FPT Algorithms for Perfect Phylogeny

Steven Lonergan

April 15, 2010

1 Introduction

Biologists have long been help back by the limits of computational power. This is because often times the most important questions in computational biology require either exponential algorithms to solve, or algorithms that are polynomial but have such massive input sizes that their running times are long. This posses a problem because not only does it mean the Biologist has to wait for hours (or days) before a result can be known, but if something was entered incorrectly the computation must be restarted from scratch.

Computer Scientists enter into the equation as ambassadors of speed. We obsess over trying to make our algorithms run faster and faster until we are able to prove that no more speed up can possible happen. This is why the marriage between Computer Science and Biology is so beautiful, we are both able to enlighten each other things things we might miss. Biologists give problems a new spin, allowing it to be seen in a whole new way, while a computer scientist will try and solve that problem as quick as possible.

In this report I am concerned with the speed of one Computational Biology problem. Building phylogenetic trees is something that Biologists have to do on a regular basis, and more importantly they often deal with the problem that the set of characteristics they wish to build the tree off of has several conflicts. The question then becomes one of conflict resolution to build the Perfect tree loosing the minimal amount of information.

The problem, called COMPATIBILITY, is NP-Complete meaning no polynomial time algorithm is know to solve it. Furthermore when we have a large set of characteristics it becomes increasingly difficult to decide which ones to remove, and if those that you did remove were in fact the minimal.

2 Problem Definition

Here we are concerned with two different problems. The first of which we covered in class and is concerned with building a perfect phylogenetic tree.

Definition 1. *(Building A Perfect Phylogeny): Given a character state matrix M, build a tree such that no conflict exists in the tree.*

This problem, as we saw in class, is computationally easy. We are able to detect any conflicts by comparing each column of the character matrix. A perfect tree is constructible if the following lemma holds.

Definition 2. *(Definition 6.1 [4]: For each column j of M, Let O_j be the set of objects whose state is 1 for j. Let $\bar{O}_j$ be the set of objects whose state is 0 for j.*

Lemma 1. *(Lemma 6.1 [4]: A binary matrix M admits a perfect phylogeny if and only if for each pair of characters i and j the sets O_i and O_j are disjoint, or one of them contains the other.*

Slide 13 of lecture 14 shows the that problem is in fact easy and can be solved in $O(MN)$ time. Finally the problem that we concern ourself with for the rest of the paper is that of COMPATIBILITY.

Definition 3. *(COMPATIBILITY):*
INSTANCE: *A character state matrix M with n objects and m directed binary characters, and a positive integer $B \leq m$.*
QUESTION: *Is there a subset L of the characters that satisfies Lemma 1 with $|L| \geq B$?*

3 NP-Complete

In this section we present a fast review of NP-Complete theory to allow the reader to understand the methods used later. Showing a problem is NP-Complete is an important area of computer science research since it allows us to categorize problems in terms of their running time and allows someone wishing to solve a problem insight on how difficult their task might be. Because they are not solvable in P-Time they are generally considered 'hard' problems and often require tremendous amounts of computational time to solve.

In order to show that a problem is NP-Complete we need to show two things:

1. Membership in NP
2. Reduction from a known NP-Complete problem

3.1 Membership In NP

To show that a problem is a member of NP we need to show that a solution for the problem can be checked in P-time. Formally it is defined as follows:

Definition 4. *(Membership In NP): A problem P can be shown to be in NP by showing a possible solution to problem be can be checked in no more then a polynomial amount of time.*

Note 1. *It is important to note here that just because we are able to check the solution to a problem in P-Time does not mean we can solve the problem in P-Time.*

3.1.1 An Example

Consider the problem of SAT. In SAT we want to decide on an assignment of variables to a expression such that the expression evaluates to TRUE. This is a known NPC problem [1]]. Consider trying to show that it was a member of NP, therefor we are asking to find a P-time verifier to a solution.

Consider walking down the street and someone hands you a piece of paper with an expression written on it and a list of variables and their assignments (such as $x_0 = 1, x_1 = 0, ..., x_n = 1$). It is easy to see how you can check to see if the solution is valid, simply trace through the expression replacing the variables with their assignments and then go back through and check to see if becomes true. This can be done naively in two passes of the data allowing us to check the solution in at most n^2 time which is within our P-Time bound.

3.2 Reductions

Formally for a problem in NP to be considered NP-Complete we need definition 4 to hold.

Definition 5. *(NP-Complete): A problem p is said to be NP-Complete if:*

1. $P \in NP$
2. $\forall Q \in NP$ *we have a polynomial time reduction from Q to P*

Statement 1 is assumed and membership can be shown as in section 3.1. Turning our attention to statement 2 we have to consider a reduction from all problems in NP to our target problem within a polynomial amount of time. Although this appears to be a daunting task at first we are able to use a different method that accomplishes the same thing.

