On a New Quartet-Based Phylogeny Reconstruction Algorithm

B. B. Zhou[1], M. Tarawneh[1], D. Chu[1], P. Wang[1], C. Wang[1], A. Y. Zomaya[1], and R. P. Brent[2]

[1]School of Information Technologies
University of Sydney
NSW 2006, Australia
bbz@it.usyd.edu.au

[2]Mathematical Science Institute
Australian National University
Canberra, ACT 0200, Australia
rpb@rpbrent.co.uk

Abstract

Recently we developed a new quartet-based algorithm for phylogenetic analysis [22]. This algorithm constructs a limited number of trees for a given set of DNA or protein sequences and the initial experimental results show that the probability for the correct tree to be included in this small set of trees is very high. In this paper we further extend the idea. We first discuss a revision to the original algorithm to reduce the number of trees generated, while keeping the high probability for the correct tree to be included. We then deal with the issue on how to retrieve the correct tree from the generated trees and our current approach is to calculate the likelihood values of these trees and pick up a few best ones which have the highest likelihood values. Though the experimental results are comparable to that obtained from currently popular ML based algorithms, we find that it is common that certain incorrect trees can have likelihood values at least as large as that of the correct tree. A significant implication of this is that even if we are able to find a truly globally optimal tree under the maximum likelihood criterion, this tree may not necessarily be the correct phylogenetic tree!

1. Introduction

The quartet-based method is one of the very important approaches for reconstruction of a large evolutionary tree from a set of smaller trees. It constructs a tree for a given number of molecular sequences based on the topological properties of each subset of four molecular sequences. The main advantage of this method is that there is a one-to-one correspondence between a tree topology and a set of four-sequence or quartet trees. If we can correctly identify the tree topology of each individual subset of four sequences, we are able to reconstruct the entire evolutionary tree for a given problem in polynomial time. In practice, however, there exist situations that the correctly resolved quartet trees are very difficult to obtain by using currently existing methods [1,14]. 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. Different methods have been introduced in the literature to deal with the problem of quartet errors, for example, those in [2,3,4,5,6,7,8,9,10, 12,13,15,16,17,18,20].

Recently, we developed a new quartet-based algorithm for the reconstruction of evolutionary trees [22]. This algorithm constructs a limited number of trees for a given set of DNA or protein sequences based on the topological information of every possible quartet trees. Our initial experimental results showed that the probability for the correct tree to be included in this small set of trees is very high.

In this paper we further extend the idea. We first present a revision to the original algorithm to reduce the number of trees generated, while keeping the high probability for the correct tree to be included. We then deal with the issue on how to retrieve the correct tree from those generated trees. Our current approach is to calculate the likelihood values of these trees and pick up a few best ones which have the highest likelihood values. It was assumed that if the maximum likelihood criterion is able to correctly identify the correct trees, our method can thus significantly outperform many existing popular methods on the basis of constructing only a single tree. Though the experimental results are comparable to that obtained from currently popular ML based algorithms, our experimental results show that it is common that certain incorrect trees can have likelihood values at least as large as that of the correct tree.

This confirms the result of a theoretical study on ML presented in [19]. A significant implication of this is that even if we are able to find a truly globally optimal tree under the maximum likelihood criterion, this tree may not necessarily be the correct phylogenetic tree!

It is well known that most quartet-based algorithms require $O(n^4)$ computational steps to complete where n is a given number of molecular sequences in the analysis. This is simply because they need to generate $O(n^4)$ quartet trees in order to obtain a reasonably good result. When the problem size n is large, we also need a large-size memory to store these trees during the computation. In comparison with some fast algorithms for phylogenetic analysis, e.g. those recently developed and presented in [11,21], this seems to be a big disadvantage. However, the experimental results presented in this paper suggest that, because of its excellent theoretical properties, the quartet-based method should never be overlooked. The problem of high computational cost can be alleviated using high-performance, or parallel computing systems [23].

The paper is organized as follows: our quartet-based algorithm is briefly described in Section 2. Its extension is discussed in Section 3. In Section 4 we present some experimental results. Conclusions are given in Section 5.

2. The Original Algorithm

In this section we briefly describe our quartet-based algorithm. A more detailed description can be found in [22].

