Tree Decomposable Models for Efficient Bioinformatics Algorithms

by

Chunmei Liu

(Under the direction of Liming Cai)

Abstract

Many important bioinformatics problems are NP-hard and it is thus difficult to find efficient algorithms to solve them exactly. However, these problems often require efficient and exact solutions for the accurate understanding of biological systems, presenting a great challenge in computation. This dissertation initiates an algorithmic framework to address this challenge, which is established on the theory and techniques of parameterized computation. In particular, the framework is to model bioinformatics problems with graphs of small tree width, where existing elegant graph theoretic results, such as the Courcelle's theorem, may be applied to achieve efficient (e.g., linear-time) exact algorithms.

The dissertation investigates two technically difficult issues in implementing this framework: (1) how to model bioinformatics problems as monadic second order (MSO) logic expressible problems on graphs of small tree width; (2) how to develop parameterized techniques that can reduce other problems to MSO logic expressible graph problems while maintaining small tree width. Various bioinformatics problems in proteomics and structural genomics are studied to address these two issues. Experiments were conducted which demonstrate the advantages of this new algorithmic framework over other methods in both accuracy and efficiency.

INDEX WORDS: Parameterized computation, Tree decomposition, Tree width, Subgraph isomorphism, Sequence-structure alignment, Antisymmetric path, Tandem mass spectra

Tree Decomposable Models for Efficient Bioinformatics Algorithms

by

Chunmei Liu

B.S., Anhui University, 1999

M.S., Anhui University, 2002

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2006

Tree Decomposable Models for Efficient Bioinformatics Algorithms

by

Chunmei Liu

Approved:

Major Professor: Liming Cai

Committee: E. Rodney Canfield
Russell L. Malmberg
Robert W. Robinson

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2006

Acknowledgments

First, I am extremely grateful to my Ph.D. advisor, Professor Liming Cai, for his instruction, guidance, and inspiration through my Ph.D. career. It is he who led me into these promising and interesting research areas. His devotion to his students, style in teaching, and excellence in research have made himself a great advisor and a good example for me to follow in my academic career.

I am also grateful to my committee member, Professor Russell Malmberg. He is always very patient to teach me both biology and computer science. It has been a great pleasure to work with him and I look forward to future collaboration.

I owe a great debt of gratitude to the other members of my committee, Professors Rodney Canfield and Robert Robinson, for their contribution to my education through the classes they taught and the invaluable guidance that they have given me. I feel fortunate to have studied under them.

I would also like to thank the student members in our research group: Yinglei Song, Jizhen Zhao, and Dongsheng Che at the Department of Computer Science, Fangfang Pan at the Department of Plant Biology, and Ping Hu at the Department of Genetics.

I would like to say thank you to Dr. Bo Yan and Professor Ying Xu at the Department of Biochemistry and Molecular Biology, who collaborated with me on the development and implementation of the work in proteomics. Others that have provided technical support on the work of protein tertiary structure prediction are Dr. Juntao Guo, Dr. Guojun Li, and Kyle Ellrott. I thank them for their wonderful support, and I expect continued collaborations.

I have been receiving financial support during my doctoral student career. Specifically, the work in this dissertation has been supported by the Graduate School at the University of Georgia, the Computer Science Department, and the National Institutes of Health.

I have also received support from a team of personnel. I thank Professors John Miller and Hamid Arabnia in graduate advising, Professor Krzysztof Kochut and departmental staff in administration, and system support.

I would also like to thank my teachers and friends both in China and in the US. They have made my life beautiful. I cannot list all their names here, but they are truly in my mind.

Finally, I would like to thank my parents: my father, Hongwen Liu, and my mother, Lanrong Sui. They are the most wonderful parents in the world. I love them so much. My brothers and sisters have also always provided me with whatever I need. I want to say special thank you to them.

Table of Contents

Chapter 1

Introduction

Due to the rapid development of biological sciences, a large amount of data has been available for the description of biological systems. Computational tools can significantly improve the efficiency and accuracy of data analysis and are thus highly desirable for the processing of the biological data. Bioinformatics is an emerging and fast growing interdisciplinary research field that reflects this type of need. In particular, one of the major objectives of bioinformatics research is to develop new methods and computational tools that can facilitate and provide new insights into the frontiers of research in molecular and systems biology. In addition, some problems in bioinformatics are interesting by themselves from the computational perspective. Developing algorithmic and complexity results for these problems can enrich research in computer science.

In contemporary molecular biology, the number of possible solutions to a given bioinformatics problem is often combinatorial in nature and in most cases, biologists are interested in the one(s) that optimize some given objective function. These are known as *combinatorial optimization* problems. Combinatorial optimization problems constitute one of the most important aspects in the research of bioinformatics, since many problems in bioinformatics can be reduced to a combinatorial optimization problem. Unfortunately, most of these combinatorial optimization problems are known to be NP-hard, which suggests that optimal solutions for these problems cannot be obtained efficiently unless P = NP.

However, optimal or near optimal solutions for these problems are often required for accurate understanding of the corresponding biological problems. The commonly used approach

to cope with the computational intractability is approximation. In particular, polynomial-time approximation algorithms can produce approximate solutions with performance guaranteed with respect to the optimal ones. However, they introduce inaccurate and an expensive time/accuracy trade-off, and they are not suitable for solving bioinformatics problems where accuracy is always desirable. Therefore, developing practically fast and yet accurate solutions for bioinformatics problems remains a very important research goal.

This dissertation initiates an algorithmic framework to address this issue based on the theory and techniques of parameterized computation. The framework models bioinformatics problems with graphs of small tree width based on the Courcelle's theorem. In particular, the dissertation investigates two technically difficult issues in implementing this framework: to model bioinformatics problems as monadic second order (MSO) logic expressible problems on graphs of small tree width, and to develop parameterized techniques that can reduce other problems to MSO logic expressible graph problems while maintaining small tree width.

1.1 Intractable Problems in Bioinformatics

1.1.1 Annotating Genomes for Non-Coding RNAs

Non-coding RNAs (ncRNAs) are biologically important and have been known to play important roles in many processes including gene regulation, chromosome replication, and RNA modification [32, 60, 103]. Recently, with the explosive growth of fully sequenced genomes, efficient computational tools are needed to search the genomes to identify and annotate ncRNA genes with high accuracy. However, no satisfactory methods have been available for this problem so far.

An RNA molecule is formed by a sequence of nucleotides. These nucleotides are usually represented as *residue bases* of the four possible types adenine(A), guanine(G), uracil(U) and cytosine(C). Non-coding RNAs fold into secondary structures due to the *base pairs* formed between nucleotides on the sequence. Stacked base pairs are energetically more stable and

form *stems*. Regions consisting of unpaired contiguous nucleotides in the sequence are called *loops*.

ncRNAs do not encode proteins. The secondary structure of a ncRNA is important and to a large extent, determines its biological functions. In general, to search for an ncRNA, a search tool needs to distinguish the structural features of the ncRNA from others in the genome. In particular, the tool needs to align all sequence segments in a genome to a model that contains the statistical profile information obtained from a family of sequences of the ncRNA. The sequence segments in the genome with statistically significant alignment scores are reported as hits. Due to the importance of secondary structure, the profile for the ncRNA needs to include information from both sequence and structure.

An RNA structure may contain *pseudoknots*, which are structure units that contain at least two interleaving stems. The optimal sequence-structure alignment between a sequence segment and a structure without pseudoknots can be performed with a dynamic programming algorithm in time $O(N^3W)$, where N is the length of the sequence segment and W is the size of the profile. The computational efficiency of this algorithm becomes a bottleneck when the length of the searched structure is larger than 300 nucleotides. For structures that contain pseudoknots, the sequence-structure alignment has been shown to be NP-hard [1]. A few models and algorithms [17, 67, 70, 92] have been developed to model pseudoknot structures and align a sequence to a pseudoknot structure. However, since the computational complexities of these algorithms are prohibitively high, they cannot be directly used for searching pseudoknot structures in large genomes.

Several different approaches [94, 54, 10] have been developed to resolve the difficulty arising from the computational efficiency of sequence-structure alignment. For example, filtering models have been developed to efficiently preprocess the genome and screen out most of the regions that do not contain the searched structure. More sophisticated structure models can then be used to process the regions that have passed the filtering process. Filtering

methods have significantly improved the efficiency of searching. However, filtering methods have not been applied to searching for pseudoknot structures.

This dissertation develops an original structure model for the secondary structure of an ncRNA. Based on the structure model, the sequence-structure alignment can be performed in time $O(k^t N^2)$ [85, 86], where k and t are parameters which are small in practice, and N is the length of the searched structure pattern. In particular, the parameter t is 2 for pseudoknot free structures and increases only slightly for pseudoknot structures. Test results show that this algorithm is significantly faster than previous ones but can achieve the same search accuracy.

1.1.2 Protein Tertiary Structure Prediction

Protein tertiary structure prediction is a classical problem in bioinformatics. The objective of this problem is to computationally determine the tertiary structure of a protein molecule from its amino acid sequence. This problem is important and has been intensively studied for more than a decade [37, 46, 82, 84, 96, 98]. Currently available methods for this problem can be roughly classified into two categories. *Ab initio* methods consider the thermodynamic energy due to the interactions of atoms in a protein sequence and minimize it to obtain the tertiary structure of a protein [84]. However, the atomic interactions in a protein sequence are complex and the *ab initio* methods have not been able to achieve satisfactory prediction accuracy. Knowledge-based methods [37, 46, 82, 96, 98] align a sequence to all the available structure profiles in a database. Prediction is based on the alignment scores. In contrast to *ab initio* methods, knowledge-based methods can achieve higher prediction accuracy and are thus often used in practice to predict protein tertiary structures.

Protein threading is a knowledge-based method for protein tertiary structure prediction. Threading methods consider both the sequence and structure information for a family of homologous sequences and include it in a structure profile. The alignments needed in protein

threading are thus sequence-structure alignments. It has been shown that the optimal alignment between a sequence and a structure profile that includes two-body residue interactions and allows gaps is NP-hard [50].

Several algorithms have been developed to optimally align a sequence to a protein structure. For example, in [96], a divide-and-conquer algorithm is shown that can perform optimal sequence-structure alignment in polynomial time for most available protein structure templates if an appropriate cut-off distance is used to model the two-body interactions. In [98], the sequence-structure alignment is solved with integer programming techniques. In practice, these two algorithms can perform most sequence-structure alignments in a few minutes. However, threading methods need to align a sequence to all structure templates in a database. Therefore, it is desirable to further improve the computational efficiency for sequence-structure alignment. This should allow the use of more sophisticated model to improve prediction accuracy.

This dissertation studies the sequence-structure alignment problem for protein threading. The protein sequence-structure alignment problem can also be reduced to the Subgraph Isomorphism problem and a parameterization of this problem can be solved with an efficient parameterized algorithm. The results of our testing show that this method is significantly faster than previous algorithms and can achieve better accuracy in sequence-structure alignment and fold recognition.

1.1.3 *De Novo* Sequencing for Peptide Sequences

Tandem mass spectrometry (MS/MS) has become an important experimental tool for protein identification in proteomics. A protein sequence can be cut into short peptide sequences by certain enzymes. A protein peptide can be fragmented into charged ions and their mass/charge ratios can be measured. Since the peptide can be fragmented at any bond on its backbone, the mass/charge ratios obtained for a peptide form a *spectrum*. A spectrum may contain several different types of ions; *b-ions* and *y-ions* are two types that are often

seen in a MS/MS spectrum. The peptide identification problem is to determine the amino acid sequence of the peptide from its MS/MS spectrum.

Database search based methods [29, 63] have been used to determine the peptide sequence from its spectrum. Specifically, the spectrum of each peptide in a database is compared with the spectrum of a new peptide and the similarity between the two spectra can be evaluated. Peptides whose spectra have significant similarity values with the new one are returned as the possible sequencing results. However, this method is unable to identify a peptide sequence that is not in the database. *De novo* sequencing is to determine the amino acid sequence of a peptide solely from its MS/MS spectrum.

A number of different methods and models have been developed for *de novo* sequencing. In [65], a spectrum is modeled with a directed graph and the corresponding amino acid sequence can be determined by finding the maximum weighted antisymmetric path in the graph. Such a graph is called a *spectrum graph*. However, finding the maximum weighted antisymmetric path is an NP-hard problem [35] and in [65], only an approximate algorithm is presented for the problem.

The spectral alignment problem is to find the optimal alignment between the peaks in two MS/MS spectra. This problem can be solved with a dynamic programming algorithm in polynomial time if both spectra contain only one type of ion. However, in the presence of both *b-ions* and *y-ions*, the spectra alignment problem is also NP-hard [35]. This problem can also be reduced to finding the maximum weighted antisymmetric problem in an *alignment graph* constructed from the two spectra that are to be aligned.

A novel parameterized algorithm is developed in this dissertation to optimally find the maximum weighted antisymmetric path in a graph that is slighted modified compared with the spectrum graph proposed in [65]. A linear time efficient parameterized algorithm has been developed to find the maximum antisymmetric path in a modified spectrum graph or an alignment graph. Experimental results show that this algorithm is efficient and effective

for both *de novo* sequencing and spectral alignment problems and can be effectively applied to identify post-translational modifications from the alignment of two spectra.

1.1.4 Protein Identification with Tag Based Database Search

Two types of methods have been developed for the peptide sequencing problem, *i.e.* methods using database search and *de novo* sequencing methods. In theory, *de novo* sequencing methods can identify a peptide that is not included in a database. However, in practice, a spectrum generally contains many noisy mass peaks and some mass peaks in the theoretical spectrum can be missing in the real spectrum due to errors that can occur in the experiments. The spectrum graph constructed from a real spectrum is generally disconnected and it is thus difficult for a *de novo* sequencing algorithm to find the needed antisymmetric path.

Database search methods are thus still important for processing real experimental spectra. However, *post-translational modifications* (PTMs) often occur on peptides and the possibility of PTMs must be considered during the database search. This will significantly increase the computation time needed to compare two spectra since an exhaustive search on all possible PTMs must be performed. Filtering [34] is a new method that can be used to improve the computational efficiency of database search. Generally, a simple model is used to efficiently preprocess all the peptides in a database and only those peptides that are likely to match the input spectrum are selected for further processing. These selected peptides are called *candidates*.

In [34], PepNovo, a tag based filtering program, is developed to speed up the database search. In particular, the spectrum is processed with a program for *de novo* sequencing. A machine-learning based method is then used to evaluate the likelihood that each sequence tag in the sequencing results is in the corresponding peptide. Peptide sequence tags with a significant likelihood are then used to filter the database. Only peptides that contain one of the tags are selected as candidates. However, the accuracy of this method is sensitive to the

quality of the training data and may not work properly in the presence of PTMs, because the spectrum graph can be disconnected and the *de novo* sequencing results cannot be easily obtained.

In this dissertation, a new method is developed for the problem of tag selection. In particular, an efficient parameterized algorithm is developed to find all the *maximal antisymmetric paths* in a spectrum graph, the tags are selected from these paths, and the likelihood for each of them to be in the peptide is evaluated. Experiments have shown that this approach can achieve better accuracy on tag selection and perform significantly faster than PepNovo.

1.2 Parameterized Computation

Parameterized computation is particularly suited for coping with computational intractability under the framework of NP-completeness. In particular, given an instance of an NP-hard problem $\langle x, k_1, k_2, \cdots, k_m \rangle$, where $k_1, k_2, \cdots, k_m$ are *parameters*, the problem is called *fixed parameter tractable* (FPT) if there exists an algorithm that can solve the problem in time $O(f(k_1, k_2, \cdots, k_m)|x|^c)$, where c is a constant independent of x and k_i's ($1 \leq i \leq m$). For example, the Vertex Cover problem, which asks whether a given graph $G = (V, E)$ contains a vertex subset of size k such that each edge in G has either one of its ends in this subset, can be solved in time $O(2^k|V|)$ using a search tree based technique. Such problems are solvable in polynomial time for small values of the parameters. In contrast, complexity classes have also been constructed for parameterized computation. More specifically, we consider parameterized decision problems that are formulated on circuits. The *weight* k of an input assignment is the number of input variables that are assigned to be "true". For a given circuit, we consider gates in the circuit with unbounded wefts, and the *depth* t of the circuit is the maximum number of such gates in a path from an input to the output of the circuit. We use W[t] to represent the class of problems $\langle k, x \rangle$ (k is the parameter) that can be reduced in time $O(f(k)|x|^c)$ to a problem that asks whether there exists a weight k' satisfying assignment for a circuit of depth t, where k' only depends on k, $f(k)$ is a recursive function,

and c is a constant independent of k. It is easy to show that FPT $\subseteq$ W[1] $\subseteq$ W[2] $\cdots$ W[t] $\cdots$, and it is believed that FPT is a proper subset of W[1] and W[t] is a proper subset of W[$t+1$] for $t \geq 1$. Analogously to the theory of NP-completeness, complete problems for these complexity classes have been found. For example, the INDEPENDENT SET problem, which is to determine whether graph $G = (V, E)$ contains a vertex subset of size k, has been shown to be W[1]-complete. This suggests that unless FPT=W[1], INDEPENDENT SET cannot be solved in time $O(f(k)|V|^c)$ for any recursive f, where c is a constant independent of k and G.

The application of parameterized computation is not limited to theoretical computer science. In practice, many intractable optimization problems have inherently small parameters and methods developed in parameterized computation can thus provide efficient solutions for them. For example, a parameterized algorithm with time complexity $O(kn+1.2852^k)$ [20] has been developed for the VERTEX COVER problem. This algorithm can be practically used to solve the problem instances where the parameter k is less than 50. In bioinformatics, exact solutions for many NP-hard optimization problems are needed and it provides an avenue to apply parameterized computation.

Tree decomposition(tree width) [71] is one of the most original and important graph theoretical concepts proposed in the last two decades. Tree decomposition was originally proposed to prove the graph minor theorems. Based on the tree decomposition, it has been shown that graphs are well-quasi ordered with respect to the relation of minor. The concepts of tree decomposition and tree width also have profound implications in algorithm study. In particular, it has been shown that many NP-hard graph optimization problems can be solved in linear time on graphs with bounded tree width. For example, the MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET in a graph $G = (V, E)$ with tree width bounded by k can be found in time $O(2^k|V|)$ and $O(2^{2k}|V|)$ respectively.

A more general result is Courcelle's theorem [23]. This theorem states that given a tree decomposition for a graph $G = (V, E)$ with tree width t, any predicate that can be expressed

in a monadic second order (MSO) logic can be decided in time $O(f(t)|V|)$, where $f(t)$ is a function whose value only dependents on t. Such work provides a general approach for us to develop efficient and exact algorithms for important bioinformatics problems.

This dissertation focuses on the study of bioinformatics problems using the concept of tree decomposition. In particular, we initiate the study of an algorithmic framework that can model NP-hard bioinformatics problems on graphs with small tree width. The algorithm developed in Courcelle's theorem can thus be applied to solve them. Technically, two difficult but important issues need to be investigated to implement this framework: how to model bioinformatics problems with MSO logic expressible problems on graphs with bounded tree width, and how to develop parameterized techniques that can reduce other problems to MSO logic expressible graph problems while maintaining small tree width. Through experiments with various bioinformatics problems, we show that this algorithmic framework can provide efficient and accurate solutions for all the intractable bioinformatics problems in section 1.1.

1.3 Outline of the Dissertation

The rest of this dissertation is organized as follows. A thorough survey of the idea of tree decomposition and its algorithmic implications including Courcelle's theorem is provided in Chapter 2. In Chapter 3, we show that the Longest Antisymmetric Path problem can be expressed with MSO logic. Based on this fact, practically efficient algorithms are developed for the *de novo* sequencing and spectral alignment problems. In Chapter 4, we study the Maximum Antisymmetric Paths problem and show that it can also be expressed with MSO logic. This result leads to an efficient parameterized algorithm that can select peptide sequence tags for database search. In Chapter 5, we consider a new parameterization of the Subgraph Isomorphism problem and show that under certain constraints, an efficient parameterized algorithm can be developed for this problem. More importantly, we show that although this problem itself cannot be expressed with MSO logic, it can be

reduced to a problem expressible with MSO logic while the graph tree width is not significantly increased. This algorithm has been applied to efficiently solve the sequence-structure alignment problem in bioinformatics. These chapters also include the experimental results of these new algorithms on biological data. Chapter 6 concludes the dissertation and envisions possible future improvements of this approach in both theory and practical applications in bioinformatics.

Chapter 2

Preliminaries

2.1 Introduction

The concept of tree decomposition was originally proposed in the deep investigations into graph minors [71]. The graph minor theorem started with a conjecture by Wagner [73, 76], which states that in a infinite series of graphs, there exist two different graphs G_i and G_j such that G_i is a minor of G_j. A *minor* of a given graph is a new graph that can be obtained from the graph by edge contraction and the deletion of edges and vertices.

Tree decomposition is the most important concept used in the proof of these graph minor theorems. In particular, the tree width of a graph restricts what types of minors it could contain. The graph minor theorems have deep implications in both graph theory and algorithmic study. For example, an important property associated with a family of minor closed graphs is that the *obstruction set* for this family is finite [75]. An *obstruction set* is a minimal set of graphs such that any graph in the family does not contain any graph in the set as a minor. Based on this property, many minor related parameterized graph problems can be shown to be fixed-parameter tractable without finding the actual algorithms for these problems [73, 75].

This chapter focuses on the algorithmic aspects of graph tree decomposition. In particular, a well known fact is that many NP-hard optimization problems can be solved in linear time on graphs with bounded tree width [6, 23]. Based on this fact, practically efficient algorithms can be developed for many NP-hard problems since many graphs in practical applications have bounded tree widths.

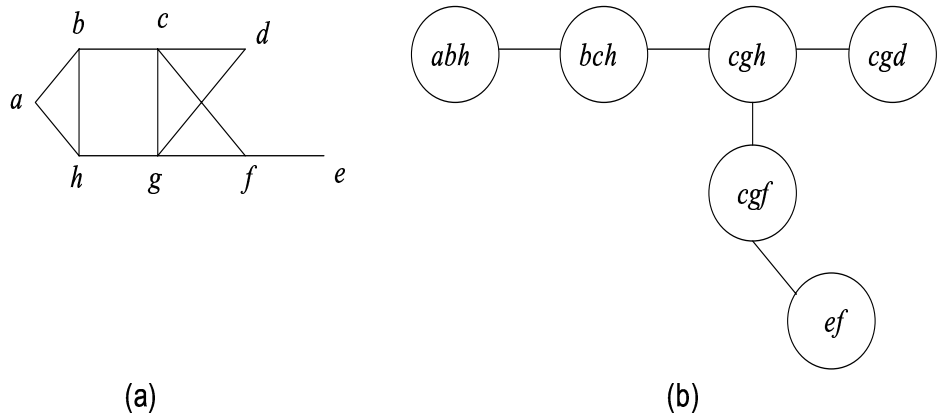

Figure 2.1: (a) An example of a graph. (b) A tree decomposition of the graph in (a).

2.2 Tree Decomposition

Definition 2.2.1 ([71]) *Let $G = (V, E)$ be a graph, where V is the set of vertices in G, E denotes the set of edges in G. Pair (T, X) is a tree decomposition of graph G if it satisfies the following conditions:*

1. *$T = (I, F)$ defines a tree, the sets of vertices and edges in T are I and F respectively,*
2. *$X = \{X_i | i \in I, X_i \subseteq V\}$, and $\forall u \in V$, $\exists i \in I$ such that $u \in X_i$,*
3. *$\forall (u, v) \in E$, $\exists i \in I$ such that $u \in X_i$ and $v \in X_i$,*
4. *$\forall i, j, k \in I$, if k is on the path that connects i and j in tree T, then $X_i \cap X_j \subseteq X_k$.*

The tree width of the tree decomposition (T, X) is defined as $\max_{i \in I} |X_i| - 1$. The tree width of the graph G is the minimum tree width over all possible tree decompositions of G.

Figure 2.1(a)(b) shows a tree decomposition. As can be seen from the figure, a tree decomposition of a graph provides alternative topological view on the graph. In a valid tree decomposition, every graph vertex must be contained in at least one tree node and the two

vertices of each edge must be both contained in at least one tree node. More importantly, the tree nodes that contain the same graph vertex must span a connected subtree in the tree decomposition.

