COMPUTATIONAL INVESTIGATION ON PROTEIN SEQUENCING AND GENOME REARRANGEMENT PROBLEMS

by

Letu Qingge

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2018

©COPYRIGHT

by

Letu Qingge

2018

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Binhai Zhu for guiding me to complete my PhD dissertation. I truly appreciate to his great support and patience from the bottom of my heart.

I also would like to thank all my committee members, Dr. Brittany Terese Fasy, Dr. David L. Millman and Dr. Brendan Mumey for their helpful comments.

TABLE OF CONTENTS

1. INTRODUCTION 1

Background 1
Contributions 2
De Novo Protein Sequencing Problem 2
Protein Scaffold Filling Problem 3
Genome Rearrangement Problems 4
Dissertation Organization 5

2. *DE NOVO* PROTEIN SEQUENCING BASED ON TOP-DOWN AND BOTTOM-UP TANDEM MASS SPECTRA 7

Related Work 7
Preliminaries 9
Experimental Dataset 12
Top-Down Scaffold Construction 13
Extending the Top-Down Scaffold with Bottom-Up Peptides 15
A Heuristic Approach 17
Experimental Results 18
Top-Down Scaffold 18
Bottom-Up Alignment 19
The Constructed Scaffold 20
Concluding Remarks 21

3. FILLING A PROTEIN SCAFFOLD WITH A REFERENCE 22

Related Work 22
Preliminaries 24
Algorithms for PSF When not Allowing Gaps 27
Algorithms and Empirical Results for PSF When Allowing Gaps 29
PSF is in P 29
Practical Algorithms for PSF 31
Simulation Results 33
Empirical Results for Antibody Proteins 35
Empirical Results for Mammalian Proteins 35
Algorithms and Empirical Results for CP-PSF 37
CP-PSF is in P 37
Practical Algorithms for CP-PSF 38
Empirical Results for Antibody Proteins 40
Empirical Results for Mammalian Proteins 42

TABLE OF CONTENTS – CONTINUED

Concluding Remarks 42

4. AN FPT APPROXIMATION ALGORITHM FOR THE DCJ DISTANCE 44

Related Work 44
Preliminaries 46
Breakpoint Graph of Signed Genomes 46
The Signed DCJ Distance Formula 47
Breakpoint Graph for Unsigned Genomes and the UDCJ Problem 47
Converting Unsigned Genomes into Signed Ones 49
An FPT-Time Approximation Algorithm 50
Why Caring Small Cycles 51
Finding 2-cycles 52
Finding 2,3-cycles 53
Performance Analysis 55
The Approximation Factor 55
The Time Complexity 60
Simulation Results 60
Concluding Remarks 61

5. THE ONE-SIDED EXEMPLAR ADJACENCY NUMBER PROBLEM 62

Related Work 63
Preliminaries 65
A New Formulation of the Problem 67
Hardness of MIS-CIG 68
Algorithms 70
A Factor 2-Approximation 70
A Local Search Improvement 71
ILP Formulation on the Reduced Instance 73
Simulation Results 78
Concluding Remarks 83

6. THE MINIMUM COPY NUMBER GENERATION PROBLEM 84

Related Work 85
The Minimum Copy Number Generation (MCNG) Problem 88
Preliminaries 88

TABLE OF CONTENTS – CONTINUED

NP-completeness for MCNG When Duplications are Tandem 89
Exemplar MCNG with Tandem Duplications 93
Some Variations of the Minimum Copy Number Generation Problem 98
Practical Solution for the Minimum Copy Number Generation Problem 101
A Practical Greedy Solution 101
Empirical Results for Simulated Data 103
Discussion 104
The Order-Switching Scenario for the Exemplar Case 108
Relation to the Whole Genome Duplication Problem 108
Concluding Remarks 109

7. CONCLUSION 111

De Novo Protein Sequencing Problem 111
Genome Rearrangement Problems 112
Future Work 113

REFERENCES CITED 114

APPENDIX: The BLOSUM62 score matrix 123

LIST OF TABLES

Table Page

2.1 The residue masses (in Da), rounded to integers, of 20 amino acids. 11

2.2 Alemtuzumab top-down scaffold results for 4-mer with different epsilon (in Da). 14

3.1 The average number of aligned pairs computed between `MabCampath-H` and the filled scaffold $\mathcal{S}'$ (by Algorithm 3.1), and between `MabCampath-H` and the filled scaffold $\mathcal{S}''$ (by Algorithm 3.2), over ten tries. Here $\mathcal{S}$ is constructed from `MabCampath-H` by deleting $x\%$ of amino acids to have 8 contigs, followed with the shuffling of y pairs of amino acids. 34

3.2 The average number of aligned pairs computed between `MabCampath-H` and the filled scaffold $\mathcal{S}'$ (by Algorithm 3.1), and between `MabCampath-H` and the filled scaffold $\mathcal{S}''$ (by Algorithm 3.2), over ten tries. Here $\mathcal{S}$ is constructed from `MabCampath-H` by deleting $x\%$ of amino acids to have 8 contigs, followed with the shuffling of y pairs of amino acids, and finally we pick two of the contigs and append randomly 2-15 amino acids in X to them. 34

3.3 Empirical results for the three computed $\mathcal{I}'_x$ using Algorithm 3.1. In all the tables, **ref_length, tar_length** represents the length of the reference and target protein sequence respectively. mp denotes matched pairs between two amino acids. 35

3.4 Empirical results for the three computed $\mathcal{I}''_x$ using Algorithm 3.2. mp denotes matched pairs between two amino acids. 36

3.5 Empirical results for the three computed $\mathcal{I}'_x$ using Algorithm 3.1, with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference. 36

3.6 Empirical results for the three computed $\mathcal{I}''_x$ using Algorithm 3.2, with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference. 37

LIST OF TABLES – CONTINUED

Table Page

3.7 Empirical results for the three computed $\mathcal{S}'_x$ using Algorithm 3.3, with respect to the antibody dataset. `M-H` (resp. `M-L`) represents the heavy (resp. light) chain of `MabCampath`, and `H-H` represents the heavy chain of `Humira`. 41

3.8 Empirical results for the three computed $\mathcal{S}''_x$ using Algorithm 3.4, with respect to the antibody dataset. `M-H` (resp. `M-L`) represents the heavy (resp. light) chain of `MabCampath`, and `H-H` represents the heavy chain of `Humira`. 41

3.9 Empirical results for the three computed $\mathcal{S}'_x$ using Algorithm 3.3. , with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference. 41

3.10 Empirical results for the three computed $\mathcal{S}''_x$ using Algorithm 3.4, with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference. 42

4.1 The computed approximation solution value k' versus the potential optimal solution value k. k' is computed over 5 tries. 61

5.1 Comparison results (signed genome) between the number of shared adjacencies n_1 from our algorithms and n_2 from GREDU. The gap _ indicate that we get a core dumped warning from GREDU and can not see the results after 10 tries. 81

5.2 The running time t_1 (seconds) and the number of adjacencies n_1 for unsigned genomes, averaged over 10 tries. No comparison is done with GREDU as it only handles signed genomes... 82

6.1 The testing results for the setting $\{L_{1j}\ (j = 1, 2, 3)\}$. Note that k_{ij} represents the average operation number, t_{ij} represents the average running time in seconds and n_{ij} represents the average final sequence length, all over five tries. 105

6.2 The testing results for the setting $\{L_{2j}$ (j=1,2,3)$\}$. The corresponding parameters are similar to Table 6.1. 106

LIST OF TABLES – CONTINUED

Table Page

6.3 The testing results for the setting $\{L_{3j}$ (j=1,2,3)$\}$. The corresponding parameters are similar to Table 6.1. 107

LIST OF FIGURES

Figure Page

2.1 Illustration for the reduction from the subset sum instance. 17

2.2 Three lines of target in this figure denotes the whole target protein sequence with 214 amino acids which is supposed to be constructed and top-down represents our constructed top-down scaffold containing four contigs using De Bruijn graph. Note that the amino acid I has the same mass to the amino acid L. So we convert all I to L in our results. 19

2.3 Illustration of bottom-up peptides alignment with top-down contigs at the first iteration. The other parameters, such as target and top-down are the same as in Figure 2.2. 20

2.4 Illustration of bottom-up alignments with updated top-down contigs at the second iteration. The other parameters, such as target and top-down are the same as in Figure 2.2. 20

2.5 Three lines of target denotes the whole target protein sequence which is supposed to be constructed and three lines of scaffold represents the final scaffold after aligning bottom-up peptides with the top-down contigs. 21

4.1 The DCJ Operation 48

4.2 (a) The breakpoint graph $G_s(\bar{X}, \bar{Y})$ for the signed case, where $\bar{X} = \{[1, -3, -2, 4], [5, -7, -8, 6, 9]\}$ and $\bar{Y} = \{[1, 2, 3, 4], [5, 6, 7, 8, 9]\}$. (b) The breakpoint graph $G(X, Y)$ for the unsigned case, where $X = \{[1, 3, 2, 4], [5, 7, 8, 6, 9]\}$ and $Y = \{[1, 2, 3, 4], [5, 6, 7, 8, 9]\}$. (a) is a cycle decomposition of (b) 50

5.1 Formulation of the one-sided EAN problem potentially as MIS in a colored interval graph. Note that in $\mathcal{G}$ the two intervals labeled with color 1 correspond to the adjacency ab, while the three intervals labeled with color 2 correspond to the adjacency de. 67

5.2 Illustration for the reduction from the 3SAT instance ϕ. To save space, only the construction for $x_1, \overline{x}_1$ and $x_2, \overline{x}_2$ are shown. 70

LIST OF FIGURES – CONTINUED

Figure Page

5.3 Construction of an instance with $3n + 2$ intervals, with $2n$ colors $\{0, 1, ..., 2n-1\}$. Illustration for the proof of Theorem 8 by swapping intervals in bottom row with intervals in top row. 72

6.1 An example for the Copy Number Transformation problem. With two duplications and one deletion, $CNP\ C_1$ is transformed into $CNP\ C_2$. 86

6.2 An example for converting a genome to the next one using a single duplication, with a total of two duplication operations. Notice that when the CNP changes from $\langle 1, 2, 2, 2, 2, 1\rangle$ to $\langle 1, 3, 2, 3, 3, 1\rangle$ the second 2 (the number of c's) does not change. In other words, this cannot be implemented with a single duplication/deletion operation on the CNP directly, while obviously with our model the third CNP (and the corresponding genome) can be obtained with one duplication from the genome associated with the second CNP $\langle 1, 2, 2, 2, 2, 1\rangle$..... 87

6.3 An X-monotone chain $\mathcal{C} = \langle (1, a_1), ..., (i, a_i), ..., (m, a_m)\rangle$ converted from the copy number file $C = \langle c_1, c_2, ..., c_m\rangle$, where $c_i = 2^{a_i}$ for $i = 1, ..., m$, and the trapezoid decomposition of $\mathcal{C}$. 96

LIST OF ALGORITHMS

Algorithm Page

3.1 *Align+Greedy($\mathcal{P}$,$\mathcal{I}$,X)* 31

3.2 *LocalSearch($\mathcal{P}$,$\mathcal{I}$,X)* 32

3.3 *ContigAlign+Greedy($\mathcal{P}$,$\mathcal{S}$,X)* 39

3.4 *ContigLocalSearch($\mathcal{P}$,$\mathcal{S}$,X)* 40

4.5 Finding 2-cycles 53

4.6 Finding 2,3-cycles 54

4.7 UDCJ-Final 54

5.8 A Factor-2 Approximation Algorithm for MIS-CIG 71

6.9 Greedy Algorithm for MCNG 110

ABSTRACT

De novo protein sequencing and genome rearrangement problems are the classical problems in bioinformatics. *De novo* protein sequencing problem try to determine the whole sequence of amino acids based on the mass spectrometry data without using the database search. Genome rearrangement problems try to recognize the evolutionary process between two species.

In this dissertation, first, we describe the process of constructing target protein sequences by utilizing mass spectrometry based data from both top-down and bottom-up tandem mass spectra. In addition to using data from mass spectrometry analysis, we also utilize techniques for *de novo* protein sequencing using a homologous protein sequence as a reference to attempt to fill in any remaining gaps in the constructed protein scaffold. Initial results for analysis on real datasets yield over 96-100% coverage and 73-91% accuracy with the target protein sequence. Second, we use different genome rearrangement operations to transform one genome to another such that the similarity between two genomes is maximized. We explore these problems in terms of theoretical and experimental analysis. For sorting unsigned genome problem by double cut and join (DCJ) operation, we design a randomized fixed parameter tractable (FPT) approximation algorithm for computing the DCJ distance with an approximation factor $4/3 + \epsilon$, and the running time $O^*(2^{d^*})$, where d^* represents the optimal DCJ distance. For one-sided exemplar adjacency number problem, we reformulate the problem as maximum independent set in a colored interval graph and hence reduce the appearance of each gene at most twice. Moreover, we design a factor-2 approximation and also show that the approximation factor can not be improved less than 2 by some local search technique. At last, we apply integer linear programming to solve the reduced instance exactly. For the minimum copy number generation problem, we analyze the complexity of different variations of this problem and show a practical algorithm for the general case based on greedy method.

1

INTRODUCTION

Background

Proteins are the building blocks of life as humans know it. They form the structure of living organisms, function as enzymes and catalysts, and perform the basic chemical reactions that organisms need to survive and flourish. Therefore no mystery that why humans would be ever so eager to reproduce and manufacture specific proteins. Ailments of our bodies could be remedied by repairing or replenishing missing or mutated proteins. In addition to internal deficiencies, diseases external to our bodies could be cured or mitigated. Because of this potential, research in the field of protein sequencing has grown rapidly, and the related technologies have advanced significantly. In modern times, getting a whole complete protein sequence is still a challenging problem for researchers. The constructed sequence coverage and accuracy were around 50% to over 80% depending on the data. Obtaining intact protein sequences is very important. We can analyze whole protein sequence structures and functions to make drugs that may help cure diseases. The protein sequences obtained from biology labs are always given by several protein peptides. Researchers usually use the method of assembling mass spectra of overlapped peptides consecutively to obtain longer contigs. We define amino acids are 20 different types of alphabet. A peptide is a sequence of amino acids. A contig is a continuous overlapping peptides, which does not include any gaps. A *scaffold* $\mathcal{S}$ is a list of ordered contigs $C_1, C_2, ..., C_m$ possibly with gaps between adjacent contigs, where each contig is a sequence of amino acids. For example, $\mathcal{S} = \langle C_1, C_2, C_3 \rangle$, where $C_1 =$ $\boxed{\text{DIQMIQS}}$, $C_2 =$ $\boxed{\text{SASVDRRTLT}}$, and $C_3 =$ $\boxed{\text{SQADYEQH}}$. Therefore the arising new problem

is how to fill the gaps into the scaffold to obtain a whole complete protein sequence.

Genomes in different species (even in similar species) are different from one another, because of the genome rearrangements and mutations. The genome rearrangements includes a variety of operations, such as reversal, translocations, fusions, fissions, transpositions and double cut and join (DCJ), etc. The mutation includes insertion, deletion and substitution operations. The definition of these genome rearrangement operations can be seen in the last section of chapter 5. We study the genome rearrangement problems to try to understand what is the evolutionary process between two genomes. For example, humans and mice share the same genes, but have different gene orders. Hence the problem is how many operations are needed to transform one to another. Given two genomes, use the minimum number of genome rearrangement operations to transform one genome to another such that the similarity between two genomes is maximized or minimized depending on which similarity measurement is applied.

Contributions

De Novo Protein Sequencing Problem

We study the problem of constructing the target protein sequence by utilizing data from both top-down, as well as bottom-up, tandem mass spectrometry. Tandem mass spectrometry is an essential technique in sequence identification of peptides and other biomolecules, which generates information on the molecular weight and amino acid sequence of a peptide.

- We construct the top-down scaffold using De Bruijn graph, which is a directed graph representing overlaps between k-mer components, where k-mer is a substring of length k. In the De Bruijn graph, each node represents a series

of overlapping k-mers and each edge denotes $k-1$ overlaps between two k-mers.

- We prove the *de novo* protein sequencing by combining top-down and bottom-up data problem is NP-complete.

- We present a heuristic algorithm to align bottom-up peptides with the top-down contigs to extend the coverage of the scaffold. The designed heuristic algorithm does not have a bound or approximation factor for the problem.

Protein Scaffold Filling Problem

We study the problem of filling an incomplete protein sequence (or *sequence scaffold*) $\mathcal{I}$ to obtain $\mathcal{I}'$, with respect to a complete reference protein $\mathcal{P}$, such that the number of aligned pairs between $\mathcal{I}'$ and $\mathcal{P}$ is maximized. When the scaffold is given as a sequence $\mathcal{S}$ of contigs, whose contents should not be altered, the problem is to fill $\mathcal{S}$ into $\mathcal{S}'$ such that the number of aligned pairs between $\mathcal{S}'$ and $\mathcal{P}$ is maximized.

- For the former problem, we present a simple factor-2 approximation algorithm based on dynamic programming and greedy method when no gaps are allowed to insert into both $\mathcal{P}$ and $\mathcal{I}$.

- We show that the former problem is polynomial-time solvable in $O(n^{22})$ time and the later problem is polynomial-time solvable in $O(n^{26})$ using dynamic programming when gaps are allowed to insert into both $\mathcal{P}$ and $\mathcal{I}$.

- We present practical heuristic algorithms based on the greedy algorithm and local search technique for both problems.

- We test our algorithms both on the simulated data and real data.

Genome Rearrangement Problems

We investigate the problem of computing the double cut and join (DCJ for short) distance between two unsigned genomes. DCJ operation unifies all traditional genome rearrangement operations. Given two genomes, apply the minimum number of the DCJ operations to transform one genome into the other. The definition of genome rearrangement operations can be seen in the last section of Chapter 5.

- We devise a fixed-parameter tractable approximation algorithm for sorting unsigned genomes by the DCJ operations, the approximation factor reaches $4/3 + \varepsilon$, and the running time is $O^*(2^{d^*})$, where O^* denotes the polynomial factors are omitted and d^* represents the optimal double cut and join distance.

We explore the one-sided exemplar adjacency number problem in which we delete duplicated genes from the generic genome with duplicated genes $\mathcal{G}$ to obtain an exemplar genome G, such that the number of adjacencies between G and given exemplar genome H is maximized.

- We reformulate and relax the one-sided problem as the maximum independent set in a colored interval graph and hence reduce the appearance of each gene in the reduced genome to at most twice.

- We show the new formulation is still NP-complete, and design a factor-2 approximation algorithm. Moreover, we also prove that the approximation factor can not be improved within $2 - \epsilon$ by some local search technique.

- We use integer linear programming to solve the relaxed problem exactly.

We study the minimum copy number generation (MCNG) problem, namely, given a genome G and a specific copy number profile (CNP) (i.e., the number of copies of each gene along a chromosome) C, use the minimum number of segment

duplications and deletions operations on G to obtain some genome H which has a CNP C.

- We show that the problem is NP-complete when the duplications are tandem. Tandem duplication means copy identical chromosome segments and insert to adjacent position for them.

- If G is exemplar (which means each gene from each gene family appears exactly once in G) and all components in C are power of two's, then the problem can be solved in a linear time.

- We also show that two variations of the MCNG problems are at least as hard as Set Cover in terms of approximability and fixed parameter tractability.

- For the general minimum copy number generation (GMCNG) problem, i.e., when both segment duplications and deletions are allowed, we design a practical algorithm with fast running time.

Dissertation Organization

The dissertation is organized as follows. In chapter 2, we divide the *de novo* protein sequencing problem into two distinct phases: constructing a scaffold from the top-down data, extending the scaffold by using bottom-up peptides.

In chapter 3, we discuss the protein scaffold filling problems. We show that the problems are polynomial-time solvable, but the running time is high. Due to the high time complexity, the exact algorithms proposed in this chapter are impractical, hence we present several heuristic algorithms based on the greedy and local search, trying to solve the problems practically.

In chapter 4, we explore the problem of computing the double cut and join (DCJ) distance between two unsigned genomes. This problem was shown to be NP-hard and the best polynomial-time approximation algorithms for the unsigned DCJ distance problem has a factor $1.408 + \epsilon$ [21]. We design a factor $4/3 + \epsilon$ fixed-parameter tractable (FPT) approximation algorithm for the DCJ distance, improving the previous (polynomial-time approximation) factor of $1.408 + \epsilon$. The running time is bounded by $O^*(2^{d^\star})$, where O^* denotes the polynomial factors are omitted and $d^\star$ represents the optimal DCJ distance.

In chapter 5, We study the one-sided exemplar adjacency number problem (One-sided EAN) in which we delete duplicated genes from the genome $\mathcal{G}$ to obtain an exemplar genome G, such that the number of adjacencies between G and given exemplar genome H is maximized. We reformulate the problem as the maximum independent set in a colored interval graph and show that the problem is NP-complete. We also design an efficient approximation algorithm for the problem and then use an integer linear programming to solve the reduced problem exactly.

In chapter 6, we investigate the minimum copy number generation problem, which is to use the minimum number of segment duplications and deletions operations to convert one genome G into another G' such that the copy number profile of G' satisfy a given copy number profile C. We show that the problem is NP-complete when the given genome is generic with tandem duplications and deletions operations and discuss several versions of the problem.

In chapter 7, we conclude our dissertation and discuss the future work.

7

DE NOVO PROTEIN SEQUENCING BASED ON TOP-DOWN AND BOTTOM-UP TANDEM MASS SPECTRA

We describe the process of constructing a target protein sequence by utilizing mass spectrometry based top-down, as well as bottom-up tandem mass spectra. The method of protein construction outlined in the chapter is shown to be a promising research avenue since combining both top-down and bottom-up tandem mass spectra.

Related Work

Protein sequencing plays an important role in identifying protein functions, quantifying proteins in complex biological samples, and analyzing protein-protein interactions. The problem was initiated as early as in 1950 by Edman, but due to the high cost of traditional Edman degradation experiments [38], since 1987 tandem mass spectrometry has become a popular method for protein sequencing [62].

There are two main approaches for mass spectrometry-based protein sequencing: database search [32, 42, 74, 84, 102] and *de novo* sequencing [1, 35, 82, 88, 97]. When the target protein sequence is included in a protein sequence database, which is commonly generated from annotated genes in genomes, tandem mass spectra can be searched against the database to identify and sequence the target protein. A variety of protein sequencing database exist, such as National Center of Biotechnology Information, Protein Data Bank and Gene Ontology etc, in which store comprehensive different species protein information containing a particular protein family or group of proteins. However, the sequences of some important proteins, such as antibodies and snake venom proteins, cannot be obtained from genomes [64]. As a result, *de novo* protein sequencing, which constructs a protein sequence without protein databases, is the method of choice for sequencing such proteins. This chapter studies

a challenging computational problem in *de novo* protein sequencing, which is to construct a complete unknown protein sequence.

There are several attempts for *de novo* protein sequencing. One popular approach is based on the *spectrum graph* [6, 20, 35]. A spectrum graph usually be represented as a vertical bar or peak graph in which each bar (peak) represents a mass to charge ratio and length of the bar corresponds to relative intensity. We use spectrum graph to generate peptides. In a given spectrum graph, each peak may generate a few nodes in the spectrum graph. An edge is added between two nodes u and v if the difference between the corresponding prefix masses of u and v is (approximately) the residue mass of an amino acid. The edge is then labeled with that amino acid. The problem is to compute a heaviest intensity weighted path in the graph, which can be solved using dynamic programming. Several software tools, such as in [25, 26, 46, 70, 71], were developed using the spectrum graph.

In the past decade, bottom-up mass spectrometry has dominated *de novo* protein sequencing [8, 10, 49, 69]. In essence, bottom-up mass spectrometry approach consists of three steps: break a protein into many peptides, then generate tandem mass spectra of the peptides, and finally stack and align their corresponding peptide sequences (obtained from *de novo* peptide sequencing) together to obtain the original protein sequence. We use PEAKS [71] to generate all bottom-up peptides.

Even though the bottom-up approach has been routinely used, it has a huge drawback, which is, overlapping regions of peptide pairs might be too short to be confidently identified, which are essential for protein sequencing. Hence, bottom-up methods often produce incomplete protein sequence fragments with many gaps. Recently, due to the development of high accuracy and high resolution mass spectrometers, such as Fourier transform ion cyclotron resonance (FTICR) and Orbitrap$^{\text{TM}}$ [83], it is now possible to sequence proteins directly using top-down mass

spectrometry, which generates high resolution tandem mass spectra of whole proteins. Unfortunately, due to missing and noise peaks, top-down mass spectrometry, while could handle proteins of a total mass up to 30,000 Dalton (Da), still fails to provide a guaranteed high protein sequence coverage . [63, 96, 98]

In 2014, Liu *et al.* [68] proposed a *de novo* protein sequencing method by combining top-down and bottom-up mass spectrometry. Simply speaking, top-down tandem mass spectra generates a skeleton of the protein sequence, usually several contigs and gaps, called a *scaffold* henceforth. Then, bottom-up tandem mass spectra are employed to fill the gaps. Currently, combining top-down and bottom-up tandem mass spectra was done in a greedy fashion. In this chapter, we investigate this method by using sequence alignment instead of using spectrum graph [68] to fill the gaps with bottom-up tandem mass spectra.

However, these methods are likely to increase the coverage of a whole protein, it is unlikely that we can always obtain a complete protein sequence even by combining top-down and bottom-up mass spectra. In other words, at the end of the sequencing steps, we are likely to see a scaffold with many gaps.

This chapter tries to construct a complete protein sequence with two distinct steps: constructing a scaffold from the top-down data and shrinking gaps in the scaffold by using bottom-up data. In the next chapter we investigate filling remaining gaps in the scaffold with respect to the target protein's homologous protein. The end result is a complete, or at least nearly complete protein sequence, which should be the same or similar to the target protein.

Preliminaries

A protein is a sequence of amino acids that performs some biological function. A homologous protein is a protein that is structurally similar by three

dimensional metric to some reference protein. It oftentimes can be inferred that these homologous proteins have a common ancestor. A protein is represented as a sequence of letters, with the alphabet being 20 amino acids, i.e., $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$. The amino acids are of various residue masses (Table 2.1). An amino acid is the building block for constructing a protein. The median length for eukaryote proteins is around 400 amino acids. A peptide is a subsequence within a protein. For example, $AYGY$ is a peptide within the protein $AYGYYYAT$. A contig is a longer substring within a protein. Top-down tandem mass spectra are generated from the intact protein using high-resolution mass spectrometers, which can be used as scaffold to help increase coverage and accuracy in *de novo* protein sequencing. Bottom-up tandem mass spectra are analyzed to generate short peptides.

Given two protein sequences $\mathcal{P} = \langle p_1, p_2, ..., p_n \rangle$ and $\mathcal{Q} = \langle q_1, q_2, ..., q_n \rangle$, (p_i, q_i) forms a *matched pair* if p_i and q_i are the same amino acid. We use $\mathcal{P}[i]$ to represent p_i, $\mathcal{P}[i, j]$ to represent the substring $\langle p_i, ..., p_j \rangle$. The length of $\mathcal{P}$ is denoted as $|\mathcal{P}|$, which is n here. We use $m(\mathcal{P}, \mathcal{Q})$ to denote the corresponding number of matched pairs. Let T_1 and T_2 denote two sets of protein sequences of the same length, we use $m(T_1, T_2)$ to denote the maximum number of matched pairs $m(\mathcal{P}, \mathcal{Q})$, for any $\mathcal{P} \in T_1, \mathcal{Q} \in T_2$. This definition also holds when $\mathcal{P}$ and $\mathcal{Q}$ are aligned from initial protein sequences, possibly of different lengths, using the standard sequence alignment algorithm by Needleman and Wunsch [78], i.e., when $\mathcal{P}$ and $\mathcal{Q}$ contain $\star$ (the gap, or space) letters. (For the ease of presentation we use $\star$ instead of $-$, as the latter is used as subtraction as well.) Note that a $\star$ does not form a matched pair with any amino acid.

Table 2.1: The residue masses (in Da), rounded to integers, of 20 amino acids.

Amino acid	Residue mass (Da)
A	71
C	103
D	115
E	129
F	147
G	57
H	137
I	113
K	128
L	113
M	131
N	114
P	97
Q	128
R	156
S	87
T	101
V	99
W	186
Y	163

12

Experimental Dataset

In this chapter, we use the *light chain of alemtuzumab (MabCampath)* data set. For this dataset, top-down and bottom-up mass spectra are created as in [68].

The initial scaffold construction uses top-down mass spectra (MS) data that has been preprocessed into a format that consists of mass-intensity pairs (pair of real numbers) after the process of converting raw spectra to prefix residue mass (PRM) spectra and the procedure of spectral selection and merging as shown in [68]. We do not need to consider suffix residue masses and several types of ions in PRM spectra. The mass field for each sample contains the mass (in Da) starting from the reading location to the start of the protein. The intensity field of each sample relates to how accurate the reading is likely to be, which has a directly proportional relationship. It is important to note that since we have the mass of readings from the start of the protein, we can determine the locations of the contigs that will be constructed from this data. The output representation after analyzing the top-down data is a scaffold containing several contigs and may have some gaps between contigs.

