

COMBINATORIAL OPTIMIZATION AND APPLICATION TO DNA SEQUENCE ANALYSIS

A Thesis
Presented to
The Academic Faculty

by

Kapil Gupta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
December 2008

COMBINATORIAL OPTIMIZATION AND APPLICATION TO DNA SEQUENCE ANALYSIS

Approved by:

Dr. Eva K. Lee, Advisor
School of Industrial and Systems
Engineering
Georgia Institute of Technology

Dr. Earl Barnes
School of Industrial and Systems
Engineering
Georgia Institute of Technology

Dr. Yuhong Fan
School of Biology
Georgia Institute of Technology

Dr. Ellis Johnson
School of Industrial and Systems
Engineering
Georgia Institute of Technology

Dr. Ming Yuan
School of Industrial and Systems
Engineering
Georgia Institute of Technology

Date Approved: July 31, 2008

*To my parents,
Dr. O.P. Gupta and Pushpa Gupta*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to everyone who assisted in the completion of this thesis, both directly and indirectly. First and foremost, I would like to thank my advisor, Dr. Eva K. Lee. I am deeply grateful to her for constant guidance, inspiration, financial support, and encouragement throughout my stay at Georgia Tech.

I am very grateful to Professors Earl Barnes, Yuhong Fan, Ellis Johnson, Ming Yuan, for serving as committee members, and for their constructive comments and suggestions.

I would like to thank all my good friends for making the best of the worst times. I would like to thank my longtime friends: Bhaskar, Gayatri, Pranava, Sandipan for their continuous support. My roommates Bobba, JP, Mudhakar, Santosh deserves a special mention for helping me during the course of my studies here. I would like to thank my friends who made my stay in Atlanta immensely memorable: Divya, Karan, Mita, Prashant, Preetham, Prerona, Rajeev, Richa, Shalini. My fellow ISyE graduate students have been very helpful and supportive. I would like to specially thank Dylan, Fatma, Juan Pablo, Kong, Kyungduck, Pete, Rebeca, Ricardo, Sid.

My thanks would not be complete without acknowledging the support of my brother Ashish, sister-in-law Vandana, and my parents. Without their unconditional love, faith and unimaginable patience, this thesis would never have been possible. Finally and most importantly, I would like to thank God for everything.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
SUMMARY	xi
I THESIS OVERVIEW	1
1.1 Background and Motivation	1
1.2 Integer Programming Formulation	2
1.3 Theoretical Study	3
1.4 Computational Study	4
1.5 Summary	4
II INTRODUCTION	5
2.1 Literature Review	7
2.1.1 Multiple Sequence Alignment	7
2.1.2 Phylogenetic Analysis	10
2.1.3 DNA Sequencing	11
2.1.4 Sequence Comparison	12
2.2 Model formulation	13
2.2.1 Definitions	13
2.2.2 Construction of a Conflict Graph from Paths of Multiple Sequences	14
2.3 Summary	16
III ALGORITHMS FOR DNA SEQUENCE ANALYSIS	17
3.1 Introduction	17
3.2 Phylogenetic Analysis	18
3.2.1 Methods Based on Pairwise Distance	19

3.2.2	Parsimony Methods	21
3.2.3	Maximum Likelihood Methods	25
3.3	Multiple Sequence Alignment	28
3.3.1	Scoring Alignment	30
3.3.2	Alignment Approaches	31
3.3.3	Progressive Algorithms	32
3.3.4	Graph Based Algorithms	33
3.3.5	Iterative Algorithms	36
3.4	Summary	39
IV	COMPLEXITY THEORY	41
4.1	Graph Construction and MWCMS complexity	42
4.2	Special Cases of MWCMS	50
4.2.1	Shortest Common Supersequence	50
4.2.2	Longest Common Subsequence	52
4.2.3	Shortest Common Superstring	53
4.3	Concluding Remarks	55
V	COMPUTATIONAL MODELS: INTEGER PROGRAMMING FORMULATION	57
5.1	Integer Programming Formulation	57
5.2	Network Formulation	59
5.2.1	Graph Construction	59
5.2.2	Notation	61
5.2.3	Network Formulation-Alphabet	63
5.2.4	Network Formulation-Supernode	63
5.2.5	Network Formulation-Clique	65
5.3	Conflict Graph Corresponding to Arcs Between Supernodes	66
5.3.1	Definitions	67
5.3.2	Maximal Cliques	67
5.3.3	Clique Network	71

5.4	Polyhedral Structure of Network flow formulation-Clique	73
5.4.1	Formulation	73
5.4.2	Dimension of the Polytope	74
5.4.3	Maximal Clique Inequalities are Facets	75
5.5	Computational Tests	75
5.6	Conclusion	76
VI	SIMULTANEOUS ROW AND COLUMN GENERATION APPROACH FOR SOLVING WEIGHTED NODE PACKING PROBLEMS	77
6.1	Formulation	77
6.2	Outline of Algorithm	78
6.3	Pricing	79
6.3.1	Constrained Shortest Path	80
6.3.2	Feasible Paths using Progressive Alignment	83
6.4	Clique Probing	84
6.5	Combining Network and Path formulation	85
6.6	Computational Tests	86
6.7	Conclusion and Future Work	92
	REFERENCES	93

LIST OF TABLES

1	Performance comparison of CPLEX default and Maximal Clique Cuts	75
2	Performance comparison of Network and Path-Adjacency formulation	87
3	Performance comparison of Path-Adjacency formulation and Path-Arc-Clique formulation	88

LIST OF FIGURES

1	Simple evolutionary tree structure	11
2	An example of evolutionary tree	18
3	Tree Terminology	18
4	Parsimony tree 1	22
5	Parsimony tree 2	22
6	The sets R_k for the first site of given three sequences	24
7	A simple tree	26
8	Two possible alignments for given three sequences	29
9	An example of Σ -cross when $\Sigma = \{A, C, G, T\}$	42
10	An example of the 3-layer supergraph for converting the sequence $S = GC$ to $B = TC$	44
11	A sample graph G_L^m of MWCMS with $S_1 = GC$ to $S_2 = TG$ where $\Sigma = \{A, C, G, T\}$	48
12	An example graph G' of an SCST with $S_1=GC$ to $S_2=TG$ where $\Sigma = \{A,C,G,T\}$	55
13	An example of Σ -cross when $\Sigma = \{A, C, G, T\}$	60
14	A sample graph G_L^m with $S_1 = GC$ to $S_2 = TG$ where $\Sigma = \{A, C, G, T\}$	62
15	An example graph G having two layers	68
16	A sample graph G with $N_1 = 3$ and $N_2 = 2$	68
17	Conflict graph for G with $N_1 = 3$ and $N_2 = 2$	69
18	Maximal Clique Property	70
19	Clique network for $N_1 = 4$ and $N_2 = 3$	71
20	A clique network for a general 2-layer graph having N_1 and N_2 supernodes	72
21	Outline of column and row generation algorithm for path formulation	79
22	A sample graph of pricing subproblem for 4 sequences	81
23	Conflict graph for sample graph of pricing subproblem	81
24	Path-Arc-Clique algorithm performance for 5 sequences	88
25	Path-Arc-Clique algorithm performance for 8 sequences	89

26	Path-Arc-Clique algorithm performance for even number of sequences	90
27	Path-Arc-Clique algorithm performance for odd number of sequences	91

SUMMARY

With recent and continuing advances in bioinformatics, the volume of sequence data has increased tremendously. Along with this increase, there is a growing need to develop efficient algorithms to process such data in order to make useful and important discoveries. Careful analysis of genomic data will benefit science and society in numerous ways, including the understanding of protein sequence functions, early detection of diseases, and finding evolutionary relationships that exist among various organisms.

Most sequence analysis problems arising from computational genomics and evolutionary biology fall into the class of \mathcal{NP} -complete problems. Advances in exact and approximate algorithms to address these problems are critical. In this thesis, we investigate a novel graph theoretical model that deals with fundamental evolutionary problems. The model allows incorporation of the evolutionary operations “insertion”, “deletion”, and “substitution”, and various parameters such as relative distances and weights. By varying appropriate parameters and weights within the model, several important combinatorial problems can be represented, including the weighted super-sequence, weighted superstring, and weighted longest common sequence problems. Consequently, our model provides a general computational framework for solving a wide variety of important and difficult biological sequencing problems, including the multiple sequence alignment problem, and the problem of finding an evolutionary ancestor of multiple sequences.

In this thesis, we develop large scale combinatorial optimization techniques to solve our graph theoretical model. In particular, we formulate the problem as two distinct but related models: constrained network flow problem and weighted node

packing problem. The integer programming models are solved in a branch and bound setting using simultaneous column and row generation. The methodology developed will also be useful to solve large scale integer programming problems arising in other areas such as transportation and logistics.

CHAPTER I

THESIS OVERVIEW

1.1 Background and Motivation

A DNA sequence is an ordered sequence of finite length involving the four nucleotides adenine(A), thymine(T), cytosine(C), and guanine(G). Associated with each DNA strand (sequence) is a complimentary DNA strand of the same length. The strands are complementary in that each nucleotide in one strand uniquely defines an associated nucleotide in the other: A and T are always paired, and C and G are always paired. Each pairing is referred to as a base pair; and bound complementary strands make up a DNA molecule. Typically, the number of base pairs in a DNA molecule is between thousands and billions, depending on the complexity of the organism. For example, a bacterium contains about 600,000 base pairs, while the human and mouse genomes have approximately 3 billion base pairs. For humans, 99.9 percent of base pairs are the same between any two unrelated persons. That still leaves millions of single-letter differences, which provide genetic variation between people.

Understanding DNA sequence structure is extremely important. It is the blueprint for manifestation of physical characteristics and function, and sequence order underlies all of life's diversity, dictating whether an organism is human or another species such as yeast or a fruit fly. It helps in understanding the evolution of mankind, identifying genetic diseases, and creating new approaches for treating and controlling those diseases.

To achieve pathbreaking scientific discoveries, both sequence data and efficient algorithms for analyzing the data are pivotal. Recently, the amount of sequence data has increased tremendously, in part due to the completion of the Human Genome

Project. There is, however, an apparent lack of efficient algorithms to analyze this huge amount of data. Most existing algorithms are heuristic in nature and suffer from various drawbacks and limitations.

Theoretical studies have shown that most sequence analysis problems fall into the class of \mathcal{NP} -Complete problems. Thus, there exists no known polynomial-time algorithm that solves every instance. Further, current algorithms lack in two important areas that can be improved upon. First, although all the sequence analysis problems have a similar structure, there does not exist a unified approach to efficiently model these problems or to exploit the common structure computationally. Second, the existing near-exact algorithms do not scale well, and thus are unable to cope up with the breathtaking growth in available biological data.

The purpose of this thesis is to

- Investigate a novel graph-theoretical approach for representing a wide variety of sequence analysis problems within a single model. By varying weights and parameters in the model, different classes of sequence analysis problems like “Multiple Sequence Alignment” and “Evolutionary Distance” can be represented.
- Develop theoretical results and computational techniques which are applicable not only for sequence analysis problems but also to variety of other real-life problems involving large scale combinatorial optimization. The constrained network flow and weighted node packing formulations studied in this thesis have numerous applications in areas such as transportation and logistics.

1.2 Integer Programming Formulation

Given a set of DNA sequences, our graph-theoretical approach first involves building a multi-layer graph in which each layer represents a sequence, and each node denotes a nucleotide. Edges between consecutive layers represent matching nucleotides. A construct of *supernode* is introduced that enables insertion, deletion and substitution

operations. From this graph, a conflict graph is constructed in which each node corresponds to a path from the first sequence to the last sequence in the multi-layer graph. An edge in the conflict graph corresponds to two paths crossing in the associated multi-layer graph. With this, one can prove that finding a node-packing is equivalent to finding a common “mutated” sequence among multiple DNA sequences. To approach the problem computationally, we formulate the problem as two distinct but related models: constrained network flow problem and weighted node packing problem. These integer programming models are solved in a branch and bound setting using simultaneous column and row generation.

1.3 Theoretical Study

Lee et al. [54] and [53] modeled various classes of sequence analysis problems as *minimum weight common mutated sequence* (MWCMS) problem. Through the introduction of *supernodes*, and the *multi-layer supergraph*, they proved that MWCMS is \mathcal{NP} -complete. Furthermore, they showed that a conflict graph derived from the multi-layer supergraph has the property that a solution to the associated node-packing problem of the conflict graph corresponds to a solution of the MWCMS problem. They demonstrated that some well-known combinatorial problems can be viewed as special cases of the MWCMS problem. In particular, they presented theoretical results implied by the MWCMS theory for the minimum weight supersequence problem, the minimum weight superstring problem, and the longest common subsequence problem.

In this thesis, we perform a polyhedral study on two integer programming formulations used for solving the MWCMS problem. For the constrained network flow model, we find strong valid inequalities to strengthen the linear relaxation. We find the dimension of these valid inequalities and prove they are facet-defining for a class of instances. For the weighted node packing problem, we exploit the polyhedral structure to develop network flow models which could be used as subroutines for effective

row and column generation.

1.4 Computational Study

The integer programming models developed for solving various sequence analysis problems have proven to be computationally intensive. In this thesis, we present various computational techniques to solve the models along with elegant heuristics to get good feasible solution.

In particular, we reformulate and implement formulations with tighter linear relaxation for solving the two integer programming models. For the constrained network flow model, we develop a formulation using only cliques instead of adjacency constraints. Moreover, we develop and implement a polynomial-time separation routine for separating these clique cuts. These clique cuts have proven to be extremely effective and results in more than a 90% reduction in computational time.

For solving the node packing problem corresponding to complete paths, we develop simultaneous row and column generation approach. The algorithm introduced herein for multiple sequence alignment, overcomes the order-dependent drawbacks of many of the existing algorithms, and is capable of returning good sequence alignments within reasonable computational time.

1.5 Summary

In this thesis, we study sequence analysis problems both from a practical and theoretical view, using tools from integer programming to develop efficient computational techniques. The methodology developed will be also be useful to solve large scale integer programming problems arising in areas such as transportation and logistics. We continue to focus on developing and investigating solution techniques to the two integer programming models, along with the development of fast heuristic procedures.

CHAPTER II

INTRODUCTION

With recent and continuing advances in bioinformatics (e.g., the completion of Human Genome Project), the volume of sequence data has increased tremendously. Along with this increase, there is a growing need to develop efficient algorithms to process such data in order to make useful and important discoveries. Careful analysis of genomic data will benefit science and society in numerous ways, including understanding protein sequence functions, early detection of diseases, and finding the evolutionary relationships that exist among various organisms.

Most sequence analysis problems arising from computational genomics and evolutionary biology fall into the class of \mathcal{NP} -complete problems. Advances in exact and approximate algorithms to address these problems are critical. In this thesis, we describe a novel graph theoretical model that deals with fundamental evolutionary problems. The model allows incorporation of evolution operations “insertion”, “deletion”, and “substitution”, and various parameters such as relative distances and weights. By varying appropriate parameters and weights within the model, several important combinatorial problems can be represented, including the weighted super-sequence, weighted superstring, and weighted longest common sequence problems. Consequently, our model provides a general computational framework for tackling a wide variety of important and difficult biological sequencing problems, including the multiple sequence alignment problem, and the problem of finding an evolutionary ancestor of multiple sequences.

Conceptually, we refer to our model as the *minimum weight common mutated sequence* (MWCMS) problem. Our motivation for first defining this problem arose

from the desire to help quantify the concept of “best” representative sequence in the evolutionary distance problem. The evolutionary distance problem involves finding the DNA sequence of the most likely ancestor associated with a given set of DNA sequences from distinct but similar organisms. In other words, find the DNA strand that best represents a possible ancestor, if each of the organisms evolved from the same ancestor. Changes that contribute to differences between the given sequences and the ancestor are referred to as insertions, deletions and substitutions. These operations account for both evolutionary mutations and experimental errors in sequencing. Mathematically, given two sequences S and B , let $ord(S, B)$ be an ordered collection of insertions, deletions and substitutions to convert sequence S to B . (For any two sequences S and B , there are an infinite number of collections $ord(S, B)$.) Let $w(ord(S, B))$ be the weight of the conversion from S to B , where the weight is the sum of an expression involving values η , δ and $\psi \in \mathfrak{R}^+$ which represent the weights associated with a single insertion, deletion and substitution, respectively. Let $ord^*(S, B)$ be such that $w(ord^*(S, B)) \leq w(ord(S, B))$ for all $ord(S, B)$. Define $d(S, B) = w(ord^*(S, B))$. Formally, MWCMS can be stated as:

Problem MWCMS: Given positive weights η , δ and ψ corresponding to a single insertion, deletion and substitution respectively, a positive threshold κ , and finite sequences S_1, \dots, S_m from a finite alphabet, does there exist a sequence B such that $\sum_{i=1}^m d(S_i, B) \leq \kappa$?

This chapter includes motivation for our study and explains how our evolutionary model addresses various important classes of applications in computational biology. We also state definitions, and lay out the graph-theoretical foundations for establishing our theoretical results. In particular, it describes the construction of a multi-layer graph for a collection of sequences, and highlights the associated conflict graph. A solution to the node-packing problem of the conflict graph corresponds to a solution of the longest common subsequence problem (LCS), which is a special case of

MWCMS. In Chapter 3 , we review some well known algorithms that facilitate DNA sequence analysis. The material is presented in a way that is interesting to both the specialists working in this area and others. Thus, this review includes a brief sketch of the algorithms to facilitate a deeper understanding of the concepts involved.

Chapter 4 includes theoretical findings for MWCMS. In this chapter, we construct *supernodes* and *multi-layer supergraphs* that allow us to model the MWCMS problem and perform theoretical investigations into the complexity of this problem. Furthermore, we show that a solution of the node-packing problem of the conflict graph derived from the multi-layer supergraph corresponds to a solution of the MWCMS problem. We also demonstrate that some well-known combinatorial problems are special cases of the MWCMS problem. In particular, the minimum weight supersequence, the minimum weight superstring, and the longest common subsequence problems are special cases. Chapter 5 presents integer programming formulations for solving various sequence analysis problems using multi-layer graph. Furthermore, we develop and implement polynomial time separation routine for class of clique cuts and present the computational results.

Chapter 6 presents algorithm and computational results for weighted node packing formulation. In this chapter, we develop simultaneous row and column generation approach for solving node packing problem on conflict graph corresponding to complete paths.

2.1 Literature Review

In this section, we describe some of the important problems that can be represented by our model and briefly summarize current methods for approaching these problems.

2.1.1 Multiple Sequence Alignment

Multiple sequence alignment (MSA) is arguably among the most studied and difficult problems in computational biology. It is routinely used in a wide range of

sequence analysis problems, for example, in identifying conserved patterns in a given set of sequences. For two sequences, optimal MSA is easily obtained using dynamic programming (Needleman-Wunsch algorithm). Unfortunately, the problem becomes much harder for multiple sequences. Researchers have tried to solve it by generalizing the dynamic programming approach to multi-dimensional space. However, this approach requires extensive computational time and strenuous memory requirements. Thus, it cannot be used in practice even for a small instance with only 5 sequences each with 100 base pairs in length. The algorithm has since been improved by identifying the portion of hyperspace that does not contribute to the solution and thus can be excluded from the computation. However, this improved approach still proves to have limited scalability, as it can only align efficiently up to ten sequences [57]. As a result, a wide variety of heuristic approaches have been developed for MSA [63].

It is interesting to notice that most of the heuristic algorithmic approaches are based on pairwise alignment information of input sequences. An order of sequences is first developed based on pairwise information, and then sequences are aligned one by one. This class of algorithm is referred to as progressive alignment[15]. There are three main shortcomings in this approach.

1. There does not exist an undisputable “best” way of ordering the given sequences.
2. Once a sequence has been aligned, that alignment will not be modified even if it conflicts with sequences added later in the process. Hence, the order in which sequences are added becomes very crucial, and since there is no undisputable best way to order the sequences, this approach returns sub-optimal solutions.
3. For given n sequences, $\binom{n}{2}$ pairwise alignments are generated; but while computing the final multiple alignment, most of these algorithms use fewer than n pairwise alignments. Thus, the resulting multiple alignment agrees with only a small amount of information available in the data.

Hence, there is a growing need for algorithms to align divergent sequences whose pairwise alignments are likely to be incorrect. In order to address all these issues, some innovative techniques have been developed. We briefly summarize a few of them below.

Kececioglu et al.(2000) use a solution of the maximum trace problem to construct alignment [43]. The algorithm starts with calculating all pairwise alignments and using them to find a trace. To achieve this, they represent pairwise alignments by a graph whose vertices are the characters of the sequences and edges are the pairs of characters matched in the pairwise alignments. The algorithm is capable of giving an exact solution under the sum-of-pairs objective function with linear gap costs. Kececioglu et al. have made a significant contribution by introducing a polyhedral approach capable of obtaining exact solutions for a subclass of MSA. However, this methodology has its own drawbacks like not being able to capture the order of insertions and deletions between two matchings and affine gap costs. Recently, Althaus et al. (2006) has proposed a general model using this approach in which arbitrary gap costs are allowed [2].

Shyong Jian et al.(2004) explore the use of minimum spanning trees to determine the order of sequences [72]. The idea of the approach is to preserve the most informative distances among the set of given sequences. The criteria used is meaningful and capable of working better than the traditional criteria like those in sum-of-pairs. The algorithm itself is very efficient for practical usage, and can be easily implemented. However, it fails to address the issue of using all pairwise alignments, since it only uses the score and not the pairwise alignments themselves. Moreover, this approach has all the drawbacks of the progressive strategy.

Zhang and Waterman (2003) proposed a new approach motivated by the Eulerian method for fragment assembly in DNA sequencing [90]. In their work, a consensus

sequence is found and later pairwise alignments are obtained between each input sequence and consensus sequence. Finally, MSA is obtained according to these pairwise alignments. The most significant advantage of this method is linear time and memory cost for finding the consensus sequence. And, if the consensus sequence is the closest one to all given sequences, good quality alignment can be obtained in a reasonable amount of time. Once again, this approach suffers from the prominent drawback of the progressive strategy and issues in graph formation while finding the consensus sequence. *The model and methods introduced in this thesis overcome many of the above issues; in particular, they do not require ordering of sequences.*

2.1.2 Phylogenetic Analysis

Phylogenetic Analysis is a major aspect of genome research. It refers to the study of evolutionary relationships of a group of organisms. These hierarchical relationships among organisms arising through evolution are usually represented by a phylogenetic tree. The model in this thesis will serve as a very important tool to the following two problems.

- *Evolutionary Tree:* The idea of using trees to represent evolution dates to Darwin. Both rooted and unrooted tree representations have been used in practice [19]. The branches of a tree represent the time of divergence and the root represents the ancestral sequence (Figure 1). Most algorithms require the ability to compute distances between every pair of sequences. Calculating these pair-wise distances is a well-known problem and has been studied extensively [71] and [78]. The MWCMS problem can be used to calculate distance between every pair of sequences.
- *Most likely ancestor:* Phylogenetic analysis becomes increasingly complex when recombination occurs between sequences [35]. It is commonly witnessed in mitochondrial DNA sequences and viral sequences. The recombination implies

and [38]. Sequencing by hybridization is the process where every possible sequence of length n (4^n possibilities) is compared to a full DNA strand. Practical values for n are 8 – 12. Each short string either binds or does not bind to the full strand. Biologists can thus determine exactly which short strings are contained in the DNA strand and which are not.

