CHEN, WENBIN. Complexity Theory and Algorithms for Graph Problems Driven by Comparative Analysis of Large-Scale Biological Networks. (Under the direction of Professor Nagiza F. Samatova.)

Many large-scale biological networks analysis problems reduce to problems on graphs, for which neither the underlying theory of their computational complexity nor the efficient algorithms are well studied. The three broad classes of such problems include: (1) the alignment of multiple genome-scale biological pathways or networks; (2) the discovery of patterns of interest in such networks; and (3) the assignment of orthologous genes.

For the first class of problems—the multiple alignment of biological networks—this research answers open questions on the parameterized complexity of the problem and proposes efficient and biologically relevant algorithms to solve it. First, for general graphs, we introduce the problem of finding the Maximum Common Subgraph (MCS) of Multiple Graphs that we refer to as the Multi-MCS problem. Based on the framework of the parameterized complexity theory, we prove parameterized complexity hardness results for the different parameterized versions of Multi-MCS. Second, when the graphs are trees, we introduce the problems of finding the Maximum Common Subtree (MCT) and the Minimum Common Supertree (MCST) of Multiple Trees that we refer to as Multi-MCT and Multi-MCST, respectively. We prove parameterized complexity hardness results for the different parameterized versions of Multi-MCT and Multi-MCST. Third, for the graphs with positive and negative edge weights (e.g., log-likelihood scores), we investigate the Maximum Weight Induced Subgraph problem (MWIS) of finding an induced subgraph with the maximum total weight in a weighted graph. We propose a fast greedy heuristic algorithm for MWIS. When compared with the state-of-the-art in the field, the proposed algorithm is an order of magnitude faster and produces biologically more relevant results. Fourth, we address the challenging problem of aligning multiple metabolic pathways. We propose the first algorithm for the problem of aligning multiple metabolic pathways based on the similarities among reactions, compounds, enzymes, and pathway topology. For the second class of problems—the discovery of patterns of interest in large-scale networks—we study the densest at-least-k-subgraph problem (DalkS) and the densest at-most-k-subgraph problem (DamkS). We present two polynomial time algorithms for DalkS when k is bounded by some constant c. We also present two approximation algorithms for DamkS. For the third class of problems—the assignment of orthologous genes—we study the maximum duo-preservation string mapping problem (MPSM). Using Linear Programming technology, we propose approximation algorithms for this problem.
Complexity Theory and Algorithms for Graph Problems Driven by Comparative Analysis of Large-Scale Biological Networks

by

Wenbin Chen

A dissertation submitted to the Graduate Faculty of North Carolina State University
in partial fulfillment of the requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Professor Carla Savage
Professor Matthias Stallmann

Professor Steffen Heber
Professor Nagiza F. Samatova
Chair of Advisory Committee
BIOGRAPHY

Wenbin Chen was born in Chaling county, Hunan Province in P. R. China. After graduating from National University of Defense Technology, he was enrolled at Institute of Software, Chinese Academy of Sciences in 2000, from which he received his M.S. degree in Mathematics in 2003. From 8/2003-7/2007, he worked in computer science department, Nanjing University of Aeronautics and Astronautics. He was enrolled in North Carolina State University’s Ph.D. program in 8/2007. He was awarded the NCSU alumni fellowship in 2007. He successfully defended his Written Examination in 12/2008 and Oral Preliminary Examination in 02/2009.
ACKNOWLEDGEMENTS

First, I would like to thank Prof. Nagiza Samatova, my advisor. She has been a great mentor guiding me through my research. I am also indebted to Profs. Carla Savage, Matthias Stallmann, and Steffen Heber for serving on my thesis committee. I would like to thank the following people for their continued support: Matt Schmidt, Wenhong Tian, William Hendrix, Andrea Rocha, Zhengzhang Chen and others in Prof. Samatova’s research group. Finally, I am thankful to my parents, brothers, sisters and friends, for their continuous encouragement and support.

The work of Wenbin Chen was funded by the U.S. Department of Energy (Office of Science) through the contract with the Oak Ridge National Laboratory. Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725.
# TABLE OF CONTENTS

List of Tables .............................................................. vii

List of Figures ............................................................. ix

Chapter 1 Introduction .................................................. 1
   1.1 Motivation, Problems, and Contributions ..................... 1
      1.1.1 The Maximum Common Subgraph Problem for Multiple Graphs (Multi-
             MCS) .............................................................. 2
      1.1.2 The Maximum Common Subtree and the Minimum Common Supertree
             for Multiple Trees .................................................. 3
      1.1.3 The Maximum Weight Induced Subgraph (MWIS) Problem .......... 4
      1.1.4 The Densest at-least-\(k\)-subgraph Problem (Dal\(k\)S) and the Densest
             at-most-\(k\)-subgraph Problem (Da\(m\)k\(S\)) ..................... 5
      1.1.5 The Multiple Alignment Problem for Metabolic Pathways without Ab-
             straction .......................................................... 5
      1.1.6 The Maximum Duo-preservation String Mapping Problem (MPSM) ... 6
   1.2 Publications Summary ............................................. 7

Chapter 2 On Parameterized Complexity of Multi-MCS Problem .... 9
   2.1 Introduction ......................................................... 9
   2.2 Parameterized Complexity ........................................ 10
   2.3 Parameterizations of Multi-MCS Problem ....................... 12
   2.4 Background Problem Definitions .................................. 12
   2.5 Parameterized complexity of Multi-MCS for labeled directed and undirected
         graphs via parameterized reduction from LCS ................. 14
   2.6 Parameterized complexity of Multi-MCS for unlabeled undirected graphs via
         parameterized reduction with CLIQUE ............................. 17
   2.7 Discussion .......................................................... 21

