FSA: A Fast Stepwise Addition Algorithm for Constructing Phylogenetic Trees

Abolfazl Ghavidel, Mahmoud Naghibzadeh, and Omid Mirshamsi

***Abstract*—Over the last few years, the exponential growth of biological diversity achievements brought to light the need for new efficient techniques to check evolutionary relationships. One of the main methods to illustrate such relationships are phylogenetic trees. A variety of approaches which are used for tree reconstruction issue fall into two main groups: distance based and sequence based. Sequence based methods, such as maximum parsimony, can be used when the tree is reconstructed on a sequence alignment. However, in most cases it is inconceivable to compute parsimony score for all trees. On the other hand, distance based methods are extremely faster because the tree is usually constructed from a given distance matrix; nevertheless, seldom can distance matrices show the true identity of organisms. Here, we propose a novel technique to approximate the parsimony tree with stepwise addition method utilizing clustering. In this strategy we only calculate parsimony score for a part of incompletely constructed tree (called "cluster") at each step instead of the whole tree. Experiments confirm our hypothesis and do indicate that the algorithm is quite fast to build the phylogenetic tree, often with a satisfactory outcome, when there is a large dataset. We also believe that our method could prove useful as a starter tree for heuristic search approaches.**

***Index Terms*—Phylogenetic tree construction, maximum parsimony, step-wise addition.**

I. Introduction

The study of evolutionary descent amongst species, organisms, or genes as well as finding the right method to classify them has always been an interesting subject for biologists. In order to trace such evolutionary relationships, phylogenetic trees are used. From mathematical point of view, a rooted bifurcating phylogenetic tree is a non-empty proper binary tree (sometimes called 2-tree), and therefore, all mathematical relations would be the same for both tree types. To reconstruct the tree, a variety of techniques have been employed which can be divided into two main groups: distance based and sequence based (otherwise known as character based) techniques. Distance based methods are widely used because they almost work with any type of data and more importantly, they are very quick [1]. In this type of approach the tree is generally constructed from a given distance matrix, but approximating DNA/RNA/protein sequence similarities and building a phylogeny from a distance matrix that have two common problems in phylogenetics. Many researchers have tried to find efficient solutionsto address these problems. For example, M. A. Khan *et al*. [2] presented Fastphylo, a software package containing implementations of efficient algorithms. Nevertheless, distance matrices could be helpful to discover the true identity of operational taxonomic units (OTUs) only on rare occasions. Thus, because of a lack of reliability, distance based methods are usually of secondary importance to evolutionists and they prefer to select the character-based approach which contains three main methods: Bayesian inference method which tries to compute posterior distribution and estimate species divergence time. Maximum likelihood aims at finding the tree which maximizes the likelihood of the observed sequences under a given evolutionary model and maximum parsimony approach which looks for a phylogenetic tree with minimum point mutations among all branches. In spite of popularity of Bayesian inference and maximum likelihood, there has always been a marked tendency for maximum parsimony (MP), especially when morphological attributes play the leading role.

Manuscript received March 4, 2014; revised May 12, 2014.

Abolfazl Ghavidel is with the Computer Engineering Department, Azad University of Mashhad, Mashhad, Iran (e-mail: a.ghavidel@mshdiau.ac.ir).

Mahmoud Naghibzadeh is with the Computer Engineering Department, Ferdowsi University of Mashhad, Mashhad, Iran.

Omid Mirshamsi is with the Department of Biology, Ferdowsi University of Mashhad, Mashhad, Iran.

As one of the most extensively used parsimony methods for evolutionary tree reconstruction, Fitch's MP method [3] is a systematic approach in which every point mutation among all branches is calculated. In other words, total number of hypothetical substitutions for all OTUs coupled with ancestors is counted up in two phases: bottom-up and top-down. To evaluate the accuracy, Louxin Zhang *et al*. [4] analyzed Fitch method on ultra-metric phylogenetic trees. Yang *et al*. [5] have studied ambiguous and unambiguous reconstruction accuracy for N-state evolutionary models, too. Even though the final goal is finding tree(s) with minimum parsimony score, calculating all possible trees is contained in nondeterministic polynomial (NP) class. In order to solve this problem, branch and bound technique (B&B) seems to be a practical method [6] because it does accelerate execution time. Yet, never does it to reduce unfavorable time complexity. Another solution would be to use machines with many processors. The algorithm is to be paralleled [7], [8] but as it was mentioned earlier, the problem is still NP-hard. According to what mentioned, it is plain that researchers often abandon exact method to find the most parsimonious tree(s) when the number of OTUs is larger than 30. Clearly, if practicality is an important property, we should avoid strategies needing a very large memory space and/or excessive computation. In this context, there are many approaches to estimate the most parsimonious tree such as star decomposition heuristic, branch swapping approaches, etc. In general, such heuristic search methods gain benefit from a starting tree produced by a fast method (i.e. neighbor joining and stepwise addition). Although neighbor joining is classified as a fast method, it cannot handle very large datasets well. To solve this problem, Shenemanet *al*. implemented Clearcut for the relaxed neighbor joining algorithm which can handle large datasets with an appropriate result [9]. Note that these estimation methods do not guarantee the most parsimonious tree but there are, however, some solutions to great approximations [10]-[12].