However, the experiment does not identify the exact location of each short string in the full strand. Hence, an important issue involves how these short strings are connected together to form the complete strand. This problem can be viewed as a shortest common superstring problem and has been studied extensively [59], [23], and [24]. Unfortunately, errors may arise during sequencing experiments. Three types of errors are: deletions (a letter appears in an input string that should not be in the final sequence), insertions (a letter is missing from an input string), and substitutions (a letter in an input string should be substituted with another letter). *The MWCMS problem can be used to model and solve this shortest common superstring problem while addressing the issue of possible errors.*

2.1.4 Sequence Comparison

Sequence Comparison is one of the most crucial problems faced by researchers in the area of bioinformatics. The sequence patterns are conserved during evolution. Given a new sequence, it will be of interest to understand how much similarity it has with pre-existing sequences. Significant similarity between two sequences implies similarities in their structure and/or function. There are lots of DNA databases containing DNA sequences and their function. The major ones are GenBank in the U.S. and the EMBL data library in Europe. If one finds a new sequence similar to existing ones in these databases, one can transfer information about the function and structure [86]. Hence, an algorithm for sequence comparison which is efficient for large number of sequences will play a pivotal role in rapid sequence analysis. *The*

MWCMS problem can be used to address this issue.

2.2 Model formulation

In this section, we provide details on construction of multi-layer graph G_L^m and corresponding conflict graph for solving the MWCMS problem.

2.2.1 Definitions

We have defined the minimum-weight common mutated sequence (MWCMS) problem — which incorporates the notions of insertion, deletion and substitution — to help quantify the concept of “best” representative sequence in the evolutionary distance problem. We now make precise the operations of *insertion*, *deletion* and *substitution*.

Let $S = \{s_1, \dots, s_n\}$ be a finite sequence of letters from a finite alphabet.

- i An *insertion* of an element x in position i of the sequence S is characterized by the addition of x between elements s_i and s_{i+1} . An insertion carries an associated penalty cost of η .
- ii A *deletion* of an element in position i of S amounts to deleting s_i from the sequence S . The penalty for deletion is represented by δ .
- iii A *substitution* of an element in position i of S amounts to replacing s_i with another letter from the alphabet. The penalty for substitution is represented by ψ .

We remark that a penalty cost for an operation could, more generally, depend on the position where the operation is performed and/or the element to be inserted, deleted, or substituted.

Let $S_1 = \{s_{11}, \dots, s_{1m}\}$ and $S_2 = \{s_{21}, \dots, s_{2n}\}$ be two finite sequences of letters from a finite alphabet Σ . We say that the *relative distance* between elements s_{1i} and s_{2j} is k if $|i - j| = k$. We define a *k-restrictive* bipartite graph as a graph

$G_k = (V_1, V_2, E_k)$ such that the nodes in V_1 and V_2 correspond respectively to each of the elements from the first and the second sequences. We assume the nodes in V_i are ordered in the same order as they appear in the sequence S_i . There is an edge between nodes $u \in V_1$ and $v \in V_2$ if u and v are identical (i.e., same letter of the alphabet Σ) and if the relative distance between these two elements is less than or equal to k . The problem of identifying the “greatest similarity” between these two sequences can then be approached as the problem of finding a maximum cardinality matching between the associated node sets, subject to restrictions on which matchings are allowed. In particular, one must take into consideration the ordering of nodes so as to preserve the relative occurrence of the elements in the matching. In addition, matchings that have edge crossings must be prevented. When $k = \max\{|S_1|, |S_2|\} - 1$, we denote the graph by $G = (V_1, V_2, E)$, and the problem is equivalent to the well-studied longest common subsequence (LCS) problem for two sequences, which is polynomial-time solvable [24].

2.2.2 Construction of a Conflict Graph from Paths of Multiple Sequences

Let $S_i, i = 1, \dots, m$, be a collection of finite sequences, each of length n , over a common alphabet Σ . Let $G_k = (V_1, \dots, V_m, E_1, E_2, \dots, E_{m-1})$ be the k -restrictive multi-layer graph in which each element in S_i forms a distinct node in V_i . Assume the nodes in V_i are ordered in the same order as they appear in the sequence S_i . E_i denotes the set of edges between nodes in V_i and V_{i+1} . There is an edge between nodes $u \in V_i$ and $v \in V_{i+1}$ if and only if u and v are the same letter in the alphabet Σ , and the relative distance between them is less than or equal to k . The multiple sequence comparison problem involves finding the longest common subsequence (LCS) within the sequences $S_i, i = 1, \dots, m$. We call a path $P = p_1, p_2, \dots, p_m$ a complete path in G_k if $p_i \in V_i$ and $p_i p_{i+1} \in E_i$. Two complete paths are said to be parallel if their node sets are disjoint and the edges do not cross. Hence, a set of parallel

complete paths in G_k corresponds to a feasible solution to LCS on the collection of sequences $S_i, i = 1, \dots, m$. We say that two complete paths P_1, P_2 *cross* if they are not parallel. We remark that the LCS problem with the number of sequences bounded, is polynomial time solvable using dynamic programming [24]. In general, the problem remains \mathcal{NP} -complete.

We can incorporate insertions by generating new paths which include inserted nodes on various layers. The weight for such a new path will be affected by the total number of insertions in the path. In particular, if L is a common subsequence for S_i and $|S_i| = n$ for all $i = 1, \dots, m$, then the total number of unmatched elements remaining will be $m(n - |L|)$. These elements can be deleted completely, or for a given unmatched element, one can increase the size of L by 1 by appropriately inserting this element into various sequences. By doing so, the number of unmatched elements decreases. Let l be the number of insertions needed to generate a new complete path. Then the number of unmatched elements will decrease by $m - l$. If we assume that at the end of the sequencing process all unmatched elements will be deleted, then the penalty for generating this new complete path will be given by $l\eta - (m - l)\delta$.

We next define the concept of conflict graph relative to complete paths in G_k .

Definition. 2.2.1 Let $\mathcal{P} = \{P_1, \dots, P_s\}$ be a finite collection of complete paths in G_k . The *conflict graph* $\mathcal{C}_{\mathcal{P}} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ associated with \mathcal{P} is constructed as follows:

- $V_{\mathcal{P}} = \{P_1, \dots, P_s\}$;
- there is an edge between two nodes P_i and P_j in $V_{\mathcal{P}}$ if and only if P_i and P_j cross each other.

This definition applies to any multi-layer graph in general. Note that any stable set of nodes in $\mathcal{C}_{\mathcal{P}}$ corresponds to a set of parallel complete paths for G_k , and thereby to a feasible solution to LCS on the collection of sequences $S_i, i = 1, \dots, m$.

For $m = 2$, the resulting conflict graph is weakly triangulated, and thus is perfect. For $m > 2$, the conflict graph can contain an antihole of size 6. However, these complete paths can be viewed as continuous functions on the interval 0 to 1, thus by construction, $\mathcal{C}_{\mathcal{P}}$ is perfect [27].

2.3 Summary

The *MWCMS* model has many applications beyond those listed in Section 2.1, which encompass a core set of very difficult problems in computational biology. Our goal for this thesis is to lay out a mathematical modeling framework that allows one to investigate theoretical and computational issues, and to forge new advances for these distinct, but related problems.

CHAPTER III

ALGORITHMS FOR DNA SEQUENCE ANALYSIS

3.1 Introduction

DNA Sequence analysis encompasses the study of DNA sequence in human and other organisms. The rate of innovation in this field has been breathtaking over the last decade, especially with the completion of Human Genome Project. The complete DNA sequence contained in the cell is called genome and its study is called genomics/genome analysis. The purpose of this chapter is to review some well known algorithms that facilitate genome analysis. The material is presented in a way that is interesting to both the specialists working in this area and others. Thus, this review includes a brief sketch of the algorithms to facilitate a deeper understanding of the concepts involved. The list of problems related to genomics is very extensive; hence the scope of this chapter is restricted to the following two related important problems: 1) Phylogenetic Analysis, and 2) Multiple Sequence Alignment. Readers interested in algorithms used in other fields of computational biology are recommended to refer to reviews by Abbas and Holmes(2004) and Blazewicz et al. (2005) [1] [9].

Understanding the DNA sequence is extremely important. It is considered as the blueprint for an organism's structure and function. The sequence order underlies all of life's diversity, even dictating whether an organism is human or another species such as yeast or a fruit fly. It helps in understanding the evolution of mankind, identifying genetic diseases, and creating new approaches for treating and controlling those diseases. In order to achieve these goals, the research in genome analysis has rapidly progressed over the last decade.

The rest of this chapter is organized as follows. Section 3.2 discusses techniques

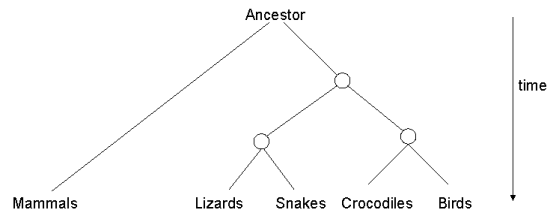


Figure 2: An example of evolutionary tree

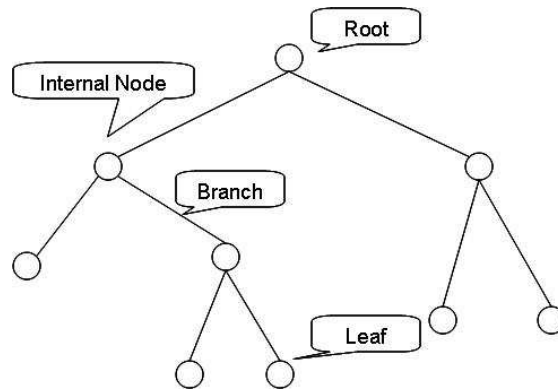


Figure 3: Tree Terminology

used to infer the evolutionary history of species and Section 3.3 presents Multiple Sequence Alignment problem and recent advances. Finally, Section 3.4 summarizes the interdependence and the pivotal role played by the above mentioned two problems in computational biology.

3.2 Phylogenetic Analysis

Phylogenetic Analysis is a major aspect of genome research. It refers to the study of evolutionary relationships of a group of organisms. These hierarchical relationships among organisms arising through evolution are usually represented by a phylogenetic tree (Figure 2). The idea of using trees to represent evolution dates back to Darwin. Both rooted and unrooted tree representation have been used in practice [19]. The branches of tree represents the time of divergence and the root represents the ancestral sequence (Figure 3).

The study of phylogenies and processes of evolution by the analysis of DNA or

amino acid sequence data is called Molecular phylogenetics. In this study, we will focus on methods that use DNA sequence data. There are two processes involved in inferring both rooted and unrooted trees. First is estimating the branching structure or topology of the tree. Second is estimating the branch lengths for a given tree. Currently, there are wide varieties of methods available to conduct this analysis ([60] [88] [18] [21]). These available approaches can be classified into three broad groups: i) Distance Methods; ii) Parsimony Methods; and iii) Maximum Likelihood Methods. Below, we will discuss each of them in detail.

3.2.1 Methods Based on Pairwise Distance

In distance methods, an evolutionary distance d_{ij} is computed between each pair i , j of sequences, and a phylogenetic tree is constructed from these pairwise distances. There are many different ways of defining pairwise evolutionary distance used for this purpose. Most of the approaches estimate the number of nucleotides substitutions per site, but other measures have also been used [78] [77]. The most popular one is Jukes-Cantor distance(1969) which defines d_{ij} as $-\frac{3}{4} \log(1 - \frac{4f}{3})$, where f is the fraction of sites where nucleotides differ in the pairwise alignment [40].

There are a large number of distance methods for constructing evolutionary trees [86]. In this article, we discuss methods based on *cluster analysis* and *neighbor joining*.

3.2.1.1 Cluster Analysis: UPGMA

The conceptually simplest and most known distance method is UPGMA (Unweighted Pair Group Method using Arithmetic averages) developed by Sokal and Michener (1958) [73]. Given a matrix of pairwise distances between each pair of sequences, it starts with assigning each sequence to its own cluster. The distances between the clusters are defined as: $d_{ij} = \frac{1}{|C_i| |C_j|} \sum_{p \in C_i, q \in C_j} d(p, q)$ where C_i and C_j denote sequences in clusters i and j respectively. At each stage in the process, the least distant pair of clusters are merged to create a new cluster. This process continues

until only one cluster is left. Given n sequences, the general schema of UPGMA is shown in Algorithm 1.

Algorithm 1 UPGMA

- 1: INPUT: Distance matrix d_{ij} , $1 \leq i, j \leq n$
 - 2: **for** $i = 1$ to n **do**
 - 3: Define singleton cluster C_i comprising of sequence i
 - 4: Place cluster C_i as a tree leaf at height zero
 - 5: **end for**
 - 6: **repeat**
 - 7: Determine two clusters i, j such that d_{ij} is minimal.
 - 8: Merge these two clusters to form a new cluster k having distance from other clusters defined as the weighted average of the comprised two clusters. If C_k is the union of two clusters C_i and C_j , and if C_l is any other cluster, then $d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}$
 - 9: Define a node k at height $\frac{d_{ij}}{2}$ with daughter nodes i and j
 - 10: **until** just a single cluster remains
-

The time and space complexity of UPGMA is $O(n^2)$, since there are $n - 1$ iterations of complexity $O(n)$. A number of approaches have been developed which are motivated by UPGMA. Li(1981) developed a similar approach which also makes corrections for unequal rates of evolution among lineages [56]. Klotz and Blanken(1981) presented a method where a present-day sequence serves as an ancestor in order to determine the tree regardless of the rates of evolution of the sequences involved [46].

3.2.1.2 Neighbor Joining

Neighbor-Joining(NJ) is another very popular algorithm based on pairwise distances [69]. This approach yields an unrooted tree and overcomes the assumption of UPGMA method that the same rate of evolution applies to each branch.

Given a matrix of pairwise distances between each pair of sequences d_{ij} , it first defines modified distance matrix \bar{d}_{ij} . This matrix is calculated by subtracting average distances to all other sequences from the d_{ij} and thus compensating for long edges. In each stage, the two nearest nodes (minimal \bar{d}_{ij}) of the tree are chosen and defined as neighbors in the tree. This is done recursively until all of the nodes are paired

together.

Given n sequences, the general schema of Neighbor Joining is shown in Algorithm

2.

Algorithm 2 Neighbor Joining

- 1: INPUT: Distance matrix d_{ij} , $1 \leq i, j \leq n$
 - 2: **for** $i = 1$ to n **do**
 - 3: Assign sequence i to the set of leaf nodes of the tree (T)
 - 4: **end for**
 - 5: Set list of active nodes(L) = T
 - 6: **repeat**
 - 7: Calculate modified distance matrix $\bar{d}_{ij} = d_{ij} - (r_i + r_j)$, where $r_i = \frac{1}{|L|-2} \sum_{k \in L} d_{ik}$
 - 8: Find the pair i, j in L having minimal value of \bar{d}_{ij}
 - 9: Define a new node u and set $d_{uk} = \frac{1}{2}(d_{ik} + d_{jk} - d_{ij})$, for all k in L
 - 10: Add u to T joining nodes i, j with edges of length given by: $d_{iu} = \frac{1}{2}(d_{ij} + r_i - r_j)$,
 $d_{ju} = d_{ij} - d_{iu}$
 - 11: Remove i and j from L and add u
 - 12: **until** Only two nodes remain in L
 - 13: Connect remaining two nodes i and j by a branch of length d_{ij}
-

Neighbor Joining has a execution time of $O(n^2)$, like UPGMA. It has given extremely good results in practice and is computationally efficient [69] [79]. Many practitioners have developed algorithms based on this approach. Gascuel(1997) improved the NJ approach by using a simple first-order model of the variances and covariances of evolutionary distance estimates [25]. Bruno et al.(2000) developed a weighted Neighbor Joining using a likelihood-based approach [12]. Goeffon et al.(2005) investigated a local search algorithm under the Maximum Parsimony criterion by introducing a new subtree swapping neighborhood with an effective array-based tree representation [26].

3.2.2 Parsimony Methods

In science, notion of parsimony refers to the preference of simpler hypotheses over complicated ones. In the parsimony approach for tree building, the goal is to identify the phylogeny that requires the fewest necessary changes to explain the differences

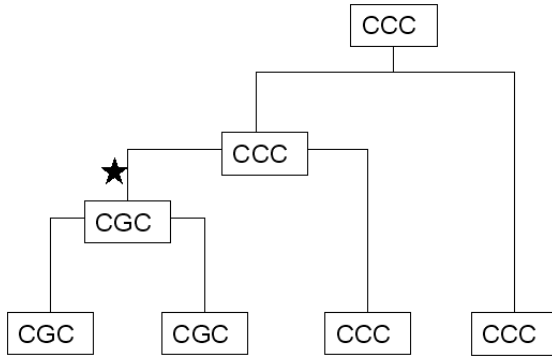


Figure 4: Parsimony tree 1

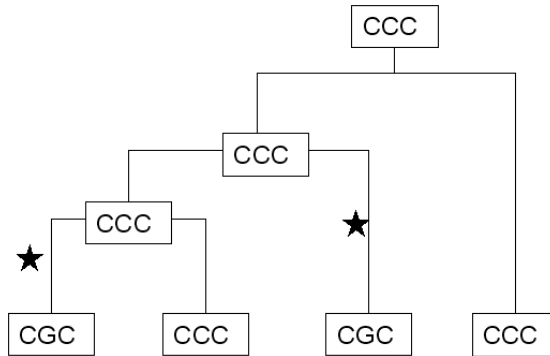


Figure 5: Parsimony tree 2

among the observed sequences. Of the existing numerical approaches for reconstructing ancestral relationships directly from sequence data, this approach is the most popular one. Unlike distance based methods which builds tree, it evaluates all possible trees and gives each a score based on the number of evolutionary changes that are needed to explain the observed sequences. The most parsimonious tree is the one that requires the fewest evolutionary changes for all sequences to derive from a common ancestor [76]. As an example, consider the trees in Figure 4 and Figure 5. The tree in Figure 4 requires only one evolutionary change (marked by the $*$) compare to the tree in Figure 5 which requires two changes. Thus, Figure 4 is the more parsimonious tree.

There are two distinct components in parsimony methods: given a labeled tree, determine the score; determine global minimum score by evaluating all possible trees,

as discussed below.

3.2.2.1 Score Computation

Given a set of nucleotide sequences, parsimony methods treat each site (position) independently. The algorithm evaluates the score at each position and then sums them up over all the positions. As an example, suppose we have the following three aligned nucleotide sequences.

CCC
GGC
CGC

Then, for a given tree topology, we would calculate the minimal number of changes required at each of the three sites and then sum them up. Here, we investigate a traditional parsimony algorithm developed by Fitch (1971), where number of substitutions required is taken as score. For a particular topology, this approach starts by placing nucleotides at the leaves and traverse toward the root of the tree. At each node, the nucleotides common to all of the descendant nodes are placed. If this set is empty then the union set is placed at this node. This continues until root of the tree is reached. The number of union sets equal the number of substitutions required.

The general schema for every position is shown in Algorithm 3.

Figure 6 shows the set R_k obtained by above algorithm. The computation is done for the first site of the three sequences shown above. The minimal score given by the algorithm is 1.

There are a wide variety of approaches developed by modifying Fitch's algorithm [75]. Sankoff and Cedergren(1983) presented a generalized parsimony method which does not just count the number of substitutions, but assigns a weighted cost for each substitution [70].

Algorithm 3 Parsimony: Score Computation

- 1: Each leaf l is labeled with set R_l having observed nucleotide at that position.
 - 2: Score $S = 0$
 - 3: **for all** Internal node k with children i and j having labels R_i and R_j **do**
 - 4: $R_k = R_i \cap R_j$
 - 5: **if** $R_k = \emptyset$ **then**
 - 6: $R_k = R_i \cup R_j$
 - 7: $S = S + 1$
 - 8: **end if**
 - 9: **end for**
 - 10: Minimal score = S
-

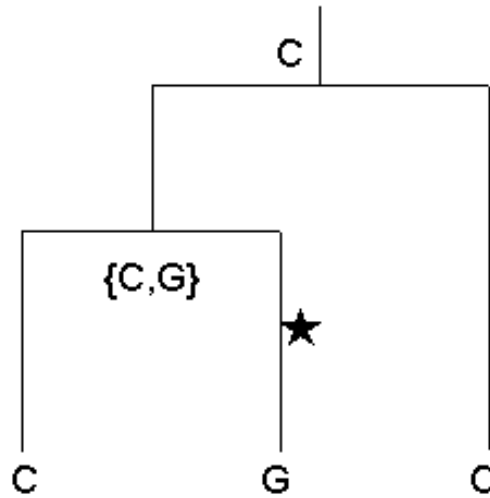


Figure 6: The sets R_k for the first site of given three sequences

Ronquist(1998) improved the computational time by including strategies for rapid evaluation of tree lengths and increase the exhaustiveness of branch swapping while searching topologies [68].

3.2.2.2 Search of possible tree topologies

The number of possible tree topologies dramatically increases with the number of sequences. Consequently, in practice usually only a subset of them are examined using efficient search strategies. The most commonly used is branch and bound methods to select branching patterns [66]. For large scale problems, heuristic methods are typically used [76]. These exact and heuristic tree search strategies are implemented in

various softwares like PHYLIP (Phylogeny Inference Package) and MEGA (Molecular Evolutionary Genetic Analysis) ([22] and [50]).

3.2.3 Maximum Likelihood Methods

The method of maximum likelihood (ML) is one of the most popular statistical tool used in practice. In molecular phylogenetics, maximum likelihood methods find the tree that has the highest probability of generating observed sequences, given an explicit model of evolution. The method was first introduced by Felsenstein (1981)[20]. We discuss herein both the evolution models and the calculation of tree likelihood.

3.2.3.1 Model of evolution

A model of evolution refers to various events like mutation, that changes one sequence to the another over a period of time. It is required to determine the probability of a sequence S_2 arising from an ancestral sequence S_1 over a period time t . Various sophisticated models of evolution have been suggested, but simple models like Jukes-Cantor are preferred in ML methods.

