Identifying character non-independence in phylogenetic data using parallelized rule induction from coverings

J. Leopold¹, A. Maglia², M. Thakur¹, B. Patel¹ & F. Ercal¹
¹Department of Computer Science, University of Missouri-Rolla, USA
²Department of Biological Sciences, University of Missouri-Rolla, USA

Abstract

Undiscovered relationships in a data set may confound analyses, particularly those that assume data independence. Such problems occur when characters used for phylogenetic analyses are not independent of one another. Although a data mining technique known as rule induction from coverings has earlier been shown to be a promising approach for identifying such non-independence, its inherent computational complexity has limited its application for large phylogenetic data sets. Herein we present a parallelized implementation of the rule induction from coverings strategy which overcomes some of these limitations. We also discuss two heuristics that have been applied to the algorithm to further improve its efficiency.

Keywords: data mining, phylogenetics, parallelization.

1 Introduction

Some types of data analyses require and/or assume independence between items in the data set. If this assumption is violated, the results of the analysis may be incorrect. For example, such a problem can occur in phylogenetic analyses (i.e., the reconstruction of evolutionary interrelationships between biological species). A phylogenetic data set consists of rows representing different taxa and columns representing characters or attributes of the taxa. Phylogenetic inference methods such as maximum likelihood and parsimony are based on the assumption that each character in the data set serves as an independent hypothesis of evolution [1, 2]. If this assumption is not true, then correlated or non-independent characters can effectively be overweighted in analyses [3], and the resulting
phylogeny may not necessarily reflect the true evolutionary history. Thus, character non-independence should be identified before conducting phylogenetic analyses.

Characters (i.e., attributes) can be non-independent in several ways—i.e., one attribute can depend upon another, attributes can co-depend on one another, or attributes can be correlated, wherein they do not depend on one another, but share a set of dependencies with other characteristics. In phylogenetics, characters that reflect homology share a set of dependencies that reflect the true evolutionary history of the group. This sort of dependency is referred to as phylogenetic autocorrelation. It is the basis of all modern methods of phylogenetic analysis and results from synapomorphy. However, if in a phylogenetic data set a set of non-independent characters reflects homoplasy their presence may lead to an incorrect reconstruction of evolutionary history.

In [4], the authors discussed the application of three data mining techniques for uncovering non-independence in phylogenetic data sets. Of these methods, rule induction from coverings was found to be the most promising. However, the computational complexity of that method made it prohibitive to run on even moderately sized data sets (i.e., more than about 20 attributes).

Herein we present a parallelized implementation of the rule induction from coverings strategy as applied to the problem of identifying character non-independence in phylogenetic data. We also discuss heuristics we have applied to the algorithm to further improve its efficiency.

2 Related work

Many systematists (e.g., [5, 6]) recognize that character dependence in phylogenetic data sets presents problems in reconstructions, but few quantitative attempts have been made to identify non-independence of phylogenetic characters. Of those methods available, all examine phylogenetic independence/autocorrelation of characters after phylogenetic analyses are conducted (e.g., [7–10]). A model phylogeny is required to conduct tests of independence. However, an assumption of independence is required to generate a model phylogeny. Clearly a better approach is to test for character non-independence before conducting phylogenetic analyses.

One attempt to develop a “pre-analysis” approach for measuring phylogenetic non-independence was presented in [11]. In this method, suites of correlated characters are identified using character compatibility. An association matrix between characters in a data set is calculated, and eigenvector analyses are conducted on values in the matrix. Using this method, the authors were able to identify characters with similar patterns of compatibility, and identified suites of characters that were more correlated with one another than expected by chance. But the results were limited because from them, one could not identify dependency or co-dependency.

In [4] the authors examined three data mining techniques to address the phylogenetic character non-independence problem: 1) Bayesian networks, 2) decision tree induction, and 3) rule induction from coverings. Bayesian networks were found to be of limited usefulness because of requirements on the ordering
of characters and the necessity of prior knowledge of relationships. Hence, the patterns of dependencies inherent in phylogenetic data sets were not immediately obvious through Bayesian analysis.

The results of decision tree induction provided a more readily understandable pattern of dependencies in phylogenetic data sets. The reported rules give clear statements of dependency relationships from one attribute to the next. However, because there is the possibility of reporting false dependency relationships, or missing some dependency relationships altogether, the usefulness of this method was also limited.

