EMAGEN: An Efficient Approach to Multiple Whole Genome Alignment *

Jitender S. Deogun **Jingyi Yang** **Fangrui Ma**

Department of Computer Science and Engineering
University of Nebraska – Lincoln,
Lincoln, NE 68588-0115, USA,
Email: deogun{jyang, fma}@cse.unl.edu

Abstract

Following advances in biotechnology, many new whole genome sequences are becoming available every year. A lot of useful information can be derived from the alignment and comparison of different genomes. However, most of the current research focuses on pairwise genome alignment, and only a few available applications can efficiently align multiple genomes. In this paper, we present an efficient approach to align closely related multiple whole genomes, combining suffix arrays, graph theoretic formulation and existing tools for gap (short sequence) alignment. Our approach first finds a maximum set of aligned conserved regions among multiple whole genomes, then aligns the gaps between consecutive conserved regions with Clustal W. We present two methods to find the maximum set of aligned conserved regions among whole genomes. In first method, called Direct Matching (DM), multiple whole genomes are aligned with their DNA sequences. However, because most parts of prokaryotic genomes are encoded regions, we introduce second method, Functional Matching (FM), to especially align multiple prokaryotic genomes with their concatenated protein sequences. We present experimental results for both methods and give the analysis of the results. The FM method generates much better results for less closely related prokaryotic genomes than DM method. It outputs more and longer conserved regions, which conveys more accurate and detailed information about the conservation and inheritance of genomes, and generates more detailed alignments.

Keywords: Multiple Whole Genome Alignment, Conserved Regions, Suffix Arrays, Cocomparability Graphs, Prokaryotic Genomes.

1 Introduction

With advances in biotechnology, more and more whole genome sequences begin to be available. At present researchers are actively working on sequencing 284 prokaryotic genomes and 195 eukaryotic genomes. The availability of whole genome sequences is opening great possibilities for understanding the evolutionary process.

Whole genome alignment is relatively new, however, several methods have become available in recent years, e.g. PIPMaker (Schwartz, Zhang, Frazer, et al 2000), and SSAHA (Ning et al 2001). Researchers have followed different approaches for whole genome alignment, for example, dynamic programming in DIALIGN (Morgenstern, Frech, Dress & Werner 1998), hashing in ASSIRC (Vincens et al 1998), and suffix trees in MUMmer (Delcher et al 1999) are employed.

Whole genome alignment between pairs of strains from a single species often reveal evidence of extensive regions that abruptly interrupt synteny. The inheritance patterns and diversity within these regions holds significant information regarding the nature of small and large-scale evolutionary events that shaped the genomes. One successful approach for pairwise alignment of the entire genome sequences is MUMmer. The MUMmer software system combines suffix tree, longest increasing subsequence (LIS), and Smith-Waterman algorithms to align a pair of whole genomes. Recently, Brudno *et al* (Brudno & Morgenstern 2002) proposed another pairwise whole genome alignment system LAGAN.

Compared with pairwise whole genome alignment, multiple whole genome alignment is much more complicated, and has higher time and space complexity. Many existing alignment algorithms are only suitable for aligning short sequences, but not sequences at genome level. One approach for multiple sequence alignment is Hidden Markov Models (HMMs) (Eddy 1995). It is a statistical method that needs a large set of training data to construct species specific models of genomes. Thus, this method is not practical for multiple whole genome sequence alignment. Widely used FASTA (Lipman & Pearson 1988) and BLAST (Altschul et al 1990) tools do not give satisfactory results even for two whole genomes. An alternative approach to align multiple genomes is to select one genome, and align all others against it. However, this approach is difficult to work with, inefficient and unpredictable. Two existing applications for multiple whole genome alignment are MLAGAN (Brudno et al 2003) and MGA (Hohl, Kurtz & Ohlebusch 2002). MLAGAN is an extension of LAGAN for multiple genome alignment. MLAGAN is based on iterative pairwise alignment approach. It compares the synteny regions from different species, however, it is not designed to efficiently produce detailed alignment of closely related multiple whole genomes. MGA uses an anchor-based alignment approach, extending the basic idea used in MUMmer to multiple genomes. It first finds a set of short subsequences, which are assumed to be part of a global alignment, as anchors, then closes the gaps between anchors. It can be considered as an approach of divide and con-

*This research was supported in part by NSF EPSCOR Grant No. EPS-0091900 and NSF Digital Government Grant No. EIA-0091530.

quer.

Aligning multiple genomes can reveal significant information for bioscientific discovery. Such information can provide a better view of the genetic inheritance and polymorphic relationship among several organisms. However, available solutions for this problem are far from mature.

In this paper, we develop a new software system, called EMAGEN (Efficient Multiple Alignment algorithm for whole GENomes), for multiple whole genome alignment. It is an anchor-based alignment system. Development of EMAGEN is based on a general approach that combines suffix arrays and graph theoretic formulation for multiple whole genome alignment. EMAGEN first finds conserved regions among multiple genomes in linear time. Then it obtains a maximum set of conserved regions by a new graph theoretic approach. This maximum set of conserved regions is used as anchors. Furthermore, short subsequences between the anchors can be aligned with existing multiple sequence alignment tool Clustal W (Thompson, Higgins & Gibson 1994).

We develop two methods, Direct Matching (DM) and Functional Matching (FM) to find the maximum set of conserved regions. FM is particularly suitable for aligning prokaryotic genomes. The FM method generates much better results for less closely related prokaryotic genomes than DM method, which conveys more accurate and detailed information about the conservation of prokaryotic genomes.

2 Preliminaries

In this section, we review and introduce concepts related to suffix arrays, Maximal Unique Match, and graph theory.

2.1 Suffix Arrays and MUMs

For a string S, its suffix array sa is a sorted array of all the suffixes of the string (in lexicographical order) (Manber & Myers 1993). Generally, the lengths of most strings to be used with suffix arrays are smaller than 2^{32}, therefore, the positions of suffixes can be represented in 4 bytes. In its basic form, a suffix array uses $5n$ bytes for a string of length n, with n bytes to store the original string and $4n$ bytes to store the sorted positions of suffixes. Compared to its variation suffix trees (Gusfield 1999), a suffix array is much more space-efficient but has competitive performance.

Text:

S = acgtcac$

Suffix Array:

	sa		lcp	ps
0	0	acgtcac$	0	
1	5	ac$	2	c
2	4	cac$	0	t
3	1	cgtcac$	1	a
4	6	c$	1	a
5	2	gtcac$	0	c
6	3	tcac$	0	g
7	7	$	0	c

Figure 1: An example of a suffix array. '$' denotes the end of the string.

