A Prototype for Multiple Whole Genome Alignment

Jitender S. Deogun, Fangrui Ma, Jingyi Yang
Department of Computer Science and Engineering
University of Nebraska – Lincoln
Lincoln, NE 68588-0115, USA

Andrew Benson
Department of Food Science & Technology
University of Nebraska – Lincoln
Lincoln, NE 68588-0919, USA

Abstract

In this paper, we briefly describe a prototype of the software system we have developed for multiple whole genome alignment. To develop our algorithm, we have to solve several problems including decomposition of genomes with a suffix tree, finding an LIS for multiple MUM sequences, and iterative pairwise multiple sequence alignment. This results in an overall linear time complexity for our algorithm for finding conserved regions; and between linear and quadratic time complexity for multiple whole genome alignment. One of the motivating application is the problem of finding maximum set of conserved regions in closely related microorganisms.

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

1 Introduction

The availability of whole genome sequences opens great new possibilities for understanding the evolutionary process. For example, it provides an opportunity for studying the genetic relationship among closely related microorganisms.

Whole genome alignment between pairs of strains from a single species often reveal evidence of extensive regions that abruptly interrupt synteny [14]. 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. The algorithm for pairwise alignment for the entire genome sequences of the prokaryotes was developed by Delcher et al. [5]. The MUMmer software system, developed by Delcher et al. combines suffix tree, longest increasing subsequence (LIS), and Smith-Waterman algorithms to align a pair of whole genomes.

The approach followed by Delcher et al. first identifies the Maximum Unique Matches (MUMs) using a data structure called a suffix tree for data decomposition . McCreight's algorithm is used to implement the suffix tree. If there are two genomes A and B, the method constructs a suffix tree for genome A, then the tree is expanded by adding the suffixes of genome B into the tree using the same algorithm. A MUM is then the concatenation of maximal edge labels from an internal node to the root. The internal node has two leaf nodes and its corresponding suffixes come from two different genomes. The MUMs are labeled and assembled as a string according to their positions in the genomes. Gusfield's LIS algorithm of $O(klogk)$ time complexity [10], where k is the number of the MUMs, is used for aligning MUMs. Since k is much smaller than n (length of the genome), the complexity of the algorithm can be thought of $O(n)$ running time. After the global MUM alignment is found, the local gaps were closed by using Smith-Waterman's algorithm. The gaps include single nucleotide polymorphisms (SNPs), insertions, highly polymorphic region, and repeats. The Smith-Waterman's algorithm has time complexity $O(m^2)$, where m is the length of a polymorphic region between MUMs [16]. Again, m is much smaller than n, and therefore the time complexity is close to linear, that is, $O(n)$. The overall time complexity of the MUMmer is close to linear time [5].

Recently, Brudno and Morgenstern [3] proposed an anchored alignment approach. It used a threaded trie data structure (Aho-Corasick algorithm [1]) to find seeds. The LIS algorithm given by Gusfield [10] was used to find the highest scoring montonically increasing subsequence of alignments. The chained seeds are considered as anchor points. The fields beween the anchor points are aligned using DIALIGN program [13]. Both the above methods are designed for finding alignment of only two sequences.

No efficient algorithm has been proposed for multiple whole genome alignment. One approach for multiple sequence alignment is hidden Markov models (HMMs) [8]. This is a statistic methods and need a large set of training data to construct species specific models of genomes. It is not practical for whole genome sequence alignment.

MUM–structure of 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 orgnisms. In this paper we present a prototype of our efficient software system for alignment of multiple whole genome sequences.

2 Prototype Development

Before discussing our linear time algorithm, we introduce some concepts and notations used in this paper.