Chapter 3 On the Parameterized Complexity of the Multi-MCT and Multi-
MCST Problems .......................................................... 23
   3.1 Introduction ......................................................... 23
   3.2 Definitions of Problems ........................................... 24
   3.3 Parameterized Complexity of the Multi-MCST Problem for Vertex-Labeled, Or-
       dered Trees ............................................................ 26
   3.4 Parameterized Complexity of the Multi-MCT Problem for Vertex-Labeled, Or-
       dered Trees ............................................................ 26
   3.5 Parameterized Complexity of the Multi-MCT Problem for Vertex-Labeled, Un-
       ordered Trees ......................................................... 27
   3.6 Conclusion .......................................................... 29
Chapter 4 The Greedy Algorithm for Maximum Weight Induced Subgraph Problem

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>30</td>
</tr>
<tr>
<td>4.1.1 Contribution</td>
<td>32</td>
</tr>
<tr>
<td>4.2 Previous methods of finding high scoring local alignment networks</td>
<td>32</td>
</tr>
<tr>
<td>4.3 Our approach to finding high scoring local alignment networks</td>
<td>34</td>
</tr>
<tr>
<td>4.4 Experimental Results</td>
<td>35</td>
</tr>
<tr>
<td>4.5 Concluding Remarks</td>
<td>38</td>
</tr>
</tbody>
</table>

Chapter 5 On Finding Dense Subgraphs with Size Constraints

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>41</td>
</tr>
<tr>
<td>5.2 Definitions</td>
<td>43</td>
</tr>
<tr>
<td>5.3 The Minimum s-t Cut</td>
<td>44</td>
</tr>
<tr>
<td>5.4 An algorithm for DalkS based on the minimum s-t cut with at least k vertices</td>
<td>46</td>
</tr>
<tr>
<td>5.5 An LP-based algorithm for DalkS when k is bounded by some constant c</td>
<td>48</td>
</tr>
<tr>
<td>5.6 Approximation Algorithms for DalkS</td>
<td>54</td>
</tr>
<tr>
<td>5.7 Concluding Remarks</td>
<td>56</td>
</tr>
</tbody>
</table>

Chapter 6 The Multiple Alignment Algorithm for Metabolic Pathways without Abstraction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Introduction</td>
<td>57</td>
</tr>
<tr>
<td>6.2 The Algorithm for the Alignment Problem of Multiple Pathways</td>
<td>59</td>
</tr>
<tr>
<td>6.3 A Polynomial Time Algorithm for the Maximum-Weighted (1, r)-Matching Problem in a Bipartite Graph</td>
<td>62</td>
</tr>
<tr>
<td>6.4 An Approximation Algorithm for the Maximum-Weighted k-partite Matching Problem</td>
<td>64</td>
</tr>
<tr>
<td>6.5 A Heuristic Algorithm for the Maximum-weighted k-partite Matching Problem</td>
<td>67</td>
</tr>
<tr>
<td>6.6 Comparison of Results for Several Algorithms</td>
<td>68</td>
</tr>
<tr>
<td>6.7 Alignment Results of Metabolic Pathways</td>
<td>71</td>
</tr>
<tr>
<td>6.7.1 Biological Significance</td>
<td>71</td>
</tr>
<tr>
<td>6.8 Conclusion</td>
<td>73</td>
</tr>
</tbody>
</table>

Chapter 7 An Approximation Algorithm for the Maximum Duo-preservation String Mapping Problem (MPSM)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Introduction</td>
<td>77</td>
</tr>
<tr>
<td>7.2 Preliminaries</td>
<td>80</td>
</tr>
<tr>
<td>7.3 A Polynomial Time Reduction from the MPSM Problem to the CMIS Problem</td>
<td>82</td>
</tr>
<tr>
<td>7.4 An Approximation Algorithm for the Constrained Maximum Induced Subgraph (CMIS) Problem</td>
<td>83</td>
</tr>
<tr>
<td>7.5 Improved Approximation Algorithms for 2-CMIS, 3-CMIS, and 4-CMIS</td>
<td>87</td>
</tr>
<tr>
<td>7.6 A 0-1 Quadratic Integer Programming Formulation for the Minimum Common String Partition Problem (MCSP)</td>
<td>90</td>
</tr>
<tr>
<td>7.7 Conclusion</td>
<td>92</td>
</tr>
</tbody>
</table>