Jukes and Cantor (1969) [40] model assumes that all nucleotides(A, C, T, G) undergo mutation with equal probability, and change to all of the other three possible nucleotides with same probability. If the mutation rate is 3α per unit time per site, the mutation matrix P_{ij} (Probability that nucleotide i changes to j in unit time), takes the form:

$$\begin{pmatrix} 1 - 3\alpha & \alpha & \alpha & \alpha \\ \alpha & 1 - 3\alpha & \alpha & \alpha \\ \alpha & \alpha & 1 - 3\alpha & \alpha \\ \alpha & \alpha & \alpha & 1 - 3\alpha \end{pmatrix}$$

Above matrix is integrated to evaluate mutation rates over time t and then used to calculate $P(nt_2|nt_1, t)$, defined as the probability of nucleotide nt_1 being substituted

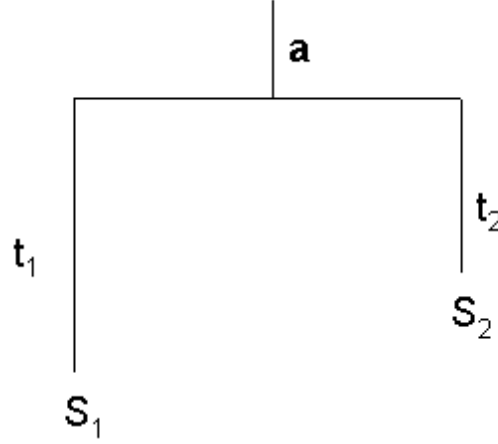


Figure 7: A simple tree

by nt_2 over time t .

Various other evolution models like Kimura Model have also been mentioned in literature. [45] [11].

3.2.3.2 Likelihood of a Tree

Likelihood of tree is calculated as the probability of observing a set of sequences given the tree.

$$L(tree) = Probability[sequences|tree]$$

We begin with the simple case of two sequences S^1 and S^2 of length n having a common ancestor “a” as shown in Figure 7. It is assumed that all different sites(positions) evolve independently, and thus the total likelihood is calculated as the product of likelihood of all sites [17]. Here, the likelihood of each site is obtained using substitution probabilities based on evolution model.

Given q_a = equilibrium distribution of nucleotide a , the likelihood for simple tree in Figure 7 is calculated as $L(tree) = P(S^1, S^2) = \prod_{i=1}^n P(S_i^1, S_i^2)$ where $P(S_i^1, S_i^2) = \sum_a q_a P(S_i^1|a)P(S_i^2|a)$. To generalize this approach for m sequences, it is assumed that diverged sequences evolve independently after diverging. Hence, likelihood for

every node in tree depends only on its immediate ancestral node and a recursive procedure is used to evaluate likelihood of the tree. The conditional likelihood $L_{k,a}$ is defined as the likelihood of the subtree rooted at node k , given that nucleotide at node k is a . The general schema for every site is shown in Algorithm 4. The likelihood is then maximized over all possible tree topologies and branch lengths.

Algorithm 4 Likelihood: Computation at given site

```

1: for all leaf  $l$  do
2:   if leaf has nucleotide  $a$  at that site then
3:      $L_{l,a} = 1$ 
4:   else
5:      $L_{l,a} = 0$ 
6:   end if
7: end for
8: for all Internal node  $k$  with children  $i$  and  $j$  do
9:   Define the conditional likelihood  $L_{k,a} = \sum_{b,c} [P(b|a)L_{i,b}][P(c|a)L_{j,c}]$ 
10: end for
11: Likelihood at given site =  $\sum_a q_a L_{root,a}$ 

```

3.2.3.3 Recent improvements

ML approach has received great attention due to the existence of powerful statistical tools. It has been made more sophisticated using advance tree search algorithms, sequence evolution models, and statistical approaches. Yang(1993) have extended it to the case where the rate of nucleotide substitutions differ over sites [89]. Huelsenbeck(1997) incorporated the improvements in substitution models [36]. Piontkivska(2004) evaluated the use of various substitution models in ML approach and inferred that simple models are comparable in both efficiency and reliability with complex models [65].

Enormously large number of possible tree topologies especially, while working with large number of sequences, makes this approach computationally intensive [79]. It has been proved that reconstructing the ML tree is NP-hard even for certain approximations [16]. In order to reduce computational time, Guindon and Gascuel(2003)

developed a simple hill-climbing algorithm based on the maximum-likelihood principle that adjusts tree topology and branch lengths simultaneously [32]. Recently, parallel computation is being used to address huge computational requirement. Stamatakis et al.(2005) have used OpenMP-parallelization for Symmetric Multi-Processing machines and Keane et al.(2005) developed distributed platform for phylogeny reconstruction by maximum likelihood [42] [74].

3.3 Multiple Sequence Alignment

Multiple sequence alignment (MSA) is arguably among the most studied and difficult problems in computational biology. It has been a vital tool since it compactly represents conserved or variable features among the family members. Alignment also allows character-based analysis compared to distance based analysis and thus helps to elucidate evolutionary relationships better. Consequently, it plays a pivotal role in a wide range of sequence analysis problems like: identifying conserved motifs among given sequences; predicting secondary and tertiary structure of protein sequences; and molecular phylogenetic analysis. It is also used for sequence comparison to find similarity of a new sequence with pre-existing ones. This helps in gathering information about function and structure of new found sequences from the existing ones in databases like GenBank in US and EMBL in Europe.

The MSA problem can be stated formally as follows. Let Σ be the alphabet and let $\hat{\Sigma} = \Sigma \cup \{-\}$, where “-” is a symbol to represent “gaps” in sequences. For DNA sequences, alphabet $\hat{\Sigma} = \{A, T, C, G, -\}$.

An alignment for N sequences S_1, \dots, S_N is given by a set $\hat{S} = \{S_1, \dots, S_N\}$ over the alphabet $\hat{\Sigma}$ which satisfy the following two properties: (1) the strings in \hat{S} are of the same length, (2) S_i can be obtained from \hat{S}_i by removing the gaps. Thus, an alignment in which each string \hat{S}_i has length K can be interpreted as an alignment matrix of N rows and K columns where row i corresponds to sequence S_i . Alphabets

C	C	C	—	C	—	C	C
C	G	G	C	C	G	G	C
C	G	—	C	C	G	C	—

Figure 8: Two possible alignments for given three sequences

that are placed into the same column of alignment matrix are said to be aligned with each other.

Figure 8 shows two possible alignments for given three sequences: $S_1 = CCC$, $S_2 = CGGC$, and $S_3 = CGC$.

For two sequences, optimal MSA is easily obtained using dynamic programming (Needleman-Wunsch algorithm). Unfortunately, the problem becomes much harder for more than two sequences, and optimal solution can be found only for a limited number of sequences of moderate length (approx. 100) [10]. Researchers have tried to solve it by generalizing dynamic programming approach to a multi-dimensional space. However, this approach has huge time and memory requirements and thus cannot be used in practice even for small problems of 5 sequences of length 100 each. This algorithm has been improved by identifying the portion of hyperspace which does not contribute to the solution and exclude it from the computation [13]. But, even this approach of Carrillo and Lipman implemented in MSA program, can only align up to ten sequences [57]. Although, Gupta and Kececioglu (1995) improved the space and time usage of this approach, it cannot align large data sets [33]. To reduce the huge time and memory expenses, wide variety of heuristic approaches for Multiple Sequence Alignment have been developed [63].

There are two components of finding MSA: i) Searching over all the possible multiple alignments, ii) Scoring each of them to find the best one.

The problem becomes more complex for remotely related homologous sequences,

i.e. sequences which are not derived from a common ancestor [29]. Numerous approaches have been proposed, but the quest for an approach which is accurate and fast continues. It must be remembered that even the choice of sequences and calculating the score of alignment is a non trivial task and is an active research field in itself.

3.3.1 Scoring Alignment

There is no unanimous way of characterizing an alignment as the correct one and the strategy depends on biological context. Different alignments are possible and we never know for sure which alignment is correct. Thus, one scores every alignment according to an appropriate objective function and alignments with the higher scores are deemed to be better. A typical alignment scoring scheme consists of the following steps:

3.3.1.1 Independent Columns

The score of alignment is calculated in terms of columns of alignments. The individual columns are assumed to be independent and thus the total score of an alignment is a simple summation over column scores. Thus, score for an alignment $Score(A) = \sum_j Score(A_j)$, where A_j is column j of the multiple alignment A . Now, score for every column j is calculated as “sum of pairs” (SP) function using the Scoring matrices described below. The SP score for column A_j is obtained as $Score(A_j) = \sum_{k < l} Score(A_j^k, A_j^l)$ where A_j^k and A_j^l are nucleotides in column j of alignment corresponding to sequences k and l respectively. If gap costs are linear, $Score(\text{nucleotide}, -)$ and $Score(-, \text{nucleotide})$ will be the insertion cost. But, this approach would not differentiate between opening a gap and extending it. So, affine gap penalties are often used where gap opening and extension penalty are treated as two different parameters. The correct value of both of these parameters is a major concern since their values can be set only empirically [83]. Also most schemes used in practice score columns as

weighted sum of pair-wise substitutions instead of just addition as described before. The weights are decided in accordance with the amount of independent information each sequence possesses [3].

Both the assumption of treating every column independent and using SP score for column has limitations. The problem increases as number of sequences increases.

3.3.1.2 Scoring Matrices

Any alignment can be obtained by performing three evolution operations: insertion, deletion, and substitution. It is assumed that all the different operations occur independently and thus the complete score is evaluated as the sum of scores from every operation. Insertion and deletion scores are calculated as either linear or affine gap penalty. Substitutions scores are stored as substitution score matrix, which contains score for every pair of nucleotides. Thus, these scores $S(A,B)$ can be treated as the score of aligning nucleotide A with B.

These substitution score matrices can be obtained in various ways. One could adopt an ad-hoc approach of setting up a score matrix which produces good alignments for a given set of sequences. The second approach would be more fundamental and look into physical and chemical properties of nucleotides. If two nucleotides are similar in their properties, they would be more likely to be substituted by one another. The third and the most prominent one is a statistical approach where maximum likelihood principle is used in conjunction with probabilistic models of evolution [4].

3.3.2 Alignment Approaches

The number of different approaches for MSA problem has steadily increased over the last decade and thus being exhaustive will not be possible. In this chapter, we will emphasize on the most widely used class of algorithms and the new emerging and most promising approaches.

1. Progressive Alignment Algorithms: Most widely used type of algorithm based on using pairwise alignment information of input sequences. It assumes that input sequences are phylogenetically related, and uses these relationships to guide the alignment [15].
2. Graph Based Algorithms: A new trend where graph based models are used to approach this problem.
3. Iterative Alignment Algorithms: Typically an alignment is produced and is then refined through series of iterations until no more improvement can be made.

3.3.3 Progressive Algorithms

Progressive Alignment constitutes one of the most simplest and effective way for multiple alignment. This strategy was introduced by various researchers like Waterman and Perlwitz(1984) [87]. Among all the progressive algorithms, ClustalW is the most famous one. It is a non-iterative, deterministic algorithm that attempts to optimize the weighted sums-of-pairs with affine gap penalties [81].

The typical progressive algorithm schema is as follows:

- Compute distance between all pairs of given sequences by aligning them. The distances represent divergence of each pair of sequences. These distances could be calculated by fast approximation methods or the slower but more precise methods like complete dynamic programming. Since for given N sequences $\frac{N(N-1)}{2}$ pairwise scores have to be calculated and the scores are used just for construction of a guide tree and not the alignment itself, it is desirable to use approximation methods like k tuple matches.
- Find a guide tree from the distance matrix. This is typically achieved using clustering algorithms discussed in construction of an evolutionary tree. Once

again, since the aim is to get the alignment and not the tree itself, approximation methods are used to construct the evolution trees.

- Align sequences progressively according to the branching order in the guide tree. The basic idea is to start from the leaves of the guide tree toward its root and to use series of pairwise alignments to align larger and larger groups of sequences. Some algorithms have only single growing alignment to which every remaining sequence is aligned whereas other approaches align subgroup of sequences and then merge the alignments.

There are three main shortcomings of the progressive algorithms.

- There does not exist an undisputable “best” way of ordering the given sequences.
- Once a sequence has been aligned, that alignment will not be modified even if it conflicts with sequences added later in the process. Hence, the order in which sequences are added becomes very crucial, and since there is no undisputable best way to order the sequences, this approach returns sub-optimal solutions.
- For a given set of n sequences, $\binom{n}{2}$ pairwise alignments are generated; but while computing the final multiple alignment, most of these algorithms use fewer than n pairwise alignments. Thus, the resulting multiple alignment agrees with only a small amount of information available in the data.

Therefore, there is a growing need for an algorithm to align extremely divergent sequences whose pairwise alignments are likely to be incorrect. In order to address all these issues, some techniques have been developed; while they are innovative, it is understandable that they have their own assumptions and drawbacks.

3.3.4 Graph Based Algorithms

Over the last few years, the field of genomics has undergone evolutionary changes with a rapid increase in new solution strategies. The use of graph based models

is easily seen as one of the most emerging and far-reaching trend. Just and Vedova (2004) use relation between facility location problem and sequence alignment to prove the NP-hardness of MSA [41]. In this section, we review the most prominent integer programming (IP) approaches for finding multiple sequence alignment.

3.3.4.1 Maximum-Weight Trace

Kececioglu et al.(2000) use a solution of the maximum trace problem to construct alignment [43]. The algorithm starts with calculating all pairwise alignments and using them to find a trace. To achieve this, given n sequences, an input alignment graph $G = (V, E)$ is constructed. It is a n -partite graph whose vertex set V represents the characters of given sequences and edge set E represents the pairs of characters matched in the pairwise alignments. The subset of matching in E realized by an alignment is called a trace.

Alignment graph $G = (V, E)$ is extended to a mixed graph $G' = (V, E, A)$ by adding arc set A which connects character of every sequence to the next character in the same sequence. The objective of the algorithm is to find maximum weight trace by finding cycles termed as “critical mixed cycles” in graph G' such that they satisfy sequence alignment properties [67].

The IP model for this problem is formulated as :

$$\text{maximize } \sum_{e \in E} w_e x_e \tag{1}$$

$$\begin{aligned} \text{subject to } \sum_{e \in P \cap E} x_e &= |E \cap P| - 1 \quad \forall \text{ critical mixed cycles } P \text{ in } G' \\ x_e &\in \{0, 1\} \text{ for all } e \in E \end{aligned} \tag{2}$$

An implementation of a branch-and-cut algorithm is used to solve the above problem. Various valid inequalities for the polytope are added as cuts, some of which are

facet-defining. The algorithm is capable of giving an exact solution under the sum-of-pairs objective function with linear gap costs. Kececioglu et al. have made a significant contribution by introducing a polyhedral approach capable of obtaining exact solutions for a subclass of MSA. However, this methodology has its own drawbacks like not being able to capture the order of insertions and deletions between two matchings and affine gap costs. Recently, Althaus et al. (2006) has proposed a general model using this approach in which arbitrary gap costs are allowed [2].

3.3.4.2 Minimum-Spanning Tree and Traveling Salesman Problem

Shyu et al.(2004) explore the use of minimum spanning trees to determine the order of sequences [72]. The idea of the approach is to preserve the most informative distances among the set of given sequences. The criterion used is meaningful and capable of working better than the traditional criteria like those in sum-of-pairs. The algorithm itself is very efficient for practical usage, and can be easily implemented. However, it fails to address the issue of using all the information in pairwise alignments, since it only uses the score and not the pairwise alignments themselves. Moreover, this approach has all the drawbacks of the progressive strategy.

A similar approach has also been developed by Korostensky and Gonnet (1999) using Traveling Salesman Problem (TSP) [47]. In this technique, a circular sum measure is used instead of sum of pairs score. The cities in TSP correspond to the sequences and the scores of pairwise alignment are taken as the distances. The problem is to find the longest tour where each sequence is visited exactly once [48].

3.3.4.3 Eulerian Path approach

Zhang and Waterman (2003) proposed a new approach motivated by the Eulerian method for fragment assembly in DNA sequencing [90]. In their work, a consensus sequence is found and later pairwise alignments are obtained between each input sequence and consensus sequence. Finally, MSA is obtained according to these pairwise

alignments. The most significant advantage of this method is linear time and memory cost for finding the consensus sequence. And, if the consensus sequence is the closest one to all given sequences, good quality alignment can be obtained in a reasonable amount of time. Once again, this approach suffers from the prominent drawback of the progressive strategy and issues in graph formation while finding the consensus sequence.

3.3.5 Iterative Algorithms

The main shortcoming of the progressive strategy is the failure to remove errors in the alignment, which are introduced early. The iterative algorithms are developed precisely to overcome this flaw. They are based on the idea of reconsidering and re-aligning previously aligned sequences with the goal of improving the overall alignment score. Each modification step is an iteration to improve the quality of the alignment.

These available approaches can be classified into two broad categories: probabilistic iterative algorithms, and deterministic iterative algorithms. We will briefly discuss them below.

3.3.5.1 Probabilistic Algorithms

We will discuss both the traditional probabilistic optimization approaches like genetic algorithm and relatively recent approaches based on bayesian idea.

- *Simulated Annealing and Genetic Algorithm*

Simulated Annealing(SA) and Genetic Algorithm(GA) are very popular stochastic methods for solving complex optimization problems. While they are often viewed as separate and competing paradigms, both of them are iterative algorithms which search for new solutions “near” to already known good solutions. The fundamental difference between SA and GA is that SA performs a local move only on one solution to create a new solution whereas GA also creates

solutions by combining information from two different solutions. Performance comparison between SA and GA varies with the problem and representation used.

The algorithms starts with an initial alignment and alignment score is taken to be the objective function [62]. Various operations like mutation, insertion and substitution constitutes the local move which are used to get new solution from existing ones. Flexibility in scoring systems and ability to correct for errors introduced during early phase makes these approaches desirable [44].

- *Hidden Markov Model and Gibbs Sampler*

Hidden Markov Model (HMM) and Gibbs Sampler are relatively recent approaches which views MSA in a statistical context. Both of them use the central Bayesian idea of simultaneously maximizing the data and the model. Gibbs sampler find motifs using local alignment techniques [52]. It is essentially similar to HMM with no insert and delete states.

HMM is a statistical model based on Markov Process which has gained importance in various fields related to pattern recognition. It determines the hidden parameters of the system based on the observable parameters of the model. For MSA, HMM model consists of three types of states: match states, insert states, and delete states [49]. Each state has its own emission probability of nucleotides and transition probability to other states. The standard expectation-maximization (EM) algorithm or gradient descent algorithms are used to train the model and evaluate the parameters.

Although HMM has been successfully used in other areas, it faces a lot of challenges. There need to be some minimum number of sequences (approx. 50) required to train the model and HMM can be easily trapped in local optima like other hill climbing approaches [37].

3.3.5.2 Deterministic Algorithms

A deterministic iterative algorithm starts with an initial alignment and then attempts to improve it. This helps in overcoming the drawback of progressive alignment strategy where partial alignments are “frozen” [6]. Typical schema is as follows:

- Given N sequences S_1, S_2, \dots, S_N , find alignment A
- Remove sequence S_1 from alignment A and realign it to the profile of other aligned sequences S_2, \dots, S_N to get new alignment A'
- Calculate the score of the new alignment A' and if better replace A by A'
- Remove sequence S_2 from A' and realign it. Continue this procedure for S_3, \dots, S_N
- Repeat the realignment steps until alignment score converges or number of iterations reaches the user specified limit.

Many iteration strategies which enable very accurate alignments have been developed [85]. The aim is to reduce the greedy nature of the algorithm and avoid getting trapped in a local optima. One approach is to remove and realign every sequence to the rest in each iteration. Then, the alignment with the best score is taken to be the input for the next iteration. The other famous approach is to randomly split set of sequences into two sets, which are then realigned.

Some researchers have incorporated the iterative strategy in progressive alignment procedure itself. For instance, a double iteration loop has been used to make the alignment, guide tree and sequence weights mutually consistent [28]. Recently, Chakrabarti et al.(2006) have developed an approach which provides a fast and accurate method for refining existing block-based alignments [14].

3.4 *Summary*

Multiple Sequence Alignment and Phylogenetic Analysis are deeply interconnected problems in computational biology. A good multiple alignment is crucial for reliable reconstruction of the phylogenetic tree [64]. On the other hand, most of the multiple alignment methods require a phylogenetic tree as the guide tree for progressive iteration.

Thus, the evolutionary tree construction might be biased by the guide tree used for obtaining the alignment. In order to avoid this pitfall, various algorithms have been developed which simultaneously find alignment and phylogenetic relationship among given sequences. Sankoff and Cedergren (1983) developed a parsimony-based algorithm using a character-substitution model of gaps [70]. The algorithm is guaranteed to find evolutionary tree and alignment which minimizes tree-based parsimony cost. Hein (1989) also developed a parsimony-type algorithm but uses an affine gap cost which is more realistic than the character-substitution gap model [34]. This algorithm is also faster than Sankoff and Cedergren's approach but makes simplifying assumptions in choosing ancestral sequences.

Like parsimony methods for finding a phylogenetic tree, both of the above approaches require search over all possible trees to find the global optimum. This makes these algorithms computationally very intensive. Hence, there has been a strong focus on developing an efficient algorithm that considers both alignment and tree. Vingron and Haeseler (1997) have developed an approach based on three-way alignment of pre-aligned groups of sequences. It also allows change in the alignment made early in the course of computation [82]. Many software, like MEGA, are trying to develop an efficient integrated computing environment that allows both sequence alignment and evolutionary analysis [51].

In this thesis, we address this issue of simultaneously finding alignment and phylogenetic relationships by presenting our novel graph-theoretical approach. Indeed,

our model can be easily tailored to find theoretically provable optimum solutions to a wide range of crucial DNA sequence analysis problems.

CHAPTER IV

COMPLEXITY THEORY

In this chapter, we present multi-layer graph construction and the complexity results that was first introduced by Lee et al. [54] and reported in [53]. We present their constructs *supernodes* and *multi-layer supergraphs* that allows to model the MWCMS problem and perform theoretical investigations into the complexity of this problem.

Recall that the notation $ord(S, B)$, $w(ord(S, B))$, $ord^*(S, B)$, and the formal definition of problem MWCMS were given in the previous chapter. As an optimization problem, MWCMS can be stated as follows. Given a set of input sequences, problem MWCMS seeks to mutate every input sequence to the same a priori unknown sequence using the operations of insertion, deletion and substitution; weights are assigned for each operation, and the total weight associated with all mutations is to be minimized. Levenshtein first considered a special case of this problem by changing a single input sequence to another sequence using insertions, deletions and substitutions [55]. Our study involves changing multiple input sequences to arrive at an a priori unknown common sequence.

Given positive weights η , δ and ψ corresponding respectively to insertions, deletions and substitutions and any two sequences S and B , clearly any $ord^*(S, B)$ will never contain more than $|B|$ insertions or substitutions. Proving that MWCMS is in \mathcal{NP} is not obvious. While one can transform MWCMS to special applications as described in previous chapter to conclude that it is in NP, here Lee et al. prove it directly for the general case. One needs to be able to evaluate $d(S, B)$ in polynomial time for any two sequences S and B . They construct a graph that can be used to establish the existence of a polynomial time algorithm for obtaining $d(S, B)$. The

constructs and arguments used here typify those used to establish many of the results presented by Lee et al.. It is noteworthy that the notions of both conflict graph and perfect graph come into play.

