A Novel Quartet-Based Method for Phylogenetic Inference

B. B. Zhou, M. Tarawneh, C. Wang, A. Zomaya
School of Information Technologies
University of Sydney
NSW 2006, Australia
{bbz,monther,cwang,zomaya}@it.usyd.edu.au
&
R. P. Brent
Mathematical Science Institute
Australian National University
Canberra, ACT 0200, Australia
rpb@rpbrent.co.uk

Abstract

In this paper we introduce a new quartet-based method. This method makes use of the Bayes (or quartet) weights of quartets as those used in the quartet puzzling. However, all the weights from the related quartets are accumulated to form a global quartet weight matrix. This matrix provides integrated information and can lead us to recursively merge small sub-trees to larger ones until the final single tree is obtained. The experimental results show that the probability for the correct tree to be among a very small number of trees constructed using our method is very high. These significant results open a new research direction to further investigate more efficient algorithms for phylogenetic inference.

1. Introduction

There has been strong interest in reconstruction of large evolutionary trees from small sub-trees in the computational biology community in recent years [2,3,4,5,6,8,10,11,18,19,22]. The quartet-based method may be the simplest and the most cost-effective one for such reconstruction. This approach takes two major stages to complete, i.e., first generates the most likely sub-tree(s) for every possible four sequences, or quartet from a set of *n* DNA or protein sequences using a general phylogeny method [21], such as maximum parsimony, maximum likelihood, or neighbour joining, and next applies a combinatorial technique to reconstruct the entire evolutionary tree topology based on the topological properties of the generated four-sequence or quartet trees. It is well known that it is hard to obtain correctly resolved quartet trees by using currently existing methods [1,14]. Though there are interesting theoretical properties, many quartet-based algorithms assume a fully resolved tree for each quartet and this unreasonable assumption seriously hinders the quartet-based approach from being widely adopted in practice. Therefore, the main concern in designing a good and practical quartet-based algorithm is how to tolerate errors in the quartet trees when reconstructing the entire tree topology in the second stage.

Several techniques were proposed to try to improve the accuracy of the quartet-based approach. A prominent approach which is widely used in practice is known as quartet puzzling [19]. This method was originally developed to try to reduce the computational cost of the maximum likelihood method. Though the method has problems, such as accuracy, the introduction of weighted quartet trees and likelihood mapping [20] gives hopes for further developments of quartet-based algorithms which can be used in practice. The quartet puzzling takes both resolved and unresolved quartet trees into consideration. However, it is essentially a local optimization process and thus is very sensitive to quartet errors [16].

In this paper we introduce a novel quartet-based algorithm. Like the quartet puzzling, this algorithm rebuilds the evolutionary tree using quartet weights. However, it takes a global view by looking at all possible quartet trees when merging a pair of sub-trees. Similar to the distance-based neighbour joining, it recursively merges one pair of sub-trees at each merge step based on global statistics provided by a symmetric matrix which is called global quartet weight matrix in this paper. Accumulating all related quartet weights from all possible quartets generates this matrix. Because we use global information from all the quartets, we are able to determine, with high confidence, which two sub-trees should be merged at each step. We can also easily identify those critical points where the ambiguity occurs, i.e., the places where several pairs of sub-trees could be merged in about the same probability, and then take proper actions. Our current algorithm generates a limited number of trees and the initial experimental results, using the benchmarks which consist of 48,000 synthetic data sets of DNA sequences generated by the LIRMM Methods and Algorithms in Bioinformatics research group [23], show that the probability for the correct tree to be among the generated trees is very high. These significant results open a new research direction to further investigate more efficient algorithms for phylogenetic inference.

The paper is organized as follows: In Section 2 we describe the global quartet weight matrix and give an efficient $O(n^2)$ algorithm for generating the matrix from a given unrooted binary tree topology. In Section 3 we show that with a simple modification this algorithm can be used to reconstruct the original tree topology from its generated weight matrix. Therefore, there is a one-to-one mapping between a given tree topology and its generated global quartet weight matrix. This forms the basis of our quartet-based neighbour-joining (QBNJ) method which is discussed in Section 4. Section 5 presents some experimental results. Conclusions and future work are discussed in Section 6.

2. Global Quartet Weight Matrix

A quartet, or a set of four sequences is associated with one of seven possible graphs [7]: three fully resolved unrooted trees, three partially resolved trees for which we are unable to distinguish between two of the three fully resolved trees, and a completely unresolved tree. The three possible fully resolved trees for a quartet $\{a, b, c, d\}$ are depicted in Fig. 1. In the figure $(xy \mid zt)$ indicates how the sequences, represented by leaf nodes, are divided into two pairs by cutting the middle edge (so-called bi-partitioning), and thus shows the neighbourhood relations of the quartet in terms of topology.

