2010

# A Parallel Non-Alignment Based Approach to Efficient Sequence Comparison using Longest Common Subsequences

Sanjukta Bhowmick

Mohammad Shafiullah

H. Rai

Dhundy Raj Bastola

# A Parallel Non-Alignment Based Approach to Efficient Sequence Comparison using Longest Common Subsequences

**S. Bhowmick [1], M. Shafiullah[1], H. Rai[1] and D. Bastola[2]**

1. Department of Computer Science, University of Nebraska at Omaha

2. School of Interdisciplinary Informatics, University of Nebraska at Omaha

Email dkbastola@unomaha.edu

**Abstract**. Biological sequence comparison programs have revolutionized the practice of biochemistry, and molecular and evolutionary biology. Pairwise comparison of genomic sequences is a popular method of choice for analyzing genetic sequence data. However the quality of results from most sequence comparison methods are significantly affected by small perturbations in the data and furthermore, there is a dearth of computational tools to compare sequences beyond a certain length. In this paper, we describe a parallel algorithm for comparing genetic sequences using an alignment free-method based on computing the Longest Common Subsequence (LCS) between genetic sequences. We validate the quality of our results by comparing the phylogenetic tress obtained from ClustalW and LCS. We also show through complexity analysis of the isoefficiency and by empirical measurement of the running time that our algorithm is very scalable.

## 1. Introduction

A fundamental operation in bioinformatics involves the comparison of genetic (DNA) sequences. Similarity between genetic sequences is a strong indicator of evolutionarily preserved characteristics. This property has been successfully used in determining pathologically important bacteria, viruses and fungi [1-3].

Among the many sequence comparison tools for mining genetic information, one extremely common technique includes the alignment-based methods. These involve aligning the entire (global alignment, Needleman-Wunsch[4]) or smaller sections (local alignment, Smith-Waterman [5]) of the genetic sequences. The choice of global or local alignment is based on the type of analysis desired. However, both these methods are heavily dependent on the quality of sequence data. Even slight discrepancies resulting from experimental or technical limitations, can significantly affect the comparison results.

Although the fine granularity of comparative analysis is desirable when analyzing specific biological properties such as the single nucleotide polymorphism (SNP), alternative approaches of sequence analysis are becoming increasingly important in dealing with the exponential growth of genetic sequence data, and the classification and the grouping of organisms based on these sequences. Such alternative approaches include the alignment-free methods, which match the relative (as opposed to the exact) order of the base pairs in the sequence [6-9].

Advancements in sequencing technology have provided a deluge of genetic data. The Genbank, a public repository of genetic sequence data, reported 120604423 sequence records in its 178th release in June 15, 2010. Analyzing such large datasets, including the 3 billion bases of the human reference

genome, on uniprocessor machines is an extremely time consuming process. It is imperative therefore, to harness the power of high performance computing to facilitate our understanding of this high throughput data.

In this paper, we present an efficient parallel non-alignment method for sequence comparison, based on finding the longest common subsequence (LCS), across genetic sequences. Our preliminary results demonstrate that this technique can be used to identify "signature-sequences" across multiple strains within different species of mycobacterium.

## 2. Sequence Comparison Using LCS

The longest common subsequence algorithm finds the longest subsequence between two strings. In contrast to the substring, the subsequence denotes a series of letters from the string which while being in order, need not be consecutive. For example, between ATCG and CTCAG, the longest common substring is TC, while the longest common subsequence is TCG.

LCS can help identify the key nucleotides across genetic sequences and is considerably less affected by the occasional sequencing error. This method is also useful for identifying potential regions of small mutations by analyzing the portions of the string not present in the LCS.

### *2.1. Computing the LCS*

Our algorithm for computing LCS is inspired by the FAST_LCS method described in [10]. The algorithm involves creating position pairs of identical letters in the sequence and combining them to potential LCS strings as long as their relative order is maintained across the pairs. Given two position pairs (i,j) and (l,m), the corresponding letters would be a potential candidate for LCS only if i<l and j<m.

