary Fig. S1).

![Fig. 2. K_r is a consistent statistic. (A) Average correlation ± SD between distance matrices based on alignments and K_r, <r>, as a function of sequence length. (B) Data of (A) replotted as a function of the total number of SNPs contained in the sample. Horizontal line: maximal correlation. Each data point is computed from 1000 simulated samples of 10 sequences.](https://academic.oup.com/bioinformatics/article-abstract/25/24/3221/236022)

5.3 Run time

Based on the run time analysis in Section 3.2, we expected two things when timing k; r. First, both versions of the program should execute in time O(n^2 L). Second, version 2 should be faster than version 1.

Figure 4 shows the run times of the two programs as a function of the number of 1 Mb input sequences. The regression line for version 1 is R_1 = 1.6262 x n^2.2126 and for version 2 R_2 = 1.9924 x n^2.3598. Notice that version 1 exceeded the expected upper bound of O(n^2 L) by executing in time O(n^2.13 L). We think this is due to inefficiencies in our particular implementation. However, it was reassuring to see that version 2 stayed well below the theoretical upper bound (1.35 < 2). The relative speed of versions 1 and 2 is an increasing function of n, which for 1 Mb sequences is R_1/R_2 = 0.82 x n^0.86. For 100 sequences this amounts to a 43-fold difference.

5.4 Evolutionary distances between genomes

The following distance computations demonstrate the versatility of k; r by clustering samples of viral, bacterial and metazoan genomes.
Efficient estimation of pairwise distances

Fig. 3. Horizontal gene transfer may reduce the accuracy of $K_r$. See text for details.

Fig. 4. The run time of the previous version of $kr$ (version 1) and the current version (version 2) as a function of the number of 1 Mb input sequences.

To put our results into the context of existing genome analysis tools, we compared them either with the phylogeny returned by MAVID (Bray and Pachter, 2004) or to distances computed from pairwise MUMmer (Kurtz et al., 2004) alignments, depending on which of the two tools gave the best results.

We started by reanalyzing the genomes of the 12 completely sequenced Drosophila species, which are on average 169 Mb long (Drosophila 12 Genomes Consortium, 2007). Version 2 of $kr$ gave the same tree with the generally accepted topology that has already been published using version 1 of $kr$ (Haubold et al., 2009). However, version 2 finished in 3.25 h, which was 16 times faster than the 2 days and 6 h which it took for version 1 to analyze the same dataset on the same computer.

This was an important stress test for version 2 as its simultaneous analysis of all input sequences consumes more memory than the pairwise approach. Accordingly, peak memory consumption in the Drosophila analysis was 72 GB for version 2 compared with 13 GB for version 1. The increase in memory requirement from 13 to 72 GB agrees with the expectation that memory usage is linear in the number of nucleotides indexed in the underlying suffix tree.

MUMmer took 6.3 times longer than $kr$ to analyze the Drosophila dozen and yielded a topology whose Symmetric Distance (Robinson and Foulds, 1981) to the accepted tree was 8.

Next, we investigated the genomes of 825 pure HIV strains with an average genome size of 9 kb (Wu et al., 2007). The new $kr$ took 10 min 25 s to compute the corresponding distance matrix compared with almost 20 times as long for version 1 (3 h 26 min 38 s). In the dendrogram based on this matrix, all sequences were assigned to their correct clade except for strain A_DQ083238, which clustered with the C strains instead of its official classification as an A strain (Fig. 5A). Closer analysis revealed that A_DQ083238 is a recombinant strain containing 30% C-derived genetic material.

MAVID analyzed the HIV strains almost as rapidly as $kr$ (12 min 10 s) and in addition clustered all strains correctly (Fig. 5B).