A graph is a *clique* or *complete graph* if any two of its vertices are connected. A *subgraph* of a graph $G = (V, E)$ is a graph H formed on a subset of V and its edges form a subset of E. An *induced subgraph* of a graph $G = (V, E)$ is a subgraph such that any pair of its vertices are connected if they are connected in G. For a graph that contain a clique subgraph, the following theorem states an important property associated with its tree decompositions.

Theorem 2.2.2 ([11]) *Let $G = (V, E)$ be a graph, where V is the set of vertices in G, E is the set of its edges. If there exists set $U \subseteq V$ such that the vertices in U induce a clique in graph V, then in any tree decomposition $T = (I, F)$ for graph G, there exists a tree node $X_i \in I$ such that $U \subseteq X_i$.*

Proof: We show this fact by induction on the size of U. When $|U| = 2$, U is an edge in G, the theorem is trivially correct from the definition of a tree decomposition. Now, we assume that when $|U| = k$ ($k \geq 2$), any tree decomposition of G must have a tree node that contains U as a subset. For a larger clique U', such that $|U'| = k + 1$, we can partition it into the disjoint union of U_1 and $\{v\}$, where v is a vertex. Vertices in U_1 thus induce a clique of size k in G. Based on the induction assumption, for any tree decomposition $T = (I, F)$, there exists a tree node $X_i \in I$ such that $U_1 \subseteq X_i$.

We now assume the subtree formed by all tree nodes that contain v is T_v. If X_i is one of the tree nodes in T_v, we are done with the proof. We then assume that X_i is not in T_v. In this case, there exists a tree node X_j in T_v such that X_j is the only tree node in T_v on the path from X_i to X_j in T. For any vertex $u \in U_1$, since there is an edge between u and v in G, there exists a tree node X_u in T_v such that X_u contains u. However, since X_j is on the path from X_i to X_u in T, and u is contained in both X_i and X_u, we know that $u \in X_j$ from the definition of tree decomposition. We thus have shown that $U' \subseteq X_j$. Based on the principle of induction, the theorem has been proved. □

Corollary 2.2.3 *The tree width of a graph $G = (V, E)$ is at least $|C| - 1$, where C is a subset of V and vertices in C induce a clique in G.*

Corollary 2.2.3 provides a simple approach to estimate the lower bound of the tree width. A maximal clique for a graph G can be computed in polynomial time and the size of this clique provides a lower bound for the tree width.

A graph is called *bipartite* if its vertices can be partitioned into two disjoint sets such that the two vertices of any edge are from each of the partition. A bipartite graph is *complete* if every pair of vertices from each of the partitions is connected. For a graph that contains an induced complete bipartite subgraph, we have the following theorem.

Theorem 2.2.4 ([11]) *Let $G = (V, E)$ be a graph, where V is the set of vertices in G, E is the set of its edges. If there exists a subset U of V such that $U = U_1 \cup U_2$, and the subgraph induced on U is a complete bipartite graph between U_1 and U_2, then in any tree decomposition $T = (I, F)$ for G, there exists a tree node $X_i \in I$ such that either $U_1 \subseteq X_i$ or $U_2 \subseteq X_i$.*

Proof: We show this theorem by induction on the size of U_1. In particular, based on the definition of tree decomposition, when $|U_1|$=1, the result is trivially true. We assume the theorem is correct when $|U_1| = k$ ($k \geq 1$). We now consider the case where $|U_1| = k + 1$. We can partition U_1 into the disjoint union of U_1' and v where $v \in U_1$. Based on the induction assumption, for any tree decomposition $T = (I, F)$, there exists a tree node X_i such that either $U_1' \subseteq X_i$ or $U_2 \subseteq X_i$. If $U_2 \subseteq X_i$, we are done with the proof. We thus assume that $U_1' \subseteq X_i$. Similar to the proof of theorem 2.2.2, we consider the subtree T_v induced by tree nodes that contain vertex v. If X_i is one of the tree nodes in T_v, we have $v \in X_i$ and the theorem has been proved.

We now assume that X_i is not in T_v and we can obtain a tree node X_j in T_v such that X_j is the only tree node in T_v in the path that connects X_i and X_j in T. Now if U_2 is not contained in X_j, then $U_1 \subseteq X_j$. This can be shown as follows. If $\exists u \in U_1$, such that $u \notin X_j$, we consider the subtree T_u formed by tree nodes that contain u. T_u and T_v must be disjoint

since if there exists $X_m \in T_u$ and $X_m \in T_v$, then we consider the path that connects X_i and X_m in T and X_j must be on this path. But since $u \in X_i$ and $u \in X_m$, we know that $u \in X_j$ from the definition of tree decomposition, which is contradictory to the assumption that $u \notin X_j$.

Now since T_u and T_v are disjoint subtrees and $X_i \in T_u$, it is not difficult to see that X_j is on the path that connects any two tree nodes from T_u and T_v respectively. For any vertex $v_1 \in U_2$, we have $(v_1, u) \in G$ and $(v_1, v) \in G$, from the definition of tree decomposition, we are able to find tree nodes $X_l \in T_u$ and $X_s \in T_v$ such that $v_1 \in X_l$ and $v_1 \in X_s$. However, since the path connects X_s and X_l in T must pass through X_j, we know that $v_1 \in X_j$. We thus obtain that $U_2 \subseteq X_j$, which leads to a contradiction. The theorem has been proved. □

The *side size* $C(G)$ of a complete bipartite graph $G = (U_1 \cup U_2, E)$ is $\min\{|U_1|, |U_2|\}$. The following corollary provides another lower bound for the tree width of a graph.

Corollary 2.2.5 *The tree width of graph G is at least the side size of any complete bipartite subgraph H of G.*

2.3 Courcelle's Theorem

Based on a tree decomposition of a given graph, many NP-hard optimization problems associated with the local topology of a graph can be efficiently solved by combining partial solutions obtained in subgraphs of smaller sizes. The optimal combination of partial solutions can be performed with the dynamic programming framework proposed in [6, 23]. Specifically, partial solutions for the subgraph induced by vertices contained in the subtree rooted at a particular tree node are obtained by combining with those for subgraphs rooted at its child nodes. Optimal combinations are obtained with exhaustive computation on vertices in the tree nodes. A more general result developed in [23] showed that for a given graph $G = (V, E)$, any problem that can be formulated in monadic second order (MSO) logic can be decided in time $O(f(t)|V|)$ based on a graph tree decomposition of tree width t, for some recursive function f.

In general, the approach maintains a dynamic programming table in each tree node. Table entries in the leaves of the tree are determined first and a bottom-up development is followed to compute the table entries for internal nodes. In particular, for a leaf node, all possible partial solutions for the subgraph induced by vertices contained in it are enumerated and stored as its table entries. The table of an internal node is filled by querying the tables of its children to obtain needed partial solutions and combining them. In the following subsections, we provide a detailed description of the dynamic programming framework with notions similar to the ones adopted by Matousek and Thomas in [57] in their study of the Subgraph Isomorphism problem. As an important general result regarding the fixed-parameter tractability of graph problems where the tree width of the graph is a parameter, we also provide a brief survey of MSO logic and Courcelle's theorem [23].

2.3.1 Transition Functions

Definition 2.3.1 ([57]) *Let $G = (V, E)$ be a graph, $T = (I, F)$ be a binary tree and (T, X) be a tree decomposition of G. Function $f(G, U, D)$ is a transition function with respect to the tree decomposition: if $|U| = |D|$, D is a binary string comprised of 0 and 1's, the value of f is determined by G, U, and D, and $f(G, U, D)$ satisfies the following conditions:*

1. *$f(G, X_i, D)$ can be computed in time $p_1(|X_i|)$, if X_i is a leaf of the tree, where $p_1(x)$ is a polynomial of x, and,*

2. *For an internal node X_i, assume X_j and X_k are its two children, $f(G, X_i, D)$ can be computed in time $p_2(|X_i|)$ based on the sets $S_j = \bigcup_{|D_j|=|X_j|}\{f(G, X_j, D_j)\}$ and $S_k = \bigcup_{|D_k|=|X_k|}\{f(G, X_k, D_k)\}$, where $p_2(x)$ is a function whose value only depends on x.*

For a tree node X_i in a given tree decomposition $(T = (I, F), X)$ for graph G, the number of possible binary strings D such that $|D| = |X_i|$ is bounded by $2^{|D|}$. The set $S_i = \bigcup_{|D|=|X_i|}\{f(G, X_i, D)\}$ may contain up to $2^{|D|}$ different numbers. The time needed to

determine S_i is bounded by $2^{|X_i|}(p_1(|X_i|) + p_2(|X_i|))$. If the tree width of the tree decomposition is bounded by t, the computation time needed to determine S_i is bounded by $2^t(p_1(t) + p_2(t))$.

2.3.2 The Dynamic Programming Framework

Theorem 2.3.2 ([57]) *Let $G = (V, E)$ be a graph, $T = (I, F)$ be a binary tree and (T, X) be a tree decomposition of G, and $f(G, U, D)$ is a transition function with respect to the tree decomposition. If R is the root of T, the computation time needed to determine the set $S_R = \bigcup_{|D|=|R|}\{f(G, R, D)\}$ is bounded by $O(2^t(p_1(t) + p_2(t))|V|)$.*

Proof: It is not difficult to see that the total number of tree nodes in a tree decomposition is bounded by $O(|V|)$ [11]. For each tree node X_i in the tree decomposition, we maintain a dynamic programming table for each tree node X_i to store the set S_i. Each row in the table contains the binary string D and the corresponding function value $f(G, X_i, D)$. The algorithm follows a bottom-up fashion and computes the rows in the dynamic programming tables for leaves of the tree. It then goes upwards to determine the tables of internal nodes. Based on the definition of a transition function with respect to T, the computation time the algorithm needs to compute the rows in a table is bounded by $O(2^t(p_1(t)+p_2(t)))$. The total amount of computation time is thus bounded by $O(2^t(p_1(t) + p_2(t))|V|)$. □

To illustrate the framework more specifically, we provide two examples where NP-hard problems can be solved in linear time. An *independent set* in a graph is a vertex subset such that every pair of vertices in the subset are disconnected. An *vertex cover* in a graph is a vertex subset such that at least one of the two vertices of each edge is in the subset. The Maximum Independent Set problem is to find the independent set of the maximum size in a given graph. In contrast, the goal of the Minimum Vertex Cover problem is its vertex cover of minimum size. The following theorem states that both problems can be solved with linear time algorithms on graphs with bounded tree width.

Theorem 2.3.3 ([6]) *The computation time needed for finding the maximum independent set and the minimum vertex cover in graph $G = (V, E)$ is bounded by $O(2^t|V|)$, where t is the tree width of the graph G. We assume that there is a tree decomposition $(T = (I, F), X)$ with width t available for G.*

Proof: For a binary string D, we use D_i to represent the ith bit in D. $(1 \leq i \leq |D|)$ For the problem of finding the maximum independent set, we construct functions $V(G, X_i, D)$ and $f(G, X_i, D)$ based on the given tree decomposition of G. We show that the two functions satisfy the definition of transition function with respect to T. Specifically, we assume that X_i contains t vertices in G and they are $u_{i_1}, u_{i_2}, \cdots, u_{i_t}$. A binary string D of length t specifies the decision on $u_{i_1}, u_{i_2}, \cdots, u_{i_t}$ for a partial independent set in the subgraph induced by vertices contained in the subtree rooted at X_i. $D_p = 1$ if vertex u_{i_p} is included in the independent set and $D_p = 0$ otherwise. $V(G, X_i, D) = 1$ if there exists a partial independent set that conforms with D and $V(G, X_i, D) = 0$ otherwise.

Similarly, $f(G, X_i, D)$ is the maximum number of vertices contained in a partial independent set that conforms with D. If X_i is a leaf, to compute $V(G, X_i, D)$, we only need to check if D assigns two connected vertices to the independent set. $V(G, X_i, D) = 0$ if it is the case and $V(G, X_i, D) = 1$ otherwise. $f(G, X_i, D)$ is computed by counting the number of vertices that are included in the partial independent set by D. The property 1 in Definition 2.3.1 is satisfied. If X_i is an internal node with children X_j and X_k, in order to determine $V(G, X_i, D)$, we refer to the tables for X_j and X_k. In both tables, we enumerate all binary strings D_j and D_k such that D_j, D_k conform with D in the decisions on vertices in $X_i \cap X_j$ and $X_i \cap X_k$ respectively. $V(G, X_i, D) = 1$ if there exist D_j and D_k such that $V(G, X_j, D_j) = 1$ and $V(G, X_k, D_k) = 1$, and $V(G, X_i, D) = 0$ otherwise. Similarly, $f(G, X_i, D)$ can be computed with the following formula:

$$f(G, X_i, D) = \max_{D_j}\{f(G, X_j, D_j)\} + \max_{D_k}\{f(G, X_k, D_k)\} + N(X_i, D) \tag{2.1}$$

where D_j and D_i conform with D on vertices in $X_i \cap X_j$ and $X_i \cap X_k$ respectively, $N(X_i, D)$ is the number of vertices that are selected by D and in neither $X_i \cap X_j$ nor $X_i \cap X_k$. This proves that the property 2 is also satisfied by $V(G, X_i, D)$ and $f(G, X_i, D)$.

It is not difficult to see that the time $p_1(|X_i|)$ needed for computing $f(G, X_i, D)$ is $O(|X_i|^2)$ since for an entry in a leaf node, we only need to check whether there exists a connected pair of selected vertices. For an internal node, the time $p_2(|X_i|)$ needed for computing $f(G, X_i, D)$ is bounded by $O(2^t)$, which is a constant when t is a constant. A closer analysis can show that the aggregate time needed for computing the $f(G, X_i, D)$ for all string D's is $O(2^t)$. From Theorem 2.3.2, we know that the maximum independent set of graph G can be computed with time $O(2^t|V|)$. For the minimum vertex cover problem, we change the definition of function $V(G, X_i, D)$ to the validity of the selection of vertex cover that conforms with D and $f(G, X_i, D)$ to the minimum number of vertices contained in a vertex cover that conforms with D. We can then obtain the same result as the maximum independent set problem. □

In fact, the requirement imposed on function $p_1(|X_i|)$ can be further relaxed to a function whose value only depends on $|X_i|$, since if $|X_i|$ is a fixed parameter, $p_1(|X_i|)$ is also fixed even if it is an exponential function of $|X_i|$. More examples can be found in [6] on NP-hard problems that can be solved in linear time on graphs with bounded tree width. These problems include the Minimum Dominating Set problem and the Hamiltonian Path problem. For a given problem instance on graph $G = (V, E)$ and a tree decomposition of tree width t, it is not difficult to show that similar transition functions can also be constructed for these two problems. In particular, for the Minimum Dominating Set problem, its transition function needs to consider two possible states associated with a vertex in a tree node X_i, i.e., being "selected" and "dominated". The time needed for computing such a transition function for an internal node X_i from its child nodes X_j and X_k is thus $O(2^{2t})$. The time needed for computing the minimum dominating set is $O(2^{2t}|V|)$ based on such a transition function. Similarly, for the Hamiltonian Path problem, the corresponding transition function needs to include the permutation information associated with the vertices in a tree

node. Therefore, the computation time needed to compute this transition function for an internal node is $O(t!2^{2t})$. An algorithm with time complexity $O(t!2^{2t}|V|)$ is thus available for the Hamiltonian Path problem.

2.3.3 Monadic Second Order Logic

Definition 2.3.4 *A second order logic sentence is a* monadic second order logic (MSO) sentence *if the relations employed in a predicate are unary relations, i.e., $Q_1R_1Q_2R_2\cdots Q_kR_k\phi$, where Q_i's are quantifiers and R_i's ($1 \leq i \leq k$) are unary relations and ϕ is a first order logic sentence.*

A logic sentence formulated on a graph in general contains predicate quantifiers $\exists$, $\forall$ and logic symbols $\vee$, $\wedge$ and $\neg$. In addition, such a logic sentence may contain relations due to the graph topology. For example, $edge(u, v)$ is "true" if there is an edge between u and v in the graph and "false" otherwise; $S(u)$ is "true" if vertex u is included in set S and "false" otherwise.

A graph can be viewed as a *structure* where its vertices are *elements* and edges are *relations* among the elements. Many graph problems are closely related to satisfying an MSO logic sentence on a given structure. For example, consider the Independent Set problem whose goal is to decide whether a given graph has an independent set of size k, a graph contains an independent set of size k if and only if the corresponding structure satisfies the following MSO logic sentence:

$$\exists S \forall v_1 \forall v_2 (((v_1 \in V) \wedge (v_2 \in V) \wedge (S(v_1) \wedge S(v_2)) \rightarrow \neg edge(v_1, v_2)) \wedge (size(S) = k)) \quad (2.2)$$

Similarly, the Dominating Set problem is equivalent to decide whether the following MSO logic sentence can be satisfied by a graph structure:

$$\exists S \forall v \exists u ((u \in V) \wedge (v \in V) \wedge (S(v) \vee (S(u) \wedge edge(u, v))) \wedge (size(S) = k)) \quad (2.3)$$

An MSO logic sentence belongs to MS_1 if it allows vertex subsets are defined by unary relations and edges are represented with binary relations, otherwise it belongs to MS_2. It

is not difficult to see that both of the above two MSO logic sentences are in MS_1. The expressing ability of MS_2 is stronger than MS_1. For example, the HAMILTONIAN PATH problem is MS_2 expressible but cannot be expressed with a MSO logic sentence in MS_1 [26].

Theorem 2.3.5 (Courcelle's Theorem) *Given a MSO logic sentence $F = Q_1R_1Q_2R_2, \cdots, Q_kR_k\phi$, where Q_i's are quantifiers and R_i's are unary relations defined on a vertex or edge subset, ($1 \leq i \leq k$) and ϕ is a first order logic sentence. Based on a graph $G = (V, E)$ and a tree decomposition $T = ((I, F), X)$ of graph G with tree width t, there exists an $O(f(t)|V|)$ time algorithm that can decide whether the corresponding structure of G satisfies F, where $f(t)$ is a function whose value only depends on t.*

For a given MSO sentence, its clauses only contain vertices, edges, and relations defined on them. For each tree bag in the tree decomposition, we can enumerate all possible vertices and edges that are set to be true by each relation in the sentence. Each of such possibilities can be represented with a table entry and it is not difficult to see that the total number of such entries in the table is bounded by a function $f(t)$ of tree width t. To determine the validity of a given entry in the table of an internal node, the tables of the child nodes need to be queried and the time needed for such an operation is also bounded by a function $g(t)$ of the tree width t. The time complexity of the overall dynamic programming procedure is thus bounded by $O(f(t)g(t)|V|)$.

Courcelle's theorem provides a generalization on graph problems that are fixed-parameter tractable when the tree width of the graph is a parameter. We show in later chapters that all the problems we have studied with tree decomposition can be either formulated with a MSO logic sentence or reduced to such a problem in polynomial time [18]. An interesting theorem, which is a converse of this theorem, is shown in [80]. It states that if a MS_2 sentence is decidable on an infinite family of graphs, the tree widths of graphs in this family are *uniformly* bounded by a constant c. The theorem is proved by an elegant but complex reduction from the HALTING problem in [80]. It shows that for a given problem instance $\langle M, x\rangle$ of the HALTING problem, it can be reduced to a problem that asks whether a MS_2

sentence can be satisfied on a graph that contains a grid minor whose size is unbounded and depends on M and x. Based on this reduction, the tree widths of all graphs in the graph family must be uniformly bounded by a constant since otherwise the HALTING problem is decidable.

2.4 Finding Tree Width and Tree Decompositions

We consider the following decision problem that asks whether the tree width of a graph is at most k or not.

Problem 2.4.1 TREE WIDTH

Input: a graph $G = (V, E)$, a positive integer k.

Output: "yes" if G has a tree decomposition with tree width k and "no", otherwise.

In [5], it was shown that finding the tree decomposition with the minimum tree width for a graph is an NP-hard problem. In particular, it is shown that the LINEAR CUT problem can be reduced to the TREE WIDTH problem on a co-bipartite graph in polynomial time. Readers can refer to [5] for more details on the reduction. However, when k is a fixed parameter, it is easy to show that the problem is in FPT from the results in the graph minor theorem. In particular, since a family of graph with tree width at most k is closed under taking graph minors, according to [75], there exists a finite obstruction set for this graph family. Determining whether the tree width of a graph is bounded by k is equivalent to deciding whether a graph contains one of the graphs in the obstruction set as a minor. Since there exists an $O(f(|H|)|V|^3)$ time algorithm to decide whether a graph $G = (V, E)$ contains a minor H [73], the verification procedure needs $O(g(k)|V|^3)$ time. This proof is nonconstructive, because the obstruction set for the graph family formed by graphs with bounded tree width k remains unknown so far. In addition, $g(k)$ is a function whose value is very large even for small k's. This algorithm thus cannot be used in practice even if it is constructive.

Theorem 2.4.2 ([73]) *Given a graph $G = (V, E)$, there exists an $O(g(k)|V|^3)$ time algorithm that can decide whether the tree width of G is bounded by k or not.*

Further improvements on the results in theorem 2.4.2 have been developed in [12]. In particular, the algorithm needs $O(2^{k^2}|V|)$ time to determine if the tree width of G is bounded by k. Experiments on this algorithm have shown that it is very slow even on small parameters, i.e. $k = 4$. This algorithm thus cannot be used in practice to compute the tree width of a graph. However, as a significant improvement on the asymptotic complexity of the problem, a sketch of the algorithm is provided in the following subsection.

2.4.1 Linear Time Algorithm

In [12], results on tree widths of a few graph classes is provided. In addition, it is shown that despite of the NP-hardness of finding the minimum tree width for a graph, the decision problem of finding the tree width of a graph can be solved in linear time [13]. This section sketches this algorithm; a few definitions and theorems that are needed to show the correctness of the algorithm are also presented.

Lemma 2.4.3 ([13]) *Suppose that v and w have at least $k + 1$ common neighbors in $G = (V, E)$. If the tree width of graph G is at most k, then the tree width of $G + (v, w)$ is also at most k. Moreover, any tree decomposition of G with tree width at most k is also a tree decomposition of $G + (v, w)$ with tree width at most k and vice versa.*

This lemma provides a certain case where adding additional edges to graphs does not increase the tree width of the graph. Based on theorem 2.2.3, this lemma can be shown with a straightforward proof. It is not difficult to see that there is a complete bipartite subgraph that contains v, w and their neighbors in G. A tree decomposition for G thus either contains $\{v, w\}$ or their neighbors in a single tree node. However, the tree width of the graph is bounded by k. The latter case cannot happen and we know that v and w must be contained

in a single tree node and the tree width of the graph is thus not increased by adding an edge between v and w.

A tree decomposition $(T = (I, F), X)$ of tree width k is *smooth*, if for all $i \in I$, $|X_i| = k+1$, and for all $(i, j) \in F$, $|X_i \cap X_j| = k$. Any tree decomposition with tree width k can be transformed to a smooth tree decomposition in linear time.

Two positive constants c_1 and c_2 are chosen. c_1 is an upper bound on the fraction of high degree vertices (defined below), and c_2 denotes a lower bound on the fraction of I-simplicial vertices (defined below). They satisfy:

$$c_2 = \frac{1}{4k^2 + 12k + 16} - \frac{c_1 k^2 (k+1)}{2} \tag{2.4}$$

A vertex with degree at most $d = \max\{k^2 + 4k + 4, 2k/c_1\}$ is defined as a *low degree vertex*, and a vertex is a *high degree vertex* if it is not a low degree vertex. A low degree vertex is *friendly* if it is connected to at least another low degree vertex.

A *matching* M of a graph $G = (V, E)$ contains a set of edges such that no two edges share a vertex. A *maximal matching* is a matching such that adding a graph edge to it turns it into an edge set that is not a matching. The following two lemmas provide an estimation of the number of high degree vertices and the number of edges in a maximal matching of a graph that contains n_f friendly vertices.