All bottom-up mass spectra are analyzed by PEAKS [71] to generate bottom-up peptides for the target protein. This dataset comprises of a large lists containing peptides which are the result of digestion of the target protein. We only keep the bottom-up peptides whose ALC score[1] is no less than 50%. Unfortunately, there is no information about the location of the peptides in the target protein, so we use alignment algorithms to place them within our scaffold. More technical details of generating processed data can be found in the paper [68]. Many short peptides are generated from PEAKS to use for alignment with top-down scaffold.

[1] ALC score: An average local confidence score for each bottom-up mass spectrum from software named PEAKS [71].

Top-Down Scaffold Construction

We use the De Bruijn graph technique for the scaffold construction, which was also applied in [100]. The goal of this step is to produce continuous, non-overlapping strings of amino acids from a PRM dataset, which were handled in [68]. Since the relative locations of these continuous regions, or contigs, are known, they can be used as a scaffold to align with the peptides in the bottom-up dataset.

The input to this procedure is a list of PRM mass-intensity pairs, ordered by increasing mass. The mass value is used to calculate potential amino acids and their locations in the range of the dataset. The intensity value comes from the raw MS data, and is used to score potential sequences.

To start the construction process using De Bruijn graph, a list of all possible k-mers needs to be generated from the PRM dataset. For a given data point, $p_i, p_j \in \mathbb{R}$, in the ordered PRM list, an amino acid exists between p_i and p_j iff $p_j - p_i \in A_{mass}$, where $i < j$ and A_{mass} is the set of the 20 amino acid masses. By repeating this lookahead k times, the set of possible k-mers from the datum p_i is obtained. The set of all possible k-mers for the dataset can then be generated by repeating this procedure for all $p \in PRM$ and collecting the results in a list. In addition to collecting the sequences of the k-mers, their relative start locations must also be stored. This will allow our functions in later steps to uniquely identify k-mers at separate locations, even if the composition of their sequences is identical.

We generate the list of all k-mers, K, and each k-mer $\in K$ is represented as the node of a De Bruijn graph. For every pair of $k_i, k_j \in K$, generate a directed edge from k_i to k_j if the following two properties hold:

1. $tail(k_i) = init(k_j)$, where $tail(k_i)$ is the peptide k_i without it's first element, and $init(k_j)$ is the peptide k_j without it's last element.

2. $start(k_i) + mass(first(k_i)) = start(k_j)$, where $start(k_i)$ is the starting location of the k-mer k_i, $mass(first(k_i))$ is the mass of the first amino acid in k_i's sequence, and $start(k_j)$ is the starting location of the k-mer k_j.

Table 2.2: Alemtuzumab top-down scaffold results for 4-mer with different epsilon (in Da).

epsilon	nodes	edges	paths	number of contigs	% coverage
0.01	58	44	15	9	34.11
0.02	103	86	35	11	47.20
0.03	163	147	65	12	55.14
0.04	234	218	104	11	55.61
0.08	760	988	480	7	67.76
0.12	859	1137	574	5	67.76
0.16	911	1201	534	4	64.49
0.20	944	1251	532	4	64.49
1.0	1473	2256	812	3	63.55

If both of these properties are satisfied, then there is a $k-1$ overlap between the two k-mers, and a directed edge between every pairs of k-mers exists in the De Bruijn graph for the top-down data. This pairwise comparison between all k-mers in K means that the entire De Bruijn graph can be constructed in $O(n^2)$ time, where n is the number of k-mers. It is important to note that the resulting De Bruijn graph may not be a singular connected component, and is more likely to be comprised of many connected components corresponding to regions within the top-down data that contain one or more possible contigs that have overlapping subsequences.

Once constructed, a path from a starting node which is corresponding to first constructed k-mer to an ending node which is corresponding to last constructed k-mer is chosen for each connected component in the De Bruijn graph. The weight of a given path is calculated as the sum of the intensities for each node within the path. The path that has the highest weight is chosen to be the representative contig for that set of paths.

In the event that some contig C_i overlaps with a portion of another contig C_j, the contig with the highest weight is kept, and the contig with the lower weight is dropped from the resulting list. The final step is to remove any conflicts from the resulting list of contigs. At this point, the scaffold generated from the top-down data is complete. The following Table 2.2 illustrate the experimental results for constructed De Bruijn graph using *light chain of alemtuzumab* data.

Extending the Top-Down Scaffold with Bottom-Up Peptides

We construct a same overlap graph as used in the paper [89]. Each peptide, including top-down contigs and bottom-up peptides, represents a node in the graph. The overlap graph construction used for the extension of the scaffold in this chapter has two degrees of freedom: (1) a minimum percentage of matching amino acids ξ and (2) a minimum mass difference within the overlapping region $\theta = \pm 1$ Da. If both of these thresholds are met, then the peptide b_i is accepted as a match to another peptide b_j or contig c_j in the scaffold. We add an edge (i, j) between these two peptides and each edge has a weight of $y_{i,j}$, where $y_{i,j}$ is the mass of non-overlapping section of the peptide. More specifically, $y_{i,j}$ is the mass of the suffix of b_j obtained by removing the overlap with b_i if it starts aligning from the tail of the previous peptide, or $y_{i,j}$ is the mass of the prefix of b_j obtained by removing the overlap with b_i if it starts aligning from the head of the previous peptide. If some contig in the scaffold or

peptides in the bottom up are totally covered by other peptides, the weight of edge between these are set to 0. So the problem becomes finding a path such that the weight of path is equal to the gap's mass within error $\pm 1Da$.

***De novo* protein sequencing by combining top-down and bottom-up data problem**

Input: Given a directed graph $G = (V, E)$, a weight on every edge, and its origin node s and destination node t.

Question: Find a path whose total weight is equal to the gap's mass within error $\pm 1Da$ between every consecutive contig in the top-down scaffold.

This problem can be constructed as the gap filling as exact path length problem which was also proved to be NP-complete presented in [89]. Here we explain the idea of NP-completeness proof of *de novo* protein sequencing by combining top-down and bottom-up data problem. The problem obviously belongs to NP. We next show that even the special case with one gap is NP-hard. To prove that the problem to be NP-hard is reduced from the subset sum problem. Given an instance of the subset sum problem with a set of integers $\{x_1, x_2, \ldots, x_n\}$, an integer X, and the problem is to decide whether a subset of elements sum to X. We construct a directed graph with the number of $2n + 1$ nodes and $3n$ edges as instance of the protein sequencing problem as shown in Figure 2.1. Each node represent one peptide. We add an edge between every overlapping peptides with the weight of 0, 0 and x_i, where x_i is the mass of non-overlapping section of the peptide. If some peptide is wholly covered by another peptide, we define $x_i = 0$. In this case, we have a solution of subset sum problem if and only if a path can be chosen with the total weight of X.

This problem could be solved in pseudo-polynomial dynamic programming algorithm presented in the paper [89]. But when there is a bottom-up peptide that has no overlap with other peptides from its both sides, the given dynamic programming

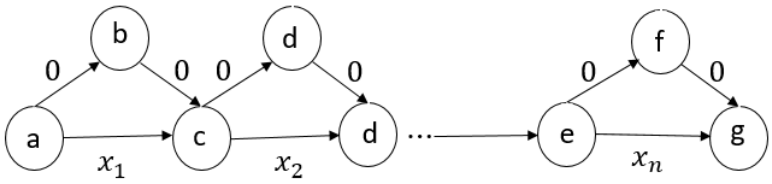


Figure 2.1: Illustration for the reduction from the subset sum instance.

algorithm fails to return any result. Considering this, we use an efficient heuristic algorithm to solve this problem. We always select the bottom-up peptides which have the longest suffix or prefix overlap with the top-down scaffold, and the process is repeated until we can not get any more bottom-up peptides to align with.

A Heuristic Approach

Once the initial scaffold is constructed, it can then be extended using the bottom-up peptides. The procedures that are used in this scaffold extension are inspired by techniques used in [68], but instead of constructing a spectral graph from combined top-down and shifted bottom-up PRM spectra, sequence alignment between the constructed scaffold and bottom-up peptides returned from PEAKS [71] are used.

As we mentioned before, due to the lacking of fragment ions in bottom-up or top-down mass spectra, the given dynamic programing may fail to return any path that satisfies the mass matching condition. So we use the following heuristic algorithm to extend the top-down as much as possible. In order to improve performance by limiting the number of peptides to align, we filter out members of bottom-up peptides whose ALC score is no less than 50%. We attempt to align bottom-up peptide b_i to each contig, $C_i \in S$, and keep track of the best alignment. The possible alignment options are at the head of C_i, entirely subsumed by b_i, or at the tail of C_i. We choose longest suffix or prefix overlap with top-down from the candidate peptides and then

we extend the contigs with b_i and remove b_i from B. We repeat the same procedure on the updated scaffold until no more contigs can be extended. This means that alignment options have been exhausted.

Given a top-down scaffold, S, along with a list of bottom-up peptides, B, an extended scaffold can be constructed using the following procedure:

We start from the first contig of the top-down scaffold and use bottom-up peptides to extend the top-down scaffold using a greedy method. We always choose the longest aligned peptide from the bottom-up candidates and repeat this procedure until the extended bottom-up peptide hits the head of the second top-down scaffold or there is no peptide that can be chosen to extend. We apply the same procedure at the head of a top-down scaffold or at the tail of a top-down scaffold. When some lists of bottom-up peptides cover the whole contig in the top-down scaffold, we choose the longest one. We give the detailed algorithm as follows.

1. Align bottom-up peptides with contigs in the top-down scaffold from left to right and choose the longest alignment within overlapping region from the candidates for each contig $C_i \in S$.

2. Extend the top-down scaffold with the chosen peptide.

3. Repeat step 1 until no more contigs in the top-down scaffold can be extended.

4. Return the final scaffold.

Experimental Results

Top-Down Scaffold

The top-down De Bruijn graph construction is tested at varying levels of data resolution, ranging from 1 to 0.001 Da with the error values ranging from 0 to 6

times the resolution. Statistics for the resulting De Bruijn graph can be seen in Table 2.2. For a resolution of 0.1 Da, and an error value of +/- 0.4 Da, we achieve 40.6% coverage of the target protein. The resulting De Bruijn graph can be seen in Figure 2.2. It is also important to note that while the number of nodes and edges remain the same for many of the variations of the parameters, the accuracy of obtaining the correct amino acids is not constant. The constructions that have a larger acceptable error could choose many incorrect amino acids in the contigs of the scaffold.

```
target:  DIQMTQSPSSLSASVGDRVTITCKASQNIDKYLNWYQQKPGKAPKLLIYNTNNLQTGVPSRFSGSGSGTDFT
top-down:                         ASQNLDQYLNWYQQQPGQAPQLLLYNTNNLQTGVPSR

target:  FTISSLQPEDIATYYCLQHISRPRTFGQGTKVEIKRTVAAPSVFIFPPSDEQLKSGTASVVCLLNNFYPREA
top-down:                                     SLAAPSVFLFPPSDEQLQSGTASVV

target:  KVQWKVDNALQSGNSQESVTEQDSKDSTYSLSSTLTLSKADYEKHKVYACEVTHQGLSSPVTKSFNRGEC
top-down:    QVDNALQSGNSQ                      SQADYEQHQVYAC
```

Figure 2.2: Three lines of target in this figure denotes the whole target protein sequence with 214 amino acids which is supposed to be constructed and top-down represents our constructed top-down scaffold containing four contigs using De Bruijn graph. Note that the amino acid I has the same mass to the amino acid L. So we convert all I to L in our results.

Bottom-Up Alignment

When we run our greedy algorithm to select bottom-up peptides, we get four bottom-up peptides to align with the top-down at the first iteration and get two bottom-up peptides at the second iteration. The resulting bottom-up peptides alignment at the first iteration can be seen in Figure 2.3. The resulting bottom-up peptides alignment at the second iteration can be seen in Figure 2.4.

```
target:   DIQMTQSPSSLSASVGDRVTITCKASQNIDKYLNWYQQKPGKAPKLLIYNTNNLQTGVPSRFSGSGSGTDFT
top-down:                         ASQNLDQYLNWYQQQPGQAPQLLLYNTNNLQTGVPSR
bottom-up:            SASVDRRTLTCKASQNLDKYLNW

target:   FTISSLQPEDIATYYCLQHISRPRTFGQGTKVEIKRTVAAPSVFIFPPSDEQLKSGTASVVCLLNNFYPREA
top-down:                                      SLAAPSVFLFPPSDEQLQSGTASVV
bottom-up:                                     SLAAPSVFLFPPSDEQLQSGTASVVCLLNNFYPR

target:   KVQWKVDNALQSGNSQESVTEQDSKDSTYSLSSTLTLSKADYEKHKVYACEVTHQGLSSPVTKSFNRGEC
top-down:     QVDNALQSGNSQ                      SQADYEQHQVYAC
bottom-up:    KVDANLQSGNSQE                                 ACEVTHQGLSSPVTKSFCHVGY
```

Figure 2.3: Illustration of bottom-up peptides alignment with top-down contigs at the first iteration. The other parameters, such as target and top-down are the same as in Figure 2.2.

The Constructed Scaffold

After aligning bottom-up peptides with the top-down contigs, we obtain a scaffold with three contigs and three gaps. Initial results for our method on the *light chain of Alemtuzumab* dataset yield 80.6% similarity rate with the homologous protein sequence, and 81.78% similarity rate with the target protein sequence. The resulting constructed scaffold can be seen in Figure 2.5.

```
target:   DIQMTQSPSSLSASVGDRVTITCKASQNIDKYLNWYQQKPGKAPKLLIYNTNNLQTGVPSRFSGSGSGTDFT
top-down:                         ASQNLDQYLNWYQQQPGQAPQLLLYNTNNLQTGVPSR
bottom-up:            SASVDRRTLTCKASQNLDKYLNW
bottom-up: QLDMTQSPSSLSASVGDRVTLTCK

target:   FTISSLQPEDIATYYCLQHISRPRTFGQGTKVEIKRTVAAPSVFIFPPSDEQLKSGTASVVCLLNNFYPREA
top-down:                                      SLAAPSVFLFPPSDEQLQSGTASVV
bottom-up:                                     SLAAPSVFLFPPSDEQLQSGTASVVCLLNNFYPR

target:   KVQWKVDNALQSGNSQESVTEQDSKDSTYSLSSTLTLSKADYEKHKVYACEVTHQGLSSPVTKSFNRGEC
top-down:     QVDNALQSGNSQ                      SQADYEQHQVYAC
bottom-up:    KVDANLQSGNSQE                                 ACEVTHQGLSSPVTKSFCHVGY
bottom-up:     VDNALQSNGSQESVTEQDSKDSTYSLSSTLTLSK
```

Figure 2.4: Illustration of bottom-up alignments with updated top-down contigs at the second iteration. The other parameters, such as target and top-down are the same as in Figure 2.2.

Concluding Remarks

In this chapter, we apply De Bruijn graph to construct the top-down scaffold, then use bottom-up peptides to align with top-down contigs to extend the scaffold. We test our method on one real dataset *light chain of Alemtuzumab*. While the breadth of the experiment is far too limited to claim that the results are conclusive, this method of protein construction is shown to be a promising for future research. From the experiment, we find that even combining the top-down and bottom-up spectra, it is still hard to obtain the whole complete protein sequence. In the next chapter, we discuss the protein scaffold filling problems in which use the homologous protein sequence as a reference to fill the remaining gaps to get the complete (or almost complete) protein sequence.

```
target:   DIQMTQSPSSLSASVGDRVTITCKASQNIDKYLNWYQQKPGKAPKLLIYNTNNLQTGVPSRFSGSGSGTDFT
scaffold:  QLDMTQSPSSLSASVDRRTLTCKASQNLDQYLNWYQQQPGQAPQLLLYNTNNLQTGVPSR

target:   FTISSLQPEDIATYYCLQHISRPRTFGQGTKVEIKRTVAAPSVFIFPPSDEQLKSGTASVVCLLNNFYPREA
scaffold:                                    SLAAPSVFLFPPSDEQLQSGTASVVCLLNNFYPR

target:   KVQWKVDNALQSGNSQESVTEQDSKDSTYSLSSTLTLSKADYEKHKVYACEVTHQGLSSPVTKSFNRGEC
scaffold:     QVDNALQSGNSQESVTEQDSKDSTYSLSSTLTLSQADYEQHQVYACEVTHQGLSSPVTKSFCHVGY
```

Figure 2.5: Three lines of target denotes the whole target protein sequence which is supposed to be constructed and three lines of scaffold represents the final scaffold after aligning bottom-up peptides with the top-down contigs.

22

FILLING A PROTEIN SCAFFOLD WITH A REFERENCE

As shown in the previous chapter, in mass spectrometry-based *de novo* protein sequencing, it is hard to complete the sequence of the whole protein. Motivated by this, we study the (one-sided) problem of filling a protein *scaffold* $\mathcal{S}$ with some missing amino acids, given a sequence of contigs none of which is allowed to be altered, with respect to a complete reference protein $\mathcal{P}$ of length n, such that the BLOSUM62 score between $\mathcal{P}$ and the filled sequence $\mathcal{S}'$ is maximized. We show that this problem is polynomial-time solvable in $O(n^{26})$ time. We define the quality of contigs in the scaffold as minimum percentage number of matched amino acids with corresponding homologous protein segment satisfy given threshold ξ. If the contig satisfy given ξ, we call it good quality contig. We also consider the case when the contigs are not of high quality and they are concatenated into an (incomplete) sequence $\mathcal{I}$, where the missing amino acids can be inserted anywhere in $\mathcal{I}$ to obtain $\mathcal{I}'$, such that the BLOSUM62 score between $\mathcal{P}$ and $\mathcal{I}'$ is maximized. We show that this problem is polynomial-time solvable in $O(n^{22})$ time. Due to the high time complexity, both of these algorithms are impractical, we hence present several algorithms based on greedy and local search, trying to solve the problems practically.

Related Work

In mass spectrometry-based *de novo* protein sequencing, it is hard to complete the sequence of the whole protein [10]. An incomplete sequence contains one or several contigs, each is a protein segment. When the order of the contigs is known (e.g., ordered contigs reported by top-down mass spectrometry-based *de novo* protein sequencing [68]), these contigs are called a *scaffold*. In many applications, it is more desirable to obtain complete protein sequences. We comment that a similar

phenomena occurs in the sequencing of genomes, firstly noticed by Mũnoz *et al.* [76], and has resulted in a series of interesting algorithmic studies [16, 56, 57, 66, 67] (also, see [104] for a recent survey).

Hence, a natural combinatorial problem is to fill the missing amino acids into scaffolds. As one must find a biologically meaningful way of filling scaffolds, it makes sense to use a complete homologous protein sequence as a reference. Here we consider two kinds of scaffolds. One kind is lists of contigs, which are computed with a good quality and should not be altered. Throughout this chapter, we use $\mathcal{S}$ to denote such a scaffold, composed of contigs $C_1, C_2, ..., C_m$. The other kind is sequences, which usually arise when contigs are computed without a very high quality — in which case it would be unrealistic not to alter the contigs. We then simply concatenate the contigs into $\mathcal{I}$, the incomplete sequence. We try to carry out the idea of filling a scaffold (resp. sequence) $\mathcal{S}$ (resp. $\mathcal{I}$) to have $\mathcal{S}'$ (resp. $\mathcal{I}'$), such that the similarity between $\mathcal{S}'$ (resp. $\mathcal{I}'$) and a given (complete) protein $\mathcal{P}$ is maximized, which are called *Contig-Preserving Protein Scaffold Filling* (CP-PSF for short) and *Protein Scaffold Filling* (PSF for short) respectively.

The BLOSUM62 matrix [53] is the most popular similarity measure to align a pair of protein sequences. (The matrix can be found in the appendix.) We will use BLOSUM62 to measure the accuracy of our scaffold filling algorithms. However, for empirical results it is not very indicative to interpret BLOSUM62 scores for measuring the similarity between sequences. The reason is that we can not see clearly the similarity between two proteins based on the BLOSUM62 score. Therefore, we will use the number of *matched pairs* as a similarity measure (this can be considered a rounded BLOSUM62 matrix: if the score between two amino acids in BLOSUM62 is less than 4, then set the score as 0; otherwise, set the score as 1). Of course, when a space (denoted as a $\star$ in this chapter) is used, with BLOSUM62 we get a negative

score; and this is not reflected in the number of matched pairs.

Firstly, we show that the PSF problem is solvable in $O(n^{22})$ time, where n is the number of missing amino acids. As this method is impractical, we make use of the standard Needleman-Wunsch algorithm to align $\mathcal{I}$ with $\mathcal{P}$, we then design two heuristic algorithms, based on greedy and local search, to insert the amino acids in $X = \mathcal{P} - \mathcal{I}$ into $\mathcal{P}$. We test our algorithm using some real datasets (i.e., 4 chains from two antibody proteins and 2 chains from two mammalian proteins). Secondly, for the CP-PSF problem of filling $\mathcal{S}$, we also present a polynomial time solution which takes $O(n^{26})$ time. We then study the problem of aligning the contigs in $\mathcal{S}$ to $\mathcal{P}$ to achieve the maximum BLOSUM62 score, and show that it is solvable in $O(n^5)$ time. Then, based on this, we design and implement two algorithms using greedy and local search methods. We test our algorithms on the same datasets.

We comment that our problems are related to but different from the *one-sided* scaffold filling problem for genomes (with gene repetitions), the main difference is that the similarity measure between genomes is different from that between protein sequences. For protein sequences, the order of its amino acids is even more critical compared with genomes. Given a complete genome $\mathcal{G}$ and a genomic scaffold $\mathcal{H}$, the one-sided scaffold filling problem, i.e., filling $\mathcal{H}$ into $\mathcal{H}'$ such that the number of adjacencies between $\mathcal{G}$ and $\mathcal{H}'$ is maximized, is NP-hard [56, 57] and constant-factor approximation algorithms are known [66, 104].

Preliminaries

We first present some necessary definitions. We denote the set of 20 amino acids as $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$. A protein sequence $\mathcal{P}$ is a sequence over Σ. We also use $c(\mathcal{P})$ to denote the multiset of elements in $\mathcal{P}$. For example, $\mathcal{P} = \langle A, D, C, I, K, W, Y, C, I\rangle$ (or simply, $\mathcal{P}$ =ADCIKWYCI) with

$c(\mathcal{P}) = \{A, C, C, D, I, I, K, W, Y\}$. In *bottom-up* mass spectrometry based *de novo* protein sequencing, we first derive peptide sequences from tandem mass spectra of the target protein, then build up longer contigs by spectral or peptide assembly [9, 69]. For real datasets, the length of a peptide is typically between 5 and 40.

BLOSUM62 (B62 for short), coming from BLOck SUbstitution Matrix [53], is based on comparisons of multiply (locally) aligned ungapped segments corresponding to the most highly conserved regions (similar or identical sequences in proteins across different species, it keeps relatively unchanged during evolutionary process) of proteins in the Blocks database [85]. BLOSUM62 is a matrix to use for protein alignments. The number of 62 means the comparisons are based on ungapped sequence alignments with $> 62\%$ identity. BLOSUM62 is the default matrix for the standard protein-BLAST program. The integer values in BLOSUM62 vary from -4 to 11, e.g., $B62[W, D] = -4$ and $B62[W, W] = 11$. It should be noted that a score -8 is applied for introducing a $\star$, which is a restriction for using too many $\star$'s. The main reason to set a small score -8 for a aligned $\star$ is that we can avoid inserting too many $\star$'s in the sequence since our goal is to maximize the sum of aligned score between every amino acids between two protein sequences. In this chapter, we use B62 to measure the similarity of protein sequences.

A scaffold $\mathcal{S}$ is a list of contigs $C_1, C_2, ..., C_m$, where each contig is a sequence of amino acids. For example, $\mathcal{S} = \langle C_1, C_2, C_3 \rangle$, where $C_1 =$ AEFGIA, $C_2 =$ CDIKLNTVW, and $C_3 =$ PQAWYA. These contigs are usually computed with some weight constraint, say, the total weight of amino acids in C_1 is roughly $w(C_1) = w(A) + w(E) + w(F) + w(G) + w(I) + w(A) = 71 + 129 + 147 + 57 + 113 + 71 = 598$ Dalton. In fact, throughout this chapter, we do not alter the amino acids in a given contig at all. A sequence $\mathcal{I}$ is an incomplete protein sequence, i.e., with some unknown missing amino acids. This sequence is obtained usually when the contigs are not of

high quality, which means amino acids in the contigs do not have enough number of correct amino acids comparing to its length and we just concatenate C_i's to obtain a sequence. We are allowed to insert missing amino acids anywhere in $\mathcal{I}$.

Given a multiset X of amino acids, and a scaffold $\mathcal{S} = \langle C_1, ..., C_m \rangle$,$\mathcal{S} + X$ is the set of all protein sequences obtained by inserting the missing amino acids in X in between C_i's (i.e., C_i's are not altered).

In practice, if the total mass of the target protein is known, which can be measured by top-down mass spectrometry, we would know the total mass of missing amino acids. In this case, one way for handling this is to enumerate the sets of amino acids which sum to a certain mass — corresponding to that of the missing amino acids. Of course, when this mass is decently large we could have an exponential number of such sets. In this chapter, we only focus on a given set X of missing amino acids, which can be computed as the difference between the reference protein and given scaffold (or sequence).

Given two protein sequences $\mathcal{P}, \mathcal{Q}$, we use $B62(\mathcal{P}, \mathcal{Q})$ to denote the maximum BLOSUM62 score when aligning them. Let $\mathcal{Z}$ be a set of protein sequences. Then $B62(\mathcal{P}, z \in \mathcal{Z})$ is the maximum BLOSUM62 score when aligning $\mathcal{P}$ and any sequence $z \in \mathcal{Z}$. Given a protein sequence $\mathcal{I}$ and a multiset X of amino acids, we denote $\mathcal{I} + X$ as the set of all protein sequences obtained by filling all the amino acids in X into $\mathcal{I}$. We use $|X|$ to denote the size of the set X.

The contig-preserving protein scaffold filling (CP-PSF) problem is defined as follows.

Contig-Preserving Protein Scaffold Filling to Maximize the B62 Score (CP-PSF)

Input: a complete protein sequence $\mathcal{P}$, a protein scaffold $\mathcal{S} = \langle C_1, ..., C_m \rangle$, and a multiset X of amino acids.

Question: maximize the B62 score $B62(\mathcal{P}, \mathcal{S} + X)$.

For most of the practical instances we could assume that X is given as $X = c(\mathcal{P}) - \cup_i c(C_i)$. Note that we have no restriction on the length of $\mathcal{P}$ and the filled sequences, as with B62 we could use $\star$'s.

When the scaffold is a sequence $\mathcal{I}$, the problem can be simplified as follows.

Protein Scaffold Filling to Maximize the B62 Score (PSF)

Input: a complete protein sequence $\mathcal{P}$ with $|\mathcal{P}| = n$, an incomplete protein sequences $\mathcal{I}$ and a multiset of amino acids X.

Question: maximize the B62 score $B62(\mathcal{P}, \mathcal{I} + X)$.

Note that, again, for most of the practical instances we could assume that X is given as $X = c(\mathcal{P}) - c(\mathcal{I})$.

As solutions for PSF could be used as subroutines for CP-PSF, in the next section we first show that the PSF problem is polynomially solvable by giving a complex dynamic programming solution which runs in $O(n^{22})$ time. We then present two practical methods and show some empirical results.

Algorithms for PSF When not Allowing Gaps

We first show some property for PSF.

Lemma 1. *Suppose that k^* is the optimal solution value for PSF, then $k^* = k_1^* + k_2^*$ where k_1^* is the number of aligned pairs between the existing amino acids in $\mathcal{I}$ and those in $\mathcal{P}$, and k_2^* is the number of aligned pairs between the inserted amino acids and those in $\mathcal{P}$.*

Proof. The proof of this lemma follows from the definition of k^*. □

We next illustrate the ideas of our approximation algorithm. We try to design two polynomial time algorithms to compute feasible aligned pairs which are at least

k_1^* or k_2^* respectively. Then, the maximum of the two gives a factor-2 approximation.