We can plot all the quartets from a given n sequences in a two-dimensional graph to see how many quartets are resolved (and how many unresolved) using the likelihood-mapping procedure [20]. With the likelihood-mapping procedure, likelihood values for each of the three possible resolved trees are first calculated and then these likelihood values are transformed into posterior probabilities, or Bayes weights (or quartet weights in this paper) w_i for i = 1, 2, and 3, by applying Bayes' theorem assuming a uniform prior for all three trees. Three quartet weights of each quartet are further mapped onto a two-dimensional simplex. With this mapping we may see how closely the maximum-likelihood (ML) method can resolve each quartet into one of the three possible trees. A generalized method, called quartet mapping, was developed based on the same principle to measure other approaches for quartet tree construction and used to compare different methods in their abilities to indicate the correct topology [12].

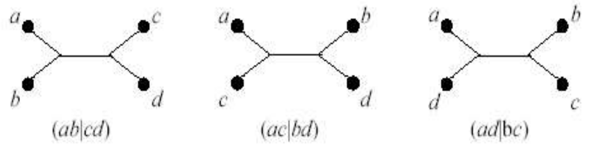

Fig. 1. The three possible fully resolved trees for a quartet {a, b, c, d}.

For a given n sequences and the associated set of quartets, we can generate a symmetric global quartet weight matrix of size $n \times n$, each row or column corresponding to one particular sequence. Let $(ij \mid kl, w)$ denote a quartet tree with a quartet weight *w*. Our global quartet weight matrix is generated by adding each *w* to entries *ij*, *ji*, *kl* and *lk*, using a complete set of quartets from the given sequences.

Given a tree topology of n leaves, we can uniquely determine a set of $\binom{n}{4}$ quartet trees which are consistent with the original tree, i.e., each quartet tree separates the four leaves into two pairs in the same way as the original tree through bi-partitioning. For each quartet there can only be one fully resolved tree in the set of these quartet trees and it is described as $(ab \mid cd, 1.0)$. From a given tree topology a global quartet weight matrix is also uniquely determined. This is because our matrix is generated using the quartet weights of the associated quartet trees. In the following we present an $O(n^2)$ algorithm to construct the weight matrix from a given tree topology. In next section we describe an algorithm which can reconstruct the original tree topology from its generated weight matrix. This shows a one-to-one mapping between a given tree topology and its generated global quartet weight matrix.

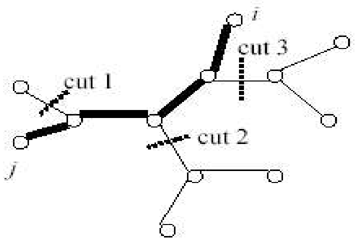

Fig. 2. Calculating the number of quartet trees which have the form of (*ij*|*xy*, 1.0) using the bi-partitioning technique.

To determine the entry value on row *i* and column *j* of the global weight matrix from a given tree topology, we need to calculate the total number of quartet trees with the form $(ij \mid pq, 1.0)$, for $p, \text{or } q \neq i, \text{or } j$. This can be done using the bi-partitioning technique. We first determine a unique path from leaf node *i* to node *j*, as shown in Fig. 2. For each internal node on that path we make a cut on the third edge which is not on the path to separate the tree into two sub-trees. This ensures that leaves *i* and *j* will always be in the same sub-tree. Counting the number of leaves in the sub-tree which does not contain leaves *i* and *j*, the number of quartet trees with the form $(ij \mid pq, 1.0)$ from this bi-partitioning will be $\binom{n_k}{2}$ where n_k is the number of leaves in that sub-tree. (Note $\binom{n_k}{2} = 0$ if $n_k < 2$.) Therefore, the total number of such quartet trees, or the value for entry *ij* in the weight matrix will be $\sum_{k=1}^{L} \binom{n_k}{2}$ where *L* is the number of internal nodes on the path.

Note that it is not efficient to use the bi-partitioning procedure to calculate each entry value of the weight matrix the same way as described above. This is because for each leaf node we need to find the paths to every other node and then for each path we also need to set cuts for every node on the path and count for each bi-partitioning the number of nodes in the other sub-tree. This way the tree has to be traversed many times and there will be a great amount of redundant calculations. Using dynamic programming, we can eliminate the redundant calculations and obtain a very efficient algorithm.