Lemma 2.4.4 ([13]) *There are less than $c_1|V|$ high degree vertices in a graph with tree width k.*

Lemma 2.4.5 ([13]) *If there are n_f friendly vertices in $G = (V, E)$, then any maximal matching of G contains at least $n_f/(2d)$ edges.*

Based on a maximal matching of the graph G, we can contract edges contained in the matching and obtain a new graph G'. The tree width of the graph G' is at most that of G. Hence if G' is found to have a tree width greater than k, so is G. On the other hand, based on a tree decomposition of G' with width k, we can obtain a tree decomposition with width $2k + 1$ for graph G.

Theorem 2.4.6 ([12]) *For all k,l, there exists a linear time algorithm that when given a graph $G = (V, E)$ together with a tree decomposition (T, X) of G with tree width at most l, determine whether the tree width of G is at most k, and if so, finds a tree decomposition of G with tree width at most k.*

The *improved* graph G' of a graph $G = (V, E)$ is obtained by connecting all pairs of vertices that have at least $k + 1$ common neighbors in G. Based on Lemma 2.4.3, the tree width of the improved graph G' is unchanged. A vertex is *simplicial* if its neighbors induce a clique in G, and is *I-simplicial* if it is simplicial in the improved graph of G.

Theorem 2.4.7 ([13]) *For every graph $G = (V, E)$, at least one of the following properties holds:*

1. *G contains at least $|V|/(4k^2 + 12k + 16)$ friendly vertices.*
2. *G contains at least $c_2|V|$ I-simplicial vertices.*
3. *The tree width of G is greater than k.*

Based on Theorem 2.4.7, deciding if a graph G has a tree width at most k can be performed with the algorithm sketched as follows.

1. Compare $|E|$ with $k|V| - \frac{1}{2}k(k+1)$. If $|E|$ is larger, return "no", otherwise proceed to step 2.
2. Compute the total number of friendly vertices. If it is larger than $|V|/(4k^2+12k+16)$, proceed to step 3, otherwise to step 6.
3. Arbitrarily find a maximal matching in G; put the edges in the matching in set S.
4. Contracting edges in S and obtain a new graph G_1.

5. Call the algorithm on graph G_1. If it returns "no", return "no", otherwise let $l = 2k+1$, use the algorithm specified in Theorem 2.4.6, and the tree decomposition returned on G_1 to decide if the tree width of G is at most k. If it returns "yes", return "yes" and the tree decomposition found, otherwise return "no".

6. Determine I-simplicial vertices in G and check the degree of every I-simplicial vertices. If there is one with more than $k+1$ neighbors, return "no", otherwise proceed to step 7.

7. Count the number of I-simplicial vertices, return "no" if it is less than $c_2|V|$, otherwise proceed to step 8.

8. Remove all I-simplicial vertices from the improved graph of G and obtain a new graph G_2. Call the algorithm on graph G_2. If it returns "no", return "no", otherwise proceed to step 9.

9. For each I-simplicial vertex v in G, find in the tree decomposition of G_2 a tree node t that contains $N(v)$, form a new tree node that contains $v \cup N(v)$ and connects it to t. Return "yes" and the constructed tree decomposition.

The algorithm is a recursive algorithm since it reduces graph G to a graph with smaller size and recursively apply the algorithm to the new graph. It is clear that if the graph G contains more than $|V|/(4k^2+12k+16)$ friendly vertices, we know any maximal matching in G contains at least $|V|/2d(4k^2+12k+16)$ edges from Lemma 2.4.5. The number of vertices in graph G_1 is less than $(1 - 1/2d(4k^2 + 12k + 16))|V|$. On the other hand, if the graph contains at least $c_2|V|$ I-simplicial vertices, the number of vertices in graph G_2 is at most $(1 - c_2)|V|$. The computation time $T(n)$ of the algorithm thus must satisfy the following recursion:

$$T(n) = T(c_3 n) + O(n) \tag{2.5}$$

where c_3 is a constant between 0 and 1. Specifically, c_3 can be determined with:

$$c_3 = \max\{1 - c_2, 1 - \frac{1}{2d(4k^2 + 12k + 16)}\} \tag{2.6}$$

We thus have $T(n) = O(n)$. However, the hidden constant in the computation time is a large exponential function of k and the algorithm thus can not be used for practical purpose. An evaluation of the algorithm has shown that even for small k's, the computation time needed by the algorithm is not acceptable for any practical application. A few improved variants of the algorithm have been proposed, including the parallel variant of the algorithm in [14] and a variant in [64] that needs $O(k^2)$ recursive calls.

2.4.2 Heuristics

A few efficient heuristics have been developed to find tree decomposition or to estimate the tree width of a graph, including fill-in heuristics [7, 22] for finding an upper bound and the Maximum Cardinality Search for a lower bound [55]. Recently, a new approach developed in [15] can estimate lower bounds of tree widths using brambles [81]. This approach can achieve significantly better estimation results on planar graphs than the Maximum Cardinality Search.

Most of the available heuristics focus on the separators of a graph. A *separator* of a graph is a vertex subset such that removing vertices in the subset disconnects the graph into at least two connected components. Since tree bags in a tree decomposition are closely related to graph separators, a few heuristics have been proposed to construct tree decompositions using a separator based divide-and-conquer technique [14, 61]. Specifically, a tree decomposition of a graph G can be constructed by the following algorithm:

1. Find a minimal separator S in graph G.

2. Obtain a new graph G' by removing S from G, find all connected components $W_1, W_2, \cdots, W_k$ in G'.

3. Completely connect all vertices in S and call the algorithm recursively on graphs $Clique(S) \cup W_1$, $Clique(S) \cup W_2$, $\cdots$, $Clique(S) \cup W_k$, and obtain tree decompositions T_1, T_2, $\cdots$, T_k.

4. Form a root node R that contains all vertices in S and connect the roots of T_1, T_2, $\cdots$, T_k to R. Return the tree decomposition.

In [14], a few criteria were developed to determine whether the "dividing" made on a minimal separator increases the tree width of the graph or not. Specifically, if the maximum tree width for graphs $Clique(S) \cup W_1$, $Clique(S) \cup W_2$, $\cdots$, $Clique(S) \cup W_k$ is at most the tree width of graph G, S is a safe separator and the tree width of the graph is not increased by the dividing procedure. A minimal separator S is safe if one of the following is the case:

1. S is a clique.

2. S is an almost clique. Specifically, it contains a clique with $|S| - 1$ vertices.

3. S contains only one, two or three vertices.

The following theorem provides a general rule for determining whether a separator is safe or not.

Theorem 2.4.8 ([14]) *Suppose S is a separator in graph $G = (V, E)$, and for every component Z in $G - S$, graph $G - Z$ contains a clique on S as its minor. Then S is a safe separator for G.*

Proof: Consider the tree width of the graph $Z \cup clique(S)$. Since $G - Z$ contains a clique minor at vertices of S, it is not difficult to see that $Z \cup clique(S)$ is a minor of the graph G. Its tree width is thus not less than that of $Z \cup clique(S)$. The divide-and-conquer technique thus does not increase the tree width on the resulting subproblems, which suggests that S is a safe separator. □

2.5 Summary

The concepts of tree decomposition and tree width have provided profound insights into graph theory, algorithm research, and complexity study. In graph theory, tree decomposition provides a tool to study properties of minor-closed graph families and reveals important ones associated with such families. In algorithmic research, the concepts of tree decomposition and tree width provide efficient algorithms for many NP-hard optimization problems on graphs of bounded tree width. In complexity study, the tree width of a graph naturally serves as a parameter and a few important parameterized complexity results have been obtained, including the Planar Subgraph Isomorphism problem [30]. This dissertation only presents a few important results associated with its algorithmic implications in bioinformatics.

Although finding the tree width of a graph is NP-hard, parameterized polynomial or linear time algorithms are available to decide whether the tree width of a graph is less than a given integer parameter k or not. In practice, separator based divide-and-conquer heuristics can be used to efficiently find optimal tree decompositions for many graphs. For solving a bioinformatics problem, approximate tree decompositions affect the running time of the algorithm but not the accuracy of solutions.

Chapter 3

The Longest Antisymmetric Path[1]

Proteomics studies the dynamic roles of protein molecules in complex biological systems. Tandem mass spectrometry is an important experimental tool in proteomics that can be used to identify proteins. Many proteomics problems can be formulated as optimization problems on graphs that can model the experimental spectra obtained from tandem mass spectrometers. For example, the *De Novo* sequencing problem can be reduced to finding the longest antisymmetric path in a spectrum graph. The optimization problems are generally NP-hard. Therefore, a critical issue in solving them efficiently and exactly is to make the corresponding graphs be of small tree widths. On the other hand, we need to show that the problem itself can be expressed with MSO logic. In this Chapter, we focus on resolving these two issues.

3.1 Introduction

Tandem mass spectrometry (MS/MS) has been extensively used in proteomics to identify and analyze proteins [65]. In this method, molecules of a protein can be cleaved into short peptide sequences by enzymes. Amino acids in these peptides are then determined and combined to obtain the sequence of the protein. To sequence a peptide, sequences with the same amino acids are fragmented into charged prefix and suffix subsequences (ions) and their mass/charge ratios can be measured by a mass spectrometer.

[1]Part of the data reported in this chapter is reprinted with permission from "Fast de novo peptide sequencing and spectral alignment via tree decomposition" by C. Liu, et al. in the *Proceedings of the 11th Pacific Symposium on Biocomputing*, (PSB 2006), 11: 255-266, Copyright 2006 by World Scientific Publishing Co. Pte. Ltd., Singapore.

In an ideal MS/MS spectrum, there are usually two types of ions present: *b-ions* associated with N-terminals and *y-ions* with C-terminals. Ideally, fragmentation may occur at any position along the peptide backbone and we thus expect to be capable of inferring the amino acids a peptide contains from its MS/MS spectrum and the masses of single amino acids. However, difficulty may arise when we intend to identify the ion types for mass peaks. In addition, experimental spectra are usually incomplete and contain noisy peaks. Therefore, the *de novo* sequencing of a peptide solely from its spectrum remains a challenging task [19, 24].

A number of algorithms have been developed for the *de novo* sequencing problem. An early developed algorithm [77] generates all amino acid sequences and the corresponding theoretical spectra to be compared with the experimental spectrum. To find out the best match, an exponential number of spectra may need to be generated. The algorithm thus is not efficient. Prefix pruning approaches have been developed to speed up the search by restricting it to sequences whose prefixes match the spectrum well [83, 104, 107]. However, heuristic pruning may adversely affect the sequencing accuracy while the computation time may remain expensive. Another method, such as used by program SEQUEST [29], is to match the spectrum of an unknown peptide against those in a peptide spectrum database using correlation functions. Such a sequencing tool may fail to provide the correct answer for peptides whose spectra are not included in the database. Recently, based on the notion of spectrum graph [24], the *de novo* sequencing problem has been reduced to finding the longest (or maximum scored) antisymmetric path in directed graphs [9, 24, 31, 42, 89]. However, a straightforward path finding algorithm may yield undesired paths containing multiple vertices associated with complementary ions. This issue was resolved later with a linear time dynamic programming algorithm [19] that ensures the path found to be antisymmetric. However, it requires quadratic time to discover one modified amino acid and more time to deal with additional noisy peaks.

Comparing and evaluating the similarity between two spectra are often used in database search for peptide identification. Traditional methods for computing the similarity identify the shared mass peaks between two spectra and use the count as a measure of the similarity. More recently, spectral alignment was proposed as a new approach to evaluating spectral similarity; it proves useful for identifying related spectra in the presence of post translational modifications/mutations [65]. In particular, based on finding the longest (k-shift) path in alignment graph, a spectral alignment algorithm can align two spectra of n peaks in time $O(n^2k)$, where k is the maximum number of peak shifts resulting from post translation modifications (PTMs) [65]. However, the algorithm considers only b-ions or y-ions, but not both. To consider both ion types, a dynamic programming algorithm in the same spirit as that for *de novo* sequencing [19] is possible. But it would require a computation time that is polynomial of a much higher degree.

To provide an efficient but optimal solution for this problem, we introduce a graphical mechanism to describe related mass peaks in spectra. In particular, those peaks of the modified or associated with complementary ions are linked with non-directed edges, yielding extended spectrum graphs and extended alignment graphs. Such graphs demonstrate small tree width t (usually $t \leq 6$) for real mass spectra, so a very efficient algorithm for finding the longest antisymmetric path can be devised based on the tree decompositions of these graphs. In particular, the resulting new algorithms for *de novo* sequencing and spectral alignment run in time $O(6^t n)$ and $O(6^t n^2)$ respectively. Based on the notion of tree decomposition, the antisymmetry of complementary vertices can be efficiently ensured on the found path; both ions types can be simultaneously considered by our algorithms. In addition, using the graphical mechanism, vertices for peaks with similar masses can be easily related and considered exclusively in the tree decomposition based dynamic programming. This allows vertices for noisy peaks to be eliminated from the found path.

This algorithm has been implemented and tested on both simulated spectra and real experimental ones with noisy peaks. For *de novo* sequencing, the obtained spectrum graphs

in general have the tree width around 5. Our algorithm is able to identify the correct peptide sequences from all the tested spectra with noisy peaks in a few seconds. In particular, the algorithm achieves more than 96% accuracy on spectra in which the number of noisy peaks is the same as that of others. In addition, we used the algorithm to identify PTMs of amino acids based on spectra generated *in silico*. Our experiments showed that the alignment graph is generally sparse and its tree width ranges from 4 to 6. Experimental results for spectral alignment demonstrated that the algorithm can identify all PTMs accurately in seconds.

3.2 Problem Description

In this section, we provide a detailed description of the graph models we employed to solve the *de novo* sequencing and spectral alignment problems and the tree decomposition based algorithm for finding the longest (or maximum scored) antisymmetric paths between two given vertices. Since finding the optimal tree decomposition is an NP-hard problem, we used an efficient heuristic algorithm to find a tree decomposition with small tree width for most of the spectrum and alignment graphs.

In general, the MS/MS spectrum S of a peptide P consists of a set of mass peaks $\{x_1, x_2, \cdots, x_{2k}\}$. We assume the total number of peaks is even since for any mass peak x_i in S, there exists a mass peak x_{2k+1-i} that is complementary to x_i and the sum of their mass values is the parent mass W of P. One of x_i and x_{2k+1-i} is b-ion and the other is y-ion. A *spectrum graph* $G = (V, E)$ can be constructed based on the mass peaks in S. Specifically, vertex $v_i \in V$ represents x_i and, in addition to the mass peaks in S, vertices v_0 and v_{2k+1} are included in G for virtual mass peaks with mass values 0 and W respectively. v_i and v_j are connected with a directed edge from v_i to v_j if the mass value of x_i is less than that of x_j and the difference is the mass of a single amino acid; vertices v_i and v_{2k+1-i} are connected with an undirected edge since they represent a pair of complementary mass peaks. Sequencing a peptide from its spectrum thus corresponds to finding the longest antisymmetric directed path from v_0 to v_{2k+1}. v_0 is the *source* of the spectrum graph and v_{2k+1} is the *sink*. A path is

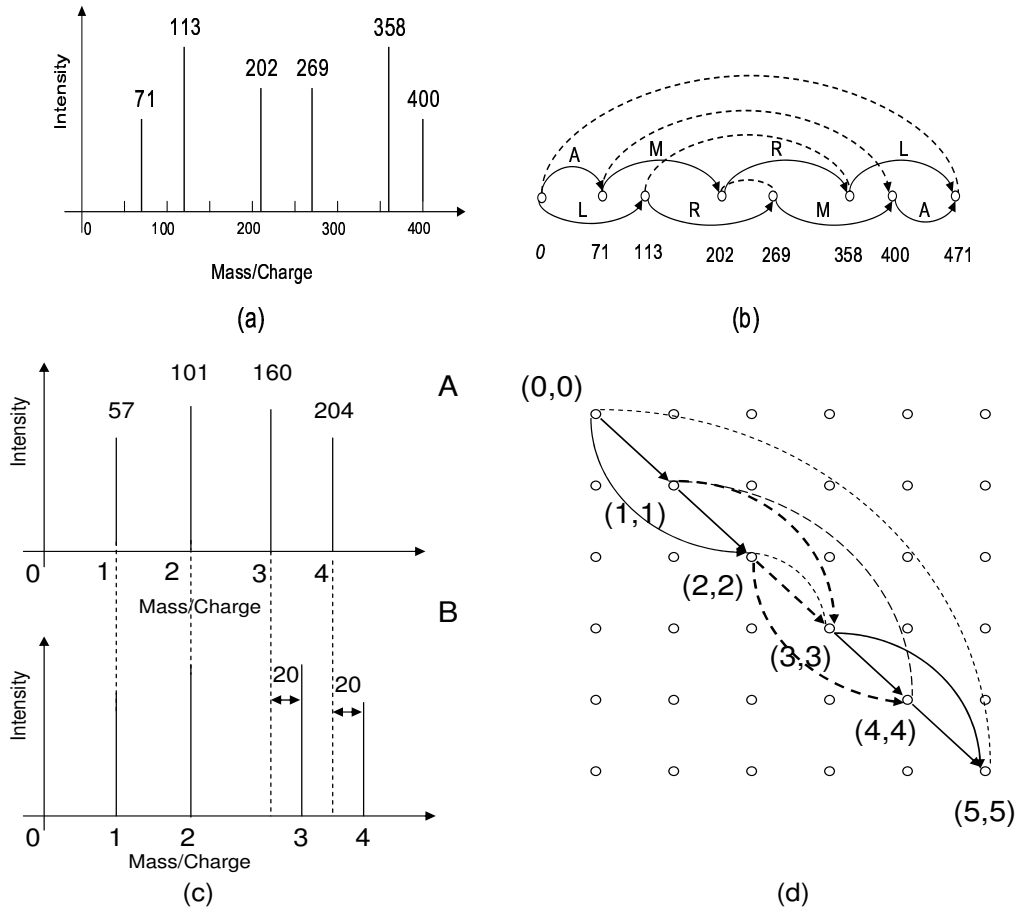

Figure 3.1: (a) The mass peaks in a tandem mass spectrum. (b) The corresponding spectrum graph, where dashed undirected edges connect complementary vertices. (c) The mass peaks in two tandem mass spectra A and B that are to be aligned. Mass peaks 3 and 4 in spectrum B have a shift of Δ in their mass values compared to those in spectrum A. (d) The alignment graph constructed based on the mass peaks in A and B. Dashed undirected edges connect complementary vertices; only edges along the diagonal vertices are drawn in the figure.

antisymmetric if it does not contain two vertices that represent complementary mass peaks. In particular, at most one of the vertices v_i and v_{2k+1-i}, where $1 \leq i \leq k$, is allowed to be present in an antisymmetric path. Figure 3.1(a)(b) provide an example for a spectrum and its corresponding spectrum graph.

Assume the sets of mass peaks for two given spectra A and B are $S_A = \{x_1, x_2, \cdots, x_{2k_1}\}$ and $S_B = \{y_1, y_2, \cdots, y_{2k_2}\}$, respectively. An *alignment* between A and B corresponds to a selection of two subsets $M_A \subseteq S_A$ and $M_B \subseteq S_B$ that satisfy:

1. $|M_A| = |M_B|$, and,
2. $x_{2k_1+1-i} \notin M_A$ if $x_i \in M_B$, where $1 \leq i \leq 2k_1$, and,
3. $y_{2k_2+1-j} \notin M_B$ if $y_j \in M_B$, where $1 \leq j \leq 2k_2$.

The *score* for a given alignment is a function $S(M_A, M_B)$ that can be computed from M_A and M_B. The *optimal alignment* of A and B is the alignment with the maximum score over all possible alignments between A and B. The objective of the spectral alignment problem is to compute the optimal alignment for the two spectra A and B based on a given scoring function $S(M_A, M_B)$. Assume for a given alignment, $|M_A| = |M_B| = k$, $M_A = \{i_1, i_2, \cdots, i_k\}$, and $M_B = \{j_1, j_2, \cdots, j_k\}$. In this dissertation, we only consider scoring functions that can be formulated as follows:

$$S(M_A, M_B) = \sum_{m=1}^{k-1} s(x_{i_m}, y_{j_m}, x_{i_{m+1}}, y_{j_{m+1}}) \tag{3.1}$$

where $s(x_{i_m}, y_{j_m}, x_{i_{m+1}}, y_{j_{m+1}})$ is often evaluated to be 1 if $x_{i_m} - x_{i_{m+1}} = y_{i_m} - y_{i_{m+1}}$, and 0 otherwise.

An *alignment graph* $G = (V, E)$ can be constructed based on the two spectra A and B that need to be aligned. Let $V = (S_A \times S_B) \cup \{(x_0, y_0), (x_{2k_1+1}, y_{2k_2+1})\}$, where x_0, x_{2k_1+1} and y_0, y_{2k_2+1} are virtual mass peaks with zero and parent mass values for spectra A and B respectively. Vertex (x_i, y_j) is connected to (x_k, y_l) with a directed edge if $x_k > x_i$ and $x_k - x_i = y_l - y_j$. Similarly, two vertices (x_i, y_j) and (x_k, y_l) are *complementary* if x_i, x_k or y_j,

y_l are complementary mass peaks. Complementary vertices are connected with undirected edges. The spectral alignment problem is thus reduced to finding in G the longest directed path that connects (x_0, y_0) and (x_{2k_1+1}, y_{2k_2+1}) and does not contain two vertices that are complementary. Figure 3.1(c)(d) provides an example of two spectra and their alignment graph. The source and sink in the alignment graph are vertices (x_0, y_0) and (x_{2k_1+1}, y_{2k_2+1}) respectively.

Both *de novo* sequencing and spectral alignment can thus be solved by finding the longest antisymmetric path in a graph where complementary vertices are connected with undirected edges. In practice, directed edges in the graph can be scored based on a few experimental parameters. The optimal results for sequencing or spectral alignment can be obtained by finding the maximum scored antisymmetric paths that connect two given vertices in the corresponding spectrum or alignment graph.

3.3 A MSO Logic Sentence for the Problem

The Longest Antisymmetric Path problem can be formulated as follows.

Problem 3.3.1 Longest Antisymmetric Path

Input: A directed acyclic graph $G = (V, E)$, two vertices s and t, and a partition $\mathcal{P}$ of vertices into disjoint pairs.

Output: The longest path that connects s and t and no two vertices in the path are in the same pair in $\mathcal{P}$.

We consider the decision version of this problem, which asks whether a given graph contains an antisymmetric path that connects s and t and its length is k. This problem can be formulated as follows.

Problem 3.3.2 Antisymmetric Path

Input: A directed acyclic graph $G = (V, E)$, two vertices s and t, and a partition $\mathcal{P}$ of vertices into disjoint pairs. A positive integer k.

Output: "yes", if there exists a length k path that connects s and t and no two vertices in the path are in the same pair in $\mathcal{P}$.

It is not difficult to show that the ANTISYMMETRIC PATH problem can be expressed with MSO logic if vertices in the same pair in $\mathcal{P}$ are connected with a graph edge, which is the case in a modified spectrum graph.

Theorem 3.3.3 *the* ANTISYMMETRIC PATH *problem can be expressed with MSO logic in a modified spectrum graph.*

Proof: To show this fact, we need to construct a MSO logic sentence that can express the ANTISYMMETRIC PATH problem. To simplify the notation, we define the following short logic sentences and combine them to get a MSO logic sentence for the problem.

$$\begin{aligned}
S_1(e_1, e, R_2) &= (e_1 \neq e) \rightarrow (\neg R_2(e_1) \vee \neg incid(e_1, u)) \\
S(u, R_2) &= ((u = s) \vee (u = t)) \rightarrow \exists e \forall e_1 (incid(e, u) \wedge R_2(e) \wedge S_1(e_1, e, R_2)) \\
T_1(u, e_1, e_2) &= (e_1 \neq e_2) \wedge incid(e_1, u) \wedge incid(e_2, u) \wedge R_2(e_1) \wedge R_2(e_2) \\
T_2(u, R_2, e, e_1, e_2) &= \forall e((e \neq e_1) \wedge (e \neq e_2) \rightarrow (\neg R_2(e) \vee \neg incid(u, e))) \\
T_3(R_1, R_2, u) &= (R_1(u) \wedge (u \neq s) \wedge (u \neq t)) \rightarrow (\exists e_1 \exists e_2 (T_1(u, e_1, e_2) \wedge T_2(u, R_2, e, e_1, e_2))) \\
A(R_1) &= \forall v((R_1(u) \wedge R_1(v)) \rightarrow \neg comp(u, v))
\end{aligned}$$

Relation $incid(e, u)$ is "true" if edge e is incident on vertex v, and "false" otherwise. Relation $comp(u, v)$ is true if vertex u and vertex v are complementary, and "false" otherwise. Relation R_1 specifies the vertices that are in the path and R_2 describes the path edges. Relations $S(u, R_2)$ combined with $S_1(e_1, e, R_2)$ suggest that for a vertex that is either the source or the sink, there exists only one edge in the resulting path incident on it. Similarly, relations $T_1(u, e_1, e_2)$ is combined with $T_2(u, R_2, e, e_1, e_2)$ and $T_3(R_1, R_2, u)$ to describe the property that for any internal node in the path, there exist and only exist two edges in the resulting path incident on it. Relation $A(R_1)$ is used to guarantee the antisymmetric property of the path.