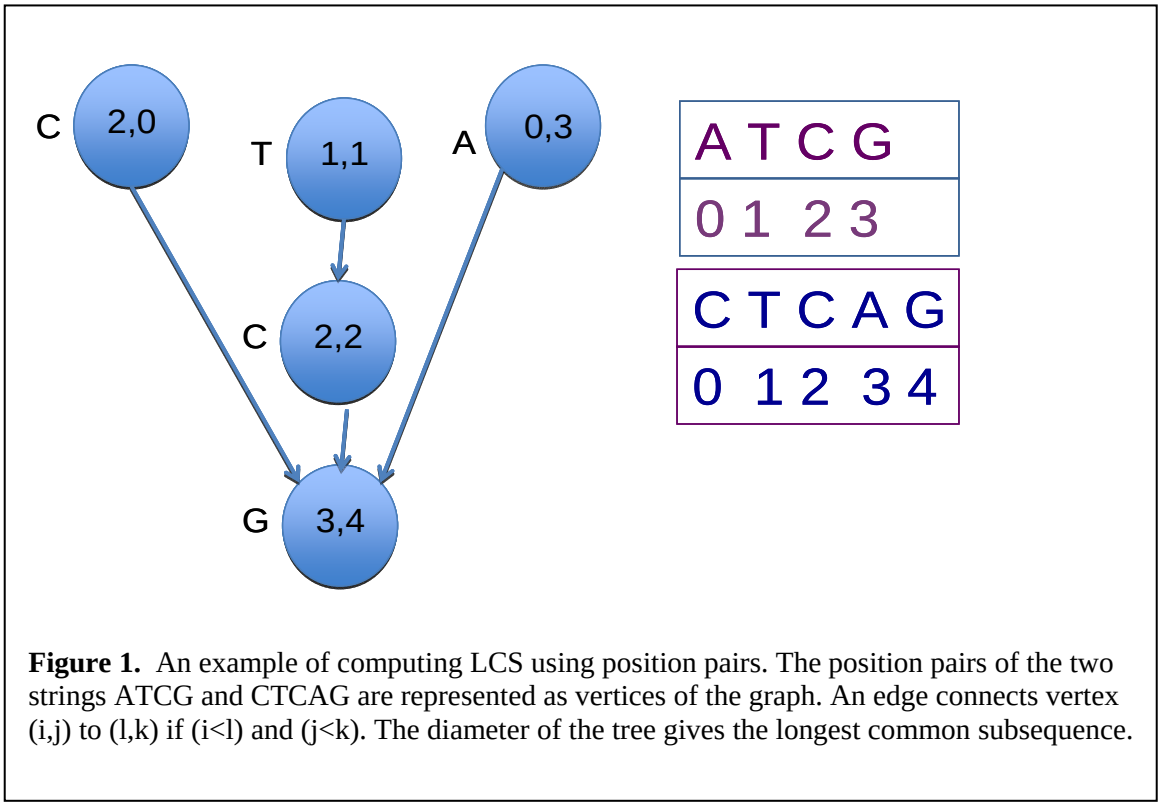


**Figure 1.** An example of computing LCS using position pairs. The position pairs of the two strings ATCG and CTCAG are represented as vertices of the graph. An edge connects vertex (i,j) to (l,k) if (i<l) and (j<k). The diameter of the tree gives the longest common subsequence.

In the sequences given above; the position pairs are; A:(0,3); C:(2,0) and (2,2); T:(1,1) and G: (3,4). Some strings preserving the relative order are: (i) AG :(0,3)->(3,4); (ii) TG: (1,1)->(3,4); and (iii) TCG: (1,1)->(2,2)->(3,4)

The algorithm consists of obtaining these position pairs and then creating a tree where the vertices represent the position pairs and the edges, their order in the subsequences. The diameter of the tree provides the LCS, as shown in the Figure1.

In order to improve the time to traverse the tree, [10] suggested pruning techniques, whereby some vertices can be deleted from their original positions and later added at a new position. This helps remove redundant edges that do not contribute to the LCS. For example though both TG and TCG are subsequences, the vertex representing the position pair of G can be later reconnected to the position pair of C, instead of being connected to T.