Let S be a string of length n, i.e. $S = S[0..n-1]$. S_i denotes the ith suffix of S, which is $S[i..n-1]$. $sa = sa[0..n-1]$ denotes the suffix array of S.

Definition 1. For a suffix array sa, the *lcp (longest common prefix)* value of an entry i, denoted by $lcp[i]$, is defined as follows: $lcp[0]$ equals 0; $lcp[i]$ equals the length of the longest common prefix of $sa[i]$ and $sa[i-1]$ when $i > 0$.

Definition 2. For a suffix array sa, the *proceeding symbol (ps)* of a suffix $S_{sa[i]}$ is denoted by $ps[i]$. So $ps[i] = S[sa[i]-1]$. $ps[i]$ is undefined if $sa[i] = 0$.

lcp[0..n-1] and *ps[0..n-1]* of a suffix array $sa[0..n-1]$ can be obtained from the suffix array in linear time (Kasai et al 2001).

Definition 3. For a suffix array sa, *Interval[i..j]* is called an *l-Interval* $[i, j]$ if both $lcp[i]$ and $lcp[j+1]$ is smaller than l, and the smallest lcp value for entry $i+1..j$ is l. The *length* of the *l-Interval* is $(j-i+1)$. The *lcp-string* of an *l-Interval*$[i, j]$ is the string $S[sa[i]..sa[i]+l-1]$.

There are several types of construction algorithms for suffix arrays. The suffix array algorithm, proposed in (Manber et al 1993), requires $O(nlogn)$ time with $4n$ bytes of extra working space. Burkhardt reduced the extra working space requirement to sublinear while maintaining the time complexity (Burkhardt & Kärkkäinen 2003). This algorithm builds suffix arrays with desirable practical performance. Some linear time construction algorithms also exist, but many of them achieve this linear time complexity by first building a suffix tree, then obtaining the suffix array from the suffix tree. In practice, these algorithms are not very efficient. The extra space requirement is also very high. Several direct linear time construction algorithms are also available. Kärkkäinen, recently, proposed a very simple linear time direct construction algorithm (Kärkkäinen & Sanders 2003).

The suffix array provides an efficient data structure to search for a query in a very long text, to find repeats in a string, and matches among multiple sequences. It has numerous applications in text compression (Burrows & Wheeler 1994), information retrieval (Sadakane 2000), and bioinformatics (Abouelhoda, Kurtz & Ohlebusch 2002).

Combined with the longest common prefixes (lcp) information, we can simulate the bottom-up traversal of a suffix tree in a suffix array. All the algorithms based on the bottom-up traversal of the suffix trees can also be implemented on suffix arrays coupled with lcp information with much less space requirement (Abouelhoda et al 2002).

Definition 4. A *Maximal Unique Match (MUM)* is a subsequence that occurs only once in each sequence of a set of multiple sequences and is not contained in a longer such sequence. The symbols bounding a MUM in all the sequences must not be the same (Delcher et al 1999).

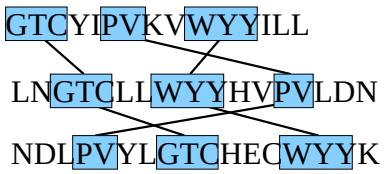

Figure 2: There are 3 MUMs among the above 3 strings: GTC, PV and WYY.

A suffix array can be used to find MUMs among a set of sequences, which will be described in later sections.

2.2 Trapezoidal Graphs

By $G = (V, E)$, we denote a finite, undirected and simple graph. For standard graph theory notations not given here we refer to (Golumbic 1980).

Definition 5 ((Deogun, Kloks, Kratsch & Müller 1999)). A *k-level trapeziodal diagram* $\mathcal{D}(\mathrm{G})$ for a graph $G = (V, E)$ assigns to each vertex v of G a collection of intervals

$$\mathcal{T}(v) = \langle [l_v^i, r_v^i] : l_v^i, r_v^i \in \{1, 2, \ldots, 2n\},$$

$$l_v^i < r_v^i, i \in \{1, 2, \ldots, k\}\rangle$$

such that for each $i \in \{1, 2, 3, \ldots, k\}$ and any pair of different vertices $v, w \in V$ the intervals $[l_v^i, r_v^i]$ and $[l_w^i, r_w^i]$ have no endpoints in common. Furthermore, $\{v, w\} \in E$ iff either there is an $i \in \{1, 2, 3, \ldots, k\}$ such that $[l_v^i, r_v^i]$ and $[l_w^i, r_w^i]$ have nonempty intersection or there are $i, j \in \{1, 2, 3, \ldots, k\}$ such that $l_v^i < r_v^i < l_w^i < r_w^i$ and $l_v^j < r_v^j < l_w^j < r_w^j$.

To understand the above definition, we show the following visualization of a k-level trapezoidal diagrm. k parallel horizontal lines $L_1, L_2, \ldots, L_k$ are drown with m points distributed on each line. A vertex $v \in V$ is a polygon Q_v, a *k-trapezoid*, obtained by joining consecutive points in the chain $l_v^1, l_v^2, \ldots, l_v^d, r_v^d, r_v^{d-1}, \ldots, r_v^1, l_v^1$. $\{v, w\} \in E$ if and only if Q_v and Q_w have nonempty intersection. A 3-level trapezoidal diagram is shown in Figure 3.

Definition 6. A graph G is a *k-dimensional trapezoidal graph* if it has a *k-level trapezoidal diagram*.

Figure 3 gives an example of a 3-dimensional trapezoidal graph and the corresponding 3-level trapezoidal diagram. In an earlier paper (Deogun et al 1999), we establish relationship between k-trapezoidal and cocomparability graphs.

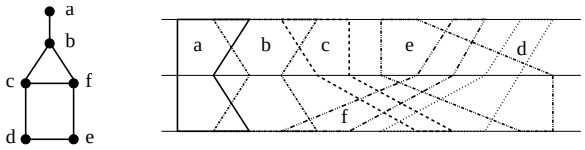

Figure 3: An example of a 3-level trapezoidal diagram and corresponding 3-dimensional trapezoidal graph.