Our final dataset comprised the complete genomes of 13 strains of Escherichia coli and Shigella. Figure 6A shows their recently published phylogeny based on a set of 169 single copy genes found in all strains (van Passel et al., 2008). The genome sizes of these bacteria range from 4.3 Mb to 5.5 Mb. Such large differences may lead to a distortion of the average shustring length, which can be mitigated by excluding the lengths of 95% of the shustrings observed by chance alone (Haubold et al., 2009). Our corresponding analysis of the enterobacterial genomes took 5 min 2 s, which was over 10 times faster than analysis with $kr$ version 1 (59 min). In the corresponding tree the two starred taxa in Figure 6B switched positions when compared with the reference tree. In contrast, MUMmer-based distance computations returned the correct tree (Fig. 6C), but took 33 min, i.e. over 6 times as long as $kr$. 
Alignment-free distance estimation is frequently used in multiple alignment; however, guide trees are usually devised for small sets of long sequences and MUMmer is designed for small sets of long sequences and 
varying amounts of synteny.

In our quest to make *K* more efficient when applied to datasets with many sequences we used the time-honored strategy of trading space for time. Where the previous version used memory proportional to the combined lengths of the two longest sequences in the sample, version 2 requires space proportional to the sum of the lengths of all sequences.

Apart from improved memory requirements, the extraction of all shustring lengths from a single suffix tree is also algorithmically more demanding than the corresponding traversal of pairwise suffix trees. However, our implementation of Algorithm 1 shows that the savings in run time are substantial, particularly for datasets with many sequences: version 2 runs 20 times faster on the 825 HIV genomes than version 1. Interestingly, analysis of the 12 *Drosophila* genomes is sped up 16-fold, while that of the 13 enterobacterial genomes, which are roughly 30 times shorter than the *Drosophila* genomes, is sped up only 10-fold. In other words, for a given sample size, savings in run time grow with the length of the input sequences. We think that this is due to the multiplication rule in line 22 of Algorithm 1. Instead of carrying out the addition to *u* for every countTermSubtree(*i,j*) times, all additions are executed in a single step. An example of this occurs when processing node *S* of our toy suffix tree in Figure 1. For real data, the proportion of shustring lengths that are added via the multiplication rule appears to grow with the lengths of the input sequences.

Apart from a comparison between the two versions of *K*, we also compare version 2 to two highly efficient established genome alignment tools, *MAVID* (Kurtz et al., 2004) and *MUMmer* (Bray and Pachter, 2004). The first thing to note about these comparisons is that *K* can rapidly analyze two types of data: large sets of short syntenic genomes (HIV) and small sets of long genomes affected by horizontal gene transfer, recent duplication and large-scale genome rearrangement (enterobacteria and *Drosophila*). In contrast, *MAVID* is designed for small sets of long sequences and *MUMmer* for syntenic sequences. On the other hand, the alignment-based methods yield better distance trees for the HIV and the enterobacterial datasets. Perhaps surprisingly, for the *Drosophila* dataset *K* gives the most accurate distances.

One reason for this is that the method of string comparison employed by *K* is as local as it gets: shustring lengths are added up at every position in a query sequence and their matches can be located anywhere in the subject sequence. This explains why *K* is applicable to genomes that have undergone rearrangements, like the enterobacteria, and to unordered contigs, as was the case for some of the *Drosophila* genomes. The strong performance of *K* on rearranged input sequences has previously been observed for other methods (Höhl et al., 2006; Sims et al., 2009).

The misclassified HIV strain (Fig. 5A) indicates the future direction of this project: instead of the global distances computed here, the shustring approach should lend itself to local homology detection at the resolution of single nucleotides. This would allow the efficient typing of recombinant DNA sequences.

---

**Fig. 6.** Phylogenies of 13 strains of *E. coli* and *Shigella*. (A) Reference phylogeny (van Passel et al., 2008) based on the multiple sequence alignment computed from 169 concatenated gene sequences using MAFFT (Katoh et al., 2005). (B) Whole-genome phylogeny based on *K*; the two starred taxa are switched with respect to (A). (C) Whole-genome phylogeny based on pairwise sequence alignment computed using MUMmer (Kurtz et al., 2004).