However, pruning is a computationally expensive algorithm since (i) it does not eliminate the initial redundant additions, (ii) deletion of edges requires more operations than additions and (iii) identifying the position of the redundant vertices requires periodic tree traversals.

To avoid these excess operations, we order the pairs such that a vertex appearing later in a string is not visited earlier. In the above example the ordered position pairs will be A:(0,3), T:(1,1), C:(2,0), C:(2,2), and G:(3,4). Therefore, TG will not be connected before TC. Furthermore, we ensure that a new letter is connected only to the longest candidate string, thereby reducing the number of edges to traverse.

Let the length of a string with L different symbols be given by;

$$M = \sum_{i=1:L} m_i ,$$

where $m_i$ is the number of occurrences of the symbol *i*. The complexity of computing LCS, for two strings of length

$$M = \sum_{i=1:l} m_i$$

and

$$N = \sum_{i=1:L} n_i ,$$

is proportional to the number of vertices and is given by

$$O(\sum_{i=1:L} (m_i \times n_i)) .$$

*2.2. Parallel Implementation of LCS*

One of the simplest methods to utilize multiple processors for sequence comparison is to execute different sets of comparisons in each processor in parallel. Though such embarrassingly parallel algorithm, as described in [10] can improve the performance, the method is not very scalable. That is, the effectiveness of the parallel algorithm is dependent on having a large number of sequence comparisons rather than being adaptable as the number of comparisons and processors change. Furthermore, the lengths of the sequences to be compared are limited by the memory in one processor.

We present a domain decomposition based parallel algorithm, which divides each sequence across the processors and *performs per sequence comparison in parallel co-ordination*. The algorithm, outlined in Figure 2, is as follow; Each sequence is divided across P processors. Then the partial LCS between the sequences in each processor is computed in parallel (Step 1 of Figure 2). Once this is completed, we compute the "unused" portions, which are portions of the sequences before (after) the first (last) positions included in the LCS. These areas are marked as grey in Figure 2.

These "unused" portions can potentially contain parts of the LCS that fall across processors. Therefore the appropriate unused portions are exchanged between processors that contain consecutive portions of the sequences. The pattern of exchange between two processors is given in Step 2 of Figure 2. Note that for each sequence comparison a processor communicates with at most two other

processors, therefore the communication pattern is scalable and independent of the total number of processors used.