Let Σ be a finite alphabet, and define Σ -cross to be a directed bipartite graph consisting of $|\Sigma|$ vertices in each bipartition such that each vertex in the bipartition represents a distinct element in Σ . There is an arc between two vertices if the vertices correspond to the same element in Σ , and the geometric layout is rigidly constructed so that every arc crosses every other arc. This graph will be used as a “supernode” for insertion and substitution operations in our model. Figure 9 shows an example for Σ -cross when $\Sigma = \{A, C, G, T\}$.

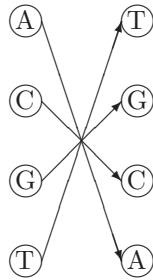


Figure 9: An example of Σ -cross when $\Sigma = \{A, C, G, T\}$.

4.1 Graph Construction and MWCMS complexity

Lee et al. construct a *3-layer supergraph*, G_L , using the sequences S and B along with the Σ -cross graphs. Layers 1 and 2 consist of exactly $|B|(|S| + 1) + |S|$ Σ -crosses. The first $|B|$ Σ -crosses represent potential insertions before the first letter in S . The next Σ -cross represents either the first letter of S or a substitution of this letter. The next $|B|$ Σ -crosses represent potential insertions between the first and second letters of S . And this is followed by a Σ -cross representing either the second letter of S or a substitution of this letter. This continues for each letter in S with the final $|B|$ Σ -crosses representing up to $|B|$ insertions after the last letter in S . Each Σ -cross is

called either an *insertion supernode* or a *substitution supernode*, according to what it represents. The weight of all of the arcs in an insertion supernode is η . An arc in a substitution supernode has weight $-\delta$ if the arc represents the original letter in the sequences, or $\psi - \delta$ if the arc represents a substitution of the original letter. Layer 3 consists of the vertices represented by B . A vertex in layer 2 is connected to a vertex in layer 3 if they have the same letter. The weight of every arc between layers 2 and 3 is $M \leq -(\eta + \delta + \psi)$. A sample of a 3-layer supergraph is given in Figure 10. The bold arcs are used to denote the original letters in S (the weight of these arcs is $-\delta$). For simplicity, we omit the first two insertion supernodes before the first letter G. The first supernode thus represents the letter G from the original sequence which allows for substitution. The second and third supernodes correspond to insertion supernodes, and the fourth supernode corresponds to the letter C and allows substitution as well. There are two more insertion supernodes which are omitted from the graph.

The main step in proving $d(S, B)$ to be polynomial-time solvable for any sequences S and B involves the use of the conflict graph as defined in Definition 2.2.1. We state some theoretical results by Lee et al. below.

Lemma 4.1.1. *The following statements are equivalent:*

- (i) *There exists a conversion from S to B using no more than a total of $|B|$ insertions or substitutions.*
- (ii) *There exists a set of noncrossing complete paths in the associated 3-layer supergraph G_L of size $|B|$.*
- (iii) *There exists a node packing of size $|B|$ in the associated conflict graph C .*

Proof. (i) \Rightarrow (ii) Let $ord'(S, B)$ be any conversion of S to B using no more than $|B|$ insertions or substitutions. There are 3 individual operations in $ord'(S, B)$. First, if an element is inserted in S , then let p be a path through the first available insertion supernode (using the appropriate letter) to the corresponding letter in B . If an element is deleted from S , then no path is generated. However, if an original letter

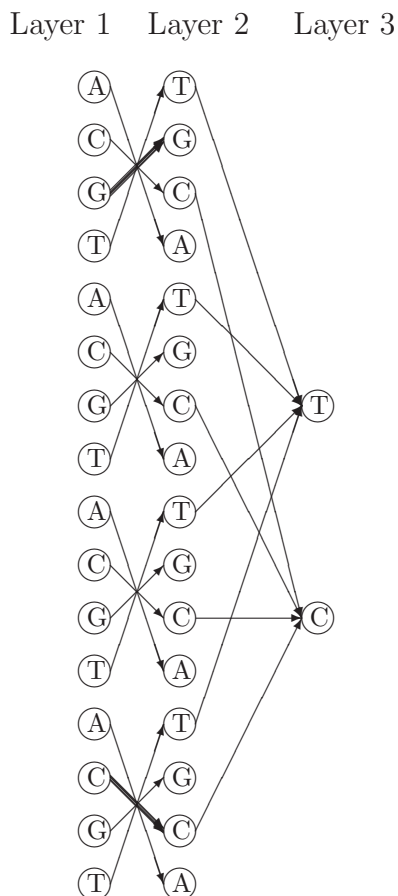


Figure 10: An example of the 3-layer supergraph for converting the sequence $S = GC$ to $B = TC$.

is not deleted, then a path p is generated by including the 2 vertices corresponding to the original letter in the substitution supernode and continues to the appropriate vertex in layer 3. Finally, a substitution can be handled in a similar manner. Due to the construction of layers 1 and 2, there will always be a sufficient number of paths available with the property that none of the paths cross.

(ii) \Rightarrow (iii) Let P be any set of noncrossing complete paths of size $|B|$ in G_L . Each path $p \in P$ is represented by a vertex $v_p \in V(\mathcal{C})$. Since p and q do not cross for every $p, q \in P$, the associated vertices v_p and v_q are not adjacent. Thus, $\{v_p : p \in P\}$ is a node packing in \mathcal{C} of size $|B|$.

(iii) \Rightarrow (i) Finally, let N represent any node packing in \mathcal{C} of size $|B|$. Form $ord'(S, B)$ by following the representation given by N . That is, each path contains one element

from B and one from the new S . This “new” element corresponds to an insertion, substitution or an original element. If an original letter is not represented in a vertex of N , then it is deleted in $ord'(S, B)$. Clearly, no more than $|B|$ insertions or substitutions are included in $ord'(S, B)$ since $|N| = |B|$. \square

Lemma 4.1.2. *Calculating $d(S, B)$ for any sequences S and B can be accomplished in polynomial time.*

Proof. Given sequences S and B generate the 3-layer supergraph G_L as indicated. The number of vertices in G_L is $O(|\Sigma||B||S|)$. From G_L generate all complete paths (paths from layer 1 to layer 3) and create a conflict graph \mathcal{C} . The number of complete paths corresponds to the number of arcs from layer 2 to layer 3. Therefore, \mathcal{C} is polynomial in the size of S and B . By construction, \mathcal{C} is perfect [27].

Finding a minimum weight node packing N on \mathcal{C} can be accomplished in polynomial time [31]. The claim is that N provides the solution to $d(S, B)$. Due to the weights between layers 2 and 3, a minimum weight node packing must contain exactly $|B|$ elements (obviously no more than $|B|$ elements are possible in any node packing) or else there is a contradiction to minimality. From Lemma 3.1, N corresponds to a series of insertions, deletions and substitutions to convert S to B ($ord'(S, B)$). By minimality, N achieves the smallest weight. Adding $|S|\delta + |B|M$ provides the same value as $w(ord'(S, B))$. Since $|S|\delta + |B|M$ is a constant for all node packings of size $|B|$, N provides the minimal solution to $d(S, B)$. Clearly, every step is polynomial in the size of S and B as indicated at each nontrivial step. \square

The 3-layer supergraph can be generalized to multi-layer when multiple sequences are considered. Clearly, such multi-layer supergraphs are much too large for practical purposes, yet polynomiality is preserved in the construction, and it is therefore sufficient. We can now arrive at the result that MWCMS is in \mathcal{NP} .

Theorem 4.1.3. *MWCMS is in \mathcal{NP} .*

Proof. Given a collection of sequences S_1, \dots, S_m , positive weights η, δ, ψ , threshold k , and a candidate solution B , using Lemma 3.2, calculating $d(S_i, B)$ for $i = 1, \dots, m$ requires polynomial effort. Therefore, checking whether $\sum_{i=1}^m d(S_i, B) \leq k$ can be accomplished in polynomial time, and thus, MWCMS is in \mathcal{NP} . \square

To prove that MWCMS is polynomial-time solvable when the number of input sequences is bounded by a positive constant, the following lemma is crucial, though trivial.

Lemma 4.1.4. *Given $\eta, \delta, \psi \in \mathfrak{R}^+$, an optimal solution B to any MWCMS problem has the following properties. B has no substitutions from letters other than the original letters in an S_i , and B will never have an element which is inserted in every sequence (in the same location). Therefore, there are at most $\sum_{i=1}^m |S_i|$ insertions in any sequence.*

In addition, we also require the construction of a (directed) $2m$ -layer supergraph, G_L^m , similar to the 3-layer supergraph, G_L .

Given sequences S_1, \dots, S_m , generate a $2m$ -layer (directed) graph $G_L^m = (V, E)$ as follows. Layers $2i - 1$ and $2i$ consist of $(\sum_{j=1}^m |S_j|)(|S_i| + 1) + |S_i|$ copies of Σ -crosses for $i = 1, \dots, m$, constructed in exactly the same manner as layers 1 and 2 of the 3-layer supergraph using the input sequence S_i . The first $\sum_{j=1}^m |S_j|$ Σ -crosses represent the possibility that $\sum_{j=1}^m |S_j|$ different letters can be inserted before the first element in S_i . The next Σ -cross corresponds to either the first letter in S_i or a substitution of this letter. This is repeated $|S_i|$ times (for each letter in S_i), and the final $\sum_{j=1}^m |S_j|$ Σ -crosses represent insertions after the final letter in S_i . Thus, the first $\sum_{j=1}^m |S_j|$ Σ -crosses represent the insertion supernodes, followed by one Σ -cross representing a letter in S_i or a substitution supernode, and so forth. An arc exists from a vertex in layer $2i$ to a vertex in layer $2i + 1$ if the vertices correspond to the same letter. Observe that G_L^m is an acyclic directed graph which is polynomial in the

size of the input sequences. Assign every arc between layers $2i$ and $2i + 1$ a weight of 0. There are three different weights for arcs between layers $2i - 1$ and $2i$ each corresponding to an insertion, deletion or substitution. The assignment of weights on such arcs is analogous to the assignment in G_L : a weight of η is assigned to every arc contained in an insertion supernode; and an arc in a substitution supernode is assigned a weight of $-\delta$ if it corresponds to the original letter, or $\psi - \delta$, otherwise.

Figure 11 shows a sample graph for two sequences: $S_1 = GC$ and $S_2 = TG$. Observe that at most two insertions are needed in an optimal solution; thus we can reduce the number of Σ -crosses as insertion supernodes from $\sum_{i=1}^2 |S_i| = 4$ to 2. For simplicity, in the graph shown in Figure 11, we have not included the two insertion supernodes before the first letter nor those after the last letter of each sequence. Thus, in the figure, the first Σ -cross represents the substitution supernode associated with the first letter in S_1 . The second and third Σ -crosses represent two insertion supernodes. And the last Σ -cross represents the substitution supernode associated with the second letter in S_1 . For simplicity, we include only arcs connecting vertices associated to the element G between layers 2 and 3. The arcs for other vertices follow similarly.

A conflict graph \mathcal{C} associated with G_L^m can be generated by finding all complete paths (paths from layer 1 to layer $2m$) in G_L^m . These complete paths correspond to the set of vertices in \mathcal{C} , as in Definition 2.2.1. If we assign a weight to each vertex equal to the weight of the associated complete path, then the following result can be established.

Theorem 4.1.5. *Every node packing in \mathcal{C} represents a candidate solution to MWCMS if and only if at most $\sum_{i=1}^m |S_i|$ letters can be inserted between any two original letters. Furthermore, the weight of the node packing is equal to the weight of the MWCMS – $\sum_{i=1}^m |S_i| \delta$.*

Proof. Let N be a node packing of \mathcal{C} . Since N represents a set of noncrossing complete

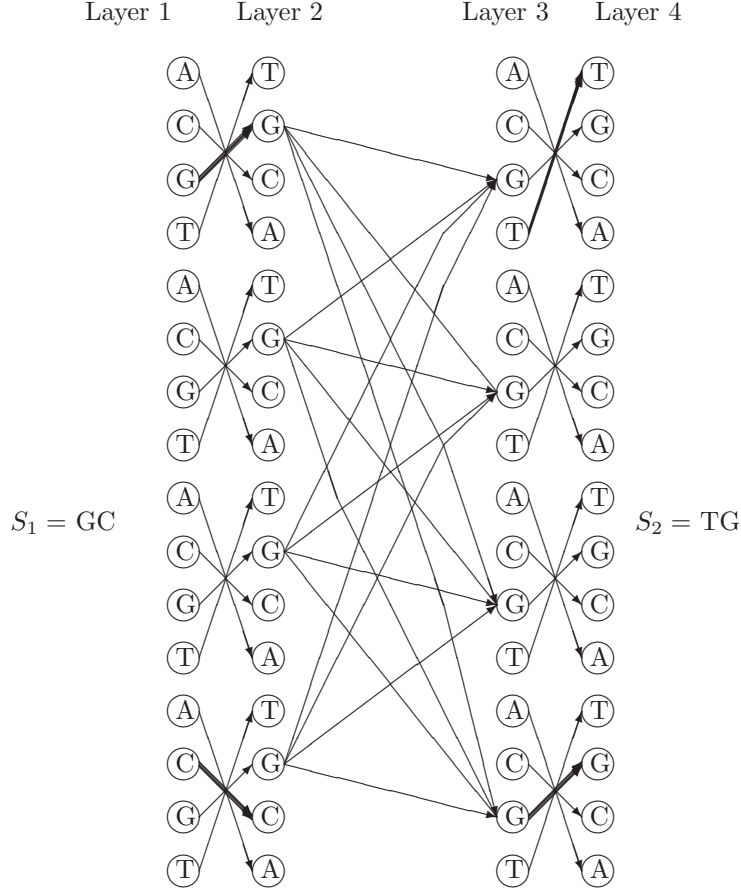


Figure 11: A sample graph G_L^m of MWCMS with $S_1 = GC$ to $S_2 = TG$ where $\Sigma = \{A, C, G, T\}$

paths, it induces a candidate solution B of MWCMS. Furthermore, by construction of G_L^m , there are no more than $\sum_{i=1}^m |S_i|$ insertions between any two original letters. Let the weight of N be $p(-\delta) + q(\psi - \delta) + r\eta$. Then the corresponding B has exactly p original letters, q substitutions and r insertions. Thus, $\sum_{i=1}^m d(S_i, B) = q\psi + r\eta + (\sum_{i=1}^m |S_i| - p - q)\delta = p(-\delta) + q(\psi - \delta) + r\eta + \sum_{i=1}^m |S_i|\delta = w(N) + \sum_{i=1}^m |S_i|\delta$ as required.

Conversely, let B be a solution to MWCMS. Because there are no more than $\sum_{i=1}^m |S_i|$ inserted letters in B between any two original letters, B can be represented by complete noncrossing paths in G_L^m . Thus, there exists a collection of nonadjacent vertices N in \mathcal{C} which correspond directly to B . Thus, N is a node packing. As

such, let B consist of p deletions, q substitutions ($nm - q$ original letters) and r insertions. Then $\sum_{i=1}^m d(S_i, B) = p\delta + q\psi + r\eta$. Minusing $\sum_{i=1}^m |S_i|\delta$ results in $(q + p - \sum_{i=1}^m |S_i|)\delta + q(\psi - \delta) + r\eta = w(N)$ as required. \square

The supergraph G_L^m and its associated conflict graph are fundamental to our proof of the following theorem on polynomial-time solvability of a restricted version of problem MWCMS.

Theorem 4.1.6. *Problem MWCMS restricted to instances for which the number of sequences is bounded by a positive constant is polynomial-time solvable.*

Proof. Since $\eta > 0$, no more than $\sum_{i=1}^m |S_i|$ insertions will be performed on any sequence. First generate G_L^m as specified previously. Observe that each sequence S_i will be represented by exactly $(|S_i| + 1) \sum_{j=1}^m |S_j| + |S_i|$ Σ -crosses. Therefore, the number of vertices in every layer is at most $(mn(n + 1) + n)|\Sigma|$ where $n = \max\{|S_i| : i \in \{1, \dots, m\}\}$. As indicated, generate the conflict graph \mathcal{C} from G_L^m by finding all complete paths (paths from layer 1 to layer $2m$) in G_L^m . The number of complete paths is bounded above by $[((m - 1)n(n + 1) + n)|\Sigma|]^{2m}$ (this can actually be reduced to $[((m - 1)n(n + 1) + n)|\Sigma|]^m$ easily, but the former is sufficient), where $n = \max\{|S_i| : i \in \{1, \dots, m\}\}$. Thus, \mathcal{C} can be generated in polynomial time since m is a constant.

Since \mathcal{C} is the conflict graph constructed from complete paths, which can be viewed as continuous functions on $[0, 1]$, \mathcal{C} is the complement of a comparability graph and is perfect [27]. Therefore, a minimum weight node packing can be found in polynomial time [31]. Using Theorem 4.1.5, such a node packing generates the best solution to MWCMS with an associated weight equal to the weight of the node packing plus $\sum_{i=1}^m |S_i|\delta$ (a constant), which establishes the result. \square

4.2 Special Cases of MWCMS

MWCMS encompasses a very broad class of problems. In computational biology, first and foremost, it represents a model for phylogenetic analysis. MWCMS as defined is the “most likely ancestor problem”, and the concept of 3-layer supergraph as described in Section 4.1 describes the evolutionary distance problem. An optimal solution to a multiple sequence alignment instance can be found using the solution of the MWCMS problem obtained on the $2m$ -layer supergraph, G_L^m . The alignment is the character matrix obtained by placing together the given sequences incorporating the insertions into the solution of the MWCMS problem. Furthermore, DNA sequencing can be viewed as the shortest common superstring problem, while sequence comparison of a given sequence B to a collection of N sequences S_1, \dots, S_N is the MWCMS problem itself.

Below are some special cases of MWCMS that are of interest in their own right as combinatorial optimization problems and for their role in complexity theory. In the discussion, let Σ be a finite alphabet and Σ^* be the set of finite strings of elements in Σ .

4.2.1 Shortest Common Supersequence

The Shortest Common Supersequence problem is formally defined as:

(SCSQ) Given a finite alphabet Σ , finite sequences $S_1, \dots, S_m \in \Sigma^*$ and a positive integer κ , does there exist a sequence $B \in \Sigma^*$ with $|B| \leq \kappa$ such that each S_i is a subsequence of B for $i = 1, \dots, m$?

SCSQ is \mathcal{NP} -complete even if $|\Sigma| = 5$ [58]. In addition, few special cases of SCSQ have known polynomial-time algorithms. Two exceptions follow. If SCSQ is restricted to instances in which there are exactly two sequences S_1 and S_2 , then the problem is solvable in polynomial time by computing the longest common subsequence and then inserting the obvious letters. SCSQ can also be solved in polynomial time if $|S_i| \leq 2$

for all $i = 1, \dots, m$. Some negative results regarding the complexity of approximation algorithms can be found in Jiang and Li [39]. Using the results of Theorem 4.1.6, we now extend the class of polynomial-time solvable cases to include a constant number of input sequences.

An alternate view of the SCSQ problem is to minimize the number of insertions (using no deletions or substitutions) required to make every input sequence identical. The following theorem demonstrates this.

Theorem 4.2.1. *An instance of SCSQ can be obtained by solving an MWCMS problem using the same input sequences with weights assigned as $\eta = 1$ and $\delta = \psi = M$, where $M > (m - 1) \sum_{i=1}^m |S_i|$.*

Proof. Let S_1, \dots, S_m be a collection of input sequences. Let B^* be an optimal solution to MWCMS with weights assigned as above, and let $v^* = \sum_{i=1}^m d(S_i, B^*)$. A candidate solution for SCSQ is $B' = S_1 S_2 \dots S_m$. The weight of B' is $(m - 1) \sum_{i=1}^m |S_i| \geq v^*$. Since $M > (m - 1) \sum_{i=1}^m |S_i|$, no deletions or substitutions can be included in B^* . Thus, each S_i is a subsequence of B^* . Since B^* attains the minimum weight, the number of insertions is smallest. Therefore, B^* is as short as possible and is the optimal solution to SCSQ. \square

The following polynomial time result for SCSQ now follows directly using Theorems 4.1.6 and 4.2.1.

Corollary 4.2.2. *Problem SCSQ restricted to instances for which the number of sequences is constant is polynomial-time solvable.*

MWCMS can now easily be shown to be \mathcal{NP} -complete by using Theorem 4.1.3, Theorem 4.2.1 and the SCSQ \mathcal{NP} -completeness result by Maier [58].

Corollary 4.2.3. *MWCMS is \mathcal{NP} -complete.*

4.2.2 Longest Common Subsequence

The Longest Common Subsequence problem is well-known. Formally, it is defined as:

(LCS) Given a finite alphabet Σ , a finite set of sequences S_1, \dots, S_m in Σ^* , and a positive integer κ , does there exist a sequence $B \in \Sigma^*$ with $|B| \geq \kappa$ such that B is a subsequence of each S_i for $i \in \{1, \dots, m\}$?

LCS is \mathcal{NP} -complete [58]. However, if there are a constant number of sequences, then the problem can be solved in polynomial time by dynamic programming [84]. Bellare and Sudan (1994) proved that the solution is not approximable within $|\Sigma|^{\frac{1}{6}-\epsilon}$ for any $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$ [7].

An alternate view of LCS is to minimize the number of deletions required to make every sequence identical. Thus, LCS is a special case of the MWCMS problem as the following theorem shows.

Theorem 4.2.4. *An optimal solution to an LCS instance can be found by solving an MWCMS problem using the same input sequences and weights of $\delta = 1$, $\eta = \psi = M$ for some $M > \sum_{i=1}^m |S_i|$.*

Proof. Let S_1, \dots, S_m be a collection of input sequences. Let B^* be an optimal sequence which solves the MWCMS instance with weights as specified above. Let $v^* = \sum_{i=1}^m d(S_i, B^*)$. Let B' denote the empty string. Since B' can be obtained from every S_i by deleting every element in S_i , the weight of B' is $\sum_{i=1}^m |S_i| \delta = \sum_{i=1}^m |S_i| < M$. Thus, no insertions or substitutions can be contained in B^* . Therefore, B^* is a subsequence of each S_i . Trivially, the longest subsequence B^* minimizes the number of deletions to convert every S_i to B^* for $i = 1, \dots, m$ and the result follows. \square

Using Theorems 4.1.6 and 4.2.4, there is now an alternate (besides dynamic programming) polynomial-time algorithm for solving LCS when a constant number of sequences is given as input.

Corollary 4.2.5. *Problem LCS restricted to instances for which the number of sequences is constant is polynomial-time solvable.*

4.2.3 Shortest Common Superstring

The next natural problem to address is the well-known *Shortest Common Superstring* (SCST) problem. Formally, this can be stated as follows:

(SCST) Given finite strings S_1, \dots, S_m from a finite alphabet Σ , and a threshold κ , does there exist a string $B \in \Sigma^*$ such that $|B| \leq \kappa$ and each S_i is a substring of B (i.e., $B = B'_i S_i B''_i$ for each $i = 1, \dots, m$, where $B'_i, B''_i \in \Sigma^*$)?

SCST remains \mathcal{NP} -complete even if $|\Sigma| = 2$ or if all S_i have $|S_i| \leq 8$ and contain no repeated letters [59]. As is the case for SCSQ, there are a few restricted classes with known polynomial time algorithms. Instances are solvable in polynomial time if the size of each sequence is limited to be at most 2. In addition, the optimal solution is approximable within 2.89 [80].