Chapter 8 Conclusion

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>93</td>
</tr>
</tbody>
</table>
Table 2.1 Parameterized complexity results of the parameterized versions of the LCS and CLIQUE problem[13, 15, 14] ............................ 14
Table 2.2 Parameterized complexity results of the parameterized versions of the Multi-MCS problem for directed graphs with vertex labels .............. 16
Table 2.3 Parameterized complexity of parameterized versions of the Multi-MCS problem for unlabeled undirected graphs obtained via linear FPT reductions to and from the m-CLIQUE problem ............................ 21
Table 2.4 Parameterized complexity results of the parameterized versions of the Multi-MCS problem for unlabeled undirected graphs ........................ 21
Table 3.1 The different parameterized versions of the MULTI-MCT, MULTI-MCST, LCS, CLIQUE, SCS problems and their associated parameters ............ 25
Table 3.2 Parameterized complexity results of the parameterized versions of the LCS, SCS and CLIQUE problem ..................................... 25
Table 3.3 Parameterized complexity results of the parameterized versions of the MULTI-MCST problem for vertex-labeled, ordered trees .............. 27
Table 3.4 Parameterized hardness results of the parameterized versions of the MULTI-MCT problem for vertex-labeled, ordered trees .............. 28
Table 3.5 Parameterized complexity class hardness results for parameterized MULTI-MCT and MULTI-MCST problems ............................ 29
Table 4.1 Comparison of some previous network alignment algorithms ............. 31
Table 4.2 Comparison of local alignment networks produced and runtime for 7 pairs of PPI networks .................................................. 36
Table 4.3 Comparison of precision-recall results for 7 pairs of PPI networks .... 40
Table 6.1 The mean value and standard deviation for weights of matchings in 3-partite graphs .................................................. 69
Table 6.2 The mean value and standard deviation for weights of matchings in 4-partite graphs .................................................. 69
Table 6.3 The mean value and standard deviation for weights of matchings in 5-partite graphs .................................................. 69
Table 6.4 The mean value and standard deviation for weights of matchings in 6-partite graphs .................................................. 70
Table 6.5 The mean value and standard deviation for weights of matchings in 7-partite graphs .................................................. 70
Table 6.6 The mean value and standard deviation for weights of matchings in 8-partite graphs .................................................. 70
Table 6.7 Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five aerobic organisms. Only the alignments for enzymes are presented in this table. Each row in the table below represents an alignment of a single enzyme across the organisms Bordetella bronchiseptica RB50 (bbr), Staphylococcus saprophyticus subsp. saprophyticus ATCC 15305 (ssp), Myxococcus xanthus DK 1622 (mxa), Leptospira interrogans serovar Lai str. 56601 (lil), and Helicobacter pylori HPAG1 (hpa). The + entries in the table indicate which enzyme was identified as being aligned for that organism, whereas ◆ indicates that no aligned enzyme was identified for that organism. Enzymes marked with ⋆ are part of the TCA cycle.

Table 6.8 Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five aerobic organisms. The parts of alignments for reactions and compounds are presented in this table.

Table 6.9 Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five facultative anaerobic organisms. Only the alignments for enzymes are presented in this table. Each row in the table below represents an alignment of a single enzyme across the organisms Escherichia coli (eco), Shewanella oneidensis MR-1 (son), Zymomonas mobilis (zmo), Listeria innocua (lin), and Streptococcus pneumonia (spn). The + entries in the table indicate which enzyme was identified as being aligned for that organism, whereas ◆ indicates that no aligned enzyme was identified for that organism. Enzymes marked with ⋆ are part of the TCA cycle.

Table 7.1 The approximation ratio summary for k-MCSP.
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of the reduction of an instance of the LCS problem to a Multi-MCS problem. The letters in the vertices of the graphs are the labels of the vertices. One can see how the maximum common subgraph of the three graphs maps to the longest common subsequence of the three strings.</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>The cumulative probability distribution of the likelihood that a KEGG pathway is at least a given size.</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>The cumulative probability distribution of the likelihood that a LAN found by an algorithm is at least a given size.</td>
<td>37</td>
</tr>
<tr>
<td>7.1</td>
<td>A letter-preserving bijective mapping $\pi$ for two strings: $A = abcab$, $B = ababc$.</td>
<td>78</td>
</tr>
<tr>
<td>7.2</td>
<td>Two strings $A = abcab$ and $B = ababc$ are transformed into a graph $G_{AB}$.</td>
<td>82</td>
</tr>
<tr>
<td>7.3</td>
<td>Two strings $A = abcab$ and $B = ababc$ are transformed into a graph $\bar{G}_{AB}$.</td>
<td>91</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation, Problems, and Contributions

Graphs are powerful and useful data structures. In different subfields of computer science, graphs can represent many objects. In a graph representation, the nodes of the graph generally represent parts of objects, while the edges of the graph denote the relations between these parts. Graphs are widely used in a variety of application areas. For example, there are many applications in machine learning, computer vision, pattern recognition, image analysis, information retrieval, and bioinformatics.

The foundational complexity theory and graph algorithms researched in this thesis are primarily inspired by the emerging field of comparative analysis of genome-scale biological networks. With the growing advancements in high-throughput experimental technologies and bioinformatics methods, many different types of biological networks are being reconstructed on a genome-scale, including metabolic networks, gene regulatory networks, signal transduction pathways, and protein-protein interaction networks. Comparative analysis of these networks—both across multiple genomes and across different types—is being recognized as a critical stepping stone toward understanding of how individual components of a biological system interact to form complex systems that dynamically adapt in response to various environmental and evolutionary perturbations.

Many large-scale biological networks analysis problems reduce to problems on graphs, for which the underlying theory of their computational complexity and the efficient algorithms are poorly understood. The three broad classes of such problems include: (1) the alignment of multiple genome-scale biological pathways or networks; (2) the discovery of patterns of interest in such networks; and (3) the assignment of orthologous genes. Depending on how the biological pathways or networks are mathematically modeled as graphs, these biological problems reduce to various kinds of graph problems.
For the first class of problems—the multiple alignment of biological networks—this research thesis answers open questions on the parameterized complexity of the problem and proposes efficient and biologically relevant algorithms to solve it. Based on the framework of parameterized complexity theory, we derive parameterized complexity of this problem for various parameters on different classes of graphs, including general graphs (Chapter 2) and trees (Chapter 3). We propose an algorithm, which is an order of magnitude faster and more accurate than the state-of-the-art, for finding the local pairwise alignment of protein-protein interaction networks with weighted edges (Chapter 4). We also propose the first algorithm for the problem of multiple alignment of metabolic pathways without topological constraints (Chapter 6). For the second class of problems—the discovery of patterns of interest in large-scale networks—we present two polynomial time algorithms for the densest at-least-\(k\)-subgraph problem (\textsc{DalkS}) when \(k\) is bounded by some constant \(c\) and propose two approximation algorithms for the densest at-most-\(k\)-subgraph problem (\textsc{DamkS}), with no prior approximations known-to-date (Chapter 5). For the third class of problems—the assignment of orthologous genes—we propose an approximation algorithm for the maximum duo-preservation string mapping problem (Chapter 7). Next, we briefly summarize our major findings and contribution to the field.