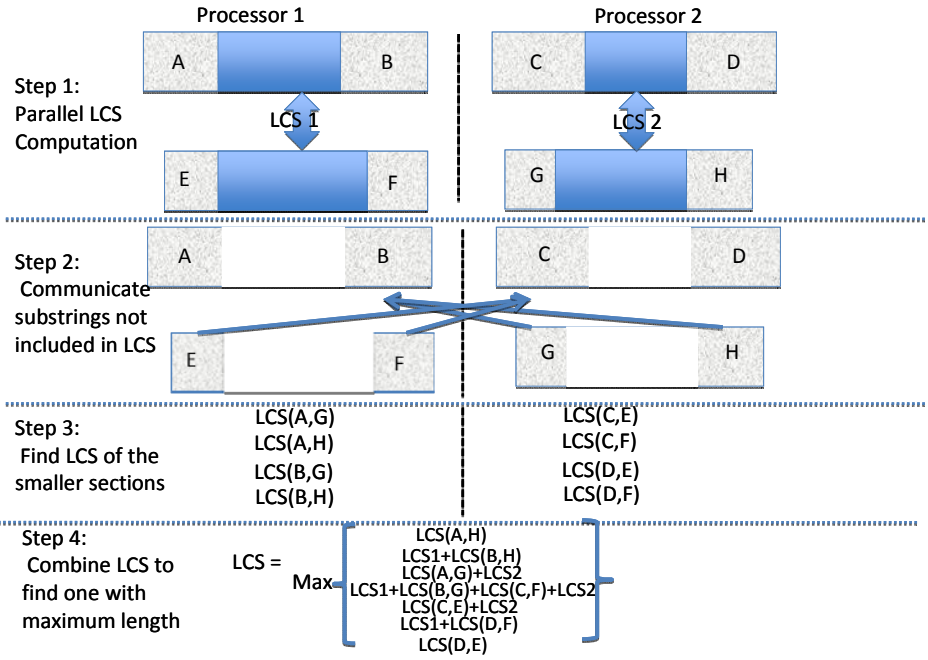


**Figure 2.** A Schematic Diagram of the Parallel LCS Algorithm

After the exchange, the LCS of the appropriate unused sequences are computed, Step 3 of Figure 2 and then finally the partial LCS portions are combined using a reduction operation. Based on the partial sequences that are compared, several overlapping LCS can be observed. We combine these portions to find the non-overlapping LCS strings of maximum length (Step 4 of Figure 2).

### *2.3. Complexity of the Parallel Algorithm*

Given two sequences with L symbols of length

$$M = \sum_{i=1:L} m_i$$

and

$$N = \sum_{i=1:L} n_i ,$$

the complexity of the sequential algorithm is

$$O\left(\sum_{i=1:L}(m_i \times n_i)\right).$$

In the parallel version each sequence is divided across P processors. Therefore the length of the sequences in each processor is approximately M/P and N/P. Based on this partition, the time to compute the partial LCS in each processor is

$$O\left(\sum_{i=1:L}((m_i/P)\times(n_i/P))\right).$$

We assume that the unused portions form approximately x% of the total length of the strings. The communication per processor is therefore for the exchange step,

$$O\left(\frac{x}{100}(m_i + n_i)\right)$$

and for the combination step,

$$O(\log P).$$

The total communication volume across all processors, when x represents a very small percentage, is therefore

$$O\left(\frac{Px}{100}(m_i + n_i) + P \log P\right) \approx O(P \log P).$$

Without loss of generality, we assume that M> N. Therefore the maximum execution time of the sequential algorithm is $O(M^2)$ and the maximum memory required to store the sequences is $O(2M)$. Therefore the memory required per processor to maintain isoefficiency should be at least

$$O(\sqrt{P} \times 1/2(\log P)/P) = O(\log P/\sqrt{P}).$$

Therefore the scalability of the algorithm is

$$O(\log P/\sqrt{P}).$$

Thus our parallel implementation is highly scalable.

## 3. Experimental Results

We conducted two sets of experiments. In the first experiments we focused on clustering different species of mycobacteria based on their genetic sequences and using multiple sequence alignment. The results showed that LCS clustering (sequential) indeed mimics the clustering obtained by multiple sequence alignment. In our second set of experiments we applied the parallel LCS algorithm to compare long sequences of *Mycobacterium tuberculosis* genome and compared the execution and change in results as more processors were used.

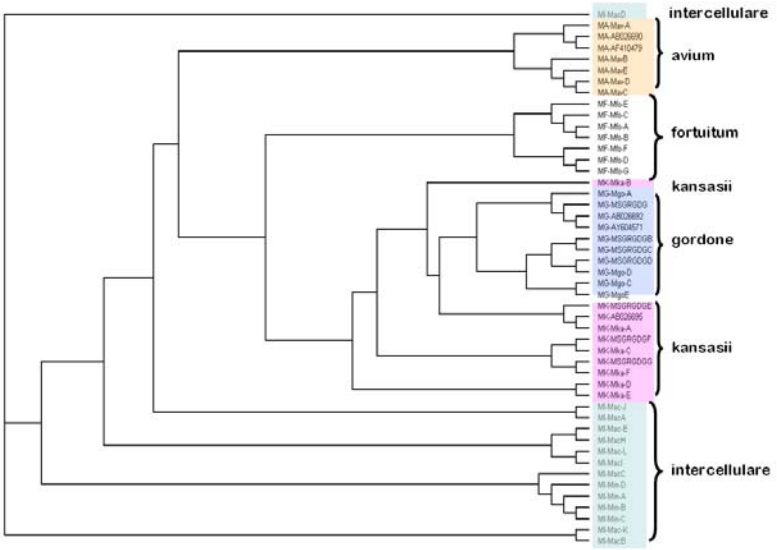


**Figure 3a**. Clustering of sequences using multiple sequence alignment. The different colors indicate the different species of mycobacterium