Lemma 2. *Let $X^\star$ be the multiset of $|X|$ gap amino acid $\star$'s, where a $\star$ does not form an aligned pair with any amino acid. The problem of computing an alignment of $\mathcal{P}$ and $\mathcal{I}$ by inserting $|X|$ gap amino acid $\star$'s such that $ap(\mathcal{I} + X^\star, \mathcal{P})$ is maximized, is polynomially solvable.*

Proof. We first align $\mathcal{P}$ and $\mathcal{I}$, using the standard Needleman-Wunsch dynamic programming algorithm for sequence alignment where a pair of matched letters (in this case, amino acids) gives a score 1 and a pair of unmatched letters gives a score 0, with $\star$ treated as a space. Then, to form a feasible solution, we use a greedy method (i.e., filling the gaps resulting in a score 1 first) to insert the elements in X at all the $\star$'s positions to maximize the number of aligned pairs. □

Lemma 3. *The problem of computing an alignment of $\mathcal{P}$ and $\mathcal{I}$ by inserting X into $\mathcal{I}$ such that the number of aligned pair between the elements in X and those in $\mathcal{P}$ is maximized, is polynomially solvable.*

Proof. We will use a greedy method as follows. First we treat all amino acids in $\mathcal{I}$ as $\star$'s. Second, we scan $\mathcal{P}$ from left to right to find the first letter (amino acid) $\mathcal{P}[i] = x$ which is in X. Then, we insert x in $\mathcal{I}$ at location i. Finally, we update $X \leftarrow X - \{x\}$, $\mathcal{P} \leftarrow \mathcal{P}[i+1..]$, $\mathcal{I} \leftarrow \mathcal{I}[i+1..]$, and repeat this process until all the letters in X are inserted and matched. Finally, we just replace the subsequence made of $\star$'s by the initial $\mathcal{I}$ and that gives us a feasible solution. □

Theorem 1. *There is a factor-2 approximation for PSF.*

Proof. Our algorithm is simple: (1) use the algorithms in Lemma 2 and Lemma 3 to compute the corresponding aligned pairs, k_1 and k_2 respectively; (2) return $\max\{k_1, k_2\}$.

We show that this is a factor-2 approximation as follows. Let APP denote the approximation solution value. By the optimality of the algorithms in Lemma 2 and Lemma 3, $k_1 \geq k_1^*$ and $k_2 \geq k_2^*$. Therefore,

$$APP = \max\{k_1, k_2\} \geq \max\{k_1^*, k_2^*\} \geq \frac{k_1^* + k_2^*}{2} = \frac{k^*}{2}.$$

□

Algorithms and Empirical Results for PSF When Allowing Gaps

PSF is in P

We present a dynamic programming solution for solving the PSF problem, given $\mathcal{P}$, $\mathcal{I}$ and X. The objective is to insert X into $\mathcal{I}$ to obtain $\mathcal{I}'$ such that $B62(\mathcal{P}, \mathcal{I}')$ is maximized.

The idea of our algorithm is as follows. In an optimal solution, when scanning $\mathcal{P}$ from left to right, there are 4 cases: (1) $\mathcal{P}[i]$ is aligned to an inserted amino acid in X, (2) $\mathcal{P}[i]$ is aligned to some amino acid $\mathcal{I}[j]$, (3) $\mathcal{P}[i]$ is inserted with a $\star$ and it is aligned to some amino acid $\mathcal{I}[j]$, and (4) $\mathcal{P}[i]$ is aligned to an inserted $\star$ in $\mathcal{I}[j]$. To denote the set X (or any of its subset), we use a 20-dimension vector $X = \langle a_1, a_2, ..., a_{20} \rangle$, where a_k is the number of amino acid k ($k = 1..20$). Define $|X| = a_1 + a_2 + \cdots + a_{20}$. It is clear that the number of distinct subsets of X is bounded by

$$\begin{aligned} & (a_1 + 1) \times (a_2 + 1) \times \cdots \times (a_{20} + 1) \\ \leq\ & (\frac{a_1 + a_2 + \cdots + a_{20} + 20}{20})^{20} \\ =\ & (\frac{|X| + 20}{20})^{20} \\ =\ & O(n^{20}). \end{aligned}$$

$$
S[i{+}1, j{+}1, t] = \max \begin{cases}
S[i, j, t] + B62(\mathcal{P}[i+1], \mathcal{I}[j+1]), \\
//\mathcal{P}[i+1] \text{ matches } \mathcal{I}[j+1] \\
\text{(case 1)} \\
\max_k \{S[i, j, t - X_k] + B62(\mathcal{P}[i+1], X_k)\}, \\
//\mathcal{P}[i+1] \text{ matches } X_k \text{ (inserted at } \mathcal{I}[j+1]) \\
\text{(case 2)} \\
S[i+1, j, t] + B62(\mathcal{P}[i+1], \star), \\
//\mathcal{P}[i+1] \text{ matches } \star \text{ (inserted at } \mathcal{I}[j+1]) \\
\text{(case 3)} \\
S[i, j+1, t] + B62(\star, \mathcal{I}[j+1]), \\
//\star \text{ (inserted at } \mathcal{P}[i+1] \text{ matches } \mathcal{I}[j+1]) \text{)} \\
\text{(case 4)} \\
S[i, j, t - X_k] + B62(\star, X_k), \\
// \star \text{ (inserted at } \mathcal{P}[i+1] \text{ matches } X_k \text{ (inserted at } \mathcal{I}[j+1])) \\
\text{(case 5)}
\end{cases}
$$

Let t be a subset of X, represented as a 20-dimension vector. Define $S[i, j, t]$ as the maximum B62 score when aligning $\mathcal{P}[1..i]$ with $\mathcal{I}[1..j]$, where $t \subseteq X$ has been inserted into $\mathcal{I}$ $[1..j]$ (to achieve the score). Let X_k be one of the (nonempty) k-th amino acid in X, for $k = 1..20$. The recurrence relation for updating $S[i, j, t]$ is as follows.

The base cases, $S[i, 0, t]'s$, can be computed by a greedy method in $O(n^{21})$ time and space. This is because we have $O(n^{20})$ subset $t's$ of X. The optimal solution value is $S[n, m, X]$.

Theorem 2. *PSF can be solved in $O(n^{22})$ time and space.*

Even though PSF is polynomially solvable, the high running time makes the solution practically infeasible to implement. We next present two practical methods.

Practical Algorithms for PSF

In the following, we assume that a complete protein sequence $\mathcal{P}$ is always given. Our first algorithm is called *Align+Greedy*: We first align $\mathcal{I}$ to $\mathcal{P}$ using the standard Needleman-Wunsch algorithm, and then we insert the missing amino acids at the $\star$ positions in the aligned $\mathcal{I}$ with a greedy method (i.e., according to the maximum of their B62 scores).

1: Align $\mathcal{I}$ with $\mathcal{P}$ to obtain $\mathcal{I}_1$ with the maximum B62 score.
2: Find a $\star$ position in the aligned $\mathcal{I}_1$ such that inserting an amino acid in X would incur the maximum B62 score among all $\star$ positions.
3: Repeat Step 2 until all elements in X are inserted into $\mathcal{I}_1$.
4: Return the filled $\mathcal{I}_1$ as $\mathcal{I}'$, with the total alignment score between $\mathcal{I}'$ and $\mathcal{P}$ being $b62(\mathcal{P}, \mathcal{I}, X)$.

Algorithm 3.1: *Align+Greedy($\mathcal{P}$,$\mathcal{I}$,X)*

An improved version of our first algorithm is based on a local search method. The idea is that if Algorithm 3.1 does not give us the best return, there must be a way to locally update the solution to have a better result. The algorithm is as follows.

We use two kinds of datasets to test the effectiveness of our algorithms, and they are from antibody proteins and mammalian proteins respectively. Our antibody dataset is based on `MabCampath` (or Alemtuzumab) and `Humira` (or Adalimumab), which are two similar antibody proteins. Both of them contain a light chain and a heavy chain, the lengths for them are 214 and 449 for `MabCampath`, and 214 and 453 for `Humira` respectively. The pairwise alignments of the two light chains and

1: Compute $b62(\mathcal{P}, \mathcal{I}, X)$ using Algorithm 3.1. Assign $U \leftarrow b62(\mathcal{P}, \mathcal{I}, X)$.

2: Insert an amino acid $x \in X$ into a position i of $\mathcal{I}$ to obtain $\mathcal{I}(i, x)$ such that the score $b62(\mathcal{P}, \mathcal{I}(i, x), X - \{x\})$ is maximum, for all $x \in X$ and for all $i \in [0, |\mathcal{I}|]$.

3: Repeat Step 2 until all elements in X are inserted into $\mathcal{I}_1$.

4: Return the filled $\mathcal{I}_1$ as $\mathcal{I}'$, with the total alignment score between $\mathcal{I}'$ and $\mathcal{P}$ being $b62(\mathcal{P}, \mathcal{I}, X)$.

5: Run Algorithm 3.1 to obtain $b62(\mathcal{P}, \mathcal{I}+x, X-\{x\})$. If $b62(\mathcal{P}, \mathcal{I}+x, X-\{x\}) \leq U$, then return the solution $\mathcal{I}''$ incurring U; else update $U \leftarrow b62(\mathcal{P}, \mathcal{I}(i, x), X-\{x\})$, $\mathcal{I} \leftarrow \mathcal{I}(i, x)$, $X \leftarrow X - \{x\}$, and repeat Step 2.

Algorithm 3.2: *LocalSearch($\mathcal{P}$,$\mathcal{I}$,X)*

two heavy chains display 91.1% and 86.6% identity respectively. For each protein sequence we compute a set of peptides from bottom up tandem mass spectra using PEAKS [71,72], which is a de novo peptide sequencing software tool. Then we simply select a maximal set of disjoint peptides for each protein sequence. For the light chain of `MabCampath` (`M-L` for short): we have two (disjoint) peptides of lengths 12 and 19. For the heavy chain of `MabCampath` (`M-H` for short): we have eight (disjoint) peptides of lengths 9, 7, 13, 12, 14, 15, 12 and 19. For the heavy chain of `Humira` (`H-H` for short): we have six (disjoint) peptides of lengths 7, 7, 9, 9, 10 and 8. For the light chain of `Humira` (`H-L` for short), PEAKS is not able to obtain any peptides of decent quality. So we will only use `Humira-L` for reference purpose. Due to that the amino acids I and L have the same mass, in the peptides and all our comparisons, all I's have been converted to L's.

Our mammalian dataset is based on two homologous protein sequences, which are carbonic anhydrase 2 (CAH2 for short) for *Bos taurus* and Human respectively. The length of these CAH2 proteins are 258 and 260 respectively. The number of

aligned pairs between these two CAH2 proteins is 206, indicating an identity close to 80%. All the datasets can be found in `www.cs.montana.edu/~qingge.letu/ResearchData.html`. Our code is written in Matlab and Java.

Simulation Results

We run Algorithm 3.1 and 3.2 to obtain some empirical results as follows. As the real scaffold datasets we have obtained so far are either too small (like containing only two contigs), or possibly interleaved (i.e., two contigs could have overlaps), we use the heavy chain of `MabCampath`, `MabCampath-H`, of length 449, to obtain some synthetic datasets. The heavy chain of `Humira`, `Humira-H`, of length 453, is our reference $\mathcal{P}$. There are two ways to construct the synthetic data.

Method 1: In this case, we first delete $x\%$ of amino acids in `MabCampath-H`, then we obtain 8 contigs (each is a substring of `MabCampath-H` whose contents are not altered), and finally we shuffle y pairs of amino acids in these contigs. Therefore, the length of contigs stay the same throughout the experiments. We then run Algorithm 3.1 and 3.2 to obtain the scaffolds and measure the actual number of reconstructed amino acids in `MabCampath-H`. The results are summarized in the following Table 3.1.

From Table 3.1, it can be seen that the algorithms work really well. But there is no improvement for Algorithm 3.2 over Algorithm 3.1. The reason might be due to that the contigs obtained are too good (in practice, a contig could be more erroneous and it definitely could contain errors caused by shuffling). This prompts us for the second method to construct $\mathcal{S}$.

Method 2: We first construct the 8 contigs from `MabCampath-H` in the same way as in Table 3.1. Then, we randomly choose two of the 8 contigs and append some random amino acids in X to the front/end of these two to obtain two new contigs. The

rational is that some contigs might contain errors not just due to shuffling. We then run Algorithm 3.1 and Algorithm 3.2 on these new datasets. Again, we measure the actual number of reconstructed amino acids in `MabCampath-H` by these two algorithms respectively. The results are summarized in the following Table 3.2.

Table 3.1: The average number of aligned pairs computed between `MabCampath-H` and the filled scaffold $\mathcal{S}'$ (by Algorithm 3.1), and between `MabCampath-H` and the filled scaffold $\mathcal{S}''$ (by Algorithm 3.2), over ten tries. Here $\mathcal{S}$ is constructed from `MabCampath-H` by deleting $x\%$ of amino acids to have 8 contigs, followed with the shuffling of y pairs of amino acids.

	x=30%	x=25%	x=20%	x=15%	x=10%
y=15	423.8/423.8	424.2/424.2	422.9/422.9	420.1/420.1	423.7/423.7
y=20	415.5/415.5	416.4/416.4	413.2/413.2	412.7/412.7	416.1/416.1
y=25	404.3/404.3	407.3/407.3	404.1/404.1	405.7/405.7	406.1/406.1
y=30	395.1/395.1	402.4/402.4	401.7/401.7	397.5/397.5	398.3/398.3

Table 3.2: The average number of aligned pairs computed between `MabCampath-H` and the filled scaffold $\mathcal{S}'$ (by Algorithm 3.1), and between `MabCampath-H` and the filled scaffold $\mathcal{S}''$ (by Algorithm 3.2), over ten tries. Here $\mathcal{S}$ is constructed from `MabCampath-H` by deleting $x\%$ of amino acids to have 8 contigs, followed with the shuffling of y pairs of amino acids, and finally we pick two of the contigs and append randomly 2-15 amino acids in X to them.

	x=30%	x=25%	x=20%	x=15%	x=10%
y=15	403.1/411.3	406.7/410.3	404.1/407.2	401.3/403.6	413.3/416.1
y=20	390.4/397.7	391.5/396.6	388.9/392.7	388.6/390.7	402.2/405.3
y=25	387.1/395.1	378.3/384.2	379.7/384.3	379.9/384.1	390.1/392.7
y=30	375.7/384.4	367.2/374.8	372.5/376.8	374.9/377.5	385.8/387.5

Empirical Results for Antibody Proteins

Let $x \in \{$`M-H, H-H, M-L`$\}$ and let the corresponding references of x be `H-H`, `M-H`, and `H-L` respectively. For each x, we use Algorithm 3.1 to compute the filled sequence $\mathcal{I}'_x$, and we use Algorithm 3.2 to compute the filled sequence $\mathcal{I}''_x$. As the B62 scores are not very indicative, in the following we use the number of matched pairs resulting from the computed B62 scores. The empirical results are shown in the following two tables. Note that in most cases, Algorithm 3.2 produces slightly better results. Of course, that should be considered as normal. In all the tables, R, T, ℓ_R, ℓ_T represent the reference protein sequence, target protein sequence and their lengths respectively; moreover, the first column indicates the scaffold represented by x. Recall that $m(-,-)$ represents the *matched pairs* between two sequences.

Empirical Results for Mammalian Proteins

We use CAH2 of *Bos taurus* (CB for short) as a target sequence and simply select different disjoint peptides as our scaffold from its bottom-up spectra, which can be generated by PEAKS. Our idea is to use CAH2 of Human (CH for short) as a reference to insert the missing amino acids into the incomplete scaffold to obtain

Table 3.3: Empirical results for the three computed $\mathcal{I}'_x$ using Algorithm 3.1. In all the tables, **ref_length, tar_length** represents the length of the reference and target protein sequence respectively. mp denotes matched pairs between two amino acids.

	x=`MabCampath-H`	x=`Humira-H`	x=`MabCampath-L`
mp($\mathcal{I}'_x$,ref)	406	442	214
mp($\mathcal{I}'_x$,ref)/ref_length	406/453=89.62%	442/449=98.44%	214/214=100%
mp($\mathcal{I}'_x$,target)	355	385	195
mp($\mathcal{I}'_x$,target)/tar_length	355/449=79.06%	385/453=84.99%	195/214=91.12%

a new complete sequence. Then we try to measure the similarity between the target sequence and a newly computed complete sequence. For the similarity measure, we again use the number of matched pairs resulting from the computed B62 scores instead of directly using the B62 scores of its own.

We have three groups of datasets which consist of distinct numbers of different disjoint peptides. The first group contains two disjoint peptides of lengths 15 and 11. The second group contains three disjoint peptides of lengths 9, 9 and 7. The third group contains four disjoint peptides of lengths 15, 9, 8 and 9. Due to the fact that the amino acids I and L have the same mass, all I's have been changed to L in the

Table 3.4: Empirical results for the three computed $\mathcal{I}_x''$ using Algorithm 3.2. mp denotes matched pairs between two amino acids.

	x=MabCampath-H	x=Humira-H	x=MabCampath-L
mp($\mathcal{I}_x''$,ref)	415	442	214
mp($\mathcal{I}_x''$,ref)/ref_length	415/453=91.61%	442/449=98.44%	214/214=100%
mp($\mathcal{I}_x''$,target)	370	385	195
mp($\mathcal{I}_x''$,target)/tar_length	370/449=82.41%	385/453=84.99%	195/214=91.12%

Table 3.5: Empirical results for the three computed $\mathcal{I}_x'$ using Algorithm 3.1, with respect to the mammalian dataset. CBi ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. CH represents CAH2 of Human, which is the reference.

	x=CB2	x=CB3	x=CB4
mp($\mathcal{I}_x''$,ref)	258	252	241
mp($\mathcal{I}_x''$,ref)/ref_length	258/260=99.23%	252/260=96.92%	241/260=79.07%
mp($\mathcal{I}_x''$,target)	206	204	194
mp($\mathcal{I}_x''$,target)/tar_length	206/258=79.84%	204/258=79.07%	194/258=75.19%

Table 3.6: Empirical results for the three computed $\mathcal{I}''_x$ using Algorithm 3.2, with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference.

	x=`CB2`	x=`CB3`	x=`CB4`
mp($\mathcal{I}''_x$,ref)	258	254	247
mp($\mathcal{I}''_x$,ref)/ref_length	258/260=99.23%	254/260=97.69%	247/260=95.00%
mp($\mathcal{I}''_x$,target)	206	203	197
mp($\mathcal{I}''_x$,target)/tar_length	206/258=79.84%	203/258=78.68%	197/258=76.36%

peptides. We represent the three groups as `CB2, CB3` and `CB4` respectively.

Let x, $x \in \{$`CB2, CB3, CB4`$\}$ and let the corresponding references all be `CH`. we use Algorithm 3.1 to compute the filled sequence $\mathcal{I}'_x$, and we use Algorithm 3.2 to compute the filled sequence $\mathcal{I}''_x$. Again, recall that ℓ_R, ℓ_T represent the length of the reference and target protein sequence respectively. The empirical results are shown in the following two tables.

Algorithms and Empirical Results for CP-PSF

CP-PSF is in P

In this section, we first show that CP-PSF is also solvable in polynomial time. As the time complexity of this algorithm is too high (hence infeasible for implementation), we just sketch a solution without necessarily trying to obtain the best running time.

Define $B[i, j, k, \ell]$ as the maximum B62 score when the last element of $S_{\ell-1}$ is aligned with $\mathcal{P}[i]$, and S_ℓ is aligned with $\mathcal{P}[j..k]$. Define $C[i, j, k, \ell]$ as the set of amino acids inserted to obtain $B[i, j, k, \ell]$. Let $INS(\mathcal{P}[i + 1..j - 1], C[i, j, k, \ell])$ be the maximum B62 score obtained by inserting the amino acids in $C[i, j, k, \ell]$ to

align with the substring $\mathcal{P}[i+1..j-1]$. Apparently, $INS(-,-)$ can be computed using the dynamic programming algorithm summarized in Theorem 3. Note that $\star$'s could be inserted in $\mathcal{P}$ and in between the scaffolds in $\mathcal{S}$. The optimal solution is $\max_{i,j,k}\{B[i,j,k,m]+INS(\mathcal{P}[k+1..n], X-C[i,j,k,m])\}$, which might not be unique. As we have $O(mn^3) = O(n^4)$ cells in the table $B[i,j,k,\ell]$, the following theorem is straightforward.

Theorem 3. *The Contig-Preserving PSF can be solved in $O(n^{26})$ time and space.*

Unfortunately, the time complexity of this algorithm is too high. Hence, we first design some practical algorithms and then show some empirical results for the CP-PSF problem using these practical algorithms.

Practical Algorithms for CP-PSF

We show some property for CP-PSF.

Lemma 4. *The problem of computing an alignment of $\mathcal{P}$ and $\mathcal{S}$ by only inserting spaces between the contigs in $\mathcal{S}$ such that $B62(\mathcal{P},\mathcal{S})$ is maximized, is polynomially solvable.*

Proof. Let $C[1], C[2], ..., C[m]$ be the sequence of contigs in $\mathcal{S}$. Let $c[i]$ be the size of $C[i]$, i.e., $c[i] = |C[i]|$, for $i = 1, .., m$. Let $d[i]$ be the last letter of $C[i]$, for $i = 1, .., m$. Note that $B62^*(\mathcal{P}[i..j], C[k])$ is the maximum B62 score when aligning $\mathcal{P}[i..j]$ and $C[k]$, where no $\star$ can be inserted into $C[k]$. This can be pre-computed in $O(c[k]^2)$ time by setting a large penalty for introducing a $\star$ in $C[k]$ (say $-c[k]$). Therefore, the problem is to align $C[k]$'s, without any gap in each $C[k]$, to disjoint positions at $\mathcal{P}$.

Define $A[i,j,k]$ as the maximum total B62 score obtained when $\mathcal{P}[i..j]$ is aligned with $C[k]$, without using any gap in $C[1], ..., C[k]$.

$$A[i,j,k+1] = \max_{i',j'}\{A[i',j',k] + B62^*(\mathcal{P}[i..j], C[k+1])\},$$

where $i' \leq i - c[k], \sum_{\ell=k+1..m} c[\ell] \leq n - j + 1$.

The initialization is done as follows. $A[i,j,1] = B62^*(\mathcal{P}[i..j], C[1]), \sum_{\ell=2..m} c[\ell] \leq n - j + 1$. The final solution can be found at $\max_{1\leq i<j\leq n}\{A[i,j,m]\}$. The algorithms takes $O(mn^4) = O(n^5)$ time and $O(mn^2) = O(n^3)$ space. □

The above lemma, though does not solve CP-PSF, does give us a heuristic algorithm. After the contigs in $\mathcal{S}$ are aligned at $\mathcal{P}$, we could use a greedy method to form a feasible solution. Among all the gap positions out of any contig in the aligned $\mathcal{S}$, we insert the elements in X at all the $\star$'s positions, from left to right, to maximize the total B62 score. We could use the lemma as a subroutine to have Algorithm 3.3, which runs in $O(mn^4)$ time. In fact, we could implement a simplified heuristic version for the above lemma; namely, we could scan from left to right in $\mathcal{P}$ to find the best locations to locate $C_1, C_2, ..., C_m$. In the worst case, that would only take $O(mn^2)$ time.

1: Align $\mathcal{S}$ with $\mathcal{P}$ to obtain $\mathcal{S}_1$ with the maximum B62 score based on Lemma 4.
2: Find a $\star$ position in the aligned $\mathcal{S}_1$ such that inserting an amino acid in X would incur the maximum B62 score among all $\star$ positions.
3: Repeat Step 2 until all elements in X are inserted into $\mathcal{I}_1$.
4: Return the filled $\mathcal{S}_1$ as $\mathcal{S}'$, with the total alignment score between $\mathcal{S}'$ and $\mathcal{P}$ being $b62(\mathcal{P}, \mathcal{S}, X)$.

Algorithm 3.3: *ContigAlign+Greedy($\mathcal{P}$,$\mathcal{S}$,X)*

We could use a local search idea to try to improve Algorithm 3.3. Here, we could augment the contigs by appending some amino acids at its two ends, but the initial

contigs are never altered.

1: Compute $b62(\mathcal{P}, \mathcal{S}, X)$ using Algorithm 3.3. Assign $U \leftarrow b62(\mathcal{P}, \mathcal{S}, X)$.

2: Insert an amino acid $x \in X$ at the beginning (resp. end) of contig $S_i \in \mathcal{S}$ to obtain $S_i + x$ such that the score $b62(\mathcal{P}, \mathcal{S} - \{S_i\} \cup \{S_i + x\}, X - \{x\})$ is the maximum, for all $x \in X$ and all $S_i \in \mathcal{S}$.

3: Run Algorithm 3.3 to obtain $b62(\mathcal{P}, \mathcal{S} - \{S_i\} \cup \{S_i + x\}, X - \{x\})$. If $b62(\mathcal{P}, \mathcal{S} - \{S_i\} \cup \{S_i + x\}, X - \{x\}) \leq U$, then return the solution $\mathcal{S}''$ incurring U; else update $U \leftarrow b62(\mathcal{P}, \mathcal{S} - \{S_i\} \cup \{S_i + x\}, X - \{x\})$, $\mathcal{S} \leftarrow \mathcal{S} - \{S_i\} \cup \{S_i + x\}$, $X \leftarrow X - \{x\}$ and repeat Step 2.

Algorithm 3.4: *ContigLocalSearch($\mathcal{P}$,$\mathcal{S}$,X)*

Empirical Results for Antibody Proteins

Let $x \in \{$`M-H`,`H-H`,`M-L`$\}$ and let the corresponding references of x be `H-H`, `M-H`, and `H-L` respectively. For each x, we use Algorithm 3.3 to compute the filled sequence $\mathcal{S}'_x$, and we use Algorithm 3.4 to compute the filled sequence $\mathcal{S}''_x$. Instead of directly using the B62 scores, we again use the number of matched pairs resulting from the computed B62 scores. The empirical results are shown in the following two tables. Note that in all cases, compared with Algorithm 3.3, with respect to a reference Algorithm 3.4 produces slightly better results. This can be seen in the first two columns in Table 3.7 and Table 3.8. However, with respect to the corresponding target, it is not always the case that Algorithm 3.4 performs better than Algorithm 3.3 (though the difference is small). This can be checked in the last two columns of Table 3.7 and Table 3.8 (and when $x =$`MabCampath-H`).

Table 3.7: Empirical results for the three computed $\mathcal{S}'_x$ using Algorithm 3.3, with respect to the antibody dataset. `M-H` (resp. `M-L`) represents the heavy (resp. light) chain of `MabCampath`, and `H-H` represents the heavy chain of `Humira`.

	x=`MabCampath-H`	x=`Humira-H`	x=`MabCampath-L`
mp($\mathcal{S}'_x$,ref)	369	421	214
mp($\mathcal{S}'_x$,ref)/ref_length	369/453=81.46%	421/449=93.76%	214/214=100%
mp($\mathcal{S}'_x$,target)	347	363	195
mp($\mathcal{S}'_x$,target)/tar_length	347/449=77.28%	363/453=80.13%	195/214=91.12%

Table 3.8: Empirical results for the three computed $\mathcal{S}''_x$ using Algorithm 3.4, with respect to the antibody dataset. `M-H` (resp. `M-L`) represents the heavy (resp. light) chain of `MabCampath`, and `H-H` represents the heavy chain of `Humira`.

	x=`MabCampath-H`	x=`Humira-H`	x=`MabCampath-L`
mp($\mathcal{S}''_x$,ref)	377	422	214
mp($\mathcal{S}''_x$,ref)/ref_length	377/453=83.22%	422/449=93.97%	214/214=100%
mp($\mathcal{S}''_x$,target)	345	364	195
mp($\mathcal{S}''_x$,target)/tar_length	345/449=76.84%	364/453=80.35%	195/214=91.12%

Table 3.9: Empirical results for the three computed $\mathcal{S}'_x$ using Algorithm 3.3. , with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference.

	x=`CB2`	x=`CB3`	x=`CB4`
mp($\mathcal{S}''_x$,ref)	248	241	235
mp($\mathcal{S}''_x$,ref)/ref_length	248/260=95.38%	241/260=92.69%	235/260=90.38%
mp($\mathcal{S}''_x$,target)	198	194	190
mp($\mathcal{S}''_x$,target)/tar_length	198/258=76.74%	194/258=75.19%	190/258=73.64%

Table 3.10: Empirical results for the three computed $\mathcal{S}''_x$ using Algorithm 3.4, with respect to the mammalian dataset. `CB`i ($i = 2, 3, 4$) represents the i peptides of CAH2 of *Bos taurus*. `CH` represents CAH2 of Human, which is the reference.

	x=`CB2`	x=`CB3`	x=`CB4`
mp($\mathcal{S}''_x$,ref)	248	242	237
mp($\mathcal{S}''_x$,ref)/ref_length	248/260=95.38%	242/260=93.08%	237/260=91.15%
mp($\mathcal{S}''_x$,target)	198	194	194
mp($\mathcal{S}''_x$,target)/tar_length	198/258=76.74%	194/258=75.19%	194/258=75.19%

Empirical Results for Mammalian Proteins

Similar to the previous section, $x \in \{$`CB2, CB3, CB4`$\}$ and let the corresponding references all be `CH`. We use Algorithm 3.3 to compute the filled sequence $\mathcal{S}'_x$, and we use Algorithm 3.4 to compute the filled sequence $\mathcal{S}''_x$. The empirical results are shown in the following two tables. It can be seen that Algorithm 3.4 performs at least as good as Algorithm 3.3, although the improvement is small.

Concluding Remarks

In this chapter, we study the protein scaffold filling problem when a reference protein is given and we solve the two corresponding versions in which fill the missing amino acids into the scaffold such that the B62 score between reference protein sequence and our constructed protein sequence is maximized, in $O(n^{22})$ and $O(n^{26})$ time, which are both impractical, respectively. We then design two practical methods, for each version, and obtain some empirical results using some datasets from four antibody protein sequences and mammalian proteins. The empirical results, based on some antibody and mammalian proteins, show that the algorithms can fill protein scaffolds with high quality, provided that a good pair of scaffold and reference are

given. Our empirical results are very promising: as long as the right reference protein and a high-quality scaffold are given, the algorithms can fill the scaffold with 73%-91% accuracy.