Based on these relations, the MSO sentence for the problem is as follows.

$$\exists R_1 \exists R_2 \forall u(S(u, R_2) \wedge T_3(R_1, R_2, u) \wedge A(R_1) \wedge (size(R_2) = k)) \tag{3.2}$$

It is not difficult to verify that this MSO logic sentence correctly expresses the ANTISYMMETRIC PATH problem. In addition, from this sentence we can see that the ANTISYMMETRIC PATH problem is in MS_2. □

3.4 The Algorithm

Theorem 3.4.1 ([53]) *Given a modified spectrum graph $G = (V, E)$ and a tree decomposition of tree width t, the longest antisymmetric path that connects the source and the sink in the graph can be found in time $O(6^t|V|)$.*

Proof: Based on a tree decomposition of a graph, the longest antisymmetric directed path that connects two given vertices (source and sink) can be identified using the generic framework of dynamic programming proposed in [6, 23]. Specifically, since the source is connected to the sink with an undirected edge, there exists a tree node R in the tree decomposition such that R contains both vertices and the algorithm selects R as the root of the tree.

The algorithm maintains a dynamic programming table in each tree node to store previously computed partial optimal solutions. For a tree node with t vertices, the dynamic programming table contains $2t + 1$ columns, of which the first t columns are used to store the *selection* of each vertex in the node to form a partial optimal path.

In addition, $t - 1$ columns are needed to store the *connection state* between each pair of consecutive selected vertices in the node. Two columns V and L are the *valid bit* and the largest length of the partial path associated with the combination of selections and connection states in the same table entry. The selection value for a vertex is 1 if it is selected to be in the partial optimal path and 0 otherwise. The value for a connection state could be one of the integers in set $\{0, 1, \cdots, l\}$, where l is the number of the children of the node.

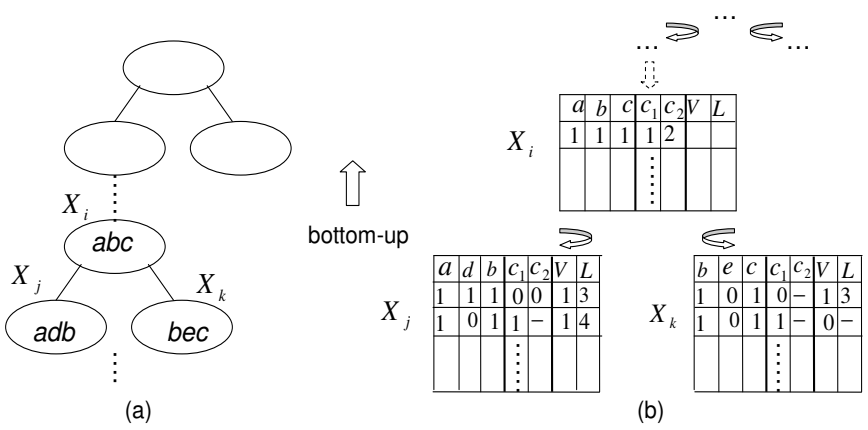

Figure 3.2: (a) The dynamic programming table for a tree node comprised of t vertices $x_1, x_2, \cdots, x_t$ in increasing order of mass values; $c_1, c_2, \cdots, c_{t-1}$ are used to store the connection states. (b) The dynamic programming tables for internal node X_i and its two children X_j and X_k. The tables of X_j and X_k need to be queried to compute the validity (V) and the largest path length (L) of a given entry in the table for X_i.

The connection state for a pair of consecutive selected vertices in the node is i if the vertices between the two vertices in the path are covered by the subtree rooted at the ith child. The number of possible combination of selections and connection states can thus be up to $(2(l+1))^t$. However, Since we can remove tree nodes with more than two children by generating extra tree nodes, the table for a tree node with t vertices may contain up to 6^t entries. The valid bit for a given entry is set to be 1 if and only if there exists a partial antisymmetric path that follow the combination of selections and connection states in the entry.

To determine the relative order of selected vertices in a partial path, the algorithm sorts the vertices in each tree node based on their corresponding mass peak values, since the mass peak values of vertices monotonously increase along any directed path from the source to the sink.

The algorithm follows a bottom-up fashion to fill the dynamic programming tables in all the tree nodes. For a leaf node, it exhaustively enumerates and directly computes the validity and largest path length for every possible combination of selections and connection states for vertices in the node. For an internal node, the algorithm refers to the tables of its children to determine the validity and largest path length for each of its table entry. In particular, for a given entry, the algorithm obtains its selections of vertices and the corresponding connection states and then queries the table contained in each of the children nodes. All valid table entries whose selections of vertices and connection states do not contradict with the considered one are queried and the one with the largest path length is selected as the *descendent entry* in the child. The algorithm sets an entry to be invalid if its selection of vertices violates the antisymmetric property or one of the children nodes contains no descendent entries. The largest path length for the entry is then computed by summing up the number of edges covered by the node itself and the largest path length for the descendent entry found in each of its child nodes.

In the last stage of the computation, the algorithm queries the table in the root node R and considers those valid entries that select both the source and the sink and finds the one with the maximum path length. The algorithm then returns the largest path length of the selected entry as the length of the longest antisymmetric directed path that connects the source and the sink. The longest directed antisymmetric path can be determined by a trace back procedure, where the descendent entries for all the internal nodes are searched in a up-bottom fashion and vertices that are selected in these entries are inserted into a list based on the mass values of their corresponding mass peaks. The directed path found by the algorithm is guaranteed to satisfy the antisymmetric property since, based on the definition of tree decomposition, any pair of complementary vertices is covered by at least one tree node. The computation time needed by the algorithm is $O(6^t N)$, where t is the tree width of the tree decomposition and N is the number of vertices in the graph. □

Figure 3.2(a) shows the dynamic programming tables for both internal nodes and leaves in the tree decomposition. Figure 3.2(b) provides an example for computing the entries in the table for an internal node X_i. Without loss of generality, we assume X_i has two child nodes X_j and X_k, and $X_i = \{a, b, c\}$, $X_j = \{a, d, b\}$ and $X_k = \{b, e, c\}$. To determine the V and L for entry $\{(1, 1, 1), (1, 2)\}$ in the table for X_i, the algorithm needs to query both of the tables for X_j and X_k since the entry suggests that the vertices on the path between a and b is covered by the subtree rooted at X_j and those between b and c is covered by that rooted at X_k. To query the table for X_j, the algorithm only checks valid entries that select both a and b since $X_i \cap X_j = \{a, b\}$, thus the leading two entries in the table for X_j are checked by the algorithm. Similarly, since $X_i \cap X_k = \{b, c\}$, the algorithm only checks valid entries that select both b and c in the table for X_k.

3.5 Experimental Results

We implemented the algorithm and tested it on both simulated and real experimental MS/MS spectra. For *de novo* sequencing, we evaluated the performance of the program on simulated spectra that contain different amount of noise, and then used it to analyze real experimental MS/MS spectra. For spectral alignment, we generated simulated spectra for peptides with Post Transcriptional Modifications (PTMs) and identified modified amino acids with spectral alignments.

3.5.1 *De Novo* Sequencing

To evaluate the performance of the program on spectra with different amount of noise, we generated simulated tandem mass spectra for 100000 fully tryptic digested peptides of proteins in the Yeast genome. We filtered out peptides of less than 5 and more than 24 amino acids. In addition to the mass peaks that result from the fragmentations of peptides, We incorporate noisy mass peaks into these simulated spectra and applied the program to obtain the sequences of amino acids for these noisy spectra. To simulate the noise generally present

Noise/Signal	Accuracy (%)	PT(< 5) (%)	PT($= 5$) (%)	PT(> 5) (%)	CT(s)
0.00	98.60	52.45	44.93	2.62	1.54
0.20	98.27	42.48	41.38	16.15	7.24
0.50	98.29	37.69	34.84	27.47	12.37
0.80	97.98	32.98	37.13	29.87	13.10
1.00	96.95	27.64	39.47	32.89	15.46

Table 3.1: The accuracy of the program on spectra with different amount of noise. Noise/Signal is the ratio of the number of noisy peaks to that of others in a spectrum. Accuracy is the percentage of amino acids that are correctly identified by the program; PT(< 5), PT($= 5$) and PT(> 5) are percentages of spectrum graphs whose tree widths are less than 5, equal to 5 and greater than 5 respectively; CT is the average amount of computation time the program needs to analyze a spectrum.

in real experimental spectra, noisy mass peaks are generated in groups and the differences of mass values for mass peaks in the same group are selected to be those of single amino acids or their combinations. Table 3.1 shows the performance of the program on spectra with different amount of noise. As we have expected, the tree widths of the spectrum graphs increases while more noisy peaks are inserted into the spectra and the program thus needs more computation time to analyze a spectrum. In addition, a slight drop in sequencing accuracy is observed when the an ideal spectrum is changed into a noisy one.

To evaluate the performance of the program on real experimental spectra, we downloaded 15 tandem mass spectra for peptides in *E. Coli* proteins from the Open Proteomics Database (OPD). We evaluated the sequencing accuracy of the program on these spectra and 3 additional experimental FT-ICR ones. Before we applied the program to a spectrum, the mass peaks in the spectrum were preprocessed and mass peaks with intensities less than 0.1 of the maximum intensity value are removed. In addition, mass peaks resulting from isotopes of carbon atoms were identified and removed from the spectrum as well. Table 3.2 shows sequencing result we obtained with the program for each spectrum and the amino acids each

Real Sequence	Obtained Sequence	TW	CT (s)
RAFDQIDNAPEEKA	RAFDQIDNAPEEQA	6	12.11
RPQFYFRT	RPQFYFRT	4	0.42
KVGEEVEIVGIKE	QVGEEVEIVGIKE	6	5.16
KMVVTLIHPIAMDDGLRF	KMVVTILHPIAMDDGIRF	5	7.30
RAGENVGVLLRG	RAGENVGVLLRG	6	13.09
KMVVTLIHPIAMDDGLRF	QMVVTIIHPIAMDDGLRF	5	6.85
KVVRTAIHALARMQHRG	KVVRTAIHAIARMQHRG	6	9.19
KFNQIGSLTETLAAIKM	KFNQIGSLTETLAAIQM	6	10.27
RKFATQYMNLFGIKQ	RKFATQYMNLFGIKK	6	10.15
KTQLIDVIAEKA	QTQIIDVIAEKA	5	5.09
KPVYSNGQAVKD	KPVYSNGQAVQD	5	2.53
KLNIDQNPGTAPKY	KINIDQNPGTAPQY	6	5.80
KNQTLALVSSRP	QNQTLALVSSRP	6	4.85
RVKSQAIEGLVKA	RVKSQAIEGLVQA	6	4.10
HGTVVLTALGGILK	HGTVVLTAIGGILQ	4	0.22
VEADIAGHGQEVLIR	VEADIAGHGQEVLLR	6	10.34
DAFLGSFLYEYSR	DAFLGSFLYEYSR	5	2.18

Table 3.2: The performance of the program on real experimental spectra. TW is the tree width of the spectrum graph; CT is the computation time of the program.

peptide really contains. Some sequencing errors can be seen from the table. However, the reason for these errors are clear, the program is unable to identify I from L and K from Q since the mass of I is equal to that of L, and the mass difference between K and Q is too small to be recognized by the program. The program is thus able to identify the correct peptide sequences for all the experimental spectra tested in a few seconds.

3.5.2 Spectral Alignment

As an application of spectral alignment, we used the program to identify modified amino acids on peptide sequences with PTMs. We generated pairs of spectra *in silico* from unmodified and modified peptides respectively and perform a spectral alignment between the spectra gener-

Peptide	Modified Peptide	TW	CT (s)
RAIKNLL	RAIK*NLL	4	0.04
FKMKRTQVFWKV	FK*MKRTQVFWK*V	6	2.43
MALPFQLLRQLGVA	M*ALPFQLLRQLGVA	4	0.12
AKYEGGL	AK*YEGGL	4	0.07
DFLIKRGV	DFLIK*RGV	5	0.73
PKDMILLFATTTTKF	PK*DMILLFATTTTK*F	6	2.31
LWEVKDRTAHS	LWEVK*DRTAHS	6	3.50
TGIAVDKPQSDSRMNAAP	TGIAVDK*PQSDSRM*NAAP	4	0.08
MAIVMGRLEVKAIS	MAIVMGRLEVK*AIS	4	0.12
FVPGQKNGIKGDLS	FVPGQK*NGIK*GDLS	4	0.04

Table 3.3: The performance of the program on identifying modified amino acids using spectral alignment. DNM is the number of modified amino acids identified by the program; TW is the tree width of the alignment graph; CT is the computation time in seconds.

ated. The mass modifications can be identified from the longest antisymmetric path found by the program. We introduced two additional parameters, k and Δ, where k is the maximum number of modifications allowed in the peptide and Δ is the maximum amount of mass modification that may occur on a single base. Based on parameters k and Δ, vertices (x_i, y_j) and $(x_{2k_1+1-i}, y_{j'})$ are connected with an undirected edge if and only if $|y_{j'} - y_{2k_2+1-j}| \leq k\Delta$. In addition, to incorporate possible PTMs in the alignment graph, two vertices (x_i, y_j) and $(x_{i'}, y_{j'})$ are connected with a directed edge if $x_{i'} > x_i$, $y_{j'} > y_j$ and the mass difference between $|x_{i'} - x_i|$ and $|y_{j'} - y_j|$ is less than Δ. In our experiment, The values of k and Δ were set to be 3 and 20.0. Table 3.3 shows the results we have obtained on identifying modified amino acids on pairs of spectra we generated *in silico*. It can be seen from the table that the tree width of an alignment graph ranges from 4 to 6 and the program is able to identify the modified amino acids in a few seconds.

3.6 Summary

In this chapter, we have developed a tree decomposition based algorithm that can efficiently solve both problems of *de novo* sequencing and spectral alignment. Based on the notions of spectrum and alignment graph, both problems can be reduced to finding the longest (or maximum weighted) antisymmetric directed path that connects the source and sink in the graph, a problem that can be solved with a linear time dynamic programming algorithm on graphs with bounded tree width. We provide a detailed description of the algorithm and show that the *de novo* sequencing and spectral alignment can be solved with time $O(6^t n)$ and $O(6^t n^2)$ respectively, where t is the tree width of the corresponding graph and n is the number of mass peaks in the spectrum (spectra).

The algorithm has been implemented and tested on both spectra generated *in silico* and real experimental ones. The results show that, for *de novo* sequencing, the tree width of a spectrum graph is around 5 and may slightly increase when the spectrum contains a large number of noisy mass peaks. The program is able to identify the correct peptide sequences in the presence of noisy mass peaks in a few seconds. Only a slight drop in sequencing accuracy is observed while the amount of noise in the spectra increases; for spectral alignment, the tree width of a spectral alignment ranges from 4 to 6 and the program can correctly identify all the modified amino acids on all the tested peptides with PTMs.

In this work, we only considered the mass differences between mass peaks and have yet incorporated an edge-scoring scheme to evaluate the relative weights of different graph edges. In addition, more experiments are needed to evaluate the performance of the program on real spectra for both *de novo* sequencing and spectral alignment. The construction and trimming of a spectrum or alignment graph may have significant effect on its tree width and thus deserves further investigation.

Chapter 4

Maximum Antisymmetric Paths[1]

4.1 Introduction

As discussed in chapter 3, it is a challenging problem to determine the amino acid sequence of a protein peptide from a tandem mass spectrum. The problem becomes more difficult when the spectrum contains post-translational modifications (PTMs). Existing computational methodologies for solving this problem can be classified into two major categories: database search based approaches and *de novo* peptide sequencing. Database search based tools such as Sequest [29] and Mascot [63] compare a query spectrum with spectra from peptide sequences in a database and output those with high correlation scores as sequencing candidates. When the query spectrum contains PTMs, it becomes very difficult to select the correct peptide sequence since calculation becomes prohibitively slow, due to the enumeration and scoring of all possible modifications for each peptide from the database. In contrast, *de novo* sequencing methods [19, 24, 31, 41, 42, 53, 56, 79, 89, 101] aim to infer a peptide sequence from its spectrum directly without looking up a protein database. However, the accuracy of *de novo* sequencing is highly sensitive to the quality of the input spectrum. Usually it cannot infer a full length peptide sequence due to missing peaks, which consequently limits its application in practice.

Most of existing approaches [63, 88, 95, 104] for identifying PTMs assume a limited set of modification types. These modification types can be modeled with pseudo amino acids;

[1]Part of the data reported in this chapter is reprinted with permission from "Peptide sequence tag based blind identification of post-translational modifications with point process model" by C. Liu, et al. in the *Proceedings of the 14th International Conference on Intelligent Systems for Molecular Biology*, (ISMB 2006), *Bioinformatics*, Copyright 2006 by Oxford University Press.

approaches developed for spectra free of PTMs can thus be directly applied to those with PTMs. However, spectra with unknown types of modifications may be erroneously processed with this method. Recently, a few approaches have been proposed for blind PTM identification [91, 102]. Both approaches optimally align an experimental spectrum with the spectra from peptide sequences in a database by allowing a shift of certain amount. In particular, [91] proposes a dynamic programming algorithm to solve this problem. Alternatively, [102] introduces a point process model to process a spectrum, in which all possible optimal alignments between two spectra are obtained feasibly by computing the correlation of their corresponding processes. Both approaches are effective and able to detect unknown types of modifications. However, due to the large size of the search space, the optimal spectral alignment may be very time consuming and both approaches may suffer high false positive rate and computing inefficiency.

Recently, the idea of database filtration based on peptide sequence tags has been introduced to speed up peptide database search [32, 87]. For example, based on a fragmentation model, GutenTag [87] generates many short sequence tags that are possibly contained in the peptide for a spectrum and compares the spectrum with ones from peptide sequences in a database that contain at least one of the selected tags. PepNovo [32, 34] evaluates the reliability of sequence tags on *de novo* sequencing results with a machine learning based approach and uses high reliable sequence tags to filter out most of the peptide sequences in a peptide database. With the reduced search space, the correct peptides can thus be identified efficiently with the conventional database search methods. Apparently, the methodology of combining *de novo* peptide sequencing and database search can dramatically improve the efficiency of peptide identification without sacrificing too much sensitivity. It thus may represent a promising approach for rapid and reliable peptide identification. However, the presence of PTMs significantly increases the difficulty of both *de novo* sequencing and database search. It is unclear what capability of these tools could be for generating correct peptide sequence

tags and further finding out the correct PTMs through database search in the presence of PTMs.

In this chapter, we introduce an *ab initio* approach to sequence tag selection, which further combines with the point process model [102], yields an efficient and accurate method for blind PTM identification. We have observed from our previous work [53] that the sequence tags can be selected from the maximum weighted antisymmetric path in a spectrum graph. Due to missing peaks or the shift of peaks in a spectrum that contains PTMs, a *de novo* sequencing algorithm may not be able to find a fully connected antisymmetric path that explains the spectrum. Nevertheless, it is possible to find all maximum weighted antisymmetric paths between certain pairs of vertices in the spectrum graph to obtain partial knowledge of the amino acid sequence of the spectrum. To efficiently implement this idea, we propose a novel tree decomposition based algorithm that can efficiently and effectively find all maximum weighted antisymmetric paths in a spectrum graph. We use the notion of *extended spectrum graph* that contains additional edges to describe the relationships between pairs of *complementary* vertices. Such a graph can deal with spectra with the presence of both b-ions and y-ions and ensure the antisymmetric property of the paths.

The algorithm consists of two major components. The fundamental component computes the maximum weighted antisymmetric paths connecting each pair of vertices contained in each tree node from a tree decomposition of the spectrum graph. Different tree decompositions are then generated from the fundamental component to find all maximum weighted antisymmetric paths between certain pairs of vertices. The time complexity of the algorithm is $O(6^t n(n+m))$, where t is the tree width of the tree decomposition and is usually small, n is the number of peaks in the spectrum, and m is the number of maximum weighted antisymmetric paths. Sequence tags [32, 34] are then selected from all maximum weighted antisymmetric paths and their reliabilities are evaluated with a score function.

We implemented our algorithm and applied it to PTM identifications. We first generated sequence tags from 2657 experimental yeast spectra downloaded from the Open Proteomics

Database (OPD) [66]. We compared the accuracy of the sequence tags with those generated by the popular tool PepNovo. Our experiments shows that our *ab initio* tag generation algorithm is significantly faster than PepNovo with comparable accuracies. We then manually added PTMs to 2620 spectra from the same data set and used our program to generate sequence tags and filter a yeast peptide database with a deterministic finite automaton (DFA) based model. The point process blind search model was then applied to the selected candidate peptides to identify the PTMs. The experiments on the spectra with PTMs show that, compared with the results without database filtration, this combined approach can achieve significantly improved accuracy with 10 times and 80 times of speedups using the filtration of sequence tags of lengths 3 and 4 respectively.

4.2 Models and Algorithms

4.2.1 Extended Spectrum Graph and Sequence Tag Selection Problem

In this chapter, we consider all possible ions that a spectrum can contain, the graph constructed for a spectrum is thus slightly different from the one in chapter 4. In particular, we consider all types of ions listed in [65]. Although a spectrum may contain a few different types of ions, all ions in the spectrum can be classified into two categories: N-terminal ions and C-terminal ions. For simplicity, we use b-ions and y-ions to represent them respectively. We assume $S = \{s_1, s_2, \cdots, s_m\}$ to be an experimental spectrum with complementary ions added if they are missing in the original experimental spectra. The possible mass values for the partial peptide for a peak s_i in the spectrum S form a set $V_i = \{s_i+\delta_1, s_i+\delta_2, \cdots, s_i+\delta_k\}$, where δ_k is the mass offset of ion i in the form of ion type k. Each of the mass values in V_i can be represented with a graph vertex and a vertex set $V = \{v_0\} \cup \bigcup_{i=1}^{m} V_i \cup \{v_n\}$ can thus be generated for S, where v_0 and v_n are two additional vertices with zero mass and the parent peptide mass respectively. A *spectrum graph* [24] can be constructed upon V by connecting a directed edge from u to v if the mass difference between them is the mass of

a single amino acid and the mass of u is less than that of v. u is an *in-neighbor* of vertex v and v is an *out-neighbor* of vertex u.