Our quartet-based algorithm for phylogenetic analysis consists of two major stages. In stage one we calculate quartet weights for every possible quartet trees from a given number of molecular sequences. In stage two we first generate a global quartet weight matrix to gather the quartet topological information from the quartet weights calculated in stage one and then reconstruct a full size tree using this quartet weight matrix. We can use any existing method for phylogenetic analysis to calculate the weights of quartet trees [15]. In the following we only discuss the computations in stage two. We first discuss how a quartet weight matrix is generated from a set of quartets defined by a given tree topology and give an efficient algorithm for reconstruction of the tree topology from its generated quartet weight matrix. This one-to-one mapping between a given tree topology and its associated quartet weight matrix forms the basis of our quartet-based algorithm for phylogenetic analysis. We next discuss the tree reconstruction algorithm and show how to deal with quartet errors.

2.1. Basic concept

A quartet, or a set of four sequences is associated with three possible fully resolved trees, as shown in Figure 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. One way to measure which of the three possible trees is more likely to be the true tree is to use Bayes weights [15], or quartet weights in this paper. The quartet weights for three possible trees of a quartet is obtained by first calculating the likelihood value for each tree and then transforming these likelihood values into posterior probabilities, or quartet weights w_i for i = 1, 2, and 3, by applying Bayes' theorem assuming a uniform prior for all three possible trees.

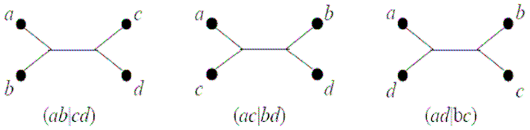


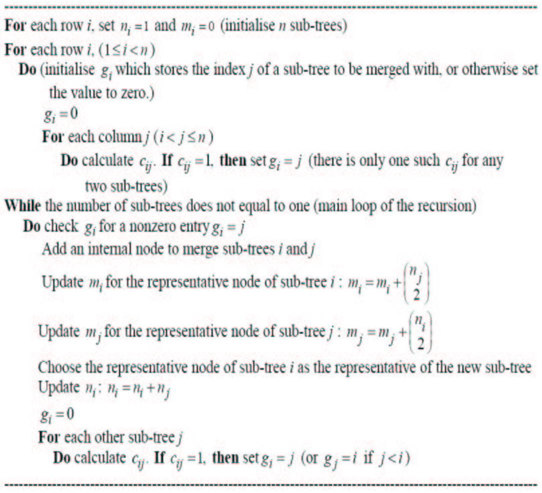
Figure 1. Three possible fully resolved trees for a quartet {a, b, c, d}.

Let $(ij \mid kl, w)$ denote a possible 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 a given number of sequences. This matrix is symmetric and its size is $n \times n$, where n is the total number of sequences and each row or column corresponds to one particular sequence. (Note this quartet weight matrix is called score matrix in [10] and it was generated using discrete weights (or scores) from a distance matrix.)

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 $(ab \mid cd, 1.0)$. For 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.

We can also reconstruct the tree topology from its generated quartet matrix by using an efficient $O(n^2)$ algorithm, as shown in Figure 2. This algorithm is derived using the dynamic programming technique

[22]. In the figure each row in quartet weight matrix corresponds to a particular leaf node and is associated with a variable m_i. One node in each sub-tree (or a row in the quartet weight matrix) is chosen as the representative node for the sub-tree which is also associated with a variable n_i. Initially every leaf node is considered as a sub-tree. Therefore, each row in the matrix will be associated with two variables which are set to $m_i = 0$ and $n_i = 1$.



```
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 $g_i$ which stores the index $j$ of a sub-tree to be merged with, or otherwise set
      the value to zero.)
     $g_i = 0$
     For each column $j$ $(i < j \le n)$
       Do calculate $c_{ij}$. If $c_{ij} = 1$, then set $g_i = j$ (there is only one such $c_{ij}$ for any
          two sub-trees)
While the number of sub-trees does not equal to one (main loop of the recursion)
  Do check $g_i$ for a nonzero entry $g_i = j$
     Add an internal node to merge sub-trees $i$ and $j$
     Update $m_i$ for the representative node of sub-tree $i$ : $m_i = m_i + \binom{n_j}{2}$
     Update $m_j$ for the representative node of sub-tree $j$ : $m_j = m_j + \binom{n_i}{2}$
     Choose the representative node of sub-tree $i$ as the representative of the new sub-tree
     Update $n_i$: $n_i = n_i + n_j$
     $g_i = 0$
     For each other sub-tree $j$
       Do calculate $c_{ij}$. If $c_{ij} = 1$, then set $g_i = j$ (or $g_j = i$ if $j < i$)
```

Figure 2. A recursive algorithm for tree reconstruction from its generated quartet matrix.