Of all the above approaches, stepwise addition method [13] is a simple greedy algorithm with top-down construction pattern, which is perfectly understandable, to estimate the most parsimonious tree at a rapid rate. The other big advantage is that it does not need a large memory space either. Such substantial benefits have made it suitable as starting tree for heuristic search algorithms (e.g. Tree Bisection and Reconnection, Subtree Prune and Regraft and Nearest Neighbor Interchange). In order to prevent confusion in naming, in this paper we call it traditional stepwise addition method. Although traditional stepwise addition seems to be very quick, in fact the number of trees grows considerably for a large number of OTUs. Thus, there would probably be an enormous amount of processing.

In this paper, too, we will focus on MP criterion as well to approximate the best tree with a stepwise addition method. We begin to construct the tree with the stepwise addition approach for the first *t* nodes where *t* is the cluster threshold size. In other words, we define maximum number of nodes for each cluster so that if cluster size exceeds *t* then we break it into two clusters. We also calculate consensus sequence for each cluster, and then it will be aligned with every new node in order to select the cluster which holds the highest alignment score. Finally, using the traditional stepwise addition method, every new node is added to the optimal position among all leaf edges of the selected cluster. As it is shown in Section II.B, our technique improves the traditional stepwise addition algorithm on time complexity and builds a phylogenetic tree in a very short period of time without a significant difference in the result. We, therefore, feel our algorithm can be valuable for evolutionary reconstruction problem based on parsimony criterion and it could also prove useful as a starting tree for heuristic search approaches especially when there are a large number of taxa.

II. Methods

A. Traditional Stepwise Addition

A fast and very simple greedy method that makes a phylogenetic tree with top-down construction pattern is the traditional stepwise addition. The strategy of the algorithm is extremely easy. To estimate a rooted bifurcating phylogenetic tree, we start with a tree that has two leaves, and then at each step the optimal position is found to add a new leaf. Since we steadily look for the best location in the constructed tree, it is likely not to reach the optimal tree in the end owing to the key fact that the local optimality at current step does not guarantee the global optimality at all. More precisely, this type of hill climbing strategy just tries for local optimality without looking ahead. However, to make a better estimation, the OTUs can be inserted into the given incomplete tree in various orders: In a random order at which each step of an OTU is randomlyselected. Ranked list based which every OTU is ranked by the average difference to the other OTUs and perhaps more interesting selection method is that at each step we check each of the remaining OTUs one at a time in all positions and pick the one that gives the best score at each step. The main drawback to these approaches is that they tend to be undertaken on the off chance that they could succeed, rather than with definite expectations about the time when they will bear fruit. That is, they do not seem to be working properly because there is not a definite and reasonable rule to get the correct result. Furthermore, extra tasks need to be processed and therefore, those techniques will definitely increase the run time. To better understand, we discuss about time complexity in detailin the following subsection.

1) Time complexity

To simplify, let us assume that the main problem is the number of trees. We begin with our discussion about the number of total trees which should be searched for the optimal position at each step. In spite of how to select nodes to insert to the tree, for each new node we must search for the optimal position in the incomplete constructed tree and therefor, for *n* nodes, searching to find the optimal position is carried out $n-2$ times, and this is while in the last steps it takes more time to find the optimal position compared to the first steps. To clarify, suppose we are going to run the algorithm for *n* OTUs. In the first step, we take two OTUs and make a rooted bifurcating tree with them. Then, for all $n-2$ remained OTUs, each one will be added after the other. For example, in step two, we have a constructed tree with two leaves and so to insert the third leaf, parsimony score for three positions should be checked; similarly, in step *i* we have an incomplete constructed phylogenetic tree for the first *i* OTUs and $2i-1$ different positions should be searched. Finally, at the last step (step $n-1$) parsimony score for $2n-3$ different trees will be calculated. Therefore, if we consider the first step to be equivalent to other steps, we arrive at the following formula:

$$N = 1 + 3 + 5 + \cdots + (2n-3) = (n-1)^2 \tag{1}$$

where *N* is the number of total trees that should be calculated. More precisely, as it was mentioned, in the second step parsimony score is calculated for three different trees while each of the incomplete trees has three leaves and in the step three, parsimony score is calculated for five trees while each one has four leaves. That is to say, in step *i* we calculate $2i-1$ trees with size of $i+1$ and therefore, it sounds, a more accurate formula is needed to consider the weight of trees because there is a great difference between the trees that is checked at the first steps with those at the last steps. A simple but effective solution would be calculating total complete trees with *n* nodes, which take equal search time for finding the optimal positions to total trees with the number of nodes in range of two and *n*. For this purpose, we can multiply the weight of each tree in the number of total trees at each step. More exactly, in step *i* there are $2i-1$ incomplete trees that should be checked while each tree has $i+1$ leaves and so, each tree has the weight of $(i+1)/n$ compared to the complete tree with *n* nodes. If we apply the weight to the formula (1), total complete trees can be calculated as follows:

$$T(n) = 1 \times {}^{2}/_{n} + 3 \times {}^{3}/_{n} + \cdots + (2n-3) \times {}^{n}/_{n} = {}^{1}/_{n} \sum_{i=1}^{n-1} (2i-1)(i+1) \in O(n^2) \qquad (2)$$