1.1.1 The Maximum Common Subgraph Problem for Multiple Graphs (Multi-MCS)

Finding common motifs among multiple objects represented as graphs is a widely known problem in many fields. Specific applications involve matching of multiple protein structures, multiple alignment of protein-protein interaction networks, or interpretations of molecular spectra. While mostly restricted to only two objects, algorithms that assist with these tasks often reduce to the problem of finding the maximum common subgraph of two graphs (MCS). For achieving higher accuracy, reducing noise and gaining novel scientific insights, the community has recognized the benefit of generalizing these applications to multiple graphs of different properties, such as directed and labeled graphs (e.g., matching metabolic pathways) or undirected unlabeled graphs (e.g., multiple alignment of protein structures). To address these needs, it is required to solve the problem of finding the maximum common subgraph of a set of \(k\) graphs that we refer to as the \textsc{Multi-MCS} problem.

Some applications of the Multi-MCS problem, however, do not require an algorithm that can solve all instances of the problem. Some applications are concerned only with instances of the Multi-MCS problem, in which certain parameters of the problem are bounded, or fixed.

In Chapter 2, we show the complexity of these types of parameterized versions of the Multi-MCS problem. We give linear fixed parameter tractable (FPT) reductions of the parameterized versions of the longest common subsequence (LCS) problem to parameterized versions of the
Multi-MCS problem for directed graphs with vertex labels. We also give linear FPT reductions of the parameterized versions of the maximum clique problem to parameterized versions of the Multi-MCS problem for unlabeled undirected graphs. These linear FPT reductions give lower bounds on the parameterized complexity of these versions of the Multi-MCS problem.

The results of this chapter have been published in the Journal *Theoretical Computer Science* [23].

### 1.1.2 The Maximum Common Subtree and the Minimum Common Supertree for Multiple Trees

In some cases, tree structures are used to represent biological data in order to simplify the computation. For example, in order to reduce the complexity of computation, a pathway is represented as an ordered, labeled tree [87]. Thus, it is necessary to study the subcase of the Multi-MCS problem: the maximum common subtree problem for multiple trees (Multi-MCT). When input graphs are trees, the Multi-MCS problem becomes the maximum common subtree problem for multiple trees (Multi-MCT). A supertree is any tree for which the original tree is a subtree. The problem of finding the minimum common supertree for multiple trees (Multi-MCST) is the dual problem of Multi-MCT. Multi-MCST also has various applications. For instance, the Multi-MCST of the ordered, labeled trees would provide a method for computing an evolutionary tree [105].

Akutsu [1] proved that the Multi-MCT problem for unordered, vertex-labeled trees is NP-hard. It is also currently unknown if a solution to the Multi-MCT problem can generally be used to generate a solution to the Multi-MCST problem. While algorithms that solve the Multi-MCT and Multi-MCST problem are unlikely in the general case, problems that arise from real-world applications may have efficient solutions. For example, if the input trees represent some structure like the carbohydrate sugar chains [5], then the alphabet of vertex-labels would be limited to monosaccharides, of which there are relatively few. In such cases, fixing the size of certain parameters may keep the complexity of solving such problems from exploding exponentially. The parameterized complexity classes introduced by Downey and Fellows in [32] define which problems are likely to have efficient solutions when certain parameters are fixed. Determining the parameterized complexity of the Multi-MCT and Multi-MCST problems can help in determining if efficient algorithms can be developed to solve real-world instances of the Multi-MCT and Multi-MCST problem.

In Chapter 3, we study the parameterized complexity of Multi-MCT and Multi-MCST problems. We get the parameterized complexity results by showing a parameterized reduction from the LCS problem to the Multi-MCT problem for ordered, vertex-labeled trees, a parameterized reduction from the CLIQUE problem to the Multi-MCT problem for unordered, vertex-labeled
trees, and a parameterized reduction from the shortest common subsequence (SCS) problem to the Multi-MCST problem for ordered, vertex-labeled trees.

The results of this chapter have been published in the *Journal of Combinatorial Optimization* [24].

1.1.3 The Maximum Weight Induced Subgraph (MWIS) Problem

Third, we consider the graph problem driven by the alignment of the protein-protein interaction (PPI) networks guided by models of evolution. In [77], in order to compute the pairwise local alignment of protein interaction networks guided by models of evolution, M. Koyutürk et al reduce the alignment problem to the Maximum Weight Induced Subgraph Problem (MWIS). M. Koyutürk et al have developed a heuristic algorithm (called MaWish) that attempts to find conserved functional protein modules by finding locally maximal alignment networks in the global alignment network defined in [77].

The MaWish algorithm heuristically grows a locally maximal subgraph. The heuristic algorithm presented has the drawback of producing relatively small local alignment networks, which make them biologically less relevant. The bottom-up method of growing subgraphs utilized by the MaWish algorithm makes it difficult to grow large subgraphs. The criteria that the vertex being added to the subgraph must be connected to the current subgraph by edges whose sum is greater than zero is a tight restriction. It requires the aligned pair of proteins represented by a vertex in the subgraph to have a large number of matching interactions. This means that proteins in the conserved functional module found by this algorithm must interact with the vast majority of other proteins in the functional module. If the functional module contains a module hub, or if the protein-protein interaction data contains a great deal of noise, then the size of the local alignment networks will be limited to relatively small numbers of proteins. Small local alignment networks are unlikely to correspond to actual conserved function modules.