Each pair of sub-trees is associated with a confidence value c_{ij}. To calculate the confidence value, we first assume that two sub-trees are connected together in the original tree and calculate the total number of quartets with a concerned form $(ab\,|\,cd, 1.0)$ where *a* is a leaf node from one sub-tree and *b* from the other, but *c* and *d* are leaf nodes not in either of these two sub-trees. The confidence value is then obtained by dividing the actual number accumulated directly from the quartet sets and stored in the matrix by this calculated value. If two sub-trees are truly connected together in the original tree, the corresponding confidence value for each pair of leaf nodes, one from each sub-tree must be equal to one. In the algorithm we use one additional variable g_i for representative node *i* to store index *j* when $c_{ij} = 1$. In each step we first try to find two sub-trees to merge by checking the variable g_i and then update the variables m_i and n_i in accordance with the merge; Next c_{ij}s are re-calculated and g_is updated for the next merge step. The process continues until all sequences are merged into a single tree.

2.2. Tree construction from inaccurate global weight matrices

In the previous subsection we discussed an algorithm for reconstructing 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 quartet weights is inaccurate and the algorithm for tree topology reconstruction discussed above 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 every leaf node pairs, to average them and then to use this averaged value as the confidence value $\bar{c}_{ij}$ for each pair of sub-trees.

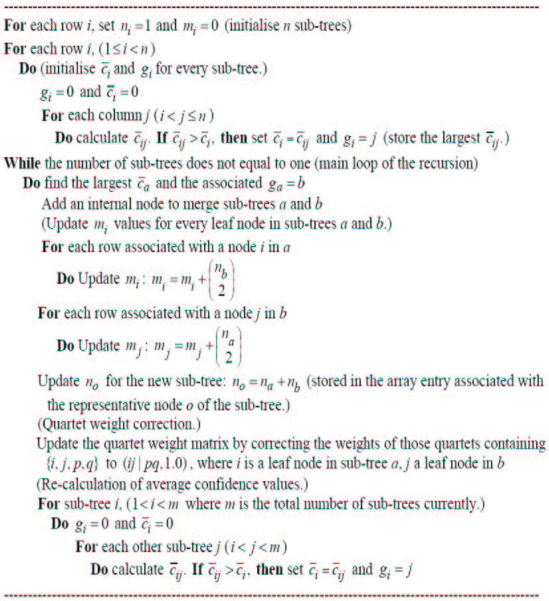
Since the entry values of the weight matrix are inaccurate, we may not obtain $\bar{c}_{ij} = 1$ for a pair of sub-trees during the computation. In addition to the three variables, namely, g_i, m_i and n_i, associated with each sub-tree, we need a new variable $\bar{c}_i$ to record the highest average 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 take an additional step to 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 the above two modifications we can have an algorithm which is able to deal with inaccurate quartet weights, as shown in Figure 3.

At each merge step the three major contributors to the total computational cost are: (1) the updating of m_i values, (2) the calculation of average confidence values for each sub-tree to find the highest one, and