In practice, with top-down mass spectrometry, we might know the length of the target protein sequence. In this case, we might only need to insert a subset $X' \subset X$ of the missing amino acids. This will be an interesting direction for future research.

In the next chapter, we start to describe our research results on genome rearrangement problems.

AN FPT APPROXIMATION ALGORITHM FOR THE DCJ DISTANCE

Comparing genomes in terms of gene order is a classical combinatorial optimization problem in computational biology. The double cut and join operation (abbreviated as DCJ) is a synthetic genome rearrangement operation, and extensively used while comparing two genomes. The problem of computing the DCJ distance between two genomes is defined as: given two genomes, apply the minimum number of DCJs to transform one genome into the other. Like many genomic distances between two genomes, e.g., Reversal and Translocation, the DCJ distance between two signed genomes can be computed in linear time, while for unsigned genomes, the problem becomes NP-hard. So far as we know, the best approximation algorithms for Sorting by Reversals, Sorting by Transpositions and Sorting by Translocations have a factor 1.375; moreover, for the DCJ distance the best factor is only $1.408+\epsilon$. Computing the DCJ distance between two unsigned genomes involves computing a proper cycle decomposition of its breakpoint graph, which becomes the bottleneck for computing the genomic distance under almost all types of genome rearrangement operations (e.g., Reversal and Translocation), and prohibits to obtain approximation factors better than 1.375 in polynomial time. In this chapter, we devise an fixed-parameter tractable approximation algorithm for computing the DCJ distance with an approximation factor $4/3+\epsilon$, and the running time is $O^*(2^{d^*})$, where d^* represents the optimal DCJ distance. This technique can be used to improve the approximation factor for other genomic distances, like Reversal and Translocation.

Related Work

Computing genomic distance on gene order is a fundamental problem in computational biology. In the last two decades, a variety of biological operations, such

as reversals, translocations, fusions, fissions, transpositions and block-interchanges, have been proposed to handle gene order. The double cut and join operation, introduced by Yancopoulos *et al.* [101], unifies all the classical operations. In the past, the rearrangement distance for signed genomes is well studied by single operations (like reversals) [52], combinations of operations (reversals, translocations, fusions and fissions) [51] and universal operations (double cut and join) [12] .

Unfortunately, as for unsigned genomes, almost all these problems are NP-hard. Then people devote to devise approximation algorithms. Christie devised a factor-1.5 approximation algorithm for sorting unsigned genomes by reversals [28], and the approximation factor was improved to 1.375 by Berman *et al.* in 2002 [14]. Cui *et al.* investigated the problem of sorting by unsigned translocations and proposed an algorithm with an approximation factor $1.5 + \epsilon$ [34]. This bound was improved to $1.408 + \epsilon$ [55] and recently further to 1.375 [86]. The problem of Sorting by Transpositions was first studied by Bafna and Pevzner [7], who devised an 1.5-approximation algorithm which runs in quadratic time. The bound was improved to 1.375 by Elias and Hartman in 2006 [41]. As far as we know, the best polynomial-time approximation algorithms for the unsigned DCJ distance problem has a factor $1.408 + \epsilon$ [21]. Among almost all these problems, a bottleneck to break the 1.375 barrier seems to be on decomposing the breakpoint graph (to be defined formally) into maximum (edge-disjoint) alternating cycles. We make fundamental contributions in this chapter on using FPT algorithms.

The design of FPT algorithm for genome rearrangement problems was started very recently. With the help of weak kernels, sorting unsigned genomes by either reversals, translocations or DCJs admits small weak kernels, hence are in FTP with a running time $O^*(4^k)$ [58, 59]. However, this algorithm is only practical for k bounded from above by around 20 to 25.

Preliminaries

Gene, Chromosome and Genome An unsigned gene is a sequence of DNA, usually denoted by a positive integer. A chromosome can be viewed as a sequence of genes and denoted by a permutation, while a genome is a set of chromosomes. A gene lies at the end of some linear chromosome is called an *ending-gene*. Gene i and j form an *adjacency* if they are consecutive in some chromosome. An adjacency (g_i, g_{i+1}) is *trivial* if it satisfies $|g_{i+1} - g_i| = 1$. A chromosome is trivial if every adjacency is trivial. A genome is trivial if all its chromosomes are trivial.

In the context of sorting genomes, the comparative order of the genes in the same chromosome does matter, but not the order of chromosomes and the direction of a whole chromosome, which implies that each chromosome can be viewed in both directions. In the case of signed genomes, a chromosome $\langle g_i, g_{i+1}, \cdots, g_j \rangle$ is equivalent to $\langle -g_j, \cdots, -g_{i+1}, -g_i \rangle$; and in the case of unsigned genomes, a chromosome $\langle g_i, g_{i+1}, \cdots, g_j \rangle$ is equivalent to $\langle g_j, \cdots, g_{i+1}, g_i \rangle$.

Breakpoint Graph of Signed Genomes

Now, we recall the well-known tool for computing the genomic rearrangement distance, the *Breakpoint Graph*. Given signed genomes X and Y, the breakpoint graph $G_s(X, Y)$ can be obtained as follows: for each chromosome $S = [x_1, x_2, \ldots, x_{n_i}]$ of X, replace each x_i with an ordered pair $(l(x_i), r(x_i))$ of vertices. If x_i is positive, then $(l(x_i), r(x_i)) = (x_i^t, x_i^h)$; and if x_i is negative, then $(l(x_i), r(x_i)) = (x_i^h, x_i^t)$. If the genes x_i and x_{i+1} are adjacent in X, then we connect $r(x_i)$ and $l(x_{i+1})$ by a black edge in $G_s(X, Y)$. If the genes x_i and x_{i+1} are adjacent in B, then we connect $r(x_i)$ and $l(x_{i+1})$ by a gray edge in $G_s(X, Y)$. Every vertex (except the ones at the two ends of a chromosome) in $G_s(X, Y)$ is incident to one black and one gray edge. Therefore,

$G_s(X, Y)$ can be uniquely decomposed into cycles, on which the black edges and gray edges appears alternatively. A cycle containing exactly i black (gray) edges is called an i-cycle.

The Signed DCJ Distance Formula

Let b (resp. c) be the number of black edges (resp. cycles) in $G_s(X, Y)$. Then, Yancopoulos *et al.* showed the following theorem [101].

Theorem 4. $d_s(X, Y) = b - c$.

Breakpoint Graph for Unsigned Genomes and the UDCJ Problem

As for unsigned genomes, the breakpoint graph is a bit different. Given two unsigned genomes X and Y on the same set of n genes, the *Breakpoint Graph* $G_u(X, Y) = (V, E_b \cup E_g)$, where $|V| = n$ and each vertex in V corresponds to a gene, every adjacency in X forms a black edge belonging to E_b and every adjacency in Y forms a gray edge belonging to E_g. the breakpoint graph $G_u(X, Y)$ can be decomposed into a set of edge-disjoint cycles, denoted by $\mathbf{D}$, and on each cycle, the black edge and gray edge appears alternatively.

The Double Cut and Join Operations The Double Cut and Join operation (abbreviated as DCJ) unifies all the traditional genome rearrangement operations such as reversal, translocation, fusion, fission, transposition and block-interchange, as well as excision, integration, circularization and linearization. The formal definition of the DCJ operation on the breakpoint graph (for both signed and unsigned genomes) is as follows.

Definition 4.0.1. *The Double Cut and Join operation acts on the Breakpoint Graph in the following four ways (Figure 4.1):*

1. *For two black edges* $b_1 = (g_i, g_{i+1})$ *and* $b_2 = (g_j, g_{j+1})$*, cut them, and either form two new black edges* $b'_1 = (g_i, g_{j+1})$ *and* $b'_2 = (g_j, g_{i+1})$ *or form two new black edges* $b'_1 = (g_i, g_j)$ *and* $b'_2 = (g_{i+1}, g_{j+1})$.

2. *For a black edge* $b = (g_i, g_{i+1})$ *and an end-gene* g_j*, cut the black edge, and either form a new black edge* $b' = (g_i, g_j)$ *and a new end-gene* g_{i+1} *or form a new black edge* $b' = (g_j, g_{i+1})$ *and a new end-gene* g_i.

3. *For two end-genes* g_i *and* g_j*, join them with a black edge* (g_i, g_j).

4. *For a black edge* $b = (g_i, g_{i+1})$*, cut it into two end-genes* g_i *and* g_{i+1}.

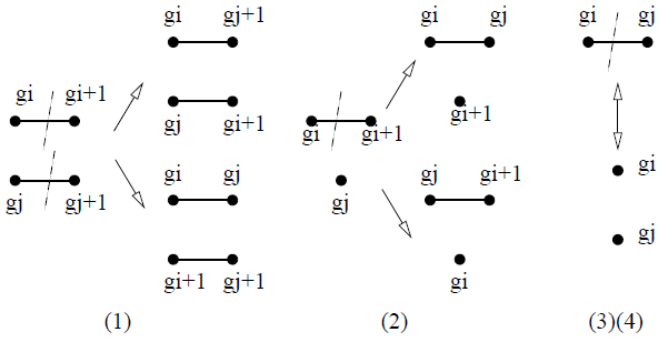


Figure 4.1: The DCJ Operation

Problem Statement We now formally formulate the problem to be investigated in this chapter.

Sorting Unsigned Genomes by the DCJs (UDCJ):

Input: Two unsigned linear genomes X and Y, Y is trivial, and an integer k.

Question: Can X be converted into Y by a series of k DCJs $\rho_1, \rho_2, \cdots, \rho_k$.

The minimum k is the unsigned *DCJ distance* between X and Y.

Throughout this chapter, we assume that the ending-gene sets of X and Y are the same, since the details to handle genomes with different ending-gene sets is not the main purpose of this chapter.

Coming back to the technical details, since each ending gene of X and Y is incident to only one black edge and one gray edge; and each of the rest genes is incident with exactly two blacks edge and two gray edges, the ways to decompose $G_u(X, Y)$ into cycles might not be unique. Caprara showed that computing a maximum alternating-cycle decomposition (MAX-ACD) of the breakpoint graph is NP-hard [17], which implies that the UDCJ is also NP-hard. We comment that the best polynomial-time approximation for MAX-ACD only has a factor $1.4193 + \epsilon$ [65].

Converting Unsigned Genomes into Signed Ones

A natural way to solve UDCJ is to convert the unsigned genome into a signed one, then resort to the algorithm for computing the signed DCJ distance. But, how to convert an unsigned genome into a 'good' signed genome, which would result in a smaller DCJ distance? Once we have a cycle-decomposition $\mathbf{D}$ of $G_u(X, Y)$, we can obtain two signed genomes $\bar{X}$ and $\bar{Y}$ by assigning a sign to each gene in X and Y, so that $G_s(\bar{X}, \bar{Y}) = \mathbf{D}$.

As Y is trivial, we arrange all its chromosomes monotonously increasing, the assign all its genes positive. Therefore, all gray edges in $G_s(\bar{X}, \bar{Y})$ have the form $((x_i)^h, (x_i + 1)^t)$.

Next, we show how to assign a proper sign to each gene in X (to obtain $\bar{X}$). An ending gene is positive if it lies at the same (i.e., both left or both right) ends of some chromosome in X and some chromosome in Y; otherwise, it is negative in X. For a non-ending gene x_i, according to the two gray edges, $((x_i)^h, (x_i + 1)^t)$ and $((x_i - 1)^h, (x_i)^t)$ in the cycle decomposition, we assign x_i positive if $((x_i - 1)^h, l(x_i))$

is a gray edge in the given cycle decomposition; if $((x_i - 1)^h, r(x_i))$ is a gray edge in the given cycle decomposition, then x_i is assigned a negative sign. See Figure 4.2 for an example.

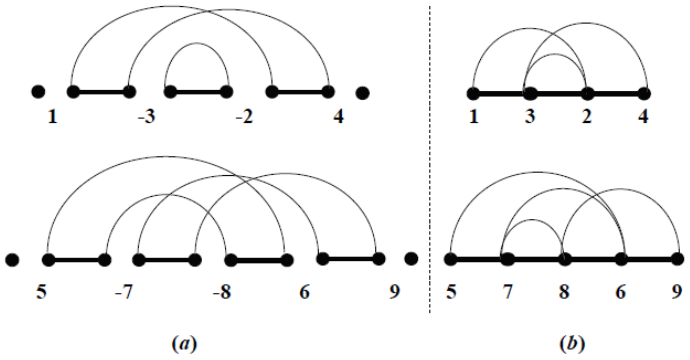


Figure 4.2: (a) The breakpoint graph $G_s(\bar{X}, \bar{Y})$ for the signed case, where $\bar{X} = \{[1, -3, -2, 4], [5, -7, -8, 6, 9]\}$ and $\bar{Y} = \{[1, 2, 3, 4], [5, 6, 7, 8, 9]\}$. (b) The breakpoint graph $G(X, Y)$ for the unsigned case, where $X = \{[1, 3, 2, 4], [5, 7, 8, 6, 9]\}$ and $Y = \{[1, 2, 3, 4], [5, 6, 7, 8, 9]\}$. (a) is a cycle decomposition of (b)

Fixed Parameter Tractable Basically, a fixed-parameter tractable (FPT) algorithm for a decision problem Π with solution value k is an algorithm which solves the problem in $O(f(k)n^{O(1)}) = O^*(f(k))$ time, where f is any function only on k, n is the input size. FPT also stands for the set of problems which admit such an algorithm [36, 44].

In summary, to solve UDCJ efficiently, we need to find a proper cycle decomposition of the breakpoint graph. We handle this NP-hard problem by designing an FPT approximation algorithm. This is the main content in the next section.

An FPT-Time Approximation Algorithm

In this section, we present a factor-$(4/3 + \epsilon)$ FPT-approximation algorithm for UDCJ, which runs in $O^*(2^{d^*})$, where d^* represents the optimal DCJ distance. We

try to decompose the breakpoint graph into enough number of small cycles (i.e., cycles containing 1,2,and 3 black edges, formally called 1-cycles, 2-cycles and 3-cycles henceforth), with a new randomized method.

Why Caring Small Cycles

Jiang *et al.* [59] showed that all the possible 1-cycles could be kept in the cycle decomposition by the following lemma.

Lemma 5. *There exists some optimal cycle decomposition containing all the existing 1-cycles.*

Hence, our cycle-decomposition will keep all the 1-cycles, and is always compared with an optimal cycle-decomposition which also keeps all the 1-cycles.

Let c_i^* be the number of i-cycle in some optimal cycle-decomposition, from the signed DCJ distance formula $d_s(X,Y) = b - \sum_{i\geq 1} c_i^*$. Since $b = \sum_{i\geq 1} i * c_i^*$, we have $\sum_{i\geq 3} c_i^* \leq \frac{1}{3}(b - c_1^* - 2c_2^*)$, that results in, $d_s(X,Y) \geq b - c_1^* - c_2^* - \frac{1}{3}(b - c_1^* - 2c_2^*) = \frac{2}{3}(b - c_1^*) - \frac{1}{3}c_2^* = \frac{2}{3}(b - c_1^* - \frac{1}{2}c_2^*)$, which implies that, to achieve an approximation factor of 1.5, it is efficient to find a half number of 2-cycles of the optimal cycle-decomposition.

Moreover, if we can find an α portion of 2-cycles and a β portion of 2-cycles and 3-cycles, then use the bound, $d_s(X,Y) = b - \sum_{i\geq 1} c_i^* \geq b - c_1^* - c_2^* - c_3^* - \frac{1}{4}(b - c_1^* - 2c_2^* - 3c_3^*)$, together with the conditions $c_2 \geq \alpha * c_2^*$, $c_2 + c_3 \geq \beta * (c_2^* + c_3^*)$, and $d_{alg}(X,Y) \leq b - c_1^* - c_2 - c_3$, the approximation factor could become $max\{\frac{4}{3}, 2-\alpha, \frac{3-\beta}{2}, \frac{3\alpha-\beta-\alpha\beta}{2\alpha-\beta}\}$ [18].

Our idea is to find an α portion of 2-cycles, and a β portion of 2-cycles as well as a γ portion of 3-cycles. We will show that, ignoring some small constant ϵ, $\alpha \geq \frac{5}{6}$, $\beta \geq \frac{3}{5}$, and $\gamma \geq \frac{3}{5} * \frac{57}{64}$, which leads to an approximation 4/3.

Now we will give the details of our algorithm. By cycle decomposition, we aim at finding small cycles, but searching cycles directly from the breakpoint graph will

not guarantee enough number, that is because cycles in the breakpoint graph could possibly share some edges, and it is necessary to compute an independent set from them. Our main idea is to fix the sign of some genes, then find cycles from the partly decomposed breakpoint graph, so that we can obtain more cycles.

Finding 2-cycles

Let V_2 be the set of vertices, each of which is involved in at least two 2-cycles in the breakpoint graph. (Following [18], the intersection graph of the 2-cycles has a maximum degree of 6. By the following random selection procedure, together with the enumeration of the signs of the selected vertices, we show that the intersection graph of these 'partial' 2-cycles has a maximum degree of 3.) We randomly choose $|V_2|/2$ vertices and enumerate all possible combinations of signs for them. Under each combination of sign assignment, the breakpoint graph could be partly decomposed. A *candidate* 2-cycle is a 2-cycle which could exist subject to the current sign assignment, and at least one vertex has a sign fixed. Nonetheless, in the partly decomposed breakpoint graph, some of the candidate 2-cycles could share edges and could not co-exist in any cycle decomposition. Let C_{c2} be the set of candidate 2-cycles. Construct a conflict graph $G_{c2} = (C_{c2}, E_{c2})$, where each 2-cycle of C_{c2} corresponds to a vertex, and there is an edge between two vertices if and only if their corresponding 2-cycles share edges in the partly decomposed breakpoint graph. Thus, an independent set of G_{c2} represents the 2-cycles we find.

Lemma 6. *There exists an approximation algorithm with ratio* $\frac{5}{r+3} - \varepsilon$*, for any* $\varepsilon > 0$*, for the maximum independent set problem on a graph with maximum degree* r*.*

Proof. Refer to reference [13]. □

1: Identify vertices of V_2.

2: Choose $|V_2|/2$ vertices randomly.

3: **for** each combination of sign assignment for these chosen vertices **do**

4: Construct the conflict graph G_{c2}.

5: Compute an approximate independent set I_{c2} of G_{c2}.

6: **end for**

7: Keep the 2-cycles corresponding to the maximum I_{c2}.

Algorithm 4.5: Finding 2-cycles

Finding 2,3-cycles

Let a 2,3-cycle be either a 2-cycle or a 3-cycle in the breakpoint graph. Let V_{23} be the set of vertices, each of which is involved in at least two 2,3-cycles in the breakpoint graph. We randomly choose $|V_{23}|/2$ vertices, enumerate all possible combinations of signs for them. Under each combination of sign assignment, the breakpoint graph has been partly decomposed. A *candidate* 2,3-cycle is a 2-cycle or a 3-cycle which could exist under the current sign assignment, and at least two vertices have their signs fixed. In the partly decomposed breakpoint graph, some of the 2-cycles and candidate 3-cycles could share edges and could not co-exist in any cycle decomposition. Each 2-cycle and candidate 3-cycles is composed of paths, where each path is composed of black edges, or gray edges, or composed of black edges and gray edges appearing alternatively. A candidate 2,3-cycles is composed of at most four such paths. We view paths as elements, and candidate 2,3-cycles as sets of elements. We can construct a set packing system S_{c23}, whose basic elements are the paths and each candidate 2,3-cycles is a subset of at most four elements. Thus, a set packing could be the 2,3-cycles we find.

1: Identify vertices of V_{23}.

2: Choose $|V_{23}|/2$ vertices randomly.

3: **for** each combination of sign assignment for these chosen vertices **do**

4: Construct the set packing system S_{c23}.

5: Compute an approximate set packing P_{c23}.

6: **end for**

7: Keep the 2,3-cycles corresponding to the maximum P_{c23}.

Algorithm 4.6: Finding 2,3-cycles

Lemma 7. *There is a ratio $\frac{3}{p+1} - \varepsilon$ approximation algorithm, for any $\varepsilon > 0$, for the maximum set packing problem with set size at most p and set degree bounded.*

Proof. Refer to reference [50]. □

1: Run Algorithm 4.5 n times, among the computed I_{c2}'s, pick the largest one $I_{c2}^{\max}$.

2: Run Algorithm 4.6 n times, among the computed P_{c23}'s, pick the largest one $P_{c23}^{\max}$.

3: If $|I_{c2}^{\max}| \geq |P_{c23}^{\max}|$, then keep the 2-cycles corresponding to $I_{c2}^{\max}$,

4: then keep the 2-cycles corresponding to $I_{c2}^{\max}$,

5: Otherwise, keep the 2,3-cycles corresponding to $P_{c23}^{\max}$.

6: Arbitrarily assign signs to the rest of genes so that every gene has a sign.

7: Compute the DCJ distance between the signed genomes.

8: Simulate the signed DCJ sorting process to the original unsigned genomes.

Algorithm 4.7: UDCJ-Final

Performance Analysis

The Approximation Factor

As aforementioned, the approximation factor performance is determined by the number of 2-cycles and 3-cycles we have found.

Lemma 8. *G_{c2} is a graph with maximum degree 3.*

Proof. Each vertex in G_{c2} corresponds to a 2-cycle with at least one fixed-sign vertex in the breakpoint graph, then each such 2-cycle is formed by a black edge, a gray edge and a colorful path of two edges. We show that, in any case, a black edge, a gray edge or a colorful path can belong to at most two such 2-cycles.

Let (a, b) be a black edge in the breakpoint graph, to form some 2-cycle, exactly one of a and b should connect to a colorful path of two edges. Once a 2-cycle has 3 edges fixed, it can be generated directly by adding the unique fourth edge. Hence, if a connects a colorful path of two edges, there could be a 2-cycle; if b connects a colorful path of two edges, there should be another 2-cycle. Thus, there are at most two 2-cycles sharing the black edge (a, b).

Since the black edge and gray edge are symmetric, the argument for gray edge is similar.

Let (a, b, c) form a colorful path where (a, b) is a black edge and (b, c) is a gray edge. To form 2-cycles, c has at most two choices: connects to the black edge on its left or the black edge on its right. In either case, the 2-cycle could be fixed. Thus, there are at most two 2-cycles sharing the colorful path (a, b, c).

Therefore, a 2-cycle with at least one fixed-sign vertex in the breakpoint graph can share its edges with at most three other such cycles. This completes the proof. □

Corollary 1. *I_{c2} is an $\frac{5}{6} - \varepsilon$ approximation for the maximum independent set of G_{c2}.*

Lemma 9. *S_{c23} is a set packing system with set size at most 4 and set degree bounded.*

Proof. Each 3-cycle has exactly 6 vertices, since at least two of them have a fixed sign, each fixed-sign vertex brings a fixed connection of a black edge and a gray edge. Then each 3-cycle has at most four undetermined connections, i.e., it is composed of at most four paths.

Now we show that each path can be shared by at most 4 such 2,3-cycles. Note that if three paths of such a 3-cycle are fixed, then the 2,3-cycle is obtained. Standing on an ending vertex of a path, we have at most two choices. After any choice, the path is extended, i.e., two paths are connected together. To form 2,3-cycles, we have twice opportunities to make choices, which would result in at most four 2,3-cycles. Thus, each subset could share elements with at most 4×(4-1)=12 other subsets. The set degree is bounded.

□

Corollary 2. *P_{c23} is an $\frac{3}{5} - \varepsilon$ approximation for the maximum set packing of S_{c23}.*

Next, we bound the number of vertices in G_{c2} and the number of subsets in S_{c23}. Let c_i^* be the number of i-cycles in the optimal cycle decomposition.

Lemma 10. *With probability $1 - \frac{1}{e^{O(n)}}$, the maximum independent set of one G_{c2} at Step 1 in Algorithm 4.7 has a size greater than $(1-\delta)\frac{15}{16}c_2^*$ for any $0 < \delta < 1$.*

Proof. We show that each 2-cycle of the optimal cycle decomposition has a probability of $\frac{15}{16}$ to fall into G_{c2}. If the 2-cycle has a vertex with a fixed sign, then it will surely become a candidate 2-cycle. If all its vertices do not have a fixed sign, then they are all in V_2. The probability that none of them is chosen is $\frac{C_{|v_2|-4}^{|v_2|/2}}{C_{|v_2|}^{|v_2|/2}} \leq \frac{1}{16}$. Hence, we can conclude that, with probability $\frac{15}{16}$, we have $\frac{15}{16}$ portion of 2-cycles of the optimal cycle decomposition in G_{c2}; moreover, they form an independent set of G_{c2}. If we view X_i

as a random variable to put a 2-cycle of the optimal cycle decomposition into G_{c2} and define $X = \sum_i X_i$, then $E[X] = \mu = \frac{15}{16}c_2^*$.

By a *Chernoff bounds*, for any $0 < \delta < 1$,

$$P[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu}{2}},$$

which means $P[X \leq (1-\delta)\mu] \leq \frac{1}{e^{O(1)}}$ when μ is only a constant. For Algorithm 4.7, as we repeat Algorithm 4.5 n times, the probability that the MIS of all the n G_{c2}'s has a size at most $(1-\delta)\frac{15}{16}c_2^*$ is at most $(\frac{1}{e^{O(1)}})^n = \frac{1}{e^{O(n)}}$.

Therefore, with probability $1 - \frac{1}{e^{O(n)}}$, the MIS of one of the G_{c2} at Step 1 in Algorithm 4.7, has a size greater than $(1-\delta)\frac{15}{16}c_2^*$ for any $0 < \delta < 1$.

□

As δ could be arbitrarily small, we would use this size as $\frac{15}{16}c_2^*$ in the proof of Theorem 5.

Lemma 11. *With probability* $1 - \frac{1}{e^{O(n)}}$*, the maximum set packing of one of the* S_{c23} *at Step 2 in Algorithm 4.7 has a size greater than* $(1-\delta)(c_2^* + \frac{57}{64}c_3^*)$ *for any* $0 < \delta < 1$.

Proof. We show that each 3-cycle of the optimal cycle decomposition has a probability of $\frac{57}{64}$ to be a candidate 3-cycle. If the 3-cycle has two vertices whose signs are fixed, then it will surely become a candidate 3-cycle. If the 3-cycle has exactly one vertex whose sign is fixed, then the other five vertices do not have a fixed sign, so they are all in V_{23}. The probability that none of them is chosen is $\frac{C_{|v_{23}|-5}^{|v_{23}|/2}}{C_{|v_{23}|}^{|v_{23}|/2}} \leq \frac{1}{32}$. If all its vertices do not have a fixed sign, then they are all in V_{23}. The probability that none of them is chosen is $\frac{C_{|v_{23}|-6}^{|v_{23}|/2}}{C_{|v_{23}|}^{|v_{23}|/2}} \leq \frac{1}{64}$. The probability that exact one of them is chosen is $6 \times \frac{C_{|v_{23}|-6}^{|v_{23}|/2}}{C_{|v_{23}|}^{|v_{23}|/2}} \leq \frac{6}{64}$. Hence, we can conclude that an expected $1 - \frac{1}{64} - \frac{6}{64} = \frac{57}{64}$ portion of 3-cycles of the optimal cycle decomposition are in S_{c23}; moreover, together with the

c_2^* 2-cycles of the optimal cycle decomposition, they form a set packing of S_{c23}. Note that a 2-cycle has only 4 edges, by Lemma 9, all of the c_2^* 2-cycles will be in the set packing solution.

By an argument similar to Lemma 10, we conclude that, with probability $1 - \frac{1}{e^{O(n)}}$, the maximum set packing one of the S_{c23}'s at Step 2 of Algorithm 4.7, has size greater than $(1-\delta)(c_2^* + \frac{57}{64}c_3^*)$ for any $0 < \delta < 1$.

□

Again, as δ could be arbitrarily small, we would use this size as $c_2^* + \frac{57}{64}c_3^*$ in the proof of Theorem 5.

Theorem 5. *With probability* $1 - \frac{1}{e^{O(n)}}$*, Algorithm 4.7 approximates the DCJ distance within a factor 4/3+ε.*

Proof. Let c_i be the number of i-cycles computed by Algorithm 4.7, c_i^* be the number of i-cycles in the optimal cycle decomposition. Let d^* and d be the optimal DCJ distance and approximated DCJ distance respectively. Let the approximation factor be ρ, where $\frac{4}{3} \leq \rho \leq \frac{3}{2}$.