In Chapter 4, we present a new greedy, heuristic algorithm for the maximum weight induced subgraph problem. We implement and use the algorithm to find the local alignment of pairwise PPI networks by models of evolution. We compare our results with MaWish algorithm. Experimental results show that our greedy algorithm is both more accurate and faster than MaWish.

The results of this chapter have been published in the Conference *International Conference on Bioinformatics and Computational Biology 2009* [25].
1.1.4 The Densest at-least-\(k\)-subgraph Problem (DalkS) and the Densest at-most-\(k\)-subgraph Problem (DamkS)

Fourth, we study the problems that are closely related to a subcase of MWIS problems. Given an undirected graph with \(n\) vertices, the density of its subgraph is the sum of subgraph’s edge weights divided by the number of subgraph’s vertices. For the MWIS problem, when all edge weights are positive, its parameterized version is the famous dense \(k\)-subgraph problem, which is to find the densest induced subgraph of exactly \(k\) vertices. The dense \(k\)-subgraph problem is closely related to the densest at-least-\(k\)-subgraph problem (DalkS) and the densest at-most-\(k\)-subgraph problem (DamkS) introduced by Andersen [4]. The densest at-least-\(k\)-subgraph problem (DalkS) is the problem of finding an induced subgraph of maximum density among all subgraphs with at least \(k\) vertices. The densest at-most-\(k\)-subgraph problem (DamkS) is the problem of finding an induced subgraph of maximum density among all subgraphs with at most \(k\) vertices.

In Chapter 5, we introduce the definition of a minimum cut with at least \(k\) vertices and give a polynomial time algorithm for DalkS when \(k\) is bounded by some constant \(c\). Based on the algorithm for the minimum cut with at least \(k\) vertices, we propose one polynomial time algorithms for DalkS when \(k\) is bounded by some constant \(c\). We also propose an LP-based polynomial time algorithm for DalkS when \(k\) is bounded by some constant \(c\). We also present two approximation algorithms for DamkS. The first approximation algorithm gives an approximation ratio of \(\frac{n-1}{k-1}\), and the second has ratio \(O(n^{\delta})\) for some \(\delta < 1/3\).

The results of this chapter have been accepted for publication in the *International journal of computer science and information technology* [22].

1.1.5 The Multiple Alignment Problem for Metabolic Pathways without Abstraction

Fifth, the problem of computing the similarities between multiple pathways is referred to as the problem of multiple pathway alignment. Tohsato et al. proposed an algorithm to align multiple metabolic pathways based on the similarity of enzymes [103]. This algorithm, however, does not consider similarities in topology or other entities, such as compounds and reactions.

In order to overcome this shortcoming, in Chapter 6, we propose the first algorithm for the problem of aligning multiple metabolic pathways based on the similarities between enzymes, compounds, reactions, and topological structure. In order to compute the alignment of multiple pathways, we create a weighted \(k\)-partite graph for each type of entity and reduce the problem of aligning multiple metabolic pathways to the maximum-weighted \(k\)-partite matching problem. Before proposing our algorithms, we introduce the maximum-weighted \((1, r)\)-matching problem
in a bipartite graph and give a polynomial time algorithm for this problem. Based on the algorithm for the maximum-weighted \((1, r)\)-matching problem, we will give an approximation algorithm and a heuristic algorithm for the maximum-weighted \(k\)-partite matching problem. We apply our heuristic algorithm to compute the alignment of multiple metabolic pathways. Our experiments show that our algorithm can correctly identify common subnetworks among multiple pathways, which can be used to find conserved entities.

The results of this chapter have been accepted for publication in the Conference *IEEE International Conference on Data Mining Workshops 2010* [21].

### 1.1.6 The Maximum Duo-preservation String Mapping Problem (MPSM)

Finally, in comparative genomics area, an important computational problem is the assignment of orthologous genes. In [26], the assignment of orthologous genes is reduced to the problem of computing the signed reversal distance with duplicates genes between two genomes. In order to compute the signed reversal distance with duplicates genes between two genomes, the minimum common string partition problem (MCSP) and the maximum cycle decomposition problem have been introduced in [26].

The minimum common string partition problem (MCSP) has been well-investigated as a fundamental problem in computer science [26, 49]. Given two finite length strings over the finite letter alphabet, MCSP is to partition strings into identical substrings with the minimum number of partitions. MCSP is also viewed as the problem of finding a letter-preserving bijective mapping \(\pi\) from letters in one string \(A\) to letters in the other string \(B\) with the minimum number of breaks, where a *letter-preserving bijective mapping* \(\pi\) means that each letter in \(A\) is mapped into the same letter in \(B\) and the mapping is bijective, and a break is a pair of consecutive letters in \(A\) that are mapped by \(\pi\) to non-consecutive letters in \(B\) [49]. In a string, a pair of consecutive letters is called a *duo* [49].

For a letter-preserving bijective mapping between two strings, on the one hand, the optimization goal can be to minimize the number of breaks that is known as the MCSP problem. On the other hand, the optimization goal can be to maximize the number of duo-preservations. We define the maximization version of the problem as the maximum duo-preservation string mapping problem (MPSM), i.e. the problem of finding a letter-preservation bijective mapping \(\pi\) from one string to the other string with the maximum number of duo-preservations. The restricted version of MPSM, where each letter occurs at most \(k\) times in each input string, is denoted by \(k\)-MPSM. The MPSM problem is complementary to the MCSP problem. From this complementary relationship, it follows that MPSM is also \(NP\)-hard since the MCSP problem is \(NP\)-hard [49].