In contrast to decision tree induction, all rules produced by rule induction from coverings are “perfect,” and can be applied to new data with complete certainty. Furthermore, the results from this method identify all dependency relationships, and the rules produced can be used to further examine the nature of those relationships.

3 Rule induction from coverings

RICO (Rule Induction from Coverings) is an implementation of an algorithm given in Ref. [13] for finding all possible coverings for a given data set. For this covering algorithm, if S is a set of attributes and R is a set of decision attributes, a covering P of R in S can be found if the following three conditions are satisfied:

(i) P is a subset of S.
(ii) R depends on P (i.e., P determines R). That is, if a pair of entities x and y cannot be distinguished by means of attributes from P, then x and y also cannot be distinguished by means of attributes from R. If this is true, then entities x and y are said to be indiscernible by P (and, hence, R), denoted x ~ P y. An indiscernibility relation ~P is such a partition over all entities in the data set.
(iii) P is minimal.

Once a covering is found, it is a straightforward process to induce rules from it. Although any single covering may be a basis for computing a rule set that describes the entire data set, it can be even more useful to identify all possible coverings. Finding all coverings can be computationally expensive because, in theory, each possible subset of attributes must be tested as a potential covering (unless that subset is a superset of a covering that has already been identified). For a data set of k attributes, there are 2^k different subsets. In a morphological data set, this may be 50–80 characters, but in a typical molecular data set this may be closer to 2,000 characters. For phylogenetic data sets, some constraints can be applied to the covering algorithm to reduce the execution time. For example, the cardinality of the candidate subsets could be limited to a small number (e.g., ≤ 5) because it would likely be difficult to conceptualize larger combinations of characters to determine the state of the character of interest. Furthermore, it is reasonable to limit the number of rules reported to only those that cover a certain number of entities in the data set.
4 Modified \textit{RICO} data structures and algorithms

Although rule induction from coverings appeared to be a promising strategy for addressing the phylogenetic character non-independence problem, the computational complexity of the approach made it prohibitive to run on most phylogenetic data sets. To overcome these scalability and efficiency issues, we developed a parallelized implementation called \textit{PRICO}.

4.1 Set representation

The input data set is an \( m \times n \) matrix \( M \) where the rows represent the different taxa and the columns represent the attributes. In a phylogenetic data set, typically there are approximately 20 rows and anywhere from 50 to 2,000 columns. Let \( A = \{a_1, a_2, \ldots, a_n\} \) be the set of attributes. For a particular decision attribute \( y \), we represent a set \( X = \{x_1, x_2, \ldots, x_k\} \), where \( 1 \leq k < n \), that determines \( y \) as a tuple \((X, y)\). Each \( x_i \) \( (1 \leq i \leq k) \) and \( y \) must be distinct elements from \( A \). A binary representation was used for a set of attributes, whereby the \( i^{\text{th}} \) bit \( (1 \leq i \leq n) \) is 1 if attribute \( a_i \) from the set \( A \) is included; otherwise, the \( i^{\text{th}} \) bit is 0.

4.2 Finding all coverings

To find all coverings the \textit{PRICO} approach (which is outlined in \textit{Algorithm 1}) makes repeated passes, generating candidate sets as tuples, and checking each such set to see if it is a legitimate covering for the decision attribute currently being considered. The primary difference from the \textit{RICO} implementation is the manner in which sets are generated and the way that sets are validated as legitimate coverings.

\textit{Algorithm 1. Find all coverings for decision attribute} \( y \)
\begin{algorithmic}
   \STATE Input: \( m \times n \) matrix \( M \), and decision attribute \( y \)
   \STATE 1. Generate a candidate tuple \((X, y)\);
   \STATE 2. Check whether \( X \) determines \( y \);
   \STATE 3. if \( X \) determines \( y \) then
   \STATE 4. \hspace{1em} if \( X \) is minimal then
   \STATE 5. \hspace{2em} add \( X \) to the set of coverings for \( y \);
   \STATE 6. \hspace{1em} end-if;
   \STATE 7. \hspace{1em} end-if;
   \STATE 8. Repeat the above steps until all possible sets \( X \) for determining \( y \) have been checked.
\end{algorithmic}

\textit{Algorithm 2} was utilized to test whether a set of attributes \( X \) determines an attribute \( y \). Currently, we are only considering coverings of size at most 5 because our end-users have stated that is difficult to conceptualize larger combinations of characters. Thus, the complexity of \textit{Algorithm 2} is simply \( O(m) \).