Unlike LCS and SCSQ, MWCMS does not solve SCST directly by setting specific weights for η , δ , and ψ . However, two slight modifications can be made to the modeling of MWCMS so that SCST can be solved by the construction of the multilayer supergraph.

The first modification is to allow each insertion, substitution and deletion to have an individual weight. This leads to a more general problem, the Minimum Independent Weight Common Mutated Sequence (MIWCMS) problem. As is the case for MWCMS, MIWCMS seeks a sequence B such that changing every input sequence to B requires the minimum total weight, where each insertion, deletion and substitution is individually weighted. Let δ^{ij} (ψ^{ij}) be the weight of a deletion (substitution) of the i^{th} letter in the j^{th} sequence, and η^{ijk} be the weight of the k^{th} insertion between the i^{th} and $(i+1)^{th}$ letter in the j^{th} sequence. Each of these weights must be nonnegative and $\eta^{ijk} \leq \eta^{ijk+1}$ for all i, j, k . The formal definition of the problem is left to the

reader; it follows closely to the formal definition of MWCMS.

Lee et al. remark that the proof of Theorem 4.1.3 is weight independent, and so the results obtained for MWCMS are also valid for MIWCMS. As is the case for LCS and SCSQ, assigning appropriate weights to an instance of MIWCMS solves SCST.

Assign $\delta^{ij} = \psi^{ij} = \sum_{\ell=1}^m |S_\ell|/m$ for all i and j ; $\eta^{ijk} = \sum_{\ell=1}^m |S_\ell|/m$ for all $i = 1, \dots, n-1, j = 1, \dots, m$, and all k ; and $\eta^{ijk} = 1$ for all $i \in \{0, n\}, j = 1, \dots, m$, and all k . From input sequences S_1, \dots, S_m , the string $B = S_1 S_2 \dots S_m$ is a candidate solution with total weight strictly less than $\sum_{\ell=1}^m |S_\ell|/m$. Therefore, by construction, the optimal solution to this MIWCMS instance will have no deletions, substitutions or insertions between any two letters. Thus, any optimal solution to the above MIWCMS instance must satisfy the conditions of SCST, and so the multilayer supergraph technique solves SCST.

An alternate modification is to change the design of the multilayer supergraph used to model MWCMS. The alternate supergraph, G' , is formed by first generating the multilayer supergraph as previously described, and then removing all insertion supernodes between any two substitution supernodes, and finally collapsing each substitution supernode to a single arc connecting its original letters (i.e., remove all possibilities of a substitution). An example supergraph G' is given in Figure 12. Set the weight of an insertion to be 1 and the weight of a deletion to be very large: $\delta \geq \sum_{i=1}^m |S_i|/m$. As in the case for SCSQ, a feasible solution is $S_1 S_2 \dots S_m$, which has weight less than or equal to a single deletion. Thus, an optimal solution to the multilayer noncrossing complete path problem on G' solves SCST.

Using either of the above two transformations results in the following lemma.

Lemma 4.2.6. *Given a constant number of sequences as input, SCST can be solved in polynomial time by the multilayer supergraph method.*

Lee et al. remark that their theoretical results extend to the case when arcs are constructed such that they are within k relative distance apart, as defined in Chapter

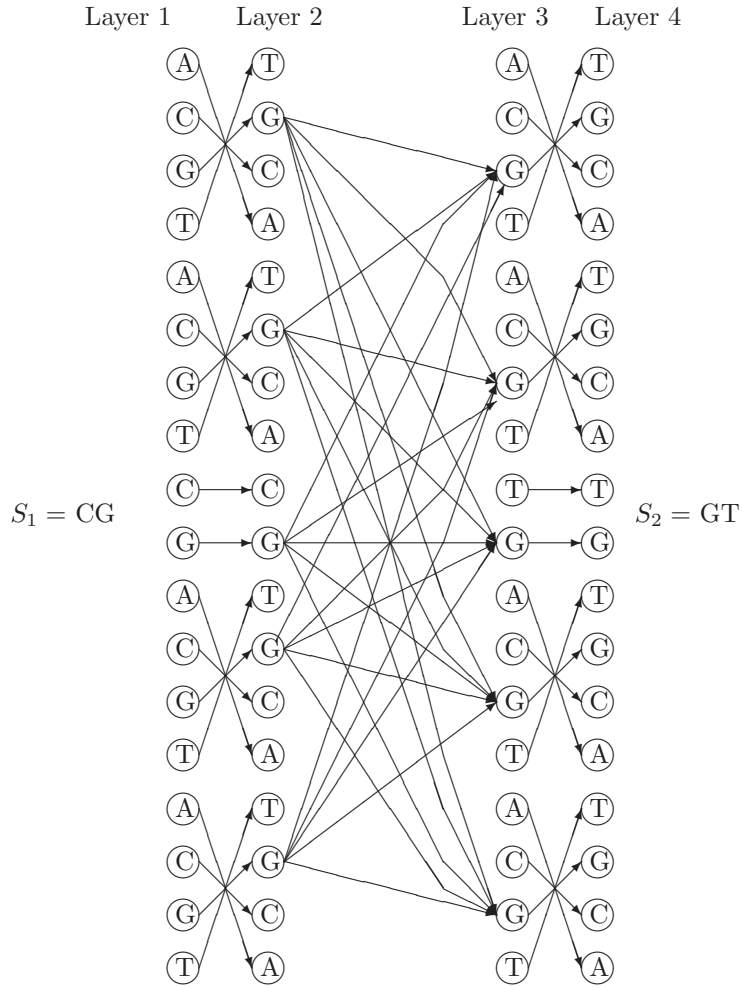


Figure 12: An example graph G' of an SCST with $S_1=GC$ to $S_2=TG$ where $\Sigma = \{A,C,G,T\}$

2. In general, the construction of the conflict graphs can be applied to any type of multi-layered graph. Their results rely on the introduction of the supernodes and assumptions on the number of insertions between each element in the sequence.

4.3 Concluding Remarks

In this chapter, through the introduction of supernodes, and the *multi-layer supergraph*, it was proved that MWCMS is \mathcal{NP} -complete. Furthermore, it was shown that a conflict graph derived from the multi-layer supergraph has the property that a solution to the associated node-packing problem of the conflict graph corresponds to a solution of the MWCMS problem. In this case, it was shown that when the

number of input sequences is a constant, MWCMS is polynomial-time solvable. It was demonstrated that some well-known combinatorial problems can be viewed as special cases of the MWCMS problem. In particular, theoretical results implied by the MWCMS theory for the minimum weight supersequence problem, the minimum weight superstring problem, and the longest common subsequence problem were presented.

CHAPTER V

COMPUTATIONAL MODELS: INTEGER PROGRAMMING FORMULATION

In this chapter, we present integer programming(IP) models for solving various classes of sequence analysis problems as *minimum weight common mutated sequence* (MWCMS) problem. Section 5.1 presents two distinct but related models: constrained network flow problem and weighted node packing problem. In Section 5.2, we reformulate constrained network flow model to obtain tighter linear relaxation. To solve the reformulated model efficiently, in Section 5.3, we study the polyhedral structure of the conflict graph corresponding to arcs between supernodes. In particular, we develop an efficient separation algorithm for violated maximal cliques. In Section 5.4, we study polyhedral structure of the constrained network flow formulation and find strong valid inequalities for it. Finally, in Section 5.5, we present the computational results.

5.1 Integer Programming Formulation

The construction of multi-layer supergraphs described in Chapter 4 lays the foundation and provides direction for computational models and solution strategies that we have explored. Although the theoretical results obtained are polynomial-time in nature, they present computational challenges. In many cases, calculating the worst case scenario is not trivial. Furthermore, the polynomial-time result of a node-packing problem for a perfect graph by Grötschel, *et. al.* [31] and [30] is existential in nature, and relies on the polynomial-time nature of the ellipsoid algorithm. The process itself involves solving an IP relaxation multiple times. In our case, the variables of the IP generated are complete paths in the multi-layer supergraph, G_L^m . Formally, the

integer program corresponding to our conflict graph can be stated as follows:

Let x_p be the binary variable denoting the use or non-use of the complete path p with weight w_p . Then the corresponding node-packing problem is:

$$\begin{aligned}
 \text{Min} \quad & \sum w_p x_p \\
 \text{subject to} \quad & x_p + x_q \leq 1 \quad \text{if complete paths } p \text{ and } q \text{ cross} \\
 & x_p \in \{0, 1\} \quad \text{for all complete paths } p \text{ in } G_L^m.
 \end{aligned}
 \tag{MIP1}$$

We call the inequality $x_p + x_q \leq 1$ an adjacency constraint. A natural approach to improve the solution time to (MIP1) is to decrease the size of the graph G_L^m and thus the number of variables. This is demonstrated in the special case SCST. Similar reductions in the size of G_L^m can be accomplished for both LCS and SCSQ. Among these three problems, the graph G_L^m is smallest for LCS. In LCS, all insertion and substitution supernodes can be eliminated.

The theoretical results thus far rely on the creation of *all* complete paths. Clearly, the typical number of complete paths will be in the order of n^m , where $n = \max |S_i|$. In this case, an instance with 3 sequences and 300 letters in each sequence generates more than 1 million variables. Hence, an exact formulation with all complete paths is impractical in general. We explore a simultaneous column and row generation approach for computational advances related to this formulation.

An alternative formulation can be obtained by examining G_L^m from a network perspective using arcs (instead of complete paths) in G_L^m as variables. Namely, let $x_{i,j}$ denote the use or non-use of arc (i, j) in the final sequence with $c_{i,j}$ the cost of the arc in G_L^m . The network formulation can be stated as

$$\begin{aligned}
\text{Min} \quad & \sum_{(i,j) \in E} c_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{i:(i,j) \in E} x_{i,j} = \sum_{k:(j,k) \in E} x_{j,k} \quad \text{for all } j \in V \text{ in layers } 2, \dots, 2m-1 \\
& x_{i,j} + x_{k,l} \leq 1 \quad \text{for all crossing arcs } (i,j) \text{ and } (k,l) \in E \\
& x_{i,j} \in \{0, 1\} \quad \text{for all } (i,j) \in E
\end{aligned} \tag{MIP2}$$

The first set of constraints ensures flow in equals flow out in all vertices contained in sequences $2, \dots, m-1$ (complete paths). The second set of constraints ensures that no two arcs cross. This formulation grows linearly in the number of sequences. This alternative integer programming formulation is still large, but is manageable for moderate size instances.

5.2 Network Formulation

In this Section, we reformulate constrained network flow model (MIP2) to obtain tighter linear relaxation. We provide three alternate formulations: Network Formulation-Alphabet, Network Formulation-Supernode, and Network Formulation-Clique. In Network Formulation-Alphabet, crossing arcs constraints of (MIP2) are formulated w.r.t alphabets. In Network Formulation-Supernode crossing arcs constraints are formulated w.r.t supernodes. In Network Formulation-Clique, maximal cliques are used to cover all the crossing arcs constraints.

5.2.1 Graph Construction

In order to facilitate clear understanding of different formulations, we first provide overview of the construction of the multi-layer graph as described in Chapter 4.

- **Σ -cross(supernode)**

Let Σ be a finite alphabet, and define Σ -cross to be a directed bipartite graph

consisting of $|\Sigma|$ vertices in each bipartition such that each vertex in the bipartition represents a distinct element in Σ . There is an arc between two vertices if the vertices correspond to the same element in Σ , and the geometric layout is rigidly constructed so that every arc crosses every other arc. This graph will be used as a “supernode” for insertion and substitution operations in our model. Figure 13 shows an example for Σ -cross when $\Sigma = \{A, C, G, T\}$.

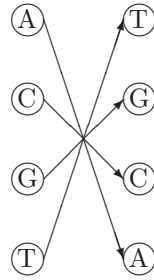


Figure 13: An example of Σ -cross when $\Sigma = \{A, C, G, T\}$.

- **2m-Layer Graph**

Given sequences S_1, \dots, S_m , generate a $2m$ -layer (directed) graph $G_L^m = (V, E)$ as follows. Layers $2i - 1$ and $2i$ consist of $(NINSERT + 1)|S_i| + NINSERT$ copies of Σ -crosses, where parameter $NINSERT$ denotes the maximum number of insertions allowed between two consecutive positions in a sequence. The first $NINSERT$ Σ -crosses represent the possibility that $NINSERT$ different letters can be inserted before the first element in S_i . The next Σ -cross corresponds to either the first letter in S_i or a substitution of this letter. This is repeated $|S_i|$ times (for each letter in S_i), and the final $NINSERT$ Σ -crosses represent insertions after the final letter in S_i . Thus, the first $NINSERT$ Σ -crosses represent the insertion supernodes, followed by one Σ -cross representing a letter in S_i or a substitution supernode, and so forth. An arc exists from a vertex in layer $2i$ to a vertex in layer $2i + 1$ if the vertices correspond to the same letter. Observe that G_L^m is an acyclic directed graph which is polynomial in the size of

the input sequences.

- **Weight of arcs**

Assign every arc between layers $2i$ and $2i + 1$ a weight of 0. There are three different weights for arcs between layers $2i - 1$ and $2i$ each corresponding to an insertion, deletion or substitution. The assignment of weights on such arcs is as follows: a weight of η is assigned to every arc contained in an insertion supernode; and an arc in a substitution supernode is assigned a weight of $-\delta$ if it corresponds to the original letter, or $\psi - \delta$, otherwise.

Figure 14 shows a sample graph for two sequences: $S_1 = GC$ and $S_2 = TG$ with $NINSERT = 2$. For simplicity, in the graph shown in Figure 14, we have not included the two insertion supernodes before the first letter nor those after the last letter of each sequence. Thus, in the figure, the first Σ -cross represents the substitution supernode associated with the first letter in S_1 . The second and third Σ -crosses represent two insertion supernodes. And the last Σ -cross represents the substitution supernode associated with the second letter in S_1 . For simplicity, we include only arcs connecting vertices associated to the element G between layers 2 and 3. The arcs for other vertices follow similarly.

5.2.2 Notation

Given m sequences where $|S_k|$ denotes the length of the sequence k , we use the following notation.

Inside and Between supernodes Layer $2k - 1$ and $2k$ represent inside supernodes; $k = 1, \dots, m$. Layer $2k$ and $2k + 1$ represent between supernodes; $k = 1, \dots, m - 1$.

Number of supernodes N_k : The number of supernodes formed by sequence k i.e. in layer $2k - 1$ and $2k$ i.e. $N_k = (NINSERT + 1)|S_k| + NINSERT$;

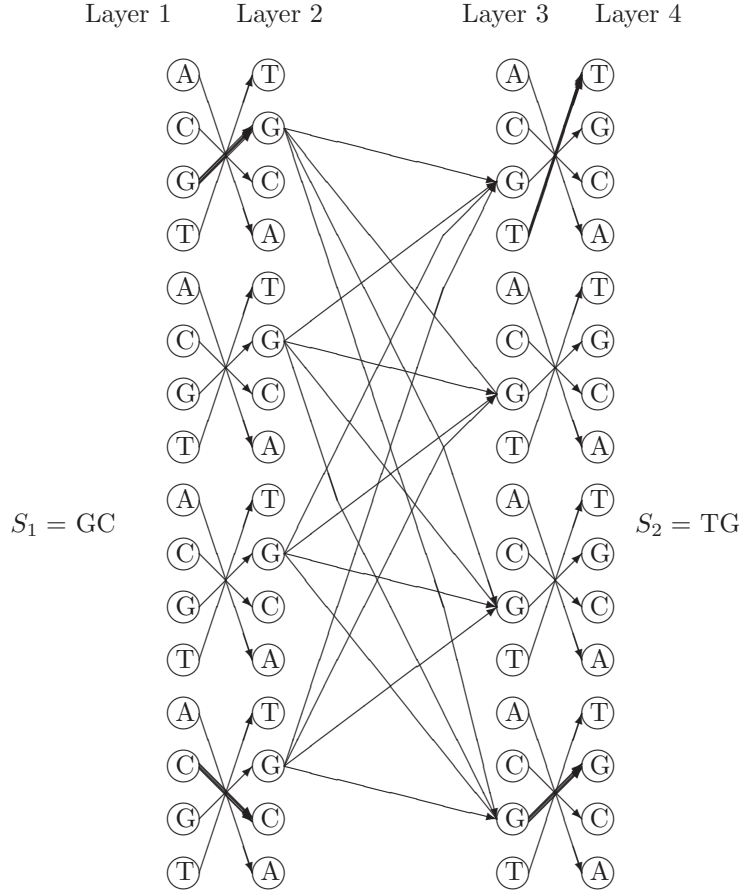


Figure 14: A sample graph G_L^m with $S_1 = GC$ to $S_2 = TG$ where $\Sigma = \{A, C, G, T\}$

$$k = 1, \dots, m$$

Supernode position P_k^u : The position u of supernode in layer $2k - 1$ and $2k$ where

$$u = 1, \dots, N_k; k = 1, \dots, m$$

Arcs inside supernodes X_{α, P_k^u} denote an arc joining alphabet α between layers

$$2k - 1 \text{ and } 2k \text{ at position } u; k = 1, \dots, m$$

Arcs between supernodes $X_{\alpha, P_k^u, P_{k+1}^t}$ denote an arc joining alphabet α between

layers $2k$ and $2k + 1$; of supernode at positions u and t respectively; $k =$

$$1, \dots, m - 1$$

5.2.3 Network Formulation-Alphabet

We will write the (MIP2) with new notation in order to compare with the other formulations clearly.

$$\begin{aligned}
\text{Min} \quad & \sum_{k,u,\alpha} C_{\alpha,P_k^u} X_{\alpha,P_k^u} + \sum_{k,u,t,\alpha} C_{\alpha,P_k^u,P_{k+1}^t} X_{\alpha,P_k^u,P_{k+1}^t} \\
\text{subject to} \quad & X_{\alpha,P_k^u} = \sum_{t=1}^{N_{k+1}} X_{\alpha,P_k^u,P_{k+1}^t} \quad \forall \alpha, u; k = 1, \dots, m-1 \\
& \sum_{u=1}^{N_k} X_{\alpha,P_k^u,P_{k+1}^t} = X_{\alpha,P_{k+1}^t} \quad \forall \alpha, t; k = 1, \dots, m-1 \\
& X_{\alpha,P_k^u} + X_{\beta,P_k^u} \leq 1 \quad \forall (\alpha, \beta), u; k = 1, \dots, m \\
& X_{\alpha,P_k^u,P_{k+1}^t} + X_{\beta,P_k^v,P_{k+1}^w} \leq 1 \quad \text{for all crossing arcs} \\
& X_{\alpha,P_k^u}, X_{\alpha,P_k^u,P_{k+1}^t} \in \{0, 1\} \quad \forall \alpha, u, t; k
\end{aligned} \tag{NFA}$$

The objective function minimizes the total cost of selected arcs. The first set of constraints ensures flow in equals flow out in all vertices contained in even numbered layers. The second set of constraints ensures flow in equals flow out in all vertices contained in odd numbered layers. The third set of constraints ensures that no two arcs cross inside every supernode. The fourth set of constraints ensures that no two arcs cross between each pair of supernodes.

5.2.4 Network Formulation-Supernode

In the formulation (NFA), both flow balance and crossing constraints are formulated w.r.t alphabets. In this new formulation, we keep flow balance w.r.t alphabets but incorporate crossing constraints w.r.t supernodes. This gives us two advantages:

- **Decrease in Size:** The number of crossing constraints decreases approximately by a factor of $|\Sigma|^2$ i.e. for DNA sequences number of constraints decreases by factor of 16.

Variables: $|\Sigma| \sum_{k=1}^m N_k$

Constraints: $\sum_{k=1}^m N_k$

- **Between Supernodes:** Layer $2k$ and $2k + 1$ represent between supernodes;
 $k = 1, \dots, m - 1$.

Variables: $|\Sigma| \sum_{k=1}^{m-1} N_k N_{k+1}$

Constraints: $\sum_{k=1}^{m-1} \binom{N_k}{2} \binom{N_{k+1}}{2} + N_k + N_{k+1}$

- **Flow Balance Constraints:** $N_1 + 2 \sum_{k=2}^{m-1} N_k + N_m$

thus we will generate/add them only when they are violated

5.2.5 Network Formulation-Clique

The formulation (NFS) presented above present unique computational challenges. The number of crossing constraints reaches billion for a moderate size instance. In this section, we replace the crossing constraints by maximal cliques which cover all crossing constraints. Clearly, the number of maximal constraints will also be enormous, thus we will generate/add them only when they are violated. In order to achieve this, we will develop efficient separation algorithms for cliques in next section. Hence, this formulation will give us the following advantages:

- **Tighter formulation:** Clearly, replacing crossing constraints by maximal cliques will result in tightening the LP relaxation. Moreover, since the conflict graph corresponding to a class of instances is *perfect*, we will be able to prove that these cliques are facet-defining under certain conditions.
- **Decrease in Computational Time:** We will develop a polynomial-time algorithm for separation of violated cliques. This should result in reduction of computational time for solving the integer programs.

$$\begin{aligned}
\text{Min} \quad & \sum_{k,u,\alpha} C_{\alpha,P_k^u} X_{\alpha,P_k^u} + \sum_{k,u,t,\alpha} C_{\alpha,P_k^u,P_{k+1}^t} X_{\alpha,P_k^u,P_{k+1}^t} \\
\text{subject to} \quad & X_{\alpha,P_k^u} = \sum_{t=1}^{N_{k+1}} X_{\alpha,P_k^u,P_{k+1}^t} \quad \forall \alpha, u, k \\
& \sum_{u=1}^{N_k} X_{\alpha,P_k^u,P_{k+1}^t} = X_{\alpha,P_{k+1}^t} \quad \forall \alpha, t, k \\
& \sum_{\alpha \in \Sigma} X_{\alpha,P_k^u} \leq 1 \quad \forall u, k \\
& \sum_{\alpha \in \Sigma} X_{\alpha,P_k^u,P_{k+1}^t} \leq 1 \quad \forall (u,t) \text{ in maximal clique, } k \\
& X_{\alpha,P_k^u}, X_{\alpha,P_k^u,P_{k+1}^t} \in \{0, 1\} \quad \forall \alpha, u, t, k
\end{aligned} \tag{NFC}$$

The objective function minimizes the total cost of selected arcs. The first set of constraints ensures flow in equals flow out in all vertices contained in even numbered layers. The second set of constraints ensures flow in equals flow out in all vertices contained in odd numbered layers. The third set of constraints ensures that at most one arc is chosen among all arcs inside a supernode. The fourth set of constraints ensures that at most one arc is chosen among arcs of maximal cliques between supernodes.