Theorem 1 ((Deogun et al 1999)). *The k-trapezoidal graphs are exactly the cocomparability graphs of partially ordered sets of interval dimension at most k.*

Cocomparability graphs are complements of comparability graphs, and thus a maximum independent set of a k-trapezoidal graph can be found by finding a maximum clique in comparability graphs. The maximum clique problem is NP-complete in general. However, an $O(|V|^3)$ algorithm for finding maximum clique in comparability graphs can be found in (Golumbic 1980) and a linear time algorithm was presented recently in (McConnell & Spinrad 1999). The theorem below follows from (McConnell et al 1999).

Theorem 2 ((McConnell et al 1999)). *A maximum clique of a comparability graph can be found in $O(|V| + |E|)$, i.e. linear time.*

2.3 Longest Increasing Subsequence (LIS) of MUMs and MUM Graphs

As mentioned before, all the MUMs among a set of sequences can be found by suffix arrays. After all the MUMs of a set of sequences have been identified, each of the MUMs is labeled with an integer from $\{1, 2, 3, \ldots, m\}$, according to their positions in the first sequence, where m is the number of MUMs. This label is a unique identifier of a MUM. The MUMs may appear in different order in different sequences according to their positions in the corresponding sequences.

Definition 7. A *MUM sequence* is a permutation of MUM identifiers $\{1, 2, 3, \ldots, m\}$, which corresponds to the order of MUMs in a particular sequence.

Definition 8. A *Longest Increasing Subsequence (LIS)* of a set of *MUMs*, denoted by *LIS-MUMs*, is the largest subset of the MUMs which appear in ascending order in each of the MUM sequences. The MUMs in LIS-MUMs are not covered by each other.

To find LIS-MUMs, we introduce the concepts of *MUM diagrams and MUM graphs*, and show that the class of *MUM graphs* is same as the class of *trapezoidal graphs*. We represent a MUM sequence by nodes on a horizontal line, where each node represents a MUM in the sequence. Such a horizontal line is called a *MUM line*. The concepts of MUM diagrams and MUM graphs are defined as follows:

Definition 9. A *k-level MUM diagram* is a diagram depicting the structure of MUMs in different MUM sequences. It consists of k MUM lines. The MUMs with the same identifier on different MUM lines are joined together with line segments resulting in a *MUM chain*, see Figure 4. A MUM chain for MUM x is denoted by C_x.

Definition 10. A *k-dimensional MUM graph*, $G_{MUM} = (V_{MUM}, E_{MUM})$, is the intersection graph of a k-level MUM diagram, where the vertex set, V_{MUM}, is the set of MUM identifiers. Two vertices u and v are adjacent, i.e. $(u, v) \in E_{MUM}$, if and only if C_u and C_v intersect. A 3-level MUM diagram, along with corresponding MUM graph, G_{MUM}, is given in Figure 4.

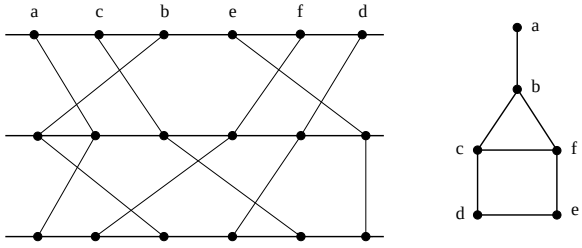

Figure 4: An example of 3-level MUM diagram and corresponding 3-dimensional MUM graph.

Lemma 3. *A k-dimemsional MUM graph can be constructed in $O(km^2)$ time from a k-level MUM diagram, where m is the number of MUMs.*

It is easy to see that the MUM diagram is, in fact, a k-level trapezoidal diagram. The proposition below follows immediately.

Proposition 4. *The class of k-dimensional MUM graphs is same as the class of k-dimensional trapezoidal graphs.*

It follows that a MUM graph, G_{MUM}, is a cocomparability graph. Therefore LIS-MUMs for a set of MUMs can be found by finding a maximum independent set in the MUM graph or equivalently by finding maximum clique in corresponding comparability graphs. The theorem below follows from the Theorems 1, 2 and Lemma 3.

Theorem 5. *The time complexity of finding LIS-MUMs for a set of k MUM sequences, each of length m, is $O(km^2), k \geq 3$.*

3 Algorithm

In our algorithm, we first find the conserved regions among a group of multiple genomes. Then we find a maximum set of these conserved regions. The maximum set of the conserved regions of a group of genomes is likely to belong to the global alignment, therefore, the conserved regions in it can be used as anchors in multiple whole genome alignment. Besides, gaps between anchors can be further aligned with existing multiple sequence alignment tool Clustal W to generate detailed alignment.

In our algorithm, we use Direct Matching (DM) and Functional Matching (FM) methods to find the anchors for global alignment for a group of genomes. In DM method, anchors are found from the entire nucleotide sequences of a group of genomes. FM method only focuses on the coding regions of the prokaryotic genomes. Therefore, FM is especially suitable for prokaryotic genomes. In the following, we describe details of both DM and FM methods.

3.1 Direct Matching (DM) method

The main problem in DM method is to find the LIS-MUMs for k sequences. We first find the MUMs, and use the MUM diagram to define a MUM graph, G_{MUM}. As described in the previous section, the problem of finding LIS-MUMs is solved by finding a maximum independent set of the MUM graph, G_{MUM}. The DM algorithm can be divided into 3 steps.

3.1.1 Step 1: Find MUMs of a group of Multiple Genomes

To find the MUMs for a group of genomes, we use the suffix array. Suppose we have k genome sequences $S_1, S_2, \ldots, S_k$. First, we develop a sequence $S = S_1\$_1S_2\$_2..\$S_k\$_k$ by concatenating all the genome nucleotide sequences, terminating different sequences with different separating symbols. Then a suffix array sa is built for S in $O(n)$.

Given k sequences, $S_1, S_2, \ldots, S_k$, the concatenated string $S = S_1\$_1S_2\$_2..\$S_k\$_k$ and suffix array sa for S, we define *sequence order (so)* value and *MUM-Interval* as follows:

Definition 11. The *sequence order (so)* value of an entry i in S, denoted by $so[i]$, is the order of the sequence within which the suffix $S_{sa[i]}$ begins. If $sa[i]$ begins from separating symbols, then $so[i]$ is undefined.

Definition 12. An *l-Interval*$[i, j]$ is called a *MUM-Interval* if the $length$ of *l-Interval*$[i, j]$ is k; $so[i], \ldots, so[j]$ are pairwise distinct; and $ps[i], \ldots, ps[j]$ are not the same value.