Consider a known NPC problem, M. From definition 4 we know that all problems in NP must reduce to problem M. Now if we are trying to show that out problem P is NPC it is enough to show a reduction from problem M. Therefor we can relax condition 2 of definition 4 to be the following:

Definition 6. *(NP-Complete): A problem p is said to be NP-Complete if:*

1. $P \in NP$
2. $\exists Q \in NPC$ *and we have a polynomial time reduction from Q to P*

The COMPATIBILITY problem is shown to be NPC in Introduction to Computational Molecular Biology. COMPATIBILITY can easily be verified in P-Time meaning that it is a member of NP. The reduction follows from building the matrix out of a graph that contains a clique. The basic idea is that whenever we have a pair of nodes that are not connected in the graph we force a conflict in the matrix. This can be done by having 3 objects in our matrix for every pair of nodes. If there is no edge between them in the graph then we assign them the following values to the three objects: 110 and 011. If there is an edge between the pair of nodes, then their three objects in the table contain just 0's. This is because 0's can never create a conflict.

It is easy to see that a conflict will be created only when there is no edge in the graph. It then suffices to show that a clique in the graph will produce the largest set of nodes to build a perfect phylogeny and our reduction is complete. Further details can be found here [4]

4 Fixed-Parameter Tractable

Fixed-Parameter Tractable (FPT) algorithms are closely related to NPC in that they are generally considered computationally hard problems. The main difference is that as the name suggests we choice a parameter, k for the problem and construct our solutions around this parameter. Our goal then becomes to run exponential parts of the problem in our parameter k, and not in the input size n.

Intuitively we are trying to solve the problem but doing as much as we can to the input in polynomial time, and then running the exponential algorithm on only a very small subset of the input. Formally FPT is defined as follows:

Definition 7. *(FPT): A problem P is said to be FPT if:*

1. *There exists some parameter k such that k is not the input size.*
2. *Run in Time $f(k)n^{O(1)}$*

Note 2. *$f(k)$ can be exponential in k as long as it is also not exponential in n.*

5 COMPATIBILITY is in FPT

From section 3.3 we know that COMPATIBILITY is NPC. We will now show that it is also in FPT. We will break the problem into two parts; 1) Building a conflict graph and finding a vertex cover and 2) Building the Phylogeny.

5.1 Conflict Graphs and Vertex Cover

The first thing that we need to do to solve the problem is to take the binary matrix, M, and convert it into a conflict graph, $G(V, E)$ where each characteristic $x_i \in M$ is an node $n \in V$ and there is an edge between node x_i and x_j if and only if there is a conflict between x_i and x_j.

From Lemma 1 we know that a conflict occurs only when one pair of vertices are not a subset of each other, and not disjoint and that we can decide this in P-Time. We now have a conflict graph where each edge represents a conflict. Now we wish to find the minimum set of nodes such that if we remove all of the those nodes and edges that connect to those nodes we are left with a graph that has no edges in it. In other words we are interesting in finding a Vertex cover of size at most B. When we have obtained the vertex cover we simply remove those nodes and we are left with a conflict graph with no conflicts. This means we are now able to build a perfect phylogeny.

5.2 Vertex Cover is in FPT

In order to finish solving part one of section 5 we need to show that Vertex Cover is in FPT. Vertex cover is a famous FPT algorithm and has been proved to be in FPT many times ([3] for example). I will present a naive approach here to confirm the proofs. First we need to proof a lemma about one of the properties of a graph that contains a vertex cover.

Lemma 2. *Given some graph G(V,E), if an edge exists between two nodes v_i and v_j, then either v_i or v_j is in the vertex cover.*

Proof. Let $C \subset V$ be a vertex cover of G. Assume $v_i \notin C$ and $v_j \notin C$. But then the edge $e_{i,j}$ is not incident to any nodes in the vertex cover. Therefor either v_i or v_j must also be added to C. □

Corollary 1. *Given a node $x \in G$ we must include either x or $N(x)$ where $N(x)$ is the set of nodes that are connected to x by an edge.*

Proof. Follows directly from Lemma 2. □

Theorem 1. *Vertex Cover is in FPT*

Proof. We are interested in finding a vertex cover of at most k. Consider Corollary 1, and do the following steps

1. Pick a node $x \in V$ and either include x or $N(x)$.
2. Let $k = k-1$ if we included x, or let $k = k-|N(x)|$ if we included $N(x)$ in the vertex cover.
3. Remove x, $N(X)$ and all edges that connected x and $N(x)$. (Figure 1 and figure 2)

After steps 1-3 we are left with a reduced graph. We then repeat steps 1-3 until either $k = 0$, $k < 0$, or $|V| = 0$. We can do this for every node in the graph, building a search tree of the input. The depth of the tree is at most k because after each step of the search we add at least one node to the vertex cover. At each branch of the tree with have two options; 1) include x or include $N(x)$ therefor it becomes possible to search a tree for a given node x in $O(2^k)$ time. Since we have n nodes we can therefor do an exhaustive search in $O(2^k n)$ time. from Definition 7 we have shown Vertex Cover to be in FPT. □

It should be noted that the proof presented above is a very naive approach and Vertex Cover has been shown to be solvable in $O(1.2738^k + kn)$ by Chen et all. in 2003 [3]. The proof this running time is technical and is left to the reader.