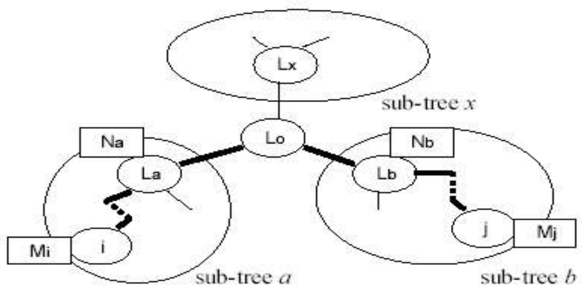

Fig. 3. Each internal node is associated with three sub-trees in an unrooted tree.

Each internal node in an unrooted binary tree has three edges. Therefore, a tree will be divided into three sub-trees when an internal node is removed. For example, in Fig. 3 three sub-trees *a*, *b* and *x* are connected by node l_o which has three edges connecting to nodes l_a, l_b, and l_x in the three sub-trees, respectively. Node l_a, (or l_b, or l_x) is called the leading node of a sub-tree since it is the only node for the sub-tree to be linked with other sub-trees. Our interest is to calculate the total number of quartet trees with the form (*ij*|*pq*, 1.0) where *i* is a leaf node in sub-tree *a* and *j* a leaf node in sub-tree *b*. It is obvious that nodes l_a, l_o and l_b must be on the path from leaf node *i* to leaf node *j*. The calculation for the total number of quartet trees which have the concerned form can then be divided into three parts according to the three sub-trees, that is, the quartet trees of the concerned form generated in sub-tree *a* from leaf node *i* along the path to the leading node l_a, the quartet trees generated in sub-tree *b* from node *j* to node l_b, and those generated through a bi-partitioning on the third edge of node l_o.

After removing the third edge of node l_o, we partition the tree into two sub-trees, one being the combination of sub-trees *a* and *b* and the other being the entire sub-tree *x*. Assume that we know the number of leaf nodes in sub-trees *a* and *b* which are n_a, and n_b, respectively. The number of leaf nodes in sub-tree *x* will be $n - (n_a + n_b)$ for *n* the total number of leaf nodes in the tree. The number of quartet trees which can be generated to have the form (*ij*|*pq*, 1.0) through this bi-partitioning is simply equal to $\binom{n-(n_a+n_b)}{2}$. Assume further that we know the total number of the quartet trees with the concerned form in sub-tree *a* from leaf node *i* along the path to the leading node l_a, is m_i, and the number in sub-tree *b* from node *j* to node l_b is m_j. The total number of the quarte trees which have the form (*ij*|*pq*, 1.0) can then be obtained as

$$m_{ij} = \binom{n-(n_a+n_b)}{2} + m_i + m_j. \qquad (1)$$

This is also the entry value on row *i* and column *j* of the global weight matrix.

After every entry value m_{ij} associated with the leaf nodes in sub-trees *a* and *b* are calculated, we merge the two sub-trees into one. Node l_o will be the leading node and the number of leaf nodes becomes $n_a + n_b$ for the new sub-tree, i.e.,

$$n_o = n_a + n_b. \qquad (2)$$

For each leaf node in sub-tree *a*, we add $\binom{n_b}{2}$ to m_i, i.e.,

$$m_i = m_i + \binom{n_b}{2}. \qquad (3)$$

This is because when we further merge this new and larger sub-tree with another sub-tree, we need to know the number of concerned quartet trees from each leaf node *i* of the original sub-tree *a* to the new leading node l_o. However, we already know the number from node *i* to node l_a. We need only to make an additional bi-partitioning by cutting the third edge of node l_o. This will make the original sub-tree *b* be completely separated from the rest. Therefore, $\binom{n_b}{2}$ more quartet trees of the concerned form can be generated on the path from leaf node *i*, which is originally in sub-tree *a*, to the new leading node l_o. For the same reason $\binom{n_a}{2}$ will be added to m_j which is associated with each leaf node in sub-tree *b*, i.e.,

