The Difficulty In Computing Ancestral DNA Sequences: Using Computational Analysis To Reconstruct DNA Sequences

Zhentao Li[a] and Mathieu Blanchette[b]

[a]School of Computer Science, McGill University, 3480 University St., Montreal, Quebec, Canada, H3A 2A7.
[b]McGill Centre for Bioinformatics, 3775 University St., Rm. 332, Montreal, Quebec, Canda, H3A 2B4.
Correspondence should be addressed to Zhentao Li: zhentao.li@mail.mcgill.ca

Intriguing work has been carried out in order to decipher the genetic codes of today's existing species. However, little is known about the genetic makeup of species that existed long ago. Exciting possibilities have recently been raised in the field of computational analysis (1), proposing that reconstruction of ancestral DNA sequences can be performed if the DNA sequences of the existing species are known. Being able to perform such reconstructions would simplify the study of the evolution of these species, and uncover many mysteries regarding life that once existed on this planet.

In order to perform reconstructions of unknown ancestral DNA sequences, many different types of problems must be solved, all of which can be approached computationally. Examples of such problems include building a phylogenetic tree of the evolutionary line in question, determining a multiple alignment of the existing species being analyzed, or working out the actual identity of the nucleotides within the ancestral sequence. The problem presented in this paper considers the level of modification within the ancestral sequence.

Key Terms

Phylogenetic tree:
a tree-like diagram demonstrating the relationship between ancestral species and contemporary species

Multiple alignment:
an alignment of DNA sequences, whereby homologous positions are lined up with respect to one another

Deletion:
takes all characters (0 or 1) in a contiguous region, and transforms them into 0's (gaps). For example, '10101' becomes '00000'

Insertion:
takes a contiguous region of 0's (gaps) and transforms some of the 0's into 1's (nucleotides). For example, '00000' becomes '10101'

Instance:
a specific case or an occurrence of a problem. For example '3 +4 = ?' is an instance of the addition problem 'a+b=?'

Problem reduction:
turning all instances of one problem into instances of another problem

Reducing:
showing that a problem can be solved by using the solution to another problem

Considering the diversity of the problems at hand, one must realize that not all of these computational problems can be solved. In other words, the complexity of certain problems is such that it would currently take a computer an unrealistic amount of time to solve these types of problems. In such situations, these problems are deemed 'hard'.

The following analysis will provide a detailed computational definition of the problem in question. The aim is to define the components of the analyzed problem, and will not provide any solution to the problem. This definition is succeeded by the problem reduction, which will assess the hardness of the problem at hand. In fact, it will be shown that the problem is 'hard', and is impractical to solve. This is achieved by reducing a standard computational problem known to be hard to the problem in question.

Problem Definition

In order to attempt to solve the problem in question, input data are necessary, which include the sequences of the existing species, the phylogenetic tree for this evolutionary line, and the multiple alignment of the present-day species' sequences (fig. 1, a-c). The phylogenetic tree is necessary to give an idea of how the species are related to one another. Existing species occupy the outside positions of the tree, termed the 'leaves', while the ancestral species occupy the internal positions of the tree, termed the 'nodes' (fig. 1, a). The multiple alignment is used to analyze a specific stretch of DNA within each species being analyzed and to compare the differences and similarities between the stretches. Once

aligned, the sequences are then transformed into an arbitrary code of 1's and 0's, based on the presence or the absence of a nucleotide at each position (fig. 1, d). If a nucleotide exists at a certain position, it is transformed into a 1. Conversely, if no nucleotide (i.e. a gap) exists at a certain position, it is transformed into a 0. The task is then to find the most probable sequences of 0's and 1's at the internal nodes, which represent the sequences of the ancestral species. Such a set of sequences is called a solution. In this model, the most probable sets of sequences are those that will minimize the total number of sequence changes in the tree. A change can manifest itself as either a deletion or an insertion. There is an extra restriction, which is that a previously deleted nucleotide within the timeline cannot be re-inserted at a later time into a specific sequence within the tree. A solution which minimizes the number of insertions and deletions is called an optimal solution. An example of a solution is shown in fig. 2.

Problem Reduction

Now that an example of the problem has been shown, the problem reduction will prove that it cannot realistically be solved. In other words, this problem is deemed 'hard'. As such, there is no general way of finding a solution to the type of problem described previously in the problem definition. Consider the problem in question to be named ADR, for 'ancestral DNA reconstruction'. The way in which this problem is shown to be hard is by proving that it is at least as difficult to solve as a "benchmark" problem, named NAE-3SAT. NAE-3SAT, or 'not all equal 3 satisfiability', is a problem that is known to be computationally difficult in the field of computer science. The way in which this is proven is to show that any instance of NAE-3SAT can be reduced to ADR.

NAE-3SAT Problem Definition

An instance of the NAE-3SAT problem consists of variables, literals and clauses. The variables (named X1, X2, etc) can be assigned a value of either 'true' or 'false'. A literal is either a variable (e.g.:X3) or the negation of a variable (e.g.:$\bar{X}$1). A clause is a set of three literals.

In an NAE-3SAT problem, the goal is to assign a value (true or false) to each of the variables that satisfy all the clauses in the problem. A clause is satisfied if at least one literal is assigned as 'true' and at least one literal is assigned as 'false'. For a given assignment of the variables, the literal Xi is assigned the same value as the variable Xi and the literal $\bar{X}$i is assigned the opposite value. An assignment which satisfies all the clauses is called a satisfying assignment. A solution to an instance of NAE-3SAT is a satisfying assignment.

The proof

The following is a simplified version of the proof. This is the general scheme used to reduce a problem.