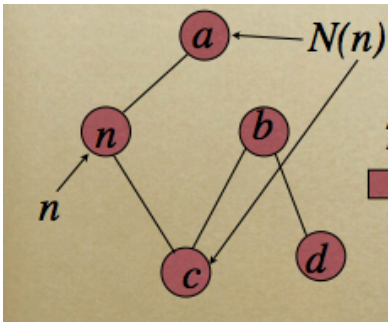

Figure 1: Before Removal

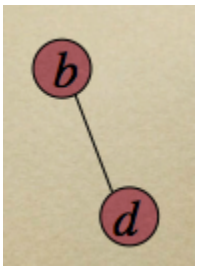

Figure 2: Before Removal

6 Algorithm

Consider an instance of the COMPATIBILITY problem from Definition 3, we now present an FPT algorithm to solve an

instance of COMPATIBILITY. We break the problem up into two parts; 1) find and remove the problem characteristics and 2) Build the tree with the set of none conflicting characters.

6.1 Finding Conflicts

Consider a matrix M as defined in Definition 3 and a graph $G(V, E)$ where each character $c_i \in M$ is a node $v_i \in V$ and an edge $e_{i,j}$ is in the graph iff there exists a conflict between c_i and c_j. We have just constructed a conflict graph for matrix M and now concern ourself with removing the conflicts.

Lemma 3. *By removing all edges from the graph G we can construct a perfect Phylogeny.*

Proof. From Lemma 1 we know that in order to build a perfect Phylogeny we need there to exist no conflicts between characteristics. Therefor removing them from the conflict graph would allow us to build a perfect phylogeny. □

In order to remove all conflicts in G we wish to find a set of nodes R in the graph such that when we remove R and all edges incident to R we are left with no edges remaining in the graph. This now becomes an instance of Vertex Cover were we are concerned with finding a vertex cover of size greater then or equal to B where B is the maximum number of characteristics we wish to remove.

Lemma 4. *Removing conflicts can be done in $O(2^k n)$ time.*

Proof. Since we have shown that removing conflicts is an instance of the vertex cover it is enough to show that vertex cover can run in $O(2^k n)$ time. Theorem 1 shows this, and therefor removing conflicts can be done in $O(2^k n)$ time. □

By running the vertex cover algorithm presented in section 5.2 we can decided if it is possible to remove the conflicts by removing up to B characteristics, and if it is possible we can find the minimal set of characteristics, C to remove in FPT time. We know turn our attention to building the perfect phylogeny.

6.2 Building The Phylogeny

Consider trying to construct a perfect phylogeny from M', where M' is M with the rows corresponding to C removed.

Lemma 5. *Construction of a perfect phylogeny tree from matrix M' can be done in $O(NM)$ time.*

Proof. Building a perfect phylogeny tree in $O(NM)$ time is shown on slide 13 of lecture 14. □

6.3 COMPATIBILITY is in FPT

Bringing together section 6.1 and section 6.2 we get the following theorem.

Theorem 2. *COMPATIBILITY is in FPT*

Proof. Let B equal the parameter k for the problem. We have shown that the problem reduces to the following:

1. Converting the problem to conflict graph.
2. Finding the minimal set of nodes that when remove resolve all conflicts.
3. Building the perfect phylogenetic tree.

Showing that all three points can be done in FPT time with parameter k will show that COMPATIBILITY is in FPT.
1: Converting M into a conflict graph can be done in $O(nm^2)$ by simply comparing each pair of elements row by row. Since there are n rows in the matrix and $O(m^2)$ such pairs the overall running time is $O(nm^2)$.
2: Finding the minimal set of nodes to remove can be done in $O(2^k n)$ time as shown in Lemma 4.
3: Building the tree itself can be done in O(NM) time as shown in Lemma 5
Bringing it all together we see that it is possible to solve COMPATIBILITY in $O(2^k n + nm^2 + mn)$ time. Therefor by definition 7 COMPATIBILITY is in FPT. □

7 Conclusions

We have shown that the problem of COMPATIBILITY is in FPT which greatly decreases the time required to build a phylogenetic tree. Consider an example of a of a matrix that contains 10 characteristics. Now let B = 3. If we were to not consider the FPT algorithm we would have to search a tree of size $2^10 = 1024$ where as the FPT algorithm could solve the problem on the order of $2^3 = 8$.

Given just this small example and considering practical applications when the size of characteristics is high, but the number we want to remove remains low we are presented with an incredibly fast speed up.

Finally the problem of conflict resolution is something that appears in various areas of computer science. Future work could explore those areas and see if this result has the potential to help in an other field.

8 References

1. Cook, S.A. (1971). "The complexity of theorem proving procedures". Proceedings, Third Annual ACM Symposium on the Theory of Computing, ACM, New York. pp. 151158. doi:10.1145/800157.805047.

2. Garey, M.R.; Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman. ISBN 0-7167-1045-5

3. Jianer Chen, Iyad A. Kanj: Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. J. Comput. Syst. Sci. 67(4): 833-847 (2003).

4. Meidanis, J and Setubal, J. *Introduction to Computational Molecular Biology*. Pacific Groove, CA. Brooks/Cole Publishing Company. ISBN0-534-95262-3