$T(n)$ denotes the number of complete trees with n leaves which will take the same time to process as N incomplete trees (see formula 1) with leaves in range of 2 and n, and it is clearly evident that it has order of n^2 time complexity. Please note that we assumed the number of trees as the main problem whereas score calculation procedure and number of attributes for OTUs (for example sequence length if OTUs are DNA, RNA or protein sequences) affect time complexity but for comparison, the same operations for both methods can be overlooked. The next section is devoted to our algorithm in detail and we will see how it reduces this time complexity to a linear order.

B. *FSA (Fast Stepwise Addition)*

Performance of hill climbing algorithms is highly dependent on implementation details. For example, having aglance ahead, it is likely to make better decisions even though local optimality is not accessed. In fact, this makes our main approach to find the best edge to insert every new leaf. In this way, we utilize a novel technique to break the incomplete constructed tree into two sub-trees named clusters. For every new leaf, we first find the closest cluster and then we apply the traditional stepwise addition method to the selected cluster to insert the new leaf in the optimal position but only among leaf edges (and not all edges). That is, not only do we try to lower the total number of searches in the incompletely constructed tree by clustering, but the number of searches to find the optimal position in the selected cluster is also reduced by half and owing to this in proper binary trees the number of external nodes (leaves) is simply one more than the number of internal nodes (common ancestors). Thus, we could rightly expect that there would be a marked difference between running times of our algorithm and the traditional method. Furthermore, we use efficient techniques to find the closest cluster and the best leaf edge in the selected cluster in order to compensate for the decrease in accuracy caused by declining the total number of searches. It is also necessary to define the cluster threshold size that denotes the maximum number of possible leaves in each cluster in which if cluster size exceeds the threshold size it will be broken into two new clusters (left child and right child). The algorithm is outlined in Fig. 1 and we elaborate on it in the following subsection.

1) *Algorithm description*

Here, we explain our technique to reconstruct rooted bifurcating phylogenetic trees. We also describe how the clustering technique can be used to reduce time complexity compared with the traditional stepwise addition. Our method of calculating consensus to prevent loss of accuracy is also presented.

In order to fully understand our algorithm illustrated in Fig. 1, let us assume that the main objective is to find the optimal position in the incompletely constructed tree at each step. As it was discussed in Section II A, in step i we have a constructed tree with i leaves and*i+1th*OTU should be inserted in the one of $2i-1$ different positions. Although in the traditional stepwise addition method all different positions are checked, while they are not in our method. It is explained step by step as follows: At first, we take t OTUs from a given dataset (*OTU_Set*) to make a rooted bifurcating tree with the traditional stepwise addition method where t determines the maximum number of OTUs in each cluster (cluster threshold size). Then we add this sub-tree to the set of all clusters (*Clusters_Set*). It also should be remembered that each cluster is a sub-tree from the approximated tree, which is already being constructed. Of course, should the closest cluster be chosen with the highest possible precision and additionally if we have some success in finding the best leaf edge in the selected cluster, a satisfactory outcome in a short time will not be out of reach. However, since we try to reduce time complexity by decreasing the number of total searches to find the optimal position at each step, it seems reasonable to assume that the accuracy is declined, and therefore it is necessary that an efficient method must be used to find the best cluster, which is as close as possible to the new OTU to compensate. Consequently, to avoid blind decision-making in the traditional stepwise addition, we use a threshold size for clusters so that if the number of OTUs in a cluster exceeds the threshold size, a predefined constant[1],we break the cluster into two new clusters as shown in Fig. 2. Thus, after inserting t+1*th* OTU to the cluster, we break it into its two children (left and right sub-trees). As of now, to insert other remained OTUs, the closest cluster to every new OTU is assessed using two components:

- We first calculate the consensus sequence for each cluster.
- Then pair wise alignment is done for the new OTU and each consensus separately.

SelectedCluster is a cluster which holds the highest alignment score between its consensus and the new OUT compared to the other clusters. Here, the new leaf is added to a leaf edge of the *selectedCluster* with stepwise addition method and then cluster size is checked if it should be divided or not. The consensus sequence will be updated too.