**6 DISCUSSION**

Alignment-free distance estimation is frequently used in multiple sequence alignment programs for the computation of guide trees (Haubold and Wiehe, 2006b). However, guide trees are usually poor phylogenies because they are not computed from evolutionary distances. To overcome this limitation, Haubold et al. (2009) have devised an efficient alignment-free estimator of the substitution rate between closely related genomes, *K*.

Here, we show that *K* is consistent when applied to ideal data and that, as long as the rate of substitution per site is <0.5, the accuracy of *K* is purely a function of the total number of SNPs in the dataset (Fig. 2). The number of SNPs in turn is affected both by the sequence length and the substitution rate.

While more SNPs make *K* more accurate, horizontal gene transfer can markedly reduce its accuracy (Fig. 3). This simulation result fits with the observation that, in HIV 30% of *C* strain material in an otherwise *A* background are sufficient to cause misclassification of the affected strain (Fig. 5).

In our quest to make *K* more efficient when applied to datasets with many sequences we used the time-honored strategy of trading space for time. Where the previous version used memory proportional to the combined lengths of the two longest sequences in the sample, version 2 requires space proportional to the sum of the lengths of all sequences.

Apart from improved memory requirements, the extraction of all shustring lengths from a single suffix tree is also algorithmically more demanding than the corresponding traversal of pairwise suffix trees. However, our implementation of Algorithm 1 shows that the savings in run time are substantial, particularly for datasets with many sequences: version 2 runs 20 times faster on the 825 HIV genomes than version 1. Interestingly, analysis of the 12 *Drosophila* genomes is sped up 16-fold, while that of the 13 enterobacterial genomes, which are roughly 30 times shorter than the *Drosophila* genomes, is sped up only 10-fold. In other words, for a given sample size, savings in run time grow with the length of the input sequences. We think that this is due to the multiplication rule in line 22 of Algorithm 1. Instead of carrying out the addition to *u* for every countTermSubtree(*i,j*) times, all additions are executed in a single step. An example of this occurs when processing node *S* of our toy suffix tree in Figure 1. For real data, the proportion of shustring lengths that are added via the multiplication rule appears to grow with the lengths of the input sequences.

Apart from a comparison between the two versions of *K*, we also compare version 2 to two highly efficient established genome alignment tools, *MAVID* (Kurtz et al., 2004) and *MUMmer* (Bray and Pachter, 2004). The first thing to note about these comparisons is that *K* can rapidly analyze two types of data: large sets of short syntenic genomes (HIV) and small sets of long genomes affected by horizontal gene transfer, recent duplication and large-scale genome rearrangement (enterobacteria and *Drosophila*). In contrast, *MAVID* is designed for small sets of long sequences and *MUMmer* for syntenic sequences. On the other hand, the alignment-based methods yield better distance trees for the HIV and the enterobacterial datasets. Perhaps surprisingly, for the *Drosophila* dataset *K* gives the most accurate distances.

One reason for this is that the method of string comparison employed by *K* is as local as it gets: shustring lengths are added up at every position in a query sequence and their matches can be located anywhere in the subject sequence. This explains why *K* is applicable to genomes that have undergone rearrangements, like the enterobacteria, and to unordered contigs, as was the case for some of the *Drosophila* genomes. The strong performance of *K* on rearranged input sequences has previously been observed for other methods (Höhl et al., 2006; Sims et al., 2009).

The misclassified HIV strain (Fig. 5A) indicates the future direction of this project: instead of the global distances computed here, the shustring approach should lend itself to local homology detection at the resolution of single nucleotides. This would allow the efficient typing of recombinant DNA sequences.
ACKNOWLEDGEMENTS

We thank Angelika Börsch-Haubold, Peter Pfaefflinhuber and Tomislav Domazet-Lošo for helpful comments on this article.

Conflict of Interest: none declared.

REFERENCES