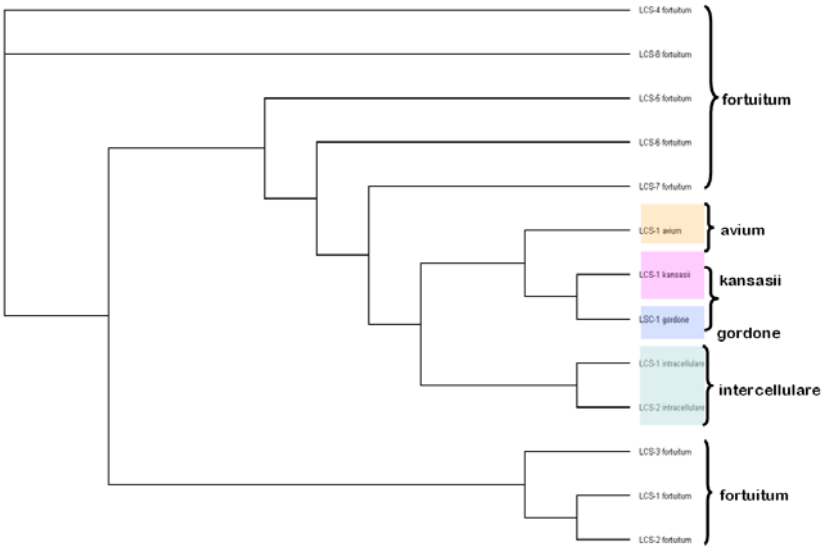


**Figure 3b.** Clustering of sequences using longest common subsequences across species. The different colors indicate the different species of mycobacterium. The species intercellulare and fouruitum have multiple LCS due to internal mutations.

*3.1. Comparison of LCS and MSA Clustering*

In our first set of experiments we computed the LCS of a portion of the genomic sequences (average length 200 base pairs) of five species of *Mycobacterium* on a sequential program. We observed that the number of distinct LCS produced per sequence pair is proportional to their variance in their base pair composition. It is observed that the sequence with low variance in composition (avium, kansasii, gordone) show a single LCS while divergent sequences (fortuitum, intercellulare) show multiple-LCS fragments.

We compared the phylogenetic clustering of the Mycobacterium using only the LCS sequence (Figure 3b) from each group of species with that obtained through multiple sequence alignment (MSA), using ClustalW[11] (Figure 3a) of all sequences across all the species. Our results indicate that the LCS clustering follows the MSA clustering very closely.

**Table 1.** The length of the LCS (over 16 to 128) for pairwise comparison of Genome Sequences from *Mycobacterium tuberculosis*.

| Processors | LCS of Sequence 1 Sequence 2 | LCS of Sequence 1 Sequence 3 | LCS of Sequence 1 Sequence 4 | LCS of Sequence 2 Sequence 3 | LCS of Sequence 2 Sequence 4 | LCS of Sequence 3 Sequence 4 |
|---|---|---|---|---|---|---|
| 16 | 1161 | 1118 | 926 | 1161 | 850 | 833 |
| 32 | 1128 | 1100 | 914 | 1087 | 843 | 830 |
| 64 | 1103 | 1098 | 914 | 1025 | 846 | 817 |
| 128 | 1111 | 1105 | 971 | 1008 | 894 | 857 |

*3.2. Parallel Implementation of LCS*

We implemented LCS based on the parallel algorithm sketched in Figure 2. Our parallel experiments were conducted on the Firefly a 1,151 node linux cluster with distributed memory architecture connected through inifiniband [12]. We used MPI to implement the parallel program.