$$m_j = m_j + \binom{n_a}{2}. \qquad (4)$$

With the above four equations we can obtain an efficient algorithm for obtaining the entry values of the global matrix from a given tree topology. In this algorithm each leaf node is associated with a variable m_i to store the number of quartet trees of the concerned form obtained from the current sub-tree. It seems that for each node *i* we might need $n-1$ variables to store the numbers for different *ij* pairs. However, a single variable will be enough since the path from node *i* to the leading node is the common sub-path for node *i* to reach any other nodes outside the sub-tree. Each sub-tree is associated with a variable n_i to record the number of leaf nodes in the sub-tree. Initially, every leaf node is considered as a separate sub-tree containing a single node. Thus the values m_i and n_i are set to zero and one, respectively. Sub-trees are recursively merged together according to the tree topology using the above four equations until there left only a single tree. A pseudocode of the algorithm is shown in Fig. 4.

```
For each leaf node set $n_i = 1$ and $m_i = 0$ (initialise $n$ sub-trees)
While the number of sub-trees does not equal to one (main loop of the recursion)
  Do find two sub-trees $a$ and $b$ with a common parent node $o$ according to the tree topology
    For each leaf node $i$ in $a$ and each node $j$ in $b$
      Do calculate the entry value: $m_{ij} = \binom{n-(n_a+n_b)}{2} + m_i + m_j$
    For each leaf node $i$ in sub-tree $a$
      Do update $m_i$: $m_i = m_i + \binom{n_b}{2}$
    For each leaf node $j$ in sub-tree $b$
      Do update $m_j$: $m_j = m_j + \binom{n_a}{2}$
    Set $n_o$ for internal node $o$: $n_o = n_a + n_b$ (node $o$ becomes the leading node for the
     merged sub-tree)
```

Fig. 4. An algorithm for generating the quartet matrix.

The size of each sub-tree will be no greater than *n*. Each **For** loop in the above takes only $O(n)$ operations. Thus it takes $O(n)$ operations to merge two sub-trees. There are $n-1$ such steps to merge all leaf nodes into a single tree according to the tree topology. Therefore, the total computational cost of this algorithm will be $O(n^2)$.

3. Tree Reconstruction from the Generated Global Weight Matrix

In the previous section we developed an $O(n^2)$ algorithm which can generate the global quartet weight matrix from a given tree topology. With a simple modification of that algorithm we can obtain a new one which can reconstruct the original tree from its generated matrix.

In the algorithm for generating the weight matrix, each row in the matrix corresponds to a particular leaf node and is associated with a variable m_i which records the number of the concerned quartet trees generated on the path from the leaf node to the leading node of the sub-tree. Each sub-tree is also associated with a variable n_i to record the number of leaf node in the sub-tree. When there is more than one leaf node in a sub-tree, we choose one node as representative for that sub-tree. Initially every leaf node is considered as a sub-tree. Therefore, each row in the matrix will be associated with the two variables and they are set to $m_i = 0$ and $n_i = 1$. Since we only have a generated matrix, but have no a tree topology to follow when merging sub-trees, to find the correct edges in the original tree we need to use a special procedure.

With two representative nodes i and j of the sub-trees, we can calculate a value d_{ij} using equation

$$d_{ij} = \binom{n-(n_i+n_j)}{2} + m_i + m_j.$$

This equation is exactly the same as that in (1) we used to calculate entry values of the matrix. The two sub-trees are directly connected in the original tree only if d_{ij} is equal to the corresponding value m_{ij} in the matrix. We can then define a confidence value

$$c_{ij} = m_{ij} / d_{ij}. \qquad (5)$$

For any two sub-trees to be merged together the corresponding confidence value must be equal to one.

When there are more than one leaf nodes in each sub-tree, there is no need to calculate all the c_{ij} values associated with every leaf node pair, one node from each sub-tree. When two sub-trees are directly connected in the original tree, the confidence value c_{ij} for every associated node pair must be equal to one. Therefore, we need only to use their representative nodes. We us one additional variable g_i for representative node i to store index j when $c_{ij} = 1$. This can greatly save the computational cost.

It is easy to verify that the total cost of this modified algorithm is also $O(n^2)$.

4. Tree Construction from Inaccurate Global Weight Matrices

In the previous section we discussed an algorithm for rebuilding the original tree from its generated global weight matrix. The same algorithm may be used to construct an evolutionary tree for a given set of n sequences if all the associated quartets are fully and correctly resolved. Unfortunately, this is only an ideal case and in reality it is very hard for us to have all the quartets fully and correctly resolved, Therefore, the global weight matrix generated from a set of quartets will be inaccurate and the algorithm in the previous section for tree topology reconstruction from its generated weight matrix cannot be used without modification. To deal with inaccurate weight matrices we make three major changes to the original algorithm.

Average confidence value $\bar{c}_{ij}$: Since the entry values of the global weight matrix are no longer ideal, different node pairs, one from each of the two sub-trees, may produce different confidence values. A simple way to alleviate this problem is to calculate the confidence values for all leaf node pairs, average them and use this averaged value as the confidence value $\bar{c}_{ij}$ for each pair of sub-trees.

Since the entry values of the weight matrix are inaccurate, we may not obtain $\bar{c}_{ij} = 1$ for a pair of sub-trees during the computation. In addition to the three variables, namely, g_i, m_i and n_i, associated with each sub-tree, we need a new variable $\bar{c}_i$ to record the highest average confidence value for sub-tree i with another sub-tree j for $i<j$. At each step we compare the stored values in $\bar{c}_i$ and choose to merge the two sub-trees which have the highest average confidence value.