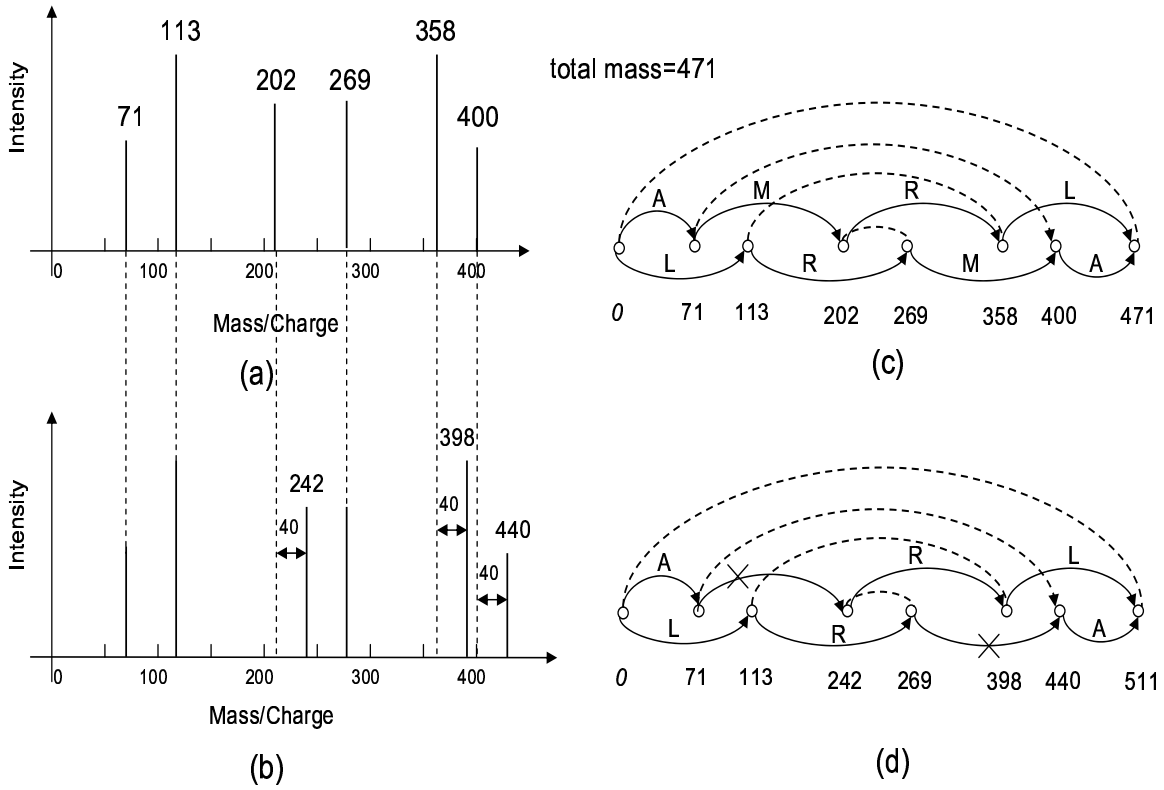

Figure 4.1: (a) A tandem mass spectrum for a short peptide AMRL; for simplicity, only b and y-ions are included. (b) the spectrum for the same peptide, but with a PTM on amino acid M. (c) The extended spectrum graph for the spectrum in (a) and a longest antisymmetric sequencing path; dashed undirected edges connect complementary vertices. (d) The extended spectrum graph for the spectrum in (c), the original antisymmetric path for sequencing is disconnected due to the modification.

Based on a stochastic model for ions and peaks in a spectrum, vertices and edges in a spectrum graph can be assigned weights. Traditional approaches for *de novo* sequencing determine the amino acid sequence of a peptide by finding the maximum weighted path in the spectrum graph that connects v_0 and v_n. However, since a valid sequencing path only contains either b-ions or y-ions, it is necessary to identify pairs of vertices that cannot appear in the same sequencing path. Given a partition $\mathcal{P}$ that contain disjoint vertex pairs, a pair of vertices are *complementary with respect to* $\mathcal{P}$ if the vertices are in the same pair. A path in

a spectrum graph is *antisymmetric with respect to* $\mathcal{P}$ if it contains at most one vertex from each pair of complementary vertices in $\mathcal{P}$. A valid sequencing path is thus the maximum weighted antisymmetric path that connects v_0 and v_n. To address this issue, in addition to the directed edges in a spectrum graph, we also connect complementary vertices in the spectrum graph with undirected edges, yielding an *extended spectrum graph* [53]. we show later in this chapter that these undirected edges are important to ensure the antisymmetry of the paths found by our algorithm. Figure 4.1(a)(c) show the spectrum of a short peptide and an *de novo* antisymmetric sequencing path contained in the corresponding extended spectrum graph.

For most of the spectra that contain PTMs, an antisymmetric path that connects v_0 and v_n may not exist in each of the corresponding spectrum graphs. As an example, Figure 4.1(b)(d) show a shift of peaks and the spectrum graph of the peptide with a PTM on one of its amino acids. However, we observe that parts of the amino acid sequence of the peptide can be obtained from maximum weighted antisymmetric paths between certain pairs of vertices. A path P in a spectrum graph is *maximum weighted antisymmetric* if it satisfies the following constraints:

1. P is antisymmetric,
2. if u, v are the two ends of the path, any antisymmetric path P_1 that connects u and v has a weight no larger than that of P,
3. there does not exist an antisymmetric path P_2 in the graph such that $P \subset P_2$.

4.2.2 A MSO Logic Sentence for the Problem

We consider the following k Maximal Antisymmetric Path problem.

Problem 4.2.1 k Maximal Antisymmetric Path

Input: A directed acyclic graph $G = (V, E)$, a positive integer k, $m, n \in V$ and a partition $\mathcal{P}$ of V into disjoint pairs.

Output: "yes" if there exists a nonextendable antisymmetric path that connects m and n and it's weight is k; "no" otherwise.

A simple reduction from the Antisymmetric Path problem can show that the Maximum Antisymmetric Path problem is NP-hard. We thus have the following theorem.

Theorem 4.2.2 *k Maximal Antisymmetric Path problem is NP-complete on general graphs.*

Proof: It is obvious that this problem is in NP. Given an instance $\langle G, k, s, t, \mathcal{P} \rangle$ of Antisymmetric Path problem, since s and t are the source and the sink in the graph G, if we assign each edge in G weight 1, the antisymmetric path of length k in G is exactly the antisymmetric path of weight k between s and t. This path obviously cannot be further extended. □

It is not difficult to modify the MSO logic sentence for the Antisymmetric Path problem to obtain a MSO logic sentence for the k Maximal Antisymmetric Path problem.

Theorem 4.2.3 *k Maximal Antisymmetric Path problem can be expressed with a MSO logic sentence.*

Proof: Similar to the proof in theorem 3.3, we add two additional intermediate relations to express the property of being nonextendable. In particular, we have the following

intermediate relations.

$$
\begin{aligned}
S_1(e_1, e, R_2) &= (e_1 \neq e) \rightarrow (\neg R_2(e_1) \vee \neg incid(e_1, u)) \\
S(u, R_2) &= ((u = m) \vee (u = n)) \rightarrow \exists e \forall e_1 (incid(e, u) \wedge R_2(e) \wedge S_1(e_1, e, R_2)) \\
T_1(u, e_1, e_2) &= (e_1 \neq e_2) \wedge incid(u, e_1) \wedge incid(u, e_2) \wedge R_2(e_1) \wedge R_2(e_2) \\
T_2(u, R_2, e, e_1, e_2) &= \forall e((e \neq e_1) \wedge (e \neq e_2) \rightarrow (\neg R_2(e) \vee \neg incid(u, e))) \\
T_3(R_1, R_2, u) &= (R_1(u) \wedge (u \neq m) \wedge (u \neq n)) \rightarrow (\exists e_1 \exists e_2 (T_1(u, e_1, e_2) \wedge T_2(u, R_2, e, e_1, e_2))) \\
A(R_1) &= \forall v((R_1(u) \wedge R_1(v)) \rightarrow \neg comp(u, v)) \\
E_1(u) &= (u = m) \rightarrow (\forall v(edge(v, u) \rightarrow (\exists q(R_1(q) \wedge comp(q, v))))) \\
E_2(u) &= (u = n) \rightarrow (\forall v(edge(u, v) \rightarrow (\exists q(R_1(q) \wedge comp(q, v)))))
\end{aligned}
$$

Compared with the proof of theorem 3.3, the two additional relations $E_1(u)$ and $E_2(u)$ are used to describe the "maximal" property of a path. More specifically, $E_1(u)$ denotes that the path cannot be further extended from the left side while maintaining the antisymmetric property and $E_2(u)$ denotes the same property on the right side of the path. Based on these relations, the MSO sentence for the problem is as follows.

$$\exists R_1 \exists R_2 \forall u(S(u, R_2) \wedge T_3(R_1, R_2, u) \wedge A(R_1) \wedge (size(R_2) = k) \wedge E_1(u) \wedge E_2(u)) \qquad (4.1)$$

It is not difficult to verify that this MSO logic sentence correctly expresses the Maximal Antisymmetric Path problem. In addition, from this sentence we can see that the K Maximal Antisymmetric Path problem is in MS_2. □

4.2.3 Algorithms for Path Finding

The algorithm for finding all maximum weighted antisymmetric paths consist of two major components. In particular, in a given tree decomposition, the fundamental component that connects each pair of the vertices contained in each tree node. To find all maximum weighted antisymmetric paths connecting certain pairs of vertices in the graph, the algorithm generates different tree decompositions and applies the procedure of the fundamental component to

each of them. The overall time complexity of the algorithm is $O(6^t n(n+m))$, where t is the tree width, n is the number of vertices in the spectrum graph, and m is the number of maximum weighted antisymmetric paths in the spectrum graph.

The Fundamental Component

The fundamental part is basically the algorithm described in the proof of theorem 3.4.1 in chapter 4. In particular, the algorithm arbitrarily selects a tree node as the root of a tree decomposition and maintains a dynamic programming table for each tree node. It then proceeds from leaves of the tree to the root to fill in all the dynamic programming tables. The table for each tree node stores the weight of the partial maximum weighted antisymmetric path connecting each pair of vertices in the tree node.

For a tree node with t vertices, the dynamic programming table contains $2t+1$ columns, of which the first t columns store the *selection* of each vertex in the node to form a subpath and the other $t-1$ columns are used to store the *connection state* between each pair of consecutive selected vertices in the tree node. Two additional columns V and L store the *valid bit* and the maximum weight of the partial antisymmetric path associated with a combination of selections and connection states in the same table entry, respectively.

The selection value of a vertex in a tree node is 1 if it is selected to be in the partial optimal path and 0 otherwise. The value of a connection state could be one of the integers in set $\{0, 1, \cdots, l\}$, where l is the number of children of the tree node. The connection state for a pair of consecutive selected vertices in the tree node is 0 if they are contiguous in the path and is i ($i > 0$) if the vertices on the path between the pair of vertices are covered by the subtree rooted at the ith child. The number of possible combinations of selections and connection states can thus be up to $(2(l+1))^t$. However, since we can remove tree nodes with more than two children by generating extra tree nodes, the table for a tree node with t vertices may contain up to 6^t entries. The valid bit for a given entry is set to be 1 if there

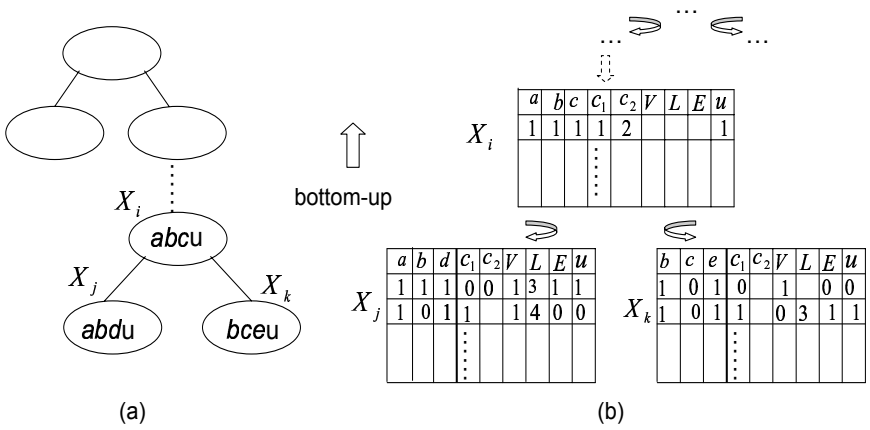

Figure 4.2: A tree decomposition and its corresponding dynamic programming tables. The algorithm follows a bottom-up fashion starting with the leaf tree nodes. When computing the dynamic programming tables for an internal node X_i, the tables of its child nodes X_j and X_k need to be queried to compute the validity (V), the maximum path weight (L) and the extendibility (E) of a given entry in the table for X_i; vertex u is added to each tree bag to compute all the maximum weighted antisymmetric paths that start with it.

exists a partial antisymmetric path that follows the combination of selections and connection states in the entry.

To determine an entry in the table for a leaf node, the algorithm exhaustively enumerates and directly computes the validity and the maximum path weight for every possible combination of selections and connection states for vertices in the node. For an internal node, the algorithm refers to the tables of its children to determine the validity and the maximum path weight for each of its table entry. Figure 4.2 provides an example for computing the table entries for an internal node X_i. The computation time needed by the algorithm is $O(6^t n)$, where t is the tree width of the tree decomposition and n is the number of vertices in the graph.

Finding All Maximum Weighted Antisymmetric Paths

Theorem 4.2.4 *Given an extended spectrum graph $G = (V, E)$ and a tree decomposition of G with tree width t, all maximum weighted antisymmetric paths in G can be identified in time $O(6^t|V|(|V| + m))$, where m is the total number of maximum weighted antisymmetric paths in G.*

Proof: The fundamental component can only compute the maximum antisymmetric paths between vertices that are included in at least one tree node. Further processing is thus needed to find all maximum weighted antisymmetric paths in the spectrum graph. For each vertex u in the spectrum graph, a new tree decomposition can be constructed by including u in all the tree nodes in the original tree decomposition. n different tree decompositions are thus generated.

To guarantee that the paths found by the algorithm satisfy the constraints of being maximum weighted antisymmetric, we further modify the previously described fundamental component. Specifically, as shown in Figure 4.2, one additional column E is added to each dynamic programming table to indicate whether the corresponding path can be extended to form an antisymmetric path with a larger weight by adding one of the in-neighbors of u to the path. This bit is set to be 1 if such an extension exists and 0 otherwise. The algorithm also sets the E bit to be 0 if the corresponding path for an entry does not start with u. This property for a given entry can be obtained by combining the E bits of its descendent entries with a direct inspection on vertices that are not included in the descendent entries. In view of the fact that the number of in-neighbors of u is bounded by 20, the aggregate computation time for this additional checking is $O(6^t n)$.

Based on the E bit of an entry, we are able to select the antisymmetric paths that start with u and cannot be extended with an in-neighbor of u. However, it is possible that some of the maximum antisymmetric paths can be extended from the other end of the path. We thus create an array S of size n and initialize all its elements to be zero. For any vertex v other than u in the spectrum graph, we can obtain $W(u, v)$, the weight of

the maximum weighted antisymmetric path that connects u and v. For each out-neighbor v_t of v, we obtain $W(u, v, v_t)$, the weight of the maximum weighted antisymmetric path that passes through v and connects u and v_t. We then check whether $W(u, v, v_t)$ is equal to $W(u, v) + w(v, v_t)$ or not, where $w(v, v_t)$ is the weight of the edge (v, v_t). If it is the case for one out-neighbor of v, we set $S[v]$ to be 1, which suggests that the maximum antisymmetric path between u and v is extendable. The correctness of this operation is obvious since only in the case where the path is extendable, we can have one out-neighbor v_t of v such that $W(u, v, v_t) = W(u, v) + w(v, v_t)$. The aggregate time for this operation is again $O(6^t n)$. We then apply the tracing back procedure in the fundamental component to obtain all the maximum weighted antisymmetric paths starting with u. Based on the E bits and the array S, we can find all maximum weighted antisymmetric paths from the n tree decompositions and the total computation time is $O(6^t n(n+m))$, where m is the total number of maximum weighted antisymmetric paths. We thus have proved theorem. □

4.2.4 Reliability of Sequence Tags

We used the scoring scheme proposed in [24] to assign weights to the vertices and edges in the extended spectrum graphs. The overall reliability of a sequence tag t_i was considered as a linear combination of normalized reliabilities $r_1(t_i)$ and $r_2(t_i)$ computed from the weights of the corresponding edges for t_i and an autocorrelation score developed in [51] respectively. In particular, the reliability $r(t_i)$ of sequence tag t_i is

$$r(t_i) = w_1 r_1(t_i) + w_2 r_2(t_i) \tag{4.2}$$

where $r_1(t_i)$ and $r_2(t_i)$ are computed with

$$r_1(t_i) = \frac{W(t_i)}{\sum_{l=1}^{q} W(t_l)} \tag{4.3}$$

$$r_2(t_i) = \frac{A(t_i)}{\sum_{l=1}^{q} A(t_l)} \tag{4.4}$$

where $W(t_i)$ is the sum of the weights of the edges that form t_i in the extended spectrum graph, q is the number of sequence tags, and $A(t_i)$ is an autocorrelation score computed with

$$A(t_i) = \sum_{k \in P(t_i)} I^*(k) I^*(n-k) \tag{4.5}$$

where $P(t_i)$ is the set of peaks that form t_i and $I^*(k)$ and $I^*(n-k)$ are adjusted intensities of complementary peaks k and $n-k$ in the spectrum. Both $r_1(t_i)$ and $r_2(t_i)$ are obtained by normalizing $W(t_i)$ and $A(t_i)$ over all sequence tags that are selected from the maximum weighted antisymmetric paths.

4.2.5 Database Filtration with Sequence Tags

From the generated peptide sequence tags, we introduced a deterministic finite automaton (DFA) based model and used it to search a yeast peptide database which consists of $670k$ tryptically digested peptides (allowed up to 2 missing cleavages). Each amino acid in the tags represents a state of the DFA. We added an additional state as the start state. The accept states are the states that correspond to the last amino acids in the tags. Upon reading the first amino acid in a peptide sequence in the database, the DFA transfers from the start state to an appropriate state that corresponds to the first amino acid in a tag. The DFA then transfers from that state to next appropriate state upon reading the following amino acid in the peptide sequence. The procedure continues until the end of the peptide sequence. The peptide reading always goes forwards in the entire procedure. However, a trie based model [32] needs to go back to the head of the trie each time when a substring of tag length in the peptide sequence has been examined. It thus may need more computation time than our DFA based model.

4.2.6 PTM Identification by Point Process Blind Search

We finally apply the point process model for peptide identification and PTM search [102] on the candidate peptides after filtration. This model is an efficient blind search approach that

does not require a list of pre-specified PTMs as input in advance. The algorithm attempts to find a set of optimal mass shifts to maximize the spectral alignment. Through one round of cross-correlation calculation, it is able to obtain all possible mass shifts feasibly (naturally includes the optimal mass shifts). The computation time is independent of the number of PTMs, which outperforms most of the existing PTM identification tools whose computation time grows exponentially with the number of PTMs.

In particular, the mass peaks in a MS/MS spectrum can be expressed with a point process.

$$x(t) = \sum_{i=1}^{N} \delta(t - t_i) \tag{4.6}$$

where t_i's $(1 \leq i \leq N)$ represent the mass values for N mass peaks and δ is the Kronecker delta function [62].

To model K mass shifts, we can group the N mass peaks in a spectrum into $K + 1$ different groups G_i, $(1 \leq i \leq K + 1)$ and for G_i, each of its mass peaks is shifted by a quantity Δ_i due to a PTM. We define for $x_i(t)$, $y_i(t)$ and $y(t)$ as follows.

$$\begin{aligned} x_i(t) &= \sum_{t_u \in G_i} \delta(t - t_u) \\ y_i(t) &= x_i(t - \Delta_i) \\ y(t) &= \sum_{i=0}^{K} y_i(t) \end{aligned}$$

It is clear that $x(t)$ is the spectrum of a peptide without PTMs and $y(t)$ is the spectrum of a peptide that may contain up to K PTMs. We now introduce a functional $C[.]$ to count the number of peaks contained in an expression of kronecker's functions. In particular, $C[x(t)] = N$ and $C[x_i(t)] = |G_i|$. We now consider the number of peaks in $x(t - \Delta_i)y(t)$. In particular, based on the following property for the Kronecker delta function.

$$\delta(t - t_i)\delta(t + \tau - t_j) = \begin{cases} \delta(t - t_i) & \text{if } \tau = t_j - t_i \\ 0 & \text{otherwise} \end{cases}$$

we are able to get

$$C[x(t - \Delta_i)y(t)] = C[y_i(t)y_i(t)] = |G_i| \tag{4.7}$$

Assuming that $c_{xy}(\tau) = C[x(t-\tau)y(t)]/N$, if we also take into consideration the coincidence of mass peaks due to the shifting, we are able to obtain the following equations.

$$\begin{aligned} c_{xy}(\tau)|_{\tau=\Delta_i} &= \frac{|G_i|}{N} \\ c_{xy}(\tau)|_{\tau=(t_j^i+\Delta_i)-t_k^l} &= \frac{1}{N} \end{aligned}$$

The second equation considers the coincidence of peaks due to the shifting. For the rest of the τ's, $c_{xy}(\tau) = 0$. Now, We can identify the mass values of PTMs by finding the peaks of $c_{xy}(\tau)$.

4.3 Experiments and Discussion

4.3.1 Datasets

We downloaded 2657 annotated yeast ion trap tandem mass spectra from the Open Proteomics Database (OPD) [66]. These spectra were selected based on the criteria with +2 precursor ion and $Xcorr \geq 2.5$ without PTMs. All the experimental mass spectra were ion trap data having a relative low mass resolution. We ran a data preprocessing procedure as described in [34] to remove isotopic peaks and tiny noise peaks. Due to short of the reliably annotated spectra with PTMs in public domain, we constructed 2620 modified ones from those spectra by artificially adding one PTM from a common PTM pool to each spectrum. The detailed procedure is referred to [102].

4.3.2 Tree Widths for Spectrum Graphs

Computing the optimal tree decomposition for a given graph is an NP-hard problem [5]. A few efficient heuristics [11] have been developed to compute a tree decomposition with small tree width for certain types of graphs. We used a greedy fill-in heuristic [11] to find tree decompositions for the spectrum graphs of the experimental spectra. Figure 4.3 shows the distribution of the tree widths of the 2657 spectrum graphs. It can be clearly seen from the

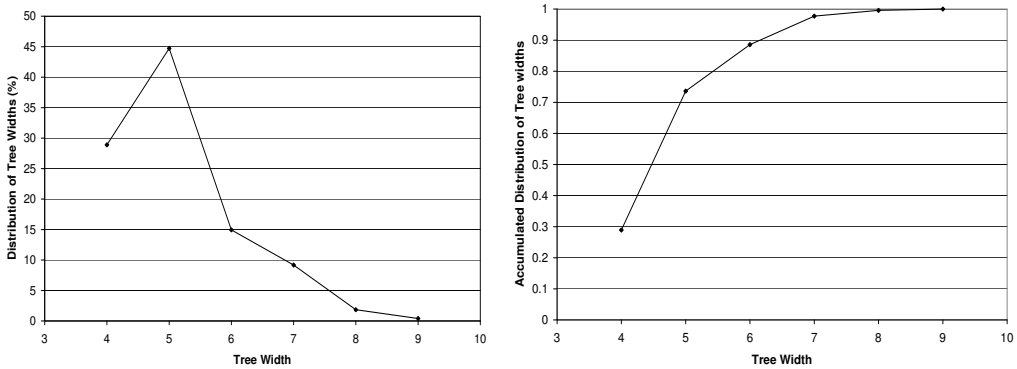

Figure 4.3: The tree widths of the extended spectrum graphs for 2657 experimental spectra; left: the distribution of the tree widths, right: the cumulative distribution of the tree widths.

figure that the tree widths of about 90% of the spectrum graphs are around 5, which are sufficiently small for developing an efficient tree decomposition based algorithm.

4.3.3 Sequence Tag Generation

We used our program to generate sequence tags at different lengths on the two datasets. We also ran the public available program PepNovo on the same datasets to obtain sequence tags. We then compared the generated tags with the sequencing results by SEQUEST and obtained the percentages of correct tags at different lengths for both of our program and PepNovo. Table 4.1 lists the results of our experiments on the two datasets and the comparison with PepNovo at different tag lengths. Our approach achieves comparable performance to PepNovo while is more computationally efficient: over 10 times faster than PepNovo at all different tag lengths. More importantly, our approach is *ab initio* and does not require a training data set as PepNovo does. We believe further improvements in accuracy can be achieved if a more sophisticated model to evaluate the reliabilities of generated sequence tags is applied in the future.