In Chapter 7, we propose an approximation algorithm for MPSM based on the randomized
rounding technology. We prove that the approximation algorithm is of approximation ratio $k^3$ for $k$-MPSM. We still propose an 2-approximation algorithm for 2-MPSM, a 9-approximation algorithm for 3-MPSM, and a 16-approximation algorithm for 4-MPSM. We also give a 0-1 quadratic Integer Programming formulation for the MCSP problem to pave the way for new approximation algorithms of MCSP using the randomized rounding method.

The results of this chapter have been accepted for publication in the Conference *the International Conference on Future Information Technology 2010* [20].

### 1.2 Publications Summary

Chapter 2:


Chapter 3:


Chapter 4:


Chapter 5:

4. **Wenbin Chen**, Nagiza F. Samatova, Matthias Stallmann, William Hendrix. On finding dense subgraphs with size constraints; Accepted to *International journal of computer science and information Technology*.

Chapter 6:


Chapter 7:


Chapter 2

On Parameterized Complexity of Multi-MCS Problem

2.1 Introduction

Finding common motifs among multiple objects represented as graphs is a widely known problem in different fields including image processing, pattern recognition, semantic networks, and bioinformatics. Specific applications in chemistry and biology, for example, involve matching of multiple 3D chemical or protein structures, multiple alignment of protein-protein interaction networks, or interpretations of molecular spectra. While mostly restricted to only two objects, algorithms that assist with these tasks often reduce to the problem of finding the maximum common subgraph of two graphs (MCS) [60, 89, 94, 109]. For achieving higher accuracy, reducing noise and gaining novel scientific insights, the community has recognized the benefit of generalizing these applications to multiple graphs of different properties such as directed and labeled graphs (e.g. matching metabolic pathways) or undirected unlabeled graphs (e.g. multiple alignment of protein structures).

To address these needs, it is required to solve the problem of finding the maximum common subgraph of a set of $k$ graphs that is refer to as the Multi-MCS problem. The Multi-MCS problem can be defined formally as:

**Definition 2.1.1** For a given set of graphs $H = \{G_1, G_2, \ldots, G_k\}$, what is the maximum size of a graph $G_{Max}$ that is isomorphic to a subgraph of every $G_i$ for $1 \leq i \leq k$?

The size of $G_{Max}$ can be measured as either the number of vertices or the number of edges in $G_{Max}$. If the size is measured by the number of vertices of $G_{Max}$ then the problem is referred to as the Multi-MCIS (Maximum Common Induced Subgraph for Multiple Graphs) problem. If the size is measured by the number of edges of $G_{Max}$ then the problem is referred to as the
Multi-MCES (Maximum Common Edge Subgraph for Multiple Graphs) problem. These are analogous to the well-known Maximum Common Induced Subgraph and Maximum Common Edge Subgraph problem for two graphs (MCIS and MCES respectively). Results from [89] show that the MCIS problem is equivalent to the MCES problem. When we refer to the Multi-MCS problem in this thesis we actually referring to the Multi-MCIS problem.

It is simple to see that the MCS problem is reducible to the Multi-MCS problem by fixing \( k = 2 \). Since the MCS problem is known to be \( NP \)-hard [65], then the Multi-MCS problem must be \( NP \)-hard as well. Therefore, it is unlikely that there exists a general algorithm that gives exact solutions to the Multi-MCS problem in practical time for large graphs.

Some applications of the Multi-MCS problem, however, do not require an algorithm that can solve all instances of the problem. Some applications are concerned only with instances of the Multi-MCS problem in which certain parameters of the problem are bounded, or fixed. Our study deals with the complexity of these types of parameterized versions of the Multi-MCS problem. We give linear fixed parameter tractable (FPT) reductions of the parameterized versions of the longest common subsequence (LCS) problem presented in [15] to parameterized versions of the Multi-MCS problem for directed graphs with vertex label. We also give linear FPT reductions of two parameterized versions of the maximum clique problem to parameterized versions of the Multi-MCS problem for unlabeled undirected graphs. These linear FPT reductions give lower bounds on the parameterized complexity of these versions of the Multi-MCS problem.

The remainder of the chapter is organized as follows. Section 2.2 reviews the concept of linear FPT reductions. Section 2.3 gives an overview of the parameterized versions of the Multi-MCS problem that we will be focusing on. Section 2.4 introduces the parameterized problems that we will use in our reductions and discusses their parameterized complexity. Section 2.5 shows the linear FPT reductions of the parameterized versions of the LCS problem to the parameterized versions of the Multi-MCS problem for directed graphs with vertex label. Section 2.6 proves that parameterized version of the Multi-MCS problem for unlabeled undirected graphs are \( W[1] \)-hard by showing linear FPT reductions of the problem from the parameterized maximum clique problem. It also shows that a special case of one of the parameterized problems is \( W[1] \)-complete. Finally Section 2.6 discusses these results as well as some open problems inspired by these findings.

### 2.2 Parameterized Complexity

The classical complexity notion of \( NP \)-hardness provides a general notion of which problems are computationally intractable to solve unless \( P = NP \). If a problem is proven to be \( NP \)-hard, then it is unlikely that any algorithm exists that can solve every instance of the problem in
polynomial-time. However, for some $NP$-hard problems, only certain instances of the problem are inefficient to solve. Other instances of these problems may be actually be solvable in polynomial-time.