Quartet weight correction: After two sub-trees are merged, we restore the associated entries in the matrix to their "true" values, i.e., change the quartet weights based on the currently reconstructed sub-trees and update the weight matrix accordingly. In particular, after each merge we need to correct the weights of all those quartets containing four nodes $\{i, j, p, q\}$ to $(ij|pq, 1.0)$ where i is a leaf node in one merged sub-tree, j is a leaf node in the other merged sub-tree and p and q the leaf nodes from the rest. If the weights are not corrected, the distributed errors may significantly affect the correct decision making in the following merge steps. With quartet weight correction procedure we are able to force the way a tree is constructed by setting certain particular average confidence value $\bar{c}_{ij}$ to one and modify the corresponding quartet weights. This is a very important procedure for us to change the direction of tree reconstruction.

With the above two modifications we can have an algorithm which is able to deal with inaccurate quartet weights. This algorithm is depicted in Fig. 5.

At each merge step the three major contributors to the total computational cost are: (1) the updating of m_i values for each leaf node, (2) the quartet weight correction, and (3) calculation of average confidence values for each sub-tree to find the highest one. We can easily prove that the costs for first and third contributors mentioned above are $O(n^2)$ and $O(n^3)$, respectively, after n merge steps. Each quartet weight is corrected once only during the computation. The total number of quartets for a given set of n sequences is $\binom{n}{4} = O(n^4)$ and thus the total cost for quartet weight correction. Therefore, the total computational cost for the algorithm will be $O(n^4)$. It should also be noted that this cost is much less than the cost for computing quartet weights using the maximum likelihood which requires $O(sn^4)$ operations where s is the length of the sequences, usually a few hundreds to a few thousands.

```
For each row $i$, set $n_i = 1$ and $m_i = 0$ (initialise $n$ sub-trees)
For each row $i$, $(1 \le i < n)$
  Do (initialise $\bar{c}_i$ and $g_i$ for every sub-tree.)
     $g_i = 0$ and $\bar{c}_i = 0$
     For each column $j$ $(i < j \le n)$
        Do calculate $\bar{c}_{ij}$. If $\bar{c}_{ij} > \bar{c}_i$, then set $\bar{c}_i = \bar{c}_{ij}$ and $g_i = j$ (store the largest $\bar{c}_{ij}$.)
While the number of sub-trees does not equal to one (main loop of the recursion)
  Do find the largest $\bar{c}_a$ and the associated $g_a = b$
     Add an internal node to merge sub-trees $a$ and $b$
     (Update $m_i$ values for every leaf node in sub-trees $a$ and $b$.)
     For each row associated with a node $i$ in $a$
        Do Update $m_i$: $m_i = m_i + \binom{n_b}{2}$
     For each row associated with a node $j$ in $b$
        Do Update $m_j$: $m_j = m_j + \binom{n_a}{2}$
     Update $n_o$ for the new sub-tree: $n_o = n_a + n_b$ (stored in the array entry associated with
       the representative node $o$ of the sub-tree.)
     (Quartet weight correction.)
     Update the quartet weight matrix by correcting the weights of those quartets containing
       $\{i, j, p, q\}$ to $(ij \mid pq, 1.0)$, for $i$ being a leaf node in sub-tree $a$, $j$ a leaf node in $b$
     (Re-calculation of average confidence values.)
     For sub-tree $i$, $(1 < i < m$ for $m$ the total number of sub-trees currently.)
       Do $g_i = 0$ and $\bar{c}_i = 0$
          For each other sub-tree $j$ $(i < j < m)$
             Do calculate $\bar{c}_{ij}$. If $\bar{c}_{ij} > \bar{c}_i$, then set $\bar{c}_i = \bar{c}_{ij}$ and $g_i = j$
```

Fig. 5. An algorithm for tree construction from inaccurate quartet weight matrices.

Multiple tree reconstruction: Since the matrix is not accurate, it may not always be the right decision to merge the two sub-trees that have the highest confidence value. After the highest confidence value $\bar{c}_{ij}$ is obtained, we then check whether there is another sub-tree *k* which has a reasonably high confidence value associated with one of the two sub-trees *i* and *j*, that is, we check whether $\bar{c}_{ik}$ (or $\bar{c}_{ki}$) $\ge \alpha \bar{c}_{ij}$, or $\bar{c}_{jk}$ (or $\bar{c}_{kj}$) $\ge \alpha \bar{c}_{ij}$ where α is a threshold which is smaller than, but close to one. (If there are several sub-trees which satisfy this condition, in our current version we simply choose the one with the highest confidence value among them.) At each of these critical points we can have three different super quartet trees with four sub-trees *i*, *j*, *k*, and the rest as its four super nodes at different places. The problem is which one will be the correct one leading us to find the correct tree topology. In the current version of our algorithm we keep all three different patterns. Therefore, we will reconstruct multiple trees and hope that the correct tree will be included in these generated trees. However, we need a control on the number of trees to be reconstructed. Otherwise, we may end up with about 3^n different trees if every merge step is a critical point. We use a parameter *s* to limit the total number of trees. Each time a critical point is encountered, two extra trees are generated until *s* such stages are encountered for each tree. Therefore, the maximum number of trees to be generated will be limited to 3^s.