The key to this idea is focusing on how to specify cluster threshold size, how to calculate consensus for each cluster and finally, determine which pair wise alignment algorithm is the best to be used. If we set cluster threshold size to a high number, the number of clusters will be decreased so that with $t = n$ we will only have one cluster all the time (it would never be broken) and our algorithm will work exactly the same as the traditional stepwise addition. On the other hand, if we set the cluster threshold size to a low number then the total number of clusters will go up (clusters will be broken at a rapid rate) so that with t=1, we will have the maximum number of possible clusters. Therefore, the tree is completely constructed by pair wise sequence alignment in which for every new node. We add it to the leaf with the highest score. From what was said it is easily seen that the number of clusters and cluster threshold size are inversely related and it can be argued which is the best threshold due to the trade-off between the local optimality of the traditional stepwise addition method and uncertain outcome in pairwise alignment. However, in our experiments, a large number of values were checked to find the best possible cluster size and

[1]For our experiments, we choose 20 as the cluster threshold size but it is adjustable in the implemented application.

we eventually concluded that trying to keep a balance between using the stepwise addition method and pair wise alignment would often yield a more reasonable evolutionary inference.

```
// FSA (Fast Stepwise Addition)
// INPUT: OTU_Set: represents the operational taxonomic units chosen by user(OTU dataset)
// OUTPUT: Approximated_Tree: the approximated tree returned from the function
// Cluster: is a sub-tree from Approximated_Tree contains some OTUs and their consensus
// t: predefined cluster threshold size which determines maximum number of OTUs in the clusters
// Clusters_Set: the set of all clusters
FastStepwiseAddition(OTU_Set)
{
    Take t OTUs from OTU_Set and make a rooted bifurcating tree with stepwise addition method
    Add aforementioned incomplete tree to the Clusters_Set as a cluster of the Approximated_Tree
for each remained OTU in OTU_Set{
for each Cluster in Clusters_Set
{  Do pairwise alignment between OTU and consensus of all OTUs in the Cluster}
selectedCluster = the cluster which holds the highest alignment score
       Using stepwise addition, insert OTU to a leaf edge of the selectedClusterof the
          Approximated_Tree
if (number of OTUs in the selectedCluster>t)   {
            Add (selectedCluster->RightSubtree) as a new cluster to the Clusters_Set
          Add (selectedCluster->LeftSubtree) as a new cluster to the Clusters_Set
          Calculate consensus for each of two new clusters
          Remove selectedCluster from Clusters_Set
}
else
          Update consensus for selectedCluster
}
returnApproximated_Tree
}
```

Fig. 1. Pseudo-code that presents our algorithm to reconstruct evolutionary tree.

In addition to the abovementioned explanation, it is also essential that we use an efficient procedure to calculate consensus for all clusters. More precisely, since there would be a pair wise alignment between each new sequence and consensuses of all clusters separately, these two important questions arise: How consensus is calculated for each cluster? And which pair wise alignment algorithm should be used?

Another point is that we have implemented our application so as to approximate the most parsimonious tree on DNA sequences. There is, nevertheless, no restriction to use gene sequences as OTUs. According to IUPAC nucleotide ambiguity codes for calculating consensus [14], we keep on counting the number of instances of different characters in the same columns of all sequences, which are contained in the cluster. If the number of the character exceeds the presence percentage, a predefined constant[2], this character would be included to calculate consensus. The idea of presence percentage is similar to the notion of "support" for association rules. For example suppose there are 10 sequences in the cluster which we are to calculate their consensus, if the *ith* column contains 1"C" and 9"T" it would be given a consensus of "Y" (where Y stands for pY rimidine C or T), providing that the presence percentage is set to *10* or lower. More exactly, if a character is presented below the presence percentage, it would be ignored for making consensus. We also ignore all non ACGT characters in calculating consensus. It means that in the *ith* column of 10 sequences contains 6 gaps, 3 "T" and 1"C", the presence percentage of "C" would be considered to 25% not 10%.

[2] Our experiments were carried out using values in the range of 10% to 30%. However, more appropriate results were obtained with 25%.

The other case that may occur is that none of the characters reach the presence percentage. Take, for instance, the presence percentage is set to 30. For an arbitrary column of five sequences in a specific cluster we have 1"A", 1"C", 1"G", 1"T" and 1 gap. It is clearly evident that the presence percentage of every character is 25% (by default, gap is ignored) while the minimum required presence percentage for calculating consensus is 30%. In such cases, that all characters are presented below the presence percentage, the consensus iscalculated by discarding the presence percentage value. Hence, it would be given a consensus of "*N*" (where *N* stands for aNy base). It is, however, possible to change the default settings of the application software to meet any other objective.

Pairwise alignment is another related subject that plays an important role in our approach. In fact, it is our main key to make better decisions by looking ahead rather than blind searches to find the optimal position in the traditional stepwise addition method. Its role would be more prominent if the cluster threshold size is set to a low value because the lower the threshold, the more the clusters and the more clusters the more consensuses. So to add new leaves, every leaf has to be aligned with all consensuses to locate a suitable cluster.

Global alignments are widely used when sequences are similar and of approximately equal size. Saul B. Needleman and Christian D. Wunsch [15] published a well-known global alignment on two sequences based on dynamic programming known as Needleman–Wunsch algorithm. On the other hand, local alignments are used for the most unlike sequences. A general local alignment algorithm was proposed by Temple F. Smith and Michael S. Waterman [16]. Smith–Waterman is also based on dynamic programming such as Needleman–Wunsch algorithm. With regard to the fact that finding common sub-sequences is the touchstone in sequence alignment, Majid Sazvar *et al.* [17] proposed a one-pass linear algorithm, based on dynamic programming, to discover the longest common sub-sequence (LCS) quickly. Reddy et al [18] also worked on Planted (l, d)-Motif finding which tries to identify meaningful patterns in biological sequences.