2.1 The Suffix Tree Method

Definition 1 *A* suffix tree τ *for an* n*-character string* S *is a rooted directed tree with exactly* n *leaves numbered 1 to* n*. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of* S*. No two edges out of a node can have edge labels beginning with the same character. The key feature of the suffix tree is that for any leaf* i*, the concatenation of the edge labels on the path form the root to leaf* i *exactly spells out the suffix of* S *that starts at position* i*. That is, it spells out subsequence* $S[i..n]$ *[10].*

The Figure 1 gives an example of a suffix tree for a string $S = a\,d\,f\,c\,d\,f\,x$. A suffix can be found by traversing from a leaf node to the root. For example, tracing back from leaf node 3 to the root, the edge labels are concatenated to get a substring $f\,c\;d\,f\,x$, which is the suffix of S starting at position 3. For constructing a suffix tree there is, however, one restriction is placed strings, that is, no suffix of S is same as a prefix of S. For this reason, a special character, such as $, is appended to the end of the string.

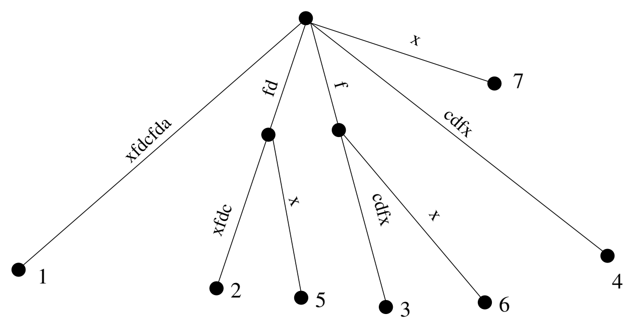

Figure 1. A suffix tree example for a string $a\,d\,f\,c\,d\,f\,x$.

A naive algorithm to build a suffix tree takes $O(n^2)$ time, where n is the length of the string [10]. A string of length n has a total of n suffixes, one starting at each position in the string. To construct the suffix tree, we start with a tree consisting of one node as a root. Suffixes are processed iteratively in the order of their positions in the string. In each iteration, a suffix is matched with the suffixes already in the tree. If no match is found, a new leaf node is attached to the root and the edge is labeled with the suffix. If a partial match is found, then a new internal node is created at the endpoint of the partial match. A new leaf node is attached to this internal node and edge labels are updated accordingly. Consider the suffix $c\,d\,f\,x$ in Figure 1 for an example, there is no match for all edge labels from the root in the current tree, so a new leaf node is created and the edge is labeled as $c\,d\,f\,x$. The suffix $d\,f\,x$, however, has a match from the root in the current tree, which is $f\,d$. So the algorithm creates a new internal node and updates the edge labels for the internal node and nodes 2 and 5 to $d\,f$, $c\,d\,f\,x$, and x, respectively.

There are, however, several linear time algorithms for building a suffix tree, such as Ukkonen, Weiner, and McCreight algorithms [12, 15, 17]. Weiner was the first researcher who developed a linear algorithm to build a suffix tree. Ukkonen's algorithm use much less memory than Weiner's and is easier to understand. McCreight's algorithm has the same space complexity as Ukkonen's.

The above algorithms, however, can also be used to build a suffix tree representing suffixes of a set of strings $S_1, S_2, S_3, \ldots, S$. This suffix tree is called a *generalized suffix tree*. A generalized suffix tree is used to solve the MUM (matching) problem. There are two approaches for constructing a suffix tree for multiple genome sequences [10]:

1. a suffix tree is first constructed for the string of the first genome sequence, then it is grown by matching the suffixes of the other genomes with edge labels of the suffix tree using the same construction methods.
2. the strings of genomes are concatenated into one string, a suffix tree ia then built for this string.

However, the first method is preferred since it can preserve the suffix positions in different genomes.

2.2 MUM Structure

After MUMs are identified using the above method, each of the MUMs is labeled with an integer from $1, 2, 3, \ldots, m$, according to their positions in the first genome, where m is the number of MUMs. This label is a unique identifier of a MUM. The MUMs may appears in different order in different genome sequences according to their positions in the corresponding genome.

Definition 2 *A* MUM sequence *is a permutation of MUM identifiers* $1, 2, 3, \ldots, m$.

A MUM sequence can be represented by intervals on a horizontal line. Such a horozontal line is called a *MUM line*. A MUM line for MUM sequence i is denoted by L . For convenience, we make all interval of unit length although MUMs are of different length. Each interval on the line represents an element in the MUM sequence, see Figure 2.