We have, $d^* = b - c_1^* - c_2^* - c_3^* - \sum_{i \geq 4} c_i^*$ Since, $\sum_{i \geq 4} c_i^* \leq (b - c_1^* - 2c_2^* - 3c_3^*)/4$, and $b - c_1^* \geq 2c_2^* + 3c_3^*$, then,

$$\begin{aligned}
d^* &\geq b - c_1^* - c_2^* - c_3^* - (b - c_1^* - 2c_2^* - 3c_3^*)/4 \\
&= \frac{3}{4}(b - c_1^*) - \frac{1}{2}c_2^* - \frac{1}{4}c_3^* \\
&= \frac{1}{\rho}((b - c_1^*) + (\frac{3}{4}\rho - 1)(b - c_1^*) - \frac{1}{2}\rho c_2^* - \frac{1}{4}\rho c_3^*) \\
&\geq \frac{1}{\rho}((b - c_1^*) + (\frac{3}{4}\rho - 1)(2c_2^* + 3c_3^*) - \frac{1}{2}\rho c_2^* - \frac{1}{4}\rho c_3^*) \\
&= \frac{1}{\rho}(b - c_1^* - (2-\rho)c_2^* - (3-2\rho)c_3^*) \qquad (4.1)
\end{aligned}$$

This implies that, if the number of cycles we found satisfy $c_2 \geq (2-\rho)c_2^*$ and $c_3 \geq (3-2\rho)c_3^*$, then the approximated DCJ distance computed by our algorithm satisfies $d \leq b - c_1^* - c_2 - c_3$; consequently, the approximation factor reaches to ρ.

From Corollary 1 and Lemma 10, with high probability, we have

$$|I_{c2}^{\max}| \geq (\frac{5}{6} - \varepsilon) \times \frac{15}{16} c_2^*. \tag{4.2}$$

From Corollary 2 and Lemma 11, with high probability, we have

$$|P_{c23}^{\max}| \geq (\frac{3}{5} - \varepsilon') \times (c_2^* + \frac{57}{64} c_3^*). \tag{4.3}$$

Now we show that, by a balanced analysis, at least one of $|I_{c2}^{\max}|$ and $|P_{c23}^{\max}|$ are greater than $(2-\rho)c_2^* + (3-2\rho)c_3^*$.

We have two cases: either (I) $(\frac{5}{6} - \varepsilon) \times \frac{15}{16} c_2^* \geq (\frac{3}{5} - \varepsilon') \times (c_2^* + \frac{57}{64} c_3^*)$ or (II) not.

In case (I): $(\frac{5}{6} - \varepsilon) \times \frac{15}{16} c_2^* \geq (\frac{3}{5} - \varepsilon') \times (c_2^* + \frac{57}{64} c_3^*)$. Since ε and ε' are very small comparing with the other constants, for the sake of computation simplification, we ignore them here. Then, $(\frac{25}{32} - \frac{3}{5}) c_2^* \geq \frac{3}{5} \times \frac{57}{64} c_3^*$; that is, $c_2^* \geq \frac{171}{58} c_3^*$. Therefore, if $\frac{25}{32} c_2^* \geq (2-\rho)c_2^* + (3-2\rho)c_3^*$, we are done. Solving this inequality,

$$(2-\rho)c_2^* + (3-2\rho)c_3^* \leq (2-\rho)c_2^* + (3-2\rho) \times \frac{58}{171} c_2^* \leq \frac{25}{32} c_2^*,$$

it holds provided that $\rho \geq 1.332$. Together with the constraint that $\rho \geq \frac{4}{3}$, we choose $\rho = \frac{4}{3}$.

In case (II): $(\frac{5}{6} - \varepsilon) \times \frac{15}{16} c_2^* < (\frac{3}{5} - \varepsilon') \times (c_2^* + \frac{57}{64} c_3^*)$. Similarly, we have $c_2^* < \frac{171}{58} c_3^*$. If $\frac{3}{5} \times (c_2^* + \frac{57}{64} c_3^*) \geq (2-\rho)c_2^* + (3-2\rho)c_3^*$, then we are done. Hence we need to prove,

$$(\frac{3}{5} \times \frac{57}{64} - 3 + 2\rho) c_3^* \geq (2 - \frac{3}{5} - \rho) c_2^*.$$

Again, this inequality holds when $\rho \geq 1.332$. Together with the constraint that $\rho \geq \frac{4}{3}$, we have $\rho = \frac{4}{3}$.

□

The Time Complexity

Theorem 6. *The time complexity of Algorithm 4.7 is $O^*(2^{d^*})$, where d^* is the optimal DCJ distance.*

Proof. The most time-consuming parts are enumerating all possible sign combination of V_2 and V_{23}. $d^* \geq b - c_1^* - c_2^* - c_3^*$ and $c_2^* + c_3^* \leq (b - c_1^*)/2$, then, $d^* \geq (b - c_1^*)/2$. Each vertex of V_2 or V_{23} is connected to two black edges, while each black edge has at most two unsigned genes as its endpoints, which means that the number of unsigned genes of V_2 or V_{23} is smaller than that of black edges, e.g., $|V_2| \leq b - c_1^* \leq 2d^*$ and $|V_{23}| \leq b - c_1^* \leq 2d^*$.

Hence while we choose a half of $|V_2|$ vertices or a half of $|V_{23}|$ vertices, and enumerating all their combination of signs, the time complexity is at most $O^*(2^{d^*})$.

□

Simulation Results

We implement the algorithm. The code is written in Matlab and we run it on a simulated genome. Initially, the genome G has two chromosomes, each containing 100 genes, so G has $n = 200$ genes. We perform a sequence of k random DCJ operations to obtain the resulting genome G'. G' is fed into the algorithm and the approximated DCJ solution is returned. The result is summarized in the following Table 4.1.

We comment that the results seem to be convincing as k'/k is usually around $1.2 \sim 1.3$. The cases taking the longest time to run, which was about 72 hours, occurred when k reaches 40.

k	12	16	20	24	28	32	36	40	44	48
k'	14.8	19.6	25.2	30.2	36.2	40.4	46.4	51.8	56.4	61.8

Table 4.1: The computed approximation solution value k' versus the potential optimal solution value k. k' is computed over 5 tries.

Concluding Remarks

We design a factor $4/3 + \epsilon$ FPT-approximation algorithm for the DCJ distance, improving the previous (polynomial-time approximation) factor of $1.408 + \epsilon$. The running time is bounded by $O^*(2^k)$; in fact, by $O^*(2^x)$ where $x \leq k$, which makes it practical for k at least as large as $40 \sim 50$. The exact FPT algorithm for the same problem takes $O^*(4^k)$ time, which is only practial for k bounded by $20 \sim 25$ from above. Our algorithm involves a new randomized method to decompose the breakpoint graph into the maximum number of alternating-cycles and could be used to improve the approximation factor for Sorting by Translocations — in FPT time, which only admits a factor-1.375 (polynomial-time) approximation and uses maximum alternating-cycle decomposition as a subroutine.

For Sorting by Reversals, note that special care must be taken as in the optimal solution the 1-cycles might not be all kept. For instance, for the sequence $S = \langle 3, 4, 1, 2\rangle$, if we keep the 1-cycles (3,4) and (1,2) then three reversals are needed to sort S into $\langle 1, 2, 3, 4\rangle$. On the other hand we could sort $S = \langle 3, 4, 1, 2\rangle$ into $\langle 1, 2, 3, 4\rangle$ using two reversals: $\langle 3, \underline{4, 1, 2}\rangle \rightarrow \langle \underline{3, 2, 1}, 4\rangle \rightarrow \langle 1, 2, 3, 4\rangle$.

THE ONE-SIDED EXEMPLAR ADJACENCY NUMBER PROBLEM

We study the one-sided exemplar adjacency number problem. Given one generic linear genome $\mathcal{G}$ with gene duplications (over n gene families), an exemplar genome G is a permutation obtained from $\mathcal{G}$ by deleting duplicated genes such that G contains exactly one gene from each gene family (i.e., G is a permutation of length n). If we relax the constraint such that G^+ is obtained in the same way but has length at least k, then we call G^+ a pseudo-exemplar genome. Given $\mathcal{G}$ and one exemplar genome H over the same set of n gene families, the One-sided Exemplar Adjacency Number problem (One-sided EAN) is defined as follows: delete duplicated genes from the genome $\mathcal{G}$ to obtain an exemplar genome G of length n, such that the number of adjacencies between G and H is maximized. It is known that the problem is NP-hard; in fact, almost as hard to approximate as Independent Set, even when each gene (from the same gene family) appears at most twice in the generic genome $\mathcal{G}$. To overcome the constraint on the length of G, we define a slightly more general problem (One-sided EAN+) where we only need to obtain a pseudo-exemplar genome G^+ from $\mathcal{G}$ (by deleting duplicated genes) such that the number of adjacencies in H and G^+ is maximized. While One-sided EAN+ contains One-sided EAN as a special case, it does give us some flexibility in designing an algorithm. First, we reformulate and relax the One-sided EAN+ problem as the maximum independent set (MIS) on a colored interval graph and hence reduce the appearance of each gene to at most two times. We show that this new relaxation is still NP-complete, though a simple factor-2 approximation algorithm can be designed; moreover, we also prove that the problem cannot be approximated within $2 - \epsilon$ by a local search technique. Second, we use integer linear programming (ILP) to solve this relaxed problem exactly. Finally, we compare our results with the up-to-date software GREDU, with various simulation

data.

Related Work

Comparing two generic genomes (with gene duplications) using different similarity measures, such as the exemplar genomic distance and the exemplar adjacency number, have been well studied in the past two decades. Sankoff first formulated the exemplar breakpoint distance problem as selecting exactly one gene from each gene family, such that the breakpoint distance between the two resulting exemplar genomes is minimized [90]. The exemplar adjacency number problem is the complement of the exemplar breakpoint distance problem [23, 24], which we will focus on in this chapter. Formally, the Exemplar Adjacency Number (EAN) problem is defined as follows: given two generic linear genomes $\mathcal{G}$ and $\mathcal{H}$ over the same set of n gene families, delete duplicated genes from them to obtain two exemplar genomes G and H each of length n, such that the number of adjacencies between G and H is maximized. The One-Sided EAN problem is the one when $\mathcal{H}(= H)$ is given exemplar.

In theory, not only computing the exemplar breakpoint distance is NP-hard [22], even computing any approximation is NP-hard when each gene appears in $\mathcal{G}$ and $\mathcal{H}$ at most three times [24] or twice [15, 60]. (They all showed that deciding whether the optimal exemplar breakpoint distance is zero, i.e, whether $G = H$, is NP-complete. Hence, computing any approximation solution is NP-hard.) Nonetheless, several algorithms have been proposed. Sankoff presented a branch-and-bound approach for the exemplar breakpoint distance problem [90]. Nguyen *et al.* gave a more efficient divide-and-conquer algorithm for the exemplar breakpoint distance problem [80]. Angibaud *et al.* developed an integer linear programming method for computing the exact breakpoint distance [5]. Shao and Moret gave a slight different formulation for the exemplar breakpoint distance problem where not all genes need to appear in

the resulting (reduced) genomes, and they also proposed a fast and exact algorithm using integer linear programming [95].

The Exemplar Adjacency Number problem (EAN) is the complement of the breakpoint distance problem. Chen *et al.* showed that if one of the genomes $\mathcal{G}$ and $\mathcal{H}$ is exemplar and the other is 2-repetitive (each gene from each gene family appears at most twice), then this One-sided EAN problem admits neither a polynomial-time factor-$n^{0.5-\varepsilon}$ approximation (where n is the size of the gene family) unless $P = NP$, nor an FPT algorithm unless FPT=W[1] [22, 23]. Chen *et al.* also designed a factor-$n^{0.5}$ approximation algorithm for the EAN problem when each gene appears at most twice in both $\mathcal{G}$ and $\mathcal{H}$ [22]. Because of the hardness result of the EAN problem (especially that a gene in each gene family must appear in G), it is not convenient to design efficient algorithms.

In this chapter, we focus on the One-sided EAN problem whose input is $\mathcal{G}$ and H, over the same set of n gene families. We first relax the problem by allowing the reduced (pseudo-exemplar) genome G^+, obtained from $\mathcal{G}$ by deleting duplicated genes, to have a length less than n (but each gene can still appear most once), the objective is to maximize the number of adjacencies between G^+ and H. We call this problem One-sided EAN+, which is a general version of One-sided EAN and at least as hard as the latter. (The two problems are not exactly the same, e.g., $\mathcal{G} = 153642$ and $H = 123456$, with the latter there is no adjacency while in the former we could delete 3 to obtain an adjacency 56.)

On the other hand, One-sided EAN+ does allow a new relaxation of the problem as the maximum independent set (MIS) on new variants of interval graph for linear genomes (where the intervals correspond to the potential adjacency ab or $-b-a$ in $\mathcal{G}$ are given the same color). While MIS on interval graphs are known to be polynomial-time solvable [48], in this chapter, we show that the MIS problem on interval graphs

becomes NP-complete when the vertices are colored. On the other hand, a simple factor-2 approximation can be obtained.

The above MIS approximation gives us a reduced/relaxed instance for the One-sided EAN+ problem. Let $\mathcal{G}'$ be the reduced instance for $\mathcal{G}$, obtained through the approximate MIS solution. $\mathcal{G}'$ has the property that each gene appears at most twice in $\mathcal{G}'$ (but some gene might not occur in $\mathcal{G}'$ at all). The next step is to use integer linear programming (ILP) to compute the exemplar adjacency between the pseudo-exemplar genome G^+ (obtained from $\mathcal{G}'$) and H. (Note that we never alter H, which is different from that of [95].)

We implement the algorithm and use simulated data (generated in the same way as in [95]. The comparison with GREDU indicates that our algorithm is more stable and generate comparable number of adjacencies (though the definitions of the adjacencies differ a little).

Preliminaries

Given n gene families (alphabet) $\mathcal{F}$, a genome $\mathcal{G}$ is a sequence of elements of $\mathcal{F}$ such that each element is with a sign (+ or -). Given a genome with a gene in each family appearing exactly once (which is called *exemplar*), $G = g_1 g_2 \cdots g_n$, we say that gene g_i immediately precedes g_j, if $j = i + 1$. Given two exemplar genomes G,H, if gene a immediately precedes b in G and neither a immediately precedes b nor $-b$ immediately precedes $-a$ in H, then they constitute a breakpoint in G. The breakpoint distance is the number of breakpoints in G or H, denoted as $bd(G, H)$. Similarly, given genomes G,H with no gene duplications, if gene a immediately precedes b in G and either a immediately precedes b or $-b$ immediately precedes $-a$ in H, then they constitute an *adjacency* in G. The adjacency number is the number of adjacencies in G or H, denoted as $an(G, H)$. In the exemplar genomes G,H, we have

$bd(G,H) + an(G,H) = n - 1$. For example, let $G = 12345, H = -5 - 4312$, then there are two adjacencies and two breakpoints between G and H.

Given generic genomes $\mathcal{G}$ and $\mathcal{H}$ the exemplar breakpoint distance between them, denoted as $ebd(\mathcal{G},\mathcal{H})$, is the minimum breakpoint distance $bd(G,H)$ where G, H are exemplar genomes obtained from $\mathcal{G}$ and $\mathcal{H}$ respectively. Similarly, the exemplar adjacency number between $\mathcal{G}$ and $\mathcal{H}$, denoted as $ean(\mathcal{G},\mathcal{H})$, is the maximum adjacency number $an(G,H)$, where G, H are exemplar genomes obtained from $\mathcal{G}$ and $\mathcal{H}$ respectively. Again, we have $ebd(\mathcal{G},\mathcal{H}) + ean(\mathcal{G},\mathcal{H}) = n - 1$.

The problem of computing $ean(\mathcal{G},\mathcal{H})$ is formally defined as the *Exemplar Adjacency Number* (*EAN*) problem. When one of $\mathcal{G},\mathcal{H}$ is given exemplar, we call the corresponding problem *One-sided EAN* problem. Throughout this chapter, we assume that $\mathcal{H}$ is given exemplar and we use H instead of $\mathcal{H}$ henceforth. A genome G^+ with length at least k ($k \leq n$), obtained from $\mathcal{G}$ by deleting duplicated genes (but each gene cannot appear more than once in G^+), is called a *pseudo-exemplar* genome. (Note that the definition of breakpoints and adjacencies between pseudo-exemplar genomes are the same as between exemplar genomes.) Given k, the problem of computing G^+ from $\mathcal{G}$ such that the number of adjacencies between G^+ and H is maximized, is hence called the *One-sided EAN+* problem. (One-sided EAN+ contains One-sided EAN as a special case when $k = n$; hence it is at least as hard as the latter.)

Finally, an interval graph $\mathcal{I} = (V, E)$ is a graph whose vertices have an one-to-one correspondence to a set of intervals on a line. An interval $u = (l(u), r(u))$ is represented by its left endpoint $l(u)$ and the right endpoint $r(u)$. There is an edge between two vertices $u, v \in V$ iff the intervals have a non-empty intersection. See Fig. 5.1 for an example.

Starting in the next section, we present a new method for the One-sided EAN+ problem. We start with a different relaxation/formulation of the problem.

A New Formulation of the Problem

Assume that we are given a generic linear genome $\mathcal{G}$ and an exemplar linear genome H, over the same gene family. Firstly, we try to identify some disjoint intervals in $\mathcal{G}$, one for each color (corresponding to some 2-substring in H). These intervals are the vertices of the corresponding colored interval graph $\mathcal{I}$. Then we try to identify the maximum number of disjoint intervals in $\mathcal{G}$, each of a different color. Clearly they correspond to a maximum independent set in $\mathcal{I}$. These intervals are formally constructed as follows.

For each 2-substring $a_i a_{i+1}$ in H, we list all the minimal intervals $a_i \beta a_{i+1}$ or $-a_{i+1}\beta - a_i$ (for unsigned genomes $a_{i+1}\beta a_i$) in $\mathcal{G}$ such that the contents β could be deleted to have a potential adjacency $a_i a_{i+1}$ or $-a_{i+1} - a_i$ ($a_{i+1}a_i$ for unsigned genomes). Note that a substring $x\beta y$ is *minimal* if the substring β does not contain x, or y, or a subsequence (potential adjacency) $-y - x$. All these minimal intervals in $\mathcal{G}$ corresponding to $a_i a_{i+1}$ in H will be given the same color. See Fig. 5.1 for an example of this construction with a colored interval graph where an unsigned genome is used.

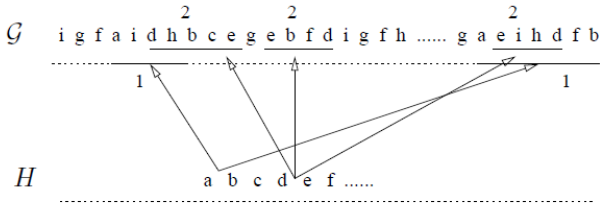


Figure 5.1: Formulation of the one-sided EAN problem potentially as MIS in a colored interval graph. Note that in $\mathcal{G}$ the two intervals labeled with color 1 correspond to the adjacency *ab*, while the three intervals labeled with color 2 correspond to the adjacency *de*.

We define the Maximum Independent Set (MIS) problem in a Colored Interval

Graph (MIS-CIG for short) as follows.

Problem: MIS-CIG

Instance: A set $\mathcal{I}$ of m intervals on a line, each is in one of the $k(k < m)$ colors, and a parameter $k_1 \leq k$.

Question: Are there k_1 disjoint intervals of different colors?

We show next that MIS-CIG is NP-complete.

Hardness of MIS-CIG

Theorem 7. *The decision version of the MIS-CIG problem is NP-complete.*

Proof. It is not hard to see that the decision version of MIS-CIG is in NP. We reduce the classic 3SAT problem to MIS-CIG to prove that it is NP-complete. Given a Boolean formula ϕ in 3-conjunctive normal form, $\phi = F_1 \wedge F_2 \wedge \cdots \wedge F_\ell$, where each of the ℓ clauses F_i has three distinct literals from a set of m boolean variables $x_1, x_2, \cdots, x_m$, the problem is to decide whether ϕ is satisfiable.

For each variable x_i and its negation $\bar{x}_i$, we construct two copies of interleaving intervals of four different colors $4i-3, 4i-2, 4i-1, 4i$. (See Figure 5.2.) If x_i is assigned TRUE, we select $4i-3, 4i-1$ at the top-right corner and $4i-2, 4i$ at the bottom-left corner. If x_i is assigned FALSE, we select $4i-3, 4i-1$ at the top-left corner and $4i-2, 4i$ at the bottom-right corner. For each clause F_j, we create three very small intervals of the same color (F_j). We put the interval with color F_j under the interval $4i-3$ corresponding to x_i if x_i appears in F_j, and we put the interval with color F_j under the interval $4i-3$ of $\bar{x}_i$ if $\bar{x}_j$ appears in F_j. We give an example to illustrate the construction. Assume we have a 3SAT formula $\phi = F_1 \wedge F_2 \wedge F_3 \wedge F_4$, with $F_1 = (x_1 \vee \bar{x}_2 \vee x_3), F_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_4), F_3 = (\bar{x}_2 \vee \bar{x}_3 \vee x_4)$, and $F_4 = (x_1 \vee \bar{x}_3 \vee \bar{x}_4)$. In Fig. 5.2, we show the construction on x_1, x_2 as the remaining construction for x_3, x_4 is similar.

We show that ϕ is satisfiable if and only if the colored interval graph has an IS of $4m + l$ intervals (all with different colors).

" $\rightarrow$ " Suppose that the 3SAT instance ϕ is satisfiable. If x_i is assigned TRUE, we select the top-right intervals $4i-3, 4i-1$, the bottom-left intervals $4i-2, 4i$ and all other F_j intervals in the bottom row under the unselected interval $4i-3$ as part of the MIS. (If multiple F_j intervals are selected, we just arbitrarily keep one of them.) If x_i is assigned FALSE, we select the top-left intervals $4i-3, 4i-1$, the bottom-right intervals $4i-2, 4i$ and all other F_j intervals in the bottom row under the unselected interval $4i-3$ as part of the MIS. Consequently, we obtain an MIS with $4m + \ell$ intervals (all with different colors).

" $\leftarrow$ " Suppose that the colored interval graph contains an independent set of $4m + \ell$ intervals, all with different colors. For the two groups of interleaving intervals $4i-3, 4i-2, 4i-1$ and $4i$ (corresponding to x_i and $\bar{x}_i$ respectively), clearly we could only select them in two different ways. We could either select the top-right $4i-3, 4i-1$ and the bottom-left $4i-2, 4i$ intervals, we could select the top-left $4i-3, 4i-1$ and the bottom-right $4i-2, 4i$ intervals. In the former case, we assign x_i TRUE, and then the F_j intervals containing x_i will be in the independent set as the top-left interval $4i-3$, which intersects all the F_j intervals containing x_i, is not selected. In this case, F_j evaluates to TRUE and is satisfied. Similarly, we could show F_j containing $\bar{x}_i$ is also satisfied if we select the top-left $4i-3, 4i-1$ and the bottom-right $4i-2, 4i$ intervals, which corresponds to assigning x_i FALSE. Then, we have a truth assignment for all the variables such that each F_j in ϕ is satisfied.

The reduction takes $O(m + \ell)$ time. Hence the theorem is proven. $\square$

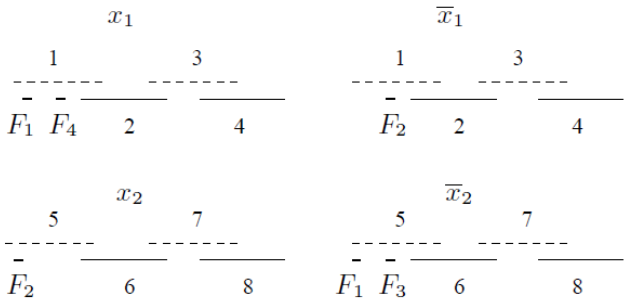


Figure 5.2: Illustration for the reduction from the 3SAT instance ϕ. To save space, only the construction for $x_1, \overline{x}_1$ and $x_2, \overline{x}_2$ are shown.

Algorithms

In this section, we design an efficient factor-2 approximation algorithm for the MIS-CIG problem. Moreover, we prove that the approximation factor cannot be improved to be less than 2 by some local search technique. This gives us a reduced instance $\mathcal{G}'$ where each gene appears at most twice. On top of this, we use integer linear programming to efficiently delete extra gene duplications while computing the maximum number of adjacencies.

A Factor 2-Approximation

A factor-2 approximation for the MIS-CIG can be obtained using the well-known greedy method. Firstly, the intervals are ordered by their monotonically increasing right endpoints. Secondly, we scan the intervals from left to right and put the first interval I_1 in the solution. Then we delete all the intervals of the same color or overlapping with I_1 and repeat this process.

It is straightforward to see that the algorithm returns a factor-2 approximation for MIS-CIG. Let $I_i \in S$ and let I_j be some interval which intersects $r(I_i)$ (the right endpoint of I_i). As the algorithm scans intervals from left to right, any optimal

solution not containing I_i must either contain an interval of the same color with I_i, or contain an interval of the same color with I_j (inclusive of I_j), or both. Hence the approximation algorithm would return at least a half of the optimal solution associated with $r(I_i)$. Applying this argument recursively would give us a factor-2 approximation.

1: $m \leftarrow$ size of $\mathcal{I}$.
2: $S \leftarrow \emptyset$.
3: Sort all intervals in $\mathcal{I}$ according to their right endpoints as $I_1, ..., I_m$.
4: Delete from $\mathcal{I}$ all the intervals of the same color with I_1 or overlapping with I_1.
5: Delete I_1 from $\mathcal{I}$.
6: $S \leftarrow S \cup \{I_1\}$.
7: **while** $\mathcal{I} \neq \emptyset$
8: $\quad I_t \leftarrow$ the first interval in $\mathcal{I}$.
9: $\quad S \leftarrow S \cup \{I_t\}$.
10: $\quad$ Delete all the intervals of the same color with I_t or overlapping with I_t.
11: $\quad$ Delete I_t from $\mathcal{I}$.
12: Return S.

Algorithm 5.8: A Factor-2 Approximation Algorithm for MIS-CIG

A Local Search Improvement

After obtaining the greedy algorithm, it is easy to come up with an improvement using a local search method. Let $\mathcal{I}$ be the input intervals for the MIS-CIG problem. We search for a subset of c intervals in S, S_c, such that putting S_c back to $\mathcal{I} \backslash S$ enables us to find locally a subset S'_c of more than c independent intervals with different colors and all the colors in S'_c do not appear in $S \backslash S_c$. Then $(S \backslash S_c) \cup S'_c$ would give us a

better solution than S and we update $S \leftarrow (S \backslash S_c) \cup S'_c$. This process will continue until no S_c can be found. For a constant c, we call the corresponding local search procedure *c-local search*. Unfortunately, for $c = 1, 2$, this local search method cannot improve the result in the worst case. We summarize the result as follows.

Theorem 8. *Algorithm 5.8 returns a 2-approximation for MIS-CIG. For $c = 1, 2$ and some constant ε, the solution obtained by Algorithm 5.8 cannot be improved to have an approximation factor smaller than $2 - \varepsilon$ using c-local search.*

Proof. Assume to the contrary that with c-local search, $c = 1, 2$, we could obtain a $2-\varepsilon$ approximation for the MIS-CIG problem. We construct an instance with $3n+2$ intervals, with $2n$ colors $\{0, 1, ..., 2n-1\}$. The greedy algorithm would select the intervals at the bottom row with colors $0, 1, 2, ..., n$. The optimal solution is to select all the intervals in the top row (except the last interval with color 0), with colors $0, 1, ..., 2n-2, 2n-1$. When context is clear, we just call an interval with color i interval i.

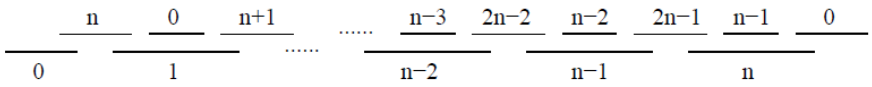


Figure 5.3: Construction of an instance with $3n + 2$ intervals, with $2n$ colors $\{0, 1, ..., 2n-1\}$. Illustration for the proof of Theorem 8 by swapping intervals in bottom row with intervals in top row.

To see that c-local search does not incur any improvement for $c = 1, 2$, it is important to notice that any interval i ($i > 0$) at the bottom row contains an interval $i-1$ at the top row; moreover, the two neighboring intervals of interval $i-1$ at the top row, $n+i-1$ and $(n+i) \mod 2n$, are both intersecting with the corresponding neighboring intervals of i at the bottom row. (For interval 0 at the bottom row, it

intersects interval n at the top row.) Hence, for 1-local search, it is impossible to obtain a better result by swapping interval i at the bottom row with two intervals in new colors at the top row. Similarly, for 2-local search, it is impossible to obtain a better result by swapping two intervals i, j at the bottom row with three intervals in new colors at the top row.

For this instance, the approximation factor is

$$\frac{2n}{n+1} = 2 - \frac{2}{n+1} > 2 - \varepsilon,$$

when n is sufficiently large at $n > \frac{2-\varepsilon}{\varepsilon}$. Hence, the approximation factor of the greedy algorithm is tight, whether or not c-local search, $c = 1, 2$, are applied. □

Note that when $c \geq 3$, the above argument does not work anymore. On the other hand, a local search with $c \geq 3$ could incur high cost. Hence, after a reduced genome $\mathcal{G}'$ is obtained through the MIS approximation, we would use integer linear programming (ILP) instead.