From the definition of the *MUM-Interval*, the *lcp-string* of a *MUM-Interval[i,j]* occurs exactly once in each of the k sequences. Besides this *lcp-string* can not be contained in a longer such sequence. Otherwise, either the $ps[i], \ldots, ps[j]$ will be same, or the length of the *l-Interval* will be larger than k. So the *lcp-string* of a *MUM-Interval[i,j]*, $S[sa[i]..sa[i] + l - 1]$, is a MUM.

We can scan the suffix array for S to find *MUM-Intervals* and output the MUMs. The simplest way is to check all *Intervals* which are of $length\ k$. We employ a different method, described below, to speed up the process of finding all the *MUM-Intervals*.

Definition 13. An *l-Interval*$[i, j]$ is called a *Maximum l-Interval* if all of $lcp[i+1], \ldots, lcp[j]$ have the same value l.

It is easy to find *Maximum l-Intervals* by scanning the suffix array. A *MUM-Interval* is a *Maximum l-Interval*, or it contains a *Maximum l-Interval*. So we first scan the suffix array to locate *Maximum l-Intervals*. For a *Maximum l-Interval* of length k, we check if it is a *MUM-Interval*. If the length of a *Maximum l-Interval* is smaller than k, we find if an *l-Interval* of length k exists which contains this *Maximum l-Interval*. If such an *l-Interval* exists, we check if it is a *MUM-Interval*. We can find all *MUM-Intervals* by scanning the suffix array once. So to find all MUMs from the suffix array costs linear time.

Example 6. Figure 5 shows a suffix array of three sequences S_1 = a b d b c e, S_2 = b c b e a d, and S_3 = a b c d e. As shown in the figure, *l-Interval*$[0, 2]$, *l-Interval*$[3, 5]$, *l-Interval*$[11, 13]$ and *l-Interval*$[14, 16]$ are *MUM-Intervals* which denote four MUMs: (a), (bc), (d), (e). These four MUMs are labeled with index set $\{1, 2, 3, 4\}$. In accordance with convention described in Section 2.3, these MUMs are presented as: 1 = (a), 2 = (d), 3 = (bc), and 4 = (e). *l-Interval*$[8, 10]$ is not a *MUM-Interval* because $ps[8], ps[9]$ and $ps[10]$ have the same value b.

Text:

S 1 = abdbce
S 2 = bcbead
S 3 = abcde
S = abdbce$bcbead#abcde!

Suffix Array:

	sa		lcp	ps	so
0	14	abcde!	0	#	3
1	0	abdb..	2		1
2	11	ad#a..	1	e	2
3	7	bcbe..	0	$	2
4	15	bcde!	2	a	3
5	3	bce$..	2	d	1
6	1	bdbc..	1	a	1
7	9	bead..	1	c	2
8	8	cbe..	0	b	2
9	16	cde..	1	b	3
10	4	ce$	1	b	1
11	2	dbc..	0	b	1
12	17	de!..	1	c	3
13	12	d#a..	1	a	2
14	10	ead..	0	b	2
15	5	e$b..	1	c	1
16	18	e!	1	d	3
17	6	$bc..	0	e	
18	13	#ab..	0	d	
19	19	!	0	e	

Figure 5: A suffix array for three sequences listed above with the *lcp, ps, so* information.

3.1.2 Step 2: Construct a k-level MUM Diagram and a k-dimensional MUM Graph

After the MUMs are found, k MUM sequences are developed, and a k-level MUM diagram is constructed. A k-dimensional MUM graph is defined accordingly. This step takes $O(km^2)$ time where k is the number of genomes and m is the number of MUMs found.

The MUM diagram and MUM graph, G_{MUM}, for the MUMs found in S_1, S_2 and S_3 are shown in Figure 6.

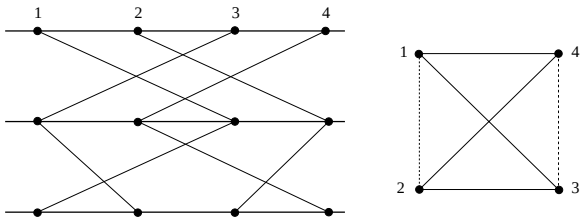

Figure 6: A 3-level MUM diagram and its MUM graph, G_{MUM}. The dashed lines denote the complement graph $\overline{G_{MUM}}$.

3.1.3 Step 3: Find Independent Set in the MUM Graph and Output the LIS-MUMs

As stated above, finding the LIS-MUMs is equivalent to finding a maximum independent set of a k-dimensional MUM graph which can be found by finding maximum clique in the corresponding comparablitiy graph.

In our example, the maximum clique in the complement graph $\overline{G_{MUM}}$ (see Figure 6) is $\{1, 2\}$, or $\{3, 4\}$, which is the independent set in G_{MUM}. So the LIS for sequences S_1, S_2 and S_3 is {a, d}, or {bc, e}. They are actually the conserved regions if we consider the sequences S_1, S_2 and S_3 as genomes.

This step requires $O(m + e)$ time, where m is the number of MUMs and e is the number of edges in $\overline{G_{MUM}}$.

3.2 Functional Matching (FM)

In prokaryotic genomes, a large part of the genome sequences are coding regions. It is known that coding regions are more conserved, and are more interesting, therefore, Functional Matching (FM) method is introduced to specifically align the coding regions among multiple prokaryotic genomes. In Functional Matching method, concatenated amino acid sequences are constructed to represent genomes instead of the original nucleotide sequences. Then maximum sets of conserved regions from these long amino acid sequences are found as anchors for alignment. These anchors are actually short amino acid subsequences, which are mapped back to the nucleotide sequences positions. Based on research objectives, we follow two different approaches in FM: *Simple Approach* and *Comprehensive Approach*.

In *Simple Approach*, the anchors are output as two independent sets while, in *Comprehensive Approach*, only one set of anchors are generated.

3.2.1 Simple Approach: *Pre-processing*

During the pre-processing, concatenated amino acid sequences are constructed to represent the prokaryotic genomes to be aligned. It is known that each prokaryotic genome has a variety of genes. Each gene is responsible for coding a protein that is presented by an amino acid sequence. A protein can be coded in two directions, forward or backward. For convenience, we call them forward or backward proteins.

Definition 14. A *forward Virtual Genome Sequence (VGS)* is obtained by concatenating all the forward protein sequences in a genome. A *backward VGS* is obtained by concatenating all the reversed backward protein sequences in a genome.