(3) the quartet weight correction. The total costs for updating m_i values and for calculating average confidence values are $O(n^2)$ and $O(n^3)$, respectively. However, each quartet weight is corrected once and only once during the entire computation. The total number of quartets for a given set of n sequences is $\binom{n}{4}$ and obviously the total cost for quartet correction is $O(n^4)$. Therefore, the total computational cost for this 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 $(ij \mid pq, 1.0)$, where $i$ is a leaf node in sub-tree $a$, $j$ a leaf node in $b$
      (Re-calculation of average confidence values.)
      For sub-tree $i$, ($1 < i < m$ where $m$ is 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 $g_i = j$
```

Figure 3. 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 in the worst case when 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.

We used the benchmarks consisting of 48,000 synthetic data sets of DNA sequences developed by the LIRMM Methods and Algorithms in Bioinformatics research group (www.lirmm.fr) to test our quartet-based algorithm. The results show that our algorithm performs much better than many existing methods [22], i.e., the probability for the correct tree to be included in a small number of generated trees is very high.

3. A Further Extension

In our original algorithm described in the previous section multiple possible trees are constructed for a given problem and the number of trees can be limited by prefixing the total number of stages. One question is if we are able to further reduce the number of trees generated in the same number of stages while keeping the high probability for the correct tree to be included in these generated trees. Note that extra trees are only generated when we encounter a so-called critical point which is identified by using a fixed threshold α. There is one problem associated with the fixed threshold, that is, we may identify certain critical points which are actually unnecessary. When two sub-trees, say i and j, have a confidence value of 0.99, this value should be considered very large and the two sub-trees should be merged without any ambiguity. However, we may create a critical point here and need to generate extra two trees when the threshold α is set to 0.85, and either of these two sub-trees have a confidence value of above 0.84 with another sub-tree k. When unnecessary critical points are created and the limit for the number of generated trees is reached in the first a few merge steps, it may be possible for us to miss the true tree generation in later merge steps. One way to alleviate this problem is to make α a variable and vary in respect to confidence values, that is, α will be set higher when the confidence value is large. We adopt a simple linear function to meet this requirement and it is depicted graphically in Figure 4. In the figure threshold α is depicted as a function of confidence value *c*. When the confidence value is smaller than c_0, the threshold will remain as a constant α_0. When the confidence value is larger than c_0, the threshold will increase linearly with the increase of confidence value.

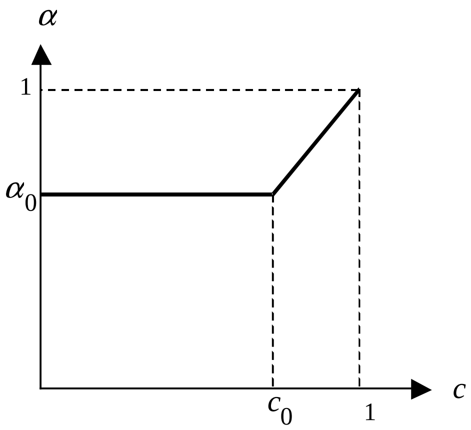


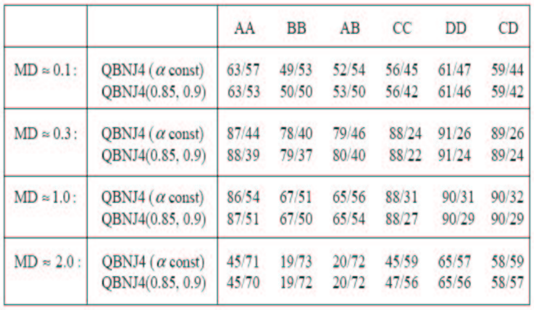
Figure 4. Threshold α as a function of confidence value c.

After a number of possible trees are generated, another important issue is how to find the correct tree from them in a posterior analysis. To address this issue we do the following: After a number of trees were constructed using our quartet-based algorithm, we calculate the likelihood values of these trees and then choose just a few best ones which have the highest likelihood values. Since the maximum likelihood is currently one of the most advanced methods for phylogeny reconstruction, we hope that the correct trees shall be identified easily. (Interestingly, our experimental results show that it is not always the case. A significant implication of this is that even if we are able to find a truly globally optimal tree under the maximum likelihood criterion, this tree may not necessarily be the correct phylogenetic tree!)