Often, a subset of problem instances for which there exists a polynomial-time algorithm is defined by fixing the value of a certain parameter of the problem. For example, a solution to the Vertex Cover problem is the smallest set of vertices of a graph such that every edge in the graph has at least one endpoint in the set. The Vertex Cover problem is known to be $NP$-hard [43]. However, when only considering the subset of Vertex Cover problems where the size of vertex cover solution is known to be $k$ and the number of vertices in the graph is $n$, there exists an algorithm that can solve the problem instance in $O(1.2745^k k^4 + kn)$ [83].

Generally, a parameterized problem is defined as follows.

**Definition 2.2.1** For a given alphabet $\Sigma$, a parameterized problem $L$ is a set of pairs $(x, k) \in \Sigma^* \times \mathbb{N}$, such that for every $(x, k) \in L$ there is no $(x, k') \in L$ for any $k' \neq k$.

A parameterized problem is considered fixed-parameter tractable (FPT) if there exists an algorithm that solves it in $O(f(k) * n^{O(1)})$ time, where $n = |x|$ and $k = |k|$. In addition, a parameterized problem can be shown to be FPT if there is an FPT-reduction of a problem known to be FPT to that problem. The concept of FPT-reducibility is formally defined as follows.

**Definition 2.2.2** A parameterized problem $L$ is FPT-reducible to a parameterized problem $L'$ if there exists a function $f(k)$ and an algorithm $\Phi(x, k)$ such that for every instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $\Phi$ produces a pair $(x', k')$, where $(x', k') \in L'$ if and only if $(x, k) \in L$, and $\Phi$ runs in time $O(f(k) * n^{O(1)})$.

Downey and Fellows established the $W$-hierarchy for classifying parameterized problems according to their complexity in [30]. The classes of parameterized problems in the $W$-hierarchy include the class FPT and the classes $W[1], W[2], \ldots, W[t], \ldots$ for every $t \geq 1$. The results of [30] show that the parameterized complexity classes in the $W$-hierarchy have the relation $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots$. The split between fixed-parameter tractable problems and fixed-parameter intractable problems is generally assumed to be between the classes FPT and $W[1]$.

A parameterized problem is $W[1]$-hard if it is at least as hard as every other problem in $W[1]$. If $P \neq NP$, there is no FPT algorithms for all $W[1]$-hard problems.

For every class $W[t]$, there is a set of complete problems [30, 83]. If a problem is complete for a complexity class, then there is an FPT-reduction of every problem in that complexity class to the complete problem. If a $W[t]$-complete problem reduces to another problem, then the problem to which it reduces is considered $W[t]$-hard. If a problem is $W[t]$-hard, then it
cannot exist in the class $W[t']$ where $t' < t$ unless the $W$-hierarchy collapses so that $FPT = W[1] = W[2] = \ldots$.

### 2.3 Parameterizations of Multi-MCS Problem

The Multi-MCS problem can be parameterized in a number of different ways. In this chapter, we will focus on the following four parameterizations of the Multi-MCS problem.

**Definition 2.3.1 (m-Multi-MCS)** - For a given set of graphs $H$ and an integer $m$, is there a graph $G_{Max}$ whose size is at least $m$ that is isomorphic to a subgraph of every graph in $H$?

**Definition 2.3.2 (k-Multi-MCS)** - For a given set of graphs $H$ and an integer $k$, if $|H| = k$, what is the maximum size of a graph $G_{Max}$ that is isomorphic to a subgraph of every graph in $H$?

**Definition 2.3.3 (km-Multi-MCS)** - For a given set of graphs $H$, an integer $k$, and an integer $m$, if $|H| = k$, is there a graph $G_{Max}$ whose size is at least $m$ that is isomorphic to a subgraph of every graph in $H$?

**Definition 2.3.4 (k$\Gamma$-Multi-MCS)** - For a given set of graphs $H$, an integer $k$, and an alphabet $\Gamma$, if $|H| = k$ and the vertices of the graphs of $H$ are labeled by the alphabet $\Gamma$, what is the maximum size of a graph $G_{Max}$ that is isomorphic to a subgraph of every graph in $H$?

In addition to these parameterized versions of the Multi-MCS in Section 2.6 we will also focus on a class of $m$-Multi-MCS problems that have a fixed number of input graphs $k$. When we fix the value of $k$ for a class of $m$-Multi-MCS problems, we are only concerned with those $m$-Multi-MCS that have $k$ input graphs. The original MCS problem is the $m$-Multi-MCS problem with $k$ fixed as the value $k = 2$.

### 2.4 Background Problem Definitions

We will show in Section 2.5 that parameterized versions of the Longest Common Subsequence (LCS) problem are linearly FPT reducible to the parameterized versions of the Multi-MCS problem for directed graphs with vertex labels. The LCS problem can be defined formally as follows.

**Definition 2.4.1** For a given set of strings $Y = \{X_1, X_2, \ldots, X_k\}$, what is the maximum length of a sequence $X_{Max}$ that is a subsequence of every $X_i$ for $1 \leq i \leq k$?
For each parameterized version of the Multi-MCS problem defined in Definitions 2.3.1 - 2.3.4 there is a corresponding parameterized version of the LCS problem defined in [15, 32]. These parameterized LCS problems are defined as follows.

**Definition 2.4.2** ($m$-LCS) - For a given set of strings $Y$ and an integer $m$, is there a sequence $X_{Max}$ of length at least $m$ that is a subsequence of every string in $Y$?

**Definition 2.4.3** ($k$-LCS) - For a given set of strings $Y$ and an integer $k$, if $|Y| = k$, what is the maximum length of a sequence $X_{Max}$ that is a subsequence of every string in $Y$?

**Definition 2.4.4** ($km$-LCS) - For a given set of strings $Y$, an integer $k$, and an integer $m$, if $|Y| = k$, is there a sequence $X_{Max}$ of length at least $m$ that is a subsequence of every string in $Y$?