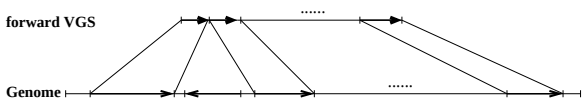

Figure 7: The concatenation of a forward VGS.

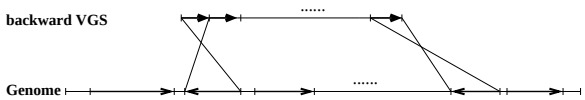

Figure 8: The concatenation of a backward VGS. The backward proteins are reversed before concatenation.

In the pre-processing step, a forward VGS and a backward VGS are constructed for each genome. The relationship between the proteins and genes, along with the positions of genes in the genome, is saved for the post-processing step.

3.2.2 Simple Approach: *Finding LIS-ASMs*

In Simple Approach, the forward and backward VGSs are grouped separately to form the *forward* and *backward* VGS sets. Either set of VGSs is aligned just like a set of nucleotide sequences.

Definition 15. An *amino acid MUM subsequence (AMS)* is a MUM found in a set of VGSs. Following the definition of MUM, an AMS has exactly one *occurrence* in each VGS.

Definition 16. An AMS is called a *forward* (or *backward*) AMS if it is found from the forward (or backward) VGS set. An AMS is called a *crossing* AMS if one or more of its occurrences are composed of amino acids from more than one protein. Otherwise, it is called a *simple* AMS.

Definition 17. *A Functional Segment Match (FSM)* is the nucleotide subsequence corresponding to an AMS.

Two sets of LIS of AMSs, denoted by LIS-AMSs, are computed from the forward and backward VGS sets by the DM algorithm described earlier. These LIS-AMSs are called forward and backward LIS-AMSs respectively.

3.2.3 Simple Approach: *Post-processing*

In the post-processing step, the forward and backward LIS-ASMs are mapped back to their coding nucleotide genome sequences.

It should be noted that a simple AMS can be mapped back directly, while a crossing AMS is split into several simple sub-AMSs, which are mapped back separately, based on composing protein sequences.

Figure 9 gives a simple illustration of the mapping back of a forward crossing ASM. In the figure, the *occurrence* of the ASM in VGS2 is composed of amino acids from two proteins. The crossing ASM is therefore split into two sub-ASMs and mapped back to two FSMs. A backward crossing ASM is mapped back similarly. The number of the outputted FSMs may be larger than the number of the LIS-ASMs as a result.

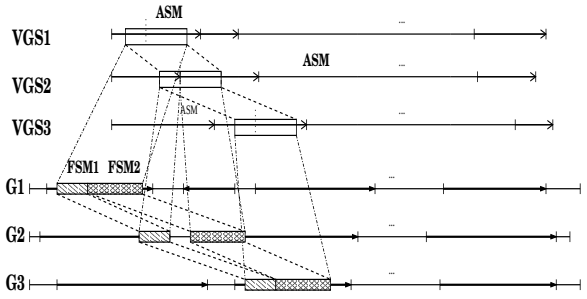

Figure 9: Mapping back of a forward crossing AMS.

Finally, two sets of LIS of FSMs, denoted by LIS-FSMs, are generated separately as the conserved coding regions, e.g. the forward and backward LIS-FSMs. These two sets of FSMs are equivalent to the LIS-MUMs output by EMAGEN and can serve as anchors for further alignments.

3.2.4 Comprehensive Approach for FM

In Simple Approach, because the forward LIS-ASMs and backward LIS-ASMs are found separately, the forward and backward LIS-FSMs can not be combined. Either set of LIS-FSMs represents conservation among the multiple genomes. To find the global conservation, a Comprehensive Approach of EMAGEN-FM can be chosen.

In the Comprehensive Approach, the pre-processing step remains the same as in Simple Approach. A *forward* VGS set and a *backward* VGS set are constructed for a group of genomes.

After the *forward* VGS set and the *backward* VGS set are formed, the forward and backward ASMs are obtained from them. Unlike the Simple Approach, the two sets of AMSs are mapped back immediately to generate two sets of FSMs. Those derived forward and backward sets of FSMs are combined to form a Comprehensive Set of FSMs, and the global LIS-FSMs are found from this set. Obviously, the post-processing step is not needed in the Comprehensive Approach.

3.3 Time Complexity of DM and FM Algorithm

As explained before, construction of a suffix array and computation of all MUMs of a group of k genomes requires linear, that is $O(n)$, time, where n is the total length of all input sequences. The next step is to transform a k-level MUM diagram into a k-dimensional trapezoidal graph. The time complexity of defining a k-dimensional trapezoidal graph, G_{MUM}, from a k-level MUM diagram is $O(m^2)$, where m is the number of MUMs and k, the number of sequences, can be treated as a constant. A maximum clique of the complement graph $\overline{G_{MUM}}$ can be found in $O(m+e)$, where e is the number of edges in $\overline{G_{MUM}}$ (McConnell et al 1999). These cover all the steps in the DM method. Now, $e = O(m^2)$, and therefore complexity of the DM is dominated by $O(m^2)$.

It is commonly accepted that $m \ll n$, the total length of all input genome sequences (Delcher et al 1999). It may be not unreasonable to assume that $m^2 \leq c * n$, where c is a constant and n is the total length of a group of whole genomes. Therefore, the overall time complexity of DM method can be deemed as linear, i.e. $O(n)$ for finding the maximum set of conserved regions for multiple whole genome alignment.

In order to determine the time complexity of FM method, we also need to analyze the pre-processing and post-processing steps. It can be easily seen that the pre-processing can be finished in $O(n)$ time, where n is the total length of the genomes. The post-processing for FM Simple Approach can be easily finished in $O(m^2)$ time, where m now is the number of AMSs found. From our previous analysis of the time complexity of DM the experimental results, it can also be assumed that $m^2 \leq c*n$, where c is a constant. In the Comprehensive Method, the mapping back process is moved to an earlier stage, which does not alter the total time complexity. Therefore, the overall time complexity of FM is not changed. It can still be deemed as linear, i.e. $O(n)$.

3.4 Closing Gaps between LIS-MUMs

As mentioned before, gaps between LIS-MUMs can be further aligned with Clustal W to get detailed alignments. A threshold is set as the maximum length of the gaps which should be aligned with Clustal W. The time to generate detailed alignments of the gaps depends on the threshold.