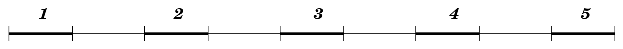

Figure 2. A representation of a MUM sequence with 5 MUMs.

Definition 3 *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 are joined together with line segments resulting in a* MUM chain*, see Figure 3. A MUM chain for MUM* x *is denoted by* C .

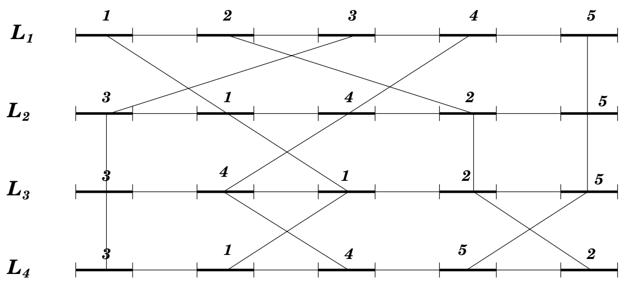

Figure 3. A MUM diagram for four MUM sequences.

To apply graph theoretic techniques to solve MUM alignment problem, we define a MUM graph.

A MUM diagram can be simplified by replacing each interval by a point on the horizontal line. The simplified version of the k-dimensional MUM diagram in Figure 3.

2.3 Discussions on Approach and Prototype

Multiple whole genome alignment can provide a powerful tool for biologists to analyze evolutionary pattern, syntenic chromosomal regions,

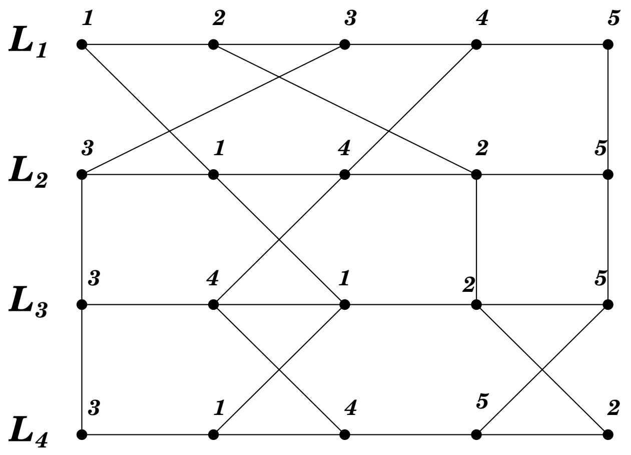

Figure 4. An example of 4-level MUM diagram.

and polymorphic regions by giving a clear picture of the similarities and differences at the genome sequence level. Since the whole genome sequence is very large, we use divide and conquer methodology to solve the problem.

First we use suffix tree method to decompose the whole genomes to find the MUMs. Secondly the MUM sequences are aligned by finding their LIS. The polymorphic regions between the MUMs are much shorter than the whole genome sequences. If the polymorphic regions are still large, a recursive call to the divide and conquer method can be employed to shorten the regions. Fanilly, this region can be aligned using iterative pairwise alignment [10]. This results in a complete alignment of multiple whole genomes.

The core problem here is to find the LIS for k MUM sequences. Gusfield [10] described a linear time algorithm for finding the LIS of one sequence using greedy cover method. However this method does not work for multiple sequences.

Our approach is based on transforming the LIS problem into a graph theoretic problem. Figure 5 gives a flow diagram of the prototype for multiple whole genome alignment.