\textit{Algorithm 2. Test whether} \( X \) \textit{determines} \( y \)
\begin{algorithmic}
   \STATE Input: \( m \times n \) matrix \( M \), and tuple \((X, y)\)
\end{algorithmic}
1. string s[ size_of(X) ];
2. hashtable h;
3. for i = 1 to m do
4.     for j = 1 to size_of(X) do
5.         s[ j ] = value of x_j in row i of M;
6.     end-for;
7.     int pos = h.find( s );
8.     if (pos ≠ h.end( )) then
9.         if (h[pos].key( ) ≠ value of y in row i of M)
10.            then return( false ); // X does not determine y
11.         end-if;
12.     else h.add( s );
13.     end-if;
14. end-for;
15. return( true ). // X does determine y

Step 4 of Algorithm 1 requires checking to see whether a set of attributes X that determines y is minimal. This procedure is outlined in Algorithm 3. The input is a set X (in binary representation) which has been found to determine an attribute y. It will be checked against the set C of all coverings (of size less than or equal to the size of X) for y that have been determined thus far to see whether X is minimal. If this algorithm returns true, then X will be added to the set C of coverings for y.

**Algorithm 3. Test whether the set of attributes X is minimal**

Input: set of attributes X that determines y, and the set C of coverings for y (that have been determined thus far)
1. for each element c_i in C do
2.     if ( ( X & c_i ) == c_i ) then
3.         return( false ); // X is a subset of another set in C, so X isn’t minimal
4.     end-if;
5. end-for;
6. return( true ). // X is minimal

The complexity of Algorithm 3 is dependent upon the size of the set of coverings for a particular decision attribute. For a data set of n attributes where we are interested in coverings of size no greater than 5, the number of distinct sets that could possibly be tested as coverings for a particular decision attribute is proportional to the sum of the number of combinations of size r from a set of size n-1 where r = 1..5, which is O(n).

4.3 Criteria for tuple generation and validation strategies

Despite limiting the size of the coverings, we determined that any efficient strategy for generating sets to be subsequently checked as valid coverings should satisfy the following two criteria:
Criterion 1: Let $G = (V, E)$ be a directed acyclic graph where, for each tuple $(X, y)$ to be generated as a potential covering for decision attribute $y$, there is a vertex $v = X$ in $V$. There exists an edge $e = (v_1, v_2)$ in $E$ for vertices $v_1$ and $v_2$ in $V$ iff $v_1$ is a subset of $v_2$. Then a tuple is only generated (and subsequently checked to see if it is a valid covering) iff all of its ancestors in $G$ have been generated. That is, do not generate a set of attributes $X$ as a possible covering for a decision attribute $y$ until all subsets of $X$ have been generated.

Criterion 2: Using the graph definition of Criterion 1, a tuple $(X, y)$ that corresponds to a vertex $v$ in $V$ that has $k$ children should be checked as a valid covering for $y$ before any tuple corresponding to another vertex $w$ in $V$ that has fewer than $k$ children. This will effectively check smaller sized sets before larger sized sets.

These criteria ensure that all generated, and subsequently validated, coverings are indeed minimal (as per condition (iii) of the definition of a covering given in section 3).

4.4 Tuple generation

As mentioned earlier, for a tuple $(X, y)$ that is to be checked to see if $X$ is a valid covering for attribute $y$, we use a binary representation for $X$. One strategy for generating tuples to be checked is to do so in ascending order of the binary representation for each set. For example, suppose that $A = \{1, 2, 3, 4\}$ and that the decision attribute is 3. Then this strategy would generate $\{1\}, \{2\}, \{1, 2\}, \{4\}, \{1, 4\}, \{2, 4\}, \{1, 2, 4\}$ to be checked as coverings for attribute 3 (in that order) since this corresponds to the binary sequence 0001, 0010, 0011, 1000, 1001, 1010, 1011. However, using the graph representation defined in Criterion 1, for two vertices $v_1$ and $v_2$ in the graph that represents possible coverings for some particular decision attribute, if there exists an edge $v_1 \rightarrow v_2$, then it must be the case that $\text{bin}(v_1) < \text{bin}(v_2)$, where $\text{bin}$ is the binary representation of a set. But the vertex representing $\{4\}$ (i.e., 1000) will have more children than the vertex representing $\{1, 2\}$ (i.e., 0011), so, according to Criterion 2, $\{4\}$ should have been checked before $\{1, 2\}$. That is not the order in which this strategy will generate and test those sets; thus, this strategy violates Criterion 2.