4. Experimental Results

In our experiment we use the benchmarks developed by Ranwez and Gascuel who used these benchmarks to test and compare different phylogeny methods [16]. Six model trees, each consisting of 12 leaf nodes, are used to generate test data sets under various situations. Three model trees, named AA, AB and BB, are molecular clock-like, while the other three, named CC, CD and DD, present varying substitution rates among lineages. Four evolutionary rates conditions are considered, ranging from low to very fast, for which the maximum pair-wise divergence (MD) is from 0.1 to about 2.0 substitutions per site. With these model trees and varying evolutionary conditions, a total of 48,000 test data sets of DNA sequences are generated using Seq-Gen[15]. A detailed description of these synthetic test data sets can be found on their Web site at www.lirmm.fr.

In our experiments we set α to 0.85. The number of stages was set to 0 (QBNJ0, to allow only a single tree), 4 (QBNJ4, maximum 81 trees), 5 (QBNJ5, maximum 243 trees) and infinity (QBNJINF, to allow unlimited number of trees). We measured the average number of trees generated per data set and the percentage of correctly inferred trees. (Note that the correctly inferred tree here means the correct tree topology is found among a number of trees generated for a data set.) Some experimental results are depicted in Table 1 for DNA sequences of length 300 and Table 2 for DNA sequences of length 600. The figure x / y in the tables denotes the percentage of correctly inferred trees / the average number of trees generated per data set. The figures are all rounded to their nearest integers.

		AA	BB	AB	CC	DD	CD
$MD \simeq 0.1$:	DNAPARS	19	16	15	13	15	17
	BIONJ	17	16	15	16	15	18
	FASTDNAML	23	20	21	16	18	18
	QBNJ0	22	11	9	10	10	10
	QBNJ4	63/57	49/53	52/54	56/45	61/47	59/44
	QBNJ5	68/94	57/89	58/82	58/68	61/69	61/65
	QBNJINF	70/124	58/117	59/120	58/86	64/88	62/80
$MD \simeq 0.3$:	DNAPARS	28	36	26	46	53	51
	BIONJ	33	34	34	56	57	56
	FASTDNAML	58	53	54	70	68	69
	QBNJ0	44	30	21	37	32	32
	QBNJ4	87/44	78/40	79/46	88/24	91/26	89/26
	QBNJ5	91/61	81/55	83/60	89/28	92/28	90/31
	QBNJINF	92/66	82/61	83/68	89/29	92/28	90/32
$MD \simeq 1.0$:	DNAPARS	6	8	6	7	9	10
	BIONJ	22	20	21	62	63	65
	FASTDNAML	44	36	37	82	83	81
	QBNJ0	43	19	13	41	39	32
	QBNJ4	86/54	67/51	65/56	88/31	90/31	90/32
	QBNJ5	91/79	72/79	72/87	89/38	94/37	92/39
	QBNJINF	91/89	74/90	72/101	90/41	95/37	92/42
$MD \simeq 2.0$:	DNAPARS	0	0	0	0	0	0
	BIONJ	2	3	3	29	31	33
	FASTDNAML	8	4	5	59	62	59
	QBNJ0	12	2	2	9	10	13
	QBNJ4	45/71	19/73	20/72	45/59	65/57	58/59
	QBNJ5	56/144	26/158	29/167	53/100	70/94	65/100
	QBNJINF	60/207	31/344	34/294	55/129	71/124	67/130

Table 1. Experimental results for DNA sequences of length 300.