	TL	AL	$r = 1(\%)$	$r = 3(\%)$	$r = 5(\%)$	$r = 10(\%)$	$r = 25(\%)$	T (s)
a	3	TDtag	75.8	89.1	94.6	96.9	98.1	0.33
		PepNovo	75.8	90.1	93.6	96.8	98.8	3.62
	4	TDtag	65.3	80.5	88.7	93.6	96.4	0.34
		PepNovo	65.5	81.0	86.6	92.3	95.3	3.69
	5	TDtag	56.4	72.8	78.3	85.1	89.8	0.33
		PepNovo	58.4	71.3	77.6	84.0	88.9	3.83
	6	TDtag	50.2	62.3	66.9	76.6	82.4	0.34
		PepNovo	49.7	61.5	67.8	75.0	81.8	4.27
b	3	TDtag	68.1	84.8	90.3	94.8	97.1	0.32
		PepNovo	62.8	83.7	89.7	94.9	97.8	3.59
	4	TDtag	53.5	71.2	78.6	84.8	90.0	0.32
		PepNovo	51.1	71.7	79.3	85.8	91.4	3.64

Table 4.1: A comparison between the performance of our tag selection program and that of PepNovo at different tag lengths: (a) on 2657 experimental spectra without PTMs and (b) on 2620 experimental spectra with one artificially added PTM. Columns for $r = 1, 3, 5, 10, 25$ represent the percentages of spectra that have at least one correct tag in top $1, 3, 5, 10, 25$ tags generated by our program and PepNovo respectively; T is the average time in seconds used for generating sequence tags for one spectrum. TL is the tag length and AL is the algorithm used for tag selection.

4.3.4 Blind PTM Identification by Database Search

After candidate peptides are filtered out with the sequence tags, our point process based blind search model is applied to evaluate these candidate peptides for further peptide identification and PTM detection. The results on 2620 modified spectra are listed in Table 4.2. It can be seen that the sequence tags of lengths 3 and 4 are able to filter out more than 98.3% and 99.8% peptides in the database respectively, which consequently speeds up the calculations dramatically. In addition, with the reduced search space and enriched signals of correct peptides, the accuracies of PTM identification by database search are significantly improved with both sequence tags of lengths 3 and 4. For example, with the filtration of tag length 3, approximately 77% and 92% of spectra are identified correctly as top 1 and within top 5

respectively, a significant improvement compared to the corresponding accuracies of 60% and 81% without database filtration. Increasing the tag length from 3 to 4 can further speed up the PTM identification by approximately 8 times. However, a slight drop in the identification accuracy is observed in this case due to the relative lower sensitivity of tag generation for tag length 4.

Tag length	Top 1	Top 2	Top 3	Top 4	Top 5	Filtration ratio	Time(s)
3	76.69	86.01	89.29	90.70	91.62	0.0167	263
4	74.98	80.77	81.71	82.17	84.40	0.0014	34
Without filtration	60.38	72.33	76.64	79.16	81.17	–	3843

Table 4.2: The accuracy of identifying PTMs from the modified spectra with selected tags of lengths 3 and 4; the values at Top $i = 1, 2, 3, 4, 5$ represent the cumulative percentages of the search results capturing the original peptide sequences exactly in Top i; Filtration ratio is ratio of the survived candidate peptides after tag filtration. Time is the total time in seconds used for the point-process blind search model to identify correct peptides and PTMs for all the 2620 experimental spectra. The last row is the results without sequence tag filtration.

4.4 Summary

In this chapter, we develop a novel tree decomposition based algorithm that can efficiently generate highly accurate sequence tags and conduct efficient PTM identification by combining sequence tag generation and database search. The algorithm models a spectrum with its corresponding extended spectrum graph and can find all maximum weighted antisymmetric paths in the spectrum graph with tree width t in time $O(6^t n(n+m))$, where n and m are the number of vertices and the number of maximum weighted antisymmetric paths in the graph respectively. Sequence tags are then selected from all the maximum weighted antisymmetric paths. Our experiments show that this *ab initio* approach can achieve accuracy comparable to that of PepNovo in a significantly reduced amount of computation time. More importantly, the sequence tags can be used to filter a peptide database effectively and thus enable the application of more accurate and sophisticated algorithms for PTM identification. In particular, we have built a rigid framework to conduct peptide identification and

blind PTM search by combining high quality sequence tag generation and efficient database search. Experiments on spectra with PTMs show that this new approach can generate highly accurate sequence tags and significantly improve the accuracy of PTM identification by blind search.

Chapter 5

Generalized Subgraph Isomorphism

In this Chapter, we investigate a few new parameterizations of the generalized subgraph isomorphism problem. As an important application of these parameterizations in the bioinformatics research, we apply the results we obtained on the problem to the sequence-structure alignment problem in bioinformatics. This the sequence-structure alignment problem is a classic and important problem in bioinformatics and computational biology where structures play an instrumental role yet has not been satisfactorily solved.

It has been shown that optimally aligning a sequence to a generic structure with pseudoknots or two-body interactions is NP-hard [1, 50]. A structure profile (or template) generally consists of a few different structure units. For example, an RNA structure generally contains stems formed by contiguously stacked base pairs and loops formed by unpaired contiguous bases. Similarly, a protein structure contains cores and loops. Cores are α-helices or β-strands while loops consist of residues that do not interact with other residues in the structure. An sequence-structure alignment is evaluated with an alignment score, which is the sum of the alignment scores obtained on sequence segments aligned to all the structure units.

The sequence-structure alignment problem at the level of structure units can be formulated as a graph theoretical optimization problem. In particular, a structure unit in a profile can be represented with a graph vertex. Each pair of vertices whose structure units interact with each other are connected with a non-directed edge. In addition, neighboring structure units along the sequence backbone are connected with a directed edge. The resulting graph is called the *structure graph* for the structure profile.

5.1 Introduction

In previous chapters, we have considered a few problems that can be expressed with MSO logic sentences and obtained fixed parameter linear time algorithm for these problems. The existence of such algorithms is due to the generalization result provided by the Courcelle's theorem. In this chapter, we consider a problem that cannot be directly expressed with MSO logic but can still be solved with a fixed parameter linear time algorithm.

A graph $G_1 = (V_1, E_1)$ is *isomorphic* to another graph $G_2 = (V_2, E_2)$ if there exists a bijective mapping f between V_1 and V_2 such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. The Subgraph Isomorphism problem is to determine, for two given graphs G and H, whether there exists a subgraph in H that is isomorphic to G. H is often called the *host graph* and G is called the *guest graph*. A simple reduction from the Independent Set problem can show that this problem is NP-complete [35].

One possible parameterization of this problem is to take the size of the guest graph as a fixed parameter. However, the same reduction from the Independent Set shows that this problem is W[1]-hard. This problem is fixed-parameter tractable for host graphs from certain restricted graph classes. For example, when the host graph $H = (V, E)$ is a planar graph, this problem can be solved in time $2^{O(|G| \log_2 |G|)}|H|$ [30]. This result can be further extended to host graphs with bounded tree width. In [57], it is shown that, when the guest graph G is of bounded degree, this problem can be solved in time $O(|G|^{t+1}|H|)$ based on a tree decomposition of the host graph H with tree width t. Recently, it is shown in [39] that this problem can be solved in polynomial time when the guest graph has the property of log-bounded fragmentation and the host graph has bounded tree width.

It is not difficult to see that the Subgraph Isomorphism problem cannot be expressed with a MSO logic sentence unless the guest graph is of fixed size, since the isomorphic mapping needs to be described by binary relations unless the size of the guest graph is fixed. In this chapter, we consider a new parameter of this problem and provide a reduction that

can reduce it to a problem that can be expressed with a MSO logic sentence on a graph with small tree width.

In the following sections, parameterized algorithms and complexity results for a few different parameterizations of the Subgraph Isomorphism problem are presented in detail. In particular, one parameterized algorithm is in FPT and can be used to develop tools for efficient sequence-structure alignment in bioinformatics. Two case studies, including annotating noncoding RNA genes in genomes and predicting protein tertiary structures, are shown in this chapter.

5.2 Previous Work on Graphs of Small Tree Width

The Subgraph Isomorphism problem is fixed parameter tractable when the guest graph is of fixed size and the host graph is planar. In particular, we consider the following problem.

Problem 5.2.1 Planar Subgraph Isomorphism

Input: A guest graph $G = (V, E)$, which is connected and of fixed size, and a planar host graph $H = (M, L)$.

Output: "yes" if H contains a subgraph that is isomorphic to G; "no" otherwise.

The Planar Subgraph Isomorphism problem is NP-complete but fixed parameter tractable. There exists a $2^{O(|G| \log |G|)}|H|$ time algorithm that can decide whether G is isomorphic to a subgraph of H. The idea for this algorithm is based on the layered decomposition of a planar graph [8, 30]. The layers can be obtained in linear time with a breadth first search. It can be further shown that the subgraphs between certain layers have bounded tree width and this leads to a linear time parameterized algorithm for the Planar Subgraph Isomorphism problem.

Lemma 5.2.2 ([30]) *The* Subgraph Isomorphism *problem can be solved in time $O((t+3)^{|G|}|H|)$ based on a tree decomposition of the host graph with tree width t.*

Lemma 5.2.3 ([8, 30]) *Given a rooted spanning tree T for a planar graph H, a tree decomposition of tree width at most $3l$ can be constructed in polynomial time, where l is the maximum distance between a leave and the root.*

This fact can be seen from the tree width of a planar triangulation H' of H. i.e., H' is on the same vertex set as that of H and H is a subgraph of H' and each face in H' is a triangle. It is easy to see that a tree decomposition of H' is also that for H. In addition, T is also a spanning tree for H'. A tree decomposition for H' can in fact be constructed from T. Since H' is a planar graph, based on a plane embedding of H', we assume the leaves of T are $v_1, v_2, \cdots, v_m$ and the path P_i from the root r to v_i is to the left of that of P_{i+1} in the plane. Now, start with P_1, we identify all the vertices on the P_i $(1 \leq i \leq m)$ and all vertices that are to the left of P_1 in the plane and form a triangle surface with one of its edges. We group such vertices in tree bag X_i. For the last step, we put all vertices in P_m to every X_i $(1 \leq i \leq m)$. The number of such vertices in X_i is bounded by $3l + 1$, because P_i contains at most l edges. Thus at most $2l$ additional vertices are included in X_i. Now we connect $X_1, X_2, \cdots, X_m$ into a path. The resulting tree decomposition is a tree decomposition for H and its tree width is bounded by $3l$.

Theorem 5.2.4 ([30]) *The* Planar Subgraph Isomorphism *can be solved in time* $2^{O(|G| \log |G|)}|H|$.

The results developed for the Planar Subgraph Isomorphism problem requires the guest graph to be of fixed size. However, this is not always true in practice. The Subgraph Isomorphism problem under a few other constraints has been studied. For example, we consider the following problem.

Problem 5.2.5 Bounded Subgraph Isomorphism

Input: A guest graph $G = (V, E)$ whose degree is bounded by Δ, a tree decomposition of tree width t for the host graph $H = (L, M)$.

Output: "yes" if G is isomorphic to a subgraph of H. "no" otherwise.

The Bounded Subgraph Isomorphism problem can be solved in time $O(|G|^{t+1}(t+1)!2^t3^{\Delta t}|H|)$ [57], which is in P when both t and Δ are fixed constants. This result thus does not require the guest graph to be of fixed size. We provide a sketch of this algorithm in this section. Basically, this algorithm is based on the following fact.

Lemma 5.2.6 *Given two graphs $G = (V, E)$ and $H = (L, M)$ that satisfy all of the followings:*

1. *H contains a subgraph $G' \subseteq H$ that is isomorphic to G;*
2. *the isomorphic mapping is f; if $u \in V$ is mapped to $v \in L$ by f, we define $f^{-1}(v) = \{u\}$; for a vertex set $V_s \subseteq V$, we define $f^{-1}(V_s) = \bigcup_{v \in V_s} f^{-1}(v)$;*
3. *$T = (X, F)$ is a tree decomposition for H.*

For a given tree node $X_i \in X$, if $X_i \cap G' \neq \phi$, and $G - f^{-1}(X_i \cap G')$ contains connected components $C_1, C_2, \cdots, C_m$, then there does not exist vertices $u, v \in G$ and tree nodes X_j and X_k satisfying all of the followings:

1. *$f(u) \in X_j$ and $f(v) \in X_k$;*
2. *$u, v \in C_i$ $(1 \leq i \leq m)$;*
3. *X_i is in the path that connects X_j and X_k in T.*

This is in fact a result implied in the definition of tree decomposition. To show this, we assume that there exists such u, v and X_j, X_k. Because u and v are connected in graph $G - f^{-1}(X_i \cap G')$, there exists a path P that connects u and v in graph G that avoids vertices in $f^{-1}(X_i \cap G')$. We now consider the vertices in $f(P)$. Because G and G' are isomorphic, there exists a path P' in G' (also H) that connects $f(u)$ and $f(v)$ and the vertices in the path are from $f(P)$.

However, it is not difficult to find that $f(u) \notin X_i$ and $f(v) \notin X_i$, since otherwise u or v is in $f^{-1}(X_i \cap G')$. In addition, for any vertex $s \in P$, $f(s) \notin X_i$. In graph H, there exists

a path P' that connects X_j and X_k and avoids X_i. This is impossible, since X_i must be a separator for $f(u)$ and $f(v)$ in H.

Lemma 5.2.6 states that for any isomorphic mapping between G and a subgraph of H, if after we remove from G the vertices that are mapped to vertices in X_i, the remaining part of G consists of a few connected components, then the mapped images for vertices in the same component must be completely in one of the connected subtrees in $T - X_i$. This property is important for us to obtain an efficient tree decomposition based algorithm to solve the Bounded Subgraph Isomorphism problem.

Theorem 5.2.7 ([57]) *Based on a tree decomposition of tree width t for the host graph, the* Bounded Subgraph Isomorphism *problem can be solved in time $O(|G|^{t+1}(t+1)!2^t3^{\Delta t}|H|)$.*

The algorithm described in theorem 5.2 can be sketched as follows. Without loss of generality, we assume that T is a binary tree and we also maintain a dynamic programming table in each tree node. Each table entry in any tree node X_i maintains a possible mapping between Y_i in the guest graph and a subset Z_i of vertices in the tree node. This entry, however, also maintains a few additional fields to store the subtrees in $T - X_i$ that each component in $G - Y_i$ is mapped to. A valid bit V_a is also associated with each table entry to indicate whether the partial mapping this entry specifies can be valid or not. Based on this table structure, we can use a bottom-up dynamic programming approach similar to the one proposed in [6]. In particular, an exhaustive search of all table entries is performed on a leaf node. For an internal node, we need to query its child nodes to determine the validity of each of its table entries. Due to lemma 5.2.6, for a given entry, we are able to store the mapping status of all other vertices that are not mapped to vertices in X_i in a scale of "components" instead of "vertices". This thus enables us to check the consistency between different table entries and guarantees the correctness of this algorithm. The maximum number of entries in a table can be estimated based on the following simple facts.

1. The number of possible Y_i's is bounded by $\sum_{l=0}^{t+1} \binom{|G|}{l}$;

2. the number of possible Z_i's is bounded by $\sum_{l=0}^{t+1} \binom{t+1}{l}$;

3. the number of possible mappings between given Y_i and Z_i is $|Z_i|!$;

4. the number of possible connected components for $G - Y_i$ is bounded by Δt;

Because the tree is a binary tree, for each node X_i, $T - X_i$ can have up to 3 connected subtrees and the number of possible ways that connected components in $G - Y_i$ can be assigned to these subtrees is thus bounded by $3^{\Delta t}$. The total number of table entries in X_i is thus bounded by:

$$\sum_{l=0}^{t+1} \binom{|G|}{l} \sum_{l=0}^{t+1} \binom{t+1}{l} (t+1)! 3^{\Delta t} \tag{5.1}$$

Also we have :

$$\sum_{l=0}^{t+1} \binom{|G|}{l} \leq (t+1)|G|^{t+1} \tag{5.2}$$

and

$$\sum_{l=0}^{t+1} \binom{t+1}{l} = 2^{t+1} \tag{5.3}$$

The time complexity of this algorithm is bounded by $O(|G|^{t+1}(t+1)!2^{t+1}3^{\Delta t}|H|)$.

5.3 Map Width for Subgraph Isomorphism

Definition 5.3.1 *Given two graphs $G = (V, E)$ and $H = (S, M)$ and a mapping f from V to 2^S, the map width of f is defined as $M = \max_{u \in V} |f(u)|$.*

Map width is defined for a mapping between vertices in the guest graph and vertex subsets in the host graph. In particular, the map width for such a mapping is the maximum cardinality over all the vertex subsets that are mapped to a vertex in the guest graph. We obtain a new parameterization of the Subgraph Isomorphism problem if we restrict the map width of the isomorphism mapping to be a fixed small parameter k. We thus consider the following parameterized subgraph isomorphism problem.

Problem 5.3.2 Parameterized Subgraph Isomorphism 1 *(PSI1):*

Input: graph G, graph H, mapping f from G to H of map width k;
Output: "yes" if and only if there is isomorphism under mapping f between G and a subgraph of H.

However, the following theorem states that PSI1 problem is unlikely to be in FPT.

Theorem 5.3.3 *PSI1 remains NP-complete when $k \geq 3$.*

Proof: It is sufficient to show that PSI1 is NP-complete when $k = 3$. We can reduce the 3SAT problem to PSI1 when $k = 3$. In particular, for the boolean formula $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, in a given 3SAT instance, we consider the literals contained in clause c_i. We assume $c_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$. For each clause, we generate three graph vertices to represent the three literals in the clause. We now connect two literals l_1, l_2 with an edge if $l_1 \neq \neg l_2$. The resulting graph is the host graph H. Guest graph G is simply a clique with m vertices. Each vertex in G represents a clause. We now have an instance of the PSI1 problem with $k = 3$ if each vertex in G can only be mapped to one of the three possible vertices that represent its literals in H. It is clear that F is satisfiable if and only if H contains a subgraph isomorphic to G. □

A different parameterization is to take the tree width t of the guest graph as a parameter. We thus have the following parameterized subgraph isomorphism problem.

Problem 5.3.4 Parameterized Subgraph Isomorphism 2 *(PSI2):*
Input: Graph H, graph G, a tree decomposition of graph G with tree width t;
Output: "yes" if and only if there is a subgraph in H that is isomorphic to G.

The following theorem states that PSI2 is W[1]-hard.

Theorem 5.3.5 ([18]) *The PSI2 is W[1]-hard.*

Proof: We reduce from the Clique problem, which asks whether a given graph contains a clique of size k. We generate a guest graph which is a clique of size k and use the graph in the Clique problem as the host graph. The tree width of the guest graph is $k - 1$. The theorem has been proved. □

Another possible parameterization is to take both the width of isomorphic mapping and the tree width of the guest graph as parameters. The parameterized complexity of this problem is still open. However, the Induced Subgraph Isomorphism problem remains W[1]-hard under this parameterization. We thus believe that this problem is unlikely to be in FPT.

5.4 Parameterized Algorithms for Subgraph Isomorphism

The Parameterized Subgraph Isomorphism problem is fixed-parameter tractable if vertices in guest graph G is ordered and vertices in H are partially ordered. And the vertices selected in the subgraph must follow the same order as in the guest graph.

Theorem 5.4.1 ([86, 85]) *For an isomorphic mapping with bounded map width k, based on a tree decomposition of tree width t for G, PSI can be solved in time $O(k^t|G|)$, if vertices in the guest graph G are ordered and those in the host graph H are partially order. Vertices in the subgraph that are selected must be ordered and follow the same order as in the guest graph.*

Proof: Based on the tree decomposition of graph G, a dynamic programming algorithm that follows the framework developed in [6] can be applied to decide whether such a subgraph exists or not. This algorithm maintains a dynamic programming table for each tree node in the tree decomposition of G. Starting with the leave nodes, it fills the table for each tree node. For a given tree node, the algorithm exhaustively enumerates all possible combinations of the candidates for its vertices. The algorithm at the same time checks whether corresponding edges in the guest graph exist between the selected candidates or not. For an internal tree node, the algorithm only needs to query the tables of its child nodes that store the partial solutions on subgraphs induced by the vertices in the subtrees rooted at these child nodes. It is not difficult to see that this algorithm can check whether their selected candidates follow the same ordering or not. The final conclusion for the problem can be obtained by

querying the table in the root of the tree decomposition. A up-bottom trace-back procedure can be applied to find the subgraph that is isomorphic to the guest graph. The algorithm is correct since all vertices in G are ordered and those in H are partially ordered. Because two contiguous vertices on the backbone of G are connected and there exists a tree node that contains both vertices, we are able to guarantee that the candidates of the two vertices follow the same order. Due to the transitivity of ordering, it is not difficult to see that this algorithm is able to guarantee the validity of the selected mapping. The computation time needed is $O(k^t|G|)$ since for each tree node, the number of combinations of candidates can be up to $O(k^t)$. The theorem has been proved. □

In practice, we often need to consider the Maximum Valued Subgraph Isomorphism problem, where edges and vertices in the host graph are associated with certain values and the objective is to find the maximum valued subgraph that is isomorphic to the guest graph. We can slightly modify the algorithm in the proof of the theorem 5.4.1 and obtain the following corollary.

Corollary 5.4.2 ([85, 86]) *For an isomorphic mapping with bounded map width k and a valued host graph H whose vertices and edges are associated with values, based on a tree decomposition of tree width t for G, the maximum valued subgraph in H that is isomorphic to G can be solved in time $O(k^t|G|)$, if vertices in the guest graph G are ordered and those in the host graph H are partially ordered. Vertices in the subgraph that are selected must be ordered and follow the same order as in the guest graph.*

5.5 Reduction to Smaller Treewidth

The Subgraph Isomorphism problem cannot be expressed with a MSO logic sentence since a binary relation is needed for the isomorphic mapping unless the host graph H is of fixed size. However, the Parameterized Subgraph Isomorphism problems we have studied take the tree width of the host graph and the map width of the isomorphic mapping

as the parameters. These problems thus cannot be expressed with a MSO logic sentence. However, if we consider the following $(t+1,k)$-Partite Clique problem.

Problem 5.5.1 $(t+1,k)$-Partite Clique

Input: A $(t+1,k)$-partite graph $G=(V_1\cup V_2\cup\cdots\cup V_{t+1},E)$, where $|V_1|=|V_2|=\cdots=|V_{t+1}|=k$.

Output: "yes" if G contains a k-clique; "no" otherwise.

This problem can be solved with an exhaustive enumeration in time $O(k^{t+1})$. However, the following theorem states that there exists a polynomial time reduction that can reduce the $(t+1,k)$-Partite Clique problem to finding the maximum weighted independent set on a graph with tree width bounded by $t\log_2 k$, a graph with tree width smaller than that of G.

Theorem 5.5.2 ([18]) *Let $G=(V,E)$ be t-partite graph, where $V=\cup_{0\le i\le t}U_i$, with each component U_i containing at most k vertices. Then $(t+1,k)$-Partite Clique problem on G can be reduced to the problem of deciding a maximum weighted independent set on a graph of tree width $(t+1)\lceil\log k\rceil$.*

Proof: Without loss of generality, we assume that $|U_i|=k$, $i=0,1,\ldots,t$. Let $U_i=\{u_0^i,u_1^i,\ldots,u_{k-1}^i\}$. We construct a new weighted graph G' as follows: $G'=(V',E')$, where $V'=Z\cup(\cup_{i=1}^{t}W_i)$. Each set

$$W_i=\{g_1^i,\bar{g}_1^i,\ldots,g_{\lceil\log k\rceil}^i,\bar{g}_{\lceil\log k\rceil}^i\}$$

contains $2\lceil\log k\rceil$ vertices. Set Z contains at most $k^2\frac{t(t+1)}{2}$ elements. That is, for every pair U_i,U_j, $i<j$ in the original graph G and every pair of vertices u_h^i,u_l^j, where $u_h^i\in U_i$ and $u_l^j\in U_j$, there is a unique vertex $z_{h,l}^{i,j}\in Z$ if $(u_h^i,u_l^j)\in E(G)$.