Regardless of using local or global alignment algorithm, setting scores for gap penalties could have an important part in the result. Y. Nozaki and M. Bellgard [19] developed a technique that allows the user to assign a priori set of the number of allowable gaps but it seems to be still difficult to precisely determine the penalties for a given pair of sequences. To avoid complexity, we use affine gap penalty function[3].

Having taken all the aforementioned issues into consideration, we now would like to explain how our method could build the phylogenetic trees quickly. Following subsection discusses time complexity in detail.

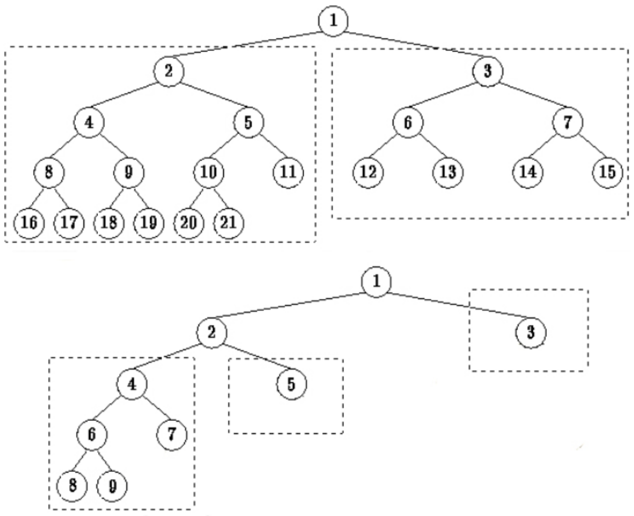

Fig. 2. Above: If the number of leaves in each cluster is greater than the cluster threshold size, then it will be broken into two clusters (left and right child) Bellow: The worst case happens if the tree is lopsided, as shown some leaves have high depths (e.g. 8, 9) in comparison to the other ones.

2) *Time complexity*

As it was explained in Section II (A), we assume that the main problem is the number of complete trees that are searched for the optimal position. Let us calculate time complexity for the worst case even though it seems not to occur. If we have *n* OTUs and define *t* as the maximum number of nodes in each cluster then the tree is being constructed with the traditional step wise addition method for first *t* OTUs and therefore, time complexity of this part is calculated exactly as the same as equation (2). As of now, the closest cluster for each remained OTU is found by pair wise sequence alignment and then, in the worst case, at most *t* different incomplete trees should be searched while each tree weighs *(i+1)/n* in step *i*.

$$W(n) = \sum_{i=1}^{t-1} \frac{(2i-1)(i+1)}{n} + \sum_{i=t}^{n-1} \left(\frac{i+1}{n} t + \zeta\right) \in O(n) \tag{3}$$

where ζ= the sum total of pair wise alignment cost at each step.

W(n) determines the number of complete trees with *n* leaves that are searched in our algorithm for the optimal position in the worst case, plus pair wise sequence alignment cost. Please note that *t* is a predefined constant and does not affect the time order of the number of trees. Thus, we easily do see that the algorithm has a linear time order of total number of trees by comparison with the traditional stepwise addition method. However, ζ is a little more difficult to talk about. More exactly, ζ is the cost of the second *"for each"* loop in our algorithm outlined in Fig. 1 and states that sequence alignment cost is obviously important, even though it is independent of the number of trees. In other words, to insert $n-t$ remained OTUs at each step, pair wise alignment is done between every OTU and consensus of each cluster separately; pair wise alignment's cost depends on clusters count at each step and clusters count depends on the number of OTUs and the value of *t*. However, for $n-t$ remained OTUs, the worst case might happen if the tree has a comb-shaped topology. For example, suppose we have constructed a rooted bifurcating tree with the first *t* nodes. Since there is only one cluster, adding the next node is to be done to this cluster. Now, cluster size is *t+1* which is greater than *t* and this cluster is broken to its left child and right child. The worst case occurs when one child has *t* leaves whereas the other has only one. Imagine that the next node has the highest alignment score with the consensus of the cluster with size of *t* compared to the cluster with size of 1. It means we should again insert the new node to the largest cluster and therefore its size would be *t+1* > *t*. After breaking such cluster to its left and right sub-trees, previous result may be obtained in order that one sub-tree has *t* leaves whereas the other has one (see Fig. 2).

III. Results and Discussion

The study investigates the impact of clustering on phylogenetic tree reconstruction with the stepwise addition method. We also compared the performance of the traditional stepwise addition method and improved stepwise method[4]. Although there is not any restriction for our algorithm for the input data types, we chose gene sequences so that the comparison is performed on run time and estimated parsimony score for a variety of different sequences of *"COI"* mitochondrial genes of animals. The impact of

[3]For our experiments, we choose −1for extend-gap penalty score and −4, −2 for open-gap penalty score for similar and semi-similar sequences respectively.