ILP Formulation on the Reduced Instance

The genes in the reduced genome $\mathcal{G}'$ occur at most twice after using the MIS approximation in a colored interval graph on the original generic genome $\mathcal{G}$. The reason is that each gene could appear in two intervals, e.g., $(i-1, i)$ and $(i, i+1)$. Moreover, there is no guarantee that a gene in each gene family appears in $\mathcal{G}'$. Hence from $\mathcal{G}'$ we could only hope to compute a pseudo-exemplar genome (i.e., each gene appears at most once, some might never appear). In this subsection, we use integer linear programming (ILP) on the reduced genome to get a pseudo-exemplar genome based on the property that every pair of genes appear in $\mathcal{G}'$ in an adjacent form already, e.g., $(i, i+1)$ or $(i+1, i)$. We keep the order of genes in the intervals

resulting from the greedy method. For example, $\mathcal{G}' = \boxed{67}\boxed{45}\boxed{32}\boxed{56}\boxed{12}\boxed{34}$.

Let x_{ij}^k denote the kth interval with content (or simply color) ij with $|j-i| = 1$. Note that for the ease of description here we use unsigned genes, though we have implemented both the signed and unsigned versions. As i, j each could appear twice in $\mathcal{G}'$ we have $k \leq 4$, i.e., we could have at most 4 different ways to form the adjacency (i, j). Here, $j = i - 1$ or $j = i + 1$. We comment that we never alter H, while using the algorithm in [95] H could be altered. An example is as follows, $\mathcal{G} = 128456573$, $\mathcal{H} = 12345678$. With the algorithm in [95], the optimal solution is $G = H = 124567$, with five adjacencies. With our problem $\mathcal{H}$ is already exemplar so we cannot alter it, and the optimal solution is $G = 12845673$ and $H = 12345678$, with only four adjacencies. Moreover, our optimal solution could be some pseudo-exemplar genomes $G' = 1245673$ or $G' = 124567$.

In $\mathcal{G}'$, there are two genes from each gene family. Let y_{ij} $(j = 1, 2)$ represent each gene from its gene family y_i. Two adjacent pairs $(i, i+1), (i+1, i+2)$ are called *neighboring* pairs. and two adjacent pairs $(i, i+1), (i+2, i+3)$ are called *consecutive* pairs. (Note that one or both of them could be in reverse order.) If these pairs are all sorted, then we call them *sorted-neighboring* and *sorted-consecutive* respectively. If two adjacent pairs have other pairs separating them, say $\boxed{45}\boxed{89}\boxed{67}$, we define them as *disjoint-neighboring* pairs or *disjoint-consecutive* pairs — depending on whether they share a gene or not. A *block* is composed of a sequence of neighboring or consecutive pairs. For example, $\boxed{12}\boxed{23}\boxed{45}\boxed{56}\boxed{76}\boxed{89}$. Our objective function is to maximize the number of adjacencies.

$$\max \sum_{i,j} x_{ij}^k.$$

Our idea on applying ILP on the reduced instance is as follows. We identify all

the blocks and try to locally solve them using ILP. Then these local solutions are put together to have a final solution. We give more details as follows. First, we perform some preprocessing by computing maximal common substrings in $\mathcal{G}'$ (with respect to H), keep the longest ones (when there is a conflict) and delete all the corresponding duplicated genes. Second, identify all the blocks and apply five groups of integer linear programming constraints: overlapping, same color, adjacency, non-adjacency and exemplar constraints.

Preprocessing:

1. We compute all maximal common (normal or reversed) substrings between $\mathcal{G}'$ and H and keep those with no conflict. For example, $\boxed{45}\boxed{67}$ is kept. If some conflicting substring appear in different such maximal substrings, we keep all the genes in the longest substrings. We then delete the corresponding duplicated genes. Example 1 below is a good example. (Example 1. $\mathcal{G}' = \boxed{7654321234}$ with 234 and 432 cause a conflict. If we delete $\boxed{432}$ from $\boxed{765432}$, we get $\boxed{7651234}$. But the optimal solution is $\boxed{7654321}$ by deleting $\boxed{234}$ in $\boxed{1234}$.)

2. If there is a pair between two disjoint-consecutive pairs, we keep the consecutive pairs and delete the pair in the middle. For example, we keep $\boxed{45}\boxed{67}$ from $\boxed{45}\boxed{89}\boxed{67}$, while $\boxed{89}$ is deleted.

3. For two sorted-neighboring pairs, we delete one gene which appears in the middle. For example in $\boxed{45}\boxed{56}$, we delete $\boxed{5}$.

The following lemma can be proved.

Lemma 12. *The preprocessing procedure cannot reduce the number of adjacencies in the optimal solution.*

We next illustrate all the five groups of constraints.

Adjacency constraints: Identify blocks of neighboring and consecutive pairs of genes and the size of its content (i.e., number of distinct genes). In each block, we apply some adjacency constraint. For example, $\boxed{56}\boxed{76}\boxed{87}$ with four distinct genes. In this case, it can form at most 3 adjacencies using the following adjacency constraint.

$$x_{56}^{i_1} + x_{56}^{i_2} + x_{67}^{j_1} + x_{67}^{j_2} + x_{67}^{j_3} + x_{78}^{k_1} + x_{78}^{k_2} \leq 3 \tag{5.1}$$

Non-adjacency constraints: For every pair $(i, i+1)$ find the disjoint-neighboring pair $(i+1, i+2)$ in other blocks such that there are more than two genes between them. In that case, one of them could be selected. Likewise, for two disjoint-consecutive pairs $(i, i+1), (i+2, i+3)$ in different blocks with more than two genes between them, both of them could be selected. The corresponding constraints are as follows. Here $1 \leq p, q, r, s \leq 4$.

$$x_{i,i+1}^{p} + x_{i+1,i+2}^{q} \leq 1 \tag{5.2}$$

$$x_{i,i+1}^{r} + x_{i+2,i+3}^{s} \leq 2 \tag{5.3}$$

Overlapping constraints: For a set of overlapping intervals, at most one could be selected. For example, for a substring of $\mathcal{G}'$, 457689. We have three overlapping intervals and the constraint is as follows.

$$x_{56}^{p} + x_{67}^{q} + x_{78}^{r} \leq 1 \tag{5.4}$$

Same color constraints: Among identical intervals, only one of them could

be selected. Note that these intervals cannot all come from the MIS solution. For example, the optimal solution in $\boxed{67}\boxed{45}\boxed{65}\boxed{78}$ is $\boxed{45678}$. We use the same color constraint to delete the first interval $\boxed{67}$:

$$x_{67}^{p} + x_{67}^{q} + x_{67}^{r} + x_{67}^{s} \leq 1 \tag{5.5}$$

Exemplar constraints: It is require that at most one gene appears from each family in the final solution.

$$y_{11} + y_{12} \leq 1 \tag{5.6}$$

$$y_{21} + y_{22} \leq 1 \tag{5.7}$$

$$\vdots \tag{5.8}$$

$$y_{n1} + y_{n2} \leq 1 \tag{5.9}$$

Finally, if the variable x_{ij}^{k} that represents an interval (formed by a pair of genes) is selected, we also require that the two variables y_{i,l_1} and y_{j,l_2}, which is the element of x_{ij}^{k}, be selected as well. If the gene y_{i,l_1} contributing to form the adjacency x_{ij}^{k} is not selected, x_{ij}^{k} should also be discarded. The constraints should be as follows.

$$y_{i,l_1} \geq x_{ij}^{k} \tag{5.10}$$

$$y_{j,l_2} \geq x_{ij}^{k} \tag{5.11}$$

Of course, x_{ij}^{k} and y_{ij} are all binary variables, representing every pair of genes

and every single gene in the reduced genome.

$$x_{ij}^{k} \in \{0,1\}(i = 1, 2, \cdots, n; j = 1, 2, \cdots, m) \tag{5.12}$$

$$y_{ij} \in \{0,1\}(i = 1, 2, \cdots, n; j = 1, 2) \tag{5.13}$$

Simulation Results

Our algorithm's simulation can be divided into two main components; namely, data generation and greedy exemplar selection written in Java, and the exemplar alignment written in Matlab using CPLEX for integer linear programming. (The GREDU software was written in C++ which is usually much faster, and the ILP package GUROBI was used.) We ran our approximation algorithm as well as GREDU on a PC with 2.5 GHz Intel Core processor and 4 GB of memory.

Our simulated data is generated using a method similar to that in [95]. The dataset generator builds an exemplar genome, H, of size n, comprising of an integer sequence $[1..n]$ representing n unique genes. This exemplar genome is then mutated M times to produce some genome, $\mathcal{G}$, where M is a user provided integer that roughly corresponds to the number of generations between H to $\mathcal{G}$. A mutation cycle on a genome is performed by traversing each gene g_k in the genome, and mutating g_k with some probability. Our generator can perform up to eight different mutations:

- Unit Reversal : Given a gene, g_k, switch the locations of g_k and g_{k+l} with a probability of p_1.
- Unit Insertion : An arbitrary gene is inserted at location k with a probability of p_2.
- Unit Deletion : The gene, g_k, is removed with a probability of p_3.

- Unit Duplication : The gene, g_k, is copied and then inserted at location $k+1$ in the genome with a probability of p_4.

- Segment Reversal : Given some length, l, and a gene, g_k, the ordering of the genome between genes g_k and g_{k+l} is reversed with a probability of p_5.

- Tandem Duplication : Given some length, l, and a gene, g_k, the genes between g_k and g_{k+l} are copied and placed at location $k+l+1$ with a probability of p_6.

- Segment Deletion : Given some length, l, and a gene, g_k, the genes between g_k and g_{k+l} are removed with a probability of p_7.

- Segment Duplication : Given some length, l, and a gene, g_k, the sequence of the genome between g_k and g_{k+l} is copied and then inserted at a random location in the genome with a probability of p_8.

Every mutation cycle is performed on the mutated genome from the last mutation cycle, until m cycles have completed. Both the exemplar genome H and the genome $\mathcal{G}$ are then written to disk. These exemplar-genome pairs are then used as test datasets for both our greedy algorithm and integer linear programming, as well as for the GREDU software that we compare performance against.

In our simulations we use three different settings for the probabilities, P_1, P_2, and P_3.

- $P_1 = \{p_1 = 0.03, p_2 = 0.03, p_3 = 0.06, p_4 = 0.03, p_5 = 0.00, p_6 = 0.00, p_7 = 0.00, p_8 = 0.00, l = 0\}$

- $P_2 = \{p_1 = 0.20, p_2 = 0.15, p_3 = 0.15, p_4 = 0.15, p_5 = 0.12, p_6 = 0.04, p_7 = 0.08, p_8 = 0.05, l = 5\}$

- $P_3 = \{p_1 = 0.15, p_2 = 0.10, p_3 = 0.18, p_4 = 0.10, p_5 = 0.10, p_6 = 0.05, p_7 = 0.09, p_8 = 0.05, l = 10\}$

While these cases are not exhaustive in regards to the coverage of genome generation free variables, they do provide the three unique cases of genomes that we are interested in for our comparison of our algorithm and GREDU. The first case, P_1 is designed to not mutate aggressively, and is designed to change slowly. This means that most of the gene families will still be in the same ordering as the exemplar, and many alignment pairs should be found. The second case, P_2, is designed for rapid mutations. Higher mutations rates coupled with a small l value mean that the genome that is generated will likely have a vastly different relative ordering of gene families compared to the exemplar, and few pairs should be found. The last case, P_3, is somewhat of a mixing between P_1 and P_2. The mutation rates are still set relatively high, but the value of l is also increased to 10. This means that while the genome is changing rapidly, large sections of it will also be moved and copied without changing the local ordering. We do not expect many adjacencies to be found, but we do expect slightly better results that from P_2.

Although the platforms (even the adjacency definitions) between our implementation and GREDU are quite different, we compare them using the same simulated data in Table 5.1. All the numbers are averaged over 10 tries. Even though that GREDU is much faster as it is written in C++, our implementation, which is based on Java and Matlab, does not need more than 20 minutes for any case we tested (which should be fine with this application). However, for many P_3 datasets, GREDU cannot run to completion. Our implementation is much stable and the number of adjacencies computed do not differ too much between the two (even though the definitions of adjacencies differ a bit). We also list our result for unsigned genomes in Table 5.2, note that GREDU only handles signed genomes.

Table 5.1: Comparison results (signed genome) between the number of shared adjacencies n_1 from our algorithms and n_2 from GREDU. The gap _ indicate that we get a core dumped warning from GREDU and can not see the results after 10 tries.

N	M	P_1		P_2		P_3	
		n_1	n_2	n_1	n_2	n_1	n_2
	1	247	242	216	224	204	198
500	3	87	41	34	34	19	20
	5	17	12	23	8	8	2
	1	512	481	391	406	301	297
1000	3	106	96	66	61	54	47
	5	24	23	33	29	23	18
	1	1552	1442	1231	1244	703	756
3000	3	360	330	232	222	171	164
	5	80	78	97	73	64	50
	1	2441	2307	2044	2067	1532	_
5000	3	658	568	439	397	204	196
	5	163	159	272	197	110	91
	1	3471	3433	2495	2525	2010	_
7000	3	823	740	630	512	515	409
	5	223	208	249	139	151	128
	1	4563	4296	3507	3588	3355	_
9000	3	1103	969	933	828	583	_
	5	257	249	276	154	177	152
	1	6183	5756	5217	5322	3423	_
12000	3	1383	1287	1247	1112	279	237
	5	365	332	379	239	257	205

Table 5.2: The running time t_1 (seconds) and the number of adjacencies n_1 for unsigned genomes, averaged over 10 tries. No comparison is done with GREDU as it only handles signed genomes.

N	M	$P_1(l=5)$		$P_2(l=1)$		$P_3(l=10)$	
		t_1	n_1	t_1	n_1	t_1	n_1
	1	0	237	0	223	0	207
500	3	0	86	0	96	0	82
	5	0	20	0	43	0	27
	1	0	497	0	425	0	377
1000	3	0	116	0	165	0	133
	5	0	41	1	82	0	49
	1	29	1450	30	1281	22	1189
3000	3	6	398	27	467	2.5	434
	5	3	127	33	207	0	181
	1	127	2214	142	2181	88	2025
5000	3	30	662	105	726	15	431
	5	8	159	145	328	2	178
	1	368	3317	398	3097	260	2694
7000	3	70	768	326	998	30	614
	5	26	263	435	453	10	235
	1	765	4368	868	4179	446	3650
9000	3	207	985	799	1245	80	857
	5	64	310	879	632	25	345
	1	1691	5789	1877	5564	1006	4822
12000	3	409	1321	1521	1532	158	1017
	5	120	465	1945	913	58	397

Concluding Remarks

We study the One-sided EAN problem by considering a general version of it, One-sided EAN+. We approach the One-sided EAN+ problem by first relaxing it as the maximum independent set in a colored interval graph, which open a new research pipeline to deal with the exemplar genomic distance problems for linear genomes. Our greedy algorithm and integer linear programming can handle large scale genomic (singed and unsigned) data with deep evolution size based on the simulation results.

For the future work along this line, we can formulate the problem as MIS in a colored 2-interval graph for two generic linear genomes with duplications. We can also define similarly MIS in a colored circular-arc graph for two generic circular genomes. Note that the greedy algorithm cannot produce a 2-approximation in these settings.

THE MINIMUM COPY NUMBER GENERATION PROBLEM

In cancer genomics, due to the fast somatic mutations (mainly random segment duplications and deletions), *copy number profiles (CNPs)* (i.e., a file containing each of the gene numbers) are used more often than genome themselves. On the other hand, algorithms with performance analysis for processing CNPs are lacking. In a recent CPM'16 paper, Shamir *et al.* studied the *copy number transformation* problem, which is to use the minimum number of duplications and deletions (on the CNPs) to convert one CNP to another, and gave a linear time algorithm. In this chapter, we consider a slightly different problem which is called *Minimum Copy Number Generation (MCNG)*, namely, given a genome G and a specific CNP C, use the minimum number of duplications and deletions on G to obtain some genome H which has a CNP C. We show that the problem is NP-complete if G is generic (i.e., contains duplicated genes) and when the duplications are tandem. On the other hand, when only tandem duplications are allowed, if G is exemplar (or is a permutation) and all components in C are power of two's, then the problem can be solved in time linear in the length of the input (or $|C|$) plus $O(|G| \log |G]|)$ (the cost for sorting $|G|$ elements). That naturally extends to a practical heuristic algorithm for the problem (when G is exemplar and the components in C are arbitrary). We also show that two variations of the MCNG problem are at least as hard as Set Cover in terms of approximability and FPT tractability. For the general Minimum Copy Number Generation problem, i.e., when both (arbitrary) segment duplications and deletions are allowed, we also design a practical greedy algorithm, present some non-trivial cases.

Related Work

In cancer genomic research, a central problem is the investigation of intra-tumor genetic heterogeneity (or simply heterogeneity) [75, 77, 92], which could potentially help cancer prognostics [30, 73] and explain drug resistance [31, 33]. In some cancers, like high-grade serous ovarian cancer (HGSOC), the heterogeneity is mainly reflected in genomic rearrangement and endoreduplications, which lead to aberrant *copy number profiles (CNPs)* [79]. In this case it is difficult to produce a valid evolution tree (or phylogeny) based on the genomic data; instead, this must be done directly by using the copy number profiles. In [91], a method was proposed to infer phylogenetic trees from unsigned integer copy number profiles. In this method, a fundamental problem is to compute the minimum number of duplications/deletions to convert a CNP to another one.

Let G be a genome containing m genes $\{g_1, g_2, ..., g_m\}$, with $|G| = n \geq m$. The *copy number profile* (CNP for short) for G, denoted as $cnp(G)$, is an m-vector $cnp(G) = \langle c_1, c_2, ..., c_m \rangle$ where $c_i \geq 0$ is the number of copies of gene g_i in G, i.e., $c_i = cn(g_i)$. (Biologically c_i is in fact related to the SNP information, but for our purpose it is good enough to simply use the number of copies of gene g_i.) Under many situations, if G is not given, then we simply denote a copy number profile as $cnp = \langle c_1, c_2, ..., c_m \rangle$ or $C = \langle c_1, c_2, ..., c_m \rangle$. If the underline set of genes is known from the context, even when an incomplete genome or a fragment of G, G', is in discussion, throughout this chapter, we represent $cnp(G')$ as an m-vector. For instance, the set of genes in consideration is $\{a, b, c, d\}$, $G' = dbdbd$, then we have $cnp(G') = \langle 0, 2, 0, 3 \rangle$.

In [91], the evolution of copy number profiles is modeled through a sequence of operations that increase or decrease copy numbers by one. These operations are defined as follows. Given a copy number profile $C = \langle c_1, c_2, ..., c_m \rangle$, a segment

duplication is to change a substring of C, $\langle c_i, c_{i+1}, ..., c_j \rangle$, into $\langle c_i+1, c_{i+1}+1, ..., c_j+1 \rangle$. Likewise, a segment *deletion* is to change a substring of C, $\langle c_i, c_{i+1}, ..., c_j \rangle$, into $\langle c_i - 1, c_{i+1} - 1, ..., c_j - 1 \rangle$.

In [93], this problem was formally defined as the Copy Number Transformation (CNT) problem: given two copy number profiles (CNP's) $C_1 = \langle c_1, c_2, ..., c_m \rangle$ and $C_2 = \langle d_1, d_2, ..., d_m \rangle$, use the minimum number of duplications and deletions to convert CNP C_1 into CNP C_2. We show an example in Fig.6.1 as follows.

Copy Number Profile	*Operations*
$cnp_1 = \langle 1, \underline{2, 2, 1, 2}, 1 \rangle$	*duplication* : $\langle 2, 2, 1, 2 \rangle$
$\langle 1, 3, \underline{3, 2, 3, 1} \rangle$	*duplication* : $\langle 3, 2, 3, 1 \rangle$
$\langle \underline{1, 3, 4}, 3, 4, 2 \rangle$	*deletion* : $\langle 1, 3, 4 \rangle$
$cnp_2 = \langle 0, 2, 3, 3, 4, 2 \rangle$	

Figure 6.1: An example for the Copy Number Transformation problem. With two duplications and one deletion, CNP C_1 is transformed into CNP C_2.

A linear-time dynamic programming algorithm was proposed for CNT in [93]. The problem we investigate is slightly different from CNT. In Fig.6.2, a simple example is shown to illustrate the difference between CNT and our problem. (In our problem, duplication/deletion operations are performed directly on the genomes, which cause the corresponding CNP's to change. The formal definition of the problem will follow.) In this example (see Figure 6.2), the third CNP $\langle 1, 3, 2, 3, 3, 1 \rangle$ cannot be obtained with a single duplication/deletion operation from the second CNP. However, in our model, it (and its corresponding genome) can be obtained by one duplication operation on the second genome corresponding to the second CNP $\langle 1, 2, 2, 2, 2, 1 \rangle$.

$GENOME$	$Copy\ Number\ Profile$
$\langle a, \underline{b, c, d, e}, f\rangle$	$\langle 1, 1, 1, 1, 1, 1\rangle$
$\langle a, b, c, \underline{d, e, b}, c, d, e, f\rangle$	$\langle 1, 2, 2, 2, 2, 1\rangle$
$\langle a, b, c, d, e, b, d, e, b, c, d, e, f\rangle$	$\langle 1, 3, 2, 3, 3, 1\rangle$

Figure 6.2: An example for converting a genome to the next one using a single duplication, with a total of two duplication operations. Notice that when the CNP changes from $\langle 1, 2, 2, 2, 2, 1\rangle$ to $\langle 1, 3, 2, 3, 3, 1\rangle$ the second 2 (the number of c's) does not change. In other words, this cannot be implemented with a single duplication/deletion operation on the CNP directly, while obviously with our model the third CNP (and the corresponding genome) can be obtained with one duplication from the genome associated with the second CNP $\langle 1, 2, 2, 2, 2, 1\rangle$.

With the above discussions, we believe that a lot of research are still needed to process CNPs. In fact, there has been some algorithmic research on processing CNPs in the recent years, for example in [27, 39, 81, 99, 103]. In the remainder of this chapter, we make an effort by considering the following fundamental problem: Given a genome $G = a_1a_2\cdots a_n$ (interchangeably, $G = \langle a_1, a_2, \cdots, a_n\rangle$) over a set of m genes $\{g_1, g_2, \cdots g_m\}$, with $n \geq m$ and $a_j \in \{g_1, g_2, ..., g_m\}$ for $j = 1..n$, and a copy number profile $CNP = \langle c_1, c_2, \cdots, c_m\rangle$, use the minimum number of duplications and deletions on G to generate a genome H such that $cnp(H) = CNP$. (Some of our theoretical results are obtained by restricting the duplications to be tandem, and/or disallowing deletions.) We call this the Minimum Copy Number Generation problem. The most general version, where both duplications and deletions are allowed, will be discussed at the end of this chapter.

We comment that there have been research on comparing genomes based on gene duplications and gene loss [54], and on rearrangement and duplications [94].

But they did not consider CNPs, hence the corresponding problems are also different from ours.

The Minimum Copy Number Generation (MCNG) Problem

Preliminaries

We briefly review some standard definitions. Given a gene set $\Sigma = \{g_1, g_2, ..., g_m\}$, a genome G is a sequence over Σ, with $|G| \geq m$. When G has length m and each gene in Σ appears exactly once (i.e., G is a permutation on Σ), we also say that G is an *exemplar* genome.

An Fixed-Parameter Tractable (FPT) algorithm is an algorithm for a decision problem with input size n and parameter k whose running time is $O(f(k)n^c) = O^*(f(k))$, where $f(-)$ is any computable function on k and c is a constant. FPT algorithms are efficient tools for handling some NP-complete problems as they introduce an extra dimension k. If an NP-complete problem, like Vertex Cover, admits an FPT algorithm, then it is basically polynomially solvable when the parameter k is a small constant [37].

Of course, it is well conceived that not all NP-hard problems admit FPT algorithms. It has been established that

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots W[t] \subseteq XP,$$

where XP represents the set of problem which must take $O(n^k)$ time to solve (i.e., not FPT). Typical problems in W[1] include Independent Set and Clique and typical problems in W[2] are Set Cover and Dominating Set. For the formal definitions and foundations, readers are referred to [37].

NP-completeness for MCNG When Duplications are Tandem

Given a sequence $S = s_1 s_2 \cdots s_n$, a segment *duplication* on S is to take a substring $S' = \underline{s_i s_{i+1} \cdots s_j}$, where $1 \leq i \leq j \leq n$ and insert it before or after some letter s_k with $k \leq i$ or $k \geq j$, i.e., after the duplication, the length of S is increased by $|j - i + 1|$ and the original content of S' is not altered. If S' is inserted right before s_i or right after s_j, then the duplication is *tandem*. While we use a lot of tandem duplications in many cases in this paper to obtain some theoretical results, the duplications in consideration at the end are not restricted to tandem duplications. (Likewise, a segment *deletion* is to delete S' from S, i.e., after the deletion, the length of S is decreased by $|j - i + 1|$.)

The Minimum Copy Number Generation (MCNG) is formally defined as follows.

INSTANCE: A genome (or a sequence) G of length n over a gene set $\{1, 2, ..., m\}$, a copy number profile C which is represented as an m-tuple with $C = \langle c_1, c_2, ..., c_m \rangle$, and, $c_i \geq 0$ for $i = 1..m$; and an integer k.

QUESTION: Is there a sequence of at most k duplications and deletions which can convert G into G' such that the copy number profile of G' is the same as C, i.e., $cnp(G') = C$?

We prove the following theorem for the case when the duplications are tandem and G is a general genome (possibly with gene duplications). When a genome is a permutation (or, it is *exemplar*) we use P to represent such a permutation (or exemplar genome).

Theorem 9. *The Minimum Copy Number Generation problem when duplications are tandem is NP-hard.*

Proof. We make a reduction from the NP-complete problem Exact Cover by 3-Sets (X3C) [47]. Recall that the input for X3C is a set of 3-sets $S = \{S_1, S_2, ..., S_m\}$. Each

set S_i contains exactly 3 elements from a base set $X = \{x_1, x_2, ..., x_n\}$, where $n = 3q$ for some integer q. The problem is to decide whether there are q 3-sets in S which cover each element in X exactly once.

G is constructed as follows. We use T_i to represent the 3-substring (block with length 3) containing the corresponding three elements in S_i, whose elements are ordered according to their indices in X.

$$G = \langle T_1, p_1, T_2, p_2, \cdots, T_{m-1}, p_{m-1}, T_m \rangle,$$

where p_i's are peg genes each appearing exactly once and are used as separators. Let $f(x_i)$ be the total number of times that x_i appears in the multiset $\cup_j S_j$. The copy numbers (cn for short) are constructed as follows: $cn(p_i) = 1$ for $i = 1..m-1$; $cn(x_i) = f(x_i) + 1$ for $i = 1..3q$.

It is straightforward to see that X3C has a solution then there are q tandem duplications on G resulting in a new sequence G' satisfies the copy number constraints. The arguments are relatively easy: due to $cn(g_i) = 1$ for $i = 1..m-1$, a sequence of tandem duplications occurring only on T_j's (corresponding to the X3C solution) will satisfy the copy number constraints. However, the reverse direction is more complex and is finished with the following two lemmas. The reduction takes linear time though. □

We show a simple example for the reduction. $X = \{1, 2, 3, 4, 5, 6, a, b, c, d, e, f\}$. $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 5, a\}$, $S_3 = \{4, 5, 6\}$, $S_4 = \{a, b, c\}$, $S_5 = \{4, b, e\}$, $S_6 = \{d, e, f\}$. Our construction is then

$$G = 123 \cdot p_1 \cdot 15a \cdot p_2 \cdot 456 \cdot p_3 \cdot abc \cdot p_4 \cdot 4be \cdot p_5 \cdot def.$$

The copy number profile C, with the first $3q$ components in the same order as listed in X, followed with those for p_j's, is

$$C = \langle 3, 2, 2, 3, 3, 2, 3, 3, 2, 2, 3, 2, 1, 1, 1, 1, 1 \rangle.$$

The 4 optimal tandem duplications are performed on 123, 456, abc and def. (But this optimal solution is not unique: we could make a whole genome duplication and then delete the three segments between 123, 456, abc and def to obtain a substring $123 \cdot 456 \cdot abc \cdot def$, before or after the original copy of G. This is handled next.)

We define a *canonical* duplication/deletion operation as one where a tandem duplication is applied on some block T_j in G, or p operations are applied on a substring of G containing p target blocks, $s = T_{i,1} \cdots T_{i,p}$. In the latter case, we first (tandem) duplicate s and then delete all the $p-1$ segments between $T_{i,j-1}$ and $T_{i,j}$ for $j = 2..p$.

Note that in both cases, it takes one tandem duplication or deletion, on average, to obtain a block T_j. Henceforth, we call each such duplication or deletion a canonical *copy+delete* operation. In the above example, the first kind of canonical operations is like $abc \rightarrow abc \cdot abc$. An example for the second is to make a whole substring of G starting at 456 then delete the 2 segments between 456 and abc, and between abc and def. The eventual string obtained is $456 \cdot abc \cdot def$, with a total of 3 canonical copy+delete operations.