5.3 *Conflict Graph Corresponding to Arcs Between Supernodes*

In all the three constrained network flow formulations discussed in earlier section, we have two different type of constraints:

- **Flow Balance:** The first set of constraints ensures flow in equals flow out at each vertex of Layer $2k$ and Layer $2k + 1$; $k = 1, \dots, m - 1$. These constraints maintain a connection between variables across different layers and thus can be termed coupling constraints.
- **Crossing Constraints:** The second set of constraints ensures that no two arcs in solution cross. Note that these constraints can be partitioned into $2m - 1$ sets: Corresponding to arcs between Layer $1 - 2, 2 - 3, 3 - 4, \dots, (2m - 1) - 2m$.

Also, notice that obtaining non-crossing arcs between layers $2k - 1$ and $2k$ which denote inside supernodes is not hard. The difficult part is obtaining non-crossing arcs between Layer $2k$ and $2k + 1$ i.e. between supernodes.

In this section, we will study efficient ways of finding non-crossing arcs between supernodes. In particular, we will develop separation algorithm for finding violated cliques in the conflict graph. To facilitate clear understanding, we consider a 2-Layer graph G which is similar to the multi-layer graph G_L^m between supernodes. This graph G is a complete bi-partite graph having N_1 and N_2 vertices in Layer 1 and 2 respectively. Refer Figure 15.

5.3.1 Definitions

Definition. 5.3.1 Let $\mathcal{A} = \{A_1, \dots, A_s\}$ be the set of arcs in G . The *conflict graph* $\mathcal{C}_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ associated with \mathcal{A} is constructed as follows:

- $V_{\mathcal{A}} = \{A_1, \dots, A_s\}$;
- there is an edge in $\mathcal{C}_{\mathcal{A}}$ between two nodes A_l and A_m if and only if A_l and A_m cross each other in G .

Figure 16 shows an example bi-partite graph G having $N_1 = 3$ and $N_2 = 2$ vertices in Layer 1 and 2 respectively. Figure 17 shows the corresponding conflict graph $\mathcal{C}_{\mathcal{A}}$ for G where each node represents an edge in G .

5.3.2 Maximal Cliques

Let nodes of the conflict graph be represented by $A_m : (i_m, j_m)$ where i_m and j_m represent the position of vertex from the top in Layer 1 and 2 of graph G respectively. Consider two arcs $A_1 : (i_1, j_1)$ and $A_2 : (i_2, j_2)$ joining layers 1 and 2. We can assume without loss of generality that $i_1 \leq i_2$. Clearly these two arcs cross each other if and only if $j_1 \geq j_2$.

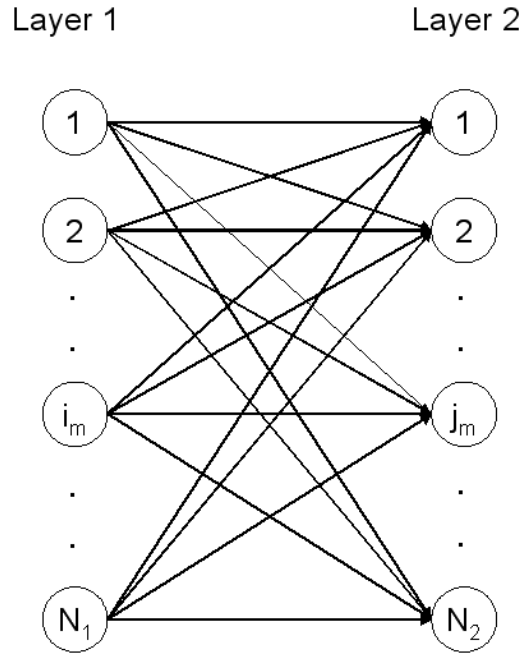


Figure 15: An example graph G having two layers

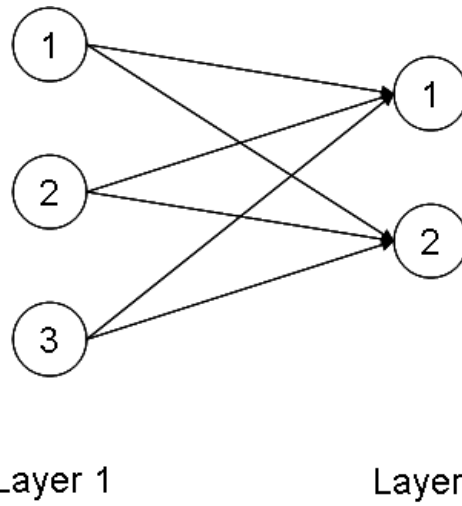


Figure 16: A sample graph G with $N_1 = 3$ and $N_2 = 2$

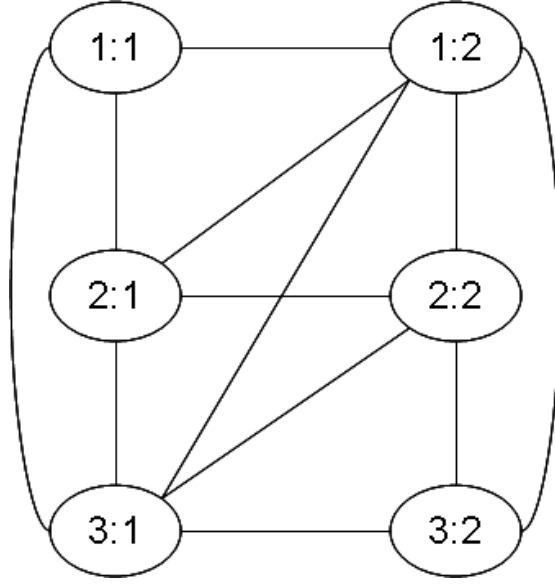


Figure 17: Conflict graph for G with $N_1 = 3$ and $N_2 = 2$

Claim 5.3.2. *Given a set of nodes $\{A_1, \dots, A_s\}$ which w.l.o.g satisfies $i_1 \leq i_2 \leq \dots \leq i_s$ forms clique in conflict graph \mathcal{C}_A if and only if $j_1 \geq j_2 \geq \dots \geq j_s$.*

Proof. We will prove the result by induction. For $s = 2$, its trivial. Let the claim be true for $s = k$, i.e. $j_1 \geq j_2 \geq \dots \geq j_k$

Now, we will prove for $s = k + 1$ i.e. $j_1 \geq j_2 \geq \dots \geq j_k \geq j_{k+1}$

$\forall m = 1 \dots k$: Arcs A_{k+1} and A_m ($i_m \leq i_{k+1}$) cross each other if and only if $j_m \geq j_{k+1}$. This implies $j_1 \geq j_2 \geq \dots \geq j_k \geq j_{k+1}$.

□

Claim 5.3.3. *Given a clique in conflict graph \mathcal{C}_A : $\{A_1, \dots, A_s\}$ which w.l.o.g satisfies $i_1 \leq i_2 \leq \dots \leq i_s$ and $j_1 \geq j_2 \geq \dots \geq j_s$.*

Every maximal clique satisfy either of these two criteria $\forall k = 1 \dots s - 1$ (Refer Figure 18).

- $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k$
- $i_{k+1} = i_k$ and $j_{k+1} = j_k - 1$

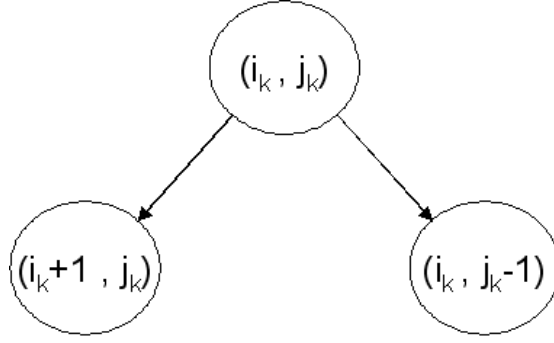


Figure 18: Maximal Clique Property

Proof. We will prove the result by contradiction. Assume $\exists k$ which does not satisfy both of the above conditions. Then, we will prove that \exists a node $A_m : (i_m, j_m)$ which could be added to the clique and hence the given clique is not maximal.

We are given that $i_1 \leq i_2 \leq \dots \leq i_s$ and $j_1 \geq j_2 \geq \dots \geq j_s$. Assume $\exists k$ for which $i_{k+1} > i_k + 1$. Clearly A_m with $i_m = i_k + 1$ and $j_m = j_k$ satisfies $i_1 \leq \dots \leq i_k \leq i_m \leq i_{k+1} \leq \dots \leq i_s$ and $j_1 \geq \dots \geq j_k \geq j_m \geq j_{k+1} \geq \dots \geq j_s$. Hence, A_m can be added to the clique.

Thus, $\forall k = 1 \dots s - 1$: $i_{k+1} = i_k$ or $i_{k+1} = i_k + 1$. Similarly, we can prove that $j_{k+1} = j_k$ or $j_{k+1} = j_k - 1$. This gives us four possible combinations of i_{k+1} and j_{k+1}

- $i_{k+1} = i_k$ and $j_{k+1} = j_k$: Implies node A_k and A_{k+1} are identical, hence this combination is not possible.
- $i_{k+1} = i_k$ and $j_{k+1} = j_k - 1$
- $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k$
- $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k - 1$: Clearly A_m with $i_m = i_k + 1$ and $j_m = j_k$ can be added to the clique. Hence this combination is also not possible.

□

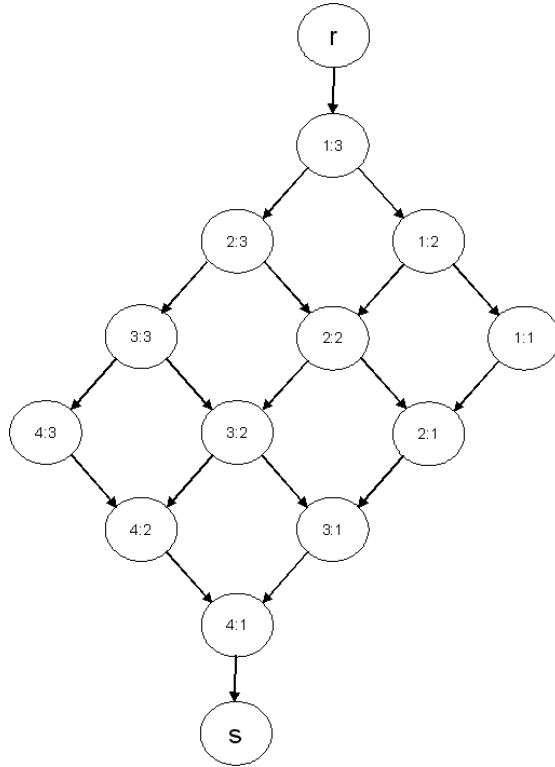


Figure 19: Clique network for $N_1 = 4$ and $N_2 = 3$

5.3.3 Clique Network

In this section, we describe the formation of clique network which will be used to generate violated cliques. Figures 19 and 20 show a sample and generalized clique network for two-layer graph respectively.

- Node: $i_m : j_m$ represents an edge A_m joining supernodes at position i_m and j_m in Layer 1 and 2 respectively. r and s represents source and sink, respectively.
- Any path from r to s represents a maximal clique.
- The incoming arcs of every node are assigned weight = negative of the solution value X of the arc represented by that node. The weight of an incoming arc at sink $s = 0$.
- The shortest path from r to s represents most violated maximal clique. If this

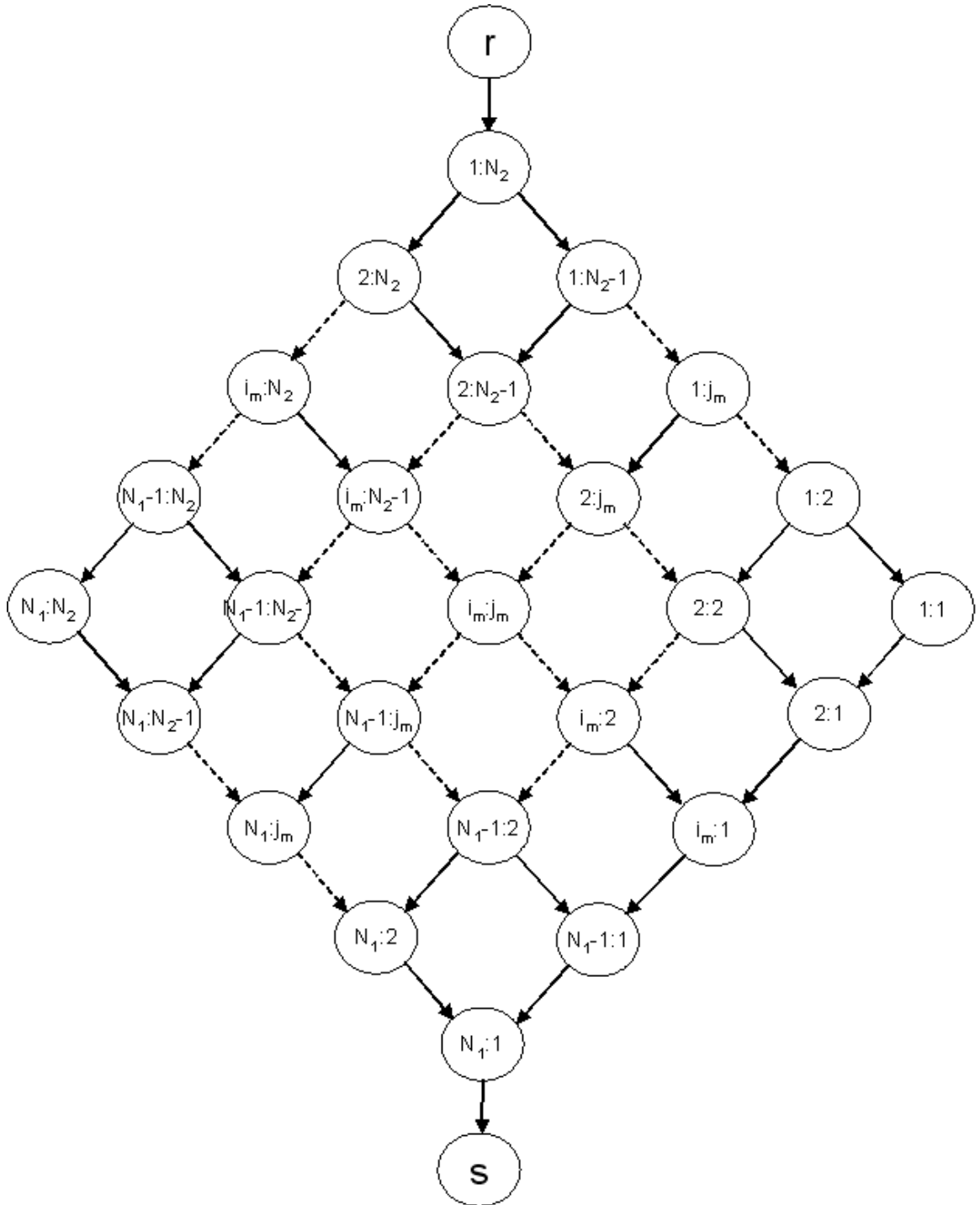


Figure 20: A clique network for a general 2-layer graph having N_1 and N_2 supernodes

one arc is chosen among all arcs inside a supernode. The fourth set of constraints ensures that at most one arc is chosen between each pair of supernodes. The fifth and sixth set of constraints ensures that at most one arc is chosen among incoming and outgoing arcs of each supernode, respectively.

5.4.1.1 Size of the formulation

- **Inside Supernodes:** Layer $2k - 1$ and $2k$ represent inside supernodes; $k = 1, \dots, m$.

Variables: $|\Sigma| \sum_{k=1}^m N_k$

Crossing Constraints: $\sum_{k=1}^m N_k$

- **Between Supernodes:** Layer $2k$ and $2k + 1$ represent between supernodes; $k = 1, \dots, m - 1$.

Variables: $|\Sigma| \sum_{k=1}^{m-1} N_k N_{k+1}$

Crossing Constraints: $\sum_{k=1}^{m-1} \binom{N_k}{2} \binom{N_{k+1}}{2} + N_k + N_{k+1}$

- **Flow Balance Constraints:** $N_1 + 2 \sum_{k=2}^{m-1} N_k + N_m$

5.4.2 Dimension of the Polytope

Claim 5.4.1. *The dimension of the polytope is given by the total number of variables (NV) - number of flow balance constraints (NFB) i.e. $(\sum_{k=1}^m N_k + \sum_{k=1}^{m-1} N_k N_{k+1}) - (N_1 + 2 \sum_{k=2}^{m-1} N_k + N_m)$*

Proof. It can easily be shown that flow balance constraints are linearly independent and thus the dimension of polytope can't be more than the claim.

It is shown that there exists $NV - NFB + 1$ affinely independent points in the polytope and hence the claim is true. In order to obtain affinely independent points, induction on number of sequences is used.

□

Table 1: Performance comparison of CPLEX default and Maximal Clique Cuts

Instance Name	No. of edges	IP Optimal Value	CPLEX default CPU min	With Clique cuts CPU min	%time reduction
N3AL100	71,640	-1158	2586	38	99
N4AL60	70,848	-1052	3370	77	98
N3AL105	69,324	-1166	114	13	88
N3AL200	160,104	-2722	397	33	92
N4AL100	113,232	-1667	471	24	95
N4AL40	42,540	-632	1935	186	90
N3AL50	32,532	-515	604	8	99
N5AL50	73,248	-1067	179	6	97
N4AL80	86,136	-1305	79	7	92
N3AL130	96,144	-1580	264	41	84

5.4.3 Maximal Clique Inequalities are Facets

Claim 5.4.2. *The maximal cliques in conflict graph corresponding to the sections between supernodes are facets.*

Proof. The result is proved by observing that there exists $\sum_{k=1}^{m-1} N_k N_{k+1} - \sum_{k=2}^{m-1} N_k$ A.I. points satisfying the clique inequality at equality. Since the hyperplane, $\sum X_{\alpha, P_k^u, P_{k+1}^t} \leq 1$ does not contain origin, any set of affinely independent points are also linearly independent. Thus, we will exhibit L.I. points at equality. The result is proved by using induction on the number of sequences. □

5.5 Computational Tests

The computational tests were performed on Intel Xeon machine at 2333 MHz with 12GB of RAM. CPLEX 9.1 is used to solve linear programs and its MIP solver is used via callbacks.

Table 1 summarizes the results obtained by adding maximal cliques as lazy constraints. These cliques have proved to be very effective and have reduced the computational time more than 90 % on average.

5.6 Conclusion

In this chapter, we have developed integer programming formulations for solving various sequence analysis problems using the concept of multi-layer graphs. Since the node packing formulation of conflict graph corresponding to paths presents huge computational challenges, we have formulated alternative formulations from network perspective. In order to obtain a tighter LP relaxation, we reformulated the network formulation using maximal cliques. Furthermore, we developed and implemented polynomial-time separation routine for separating these clique cuts.

CHAPTER VI

SIMULTANEOUS ROW AND COLUMN GENERATION APPROACH FOR SOLVING WEIGHTED NODE PACKING PROBLEMS

In this chapter, we present a branch and bound algorithm, involving both row and column generation (branch-and-cut-and-price) for solving the node packing problem on conflict graph of complete paths. The chapter is organized as follows. In Section 6.1 we describe our notation and mathematical formulation of the problem. Section 6.2 outline our algorithm which involves both row and column generation.

Next, Section 6.3 provides overview of the pricing algorithm for generating columns. Section 6.4 gives details on clique generation for node packing formulation. Section 6.5 introduces new approach for solving this problem by combining the strength of the network formulation and the node packing formulation. This is then followed by Section 6.6 which provides our implementation details and present computational experience on collection of DNA instances.

6.1 Formulation

$$\begin{aligned} \text{Min} \quad & \sum w_p x_p \\ \text{subject to} \quad & x_p + x_q \leq 1 \quad \text{if complete paths } p \text{ and } q \text{ cross} \\ & x_p \in \{0, 1\} \quad \text{for all complete paths } p \text{ in } G_L^m. \end{aligned} \tag{NodePacking}$$

Recall from Chapter 5, the various parameters involved in this model. Each variable x_p denotes a complete path in the multi-layer graph. The objective is to

minimize the total cost of chosen non-crossing paths. Each edge in the conflict graph is represented as an adjacency constraint. These constraints guarantee that set of paths chosen are non-intersecting each other.

In most of the column generation applications, the number of rows is tractable. Similarly, constraint generation applications involves limited number of columns. However, the above node packing formulation involves exponential number of both columns (paths) and rows(adjacency constraints).

For example, the instance for 3 DNA sequences of length 300 consists of more than 1 million variables and the number of adjacency constraints reaches billion. Hence, we must develop efficient methodology involving both row and column generation to tackle this huge problem.

6.2 Outline of Algorithm

An outline of the algorithm is shown in Figure 21. We first provide an overview of the algorithm, and then discuss some of the steps in detail.

One of the major advantages of using row and column generation to solve huge linear programs lies in the ability to obtain a feasible solution early on in the process. Hence, finding a good initial solution is pivotal to the performance of this algorithm.

We use progressive alignment technique to generate feasible set of paths for restricted master LP (LP relaxation of node packing formulation). These paths are used for initial solution and also to generate additional paths during column generation step. Section 6.3.2 describes the methodology in detail.

We have formulated the pricing subproblem as a constrained shortest path on the auxiliary multi-layer graph G_L^m . This auxiliary multi-layer graph is a modified subgraph of original multi-layer graph G_L^m . Section 6.3.1 provides detail on solving the pricing subproblem via the solution to the constrained shortest path problem on G_L^m .

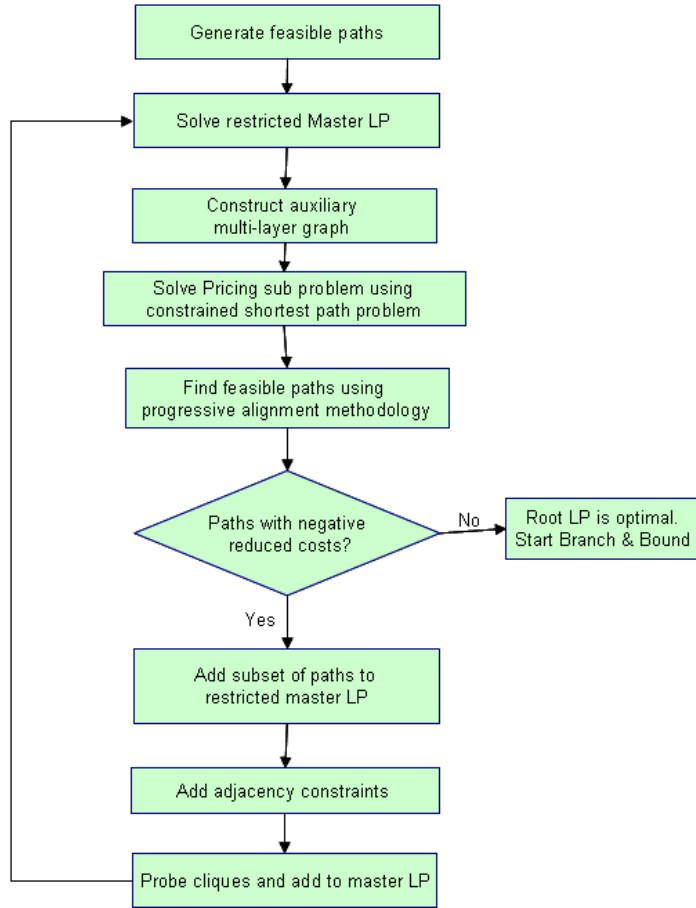


Figure 21: Outline of column and row generation algorithm for path formulation

In order to strengthen the LP relaxation, we also insert into the adjacency formulation a set of maximal cliques obtained from the conflict graph. These cliques are generated in a way similar to those described in Lee and Bixby (1998) [8]

Finally, after the convergence of the LP relaxation of the master problem, the resulting integer program is solved via a branch-and-bound approach.