4 Experimental Results

Our software system, EMAGEN, can be used to study properties of conserved regions and inheritance of the microorganisms. It can also identify the regions that differentiate one strain from others. In practice, we need to set the threshold of the minimum length of MUMs. More than one thresholds in decreasing order can be set. For example, we can input two thresholds. The first threshold should be larger than the second one. MUMs above the first threshold will be processed first to find the LIS-MUMs. Other MUMs above the second threshold between the consecutive LIS-MUMs above the first threshold will be aligned recursively.

In this section, we display experimental results of EMAGEN and compare its performance to that of MGA. MGA is also an anchor-based multiple whole genome alignment application. It uses *virtual suffix tree* to find MultiMEMs, which are maximal matches among multiple sequences. Then MGA finds chained matches from MultiMEMs. The chained matches in MGA are equivalent to LIS-MUMs. The gaps between chained matches can be further aligned recursively, or with existing tools. All the experiments are made on a SunBlade 1000 workstation with $1G$ memory. A detailed study of extensive experimental results will be reported in a later paper. We test on both methods, Direct Matching and Functional Matching, to find the anchors. Short gaps between anchors, whose lengths are shorter below a user-specified threshold, can be further aligned with Clustal W. The

final detailed alignment depends on the gap threshold and the options the user choose, so we don't include the analysis of final detailed alignment in this section. Compared to MGA, EMAGEN shows several advantages. For highly conserved genomes, like three strains of *E. coli*, EMAGEN-DM is more efficient than MGA to find anchors, more than two times faster, but produces almost same or comparable results. For less closely related prokaryotic genomes, EMAGEN-FM generates many more anchors than MGA, and thus leads to more revelation of the conservation and the better detailed alignment.

4.1 Experimental Results of EMAGEN-DM

In this section, we describe experiments performed on different groups of genomes with EMAGEN-DM.

We choose different strains of *E. coli* and *S. pyogenes* as data set to show the performance of EMAGEN-DM and compare it with MGA. The name of the genomes are listed in the following table.

Group I	
1	*E. coli K12*
2	*E. coli O157H7*
3	*E. coli O157H7 EDL933*
Group II	
1	*S. pyogenes*
2	*S. pyogenes MGAS315*
3	*S. pyogenes MGAS8232*
4	*S. pyogenes SSI-1*

Table 1: Experiment data set.

The MUMs and LIS-MUMs for three strains of *E. coli* are shown in Figure 10. In EMAGEN-DM, we impose a threshold of 20 bps on the minimum length of a MUM. As we can observe from Figure 10(a), there are only a few inconsistent or scattered MUMs. It should be noted that the number of MUMs found varies with the threshold of MUMs. As the threshold increases, the total number of MUMs decreases. After the alignment, the LIS-MUMs are perfectly aligned on a smooth diagonal line as shown on in Figure 10(b). The only obvious break between bp positions of 1 and 2 millions indicates that the major difference among three strains is located in this region.

Experimental statistics for both EMAGEN-DM and MGA are displayed in Table 2 and Table 3. The running time of both EMAGEN-DM of MGA has two values. For EMAGEN-DM, the first value of running time is the time to build a suffix array for a group of genomes; the second value is the time to the find LIS-MUMs using the suffix array. For MGA, the first value of running time is the time to build virtual suffix tree for a group of genomes, the second values is the time to find the chain matches from the virtual suffix tree. Compared with MGA, EMAGEN-DM shows some favorable results. For three highly conserved strains of *E. coli*, EMAGEN-DM method generates almost the same results as MGA with noticeably shorter time. For four less highly conserved strains of *S. pyogenes*, EMAGEN-DM method also generates comparable results in much less time.

The graphs in Figure 11 show the alignments of the LIS-MUMs of other groups of three and four genomes. From the graphs of the LIS-MUMs, it is easy to watch the conservation of a group of genomes. As the number of

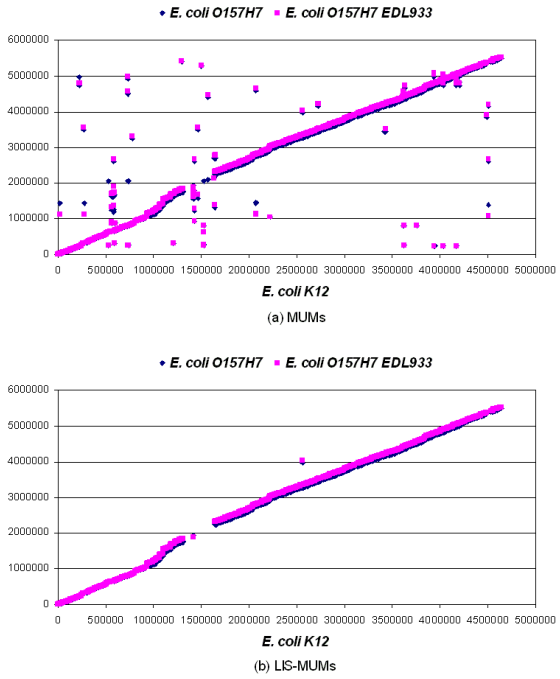

Figure 10: (a) shows the plot of MUMs and (b) that of LIS-MUMs. Each point in the graphs corresponds to a MUM. The x-axis is the positions in *E.coli K12*, y-axis is the positions in other genomes.

	EMAGEN-DM	MGA
Running time (second)	178.58 + 223.46	534.51 + 441.46
Number of LIS-MUMs (chained matches)	34612	34922
Total length of LIS-MUMs (matches)	3484053	3547621

Table 2: Experimental statistics for Group I: three strains of *E. coli*. We choose 1000:20 as threshold for both DM and MGA to generate anchors.

the genomes in a group increases, the conserved regions become shorter and fewer.

4.2 Experimental results of EMAGEN-FM

For less closely related prokaryotic genomes, we can use EMAGEN-FM to provide much better results. We use a group of three genomes, *L. monocytogenes*, *L. innocua* and *S. aureus*, as the experiment data to show the improvement of EMAGEN-FM. We set the threshold of the minimum length of ASMs as 7 which is equivalent to MUM length 21 because an amino acid is encoded by three nucleotides. The output results of the forward LIS-FSMs, backward LIS-FSMs by Simple Approach and global LIS-FSMs by Comprehensive Approach, along with the LIS-MUMs generated by the EMAGEN-DM with the same group of genomes, are displayed in the following. Their results are summarized in Table 4.