An alternate strategy is to generate (and test) sets of attributes in canonical order. Again suppose that $A = \{1, 2, 3, 4\}$ and the decision attribute is 3. Then this strategy would generate $\{1\}, \{2\}, \{4\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{1, 2, 4\}$ to be checked as coverings (in that order), which corresponds to the binary sequence 0001, 0010, 1000, 0011, 1001, 1010, 1011. Using the graph representation defined in Criterion 1, if there exists an edge $v_1 \rightarrow v_2$, then it must be the case that the number of 1’s in $v_2$ is greater than the number of 1’s in $v_1$. It will also be the case that $v_1$ will have more children than $v_2$. Therefore, since this strategy would generate and test $v_1$ before $v_2$, both Criterion 1 and Criterion 2 would be satisfied.
By utilizing this strategy for generating tuples to be checked as coverings for a particular decision attribute, sequential execution of Algorithm 1 requires time proportional to $O(n^2)$ for each of n decision attributes.

### 4.5 Parallelization of the algorithm

The most important aspect of any parallel implementation is to minimize the communication among the processors. In the context of the covering problem, there are two basic strategies that could be employed for distributing the workload. One approach is to distribute the workload for finding the coverings for one particular decision attribute before proceeding with finding the coverings for a different decision attribute. However, any such approach that would satisfy Criterion 1 and Criterion 2 would require considerable inter-processor communication to utilize the graph properties defined for those criteria.

A different approach for distributing the processing takes advantage of the fact that generation and testing of a tuple $(X, y_1)$ is completely independent of the processing for a tuple $(X, y_2)$ where $y_1 \neq y_2$. Thus, each processor could be responsible for determining the coverings for a different decision attribute, a strategy that could easily satisfy both Criterion 1 and Criterion 2. This is the parallelization strategy that we utilized in PRICO.

A parallel algorithm is considered to be optimal if the product of the number of steps and number of processors is of the same order as the best sequential algorithm. If executed sequentially, Algorithm 1 has complexity $O(m \cdot n)$ for determining the coverings for each of n decision attributes, for a total time of $O(m \cdot n^2)$. Let $p$ be the number of processors. If each processor utilizes Algorithm 1 (which is $O(m \cdot n)$ for one decision attribute) to find the coverings for $n/p$ decision attributes, then the total time required will still be $O(m \cdot n^2)$.

### 4.6 Statistical analysis of the processing

There are two primary stages of processing that a tuple goes through in Algorithm 1. Stage 1 occurs at steps 1 and 2 where a tuple $(X, y)$ is generated and tested to see if the set of attributes $X$ determines the attribute $y$. Stage 2 occurs at step 4 of Algorithm 1 where $X$ is tested for minimality against the set of coverings for $y$ found thus far. Another way to think of this is that the set of all tuples found at Stage 1 are effectively filtered at Stage 2 to produce the set of all coverings for a decision attribute $y$; that is, a set of attributes that determines $y$ is filtered out in Stage 2 if it is not minimal, and, hence, not a covering for $y$.

We ran PRICO on a hardware cluster that has 40 compute nodes, a single controlling node, and a single compilation node. The number of nodes available for parallel use is 32, each of which has two processors; thus, we utilized a total of 64 processors. Executing PRICO using this hardware configuration for a data set containing 20 rows and 73 attributes, we found that the average filter ratio for Stage 1 increases as the size of the sets of attributes being considered increases. This is to be expected since, as we increase the size the sets can be, we increase the number of candidate sets that will be considered at this stage. Then the probability that those larger sets will be filtered out at Stage 1 (i.e., because the candidate set does not actually determine the decision attribute) increases.
In contrast, the average filter ratio for Stage 2 decreases as the size of the sets being tested for minimality increases. Recall that the strategy employed at Stage 2 filters out sets according to Criterion 2, whereby smaller sized sets will be checked before larger sized sets. As the size of the set being considered increases, the probability that it will not be found to be minimal (and, thus, will be filtered out) increases.