6.3 Pricing

In this section, we discuss techniques for generating columns and efficient ways of calculating their reduced costs. In particular, we show that solving the shortest path problem on auxiliary multi-layer graph G_L^m corresponds to approximately solving the pricing subproblem. Furthermore, we also find a set of “candidate” columns using

progressive alignment technique and calculate their reduced costs. However, it is often not possible to add all the columns with negative reduced costs. Hence, we add only a subset with the least reduced costs.

In the context of multiple sequence alignment problem, it is not critical to get an alignment with minimum objective value, since the objective function only approximates the true properties of good alignment. Using this concept, solving the pricing subproblem approximately is sufficient for obtaining good quality alignments. In the following discussion, we provide details on two different strategies for solving the pricing problem approximately.

6.3.1 Constrained Shortest Path

We now present the construction of the auxiliary multi-layer graph $G'_L{}^m$ which allows us to solve the pricing subproblem approximately using a shortest path approach.

Let p represent a complete path on multi-layer graph G_L^m . The cost of path p , C_p is given by sum of the cost of arcs $a \in p$. To compute the reduced cost of paths, we shall first understand the structure of the conflict graph in node-packing formulation. Figure 22 shows multi-layer graph G_L^m and Figure 23 shows the corresponding conflict graph. Each node X_i in the conflict graph represents a complete path P_i in the multi-layer graph G_L^m . Each edge in the conflict graph (Figure 23) corresponds to a constraint in the node packing formulation described in Section 6.1. Hence, the constraint matrix of the node packing formulation is as given below.

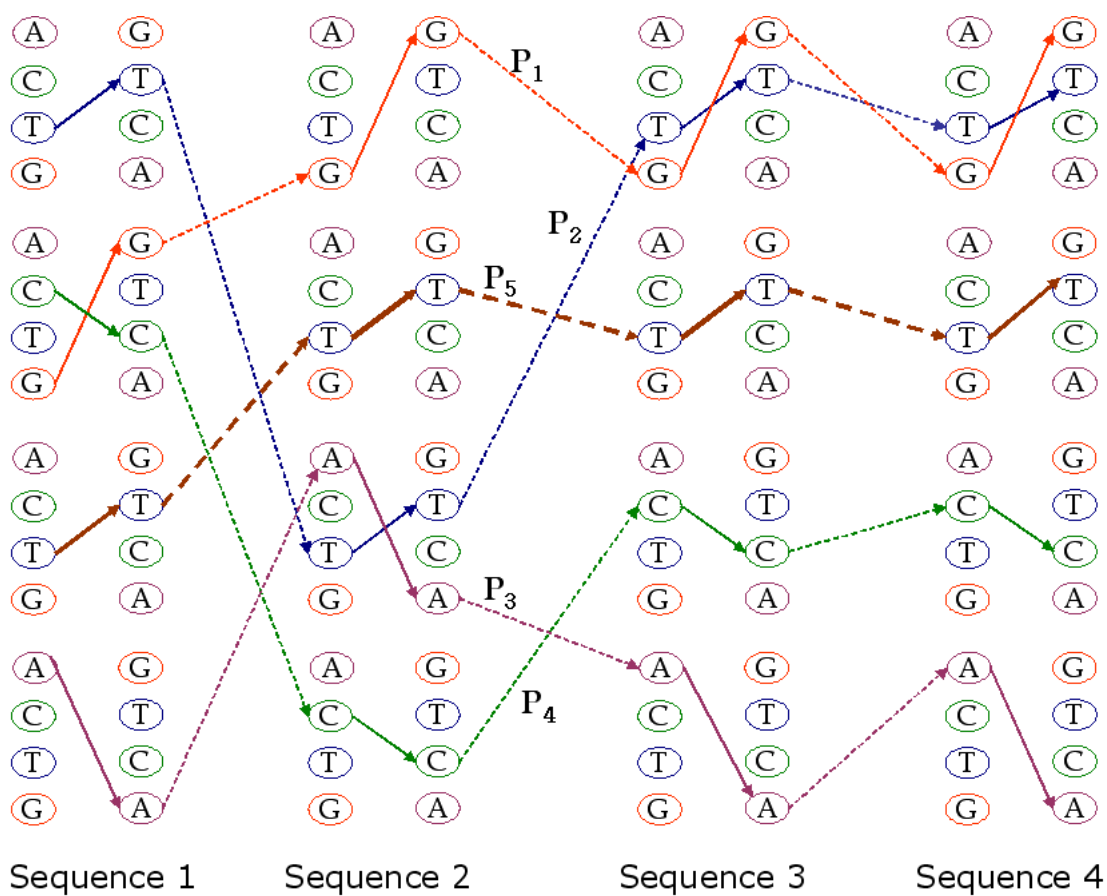


Figure 22: A sample graph of pricing subproblem for 4 sequences

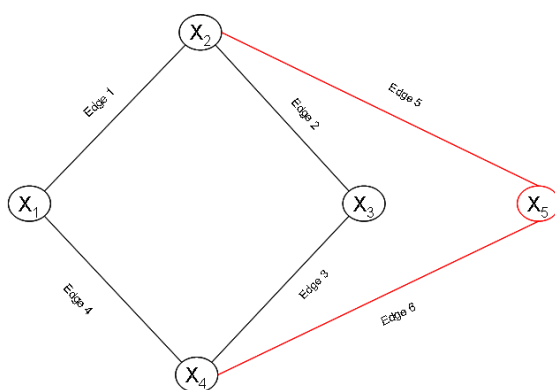


Figure 23: Conflict graph for sample graph of pricing subproblem

$$\begin{array}{l}
\text{Edge1} \\
\text{Edge2} \\
\text{Edge3} \\
\text{Edge4} \\
\mathbf{Edge5} \\
\mathbf{Edge6}
\end{array}
\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 \\
1 & 1 & & & \\
& 1 & 1 & & \\
& & 1 & 1 & \\
1 & & & 1 & \\
& 1 & & & 1 \\
& & & 1 & 1
\end{pmatrix}
\leq
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{pmatrix}$$

Now, it can be easily shown that reduced cost of given path p is given as

$$\bar{C}_p = C_p - \sum \pi_q, \text{ where path } p \text{ crosses path } q \text{ and } \pi_q \text{ is the dual price of } X_q \leq 1.$$

Hence, reduced cost: $\bar{C}_p = \sum_{a \in p} C_a - \sum \pi_q$, where C_a represent cost of arc a in G_L^m .

In order to find least \bar{C}_p , we modify G_L^m to construct an auxiliary multi-layer graph $G_L'^m$ as follows:

- Duplicate graph G_L^m to construct $G_L'^m$
- Add a source node r connecting every node in first layer of $G_L'^m$ by an arc of cost 0
- Add a sink node s connecting every node in last layer of $G_L'^m$ by an arc of cost 0
- For every path q :
 - Compute π_q : dual price of $X_q \leq 1$.
 - For every arc a' s.t. a' crosses arc $a \in q$, update the cost of arc a' , $C_{a'} = C_{a'} - \pi_q$.

Clearly, finding a shortest path from source r to sink s on $G_L'^m$ approximates finding the least reduced cost path p in G_L^m .

6.3.2 Feasible Paths using Progressive Alignment

We now present our approach for generating feasible paths using the progressive alignment approach. This approach is used to provide an initial set of paths for the master LP and to generate new paths and add ones with negative reduced costs to master LP.

1. Given N sequences, randomly choose two sequences S_1 and S_2 .
2. Align sequences S_1 and S_2 using dynamic programming algorithm (Needleman-Wunsch) . This alignment is now fixed and follows the rule of “once a gap, always a gap”.
3. From the remaining sequences, choose randomly a sequence S_3 and align it to the previously obtained alignment.
4. Repeat the last step until all the sequences have been aligned and we thus obtain alignment as a set $\hat{S} = \{S_1, \dots, S_N\}$ over the alphabet $\hat{\Sigma} \{A, T, C, G, -\}$ which satisfy the following two properties: (1) the strings in \hat{S} are of the same length, (2) S_i can be obtained from \hat{S}_i by removing the gaps. This alignment in which each string \hat{S}_i has length K can also be interpreted as an alignment matrix of N rows and K columns where row i corresponds to sequence S_i .
5. The alphabets that are placed into the same column of the alignment matrix are said to be aligned with each other. Note that every column $k = 1 \dots K$ of the alignment matrix corresponds to a complete path and all of these K paths are mutually non-intersecting with each other.
6. Translate above K paths in terms of multi-layer graph as P_1, \dots, P_K . For every path P_i find paths $P_i^1 \dots P_i^l \dots P_i^L$ which intersects path P_i . Let x_i be the node in conflict graph corresponding to path P_i in multi-layer graph. Also let π_l be dual price of constraint $x_l \leq 1$. Calculate reduced cost of $x_i = C_i - \sum_{l=1}^L \pi_l$

7. Find the paths with negative reduced cost and add a subset of them to the master LP.

6.4 *Clique Probing*

In earlier sections, we have focussed on edge formulation of conflict graph. But, the LP relaxation of node packing problem is fractional with all $x_i = 0, \frac{1}{2}, 1$. [61]. Hence, in order to tighten the LP relaxation, we generate set of maximal cliques in conflict graph in a way similar to those described in Lee and Bixby (1998) [8]

1. Find the node u with maximum degree and initialize the clique set C with it. Update the conflict graph by deleting all nodes not adjacent to u and removing the associated edges.
2. Continue to add nodes to C as follows
 - If v is the last node added to clique C , find node w with maximum degree among neighbors of v .
 - Add w to C and update the conflict graph by deleting all nodes not adjacent to w and removing the associated edges.
 - If no such w exists, stop; a clique C is found.
3. Check if the clique C is violated by current fractional solution.

The procedure is repeated on the conflict graph by deleting all the nodes \in clique C and associated edges. We continue to update conflict graph and repeat this procedure, until either specified number of cliques have been found or number of nodes left in conflict graph reaches a user-specified limit.

6.5 Combining Network and Path formulation

In Chapter 5, we presented the following node packing model which is formulated in terms of the complete paths. Each variable represents a complete path in multi-layer graph. We denote the formulation as ‘Path-Adjacency’ since each edge in conflict graph is represented as adjacency constraint.

$$\begin{aligned}
 & \text{Min} && \sum w_p x_p \\
 & \text{subject to} && \\
 & && x_p + x_q \leq 1 \quad \text{if complete paths } p \text{ and } q \text{ cross} \\
 & && x_p \in \{0, 1\} \quad \text{for all complete paths } p \text{ in } G_L^m.
 \end{aligned}$$

(Path-Adjacency)

The conflict graph of path formulation is dense and hence number of adjacency constraints grow very rapidly. For 5000 columns, the number of adjacency constraints reaches 5 million. This increases both time and memory requirements for solving the restricted master LP problem.

In order to tackle this issue, we have developed approach which uses the maximal cliques developed for network formulation in Chapter 5 to cover all adjacency constraints in node packing formulation.

We replace each adjacency constraint by crossing in terms of arcs of the multi-layer graph G_L^m . In order to do this, we introduce a binary variable $z_{i,j}^p$ to denote if arc (i, j) in G_L^m is a part of complete path p in G_L^m . We then replace, adjacency constraints w.r.t. paths with crossing constraints w.r.t. arcs. We denote this formulation as “Path-Arc-Crossing”.

$$\text{Min} \quad \sum w_p x_p$$

subject to

$$\begin{aligned} y_{i,j} &= \sum_p z_{i,j}^p x_p && \text{where } z_{i,j}^p = 1 \text{ if arc } (i,j) \in \text{complete path } p \\ y_{i,j} + y_{k,l} &\leq 1 && \text{if arcs } (i,j) \text{ and } (k,l) \text{ cross} \\ x_p, y_{i,j} &\in \{0, 1\} && \text{for all complete paths } p \text{ and arcs } (i,j) \text{ in } G_L^m. \end{aligned}$$

(Path-Arc-Crossing)

Now, we use maximal cliques developed in Chapter 5 to replace the crossing constraints w.r.t. arcs. And, we use the separation routine based on shortest path on the “clique network” to find violated cliques in polynomial time. We denote this formulation as “Path-Arc-Clique”.

$$\text{Min} \quad \sum w_p x_p$$

subject to

$$\begin{aligned} y_{i,j} &= \sum_p z_{i,j}^p x_p && \text{where } z_{i,j}^p = 1 \text{ if arc } (i,j) \in \text{complete path } p \\ \sum_{(i,j) \in C} y_{i,j} &\leq 1 && \text{for all maximal cliques } C \\ x_p, y_{i,j} &\in \{0, 1\} && \text{for all complete paths } p \text{ and arcs } (i,j) \text{ in } G_L^m. \end{aligned}$$

(Path-Arc-Clique)

6.6 Computational Tests

We have so far described various algorithmic procedures for row and column generation. Now we present various computational tests to benchmark the effectiveness of our proposed methodology.

The computational tests were executed on Intel Xeon machine at 2333 MHz with 12GB of RAM. CPLEX 9.1 is used to solve the linear programs and its MIP solver

Table 2: Performance comparison of Network and Path-Adjacency formulation

Instance Name	No.of edges	Network-Clique		Path-Formulation			%time reduction
		IP Opt.	CPU min	IP Obj	CPU min	Opt. gap	
N3AL100	71,640	-1158	38	-1139	0.66	1.64%	98
N4AL60	70,848	-1052	77	-1022	1.01	2.85%	99
N3AL105	69,324	-1166	13	-1132	0.64	2.92%	95
N3AL200	160,104	-2722	33	-2690	0.54	1.18%	98
N4AL100	113,232	-1667	24	-1625	1.11	2.52%	95
N4AL40	42,540	-632	186	-624	1.00	1.27%	99
N3AL50	32,532	-515	8	-513	0.58	0.39%	93
N5AL50	73,248	-1067	6	-1052	1.23	1.41%	78
N4AL80	86,136	-1305	7	-1285	1.08	1.53%	84
N3AL130	96,144	-1580	41	-1555	0.71	1.58%	98

is used via callbacks. Table 2 compares the algorithm performance between the network and path formulation. The path formulation has returned solution within 3% of optimality with average of 95% reduction in time.

Table 3 provides comparison between performance of Path-Adjacency and Path-Arc-Clique formulation. With the introduction of cliques based on network formulation, we are able to reduce the computational time on average of 50% and also improve the solution quality.

Figures 24 and 25 provide results for computational tests used for understanding the impact of the length of sequences on computational time. As length of sequences increases, the number of arcs in multi-layer graph G_L^m increases from 5 million to 25–50 million. Since, the complexity of both column and row generation subproblems is heavily influenced by number of arcs in G_L^m , steady increase in computational time is observed.

Figure 26 and 27 show performance as both number and length of sequences vary. For a clear understanding, we have shown performance of instances with even and odd number of sequences separately. Once again, the running time of algorithm shows a clear correlation with increase in number and length of sequences.

Table 3: Performance comparison of Path-Adjacency formulation and Path-Arc-Clique formulation

Instance Name	No. of edges	Path-Adjacency		Path-Arc-Clique		% time reduction
		IP Obj	CPU min	IP Obj	CPU min	
N5AL300	6,276,608	-5429	2106	-5487	53	98
N5AL410	9,482,752	-7746	95	-7828	87	8
N3AL625	7,210,624	-7523	138	-7529	42	69
N4AL315	5,483,136	-4833	194	-4883	41	79
N10AL125	6,559,488	-3811	227	-3828	59	74
N10AL325	17,263,872	-11174	213	-11333	242	-13
N3AL825	9,343,744	-9605	33	-9629	18	46
N4AL550	8,663,424	-8115	54	-8185	16	70
N3AL1000	11,709,568	-11886	54	-11947	27	49
N4AL1050	17,544,960	-15769	59	-15827	36	39