[4]All experiments were performed on Windows 7(64-bit, Ultimate edition) with a 3.3GHz Intel64 Core i3family 6 model 42 processor and 4 GB of RAM

clusterthreshold size is discussed in our experiments too.

TABLE I: EXPERIMENTS FOR SOME SAMPLE DATASETS OF ANIMALS

Datasets		Average MP Score		Execution Time (s)	
Class	*#taxa– #sites*	*Traditional Step-wise*	*FSA*	*Traditional Step-wise*	*FSA*
	21 – 628	51	52	3	3.5
	33 – 540	40	46	11	11
Amphibia	49 – 644	42	40	48	47
	77 – 540	87	81	189	88
	109 – 543	129	141	737	232
	23 – 914	926	945	6	7
	47 – 1237	187	188	102	77
Aves	57 – 379	844	854	51	27
	63 – 666	12087	10453	146	81
	111 – 455	32	23	719	170
	25 – 677	1361	1296	6	6
	50 – 657	1439	1297	105	49
Mammalia	68 – 411	1042	901	105	51
	94 – 470	1609	1132	355	128
	107 – 657	106	114	924	242
	36 – 599	623	567	17	15
	63 – 563	1341	1058	114	69
Reptilia	66 – 627	1014	961	144	95
	73 – 645	3843	4327	199	106
	110 – 552	253	289	700	193

Fifty *"COI"* datasets with sequences of different length and number were selected from NCBI GeneBank[5].

Implemented application software coupled with all sample datasets is also available at our website[6].

A. Fitch's Standard Parsimony

In order to assess the performance of the traditional stepwise addition method and our technique to reconstruct the phylogenetic tree, we consider the standard Fitch algorithm to calculate parsimony score, which is one of the most extensively used parsimony methods for evolutionary tree reconstruction. In this systematic approach, every point mutation among all branches is calculated and total number of hypothetical substitutions for all OTUs and ancestors are counted in two phases: bottom-up and top-down. We have compared our algorithm against the traditional stepwise addition method using *"COI"* datasets. Table I shows the average MP score and execution time of sample datasets calculated by our method. Looking at Table I, for more than 25 taxa, we saw how run time of FSA has reduced compared with the traditional stepwise addition method. However, when the number of taxa is lower than 25 improvements cannot be seen in the run time because of additional computations to calculate consensus and pair wise sequence alignments.

[5]National Center for Biotechnology Information (The GenBank® nucleic acid sequence database http://www.ncbi.nlm.nih.gov/nucleotide/)

[6] Application software and datasets can be accessed through: www.blueweb.ir/phylogeny

B. The Impact of Phylogenetic Tree Shape and the Number of Clusters

The analysis of phylogenetic tree shape can provide important clues as to understanding the evolutionary forces [20]. In this context, Nicolas Bortolussi*et al*. [21] described the computer package "apTreeshape" that is dedicated to simulation and analysis of phylogenetic tree topologies using statistical indices. Although analysis of the tree shape is not completely studied in this paper, it is important in our algorithm because it can influence the run time. Clearly, as it was explained earlier, the worst case occurs if our tree has a comb-shaped topology. In other words, comb-shaped means that the incomplete tree with n leaves consists of $n-t+1$ clusters (in the worst case) and it is the maximum number of possible clusters where t determines the cluster threshold size.

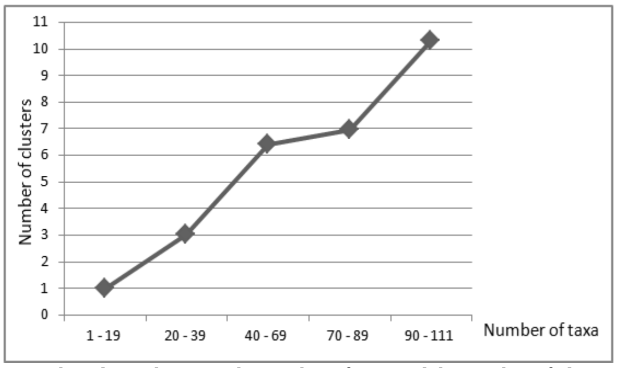

Fig. 3. The relation between the number of taxa and the number of clusters with t=20.

For example, suppose our incomplete tree has *5* leaves.

Also assume we set t=3. Therefore, the worst case occurs if we have three clusters in which one cluster has 3 leaves and each of the other two clusters has 1 (see Fig. 2). Though we calculated time complexity for the worst case, our experiments showed that this case almost does not occur (see Fig. 3).