We performed a pairwise comparison over portions of four genome sequences of *Mycobacterium tubercolosis*, with an average length of 4000 base pairs. The genome sequences used for this study were from Mycobacterium_tuberculosis_CDC1551, Mycobacterium_tuberculosis_F11, Mycobacterium_tuberculosis_H37Ra and Mycobacterium_tuberculosis_H37Rv.

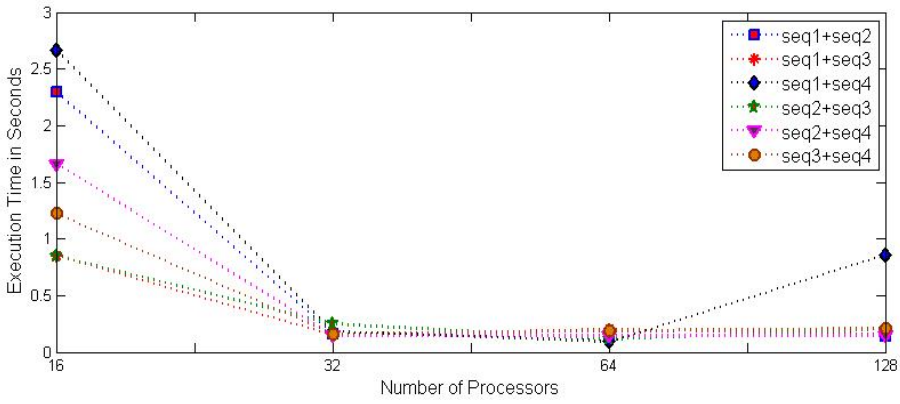


**Figure 4.** The execution time of parallel LCS algorithm as the number of processors are increased.

Each pair of sequences (total six pairs) was divided into a set of processors ranging from 16 to 128. In the first step we compared portions of the sequence that were allocated to the same processor (Step 1 in Figure 2). During Step 2 we observed that the LCS for the 16 and 32 processors started from at most the third position in the partial sequences, that is we miss only 3-4 base pairs from the left end of each sequence due to the domain decomposition. This indicates that there are only miniscule unused portions in each string that have not been compared. The total fraction of base pairs not computed by forgoing the exchange step per processor is at most 4XP/4000=.001xP. Therefore for 16 and 32 processors the maximum loss is at most 1.6% and 3.2% respectively. Given that this value is less than the percentage of errors in generating the genetic sequences [13], we decided that the comparison results would not be significantly affected by eliminating the exchange step. The cutoff portions were even smaller for the 64 and 128 processors (1-2 sequences from the left) resulting in similar percentage of potential errors.

The total length of the LCS was obtained by adding the partial LCS (which are of course non-overlapping) obtained from each processor. Table 1 shows the length of the LCS across the six sequence pairs. Since we are not computing the LCS across processors (Step 2 in Figure 2) the values differ as the number of processors change. However note that there is only a slight variation in the values for each sequence pair, indicating that they can be reliably used for clustering. This is in accordance to the spirit of using alignment-free methods, where the focus is on obtaining the relative rather than the absolute values.

These results lead us to conjecture that, if the unused regions are small, as in this case, we can obtain an accurate clustering of genome sequences by just obtaining this partial value of the LCS. Figure 4, gives the lower bound on the execution time of the parallel LCS. As can be seen from the results, the current version of the algorithm (implementing only Step 1) is highly scalable.

Note that due to the availability of over a thousand nodes and the mutual independence between the LCS computations of any two pairs of sequences, we could potentially execute the parallel

experiments for pair-wise comparison simultaneously. That is, when the LCS of one pair of sequences is being computed over n processors, the LCS of another pair can be computed at the same time over other *n* processors. Thus, we are employing a two level parallelism. In the first level we use the embarrassingly parallel nature of pair wise comparison to allocate each pair over a group of processors. In the second level we use a more tightly coupled distributed memory paradigm to compute the actual LCS.