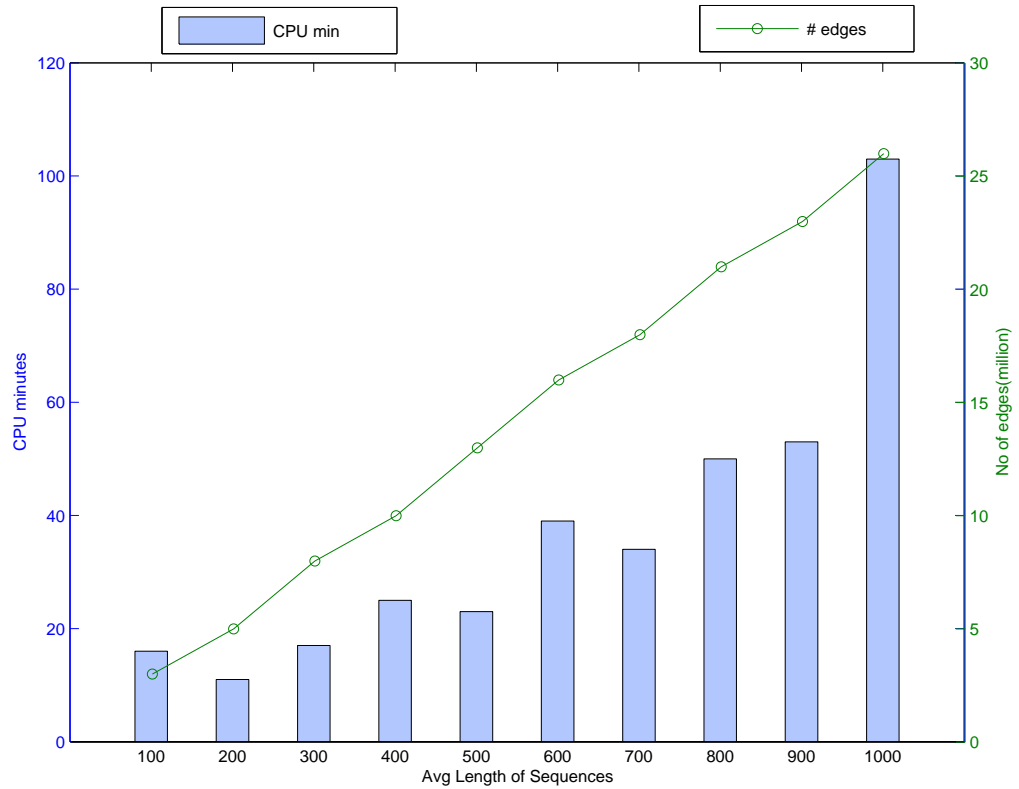


Figure 24: Path-Arc-Clique algorithm performance for 5 sequences

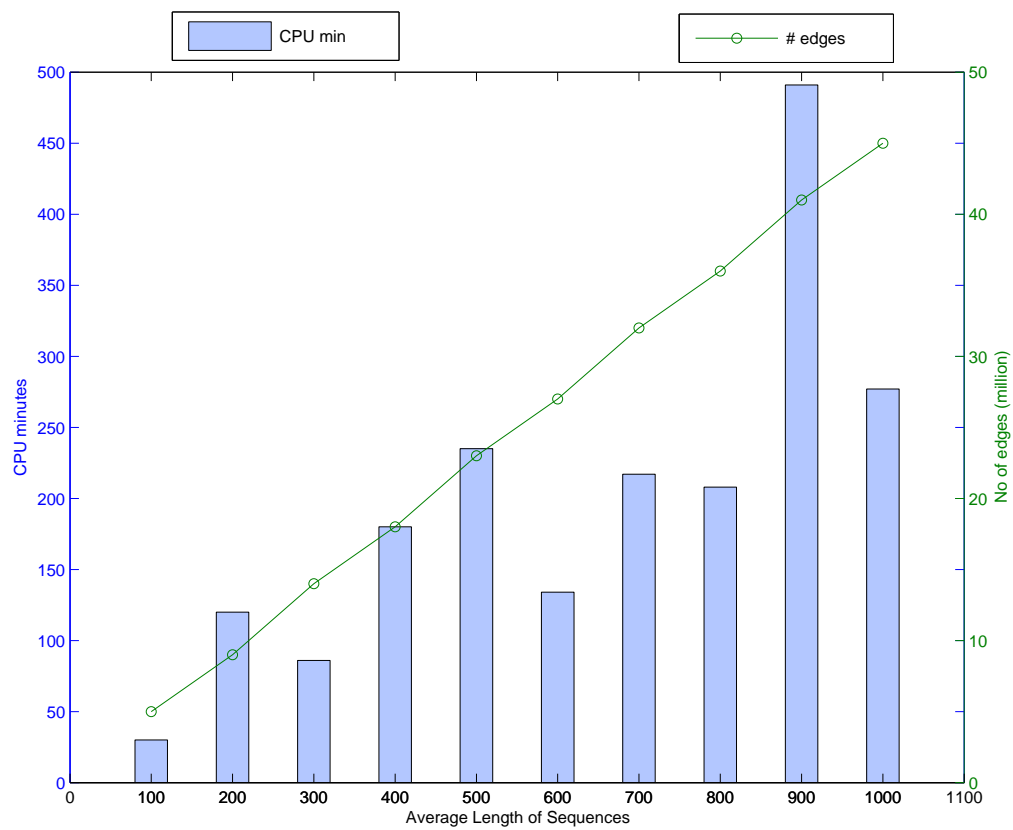


Figure 25: Path-Arc-Clique algorithm performance for 8 sequences

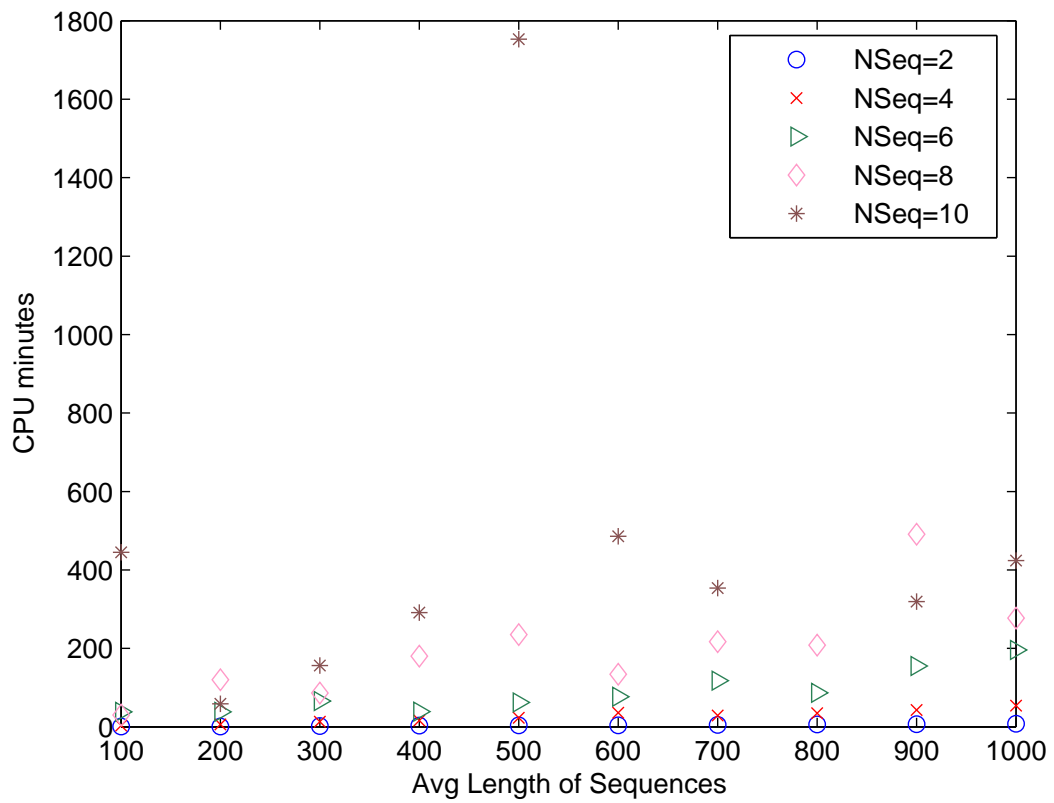


Figure 26: Path-Arc-Clique algorithm performance for even number of sequences

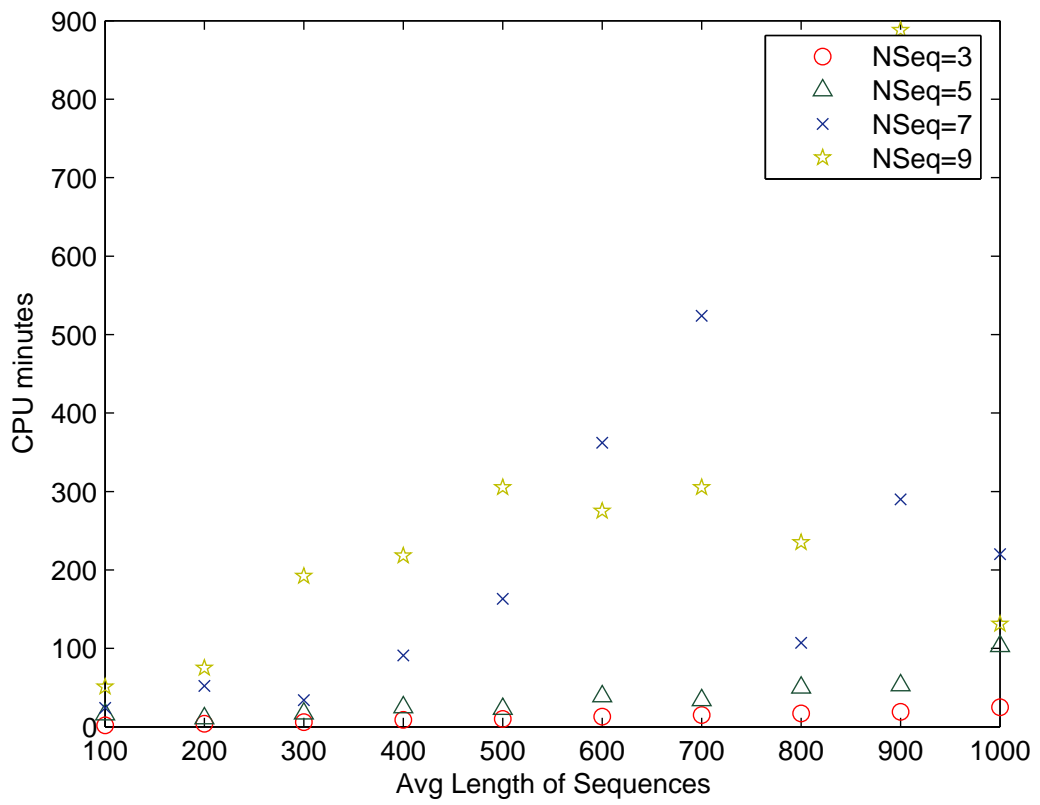


Figure 27: Path-Arc-Clique algorithm performance for odd number of sequences

6.7 Conclusion and Future Work

In this chapter, we have developed simultaneous row and column generation approach for the solving node packing problem on a conflict graph corresponding to complete paths. The algorithm introduced herein for multiple sequence alignment overcomes the order-dependent drawbacks of many of the existing algorithms, and is capable of returning good sequence alignments within reasonable computational time.

The methodology developed here will be also be useful to solve large scale integer programming problems arising in other areas such as transportation and logistics. We continue to focus on developing and investigating solution techniques to the two integer programming models, along with the development of fast heuristic procedures.

Algorithmic decomposability of the developed methodology lends itself naturally amenable for parallel distributed computing, and we continue to explore its flexibility and scalability in a massive parallel environment.

REFERENCES

- [1] ABBAS, A. and HOLMES, S., “Bioinformatics and management science: Some common tools and techniques,” *Operations Research*, vol. 52, no. 2, pp. 165–190, 2004.
- [2] ALTHAUS, E., CAPRARA, A., LENHOF, H., and REINERT, K., “A branch-and-cut algorithm for multiple sequence alignment,” *Mathematical Programming*, vol. 105, no. 2-3, pp. 387 – 425, 2006.
- [3] ALTSCHUL, S. F., CARROLL, R. J., and LIPMAN, D. J., “Weights for data related by a tree,” *Journal of Molecular Biology*, vol. 207, no. 4, pp. 647–653, 1989.
- [4] ALTSCHUL, S., “Amino acid substitution matrices from an information theoretic perspective,” *Journal of Molecular Biology*, vol. 219, no. 3, pp. 555–565, 1991.
- [5] BAINS, W. and SMITH, G., “A novel method for DNA sequence determination,” *Journal of Theoretical Biology*, vol. 135, pp. 303–307, 1988.
- [6] BARTON, G. J. and STERNBERG, M. J. E., “A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons,” *Journal of Molecular Biology*, vol. 198, pp. 327–337, 1987.
- [7] BELLARE, M. and SUDAN, M., “Improved non-approximability results,” *Proc. 26th ACM Symp. on Theory of Computing*, pp. 184–193, 1994.
- [8] BIXBY, R. E. and LEE, E. K., “Solving a truck dispatching scheduling problem using branch-and-cut,” *Operations Research*, vol. 46, pp. 355–367, 1998.
- [9] BLAZEWICZ, J., FORMANOWICZ, P., and KASPRZAK, M., “Selected combinatorial problems of computational biology,” *European Journal of Operational Research*, vol. 161, pp. 585–597, 2005.
- [10] BONIZZONI, P. and VEDOVA, G., “The complexity of multiple sequence alignment with SP-score that is a metric,” *Theoretical Computer Science*, vol. 259, pp. 63–79, 2001.
- [11] BOS, D. and POSADA, D., “Using models of nucleotide evolution to build phylogenetic trees,” *Developmental and Comparative Immunology*, vol. 29, no. 3, pp. 211–227, 2005.
- [12] BRUNO, W. J., SOCCI, N. D., and HALPERN, A. L., “Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction,” *Molecular Biology and Evolution*, vol. 17, pp. 189–197, 2000.

- [13] CARRILLO, H. and LIPMAN, D., “The multiple sequence alignment problem in biology,” *SIAM Journal on Applied Mathematics*, vol. 48, no. 5, pp. 1073–1082, 1988.
- [14] CHAKRABARTI, S., LANCZYCKI, C. J., PANCHENKO, A. R., PRZYTYCKA, T. M., THIESSEN, P. A., and BRYANT, S. H., “Refining multiple sequence alignments with conserved core regions,” *Nucleic Acids Research*, vol. 34, no. 9, pp. 2598–2606, 2006.
- [15] CHENNA, R., SUGAWARA, H., KOIKE, T., LOPEZ, R., GIBSON, T. J., HIGGINS, D. G., and THOMPSON, J. D., “Multiple sequence alignment with the clustal series of programs,” *Nucleic Acids Research*, vol. 31, no. 13, pp. 3497–3500, 2003.
- [16] CHOR, B. and TULLER, T., “Maximum likelihood of evolutionary trees: hardness and approximation,” *Bioinformatics*, vol. 21, no. Suppl. 1, pp. I97–I106, 2005.
- [17] CLOTE, P. and BACKOFEN, R., *Computational Molecular Biology: An Introduction*. John Wiley and Sons Ltd, 2000.
- [18] DELSUC, F., BRINKMANN, H., and PHILIPPE, H., “Phylogenomics and the reconstruction of the tree of life,” *Nature reviews. Genetics*, vol. 6, no. 5, pp. 361–375, 2005.
- [19] DURBIN, R., EDDY, S., KROGH, A., and MITCHISON, G., *Biological Sequence Analysis*. Cambridge University Press, UK, 1998.
- [20] FELSENSTEIN, J., “Evolutionary trees from DNA sequences: a maximum likelihood approach,” *Journal of Molecular Evolution*, vol. 17, no. 6, pp. 368–376, 1981.
- [21] FELSENSTEIN, J., “Phylogenies from molecular sequences: Inference and reliability,” *Annual Review of Genetics*, vol. 22, pp. 521–565, 1988.
- [22] FELSENSTEIN, J., “PHYLIP - phylogeny inference package (version 3.2),” *Cladistics*, vol. 5, pp. 164–166, 1989.
- [23] GALLANT, J., MAIDER, D., and STORER, J., “On finding minimal length superstrings,” *Journal of Computer and System Sciences*, vol. 20, pp. 50–58, 1980.
- [24] GAREY, M. and JOHNSON, D., *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, USA, 1979.
- [25] GASCUEL, O., “BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data,” *Molecular biology and evolution*, vol. 14, no. 7, pp. 685–695, 1997.

- [26] GOEFFON, A., RICHER, J., and HAO, J., “Local search for the maximum parsimony problem,” *Lecture Notes in Computer Science*, vol. 3612, pp. 678–683, 2005.
- [27] GOLUMBIC, M. C., ROTEM, D., and URRUTIA, J., “Comparability graphs and intersection graphs,” *Discrete Mathematics*, vol. 43, pp. 37–46, 1983.
- [28] GOTOH, O., “Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments,” *Journal of Molecular Biology*, vol. 264, no. 4, pp. 823–838, 1996.
- [29] GOTOH, O., “Multiple sequence alignment: algorithms and applications,” *Advances in Biophysics*, vol. 36, pp. 159–206, 1999.
- [30] GRÖTSCHEL, M., LOVÁSZ, L., and SCHRIJVER, A., “Polynomial algorithms for perfect graphs,” *Annals of Discrete Mathematics*, pp. 325–356, 1984.
- [31] GRÖTSCHEL, M., LOVÁSZ, L., and SCHRIJVER, A., *Geometric algorithms and combinatorial optimization*. New York: Springer-Verlag, 1988.
- [32] GUINDON, S. and GASCUEL, O., “A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood,” *Systematic biology*, vol. 52, no. 5, pp. 696–704, 2003.
- [33] GUPTA, S., KECECIOGLU, J., and SCHAEFFER, A., “Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment,” *Journal of Computational Biology*, vol. 2, pp. 459–472, 1995.
- [34] HEIN, J., “A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given,” *Molecular Biology and Evolution*, vol. 6, no. 6, pp. 649–668, 1989.
- [35] HEIN, M. H. S. J., “Consequences of recombination on traditional phylogenetic analysis,” *Genetics*, vol. 156, no. 2, pp. 879–891, 2000.
- [36] HUELSENBECK, J. and CRANDALL, K., “Phylogeny estimation and hypothesis testing using maximum likelihood,” *Annual Review of Ecology and Systematics*, vol. 28, pp. 437–66, 1997.
- [37] HUGHEY, R. and KROGH, A., “Hidden markov models for sequence analysis: extension and analysis of the basic method,” *Computer Applications in the Biosciences*, vol. 12, no. 2, pp. 95–107, 1996.
- [38] IDURY, R. M. and WATERMAN, M. S., “A new algorithm for DNA sequence assembly,” *Journal of Computational Biology*, vol. 2, no. 2, pp. 291–306, 1995.
- [39] JIANG, T. and LI, M., “On the approximation of shortest common supersequences and longest common subsequences,” *SIAM J. Comput.*, vol. 24, no. 5, pp. 1122–1139, 1995.

- [40] JUKES, T. H. and CANTOR, C. R., “Evolution of protein molecules,” in *Mammalian Protein Metabolism* (MUNRO, H. N., ed.), pp. 21–123, Academic Press, New York, 1969.
- [41] JUST, W. and VEDOVA, G., “Multiple sequence alignment as a facility-location problem,” *INFORMS Journal on Computing*, vol. 16, no. 4, pp. 430–440, 2004.
- [42] KEANE, T., NAUGHTON, T., TRAVERS, S., MCINERNEY, J., and MCCORMACK, G., “DPRml: distributed phylogeny reconstruction by maximum likelihood,” *Bioinformatics*, vol. 21, no. 7, pp. 969–974, 2005.
- [43] KECECIOGLU, J., LENHOF, H., MEHLHORN, K., MUTZEL, P., REINERT, K., and VINGRON, M., “A polyhedral approach to sequence alignment problems,” *Discrete Applied Mathematics*, vol. 104, pp. 143–186, 2000.
- [44] KIM, J., PRAMANIK, S., and CHUNG, M. J., “Multiple sequence alignment using simulated annealing,” *Bioinformatics*, vol. 10, no. 4, pp. 419–426, 1994.
- [45] KIMURA, M., “A simple method for estimating evolutionary of base substitution through comparative studies of nucleotide sequences,” *Journal of Molecular Evolution*, vol. 16, pp. 111–120, 1980.
- [46] KLOTZ, L. and BLANKEN, R., “A practical method for calculating evolutionary trees from sequence data,” *Journal of Theoretical Biology*, vol. 91, no. 2, pp. 261–272, 1981.
- [47] KOROSTENSKY, C. and GONNET, G. H., “Near optimal multiple sequence alignments using a traveling salesman problem approach,” *Proceedings of the String Processing and Information Retrieval Symposium*, p. 105, 1999.
- [48] KOROSTENSKY, C. and GONNET, G. H., “Using traveling salesman problem algorithms for evolutionary tree construction,” *Bioinformatics*, vol. 16, no. 7, pp. 619–627, 2000.
- [49] KROGH, A., BROWN, M., MIAN, I. S., SJOLANDER, K., and HAUSSLER, D., “Hidden markov models in computational biology: Applications to protein modeling,” *Journal of Molecular Biology*, vol. 235, pp. 1501–1531, 1994.
- [50] KUMAR, S., TAMURA, K., and NEI, M., “MEGA: Molecular evolutionary genetics analysis software for microcomputers,” *Computer Applications in Biosciences*, vol. 10, pp. 189–191, 1994.
- [51] KUMAR, S., TAMURA, K., and NEI, M., “MEGA3: integrated software for molecular evolutionary genetics analysis and sequence alignment,” *Briefings in Bioinformatics*, vol. 5, no. 2, pp. 150–163, 2004.
- [52] LAWRENCE, C., ALTSCHUL, S., BOGUSKI, M., LIU, J., NEUWALD, A., and WOOTTON, J., “Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment,” *Science*, vol. 262, pp. 208–214, 1993.

- [53] LEE, E. K., EASTON, T., and GUPTA, K., “Novel evolutionary models and applications to sequence alignment problems,” *Annals of Operations Research*, vol. 148, no. 1, pp. 167–187, 2006.
- [54] LEE, E. K., EASTON, T., and JOHNSON, E., “Combinatorial approach to the minimum weight common mutated sequence problem and its application to DNA sequencing problems,” tech. rep., Industrial and Systems Engineering, Georgia Institute of Technology, 2001.
- [55] LEVENSHTAIN, V. L., “Binary codes capable of correcting deletions, insertions, and reversals,” *Cybernetics Control Theory*, vol. 10, no. 9, pp. 707–710, 1966.
- [56] LI, W., “Simple method for constructing phylogenetic trees from distance matrices,” *Proc Natl Acad Sci U.S.A.*, vol. 78, no. 2, pp. 1085–1089, 1981.
- [57] LIPMAN, D., ALTSCHUL, S., and KECECIOGLU, J., “A tool for multiple sequence alignment,” *Proc Natl Acad Sci U.S.A.*, vol. 86, no. 12, pp. 4412–4415, 1989.
- [58] MAIER, D., “The complexity of some problems on subsequences and supersequences,” *J. Assoc. Comput. Mach.*, vol. 25, pp. 322–336, 1977.
- [59] MAIER, D. and STORER, J., “A note on the complexity of the superstring problem,” Tech. Rep. 233, Princeton University, U.S.A, 1977.
- [60] NEI, M., “Phylogenetic analysis in molecular evolutionary genetics,” *Annual review of genetics*, vol. 30, pp. 371–403, 1996.
- [61] NEMHAUSER, G. and TROTTER, L., “Vertex packings: Structural properties and algorithms,” *Mathematical Programming*, vol. 8, pp. 232–248, 1975.
- [62] NOTREDAME, C. and HIGGINS, D., “SAGA: sequence alignment by genetic algorithm,” *Nucleic Acids Research*, vol. 24, no. 8, pp. 1515–1524, 1996.
- [63] NOTREDAME, C., “Recent progress in multiple sequence alignment: a survey,” *Pharmacogenomics*, vol. 3, no. 1, 2002.
- [64] PHILLIPS, A., JANIES, D., and WHEELER, W., “Multiple sequence alignment in phylogenetic analysis,” *Molecular Phylogenetics and Evolution*, vol. 16, no. 3, pp. 317–330, 2000.
- [65] PIONTKIVSKA, H., “Efficiencies of maximum likelihood methods of phylogenetic inferences when different substitution models are used,” *Molecular Phylogenetics and Evolution*, vol. 31, no. 3, pp. 865–873, 2004.
- [66] PURDOM, P., BRADFORD, P. G., TAMURA, K., and KUMAR, S., “Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees,” *Bioinformatics*, vol. 16, pp. 140–151, 2000.

- [67] REINERT, K., LENHOF, H., MUTZEL, P., MEHLHORN, K., and KECECIOGLU, J., “A branch-and-cut algorithm for multiple sequence alignment,” *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*, pp. 241 – 249, 1997.
- [68] RONQUIST, F., “Fast fitch-parsimony algorithms for large data sets,” *Cladistics*, vol. 14, pp. 387–400, 1998.
- [69] SAITOU, N. and NEI, M., “The neighbor-joining method: a new method for reconstructing phylogenetic trees,” *Molecular Biology and Evolution*, vol. 4, pp. 406–425, 1987.
- [70] SANKOFF, D. and CEDERGREN, R. J., “Simultaneous comparison of three or more sequences related by a tree.,” in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* (SANKOFF, D. and KRUSKAL, J. B., eds.), pp. 253–264, Addison-Wesley, 1983.
- [71] SELLERS, P. H., “On the theory and computation of evolutionary distances,” *SIAM Journal on Applied Mathematics*, vol. 26, no. 4, pp. 787–793, 1974.
- [72] SHYU, S. J., TSAI, Y. T., and LEE, R., “The minimal spanning tree preservation approaches for DNA multiple sequence alignment and evolutionary tree construction,” *Journal of Combinatorial Optimization*, vol. 8, no. 4, pp. 453–468, 2004.
- [73] SOKAL, R. and MICHENER, C., “A statistical method for evaluating systematic relationships,” *University of Kansas Scientific Bulletin*, vol. 38, pp. 1409–1438, 1958.
- [74] STAMATAKIS, A., OTT, M., and LUDWIG, T., “RAxML-OMP: An efficient program for phylogenetic inference on smps,” *Lecture Notes in Computer Science*, vol. 3606, pp. 288–302, 2005.
- [75] SWOFFORD, D. L. and MADDISON, W. P., “Reconstructing ancestral character states under wagner parsimony,” *Mathematical Biosciences*, vol. 87, pp. 199–229, 1987.
- [76] SWOFFORD, D. L. and OLSEN, G. J., “Phylogeny reconstruction,” in *Molecular Systematics* (HILLIS, D. M. and MORITZ, G., eds.), pp. 411–501, Sinauer Associates, 1990.
- [77] TAJIMA, F. and NEI, M., “Estimation of evolutionary distance between nucleotide sequences,” *Molecular Biology and Evolution*, vol. 1, no. 3, pp. 269–85, 1984.
- [78] TAJIMA, F. and TAKEZAKI, N., “Estimation of evolutionary distance for reconstructing molecular phylogenetic trees,” *Molecular Biology and Evolution*, vol. 11, pp. 278–286, 1994.

- [79] TAKAHASHI, K. and NEI, M., “Efficiencies of fast algorithms of phylogenetic inference under the criteria of maximum parsimony, minimum evolution, and maximum likelihood when a large number of sequences are used,” *Molecular Biology and Evolution*, vol. 17, pp. 1251–1258, 2000.
- [80] TENG, S. and YAO, F., “Approximating shortest supersequences,” *Proc. of 34th Ann. IEEE Symp. on Foundations of Comp. Sci. IEEE Computer Society.*, pp. 158–165, 1993.
- [81] THOMPSON, J. D., HIGGINS, D. G., and GIBSON, T. J., “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice,” *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673 – 4680, 1994.
- [82] VINGRON, M. and HAESLER, A., “Towards integration of multiple alignment and phylogenetic tree construction,” *Journal of Computational Biology*, vol. 4, no. 1, pp. 23–34, 1997.
- [83] VINGRON, M. and WATERMAN, M., “Sequence alignment and penalty choice. review of concepts, case studies and implications,” *Journal of Molecular Biology*, vol. 235, no. 1, pp. 1–12, 1994.
- [84] WAGNER, R. A. and FISCHER, M. J., “The sequence-to-sequence correction problem,” *J. Assoc. Comput. Mach.*, vol. 21, pp. 168–173, 1974.
- [85] WALLACE, I. M., O’SULLIVAN, O., and HIGGINS, D. G., “Evaluation of iterative alignment algorithms for multiple alignment,” *Bioinformatics*, vol. 21, no. 8, pp. 1408–14, 2005.
- [86] WATERMAN, M. S., *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman and Hall, UK, 1995.
- [87] WATERMAN, M. and PERLWITZ, M., “Line geometries for sequence comparisons,” *Bulletin of Mathematical Biology*, vol. 46, no. 4, pp. 567–577, 1984.
- [88] WHELAN, S., LIO, P., and GOLDMAN, N., “Molecular phylogenetics: state-of-the-art methods for looking into the past,” *Trends in genetics*, vol. 17, no. 5, pp. 262–272, 2001.
- [89] YANG, Z., “Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites,” *Molecular Biology and Evolution*, vol. 10, no. 6, pp. 1396–401, 1993.
- [90] ZHANG, Y. and WATERMAN, M., “An eulerian path approach to global multiple alignment for DNA sequences,” *Journal of Computational Biology*, vol. 10, no. 6, pp. 803–819, 2003.