Let G' be obtained from G with q canonical copy+delete operations while satisfying the copy number constraints. Let a substring s of $G' - G$, $|s| = 3k$, be the result of some canonical copy+delete operations where s is a list of some T_i's. Then we say that s is a *canonical* substring. A subsequence of $G' - G$ is *canonical* if it is composed of a list of canonical substrings of $G' - G$. We have the following lemma.

Lemma 13. *Let G'' be obtained from G with some tandem duplication or deletion*

operations with the copy number constraints satisfied, i.e., $cn(p_i) = 1$ *and* $cn(x_i) = f(x_i) + 1$. *Let* $t = G'' - G$, *with* $|t| = 3q$ *and* $cnp(t) = I_{3q}$ *(the unit vector of* $3q$ *1's). Then* t *is canonical if and only if* t *is obtained with* q *tandem duplications or deletions from* G.

Proof. 'Only if' part: if t is canonical then by definition it is obtained with q canonical copy+delete operations (certainly these are q tandem duplications or deletions on G).

'If part': we consider the contrapositive statement. If t is not canonical, then at least one block T' is obtained in a non-canonical way such that, together with the other $q-1$ blocks, they satisfy $cnp(t) = I_{3q}$. Then T' is obtained using at least two tandem duplication/deletion operations, and the corresponding 3-set for T', S', satisfies that $S' \notin S$ (otherwise, we could obtain T' using only one tandem duplication). Even in the ideal case that the remaining $q-1$ blocks needs $q-1$ tandem duplications and/or deletions, we still need at least $2 + (q-1) = q+1 > q$ operations to obtain t. Hence, if we use at most q tandem duplication or deletion operations to obtain t then t must be canonical. The canonical copy/delete definition implies that q tandem duplication or deletion operations are also necessary (i.e., we cannot fulfill the task of obtaining t with fewer than q such operations). □

Lemma 14. *If the MCNG problem admits a solution of* q *tandem duplication or deletion operations, then the X3C instance has a solution.*

Proof. From the previous lemma, if MCNG has a solution with size q then the q tandem duplication or deletion operations must be canonical. Let G' be obtained from G with q canonical copy+delete operations while satisfying the copy number constraints. Let $t = G' - G$, $|t| = 3q$, then t must be composed of a list of canonical substrings, each is a list of 3-strings T_j's such that their union is exactly equal to the set X. This gives, in linear time, a solution for the corresponding X3C instance. □

Note that, in the above proof, if arbitrary duplications are used then the reverse direction is problematic. For example, 123, 456, *abc* and *def* might be copied and then inserted into each other to have a messy permutation, which is not easy to check whether it is constructed from these copied blocks. In addition, it remains an open problem whether any approximation or FPT algorithm can be designed for the Minimum Copy Number Generation problem. In the next subsection, we solve the exemplar version of the Minimum Copy Number Generation problem optimally in roughly linear time (in the length of the input) when all the components in C are power of two's and when only tandem duplications are allowed.

Exemplar MCNG with Tandem Duplications

When the input genome P is *exemplar*, or, P is a permutation on Σ, whether the corresponding Exemplar Minimum Copy Number Generation (EMCNG) problem is NP-complete is open, even if only tandem duplications are allowed and deletions are disallowed. (We abbreviate the problem as EMCNG-TD.) However, we will show some properties of it and further show that it can be solved optimally when all the components of the target copy number profile are power of two's.

On the other hand, in terms of measuring the time complexity, we count $\sum_{i=1..m} c_i$ letters, where $C = \langle c_1, c_2, ..., c_m \rangle$, as part of the input size. Hence, any algorithm for the EMCNG-TD problem must take exponential time under a traditional algorithm design model. This is because if K is the number of binary bits for representing $\sum_{i=1..m} c_i$, where $C = \langle c_1, c_2, ..., c_m \rangle$, then the target genome must have length $\Theta(2^K)$. To make the statement of our result more precise, we make a natural assumption that any duplication takes $O(1)$ time, regardless of the length of the segment. Then, we simply measure the running time of our algorithm under this assumption (or, we simply look at the number of tandem duplications).

We define some extra concepts that will be used henceforth. Given $S = \langle s_1, s_2, \cdots, s_m \rangle$, a *superstring* of $T = \langle s_i, s_{i+1}, \cdots, s_j \rangle$ is a substring S' of S such that T is a substring of S'. Let $T[first]$ be s_i and $T[last]$ be s_j respectively. A *left superstring* of $T = \langle s_i, s_{i+1}, \cdots, s_j \rangle$ is a substring S'' of S such that T is a substring of S'' and the rightmost letter of S'' is $T[last]$. The *right superstring* can be defined symmetrically.

An example is given as follows. Let $S = \langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$ and let $T = \langle 4, 5, 6 \rangle$. Then, $S' = \langle 3, 4, 5, 6, 7, 8 \rangle$ is a superstring of T, $S_1 = \langle 2, 3, 4, 5, 6 \rangle$ is a left superstring of T, while $S_2 = \langle 4, 5, 6, 7 \rangle$ is a right superstring of T.

We now approach the Exemplar Copy Number Generation problem with tandem duplications (EMCNG-TD for short) by first proving two relevant lemmas.

Lemma 15. *In the Exemplar Copy Number Generation problem with tandem duplications, all input c_ℓ must be greater than zero.*

Proof. Assume for some gene ℓ we have $c_\ell = 0$, then in the final genome there is no gene ℓ. As only tandem duplications are allowed, all genes in P must appear at least once. This is a contradiction with the assumption that $c_\ell = 0$ for some gene c_ℓ. □

Lemma 16. *In the Exemplar Copy Number Generation problem with tandem duplications, if there is an input $c_\ell = 1$, then there is an optimal solution which does not perform any tandem duplication on gene ℓ.*

Proof. Again, this can be proved by contradiction. If $c_\ell = 1$ and some tandem duplication is performed on gene ℓ, then c_ℓ will be at least two (as no deletion is allowed). This is a contradiction with the assumption that $c_\ell = 1$. □

We now present the main theorem in this subsection.

Theorem 10. *In the Exemplar Minimum Copy Number Generation problem with tandem duplications, if* $c_i = 2^{a_i}$ *with* $a_i \geq 2$*, for all* $i = 1..m$ *and* $t = \min_i a_i$*, then the optimal solution can be computed in time linear in terms of the length of the input, plus* $O(m \log m)$*.*

Proof. It is obvious that when only tandem duplications are allowed, the optimal way to generate 2^t copies of c is to double c, and then cc,..., for t rounds. Here c is either a letter or a sequence. We call this process 'exponential duplication'.

To have this theorem, we present a recursive algorithm as follows. Starting with the input exemplar genome (or, permutation) P, we recursively generate a sequence T with copy number profile $\langle 2^{a_1-t}, 2^{a_2-t}, ..., 2^{a_m-t} \rangle$. Note that at least one of the components in $\langle 2^{a_1-t}, 2^{a_2-t}, ..., 2^{a_m-t} \rangle$ is equal to one. Hence we could apply Lemma 14 to have a recursive solution to handle subproblems with smaller sizes, until T is generated. (The base case is when we have a segment of 2^a's, possibly separated by 1's in the CNP, e.g., $\langle \cdots, 1, \underline{2^a, 2^a, ..., 2^a}, 1, \cdots \rangle$. Then we simply obtain that segment with an exponential duplication.) Finally, we duplicate T for t rounds exponentially to have $2, 2^2, ..., 2^t$ copies of T, i.e., the copy number profile eventually becomes $\langle 2^{a_1}, 2^{a_2}, ..., 2^{a_m} \rangle$.

To analyze the running time of this algorithm, we convert $C = \langle c_1, c_2, ..., c_m \rangle$, $c_i = 2^{a_i}$ for $i = 1..m$, to an X-monotone chain $\mathcal{C} = \langle (1, a_1), ..., (i, a_i), ..., (m, a_m) \rangle$ (or, more precisely, an X-monotone polygon with two extra vertices $(1, 0(= a_0))$ and $(m, 0(= a_{m+1}))$ which would be called interchangeably polygon $\mathcal{C}$). Then, the horizontal trapezoidalization of polygon $\mathcal{C}$ is computed by drawing a maximal horizontal line segment through every vertex of the polygon and the segment stops when it hits the boundary of polygon $\mathcal{C}$. A trapezoid is simply a quadrilateral with two parallel *supporting* segments (one of which could degenerate to a point). It is well-known that the number of trapezoids is linear in the number of vertices in polygon $\mathcal{C}$

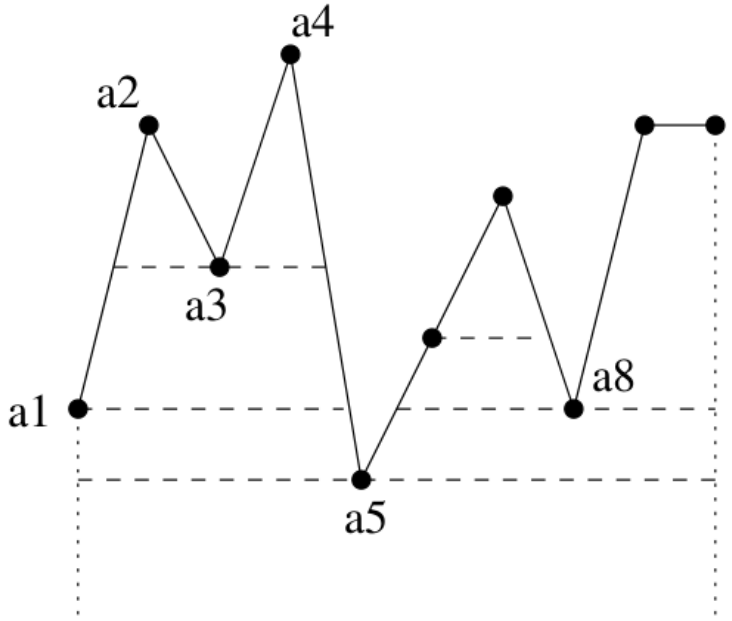


Figure 6.3: An X-monotone chain $\mathcal{C} = \langle (1, a_1), ..., (i, a_i), ..., (m, a_m) \rangle$ converted from the copy number file $C = \langle c_1, c_2, ..., c_m \rangle$, where $c_i = 2^{a_i}$ for $i = 1, ..., m$, and the trapezoid decomposition of $\mathcal{C}$.

(in this case $O(m)$); moreover, the trapezoidalization can be computed in $O(m \log m)$ time (which is the cost for sorting m elements) [19, 45].

It can be easily seen in Figure 6.3 that, going from top to bottom, given a trapezoid bounded vertically by supporting segments with y-coordinates $y = a_i$ and $y = a_j$, with $a_i > a_j$, the number of duplications performed on the subchain of the genome above the segment $y = a_i$ (possibly recursively duplicated already) by our recursive algorithm is exactly $a_i - a_j$ (or, the height of the corresponding trapezoid). As the number of trapezoids is $O(m)$, the total running time of our algorithm is proportional, up to a constant factor, to the sum of heights of all trapezoids, which is $O(\sum_{i=1..m} a_i)$, or linear in the input size of the EMCNG-TD problem, plus the cost for sorting m elements (i.e., $O(m \log m)$). □

We use an example to illustrate this theorem. Let $P = abcde$ and let the target

CNP be $\langle 8, 4, 8, 16, 8\rangle$. The optimal way is to generate $P' = aabcddecdde$ (with CNP $\langle 2, 1, 2, 4, 2\rangle$ in 3 duplications: $a \cdot a$, $d \cdot d$, and $cdde \cdot cdde$). Then we obtain an exponential duplication as follows. We first obtain $P'P'$ with one duplication and then $P'P'P'P'$ with one extra duplication, and the final CNP becomes $\langle 8, 4, 8, 16, 8\rangle$. This gives us a total of 5 tandem duplications.

We comment that the above theorem does not hold if (arbitrary) duplications are used: Consider $P = abc, C = \langle 4, 1, 4\rangle$. With the above theorem (and tandem duplications), we have a solution of 4 tandem duplications. However, with arbitrary duplications, we can have a solution of 3 duplications, e.g., $abc \to abc{\cdot}a \to ab{\cdot}ca{\cdot}ca \to ab \cdot cac \cdot cac \cdot a$.

We show below, with some example, that some natural ideas to solve the EMCNG-TD problem (i.e., when C is general) does not work. We list this as follows. **Rounding C to the Power of Two's Below C**: A natural idea for solving the problem is to round C to the closest power of two's which is bounded from above by C, using Theorem 10 to solve the corresponding case optimally and then try to cover the offsets. This would give us a natural heuristic solution for EMCNG-TD. But this method does not always give us an optimal solution. When $P = abcd$ and the target CNP $C = \langle 7, 2, 7, 4\rangle$, the optimal solution uses at most 7 tandem duplications: $aa \cdot aa \cdot b \cdot cc \cdot cc \cdot dd$ (5 duplications), then one more on $aaabccc$ and one more on dd, i.e., $a \cdot aaabccccdd \to a \cdot aaabccc \cdot aaabccc \cdot cdd$ followed with a duplication on dd. On the other hand, if we round C to the closest power of two's below C first, which is $\langle 4, 2, 4, 4\rangle$, we would need 8 tandem duplications: we first obtain $(aa \cdot b \cdot cd \cdot cd) \cdot (aa \cdot b \cdot cd \cdot cd)$ with 3 tandem duplications, then we duplicate aa and a to have seven a's and finally duplicate c three times to have seven c's.

It is unknown whether we could adapt the above idea into a valid approximation algorithm for EMCNG-TD. The main reason is that the properties of the optimal

solutions are not well-known yet. In summary, when C is arbitrary whether the EMCNG-TD problem is NP-complete or not is still open. In the next section, we present some negative results for some variations of the Minimum Copy Number Generation Problem.

Some Variations of the Minimum Copy Number Generation Problem

In this section, we investigate two variations of the Minimum Copy Number Generation problem targeting at studying the approximability and FPT tractability of MCNG. The problems are called MCNG($\geq$) and MCNG($\leq$) and are defined as follows.

Problem MCNG($\geq$):

INSTANCE: A genome (or a sequence) G of length n over a gene set $\{1, 2, ..., m\}$, a copy number profile $C = C_1 \circ C_2$, where $C_1 = \langle c_1, c_2, ..., c_q \rangle$ and $C_2 = \langle c_{q+1}, c_{q+2}, ...c_m \rangle$; and an integer k.

QUESTION: Is there a sequence of at most k tandem duplications which can convert G into G' such that the copy number profile of G', $cnp(G')$, satisfies $CNP(G')[1..q] \geq C_1$ and $CNP(G')[q+1..m] = C_2$?

We could think of this variation of MCNG as one such that the malicious genes duplicated are at least above certain thresholds. Naturally we could have the symmetric case where the benign genes eventually obtained are below some given thresholds.

Problem MCNG($\leq$):

INSTANCE: A genome (or a sequence) G of length n over a gene set $\{1, 2, ..., m\}$, a copy number profile $C = C_1 \circ C_2$, where $C_1 = \langle c_1, c_2, ..., c_q \rangle$ and $C_2 = \langle c_{q+1}, c_{q+2}, ...c_m \rangle$; and an integer k.

QUESTION: Is there a sequence of at most k segment deletions which can convert G into G' such that the copy number profile of G', $CNP(G')$, satisfies $CNP(G')[1..q] \leq C_1$ and $CNP(G')[q+1..m] = C_2$?

As both problems are similar, we will mainly focus on MCNG($\geq$). We will reduce the Set Cover problem to MCNG($\geq$), which is defined as follows.

INSTANCE: Integer k, a base set $X = \{x_1, ..., x_n\}$, and a set $S = \{S_1, ..., S_m\}$ where $S_i \subset X$.

QUESTION: Is there a subset of k elements of S which cover all the elements in X, namely, $S' = \{S'_1, ...S'_k\}, S' \subseteq S$ and $\cup_{i=1..k} S'_i = X$?

The set-cover problem is a classical and fundamental problem in computer science with many applications. It is a well-known NP-complete problem showed in Karp's 21 NP-complete problems in 1972 and its solutions give rise to the development of the entire field of approximation algorithms [4]. The well known approximation algorithm for the set-cover problem which has a ratio of $1 + \ln n$ was given by Johnson [61] in 1974. Chvatal [29] improved the upper bound on the approximation ratio to $\ln n - \ln \ln n + O(1)$ in 1979. These ratios are tight because of a famous inapproximability result of Feige [43] which states that there is no $(1 - \varepsilon) \ln n$-approximation algorithm for the set-cover problem unless there are subexponential $O(n^{polylog(n)})$ time deterministic algorithms for problems in NP. A lower bound of $c \ln n$ was established by Raz and Safra [87] in 1997, where c is a constant, under the weaker assumption that P$\neq$NP. A similar result with a higher value of c was recently proved by Alon, Moshkovitz and Safra [4] in 2006.

Theorem 11. *MCNG($\geq$) cannot be approximated within a factor $O(\log n)$ unless P=NP; moreover, the parameterized version of MCNG($\geq$) in W[2]-hard, i.e., it does not admit any FPT algorithm unless FPT=W[2].*

Proof. We reduce the Set Cover problem to MCNG($\geq$). Recall that in Set Cover we are given an integer k, a base set $X = \{x_1, ..., x_n\}$, and a set $S = \{S_1, ..., S_m\}$ where $S_i \subset X$, the problem is to identify a subset of k elements of S which cover all the elements in X, i.e., $S' = \{S'_1, ...S'_k\}, S' \subseteq S$ and $\cup_{i=1..k} S'_i = X$.

Let $f(x_i)$ be the total number of times that x_i appears in the multiset $\cup_{i=1..m} S_i$. We construct G as follows. The gene set is $X \cup Y$, where $Y = \{p_1, p_2, ..., p_m\}$ is the set of peg genes. For each set S_i, let T_i be the sequence of genes (letters) in S_i (ordered by their indices). Then

$$G = \langle T_1, p_1, ..., T_i, p_i, ..., T_m, p_m \rangle.$$

We take C_1 as the copy number profiles for genes in $\langle x_1, x_2, ..., x_n \rangle$ and C_2 as the copy number profiles for genes in $\langle p_1, p_2, ..., p_m \rangle$; more specifically, we set $C_1 = \langle f(x_1) + 1, f(x_2) + 1, ..., f(x_n) + 1 \rangle$ and $C_2 = 1^m$.

We claim that there is a solution of size k for Set Cover if and only if there is a sequence of k tandem duplications for the MCNG($\geq$) problem. The crucial point is that due to $C_2 = 1^m$, there must be an optimal solution where the tandem duplications only occur on T_i's — similar to the previous lemmas. The 'iff' relation is then trivial and omitted. We remind the readers that this reduction preserves approximability and is also an FPT-reduction. □

We show an example for the reduction. $X = \{a, b, c, d, e, f\}, S = \{S_1, S_2, ..., S_5\}$ with $S_1 = \{a, c\}$, $S_2 = \{b, d\}$, $S_3 = \{b, e, f\}$, $S_4 = \{c, d\}$, and $S_5 = \{c, f\}\}$. $S' = \{\{a, c\}, \{b, d\}, \{b, e, f\}\}$ is a set cover with size 3. Our construction is

$$G = ac \cdot p_1 \cdot bd \cdot p_2 \cdot bef \cdot p_3 \cdot cd \cdot p_4 \cdot cf \cdot p_5.$$

For the corresponding elements as listed in X, $C_1 = \langle 2, 3, 4, 3, 2, 3\rangle$. For the peg genes in Y we have $C_2 = \langle 1, 1, 1, 1, 1\rangle$. The 3 optimal tandem duplications are on ac, bd and bef.

The next theorem is similarly obtained.

Theorem 12. *MCNG(≤) cannot be approximated within a factor $O(\log n)$ unless P=NP; moreover, the parameterized version of MCNG(≤) in W[2]-hard, i.e., it does not admit any FPT algorithm unless FPT=W[2].*

Proof. The proof is similar to that of Theorem 11. We reduce the Set Cover problem to MCNG(≤) as follows.

Let $f(x_i)$ be the total number of times that x_i appears in $\cup_{i=1..m} S_i$. We construct G as follows. The gene set is $X \cup P$, where $P = \{p_1, p_2, ..., p_m\}$ is the set of peg genes. For each set S_i, let T_i be the sequence of genes (letters) in S_i (ordered by their indices). Then

$$G = \langle T_1, p_1, ..., T_i, p_i, ..., T_m, p_m\rangle.$$

We take C_1 as the copy number profiles for genes in $\langle x_1, x_2, ..., x_n\rangle$ and C_2 as the copy number profiles for genes in $\langle p_1, p_2, ..., p_m\rangle$; more specifically, we set $C_1 = \langle f(x_1) - 1, f(x_2) - 1, ..., f(x_n) - 1\rangle$ and $C_2 = 1^m$.

We claim that there is a solution of size k for Set Cover if and only if there is a sequence of k deletions for the MCNG(≤) problem. The details are similar to Theorem 11 and hence omitted. □

Practical Solution for the Minimum Copy Number Generation Problem

A Practical Greedy Solution

In this section, we design a practical greedy algorithm for the Minimum Copy Number Generation (MCNG) problem. Recall that for MCNG, both (arbitrary)

duplications and deletions are allowed. The input is the original genome (sequence) G with CNP $C_1 = cnp(G)$ and the target profile CNP C_2, and the goal is to use the minimum number of duplications and deletions to convert G into a genome H with $cnp(H) = C_2$. Our idea on solving this problem is based on the greedy method. Define $Z = C_2 - C_1$ and let $cn_Z(g_k)$ be the number of copies of gene g_k in Z to be changed (with could be negative). We say a substring of G, $S = g_i g_{i+1} \cdots g_j$, is a '+'-*substring* with respect to Z, if $cn_Z(g_k) > 0$, for $k = i, ..., j$; otherwise, if $cn_Z(g_k) < 0$ for $k = i, ..., j$, then we call S a '-'-*substring* with respect to Z.

An example of the above setting is as follows. $G = abcdeaceb, C_1 = \langle 2, 2, 2, 1, 2\rangle$, and $C_2 = \langle 4, 3, 5, 0, 1\rangle$. Then $Z = \langle 2, 1, 3, -1, -1\rangle$, which means that we need to duplicate two copies of a, one copy of b and three copies of c; moreover, we need to delete one copy of d and one copy of e. In this case, abc and ac are '+'-substrings with respect to Z; and, de and e are '-'-substrings with respect to Z.

Our method is a greedy one. We compute the longest '+'-substring (corresponding to a duplication) and '-'-substring (corresponding to a deletion) with respect to Z. Whichever one is longer, the corresponding operation will be performed first. The tricky part is where to put a duplicated segment. In [54], the goal is to transform a genome G_1 into G_2 with the minimum number of duplications and deletions, hence the alignment of G_1 and G_2 is important. Here, we care more whether the target genome H satisfies $cnp(H) = C_2$. Hence, we could either put the duplicated segment randomly or simply put it at the end of the current sequence G. We tried both methods and the difference seems to be small. Therefore, in this implementation we just put a duplicated segment always at the end of the current G, which makes exponential duplication possible (in the later rounds of duplications) and is also easy to chase visually the change of the genome (at least for small datasets). More details of our algorithm can be found in Algorithm 6.9.

Empirical Results for Simulated Data

We use Java to implement our greedy algorithm on a PC with Intel Core 3.40 GHz processor and 16 GB RAM for the simulation data. The dataset generator builds an exemplar genome, g, of size n, comprising of an integer sequence $[1..n]$ representing n unique genes. Note that for the ease of description here we use unsigned integers (genes). Our generic genome G is created by doubling the size of the exemplar genome and randomly shuffling the genes. Then, we perform a sequence of $K = 10, 15, 20$ duplication and deletion operations with the duplication probability of p_{i1} and the deletion probability of $p_{i2}, i = 1, 2, 3$, on genome G to generate our target genome G' so that we have the target CNP $C_2 = cnp(G')$. We also set the size/length of duplication and deletion $l_j (j = 1, 2, 3)$ to be no greater than $2\%, 4\%, 6\%$ the size of the sequence at each iteration respectively. The 9 cases of L_{ij}'s are summarized as follows,

- $L_{11} = \{p_{11} = 0.50, p_{12} = 0.50, l_1 = 2\%\}$
- $L_{12} = \{p_{11} = 0.50, p_{12} = 0.50, l_2 = 4\%\}$
- $L_{13} = \{p_{11} = 0.50, p_{12} = 0.50, l_3 = 6\%\}$
- $L_{21} = \{p_{21} = 0.70, p_{22} = 0.30, l_1 = 2\%\}$
- $L_{22} = \{p_{21} = 0.70, p_{22} = 0.30, l_2 = 4\%\}$
- $L_{23} = \{p_{21} = 0.70, p_{22} = 0.30, l_3 = 6\%\}$
- $L_{31} = \{p_{31} = 0.30, p_{32} = 0.70, l_1 = 2\%\}$
- $L_{32} = \{p_{31} = 0.30, p_{32} = 0.70, l_2 = 4\%\}$
- $L_{33} = \{p_{31} = 0.30, p_{32} = 0.70, l_3 = 6\%\}$

It is possible to duplicate or delete the prior duplicated segment from last iteration in our simulation and the total length of our target sequence could become larger or smaller in each iteration (depending on whether the operation is a duplication or deletion). The final calculated operation number from our greedy algorithm is an average over five tries for different genome sizes N. In our simulation, we have nine different settings summarized in three tables, $(k_{ij}, t_{ij}, n_{ij}), i, j = 1, 2, 3$, in which k_{ij} represents the final operation number from our algorithm, t_{ij} denotes the running time in seconds and n_{ij} represents the final sequence length generated from K duplication and deletion operations, all averaged over five tries.

From our simulation results in Table 6.1-6.3, we conclude that our simple greedy algorithm is performing very well for the simulated data with a small duplication and deletion length during the evolutionary process. For example, in the case L_{11}, the average operation number of our algorithm is less than three times of the given operation number K in the worst case. When the size of duplication and deletion gets larger and the number of duplication and deletion operations increases, our algorithm does not work quite well for the larger sizes (≥ 5000) of the genome in cases $L_{1j}, j = 2, 3$. This conclusion also holds roughly for L_{ij} with $i = 2, 3$, which implies that further research is necessary (maybe using an algorithm more complex than the greedy method). But the program runs very fast, even for the larger cases.

Discussion

In this section, we discuss some aspects of the General Minimum Copy Number Generation (GMCNG) problem. It is easy to see that in this case, even the proof of Theorem 10 does not work anymore, as one could use k duplications and deletions to have an optimal solution, but such a solution has nothing to do with the solution for X3C. In the following, we present a more complicated case.

Table 6.1: The testing results for the setting $\{L_{1j}\ (j = 1, 2, 3)\}$. Note that k_{ij} represents the average operation number, t_{ij} represents the average running time in seconds and n_{ij} represents the average final sequence length, all over five tries.

N	K	L_{11}			L_{12}			L_{13}		
		k_{11}	t_{11}	n_{11}	k_{12}	t_{12}	n_{12}	k_{13}	t_{13}	n_{13}
	10	7.2	0	507.2	8	0	519.4	9.4	0	493
500	15	13.6	0	481.4	15.4	0	536	17.2	0	481.6
	20	16.6	0	478.8	17	0	479	22	0	496.4
	10	9	0	696	10.2	0	742	12.8	0	659.2
700	15	14	0	701.8	16.8	0	697.2	18.4	0	661
	20	16.6	0	709.2	20.6	0	679.6	32	1	686
	10	9	0	965.6	10.8	0	1053	13.6	0	1142
1000	15	15.4	0	956.8	18	0	1062.8	25.6	1	1005.2
	20	21.8	0	1009.2	29	1	1007.2	39.6	2	1028.8
	10	12.8	2.2	2941.6	17.4	3	2983	22.4	4	2974
3000	15	16.6	4.8	3018.6	29.2	4.4	2964.4	51.8	8.4	3045
	20	29.4	5.2	2982	53.4	8.6	3184	67	12	2978.6
	10	22	10.2	5036.4	38.2	12	4901	32.2	8.8	4965.2
5000	15	27	9.4	4933.2	52	14.6	4960.2	65.2	16.8	4665.4
	20	44.2	13.6	5157.8	90.6	24.2	4884.4	97.4	25.4	4626.2
	10	29	9.8	7148.4	26.4	11	6982.6	64.2	31.8	7267
7000	15	33	10.8	7015.2	58.8	24.4	6819.2	90.8	38.2	7024
	20	55.4	16.8	7051	101	40.6	7138	143	69	6867.4
	10	28.2	15.8	9762.4	36.8	24.6	9821	77	49.2	10106
10000	15	35.8	24.6	10039	95.2	63.6	9896.2	158.8	114	12111
	20	56	36	10025.4	160.8	108.4	9887	222	168	10193.2

Table 6.2: The testing results for the setting $\{L_{2j}$ (j=1,2,3)$\}$. The corresponding parameters are similar to Table 6.1.