Now we consider the edge set E' in graph G'. For each set W_i, $i=0,1,\ldots,t$, E' contains an edge $(g_h^i,\bar{g}_h^i)$ for every $h=1,\ldots,\lceil\log k\rceil$. In addition, E' contains edges connecting vertices in W_i, $i=1,\ldots,t$ to vertices in Z. In particular, for edge $(u_h^i,u_l^j)\in E$, where $u_h^i\in U_i$ and

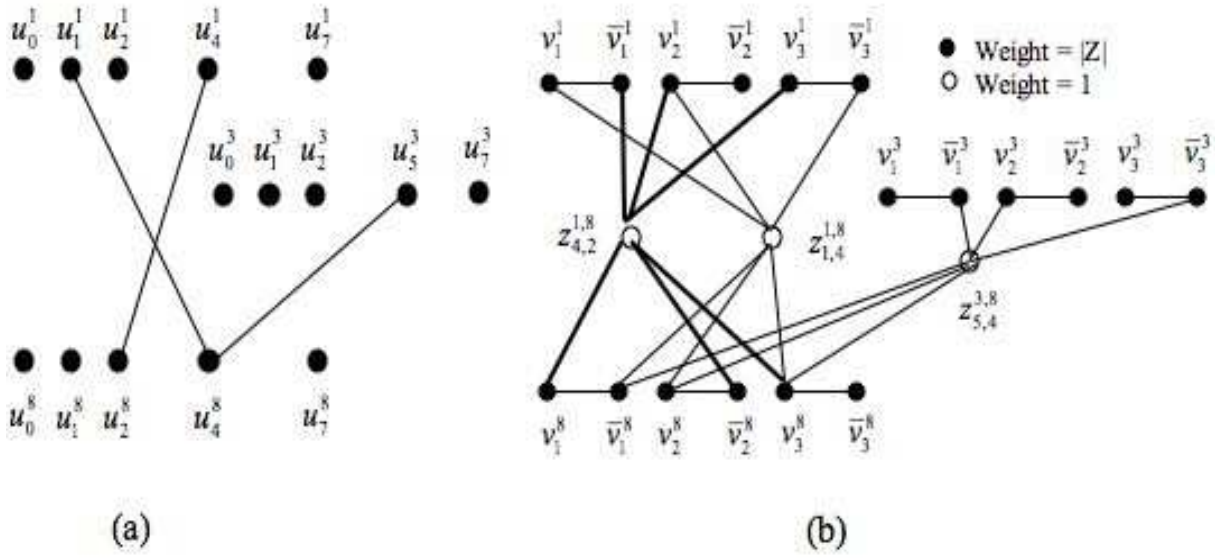

Figure 5.1: An example to illustrate the reduction to the maximum weighted independent set problem. (a) The $(t+1)$-partite graph G, where only components 1, 3 and 8 and three example edges $(u_1^1, u_4^8), (u_4^1, u_2^8), (u_5^3, u_4^8)$ are shown, $k = 8$. (b) The constructed graph G', $\log k = 3$, where three vertices $z_{4,2}^{1,8}, z_{8,4}^{1,1}, z_{8,4}^{3,5}$ correspond to the three edges in (a). Since $\beta(4)$, the binary representation of 4, is [100], $\beta(2) = [010]$, vertex $z_{4,2}^{1,8}$ connects to vertices $\bar{v}_1^1, v_2^1, v_3^1$ and $v_1^8, \bar{v}_2^8, v_3^8$ (bold line edges).

$u_l^j \in U_j$, let $\beta(h) = (a_1 a_2 \ldots a_{\lceil \log k \rceil})$ be the canonical binary representation of number h, where $a_s \in \{0, 1\}$, $s = 1, \ldots, \lceil \log k \rceil$. Also let $\beta(l) = (b_1 b_2 \ldots b_{\lceil \log k \rceil})$ be the canonical binary representation of number l, where $b_s \in \{0, 1\}$, $s = 1, \ldots, \lceil \log k \rceil$. Vertex $z_{h,l}^{i,j}$ is connected to vertex g_s^i if $a_s = 0$ (and connected to vertex $\bar{g}_s^i$ if $a_s = 1$), for all $s = 1, \ldots, \lceil \log k \rceil$. Also vertex $z_{h,l}^{i,j}$ is connected to vertex g_s^j if $b_s = 0$ (and connected to vertex $\bar{g}_s^j$ if $b_s = 1$), for all $s = 1, \ldots, \lceil \log k \rceil$. The weight of each vertex in W_i is assigned $|Z|$. But the weight of each vertex in set Z is 1. Figure 5.1 shows an example to illustrate the construction of G'.

We now show that G has a clique of size $(t+1)$ if and only if G' has an independent set of weight $|Z|(t+1)\lceil \log k \rceil + t(t+1)/2$.

Now suppose that there is a clique C in G. Without loss of generality, assume the vertices in C are arranged such that $C = \{u^0_{h_0}, u^1_{h_1}, \ldots, u^t_{h_t}\}$, because vertices in every U_i are independent to each other. Then in the constructed graph G', there is an independent set I_C of weight $|Z|(t+1)\lceil \log k \rceil + t(t+1)/2$ containing $\lceil \log k \rceil$ vertices from every set W_i, $i = 0, 1, \ldots, t$, and exactly $t(t+1)/2$ vertices from Z. In particular, corresponding to the clique C, the independent set I_C contains vertex g^i_s if the sth bit of the binary presentation of h_i is 1 and contains vertex $\bar{g}^i_s$ if it is 0, for all $i = 0, \ldots, t$ and $s = 1, \ldots, \lceil \log k \rceil$. In addition, I_C contains the following vertices also: $z^{i,j}_{h_i,h_j}$, for $i < j$ and $i, j = 1, \ldots, r$. It is not hard to verify that I_C is an independent set with the desired weight.

Conversely, if there is a maximum weight independent set I in G' with weight $|Z|(t+1)\lceil \log k \rceil + t(t+1)/2$, it should contain exactly $\lceil \log k \rceil$ vertices from every set W_i that is involved, assume $i = 0, \ldots, t$. Moreover, it is not hard to see that the subset of involved vertices from W_i is a "binary encoding" of some number, say h_i. It is not hard to see either that vertex $z^{i,j}_{h_i,h_j}$ is independent to the involved vertices from both W_i and W_j and so it belongs to I. Including any other vertex from Z into the independent set I would cause conflicts with some of the vertices from W_i's that are already in I. Removing any of these vertices would result in a decrease of total weight by at least $|Z| - 1$, an amount that cannot be compensated by the weight of the additional vertex from Z. Moreover, one can check easily that the independent set I corresponds to a clique C_I $C_I = \{u^0_{h_0}, u^1_{h_1}, \ldots, u^t_{h_t}\}$.

The constructed graph G' is actually a bipartite graph with W_i's as one vertex set and Z the other. So the graph can be decomposed as a tree of at most $1 + k^2 \frac{t(t+1)}{2}$ nodes. There are at most $(t+1)\lceil \log k \rceil + 1$ vertices in each bag. That is $(\cup^t_{i=1} W_i) \cup \{z^{i,j}_{h,l}\}$ for every $i, j, (i < j), h$, and l, $(h < l)$. Therefore, the tree decomposition has tree width $(t+1)\lceil \log k \rceil$. □

This upper bound on the tree width on G' allows us to derive an alternative algorithm based on Courcelle's theorem. That is, PSI is solvable in time $O(2^{(t+1)\lceil \log k \rceil} n) = O(k^{t+1} n)$, the same complexity as claimed in theorem 5.4.1. The difference is that this new problem can be expressed with a MSO logic sentence.

5.6 Sequence-Structure Alignment in Bioinformatics

The sequence that is to be aligned to the structure profile can be preprocessed and the possible sequence segments that can be aligned to each structure unit can be determined. These sequence segments are *candidates* for the given structure unit. Similarly, each candidate can be represented with a graph vertex and vertices for candidates of interacting structure units can be connected to form a *sequence graph*. In this dissertation, we use G to denote a structure graph and H for a sequence graph.

It is clear that a possible sequence-structure alignment corresponds to a subgraph contained in the sequence graph H and isomorphic to the structure graph G. Each candidate can be valued based on the alignment score between its sequence segment and the profile of the structure unit. In addition, each edge in H can also be associated with a value. In particular, this value can be computed using the energy or statistical functions modeling the interactions between structure units. The optimal sequence-structure alignment thus corresponds to a maximum (or minimum) valued subgraph in the sequence graph that is isomorphic to the structure graph. Since the sequence backbone provides an ordering of structure units and sequence segments, the parameterized algorithm developed in theorem 5.4.1 and corollary 5.4.2 can possibly be used to develop efficient computational tools for the sequence-structure alignment.

5.6.1 Annotating Noncoding RNAs on Genomes

An RNA structure consists of stacked base pairs, these stacked base pairs form stems. An RNA structure can be modeled with a conformational graph, in which each vertex represents a base region of a stem and each edge connects two base regions if they form a stem or they constitute the two ends of a loop. Figure 5.2 (a)(b) show a secondary structures and its corresponding structure graph. Figure 5.2 (c)(d) provide an example on the construction of the sequence graph for a sequence that is to be aligned to the structure. Based on this

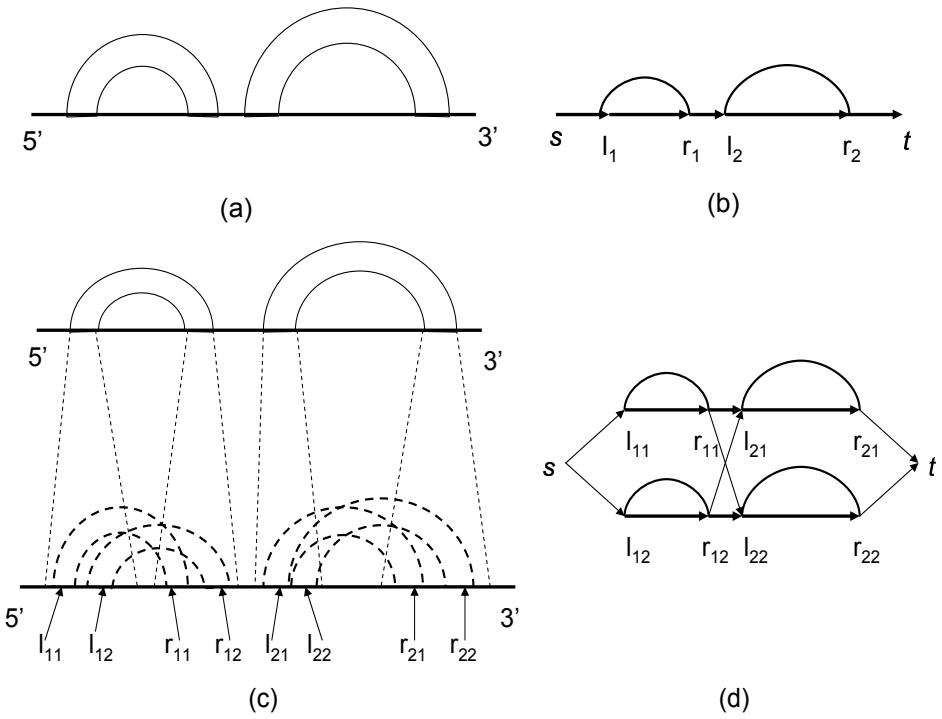

Figure 5.2: (a) An RNA structure that contains parallel stems. (b) The corresponding structure graph. (c) An RNA secondary structure (top), and the mapped regions and images of the stems on the target sequence (bottom). (d) The sequence graph constructed from the candidates of the stems. (l_{11}, r_{11}) and (l_{12}, r_{12}) are the candidate stems of the first stem, and (l_{21}, r_{21}) and (l_{22}, r_{22}) are those of the second stem.

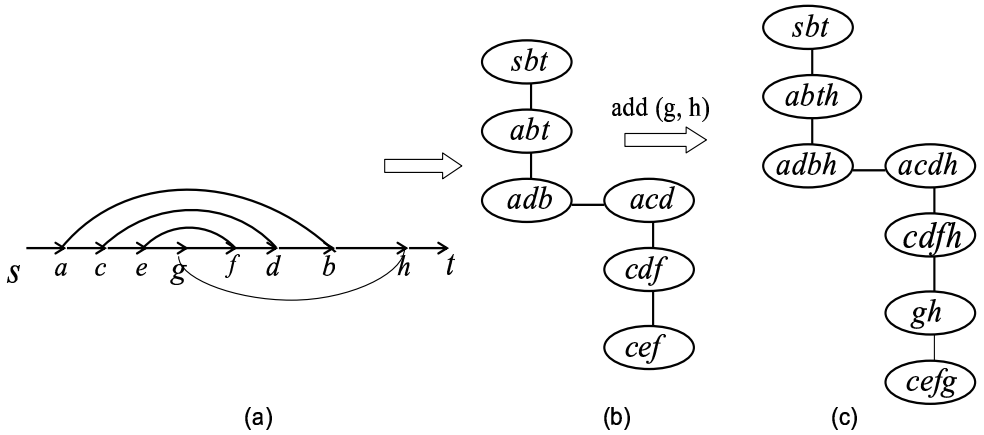

Figure 5.3: (a) An RNA structure that contains both nested and pseudoknot stems. (b) A tree decomposition of its corresponding structure graph without the crossing stem (g, h). (c) A tree decomposition of its corresponding structure graph added back the crossing stem (g, h).

representation of sequence and structure, the optimal sequence-structure alignment corresponds to a generalized maximum valued subgraph isomorphism problem in which the host graph is the conformational graph, usually of a naturally small tree width t. From figure 5.3, it's not difficult to see that the tree width of the conformational graph for a pseudoknot free structure is bounded by 2 since it is a outer-planar graph [11]. For a conformational graph that contain pseudoknots, the tree width is in general slightly larger than 2. Using the dynamic programming algorithm developed in theorem 5.4.1, an optimal alignment can be found in time $O(k^t N^2)$ for a given integer parameter k. The value of k can be effectively determined by a statistical cutoff and is also small in nature. The small tree widths of the conformational graphs in this problem guarantee the computational efficiency of searching. Compared with the CYK algorithm developed based on the Covariance Models (CMs) [28], this new algorithm is significantly faster.

This new algorithm was tested on several ncRNA families to test the accuracy and efficiency of the searching algorithm. Experiments have shown that using a significantly reduced amount of computation time, the searching algorithm based on this new model can achieve the same accuracy as the CM based searching does. Specifically, on average, the algorithm is about 24 and 50 times faster than CM based methods on searching for pseudoknot free sequences that contain around 90 and 150 nucleotides respectively. Our experiments also demonstrated an even more significant advantage of the algorithm over the CM based search in computation time when the profiled RNAs contain pseudoknots. As a test of the model on real genomes, we used the algorithm to search for the tmRNA gene in two bacterial genomes and the telomerase RNA gene in two yeast genomes. Both the tmRNA and the telomerase genes were detected in both genomes with high accuracy in days, a task that would have needed months of computation time if a CM based searching model has been used.

5.6.2 Predicting Protein Tertiary Structures

Protein threading is an important method for predicting the tertiary structure of a protein sequence. Specifically, protein threading aligns a protein sequence to all available structure templates in a database and the tertiary structure is predicted based on the alignment scores. The core part of a protein threading is thus the sequence-structure alignment. However, it has been shown that this problem is NP-hard for a generic structure profile that includes two body interactions between amino acids [50]. Similar to the sequence-structure alignment for noncoding RNAs, we consider this alignment problem at structure unit level. The structure units in a structure template are *cores*. Cores usually represent α-helices and β-strands. To simplify the problem, gaps are not allowed to appear in cores in a valid sequence-structure alignment. We model a structure template with a structure graph where the interacting cores are connected with nondirected edges, and cores neighboring in the backbone are connected with directed edges. A protein sequence can be preprocessed with the profiles of each structure unit and an image graph can be similarly constructed. Figure 5.4 (a)(b) show

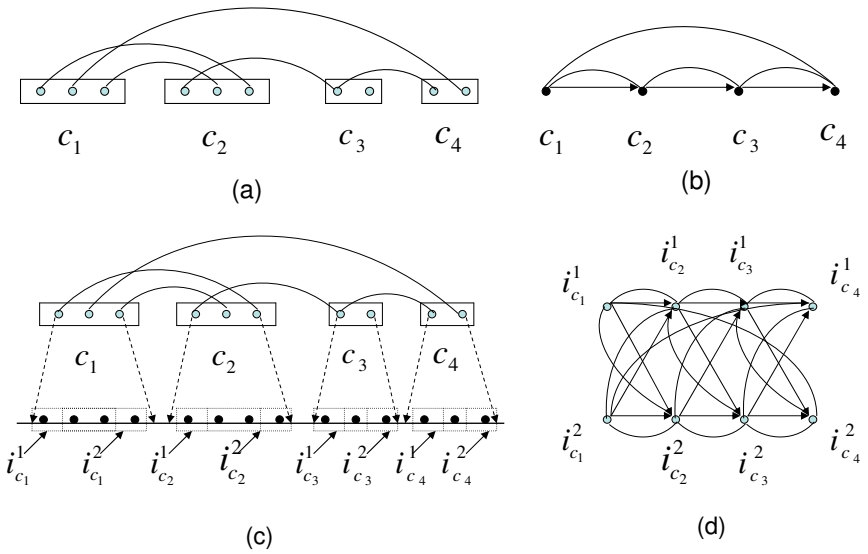

Figure 5.4: (a) A protein structure template that contains both cores and loops. (b) The corresponding structure graph. (c) The mapped region and candidates for each core in the structure template (bottom). The dashed lines specify the possible mappings between cores and their images. Core c_i has two possible images $i^1{}_{c_i}$ and $i^2{}_{c_i}$. (d) The sequence graph formed by the candidates of cores in the template.

a protein structure template and its corresponding structure graph, while Figure 5.4 (c)(d) show the construction of sequence graph from the candidates selected for each structure unit.

The optimal alignment between a sequence and a structure template corresponds to finding the minimum valued subgraph isomorphism between the conformational graph and the image graph. Based on graph tree decomposition, alignment between a sequence of M amino acids and a structure template of size N can be performed in time $O(MN+k^t n)$, where n is the number of cores in the structure template and t is the tree width of the conformational graph. Parameter k, usually small, represents an upper bound for the possible number of locations in the sequence that can be aligned to a given core. In particular, our experiments show that among 3890 protein tertiary structure templates compiled using PISCES [93], only 0.8% of them have treewidth $t > 10$ and 92% have $t < 6$, when using 7.5 Å C_β-C_β

distance cut-off for defining two-body interactions. The naturally small tree width t and small parameter k allows the alignment algorithm to run efficiently.

The technique of tree decomposition has been used in protein side-chain packing when back bone is known [99]. A graph theoretical algorithm proposed in [100] formulates the protein threading problem as a graph labeling problem, which can be solved with graph tree decomposition. But the algorithm was not implemented and tested for protein threading. Our work formulates the alignment into subgraph isomorphism. The introduction of parameter k makes it possible to achieve efficiency in threading.

For testing and evaluating the efficiency and accuracy of the algorithm, we implemented the algorithm in program PROTTD and compared its performance with that of PROSPECT II [49] and RAPTOR. We used protein pairs in the Dali data set [43] to test the alignment accuracy of PROTTD. The alignment accuracy was evaluated by comparing the obtained alignments with those provided by FAST [106]. Our experiments showed that, on average, PROTTD is about 50 times faster than PROSPECT II to obtain better or same alignment accuracy. In addition, we tested the algorithm on the lindahl data set [82] for fold recognition and compared the accuracy with that of RAPTOR at all similarity levels. Our testing results showed that PROTTD achieved significantly improved fold recognition accuracy on superfamily and fold levels.

5.7 Summary

The Subgraph Isomorphism problem cannot be expressed with a MSO logic sentence unless the guest graph is fixed. Our study on the relationship between this problem and MSO logic has extended the Courcelle's theorem based on the development of a parameterized algorithm. The algorithm solves the problem by reducing the Maximum Weighted Independent Set problem on a graph with a smaller tree width, which can be expressed with a MSO logic sentence.

The Subgraph Isomorphism problem is an NP-complete problem and has been intensively studied during the past decade. Tree decomposition provides an important algorithmic tool for developing parameterized algorithms for this problem and a few important results have been obtained, including a linear time algorithm for Planar Subgraph Isomorphism when the guest graph is of fixed size [30] and a polynomial time algorithm [57] when the guest graph is of bounded degree and the tree width of the host graph is bounded by a constant.

In this chapter, we study a different parameterization of the Subgraph Isomorphism problem. We consider the map width for the isomorphic mapping between the guest graph and the host graph. Based on this new parameter and the tree width of the guest graph, the parameterized complexities of a few different parameterizations of this problem are investigated. In particular, the Subgraph Isomorphism problem remains intractable while only one of the parameters, the map width and tree width of the host graph, is fixed. In addition, a slightly different problem, the Induced Subgraph Isomorphism problem, remains W[1]-hard even both of them are fixed parameters. However, our study also show that under certain constraints, the Subgraph Isomorphism problem is fixed parameter tractable when both the map width and tree width of the guest graph are fixed parameters.

This parameterized algorithm can be used to solve the sequence-structure problem in bioinformatics. In particular, the sequence-structure alignment problem can be formulated as the Maximum Valued Subgraph Isomorphism problem and the corresponding graphs satisfy the constraints needed to guarantee the correctness of the algorithm. In addition, the tree widths of the guest graphs in the sequence-structure alignment problem are in general small (less than 6 for majority of problem instances). This algorithm can thus be used to develop a new computational tool for sequence-structure alignment. Case studies performed on both ncRNA gene annotation and protein tertiary structure prediction have demonstrated the advantage of this algorithm over others in both computational efficiency and accuracy.

Chapter 6

Conclusions and Future Work

NP-hard optimization problems constitute one of the important aspects of bioinformatics research. Finding efficient but optimal solutions for these problems is thus of fundamental and important value. In this dissertation, we provide an original perspective on the effort to develop optimal algorithms for these problems. We notice that, due to the biological or biochemical nature for these problems, many optimization problems in bioinformatics contain inherently small parameters. These parameters provide an avenue to study these problems from the views of parameterized computation. We identified a structure parameter that is inherently small for several bioinformatics problems. Based on this parameter, we developed a new theoretical framework that can be used to formulate these problems on graphs and efficient optimal algorithms for these problems can be developed based on this framework.

The structure parameter we have identified is graph tree width, which is a concept originally developed in graph minor theory [71]. Tree width is a parameter associated with tree decomposition, which develops a new topological view on the graphs. Tree decomposition also has algorithmic implications and has been used to develop parameterized algorithms for many NP-hard graph optimization problems. Tree width is in general an important parameter in these algorithms.

A generic result on the problems that can be solved in linear time on graphs with bounded tree width is Courcelle's theorem, which states that any problem that can be expressed with MSO logic can be solved in linear time on graphs with bounded tree width. With this in mind, we find that many NP-hard optimization problems in bioinformatics can be formulated on graphs. The tree width of these graphs are inherently small due to the biochemical or

biological property of these problems. In addition, the objectives to be optimized can also be expressed in MSO logic. These algorithms can thus be solved efficiently with the dynamic programming method developed in Courcelle's theorem.

We have designed and implemented the parameterized algorithms for a few NP-hard optimizations in bioinformatics, including the sequence-structure alignment for ncRNA and proteins, protein identification from MS/MS spectrum and the identification of PTMs with peptide sequence tag selection. Experiments performed on biological data have demonstrated the advantage of these algorithms over other methods in both accuracy and computational efficiency.

This dissertation, however, only serves as a starting point for the development of parameterized algorithms and complexity results for problems in bioinformatics research. With the emergence of more and more new problems, new techniques and methods are definitely needed and will be developed. These techniques will enhance the understanding of both the parameterized computation and possibly the biological mechanism behind these problems. In the following sections, we provide a vision of possible future work to extend the research in this dissertation.

6.1 Improving the Constant Factor

Parameterized computation provides efficient solutions for some NP-hard problems when the parameters in these problems are small. In addition, for a parameterized algorithm that can solve an NP-hard problem $\langle k, n\rangle$ in time $O(f(k)n^c)$, the constant factor $f(k)$ is often an exponential function and its value determines the computation time needed by the algorithm in practice. Improvements on the constant factors are thus important for reducing the computation time needed to solve these problems. For example, the first parameterized algorithm developed to solve the Vertex Cover problem uses a search tree technique and the computation time needed by this algorithm is bounded by $O(2^k|V|)$. Techniques developed later have led to parameterized algorithms that only need time $O(k|V| + 2^k)$. In

[20], a new $O(k|V| + 1.2852^k)$ time algorithm is developed for this problem and it can be practically used to solve problem instances where the parameter k is not larger than 50.