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 [17]. 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. A detailed description of these synthetic test data sets can be found on their Web site at www.lirmm.fr.

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



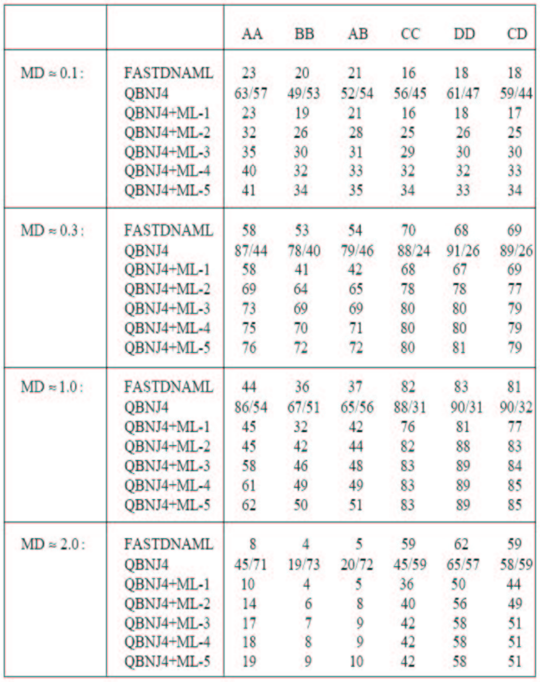
		AA	BB	AB	CC	DD	CD
MD ≈ 0.1 :	QBNJ4 (α const)	63/57	49/53	52/54	56/45	61/47	59/44
	QBNJ4(0.85, 0.9)	63/53	50/50	53/50	56/42	61/46	59/42
MD ≈ 0.3 :	QBNJ4 (α const)	87/44	78/40	79/46	88/24	91/26	89/26
	QBNJ4(0.85, 0.9)	88/39	79/37	80/40	88/22	91/24	89/24
MD ≈ 1.0 :	QBNJ4 (α const)	86/54	67/51	65/56	88/31	90/31	90/32
	QBNJ4(0.85, 0.9)	87/51	67/50	65/54	88/27	90/29	90/29
MD ≈ 2.0 :	QBNJ4 (α const)	45/71	19/73	20/72	45/59	65/57	58/59
	QBNJ4(0.85, 0.9)	45/70	19/72	20/72	47/56	65/56	58/57

Our first experiment is to compare the algorithm using variable thresholds to the one using a fixed threshold. In our experiment the number of stages is set to 4 (QBNJ4) to allow a maximum of 81 trees to be generated for each test data set. We set α to 0.85 in the fixed threshold algorithm which is called QBNJ4 (α const) and set α_0 to 0.85 and c_0 to 0.9 in the variable threshold algorithm which is called QBNJ4 (0.85, 0.9). Some experimental results are presented in Table 1 for DNA sequences of length 300. 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. From the table we can see that using the variable threshold we can indeed reduce the number of trees. It can be figured out easily from the table that the reduction is about 5.9 percent on the average. We can also see that using the variable threshold the percentage of correct trees included in the generated trees is not decreased, but rather increased a bit in certain cases. This result is consistent with our expectation.

In our second experiment we calculate the likelihood values of the generated trees using the relevant functions in TREE-PUZZLE and then choose just a few best ones which have the highest likelihood values. Some experimental results are presented in Table 2.

In Table 2 QBNJ4+ML-*j* denotes that the QBNJ method with 4 stages (using a fixed threshold) is used to construct multiple trees and then *j* best trees are chosen under the ML criterion, and the corresponding columns show the percentages of correct trees among these *j* best trees for six different model trees. The results obtained from FASTDNAML for the same test data set [17] are also included in the table for the purpose of comparison.

Table 2. Experimental results of using QBNJ4 followed by ML for sequences of length 300.



		AA	BB	AB	CC	DD	CD
MD ≈ 0.1:	FASTDNAML	23	20	21	16	18	18
	QBNJ4	63/57	49/53	52/54	56/45	61/47	59/44
	QBNJ4+ML-1	23	19	21	16	18	17
	QBNJ4+ML-2	32	26	28	25	26	25
	QBNJ4+ML-3	35	30	31	29	30	30
	QBNJ4+ML-4	40	32	33	32	32	33
	QBNJ4+ML-5	41	34	35	34	33	34
MD ≈ 0.3:	FASTDNAML	58	53	54	70	68	69
	QBNJ4	87/44	78/40	79/46	88/24	91/26	89/26
	QBNJ4+ML-1	58	41	42	68	67	69
	QBNJ4+ML-2	69	64	65	78	78	77
	QBNJ4+ML-3	73	69	69	80	80	79
	QBNJ4+ML-4	75	70	71	80	80	79
	QBNJ4+ML-5	76	72	72	80	81	79
MD ≈ 1.0:	FASTDNAML	44	36	37	82	83	81
	QBNJ4	86/54	67/51	65/56	88/31	90/31	90/32
	QBNJ4+ML-1	45	32	42	76	81	77
	QBNJ4+ML-2	45	42	44	82	88	83
	QBNJ4+ML-3	58	46	48	83	89	84
	QBNJ4+ML-4	61	49	49	83	89	85
	QBNJ4+ML-5	62	50	51	83	89	85
MD ≈ 2.0:	FASTDNAML	8	4	5	59	62	59
	QBNJ4	45/71	19/73	20/72	45/59	65/57	58/59
	QBNJ4+ML-1	10	4	5	36	50	44
	QBNJ4+ML-2	14	6	8	40	56	49
	QBNJ4+ML-3	17	7	9	42	58	51
	QBNJ4+ML-4	18	8	9	42	58	51
	QBNJ4+ML-5	19	9	10	42	58	51