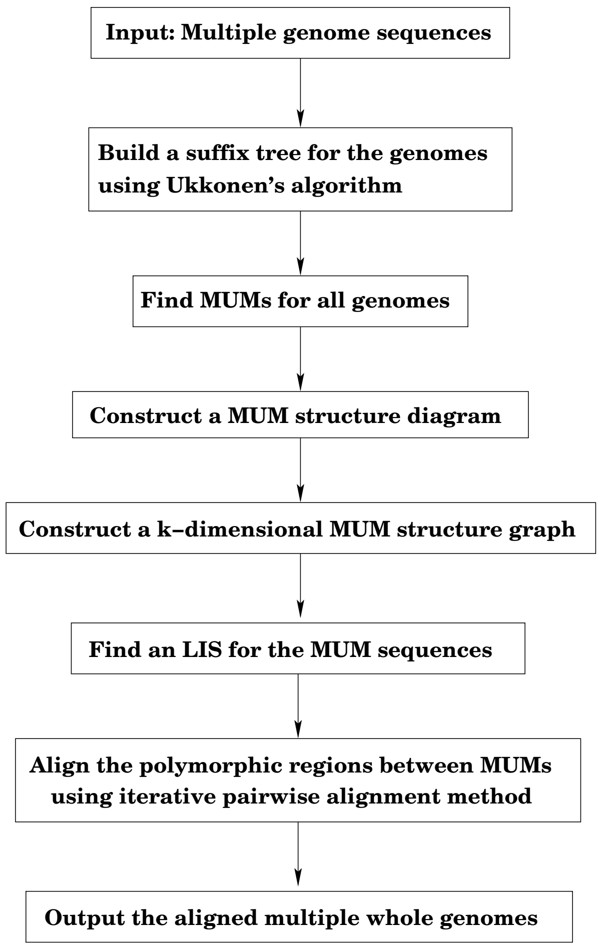
Input: Multiple genome sequences
Build a suffix tree for the genomes using Ukkonen's algorithm
Find MUMs for all genomes
Construct a MUM structure diagram
Construct a k-dimensional MUM structure graph
Find an LIS for the MUM sequences
Align the polymorphic regions between MUMs using iterative pairwise alignment method
Output the aligned multiple whole genomes

Figure 5. A flow diagram of the prototype for mutliple genome alignment.

3 Brief Description of the Algorithm

In this section, we briefly describe the important problems comprising the multiple whole genome alignment as well as algorithm used to solve them.

3.1 Find MUMs of Multiple Genomes

The MUMs are found by using the first method of generalized suffix tree structure described in section 2.1 with Ukkonen's algorithm. Each leaf node represents a suffix from one the the k genomes and has a unique identifier from 1 to k to indicate which genome the suffix comes from. If each genome is given a distinct termination symbol, identical suffixes in different genome sequences end at distinct leaves in the generalized suffix tree. Therefore, each leaf node in the tree has only one string identifier. Figure 6 shows the suffix tree constructed for three strings $S_1 = a\,d\,b\,c\,e$, $S_2 = b\,c\,b\,e\,a\,d$, and $S_3 = a\,b\,c\,d\,e$. If the child nodes of an internal node are all leaf node and have all the string identifiers, then a MUM can be found by concatenating all the edge labels from this internal node to the root. As in the example, MUM, ($b\ c$), was found by concatenating the edge label b and c.

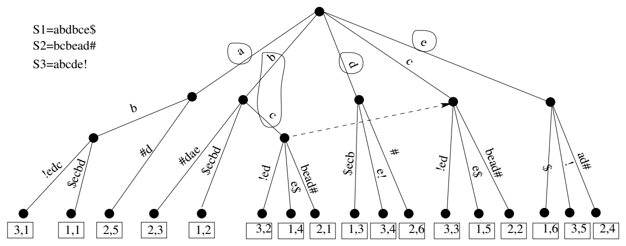

Figure 6. Generalized suffix tree for three strings listed above. In the leaf node label, the first number shows the string number, and second number indicates the starting position of the suffix in the string. The dish line with an arrow is a suffix link.

After the MUMs are found, the MUM sequences and k-level MUM structure diagram are constructed. In this example four MUMs were found. MUMs 1, 2, 3, and 4 are (a), (d), $(b,\ c)$,and (e), respectively.

Figure 7 displays all the MUMs found in the 3 sequences. We use this figure as an example as we discuss the use of graph theoretic methods to solve the LIS problem for multiple sequences.

It is worth mentioning that a suffix link is very important property in the linear time suffix algorithm. This is because the subtrees below the two suffix link nodes are same. Therefore the algorithm can move around the tree efficiently without going back to the root during the traversal.