The algorithms we have developed and implemented in this dissertation are based on graph tree decomposition and the parameter in these algorithms is the tree width of the underlying graph. For a structure parameter like tree width, techniques and methods that can reduce the constant factor in the corresponding parameterized algorithms have not been available. Developing tree decomposition based algorithms with reduced constant factors is thus important to further improve the practical value of parameterized computation in solving bioinformatics problems. In addition, methods and techniques that can lead to such algorithms are of interest for algorithmic research itself.

6.2 Graphs with Large Tree Width

For all the problems we have studied in this dissertation, we assume the tree width of the underlying graph is a small integer. Due to the biochemical properties associated with these problems, the assumption is correct for most of the problem instances in practice. However, this assumption may not be the case for some of the bioinformatics problems that can be reduced to a graph optimization problem.

For example, the goal of the RNA folding problem is to find the secondary structure with the minimum energy for a given RNA sequence. The problem is NP-hard when a sequence may fold into a structure that contains pseudoknots [1]. All stems that are energetically stable can be found with a dynamic programming algorithm in polynomial time and based on these stems, this problem can be reduced to a graph optimization problem. More specifically, a graph vertex can be used to represent a stem and a graph vertex can be associated with a weight which is the inverse of the thermal dynamic energy of the corresponding stem. Two vertices are connected with a graph edge if the corresponding stems conflict on the sequence. A graph can thus be constructed to represent the sequence and the secondary structure with the minimum energy is the independent set with the maximum weight in such a graph.

This problem can be solved efficiently if the graph constructed from stems is of small tree width. Unfortunately, our study has suggested that this is not the case and the tree width of the graph may significantly increase when the length of the sequence increases. Tree decomposition based dynamic programming thus cannot provide a practically efficient solution for this problem.

Specific problem properties can sometimes help resolve the difficulty that arises from a graph with a large tree width. For example, in the RNA folding problem, although the tree widths of the underlying graphs are in general not small, vertices in a tree bag can often be partitioned into cliques of large size. This partition has been utilized to reduce the computation time needed for enumerating all possible combinations of selected vertices since two vertices in the same clique cannot be selected in the same independent set. This property has led to an algorithm that can be practically used for structure prediction.

For future research, it is important to develop reductions that can formulate problems on graphs with tree widths as small as possible. If such a goal cannot be achieved, we need to develop techniques that can use specific problem properties to resolve the difficulty arising from a large tree width.

6.3 Summary

In this chapter, we summarize the previous chapters in this dissertation and describe future research that may deserve further study in the future work. As we have pointed out, parameterized and exact computation plays important roles in bioinformatics and the design and implementation of parameterized algorithms is thus one of the most promising research areas that can possibly lead to the efficient solutions for NP-hard optimization problems in bioinformatics. This dissertation only initiates the effort to consider solving these problems this way. Additional techniques and methods are definitely needed and will be developed to solve more NP-hard problems in bioinformatics.

Bibliography

[1] T. Akutsu, "Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots", *Discrete Applied Mathematics*, 104: 45-62, 2000.

[2] N. Alon, R. Yuster, and U. Zwick, "Color-coding", *Journal of the ACM*, 42: 844-856, 1995.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Research*, 25(17): 3389-3402, 1997.

[4] E. Amir, "Efficient approximation for triangulation of minimum treewidth", *Proceedings of 17th Conference on Uncertainty in Artificial Intelligence*, 7-15, 2001.

[5] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree.", *SIAM Journal on Algebraic and Discrete Methods*, 8: 277-284, 1987.

[6] S. Arnborg and A. Proskurowski, "Linear time algorithms for NP-hard problems restricted to partial k-trees.", *Discrete Applied Mathematics*, 23: 11-24, 1989.

[7] E. Bachoore and A. Proskurowski, "Characterization and recognition of partial 3-trees", *SIAM Journal on Algebraic and Discrete Methods*, 7: 305-314, 1986.

[8] B. S. Baker, "Approximate algorithms for NP-complete problems on planar graphs.", *Journal of the ACM*, 41: 153-180, 1994.

[9] C. Bartels, "Fast algorithm for peptide sequencing by mass spectrometry.", *Biomed. Environ. Mass Spectrom.*, 19: 363-368, 1990.

[10] V. Bafna and S. Zhang, "FastR: Fast database search tool for non-coding RNA.", *Proceedings of the 3rd IEEE Computational Systems Bioinformatics Conference*, 52-61, 2004.

[11] H. L. Bodlaender, "Classes of graphs with bounded tree-width.", *Tech. Rep. RUU-CS-86-22*, Dept. of Computer Science, Utrecht University, the Netherlands, 1986.

[12] H. L. Bodlaender, "Better algorithms for the pathwidth and treewidth of graphs.", *Proceedings of the 18th international Colloquium on Automata, Languages and Programming*, Springer Verlag, Lecture Notes in Computer Science, 510: 544-555, 1991.

[13] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth", *SIAM Journal on Computing*, 25: 1305-1317, 1996.

[14] H. L. Bodlaender and A. M. C. A. Koster, "Safe separators for treewidth", *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments*, 70-94, 2004.

[15] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle, "Contraction and Treewidth Lower Bounds", *Proceedings of the 12th Annual European Symposium on Algorithms*, 628-639, 2004.

[16] M. Brown and C. Wilson, "RNA Pseudoknot Modeling Using Intersections of Stochastic Context Free Grammars with Applications to Database Search.", *Pacific Symposium on Biocomputing*, 109-125, 1995.

[17] L. Cai, R. Malmberg, and Y. Wu, "Stochastic Modeling of Pseudoknot Structures: A Grammatical Approach.", *Bioinformatics*, 19, $i66 - i73$, 2003.

[18] L. Cai, Y. Song, X. Huang, C. Liu, and J. Zhao, "A Case Study of Parameterized Reduction to MSO logic", manuscript.

[19] T. Chen, M. Y. Kao, M. Tepel, J. Rush, and G. M. Church, "A Dynamic Programming Approach to De Novo Peptide Sequencing via Tandem Mass Spectrometry.", *Journal of Computational Biology*, 8(3): 325-337, 2001.

[20] J. Chen, I. A. Kanj, and W. Jia, "Vertex Cover: Further Observations and Further Improvements", *Journal of Algorithms* 41(2): 280-301, 2001.

[21] B. Chor, M. Fellows, and D. W. Juedes, "Linear Kernels in Linear Time, or How to Save k Colors in $O(n^2)$ Steps", *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science*, 257-269, 2004.

[22] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre, "New lower and upper bounds for graph treewidth", *Proceedings of International Workshop on Experimental and Efficient Algorithms, Lecture Notes in Computer Sciences*, 2647: 70-80, 2003.

[23] B. Courcelle, "The Monadic Second Order Theory of Graphs I: Recognizable Sets of Finite Graphs", *Information and Computation*, 85(1):12-75, 1990.

[24] V. Dancík, T. A. Addona, K. R. Clauser, J. E. Vath, and P. A. Pevzner, "*De Novo* Peptide Sequencing via Tandem Mass Spectrometry.", *Journal of Computational Biology*, 6(3/4): 327-342, 1999.

[25] A. T. Dandjinou, N. Lévesque, S. Larose, J. Lucier, S. A. Elela, and R. J. Wellinger, "A Phylogenetically Based Secondary Structure for the Yeast Telomerase RNA.", *Current Biology*, 14: 1148-1158, 2004.

[26] R. G. Downey and M. R. Fellows, "Parameterized Complexity", Monographs in Computer Science, *Springer*, 1997.

[27] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, Cambridge Univ. Press, 1998.

[28] S. Eddy and R. Durbin, "RNA sequence analysis using covariance models.", *Nucleic Acids Research*, 22: 2079-2088, 1994.

[29] J. K. Eng, A. L. McCormack, and J. R. Yates III, "An Approach to Correlate Tandem Mass Spectral Data of Peptides with Amino Acid Sequences in A Protein Database", *Journal of the American Society of Mass Spectrometry*, 5(11): 976-989, 1994.

[30] D. Eppstein, "Subgraph Isomorphism in Planar Graphs and Related Problems.", *Journal of Graph Algorithms and Applications*, 3.3: 1-27, 1999.

[31] J. Ferńandez de Cossío, J. Gonzales, and V. Besada, "A computer program to aid the sequencing of peptides in collision-activated decomposition experiments.", *CABIOS* 11(4): 427-434, 1995.

[32] D. N. Frank and N. R. Pace, "Ribonuclease P: unity and diversity in a tRNA processing ribozyme.", *Annu Rev Biochem.*, 67: 153-180, 1998.

[33] A. Frank, S. Tanner, P. Pevzner, "Peptide Sequence Tags for Fast Database Search in Mass-Spectrometry", *Journal of Proteome Research*, 4(4), 1287 -1295, 2005.

[34] A. Frank and P. Pevzner, "PepNovo: De Novo Peptide Sequencing via Probabilistic Network Modeling", *Anal. Chem.*, 77(4), 964 -973, 2005.

[35] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *Freeman*, New York, 1979.

[36] D. Gautheret and A. Lambert, "Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles.", *Journal of Molecular Biology*, 313: 1003-1011, 2001.

[37] A. Godzik, A. Kolinski, and J. Skolnick, "Topology fingerprint approach to the inverse folding problem", *Journal of Molecular Biology*, 227: 227-238, 1992.

[38] S. J. Geobel, B. Hsue, T. F. Dombrowski, and P. S. Masters, "Characterization of the RNA components of a Putative Molecular Switch in the 3' Untranslated Region of the Murine Coronavirus Genome.", *Journal of Virology*, 78: 669-682, 2004.

[39] M. Hajiaghayi and N. Nishimura, "Subgraph isomorphism, log-bounded fragmentation and graphs of (locally) bounded treewidth", *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, 305-318, 2002.

[40] C. W. Hamm, W. E. Wilson, and D. J. Harvan, "Peptide sequencing program.", *CABIOS* 2, 365.

[41] Y. Han, B. Ma, and K. Zhang, "SPIDER: Software for Protein Identification from Sequence Tags Containing Sequencing Error", *Journal of Bioinformatics and Computational Biology*, 3(3):697-716, 2005.

[42] W. M. Hines, A. M. Falick, A. L. Burlingame, and B. W. Gibson, "Pattern-based algorithm for peptide sequencing from tandem high energy collision-induced dissociation mass spectra.", *J. Am. Soc. Mass. Spectrom.*, 3: 326-336, 1992.

[43] L. Holm and C. Sander, "Decision support system for evolutionary classification of protein structures", Proceedings of International Conference on Intelligent Systems for Molecular Biology, 5: 140-146, 1997.

[44] K. Ishikawa and Y. Niva, "Computer-aided peptide sequencing by fast atom bombardment mass spectrometry.", *Biomed. Environ. Mass Spectrom.*, 13: 373-380, 1986.

[45] R. J. Johnson and K. Biemann, "Computer program (seqpep) to aid in the interpretation of high-energy collision tandem mass spectra of peptides.", *Biomed. Environ. Mass Spectrom.*, 18: 945-957, 1989.

[46] S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. R. Eddy, "Rfam: an RNA family database.", *Nucleic Acids Research*, 31: 439-441, 2003.

[47] D. T. Jones, "Protein secondary structure prediction based on position-specific scoring matrices", *Journal of Molecular Biology*, 292(2): 195-202, 1999.

[48] R. J. Klein and S. R. Eddy, "RSEARCH: Finding Homologs of Single Structured RNA Sequences.", *BMC Bioinformatics*, 4:44, 2003.

[49] D. Kim, D. Xu, J. Guo, K. Ellrott, and Y. Xu, "Prospect II: protein structure prediction program for genome-scale applications", *Protein Engineering*, 16(9): 641-650, 2003.

[50] R. H. Lathrop, "The protein threading problem with sequence amino acid interaction preferences is NP-complete", *Protein Engineering*, 7(9): 1059-1068.

[51] C. Liu, Y. Song, R. Malmberg, and L. Cai, "Profiling and Searching for RNA Pseudoknot Structures in Genomes.", *Lecture Notes in Computer Science* 3515, 968-975.

[52] J. Liu, B. Ma, and M. Li, "PRIMA: Peptide Robust Identification from MS/MS Spectra", *Proceedings of the Third Asia-Pacific Bioinformatics Conference*, 181-190, 2005.

[53] C. Liu, Y. Song, B. Yan, Y. Xu, and L. Cai, "Fast De Novo Peptide Sequencing and Spectral Alignment", *Proceedings of the Pacific Symposium on Biocomputing 2006*, 255-266, 2006.

[54] T. M. Lowe and S. R. Eddy, "tRNAscan-SE: A Program for Improved Detection of Transfer RNA genes in Genomic Sequence.", *Nucleic Acids Research*, 25: 955-964, 1997.

[55] B. Lucena, "A new lower bound for tree-width using maximal cardinality search.", *SIAM Journal on Discrete Mathematics*, 16: 345-353, 2003.

[56] B. Ma, K. Zhang, C. Hendrie, C. Liang, M. Li, A. Doherty-Kirby, and G. Lajoie, "PEAKS: Powerful Software for Peptide De Novo Sequencing by Tandem Mass Spectrometry", *Rapid Communication in Mass Spectrometry*, 17(20): 2337-2342. 2003.

[57] J. Matousek and R. Thomas, "On the complexity of finding iso- and other morphisms for partial k-trees.", *Discrete Mathematics*, 108: 343-364, 1992.

[58] N. Nameki, B. Felden, J. F. Atkins, R. F. Gesteland, H. Himeno, and A. Muto, "Functional and structural analysis of a pseudoknot upstream of the tag-encoded sequence in E. coli tmRNA.", *Journal of Molecular Biology*, 286(3): 733-744, 1999.

[59] G. L. Nemhauser and L. E. Trotter, "Vertex packing: structural properties and algorithms", *Mathematical Programming*, 8: 232-248, 1975.

[60] V. T. Nguyen, T. Kiss, A. A. Michels, and O. Bensaude, "7SK small nuclear RNA binds to and inhibits the activity of CDK9/cyclin T complexes.", *Nature* 414: 322-325, 2001.

[61] K. G. Olesen and A. L. Madsen, "Maximal prime subgraph decomposition of Bayesian networks", *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 32: 21-31, 2002.

[62] A. V. Oppenheim and R. W. Schafer, "Discrete-time Signal Processing", *Prentice Hall*, 1989.

[63] D. N. Perkins, D. J. Pappin, D. M. Creasy, and J. S. Cottrell, "Probability-based Protein Identification by Searching Sequence Databases Using Mass Spectrometry Data", *Electrophoresis*, 20(18), 3551-3567, 1999.

[64] L. Perković and B. Reed, "An improved algorithm for finding tree decompositions of small tree width", *Proceedings of the 25th International Workshop on Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science*, 1665: 148-154, 1999.

[65] P. A. Pevzner, V. Dancík, and C. L. Tang, "Mutation-Tolerant Protein Identification by Mass-Spectrometry.", *Proceedings of The Fourth Annual International Conference on Computational Molecular Biology*: 231-236, 2000.

[66] J. T. Prince, M. W. Carlson, R. Wang, P. Lu, and E. M. Marcotte, "The Need for a Public Proteomics Repository", *Nature Biotechnology*, 22(4): 471-472, 2004.

[67] E. Rivas and S. Eddy, "The language of RNA: a formal grammar that includes pseudoknots.", *Bioinformatics*, 16: 334-340, 2000.

[68] E. Rivas and S. R. Eddy, "Noncoding RNA gene detection using comparative sequence analysis.", *BMC Bioinformatics*, 2:8, 2001.

[69] E. Rivas, R. J. Klein, T. A. Jones, and S. R. Eddy, "Computational identification of noncoding RNAs in E. coli by comparative genomics.", *Current Biology*, 11: 1369-1373, 2001.

[70] E. Rivas and S. R. Eddy, "A dynamic programming algorithm for RNA structure prediction including pseudoknots.", *Journal of Molecular Biology*, 285: 2053-2068, 1999.

[71] N. Robertson and P. D. Seymour, "Graph Minors II. Algorithmic aspects of tree-width.", *Journal of Algorithms*, 7: 309-322, 1986.

[72] N. Robertson and P. D. Seymour, "Graph Minors III. Planar tree width.", *Journal of Combinatorial Theory Series B*, 36: 49-64, 1984.

[73] N. Robertson and P. D. Seymour, "Graph Minors IV. Tree width and well-quasi-ordering.", *Journal of Combinatorial Theory Series B*, 48: 227-254, 1990.

[74] N. Robertson and P. D. Seymour, "Graph Minors V. Excluding a planar graph.", *Journal of Combinatorial Theory Series B*, 41: 92-114, 1986.

[75] N. Robertson and P. D. Seymour, "Graph Minors X. Obstructions to tree decomposition.", *Journal of Combinatorial Theory Series B*, 52: 153-190, 1991.

[76] N. Robertson and P. D. Seymour, "Graph Minors XIII. The disjoint paths problem.", *Journal of Combinatorial Theory Series B*, 63: 65-110, 1995.

[77] T. Sakurai, T. Matsuo, H. Matsuda, and I. Katakuse, "Paas 3: A computer program to determine probable sequence of peptides from mass spectrometric data.", *Biomed. Mass Spectrom.*, 11(8): 396-399, 1984.

[78] J. Scott, T. Ideker, R. M. Karp, R. Sharan, "Efficient algorithms for detecting signaling pathways in protein interaction networks", *Proceedings of the Ninth Annual International Conference on Research in Computational Molecular Biology*, 1-13, 2005.

[79] B. C. Searle, S. Dasari, M. Turner, A. P. Reddy, D. Choi, P. A. Wilmarth, A. L. McCormack, L. L. David, and S. R. Nagalla, "High-Throughput Identification of Proteins and Unanticipated Sequence Modifications Using a Mass-Based Alignment Algorithm for MS/MS De Novo Sequencing Results", *Anal. Chem*, 76(8):2220-30, 2004.

[80] D. Seese, "The Structure of Models of Decidable Monadic Theories of Graphs", *Annals of Pure and Applied Logic*, 53: 169-195, 1991.

[81] P. D. Seymour and R. Thomas, "Call routing and the rat catcher", *Combinatorica* 14: 127-241, 1994.

[82] J. Shi, T. L. Blundell, and K. Mizuguchi, "FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure- dependent gap penalties", *Journal of Molecular Biology*, 310(1): 243-257, 2001.

[83] M. M. Siegel and N. Bauman, "An efficient algorithm for sequencing peptides using fast atom bombardment mass spectral data.", *Biomed. Environ. Mass Spectrom.*, 15: 333-343, 1988.

[84] R. Srinivasan and G. D. Rose, "Ab Initio Prediction of Protein Structure Using LINUS.", *PROTEINS: Structure, Function, and Genetics*, 47:489-495, 2002.

[85] Y. Song, C. Liu, X. Huang, R. Malmberg, Y. Xu, and L. Cai, "Efficient parameterized algorithms for biopolymer structure-sequence alignment", *Proceedings of the Fifth*

Workshop on Algorithms in Bioinformatics, Lecture Notes in Bioinformatics 3692, 376-388, 2005.

[86] Y. Song, C. Liu, R. L. Malmberg, F. Pan, and L. Cai, "Tree decomposition based fast search of RNA structures including pseudoknots in genomes", *Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference*, 223-224, 2005.

[87] D. L. Tabb, A. Saraf, and J. R. Yates, "GutenTag: High-Throughput Sequence Tagging via an Empirically Derived Fragmentation Model", *Analytical Chemistry*, 75: 6415-6421, 2003.

[88] S. Tanner, H. Shu, A. Frank, L. C. Wang, E. Zandi, M. Mumby, P. A. Pevzner, and V. Bafna, "InsPecT: Identification of Posttranslationally Modified Peptides from Tandem Mass Spectra", *Analytical Chemistry*, 77(14): 4626-4639, 2005.

[89] J. A. Taylor and R. S. Johnson, "Sequence database searches via *de novo* peptide sequencing by tandem mass spectrometry.", *Rapid Commun. Mass Spectrom.*, 11: 1067-1075, 1997.

[90] J.A. Taylor and R.S. Johnson, "Implementation and uses of automated de novo peptide sequencing by tandem mass spectrometry", *Anal.Chem.*, 73(11), 2594-2604, 2001.

[91] D. Tsur, S. Tanner, E. Zandi, V. Bafna, and P. Pevzner, "Identification of Post-translational Modifications by Blind Search of Mass Spectra", *Nature Biotechnology*, 23(12):1562-1567, 2005.

[92] Y. Uemura, A. Hasegawa, Y. Kobayashi, and T. Yokomori, "Tree adjoining grammars for RNA structure prediction.", *Theoretical Computer Science*, 210: 277-303, 1999.

[93] G. Wang and R. L. Dunbrack, Jr. "PISCES: a protein sequence culling server", *Bioinformatics*, 16: 257-268, 2000.

[94] Z. Weinberg and W. L. Ruzzo, "Faster genome annotation of non-coding RNA families without loss of accuracy.", *Proceedings of the Eighth Annual International Conference on Computational Molecular Biology*, 243-251, 2004.

[95] M. R. Wilkins, E. Gasteiger, A. A. Gooley, B. R. Herbert, M. P. Molloy, P. A. Binz, K. Ou, J. C. Sanchez, A. Bairoch, K. L. Williams, and D. F. Hochstrasser, "High-throughput Mass Spectrometric Discovery of Protein Post-Translational Modifications", *Journal of Molecular Biology*, 289(3): 645-657, 1999.

[96] Y. Xu, D. Xu and E. C. Uberbacher, "An Efficient Computational Method for Globally Optimal Threading", *Journal of Computational Biology*, 5: 597-614, 1998.

[97] J. Xu, M. Li, D. Kim, and Y. Xu, "RAPTOR: Optimal Protein Threading by Linear Programming", *Journal of Bioinformatics and Computational Biology*, 1(1): 95-117, 2003.

[98] J. Xu, M. Li, D. Kim, and Y. Xu, "Protein threading by linear programming", *Biocomputing: Proceedings of the 2003 Pacific Symposium*, 264-275, 2003.

[99] J. Xu, "Rapid Problem Side-Chain Packing via Tree Decomposition", Proceedings of the Ninth Annual International Conference on Research in Computational Molecular Biology, 423-429, 2005.

[100] J. Xu, F. Jiao, and B. Berger, "A Tree-Decomposition Approach to Protein Structure Prediction", *Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference*, 247-256, 2005.

[101] B. Yan, C. Pan, V. N. Olman, R. L. Hettich, and Y. Xu, "A graph-theoretic approach for the separation of b and y ions in tandem mass spectrometry.", *Bioinformatics*, 21(5): 563-574, 2005.

[102] B. Yan, T. Zhou, P. Wang, Z. Liu, V. A. Emanuele II, V. Olman, and Y. Xu, "A Point-Process Model for Rapid Identification of Post-Translational Modifications", *Proceedings of 2006 Pacific Symposium on Biocomputing*, 327-338, 2006.

[103] Z. Yang, Q. Zhu, K. Luo, and Q. Zhou, "The 7SK small nuclear RNA inhibits the Cdk9/cyclin T1 kinase to control transcription.", *Nature* 414: 317-322, 2001.

[104] J. R. Yates, P. R. Griffin, L. E. Hood, J. X. Zhou, "Computer aided interpretation of low energy ms/ms mass spectra of peptides.", *Techniques in Protein Chemistry II*, 477-485, Academic Press, San Diego.

[105] J. R. Yates III, J. K. Eng, and A. L. McCormack, "Mining Genomes: Correlating Tandem Mass Spectra of Modified and Unmodified Peptides to Sequences in Nucleotide Databases", *Analytical Chemistry*, 67(18): 3202-3210, 1995.

[106] J. Zhu and Z. Weng, "FAST: A Novel Protein Structure Alignment Algorithm", *Proteins: Structure, Function and Bioinformatics*, 58(3): 618-627, 2005.

[107] D. Zidarov, P. Thibault, M. J. Evans, and M. J. Bentrand, "Determination of primary structure of peptides using fast atom bombardment mass spectrometry.", *Biomed. Environ. Mass Spectrum.*, 19: 13-16, 1990.