N	K	L_{21}			L_{22}			L_{23}		
		k_{21}	t_{21}	n_{21}	k_{22}	t_{22}	n_{22}	k_{23}	t_{23}	n_{23}
	10	6.4	0	510.4	8.6	0	514	10.2	0	559
500	15	11.4	0	510.6	13.8	0	537	17	0	511.4
	20	13.2	0	516.8	20	0	531.8	24	0	533.4
	10	8	0	713	9.6	0	712.4	15	0	756.6
700	15	11.8	0	703.2	15.2	0	732.4	21.8	0	737.6
	20	15.4	0	718	24	0	710.2	31.6	0.2	759.4
	10	9.4	0	1026.2	12	0	1042.6	15.2	0.2	1071.8
1000	15	12.4	0	1037.2	15.6	0	1046.4	32	1	1083.8
	20	17.8	0	1031.6	26.8	0.8	1110.4	31.6	0.4	1115
	10	10.8	1	3096	21.4	2	3156	38.8	4.4	3118.8
3000	15	16.8	1.8	3117	26.6	2.8	3206.6	44	5.6	3319.6
	20	23.6	2.4	3169.2	56.8	6	3291.2	108.6	13.6	3256.6
	10	15.4	2.6	5090.4	28.8	5.2	5101.2	69.6	16.4	5382.4
5000	15	26	4.4	5109.6	51.2	10.2	5403	93.8	22.6	5517
	20	42.8	8	5168.8	106.2	21.6	5399.6	177.2	48.2	5743.4
	10	11.8	3.8	7164.2	26	7.4	7211	51.8	19.8	7783.8
7000	15	28.4	8.2	7202.4	76.8	22.2	7350.2	161	55.2	7239.6
	20	47	13.6	7254.4	101.6	32.4	7772	229.6	84.6	7831.8
	10	19.8	9.6	10222.6	40.8	21.8	10699.2	65.8	36.6	10996.8
10000	15	30	13.8	10121.2	104.8	54.8	10724.8	146.6	79.8	10663.4
	20	58	26	10380.6	147.8	79.6	10947	212.4	134.2	11659.6

Table 6.3: The testing results for the setting $\{L_{3j}$ (j=1,2,3)$\}$. The corresponding parameters are similar to Table 6.1.

N	K	L_{31}			L_{32}			L_{33}		
		k_{31}	t_{31}	n_{31}	k_{32}	t_{32}	n_{32}	k_{33}	t_{33}	n_{33}
	10	7.6	0	495.4	11	0	491	11.2	0	467
500	15	10	0	482.4	15	0	460	16.8	0	464.2
	20	13.8	0	474.8	18.4	0	439.8	22.2	0	426.2
	10	7.6	0	686.6	11.8	0	688.8	13.4	0	686.6
700	15	13	0	686.6	17	0	961.2	14.2	0	672
	20	16.2	0	674.2	22.8	0	632.6	30	0	625.2
	10	9.8	0	972.8	10.4	0	962	14.8	0	988.6
1000	15	13	0	973.6	17.2	0	971.8	16.6	0	933.2
	20	16.8	0	947.6	30.8	0	933.8	33.8	0.6	887.4
	10	12	1	2926.8	13.4	1	2724	24.6	1.6	2840.6
3000	15	21.4	1.6	2946	42.4	3	2925.4	56.6	5	2882.4
	20	28.6	2	2859.4	45.4	3.6	2814.2	70	5.8	2780.4
	10	15.2	1.8	4865	30.4	5	4809.8	38	5.8	474.8
5000	15	23.6	3.8	4846.2	52.6	8.2	4696.8	52.6	7.6	4428.2
	20	47	7.8	4832.4	80.2	16.6	4611	126	24.4	4284.6
	10	17.8	4.8	6939	30.8	7	6737.6	55	14.6	6659.4
7000	15	37.6	9.2	6892.4	67.8	14.4	6672.2	129	36	6680.9
	20	52.8	12.6	6675	121	30.4	6614.6	140.6	30.4	5808
	10	19.8	7.6	9846.6	55.2	20.4	9560	89.6	33.6	9361.8
10000	15	39.6	15.2	9660.4	129.6	52.8	9977.6	203.2	78	9351.2
	20	46	16.2	9428.8	141	50.8	9171.6	266.2	102.2	9221.6

The Order-Switching Scenario for the Exemplar Case

We show a scenario such that the corresponding Lemma 16 does not hold for the Exemplar Minimum Copy Number Generation problem (EMCNG), when deletions are also allowed; in fact, even if we still use tandem duplications. Let the input permutation be $P = \langle a, b, c \rangle$, and let the target CNP be $C = \langle 16, 1, 16 \rangle$. If we do not duplicate b at all, then we duplicate a and c each for 4 time (e.g., $a \to a \cdot a \to aa \cdot aa \to aaaa \cdot aaaa \to aaaaaaaa \cdot aaaaaaaa$. This gives us a solution of 8 tandem duplications.

On the other hand, we could duplicate abc, then delete the first a and the second bc to have bca, then we could duplicate ca using 4 exponential (tandem) duplications. The transformation sequence is $abc \to abc \cdot abc \to \cancel{a}bcabc \to bca\cancel{bc} \to bca \cdot ca \to bcaca \cdot caca \to bcacacaca \cdot cacacaca \to bcacacacacacacaca \cdot cacacacacacacaca$. This gives us a solution of 5 tandem duplications and 2 deletions.

For this example, if we use arbitrary duplications but no deletions (e.g., apply $abc \to abc \cdot a$ to have ca), we could obtain a solution with 5 duplications.

This example shows that the EMCNG problem is much more complex, even in the case when the genome is a permutation. In fact, we suspect that there might be some scenario more complex than order-switching. On the other hand, to obtain some positive results we must be able to extract some properties of the optimal solution.

Relation to the Whole Genome Duplication Problem

We comment that the MCNG problem is related to but different from (and more difficult than) the Whole Genome Duplication Problem (or the genome halving problem), which is to identify the single whole genome duplication event in its evolution history [2, 3, 40]. In our problem, the number of duplications and deletions is not restricted. Our problem is also different from the genome alignment problem

with tandem duplications and deletions studied by Berard and Rivals [11], as in our case we have no restrictions on the number of whole genome duplication.

Concluding Remarks

Motivated by somatic duplications and deletions in cancer genomics, we study the Minimum Copy Number Generation (MCNG) problem which is to generate a genome with a target copy number profile from a given genome. This direction is almost completely new and a lot of research is needed on designing efficient (approximation/FPT) algorithms. We summarize our results and also list some open problems as follows.

1. We show that the Minimum Copy Number Generation (MCNG) problem is NP-complete when the duplications are restricted to be tandem. But is there any approximation or FPT algorithm for it? (For two restricted variations of the problem, we showed that they are as hard as Set Cover, hence cannot have any good approximation and cannot have any FPT algorithm unless FPT=W[2].) What if the duplications are arbitrary segment duplications? Is MCNG NP-complete?

2. We show that the Exemplar Minimum Copy Number Generation problem can be solved optimally (in linear time in terms of the length of the input, plus $O(m \log m)$ where m is the number of genes in the genome) when the components of the target copy number profile are power of two's and when only tandem duplications are allowed. But is the Exemplar Minimum Copy Number Generation problem NP-complete or polynomially solvable? What if the duplications are arbitrary segment duplications (while deletions are disallowed)?

Input: Source profile $CNP\ C_1$, target profile $CNP\ C_2$ and the original sequence G with $cnp(G) = C_1$.

Output: The set of duplication and deletion operations S with $S = \emptyset$ initially, such that G is transformed into H with $cnp(H) = C_2$.

1: Operation number $i = 0$; $Z = C_2 - C_1$

2: **While** (Z is not a zero vector)

3: Compute the longest $'+'$-substring of G with respect to Z, G_Z^+ and the longest $'-'$-substring of G with respect to Z, G_Z^-.

4: **if** ($|G_Z^-| \geq |G_Z^+|$)

5: Delete the longest $'-'$-substring S_Z^- from G.

6: Record this deletion as S_i and update $S \leftarrow S \cup \{S_i\}$.

7: Update $G \leftarrow G - G_Z^-$ and $Z \leftarrow Z - cnp(G_Z^-)$.

8: $i = i + 1$.

9: **end**

10: **else**

11: Concatenate the longest $'+'$-substring G_Z^+ to the end of G.

12: Record this duplication as S_i and update $S \leftarrow S \cup \{S_i\}$.

13: Update $G \leftarrow G \circ G_Z^+$ and $Z \leftarrow Z - cnp(G_Z^+)$.

14: $i = i + 1$.

15: **end**

16: **end**

17: Return S and the resulting sequence $H \leftarrow G$.

Algorithm 6.9: Greedy Algorithm for MCNG

CONCLUSION

In this dissertation, we cover the *de novo* protein sequencing problem and the classical genome rearrangement problems.

De Novo Protein Sequencing Problem

In *de novo* protein sequencing area, we explore the process of attempting to construct a target complete protein sequence by utilizing data from both top-down and bottom-up, tandem mass spectra processes as follows:

- Constructing top-down scaffold using top-down tandem mass spectra
- Aligning bottom-up peptides with top-down to extend the coverage of the scaffold
- Filling the extended scaffold by inserting missing amino acids with respect to its homologous protein sequence

We consider two versions of protein scaffold filling problems. We present an $O(n^{22})$ time solution and two practical algorithms for PSF, and we present some empirical results using some real datasets. We also show that CP-PSF can be solved in $O(n^{26})$ time and a special case for the CP-PSF problem is solvable in $O(n^5)$ time, and then, based on it, we design two practical algorithms for CP-PSF. We then present some empirical results both on the simulated and real datasets. The empirical results, based on some antibody and mammalian proteins, show that the algorithms can fill protein scaffolds with high quality, provided that a good pair of scaffold and reference are given.

Genome Rearrangement Problems

We investigate several classical genome rearrangement problems. First, we consider using minimum number of DCJ distance to convert one genome to another. For this problem, we design a FPT approximation algorithm to improve the approximation factor to $4/3 + \epsilon$. Our algorithm can also be used to improve the approximation factor for Sorting by Translocations in FPT time.

Second, we approach the One-sided Exemplar Adjacency Number problem by first reformulating it as the maximum independent set on colored interval graph, provide the NP-hardness proof of the MIS-problem and then design a simple 2-approximation algorithm (to obtain a reduced instance). Finally we use integer linear programming to solve the reduced instance exactly. We report our test results in comparison with GREDU on simulated data datasets. It turns out that our algorithm is more stable and can process genomes of length up to 12,000 (while GREDU sometimes can falter on such a large dataset).

Third, we show that the Minimum Copy Number Generation problem in which use the minimum number of segment duplications and deletions operations on G to obtain some genome G' which has a copy number profile (CNP) C, is NP-complete when the input genome is generic (i.e., containing duplicated genes); on the other hand, when G is exemplar (i.e., a permutation) and the components of the target copy number profile C are power of two's then the problem can be solved optimally in a linear time in the length of the input (or $|C|$), plus the cost for sorting $|G|$ elements. We also study two variations of the Minimum Copy Number Generation problem and prove their hardness in terms of approximability and FPT tractability. Finally, we discuss the general version when both segment duplications and deletions are allowed. We conclude that our simple greedy algorithm is performing very well for the

simulated data with a small duplication and deletion length during the evolutionary process. However, the empirical results are not quite promising for the larger sizes (≥ 5000) of the genome when the size of duplication and deletion get larger and the number of duplication and deletion operations increases.

Future Work

In the area of *de novo* protein sequencing, the top-down mass spectra sometimes contains both prefix ion peaks and suffix ion peaks, and it is not easy to determine which one should be selected for top-down scaffold construction. I am interested in further investigating a sophisticated scoring scheme to identify which pattern of the top-down is correct based on Bayesian model.

For the Minimum Copy Number Generation problem (MCNG), where both segment duplications and deletions are allowed, is completely open. Our simple greedy algorithm seems to work for small datasets, but further research is necessary when the duplication/deletion numbers are larger. One possible improvement over the greedy method might be looking a few steps ahead: For example, we could evaluate all '+'-substrings such that such a substring could cause a longest (tandem) duplication in the next round (aka. a 2-step greedy method).

REFERENCES CITED

[1] R. Aebersold and M. Mann. Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207, 2003.

[2] M. A. Alekseyev and P. A. Pevzner. Whole genome duplications, multi-break rearrangements, and genome halving problem. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 665–679. Society for Industrial and Applied Mathematics, 2007.

[3] N. Alexeev, P. Avdeyev, and M. A. Alekseyev. Comparative genomics meets topology: a novel view on genome median and halving problems. *BMC bioinformatics*, 17(14):418, 2016.

[4] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177, 2006.

[5] S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. A pseudo-boolean framework for computing rearrangement distances between genomes with duplicates. *Journal of Computational Biology*, 14(4):379–393, 2007.

[6] V. Bafna and N. Edwards. On de novo interpretation of tandem mass spectra for peptide identification. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 9–18. ACM, 2003.

[7] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM J. Discrete Math.*, 11(2):224–240, 1998.

[8] N. Bandeira, K. R. Clauser, and P. A. Pevzner. Shotgun protein sequencing assembly of peptide tandem mass spectra from mixtures of modified proteins. *Molecular & Cellular Proteomics*, 6(7):1123–1134, 2007.

[9] N. Bandeira, V. Pham, P. Pevzner, D. Arnott, and J. R. Lill. Automated de novo protein sequencing of monoclonal antibodies. *Nature biotechnology*, 26(12):1336–1338, 2008.

[10] N. Bandeira, H. Tang, V. Bafna, and P. Pevzner. Shotgun protein sequencing by tandem mass spectra assembly. *Analytical chemistry*, 76(24):7221–7233, 2004.

[11] S. Bérard and E. Rivals. Comparison of minisatellites. *Journal of Computational biology*, 10(3-4):357–372, 2003.

[12] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *Algorithms in Bioinformatics, 6th International Workshop, WABI*

2006, Zurich, Switzerland, September 11-13, 2006, Proceedings, pages 163–173, 2006.

[13] P. Berman and M. Fürer. Approximating maximum independent set in bounded degree graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 365–371, 1994.

[14] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, pages 200–210, 2002.

[15] G. Blin, G. Fertin, F. Sikora, and S. Vialette. The exemplar breakpoint distance for non-trivial genomes cannot be approximated. In *International Workshop on Algorithms and Computation*, pages 357–368. Springer, 2009.

[16] L. Bulteau, A. P. Carrieri, and R. Dondi. Fixed-parameter algorithms for scaffold filling. *Theoretical Computer Science.*, 568:72–83, 2015.

[17] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM journal on discrete mathematics*, 12(1):91–110, 1999.

[18] A. Caprara and R. Rizzi. Improved approximation for breakpoint graph decomposition and sorting by reversals. *Journal of Combinatorial Optimization.*, 6(2):157–182, 2002.

[19] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Transactions on Graphics (TOG)*, 3(2):135–152, 1984.

[20] 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.

[21] X. Chen, R. Sun, and J. Yu. Approximating the double-cut-and-join distance between unsigned genomes. *BMC Bioinformatics*, 12(S-9):S17, 2011.

[22] Z. Chen, B. Fu, R. Goebel, G. Lin, W. Tong, J. Xu, B. Yang, Z. Zhao, and B. Zhu. On the approximability of the exemplar adjacency number problem for genomes with gene repetitions. *Theoretical Computer Science*, 550:59–65, 2014.

[23] Z. Chen, B. Fu, J. Xu, B. Yang, Z. Zhao, and B. Zhu. Non-breaking similarity of genomes with gene repetitions. In *Annual Symposium on Combinatorial Pattern Matching*, pages 119–130. Springer, 2007.

[24] Z. Chen, B. Fu, and B. Zhu. The approximability of the exemplar breakpoint distance problem. In *International Conference on Algorithmic Applications in Management*, pages 291–302. Springer, 2006.

[25] H. Chi, H. Chen, K. He, L. Wu, B. Yang, R.-X. Sun, J. Liu, W.-F. Zeng, C.-Q. Song, S.-M. He, et al. pnovo+: de novo peptide sequencing using complementary hcd and etd tandem mass spectra. *Journal of proteome research*, 12(2):615–625, 2012.

[26] H. Chi, R.-X. Sun, B. Yang, C.-Q. Song, L.-H. Wang, C. Liu, Y. Fu, Z.-F. Yuan, H.-P. Wang, S.-M. He, et al. pnovo: de novo peptide sequencing and identification using hcd spectra. *Journal of proteome research*, 9(5):2713–2724, 2010.

[27] S. Chowdhury, S. Shackney, K. Heselmeyer-Haddad, T. Ried, A. Shaeffer, and R. Schwartz. Algorithms to model single gene, single chromosome, and whole genome copy number changes jointly in tumor phylogenetics. *PLOS Computational Biology*, 10(7), 2014.

[28] D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 244–252, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[29] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[30] S. Cooke, J. Temple, S. Macarthur, M. Zahra, L. Tan, R. Crawford, C. Ng, M. Jimenez-Linan, E. Sala, and J. Brenton. Intra-tumour genetic heterogeneity and poor chemoradiotherapy response in cervical cancer. *British journal of cancer*, 104(2):361, 2011.

[31] S. L. Cooke and J. D. Brenton. Evolution of platinum resistance in high-grade serous ovarian cancer. *The lancet oncology*, 12(12):1169–1174, 2011.

[32] J. S. Cottrell and U. London. Probability-based protein identification by searching sequence databases using mass spectrometry data. *electrophoresis*, 20(18):3551–3567, 1999.

[33] P. A. Cowin, J. George, S. Fereday, E. Loehrer, P. Van Loo, C. Cullinane, D. Etemadmoghadam, S. Ftouni, L. Galletta, M. S. Anglesio, et al. Lrp1b deletion in high-grade serous ovarian cancers is associated with acquired chemotherapy resistance to liposomal doxorubicin. *Cancer research*, 72(16):4060–4073, 2012.

[34] Y. Cui, L. Wang, D. Zhu, and X. Liu. A (1.5 + ϵ)-approximation algorithm for unsigned translocation distance. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 5(1):56–66, Jan. 2008.

[35] V. Dančík, 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.

[36] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012.

[37] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[38] P. Edman. Method for determination of the amino acid sequence in peptides. *Acta chem. scand*, 4(7):283–293, 1950.

[39] M. El-Kebir, B. Raphael, R. Shamir, R. Sharan, S. Zaccaria, M. Zehavi, and R. Zeira. Copy-number evolutions: complexity and algorithms. In *Proceedings of WABI'2016, LNCS*, volume 9838, pages 137–149. Springer, 2016.

[40] N. El-Mabrouk, J. H. Nadeau, and D. Sankoff. Genome halving. In *Annual Symposium on Combinatorial Pattern Matching*, pages 235–250. Springer, 1998.

[41] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3(4):369–379, Oct. 2006.

[42] J. K. Eng, A. L. McCormack, and J. R. Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989, 1994.

[43] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[44] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics (TOG)*, 3(2):153–174, 1984.

[46] A. Frank and P. Pevzner. Pepnovo: de novo peptide sequencing via probabilistic network modeling. *Analytical chemistry*, 77(4):964–973, 2005.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[48] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[49] A. Guthals, K. R. Clauser, and N. Bandeira. Shotgun protein sequencing with meta-contig assembly. *Molecular & Cellular Proteomics*, 11(10):1084–1096, 2012.

[50] M. M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 160–169, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[51] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 581–, Washington, DC, USA, 1995. IEEE Computer Society.

[52] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, Jan. 1999.

[53] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[54] P. Holloway, K. Swenson, D. Ardell, and N. El-Mabrouk. Evolution of genome evolution by duplication and loss: an alignment approach. In *Proceedings of RECOMB'2012, LNCS*, volume 7262, pages 94–112. Springer, 2012.

[55] H. Jiang, L. Wang, B. Zhu, and D. Zhu. A (1.408+ ε)-approximation algorithm for sorting unsigned genomes by reciprocal translocations. In *International Workshop on Frontiers in Algorithmics*, pages 128–140. Springer, 2014.

[56] H. Jiang, C. Zheng, D. Sankoff, and B. Zhu. Scaffold filling under the breakpoint and related distances. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(4):1220–1229, 2012.

[57] H. Jiang, F. Zhong, and B. Zhu. Filling scaffolds with gene repetitions: Maximizing the number of adjacencies. In *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, pages 55–64, 2011.

[58] H. Jiang and B. Zhu. Weak kernels. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 17, 2010.

[59] H. Jiang, B. Zhu, and D. Zhu. Algorithms for sorting unsigned linear genomes by the dcj operations. *Bioinformatics*, 27(3):311–316, 2011.

[60] M. Jiang. The zero exemplar distance problem. *Journal of computational biology*, 18(9):1077–1086, 2011.

[61] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[62] R. S. Johnson and K. Biemann. The primary structure of thioredoxin from chromatium vinosum determined by high-performance tandem mass spectrometry. *Biochemistry*, 26(5):1209–1214, 1987.

[63] N. L. Kelleher. Top-down. 2004.

[64] R. J. Lewis and M. L. Garcia. Therapeutic potential of venom peptides. *Nature Reviews Drug Discovery*, 2(10):790–802, 2003.

[65] G. Lin and T. Jiang. A further improved approximation algorithm for breakpoint graph decomposition. *Journal of Combinatorial Optimization*, 8(2):183–194, 2004.

[66] N. Liu, H. Jiang, D. Zhu, and B. Zhu. An improved approximation algorithm for scaffold filling to maximize the common adjacencies. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 10(4):905–913, 2013.

[67] N. Liu, D. Zhu, H. Jiang, and B. Zhu. A 1.5-approximation algorithm for two-sided scaffold filling. *Algorithmica*, 74(1):91–116, 2016.

[68] X. Liu, L. J. Dekker, S. Wu, M. M. Vanduijn, T. M. Luider, N. Toli, Q. Kou, M. Dvorkin, S. Alexandrova, K. Vyatkina, et al. De novo protein sequencing by combining top-down and bottom-up tandem mass spectra. *Journal of proteome research*, 13(7):3241–3248, 2014.

[69] X. Liu, Y. Han, D. Yuen, and B. Ma. Automated protein (re)sequencing with ms/ms and a homologous database yields almost full coverage and accuracy. *Bioinformatics*, 25(17):2174, 2009.

[70] B. Ma. Novor: real-time peptide de novo sequencing software. *Journal of the American Society for Mass Spectrometry*, 26(11):1885–1894, 2015.

[71] 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 communications in mass spectrometry*, 17(20):2337–2342, 2003.

[72] B. Ma, K. Zhang, and C. Liang. An effective algorithm for peptide de novo sequencing from MS/MS spectra. *J. Comput. Syst. Sci.*, 70(3):418–430, 2005.

[73] C. C. Maley, P. C. Galipeau, J. C. Finley, V. J. Wongsurawat, X. Li, C. A. Sanchez, T. G. Paulson, P. L. Blount, R.-A. Risques, P. S. Rabinovitch, et al. Genetic clonal diversity predicts progression to esophageal adenocarcinoma. *Nature genetics*, 38(4):468, 2006.

[74] M. Mann and M. Wilm. Error-tolerant identification of peptides in sequence databases by peptide sequence tags. *Analytical chemistry*, 66(24):4390–4399, 1994.

[75] A. Marusyk, V. Almendro, and K. Polyak. Intra-tumour heterogeneity: a looking glass for cancer? *Nature Reviews Cancer*, 12(5):323, 2012.

[76] A. Muñoz, C. Zheng, Q. Zhu, V. A. Albert, S. Rounsley, and D. Sankoff. Scaffold filling, contig fusion and comparative gene order inference. *BMC bioinformatics*, 11(1):304, 2010.

[77] N. Navin, A. Krasnitz, L. Rodgers, K. Cook, J. Meth, J. Kendall, M. Riggs, Y. Eberling, J. Troge, V. Grubor, et al. Inferring tumor progression from genomic heterogeneity. *Genome research*, 20(1):68–80, 2010.

[78] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[79] C. G. A. R. Network et al. Integrated genomic analyses of ovarian carcinoma. *Nature*, 474(7353):609, 2011.

[80] C. T. Nguyen, Y. Tay, and L. Zhang. Divide-and-conquer approach for the exemplar breakpoint distance. *Bioinformatics*, 21(10):2171–2176, 2005.

[81] L. Oesper, A. Mahmoody, and B. Raphael. Theta: inferring intra-tumor heterogeneity from high-throughput dna sequencing data. *Genome Biology*, 14(7), 2013.

[82] S.-E. Ong and M. Mann. Mass spectrometry–based proteomics turns quantitative. *Nature chemical biology*, 1(5):252–262, 2005.

[83] J. J. Pesavento, C. A. Mizzen, and N. L. Kelleher. Quantitative analysis of modified proteins and their positional isomers by tandem mass spectrometry: human histone h4. *Analytical chemistry*, 78(13):4271–4280, 2006.

[84] P. A. Pevzner, V. Dančík, and C. L. Tang. Mutation-tolerant protein identification by mass spectrometry. *Journal of Computational Biology*, 7(6):777–787, 2000.

[85] S. Pietrokovski, J. G. Henikoff, and S. Henikoff. The blocks databasea system for protein classification. *Nucleic acids research*, 24(1):197–200, 1996.

[86] L. Pu, D. Zhu, and H. Jiang. A new approximation algorithm for unsigned translocation sorting. In *International Workshop on Algorithms in Bioinformatics*, pages 269–280. Springer, 2016.

[87] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM, 1997.

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

[89] L. Salmela, K. Sahlin, V. Mäkinen, and A. I. Tomescu. Gap filling as exact path length problem. In *Research in Computational Molecular Biology - 19th Annual International Conference, RECOMB 2015, Warsaw, Poland, April 12-15, 2015, Proceedings*, pages 281–292, 2015.

[90] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

[91] R. F. Schwarz, A. Trinh, B. Sipos, J. D. Brenton, N. Goldman, and F. Markowetz. Phylogenetic quantification of intra-tumour heterogeneity. *PLoS computational biology*, 10(4):e1003535, 2014.

[92] S. P. Shah, R. D. Morin, J. Khattra, L. Prentice, T. Pugh, A. Burleigh, A. Delaney, K. Gelmon, R. Guliany, J. Senz, et al. Mutational evolution in a lobular breast tumour profiled at single nucleotide resolution. *Nature*, 461(7265):809, 2009.

[93] R. Shamir, M. Zehavi, and R. Zeira. A linear-time algorithm for the copy number transformation problem. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 54. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[94] M. Shao and B. Moret. Comparing genomes with rearrangements and segmental duplications. *Bioinformatics*, 31(12):i329–i338, 2015.

[95] M. Shao and B. M. Moret. A fast and exact algorithm for the exemplar breakpoint distance. *Journal of Computational Biology*, 23(5):337–346, 2016.

[96] N. Siuti and N. L. Kelleher. Decoding protein modifications using top-down mass spectrometry. *Nature methods*, 4(10):817–821, 2007.

[97] H. Steen and M. Mann. The abc's (and xyz's) of peptide sequencing. *Nature reviews Molecular cell biology*, 5(9):699–711, 2004.

[98] J. C. Tran, L. Zamdborg, D. R. Ahlf, J. E. Lee, A. D. Catherman, K. R. Durbin, J. D. Tipton, A. Vellaichamy, J. F. Kellie, M. Li, et al. Mapping intact protein isoforms in discovery mode using top-down proteomics. *Nature*, 480(7376):254–258, 2011.

[99] P. Van Loo, S. Nordgard, O. Lingierde, H. Russnes, et al. Allele-specific copy number analysis of tumors. *PNAS*, 107(39):16910–16915, 2010.

[100] K. Vyatkina, S. Wu, L. J. Dekker, M. M. VanDuijn, X. Liu, N. Tolic, M. Dvorkin, S. Alexandrova, T. M. Luider, L. Pasa-Tolic, et al. De novo sequencing of peptides from top-down tandem mass spectra. *Journal of proteome research*, 14(11):4450–4462, 2015.

[101] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[102] J. Yates 3rd, J. K. Eng, A. L. McCormack, and D. Schieltz. Method to correlate tandem mass spectra of modified peptides to amino acid sequences in the protein database. *Analytical chemistry*, 67(8):1426–1436, 1995.

[103] S. Zaccaria, E.-K. Mohammed, K. Gunnar, and B. Raphael. The copy-number tree mixture deconvolution problem and applications to multi-sample bulk sequencing tumor data. In *Proceedings of RECOMB'2017, LNCS*, volume 10229, pages 318–335. Springer, 2017.

[104] B. Zhu. Genomic scaffold filling: a progress report. In *International Workshop on Frontiers in Algorithmics*, pages 8–16. Springer, 2016.

APPENDIX: THE BLOSUM62 SCORE MATRIX

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	9																			
S	-1	4																		
T	-1	1	5																	
P	-3	-1	-1	7																
A	0	1	0	-1	4															
G	-3	0	-2	-2	0	6														
N	-3	1	0	-2	-2	0	6													
D	-3	0	-1	-1	-2	-1	1	6												
E	-4	0	-1	-1	-1	-2	0	2	5											
Q	-3	0	-1	-1	-1	-2	0	0	2	5										
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8									
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5								
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5							
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5						
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4					
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4				
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4			
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6		
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7	
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11