Figure 12 and 13 gives a flavor of differences between the EMAGEN-DM and EMAGEN-FM, where a

	EMAGEN-DM	MGA
Running time (second)	70.61 + 15.57	338.13 + 382.14
Number of LIS-MUMs (chained matches)	3781	5503
Total length of LIS-MUMs (matches)	425309	626112

Table 3: Experimental statistics for Group II: four strains of *S. pyogenes*. We choose 500:20 as threshold for both EMAGEN-DM and MGA to generate anchors.

	Number of MUMs/FSMs/matches
Forward LIS-FSMs	1089
Backward LIS-FSMs	1206
Global LIS-FSMs	2244
LIS-MUMs	88
chained matches by MGA	334

Table 4: Experimental statistics for three genomes: *L. monocytogenes*, *L. innocua* and *S. aureus*.

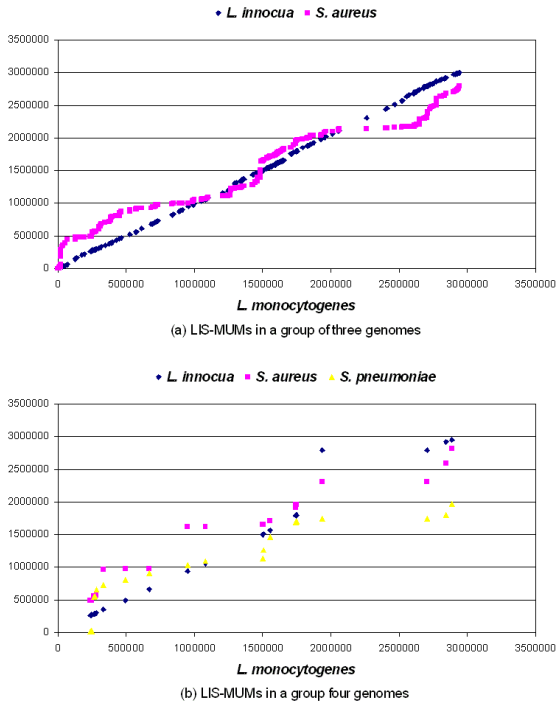

Figure 11: Graph (a) shows the plot of LIS-MUMs of the group of three genomes *L. monocytogenes*, *L. innocua* and *S. aureus*. (b) shows the LIS-MUMs of the group of four genomes *L. monocytogenes*, *L. innocua*, *S. aureus* and *S. pneumoniae*. The points in each graph represent LIS-MUMs. The x-axis represents the positions in genome *L. monocytogenes*, and the y-axis denotes the positions in other genomes of the group.

comparison of LIS-MUMs by EMAGEN-DM with different sets of LIS-FSMs by EMAGEN-FM is shown.

The number of LIS-FSMs found with EMAGEN-FM are much larger than the number of LIS-MUMs found with EMAGEN-DM and that of chained matches by MGA. Compared with the curve of LIS-MUMs, the curves for LIS-FSMs in the graphs are much more smooth. Besides, global LIS-FSMs show more similarities among the genomes than others. One question may arise that if those LIS-FSMs are real conserved regions or just random matches. We check the LIS-ASMs which are mapped back to generate these LIS-FSMs. All LIS-ASMs have exactly one occurrence in one VGS by definition. Those occurrence are in one or more proteins in the VGSs. Most LIS-ASMs have all the occurrences in the proteins in different VGSs with same or very similar properties. So these LIS-FSMs are likely true conserved regions. From the experimental results, it can be seen that EMAGEN-FM produces much larger maximum sets of conserved regions than EMAGEN-DM, which convey more detailed information about conservation and inheritance of prokaryotic genomes. This is because that different codons can encode the same amino acid, which has the effect of approximate matching. The different sets of LIS-FSMs are also used as anchors for multiple alignment to produce much more detailed global alignments. Since this method only finds anchors from the coding regions, it is more efficient, and the results may lead to more biological insight.

5 Conclusions

In this paper, we developed a new software system, called EMAGEN, for multiple whole genome alignment. Alignment of multiple whole genome sequences is an important problem, because it can provide significant information about inheritance and polymorphism of multiple whole genomes. We present an efficient anchor-based algorithm in EMAGEN to align multiple whole genome sequences. Development of EMAGEN is based on a general approach that combines suffix arrays, graph theoretic formulation, and short sequence alignment tool for multiple whole genome alignment. We present graph theoretic argumentation to prove that our algorithm has almost linear complexity to find the maximum set of conserved regions as anchors.

Two methods are developed to find anchors among multiple genomes. The first method, Direct Matching (EMAGEN-DM) is used for finding anchors of multiple whole genomes with their DNA sequences. The second method, Functional Matching (EMAGEN-FM) is used for finding anchors of multiple prokaryotic genomes. Gaps between anchors are further aligned by Clustal W to produce detailed alignments.

Experiments conducted to evaluate both methods and analysis of experimental results are also presented. Both EMAGEN-DM and EMAGEN-FM successfully find the maximum set of conserved regions as anchors among a group of multiple whole genomes. For the limited experiments, we conduct that EMAGEN-DM is more than two times faster than MGA to find anchors for highly conserved genomes. EMAGEN-FM generates much better results for less closely related prokaryotic genomes than EMAGEN-DM and MGA. These results shows more accurate and detailed information about the conservation of

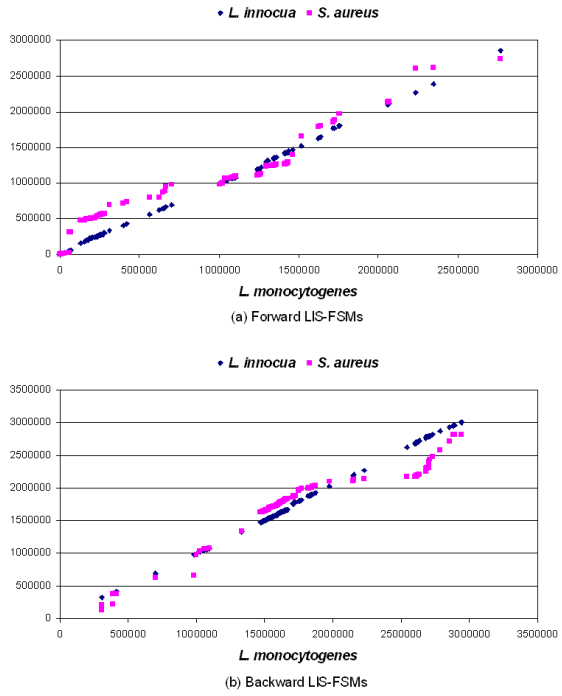

Figure 12: The alignment of the forward LIS-FSMs, backward LIS-FSMs of three genomes: *L. monocytogenes*, *L. innocua* and *S. aureus*.