Assume that there is a way to solve ADR, without performing an unrealistically large amount of computations. NAE-3SAT can then be solved computationally in the following manner:

1. Given an instance of an NAE-3SAT problem, construct an appropriately chosen instance of ADR. In other words, determine the input that one wishes to analyze.
2. Solve this instance of ADR, which consists of inputting the data that was determined in (1). This is possible, as it was previously assumed that there was a way of solving ADR.
3. Obtain the solution that is found for the ADR problem and translate it into a solution to NAE-3SAT.

If it is determined that ADR can be solved once these computational analyses have been performed, then NAE-3SAT can also be solved. This would contradict the hardness of NAE-3SAT. Therefore, the initial assumption that we could solve ADR quickly must have been false. The whole proof now relies on constructing the so-called 'appropriate chosen instance' of ADR given an instance of NAE-3SAT. The rest of the proof is a description and explanation of this construction. Recall that a phylogenetic tree must be given in an instance of ADR. In this construction, it will have the shape shown in fig. 2, b.

If there is a satisfying assignment to the given instance of NAE-3SAT, it must somehow be found. In this construction, this shall be obtained from the sequence found at the S node of the phylogenetic tree shown in fig. 3, b. On the other hand, if there is no satisfying assignment to the instance of NAE-3SAT, the construction will force the total number of operations (i.e. insertions or deletions) in any optimal solution to be greater than some fixed number.

In preparation for step 3, any optimal solution for the sequence in the S node will be forced to have a certain

structure, allowing it to be translated in such a way that it becomes a solution to NAE-3SAT.

Initially, some of the positions of the sequence within the S node are forced to be 1 in any optimal solution. This is done by setting some of the leaves to specific strings. In so doing, other solutions will be forced to have more operations. This will be useful since if both the sequences within the S node and within a given leaf have a 1 at any given position, no insertion or deletion can go through this position (recall that a nucleotide cannot be re-inserted once deleted).

A region of contiguous positions in the S node will be reserved for each variable in this particular instance of NAE-3SAT. Depending on what is in this region of the S node sequence, one will know what to assign the variables in the NAE-3SAT problem (if a satisfying assignment exists). In fact, the idea is to force each region to only have two possible values in any optimal solution. Again, this is achieved by forcing other solutions to have more operations. Depending on which of the two values is assigned, it will become clear as to what value (true or false) is to be assigned to each variable.

The only remaining task is to ensure that the solution obtained will satisfy each clause. A different set of leaves can be used for each clause. Such a task is not as simple as it first seems. However, one easy aspect of this operation is to construct leaves in which the number of literals in a clause assigned 'true' can be counted. In this particular sense, the term counted refers to the fact that the number of operations needed is proportional to the number of literals set to true. But more than just counting is needed.

It is necessary to force any optimal solution to have at least one literal set to true and at least one literal set to false in any clause, if there is indeed a satisfying assignment. To do so, strings are designed at the leaves which can count the number of literals set to true, two at a time. Along with other such tricks, like taking combinations of different strings, this will guarantee to find a satisfying assignment, if it exists.

Conclusion

The most immediate implication of the hardness of the ADR problem is that large instances of the problem (ex.: 1,000,000 nucleotides) are most likely not going to be solved in a rapid manner. This is problematic in practice as the large problems, such as reconstructing whole genes, are those that generate the most interest. However, the findings described above do not render this reconstruction hopeless, as other smaller instances of the problem may still be solved. It would also be possible to obtain sub-optimal solutions for these larger instances, though accuracy would be lost in such a reconstruction. Though many obstacles remain towards achieving the goal of ancestral genome reconstruction, computational analysis is still a powerful tool by which such reconstructions can be performed.

Special thanks to Leonid Chindelevitch for his contributions to this project.

References

1. Mathieu Blanchette, Eric D. Green, Webb Miller, and David Haussler. Reconstructing large regions of an ancestral mammalian genome in silico. Genome Res., 14(12):**2412-2423**, December 2004.

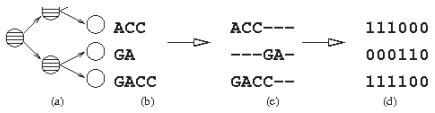

Figure 1. Example of information input. Clear circles represent the existing species, termed the leaves of the tree. Dashed circles represent the ancestral species, termed the internal nodes of the tree. (a) A phylogenetic tree. (b) DNA sequences of the existing species. (c) Multiple alignment for sequences found in (b). A '-' represents a gap. (d) Conversion of the nucleotide input into an arbitrary code of 1's and 0's.

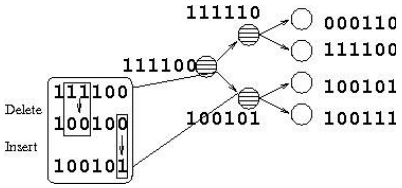

Figure 2. Example of a solution. The highlighted branch, denoted by a box, shows an insertion and a deletion.

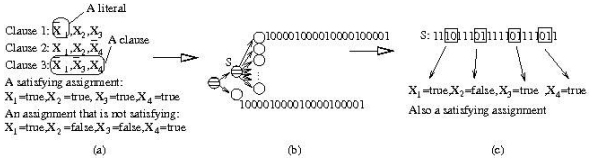

Figure 3. (a) Example of an instance of NAE-3SAT along with a satisfying assignment. The second assignment is not satisfying since all literals within clause 1 are set to false. (b) An instance of ADR is constructed and then solved. (c) The solution is analyzed, looking specifically at the string within the S .node. From this, a satisfying assignment for NAE-3SAT can be found