		AA	BB	AB	CC	DD	CD
$MD \simeq 0.1$:	DNAPARS	57	53	52	54	57	56
	BIONJ	52	50	54	60	57	56
	FASTDNAML	69	63	65	66	61	62
	QBNJ0	56	43	36	46	39	37
	QBNJ4	94/25	88/25	91/24	92/16	93/17	93/19
	QBNJ5	95/29	89/28	92/27	92/17	93/18	93/19
	QBNJINF	95/29	89/29	92/29	92/18	93/18	93/19
$MD \simeq 0.3$:	DNAPARS	65	77	69	83	86	88
	BIONJ	76	77	81	90	92	92
	FASTDNAML	93	87	91	94	97	96
	QBNJ0	83	74	64	83	84	78
	QBNJ4	99/14	97/16	99/16	99/5	99/7	99/6
	QBNJ5	99/14	97/16	99/16	99/5	99/8	99/6
	QBNJINF	99/14	97/16	99/16	99/5	99/8	99/6
$MD \simeq 1.0$:	DNAPARS	22	33	25	17	12	15
	BIONJ	62	62	65	92	92	93
	FASTDNAML	85	77	82	99	99	98
	QBNJ0	77	63	49	87	82	78
	QBNJ4	99/24	95/26	96/26	99/8	100/13	99/11
	QBNJ5	100/25	96/28	97/30	99/8	100/13	99/11
	QBNJINF	100/26	96/28	97/30	99/17	100/13	99/11
$MD \simeq 2.0$:	DNAPARS	4	2	1	0	0	0
	BIONJ	22	18	17	76	71	73
	FASTDNAML	35	26	28	93	91	94
	QBNJ0	42	20	12	48	42	40
	QBNJ4	86/50	63/52	64/56	89/40	93/33	93/36
	QBNJ5	91/74	67/84	72/87	92/57	94/40	94/45
	QBNJINF	92/79	69/106	73/103	92/62	94/40	94/48

Table 2. Experimental results for DNA sequences of length 600.

In the tables we also present the results, i.e., percentages of correctly inferred trees, obtained from DNAPARS (the parsimony program (version 3.5c) of the PHYLIP package), BIONJ (an improved neighbour joining method [9]) and FASTDNAML (the maximum-likelihood program (version 1.2) [13]) for the same data sets. These data are simply copied from [16] for the purpose of comparison.

We can see from the tables that the basic version of our algorithm, i.e., QBNJ0 for which the number of stages is set to 0, does not perform as well as all other three methods except model tree AA for which some results obtained by QBNJ0 are comparable to the others. The accuracy of quartet-based algorithms is dependent on the quality of quartet weights (or quartet trees generated). Even though the globally integrated information from all the quartets is used at each step to determine the merge operation (and we saw that QBNJ0 constantly performs better than TREE-PUZZLE using the same sets of quartet trees in our experiments), the basic version of our algorithm is still sensitive to the errors of quartet trees. When the number of trees is allowed to increase, however, the percentage of correctly inferred trees is higher than all other three methods under all the categories and in certain cases can be much higher. For DNA sequences of length 300 and $MD \approx 0.1$, for example, the average number of trees constructed using QBNJ4 is only around 50. This number, or even the maximum number of trees allowing to be constructed, is tiny when it is compared with the total number of possible trees for a set of 12 sequences. However, the percentage of the correct trees which are found among the constructed trees, on an average from all six model trees, is about 3.5 times the percentage obtained using DNAPARS, or BIONJ, and about 3 times the percentage obtained using FASTDNAML. For QBNJ5 and QBNJINF the figures are even higher. This comparison seems unfair since our method builds a number of trees for each data set, but the others construct only one. To our knowledge so far, however, no existing phylogenetic methods can guarantee to produce better results than ours for obtaining the correct tree by constructing only a very limited number of trees. These significant results obtained from our experiments verify the effectiveness of our quartet-based neighbour-joining method.

5. Conclusions

In this paper we introduced a new quartet-based method for phylogenetic inference. The experimental results show that our method can significantly outperform most existing popular methods if a limited number of trees is allowed to be generated. We are conducting more experiments using other data sets both synthetic and practical to test our method and then make further improvement.

In practice it is very important and desirable to construct a number of trees for a posterior analysis since no existing methods can guarantee to construct a correct tree for many hard problems in practice. However, the issues are the probability that the correct tree is among the generated trees, and the probability that a correct tree can be derived from a number of generated trees.

The probability is very high for the correct tree to be among a very limited number of trees constructed using our method. We can also provide important information on where the trees differ (i.e., critical points). We are doing experiments using the maximum likelihood as the criterion at each critical point to try to find the right direction for constructing correct trees and so to reduce the number of trees to be generated.

We also use the maximum likelihood to calculate the likelihood values of the reconstructed trees obtained using our quartet-based method and then select one tree with the highest likelihood value. If the maximum likelihood is able to correctly identify the correct trees, it is clear that just adding this one more procedure at the end, our method will significantly outperform most existing popular methods on the basis of constructing only a single tree! We shall present some interesting results in the near future.