From mathematical point of view, if we consider clusters as leaves of tree, in the worst case, the following relation between the number of clusters and the depth of tree is certified:

$$depth\_of\_tree = number\_of\_clusters \tag{4}$$

Please note that we assumed each cluster as a leaf of tree (tree of trees). Additionally, we assumed that the level of root is 1. It should also be remembered that we are talking about rooted bifurcating trees; similarly, with these assumptions the minimum number of clusters appears in balanced trees (see Fig. 2) and we have the following formula:

$$depth\_of\_tree = [Log(number\_of\_clusters)] + 1 \tag{5}$$

On the other hand, we can also show the relation between the minimum number of clusters and total number of taxa as follows:

$$x = [number\_of\_taxa / \min\_number\_of\_clusters \tag{6}$$

Equation (6) is generalized version of the pigeonhole principle and states that, at least one cluster must hold no fewer than x taxa. Another form of this equation could be:

$$min\_number\_of\_clusters = [\frac{number\_of\_taxa}{t}] \tag{7}$$

where t is a predefined variable for the cluster threshold size. According to equation (7), we will have the minimum number of clusters if all clusters perhaps except one; have maximum number of nodes (cluster threshold size). Similarly, the worst case occurs when we have one cluster with t nodes and $n-t$ clusters with one node, where n is the number of taxa.

$$max\_number\_of\_clusters = n - t + 1 \tag{8}$$

According to the mentioned equations, let us calculate the more exact cost for ζ in equation (3). As it was discussed, ζ is the total cost of pairwise alignments at each step and it depends on the number of clusters at each step. Therefore, we have the following equation in step i:

$$\zeta = \sum_{j=1}^{m} \varphi \ , \ \lceil i/t \rceil \le m \le i - t + 1 \tag{9}$$

where φ is pairwise sequence alignment cost between two sequences (the new sequence, which we want to insert to the incomplete tree, and consensus sequence of any cluster).

The minimum and maximum values for m are directly related to the minimum and maximum number of clusters at each step respectively. If we apply the value of ζ to the equation (3), for the worst case $W(n)$ can be calculated as follows:

$$W(n) = \sum_{i=1}^{t-1} \frac{(2i-1)(i+1)}{n} + \sum_{i=t}^{n-1} (\frac{i+1}{n} t + \sum_{j=1}^{i-t+1} \varphi) \tag{10}$$

Although the total cost of pairwise alignment has order of n^2 time complexity in our algorithm, the number of total positions that should be searched has order of n time complexity and therefore our algorithm, as it was shown in Table I, is much faster. Moreover, with a quick glance at Fig. 3, we do say that the worst case in our experiments not only did not occur, but on average, each cluster has also $t/2$ nodes as well.

IV. Conclusion

In this paper, we detailed a novel algorithm to estimate the most parsimonious tree with clustering technique that employs local and global alignment algorithms. In a brief summary, we utilized clustering technique to decline the number of searches to find the optimal position at each step. For that, we defined the maximum number of possible leaves in each cluster (cluster threshold size) in which if the number of leaves in a cluster exceeds the threshold size then we break it into two clusters (left and right sub trees). Though experiments showed that the cluster threshold size is a good criterion to decide when a cluster should be divided, the variance could be a good parameter to find the best time to break the cluster too. However, calculating the variance for sequences in the cluster and define a threshold value to check with it does not seem to be so easy and usually needs practical experiments too.

If we consider that the main problem is the number of incomplete trees that should be searched to find the optimal position at each step, by our construction technique, the tree is built with computational complexity of $O(n)$ while the computational complexity of the traditional stepwise addition method is $O(n^2)$. In practice, generally our method constructs phylogenetic trees with lower parsimony scores rather than the traditional stepwise addition method provided that reasonable values are selected for both the presence percentage constant and especially cluster threshold size. Since we used DNA gene sequences as input data –due to the simple assumption of equal probability of A, C, G or T– it seems reasonable that we set *25%* for the presence percentage constant and more, the inverse of the number of various attributes could be a proper value for presence percentage when morphological attributes play the leading role. It may be, however, necessary to do some practical experiments to find the best number for the cluster threshold size because it is a parameter that directly affects the outcome.

The source code of FSA was written in *C#*.net4programming language with a sequential single thread programming model[7] and it shown that the FSA implementation for gene sequences has a better performance

[7] Application software and datasets can be accessed through: http://www.blueweb.ir/phylogeny

compared with the traditional stepwise addition method. We believe that our method is useful as a starter tree for heuristic search approaches. Furthermore, we also think that it can be improved by multi-threading programming model, however parallelism brings some challenges.

ACKNOWLEDGMENT

We wish to thank Mostafa Sabzekar and Amin Rezaeian for helpful discussions and suggestions on some of the issues, and A. Ghavidel would especially like to thank Ferdowsi University of Mashhad's knowledge engineering research group (KERG) as well as all those who helped with this research.

REFERENCES

[1] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic Networks: Concepts, Algorithms and Applications*, Cambridge University Press, 2010.

[2] M. A. Khan, I. Elias, K. Nylander, R. V. Guimera, R. Schobesberger, P. Schmitzberger *et al.*, "Fastphylo: Fast tools for phylogenetics," *BMC bioinformatics*, vol. 14, pp. 334, 2013.

[3] W. M. Fitch, "Toward defining the course of evolution: minimum change for a specific tree topology," *Systematic Biology*, vol. 20, pp. 406-416, 1971.