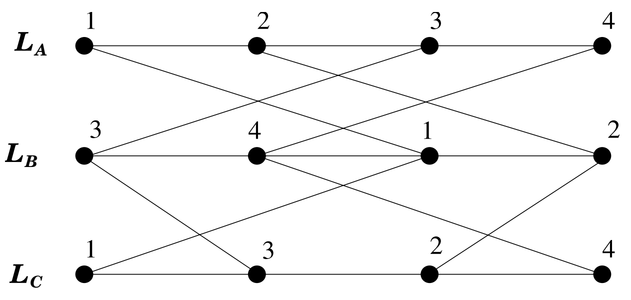

Figure 7. A 3-level MUM diagram

3.2 Finding LIS of multiple MUM sequences

After the MUMs are found, the LIS of multiple MUM sequences is computed according to our linear time algorithm [7]. So the LIS for sequences A, B, and C is $1, 2$, or $3, 4$. They are actually the conserved segments if we consider the sequences A, B, and C as genomes.

3.3 Alignment of Polymorphic Regions

This can be done by commonly used method, iterative pairwise alignment [10]. The basic idea is that this approach uses pairwise alignment scores to iteratively add one additional string to a growing multiple alignment. For detailed algorithm, please refer to [4, 10, 11]. In our example, if we choose the independent set of MUM 3 and 4, the alignment of the three sequences is given in Figure 8.

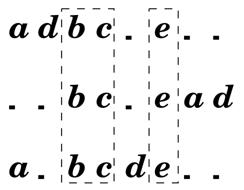

Figure 8. The alignment of the three sequences, A, , and .

4 Time Complexity Analysis

The time complexity of construction of a suffix tree and finding all MUMs is $O(n)$, where n is the length of the longest of the input genome sequences. Our algorithm of finding LIS takes $O(m^2)$ [7]. The iterative pairwise alignment takes $O(kmq^2)$, where q is the maximum number of bases in any polymorphic region. Usually, k is considered to be a constant.

It is commonly accepted that m is extremly small compared to the length of any input genome sequence. In fact, it can be assumed without loss of generality that $m \quad \overline{n}$ [5]. Therefore, the overall time complexity is linear i.e. $O(n)$ for finding the conserved regions; and between linear and quadratic time complexity for multiple whole genome alignment.

5 Implementation Issues

On the implementation of a suffix tree, the string size of a whole genome is normally a few millions of bases, the space complexity is very important for the implementation of the suffix tree algorithm. The size of alphabet has also a profound impact [10]. A linked list is prefered to an array for the DNA string, as there are maximum four possible branches since the alphabet Σ contains only 4 letters, i.e. $\Sigma = \quad A, C, G, T$. Also because of the small size of the alphabet, we can use a simple array implementation with the characters from Σ to index the array.

As the suffix tree grows, more edges may be added. Array elements also hold the edge label information [10]. Another pointer to be implemented is the suffix link, which makes the algorithm traverse the tree efficiently. Each child node needs to keep track of the suffix position and the string identification. Figure 9 gives the internal node data structure.

C++ language is used for the software development because of its advantage in maintaining and reusing the software.

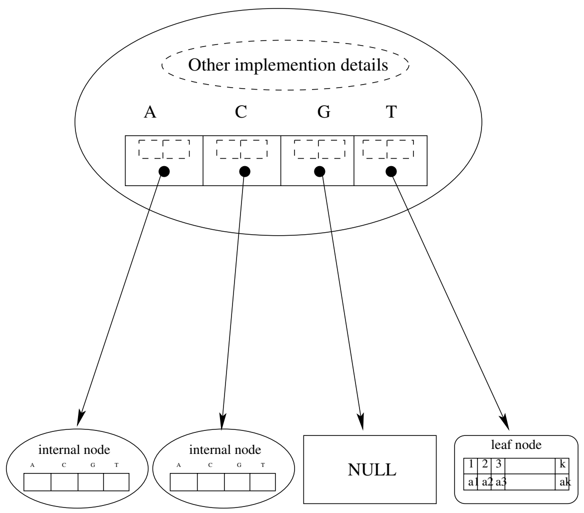

Figure 9. The structure of a internal node in the suffix tree.