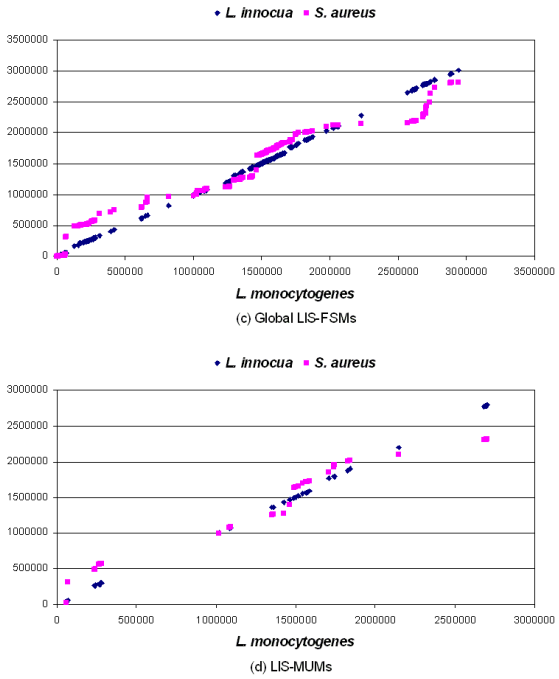

Figure 13: The alignment of the global LIS-FSMs, and LIS-MUMs of three genomes: *L. monocytogenes*, *L. innocua* and *S. aureus*.

prokaryotic genomes.

References

Abouelhoda, M.I., Kurtz, S. & Ohlebusch, E. (2002), The Enhanced Suffix Array and its Applications to Genome Analysis, *in* 'The 2nd Workshop on Algorithms in Bioinformatics', pp. 449-463.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990), 'Basic Local Alignment Search Tool', *Journal of Molecular Biology* **215**, 403–410.

Brudno, M. & Morgenstern, B. (2002), Fast and sensitive alignment of large genomic sequences, *in* 'IEEE Computer Society Bioinformatics Conference 2002', Stanford University, CA, USA, pp. 138–147.

Brudno, M., Do, C.B., Cooper, G.M., et al (2003), 'LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA', *Genome Res.* **13**(4), 721–731.

Burkhardt, S. & Kärkkäinen, J. (2003), Fast lightweight sux array construction and checking, *in* '14th Annual Symposium on Combinatorial Pattern Matching', Springer.

Burrows, M. & Wheeler, D.J. (1994), A block-sorting lossless data compression algorithm, *in* 'Technical Report 124', SRC (digital), Palo Alto.

Chuong, H. (2002), Genome Analysis and Genome Comparison, NIH/NLM/NCBI.

Delcher, A.L., Kasif, S., Fleishmann, R.D., Peterson, J., White, O. & Salzberg, S.L. (1999), 'Alignment of whole genomes', *Nucleic Acids Research* **27**, 2369–2376.

Deogun, J.S., Kloks, T., Kratsch, D. & Müller, H. (1999), 'On the Vertex Ranking Problem for d-trapezoid Graphs and for Circular-arc Graphs', *Discrete Appl. Math.* **98**, 39–63.

Eddy, S. (1995), Multiple alignment using hidden Markov models, *in* 'The third international conf. on Intelligent systems for molecular biology', AAAI Press, pp. 114–120.

Gusfield, D. (1999), *Algorithms on Strings, Trees, and Sequences–Computer Science and Computational Biology*, Cambridge University Press, British.

Golumbic M.C. (1980), *Algorithmic graph theory and perfect graphs*, Academic Press, New York.

Hohl, M., Kurtz, S. & Ohlebusch E. (2002), 'Efficient multiple genome alignment', *Bioinformatics* **18**(Suppl 1), S312-320.

Kärkkäinen, J. & Sanders, P. (2003), Simple Linear Work Suffix Array Construction, *in* '13th International Conference on Automata, Languages and Programming', Springer.

Kasai, T., Lee, G., Arimura, H., Arikawa, S. & Park, K. (2001), Linear-time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications, *in* 'the 12th Annual Symposium on Combinatorial Pattern Matching', LNCS 2089.

Kurtz, S. & Schleiermacher, C. (1999), 'REPuter: Fast Computation of Maximal Repeats in Complete Genomes', *Bioinformatics* **15**(5), 426–427.

Lipman, D. & Pearson, W. (1988), 'Improved tools for biological sequence comparison', *Proceedings of the National Academy of Science USA* **85**, 2444–2448.

McConnell, R.M. & Spinrad, J.P. (1999), 'Modular decomposition and transitive orientation', *Discrete Mathematics* **201**, 189–241.

Manber, U. & Myers, G. (1993), 'Suffix arrays: a new method for on-line search', *SIAM Journal on Computing* **22**, 935948.

Morgenstern, B., Frech, K., Dress, A. & Werner, T. (1998), 'DIALIGN: Finding local similarities by multiple sequence alignment', *Bioinformatics* **14**, 290–294.

Morgenstern, B. (1999), 'DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment', *Bioinformatics* **15**, 211–218.

Ning, Z., et al (2001), 'SSAHA: a fast search method for large DNA databases', *Genome Research* **11**(10), 1725–1729.

Perna, N.T., et al (2001), 'Genome sequence of enterohaemorrhagic *Escherichia coli* O157 H7', *Nature* **409**, 529–533.

Sadakane, K. (2000), Compressed Text Databases with Efficient Query Algorithms based on the Compressed Suffix Array, *in* 'ISAAC'00', LNCS 1969, pp. 410–421.

Schwartz, S., Zhang, Z., Frazer, K.A., et al (2000), 'PipMaker-A Web Server for Aligning Two Genomic DNA Sequences', *Genome Research* **10**, 577–586.

Thompson, J.D., Higgins, D.G. & Gibson, T.J. (1994), 'CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice', *Nucleic Acids Research* **22**(22), 4673-4680.

Ukkonen, E. (1995), 'On-line construction of suffix-trees', *Algorithmica* **14**, 249–260.

Vincens, P., et al (1998), 'A strategy for finding regions of similarity in complete genome sequences', *Bioinformatics* **14**, 715-725.

Waterman, M. (1995), *Introduction to Computational Biology*, Chapman & Hall, London.