[4] L. Zhang, J. Shen, J. Yang, and G. Li, "Analyzing the Fitch method for reconstructing ancestral states on ultrametric phylogenetic trees," *Bulletin of Mathematical Biology*, vol. 72, pp. 1760-1782, 2010.

[5] J. Yang, J. Li, L. Dong, and S. Grünewald, "Analysis on the reconstruction accuracy of the Fitch method for inferring ancestral states," *BMC Bioinformatics*, vol. 12, p. 18, 2011.

[6] M. Hendy and D. Penny, "Branch and bound algorithms to determine minimal evolutionary trees," *Mathematical Biosciences*, vol. 59, pp. 277-290, 1982.

[7] D. A. Bader, V. P. Chandu, and M. Yan, "ExactMP: An efficient parallel exact solver for phylogenetic tree reconstruction using maximum parsimony," *Parallel Processing*, 2006, pp. 65-73.

[8] W. T. J. White and B. R. Holland, "Faster exact maximum parsimony search with XMP," *Bioinformatics*, vol. 27, pp. 1359-1367, 2011.

[9] L. Sheneman, J. Evans, and J. A. Foster, "Clearcut: a fast implementation of relaxed neighbor joining," *Bioinformatics*, vol. 22, pp. 2823-2824, 2006.

[10] I. Elias and J. Lagergren, "Fast neighbor joining," *Theoretical Computer Science*, vol. 410, pp. 1993-2000, 2009.

[11] J. Felenstein, *Inferring Phylogenies*, vol. 2, Sinauer Associates Sunderland, 2004.

[12] U. W. Roshan, T. Warnow, B. M. Moret, and T. L. Williams, "Rec-I-DCM3: a fast algorithmic technique for reconstructing phylogenetic trees," in *Proc. Computational Systems Bioinformatics Conference*, 2004, pp. 98-109.

[13] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *Journal of Molecular Evolution*, vol. 17, pp. 368-376, 1981.

[14] A. Cornish-Bowden, "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984," *Nucleic Acids Research*, vol. 13, p. 3021, 1985.

[15] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, pp. 443-453, 1970.

[16] T. F. Smith and M. S. Waterman, "Comparison of biosequences," *Advances in Applied Mathematics*, vol. 2, pp. 482-489, 1981.

[17] M. Sazvar, M. Naghibzadeh, and N. Saadati, "Quick-MLCS: a new algorithm for the multiple longest common subsequence problem," in *Proc. the Fifth International C* Conference on Computer Science and Software Engineering*, 2012, pp. 61-66.

[18] U. S. Reddy, M. Arock, and A. Reddy, "A particle swarm optimization solution for challenging planted (l, d)-Motif problem," in *Proc. Computational Intelligence in Bioinformatics and Computational Biology* , 2013, pp. 222-229.

[19] Y. Nozaki and M. Bellgard, "Statistical evaluation and comparison of a pairwise alignment algorithm that a priori assigns the number of gaps rather than employing gap penalties," *Bioinformatics*, vol. 21, pp. 1421-1428, 2005.

[20] F. A. Matsen, "A geometric approach to tree shape statistics," *Systematic Biology*, vol. 55, pp. 652-661, 2006.

[21] N. Bortolussi, E. Durand, M. Blum, and O. François, "apTreeshape: statistical analysis of phylogenetic tree shape," *Bioinformatics*, vol. 22, pp. 363-364, 2006.

Abolfazl Ghavidel is a M.S. student in software engineering at Azad University of Mashhad, Mashhad, Iran. He was born in 1983, received his BS degree in 2007.

He currently works as an expert of "Statistics & IT" as well as network administrator in Cultural Heritage and Tourism Organization of Mashhad. He was the head of"IT Office" of the organization between 2009 and 2010.

His research interests are in computational vision relate to algorithms, computational complexity, bioinformatics and phylogeny. He has developed FSA software application which can be accessed through: http://www.blueweb.ir

Mahmoud Naghibzadeh received his B.S. degree in statistics and computer science from Ferdowsi University of Mashhad, Iran and his MS and PhD degrees in computer science and computer engineering, respectively, from University of Southern California, USA. He is now a full professor at the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran.

He is currently teaching graduate courses and supervising both MS and PhD students. In addition, he is the director of Knowledge Engineering Research Group (KERG) laboratory. His research interests include the scheduling aspects of real-time systems, Grid, Cloud, Multiprocessor, and Multicore and Bioinformatics computer algorithms. He has published numerous papers in international journals and conference proceedings as well as eight books in the field of computer science and engineering.

Prof. Naghibzadeh was the general chair of an international computer conference and technical chair of two others. He is the reviewer of many journals and a member of many computer societies as well as a senior member of IEEE. He is the recipient of many awards including MS and PhD study scholarship and outstanding professor award.

Omid Mirshamsi is an assistant professor in zoology at the Department of Biology, Ferdowsi University of Mashhad, Mashhad, Iran. His research interests include phylogeny and systematics of arachnids in Iran.