We can see from Table 2 that when only a single tree with the largest ML value is selected (QBNJ+ML-1), the result on an average is not as good as (but very close to and some are better than) that obtained using FASTDNAML. When allowing the selection of more than one tree, our QBNJ4+ML-*j* performs much better than FASTDNAML in almost all the categories. We can also see from the table that the ML criterion is unable to identify all the correct trees among a very small number of trees generated by QBNJ4! Though the methods based on maximum likelihood are often found to outperform other methods, simple theoretical study showed that there can exist multiple optima for a given problem [19] and our experimental results confirm that certain incorrect trees can indeed have likelihood values at least as large as that of the correct tree.

5. Conclusions

In this paper we first discussed a revision to our recently developed quartet-based algorithm which constructs a limited number of trees for a given set of molecular sequences in phylogenetic analysis. In this revision we make the threshold a variable to try to reduce the construction of certain unnecessary trees for a given problem. Using the benchmarks consisting of 48,000 synthetic data sets of DNA sequences, we see that the reduction of the total number of tree reconstruction is about 5.9 percent on an average while keeping about the same probability for the correct tree to be included in the generated trees.

When trying to identify the correct tree in the generated trees using the ML criterion, we find it is common that certain incorrect trees can have likelihood values at least as large as that of the correct tree. This important finding suggests that the ML criterion alone may not be sufficiently enough to determine the true phylogeny for certain problems even if we are able to obtain a truly globally optimal tree! Therefore, it is very important in practice for us to use different criteria and construct multiple trees for posterior analyses in order to obtain more accurate and reliable results.

6. Acknowledgement

This research was partially funded by Discovery Grants (DP0557909) from the Australian Research Council.

References

[1] J. Adachi and M. Hasegawa, Instability of quartet analyses of molecular sequence data by the maximum likelihood method: the cetacean/artiodactyla relationships, *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] P. Buneman, The recovery of trees from measures of dissimilarity, in: Mathematics in Archaeological and Historical Sciences (F. R. Hobson, D. G. Kendal and P. Tautum, eds.) University Press, Edinburgh, 1971, pp. 387-395.

[8] A. W. M. Dress and D. H. Huson, Constructing splits graphs, *IEEE Trans on Computational Biology and Bioinformatics*, Vol. 1, no. 3, 2004, pp.109-115.

[9] 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.

[10] W. M. Fitch, A non-sequential method for constructing trees and hierarchical classifications, *J. Mol. Evol.* 18, 1981, pp. 30-37.

[11] S. Guindon and O. Gascuel, A simple, fast and accurate algorithm to estimate large phylogenies by maximum likelihood, Syst. Biol., 52, 2003, pp. 696-704.

[12] 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.

[13] D. H. Huson, Splitstree: A program for analyzing and visualizing evolutionary data, *Bioinformatics*, vol. 14, no. 10, 1998, pp. 68-73.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] M. Steel, The maximum likelihood point for phylogenetic tree is not unique, *Syst. Biol.*, Vol. 43, 1994, pp.560-564.

[20] 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.

[21] L. S. Vinh and A. von Haeseler, IQPNNI: moving fast through tree space and stopping in time, *Mol. Biol. Evol.*, Vol. 21, no. 8, 2004, pp. 1565-1571.

[22] B. B. Zhou, M. Tarawneh, C. Wang, D. Chu, A. Y. Zomaya and R. P. Brent, A novel quartet-based method for phylogenetic inference, *Proceedings of IEEE International Symposium on BIBE*, Minneapolis, Oct. 2005.

[23] B. B. Zhou, D. Chu, M. Tarawneh, P. Wang, C. Wang, A. Y. Zomaya and R. P. Brent, *Parallel implementation of a quartet-based algorithm for phylogenetic analysis*, Proceedings of the Fifth IEEE International Workshop on High Performance Computational Biology, Rhodes Island, Greece, April 2006.