6 Experimental Results

Our program was tested on SGI O300 machine with 32 500 MHz CPU. Three Ecoli whole genomes were downloaded from GenBank website, Escherichia coli K12 (4.5 MB), Escherichia coli O157:H7 (5.3 MB), and Escherichia coli O157:H7 EDL933 (5.3 MB). The minimum length of a MUM is set to 15 bp for the test run. The program output 81,778 MUMs for the Ecolis bacteria. The MUM finding operation takes total 378.327 seconds (user + system). The LIS operation takes 11,238.365 seconds.

7 Conclusions

In computational biology, the alignment of multiple whole genome sequences is an important problem, because it can provide significant information about inheritance and polymorphsim of multiple whole genomes. In this paper, we present a new algorithm for aligning multiple whole genome sequences. To develop our algorithm, we have to solve several problems including decomposition of genomes with a suffix tree, finding LIS for multiple MUM sequences, and iterative pairwise multiple sequence alignment. These problems are solved based on existing alignment and graph theoretical concepts and algorithms. A prototype of the software system developed is described in this paper. A more detailed description of the algorithm can be found in [7].

References

[1] A. Aho and M. Corasick, Efficient string matching: an aid to bibliographic search. *Comm. ACM*, **18**, pp. 333–340, 1975

[2] H.L. Bodlaender, T. Kloks, D. Kratsch, and H. Müller, Treewidth and mimimum fill-in on d-trapezoid graphs. *J of Graph Algo. and Appl.*, **2**, pp. 1–23, 1998

[3] M. Brudno and B. Morgenstern, Fast and sensitive alignment of large genomic sequences. *Proceedings of IEEE Computer Society Bioinformatics Conference, Stanford University, California*, **14-16 August**, 2002

[4] S. Chan, A. Wong, and D. Chiu, A survey of multiple sequence comparison methods. *Bull. Math. Biology*, **54**, 563–598, 1992.

[5] A.L. Delcher, S. Kasif, R.D. Fleishmann, J. Peterson, O. White, and S.L. Salzberg, Alignment of whole genomes. *Nuc. Acids. Res.*, **27**, pp. 2369–2376, 1999

[6] J.S. Deogun, T. Kloks, D. Kratsch, and H. Müller, On the Vertex Ranking Problem for d-trapezoid Graphs and for Circular-arc Graphs. *Discrete Appl. Math.*, **98**, pp. 39–63, 1999.

[7] J.S. Deogun, F. Ma, J. Yang, and A. Benson, Multiple Whole Genome alignment, *under preparation*.

[8] S. Eddy, Multiple alignment using hidden Markov models, *Proceedings of the third international conference on Intelligent systems for molecular biology, Menlo Park, California*. **16-19 July**, 1995

[9] M.C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, New York, 1980.

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

[11] M. McClure, T. Vasi, and W. Fitch, Comparitive analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evolution*, **11**, 571–592, 1994.

[12] E.M. McCreight, A space-economical suffix tree construction algorithm. *J. ACM*, **23**, pp. 262–272, 1976

[13] B. Morgenstern, DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, **15**, pp. 211–218, 1999

[14] N.T. Perna, G. Plunkett III, V. Burland, B. Mau, J.D. Glasner, D.J. Rose, G.F. Mayhew, P.S. Evans, J. Gregor, H.A. Kirkpatrick, G. Posfai, J. Hackett, S. Klink, A. Boutin, Y. Shao, L. Miller, E.J. Grotbeck, N.W. Davis, A. Lim, E.T. Dimalanta, K.D. Potamousis, J. Apodaca, T.S. Anantharaman, J. Lin, G. Yen, D.C.Schwartz, R.A. Welch, and F.R. Blattner, Genome sequence of enterohaemorrhagic *Escherichia coli* O157:H7. *Nature*, **409**, pp. 529–533, 2001.

[15] E. Ukkonen, On-line construction of suffix-trees. *Algorithmica*, **14**, pp. 249–260, 1995.

[16] M. Waterman, *Introduction to Computational Biology* , Chapman & Hall, London, 1995.

[17] P. Weiner, Linear pattern matching algorithms, *Proc. of the 14th IEEE Symp. on Switching and Automata Theory* , pp. 1–11, 1973.