## 4. Conclusions

We have developed a scalable parallel algorithm for computing LCS between strings. Our results demonstrate that longest common subsequence can be used as an effective non-alignment based technique for genetic sequence comparisons. Our parallel experiments show that LCS is indeed very scalable and can be used to long genomic sequences. In future, we intend to investigate between occurrences of multiple LCS across entire genome sequences of the same species and their relation to intra-species mutations.

## 5. References

[1] Rispail N, Soanes D.M., Ant C, Czajkowski R, Grünler A, Huguet R, Perez-Nadales E, Poli A, Sartorel E, Valiante V, Yang M, Beffa R, Brakhage AA, Gow NA, Kahmann R, Lebrun MH, Lenasi H, Perez-Martin J, Talbot NJ, Wendland J and Di Pietro A 2009 Comparative genomics of MAP kinase and calcium-calcineurin signalling components in plant and human pathogenic fungi. *Fungal Genet Biol.* **46** 287-98

[2] Dong QJ, Wang Q, Xin YN, Li N and Xuan SY 2009 Comparative genomics of Helicobacter pylori. *World J Gastroenterol* **15** 3984-91

[3] Ma LJ, van der Does HC, Borkovich KA, Coleman JJ, Daboussi MJ, Di Pietro A, Dufresne M, Freitag M, Grabherr M, Henrissat B, Houterman PM, Kang S, Shim WB, Woloshuk C, Xie X, Xu JR, Antoniw J, Baker SE, Bluhm BH, Breakspear A, Brown DW, Butchko RA, Chapman S, Coulson R, Coutinho PM, Danchin EG, Diener A, Gale LR, Gardiner DM, Goff S, Hammond-Kosack KE, Hilburn K, Hua-Van A, Jonkers W, Kazan K, Kodira CD, Koehrsen M, Kumar L, Lee YH, Li L, Manners JM, Miranda-Saavedra D, Mukherjee M, Park G, Park J, Park SY, Proctor RH, Regev A, Ruiz-Roldan MC, Sain D, Sakthikumar S, Sykes S, Schwartz DC, Turgeon BG, Wapinski I, Yoder O, Young S, Zeng Q, Zhou S, Galagan J, Cuomo CA, Kistler HC and Rep M 2010 Comparative genomics reveals mobile pathogenicity chromosomes in Fusarium. *Nature.* **464** 367-73

[4] Needleman SB and Wunsch CD 1970 A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **48** 443-53

[5] Smith TF and Waterman M S 1981 Comparison of biosequences. *Adv. Appl. Math.* **2** 482–89

[6] Domazet-Loso M and Haubold B 2009 Efficient estimation of pairwise distances between genomes. *Bioinformatics.* **25** 3221-7

[7] Elemento O and Tavazoie S 2007 Fast and systematic genome-wide discovery of conserved regulatory elements using a non-alignment based approach. *Genome Biology* **6** R18

[8] Quest D, Tapprich W and Ali H 2007 A grammar based methodology for structural motif finding in ncRNA database search. *Comput Syst Bioinformatics Conf.* **6** 215-25

[9] Jia C, Liu T, Zhang X, Fu H and Yang Q 2009 Alignment-free comparison of protein sequences based on reduced amino acid alphabets. *J Biomol Struct Dyn.* **26** 763-9

[10] Chen Y, Wan A and Liu W 2006 A fast parallel algorithm for finding the longest common sequence of multiple biosequences. *BMC Bioinformatics* **7** S4

[11] Larkin MA, Blackshields G, Brown NP, Chenna R, McGettigan PA, McWilliam H,Valentin F, Wallace IM, Wilm A, Lopez R, Thompson JD, Gibson TJ, Higgins DG 2007 Clustal W and Clustal X version 2.0. *Bioinformatics*. **23** 2947-8

[12] Holland Computing Center. http://hcc.unl.edu/firefly

[13] Sundquist A, Ronaghi M, Tang H, Pevzner P and Batzoglou S 2007 Whole-Genome Sequencing and Assembly with High-Throughput, Short-Read Technologies. *PLoS ONE* e484

## Acknowledgement

This project was supported by the NIH grant number P20 RR016469 from the INBRE Program of the National Center for Research Resources.