It is well known that the accuracy of a quartet-based method strongly depends on the quality of quartet trees. Our method is quartet based and has no exception. It is thus important to find more efficient methods to improve the quality of quartet trees, or reduce quartet errors. This will be our future research topic.

10. References

[1] J. Adachi and M. Hasegawa, Instability of quartet analyses of molecular sequence data by the maximum likelihood method: the cetacean/artiodactyla relashionships, *Cladistics*, Vol. 5, 1999, pp.164-166.

[2] H. J. Bandelt and A Dress, Reconstructing the shape of a tree from observed dissimilarity data, *Adv. Appl. Math*., Vol. 7, 1986, pp.309-343.

[3] V. Berry and D. Bryant, Faster reliable phylogenetic analysis, Proceedings of 3rd *Annual International Conference on Comp. Mol. Biol*, 1999, pp.59-68.

[4] V. Berry, T. Jiang, P. Kearney, M. Li, T. Wareham, Quartet cleaning: improved algorithms and simulation, *Lecture Notes Computer Science*, Vol. 1643, 1999, pp.313-324.

[5] V. Berry, D. Bryant, P. Kearney, M. Li, T. Jiang, T. Wareham and H. Zhang, A practical algorithm for recovering the best supported edges in an evolutionary tree. *Proceedings of Symposium on Discrete Algorithms*, San-Francisco, 2000, pp.287-296.

[6] V. Berry and O. Gascuel, Inferring evolutionary trees with strong combinatorial evidence, *Theoret. Comput. Sci*., 240(2), 2000, pp. 271-298.

[7] M. Eigen, B. F. Lindemann, M. Tietze, R. Winkler-Oswatitsch, A. Dress, and A. von Haeseler, How old is the genetic code? Statistical geometry of tRNA provides an answer, *Science*, New Series, Vol. 244, No. 4905, 1989, pp.673-679.

[8] P. Erdos, M. Steel, L. Szekely and T. Warnow, Constructing big trees from short sequences, *Lecture Notes Computer Science*, Vol. 1256, 1997, pp.827-837.

[9] O. Gascuel, BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data, *Mol. Biol. Evol*., 14, 1997, pp.685-695.

[10] D. H. Huson, S. Nettles and T. Warnow, Obtaining highly accurate topology estimates of evolutionary trees from very short sequences, *Proceedings of The 3rd Annual Int. Conf. Comp. Mol. Biol*., 1999, pp. 198-209.

[11] T. Jiang, P. E. Kearney and M. Li, Orchestrating quartets: Approximation and data correction, *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998, pp.416-425.

[12] K. Nieselt-Struwe and A. von Haeseler, Quartet-mapping, a generalization of the likelihood-mapping procedure, *Mol. Biol. Evol.*, 18(7), 2001, pp.1204-1219.

[13] G. Olsen, H. Matsuda, R. Hagstrom and R. Overbeek, A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood, *Comput. Appl. Biosci*., Vol. 10, 1994, pp.41-48.

[14] H. Philippe and E. Douzery, The pitfalls of molecular phylogeny based on four species, as illustrated by the cetacean/artiodactyla relationship, *J. Mamm. Evol.*, Vol. 2, 1994, pp.133-152.

[15] A. Rambaut and N. Grassly, Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees, *Comput Appl Biosci* 13, 1997, pp. 235-238.

[16] V. Ranwez and O. Gascuel, Quartet-based phylogenetic inference: Improvements and limits, *Mol. Biol. Evol*., 18(6), 2001, pp.1103-1116.

[17] H. A. Schmidt, K. Strimmer, M. Vingron and A. von Haeseler: TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3), Mar 2002, pp.502-504.

[18] M. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, *J. Classification*, Vol. 9, 1992, pp.91-116.

[19] K. Strimmer and A. von Haeseler, Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies, *Mol. Biol. Evol.*, 13(7), 1996, pp.964-969.

[20] K. Strimmer, and A. von Haeseler, (1997) Likelihood–mapping: A simple method to visualize phylogenetic content of a sequence alignment, *Proc. Natl. Acad. Sci.* USA, 94, 1997, pp.6815–6819.

[21] D. L. Swofford, G. J. Olsen, P. J. Waddell and D. M. Hillis, Phylogeny reconstruction in D. M. Hillis, C. Moritz and B. K. Mable, eds., Molecular Systematics, 2nd edition, Sinauer Associates, Sunderland, Mass., 1996.

[22] S. Willson, Measuring Inconsistency in phylogenetic trees, *J. Theoret. Biol*., Vol. 190, 1998, pp.15-36.

[23] The LIRMM Methods and Algorithms in Bioinformatics research group, www.lirmm.fr.