We also found that the average table utilization ratio for Stage 1 increases as the size of the sets being considered increases. This is expected since, as the size of the set increases, the more lookups must be made in the data set to conclude whether the set of attributes actually determines the decision attribute.

In contrast, the average table utilization ratio for Stage 2 decreases as the size of the sets being tested for minimality increases. This is again due to the fact that the Stage 2 strategy considers smaller sets before larger sets. Clearly, there will be more comparisons (i.e., “lookups”) for smaller sets against the set of coverings found thus far than there will be when Stage 2 examines larger sets, which will more quickly be eliminated as not being minimal. Similar results were found for the average filter ratios and table utilization ratios when PRICO was executed on a much larger molecular data set (81 rows, 3819 columns).

5 Heuristics

Two heuristics were implemented to further improve the performance of PRICO – specifically, to decrease the table utilization for Stage 1. Both heuristics consider a reordering of rows in the data set prior to running PRICO, with the objective of more quickly rejecting a set of attributes as being a determining set for a particular decision attribute. This would reduce the number of lookups required, which, in turn, would reduce the table utilization factor.

5.1 Different-on-top

The objective of the first heuristic that we applied was to rearrange the rows in the data set such that the rows that differed the most from other rows (considering their respective attribute values) would be at the beginning, or the “top”, of the data matrix. These rows, which would be examined early, and would be more likely to produce smaller determining sets for a decision attribute, which, in turn, would result in fewer lookups at Stage 1 of Algorithm 1. To quantify the “difference” between two rows in the data set, we utilize a measure that computes the Hamming distance between row i and the other rows. Rows in the data set were then reordered in descending order of their distance value prior to running PRICO.

5.2 Slightly-different-on-top

The second heuristic that we applied was a slight variation of the “different-on-top” reordering of rows. For a “slightly-different-on-top” reordering we compared each row i with each other row in the data set based on the absolute value of the Hamming distance between the two rows minus the size of the
determining set being considered. When PRICO was executed with this heuristic on a data set of 20 rows and 73 columns we found that the “slightly-different-on-top” heuristic improved table utilization somewhat compared to the “different-on-top” heuristic. The results were similar for a much larger molecular data set.

6 Utility for phylogenetic analyses

In typical practice, if characters in a phylogenetic analysis are thought to be non-independent, all but one of the non-independent characters is deleted (or all are combined into a single “character suite”), or a weighting scheme is invoked that results in each character having less importance in the analysis. The results of PRICO analyses should be used only as a way to identify possible character non-independence problems in phylogenetic data sets, and data should not be altered based only on PRICO results. PRICO, as with all data mining techniques, recognizes patterns in data, and it is possible that some of those patterns reflect homology (= true evolutionary history). Thus, data should not be removed until it has been determined by the systematist that the patterns of non-independence reflect homoplasy (= parallelism or convergence).

The best possible scenario for utilizing PRICO is to: (1) run the PRICO analysis before any phylogenetic analysis and make note of suites of non-independent characters; (2) run the phylogenetic analysis and plot the distribution of characters on the resulting tree; and (3) compare the PRICO results to the resulting phylogeny. If there is a high degree of homoplasy (e.g., reversals, parallelisms) in the phylogenetic analysis, particularly at nodes where characters identified by PRICO show support, then it is likely that the dependency relationships identified by PRICO are not the result of homology, and removing or reweighing them is warranted. We plan to continue this research by examining several existing phylogenetic data sets (including those of experimentally-manipulated “known” phylogenies) to determine how prevalent homoplasious non-independence is, and what effect removal/reweighing the data will have on resulting phylogenetic reconstructions.

7 Summary

Because of the fairly large size of most phylogenetic data sets, applying data mining techniques to identify non-independence has been an intractable problem. However, our parallel implementation of rule induction from coverings has overcome at least some of these computational limitations. We next look forward to examining the implications of identifying non-independence for phylogenetic analyses, as well as possibly applying PRICO to other problem domains.

References