**Definition 2.4.5** ($k\Gamma$-LCS) - For a given set of strings $Y$, an integer $k$, and an alphabet $\Gamma$, if $|Y| = k$ and the strings of $Y$ are all in $\Gamma^*$, what is the maximum length of a sequence $Y_{Max}$ that is a subsequence of every string in $Y$?

Additional classes of parameterized LCS problems have been researched by others. For instance, the authors of [86] discuss the $k$-LCS problem where $|\Sigma|$ is fixed as a constant and give the parameterized complexity of this problem to be $W[1]$-hard. We do not focus on these additional parameterizations of LCS in this thesis, but our reduction of the LCS problem to the Multi-MCS problem may have applications to these other parameterized versions of LCS.

In Section 2.6 we will show linear FPT reductions of parameterized versions of the maximum clique (CLIQUE) problem to some of the parameterized versions of the Multi-MCS problem for unlabeled undirected graphs. The CLIQUE problem can be defined formally as follows.

**Definition 2.4.6** For a given graph $G$, what is the maximum size of a clique in $G$?

The size of a clique is defined by the number of vertices in it. If we view the size of the maximum clique, $m$, as a parameter, we call the parameterized problem $m$-CLIQUE. The $m$-CLIQUE problem is defined as follows.

**Definition 2.4.7** For a given graph $G$ and an integer $m$, is there a clique of size at least $m$ in $G$?

The reason that we are interested in the parameterized versions of the LCS and CLIQUE problem is because their parameterized complexity is known. The complexity of each parameterized version of the LCS and CLIQUE problem is shown in Table 2.1. We will obtain parameterized complexity results for the parameterized versions of the Multi-MCS problem when we linearly FPT reduce the parameterized versions of the LCS and CLIQUE problems to them.
Table 2.1: Parameterized complexity results of the parameterized versions of the LCS and CLIQUE problem[13, 15, 14]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>m-LCS</td>
<td>$W[2]$-hard</td>
</tr>
<tr>
<td>$k$-LCS</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
<tr>
<td>$km$-LCS</td>
<td>$W[1]$-Complete</td>
</tr>
<tr>
<td>$k\Gamma$-LCS</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
<tr>
<td>$m$-CLIQUE</td>
<td>$W[1]$-Complete</td>
</tr>
</tbody>
</table>

Figure 2.1: An example of the reduction of an instance of the LCS problem to a Multi-MCS problem. The letters in the vertices of the graphs are the labels of the vertices. One can see how the maximum common subgraph of the three graphs maps to the longest common subsequence of the three strings.

2.5 Parameterized complexity of Multi-MCS for labeled directed and undirected graphs via parameterized reduction from LCS

In this section we will show linear FPT-reductions of the parameterized versions of the LCS problem in Section 4 to parameterized versions of the Multi-MCS problem for directed graphs with vertex labels. First we will show a reduction of the LCS problem to the Multi-MCS problem for directed graphs with vertex labels. Then we will show how each parameter in the parameterized versions of the Multi-MCS problem can be found as a function of the parameters in the corresponding parameterized versions of the LCS problem. This will suffice to show that the parameterized versions of the LCS problem are linear FPT reducible to the parameterized versions of the Multi-MCS problem.

Lemma 2.5.1 For every instance of the LCS problem, there is an instance of the Multi-MCS...
problem for directed graphs with vertex labels that has a maximum common subgraph of size $m$ if and only if the instance of the LCS has a longest common subsequence of length $m$.

Proof. For a given instance of the LCS problem, $(Y = \{X_1, X_2, \ldots, X_k\})$, we construct an instance of the Multi-MCS problem, $(H = \{G_1, G_2, \ldots, G_k\})$ as follows. For each string $X_i = x_{i1}x_{i2} \ldots x_{i|X_i|}$ in the instance of the LCS problem, the graph $G_i = (V_i, E_i)$ is constructed as follows:

1. $|V_i| = |X_i|$
2. For every $i$ and $j$ such that $1 \leq i \leq k$ and $1 \leq j \leq |V_i|$, the label of a vertex $v_{ij} \in V_i$ is equal to the $j$th character $x_{ij}$ in the string $X_i$
3. For every pair of vertices $v_{ij}, v_{ij'} \in V_i$, an edge $(v_{ij}, v_{ij'}) \in E_i$ if and only if $j < j'$

Assume that the string $Z = z_1z_2\ldots z_m$ is the longest string that is a subsequence to each string $X_i$ in the set of strings $Y$. Then based on the properties of the graphs constructed by above reduction, each of the graphs, $G_i$ in the set of graphs $H$ will have an induced subgraph isomorphic to the graph $G_{Max} = (V_{Max}, E_{Max})$, where $V_{Max} = \{v_1, v_2, \ldots, v_m\}$, $E_{Max} = \{(v_i, v_j) \mid i < j\}$, and the label of $v_i$ is $z_i$. Therefore, the size of the maximum common subgraph of the set $H$ must be at least as large as the size of the longest common subsequence of $Y$.

Now assume that each graph $G_i$ in the set of graphs $H$ has an induced subgraph isomorphic to the graph $G_{Max} = (V_{Max}, E_{Max})$, where $V_{Max} = \{v_1, v_2, \ldots, v_m\}$, $E_{Max} = \{(v_i, v_j) \mid i < j\}$, the label of $v_i$ is $z_i$, and there is no graph $G' = (V', E')$ where $|V'| > |V_{Max}|$, and $G'$ is isomorphic to an induced subgraph of every graph $G_i$. Then based on the properties of the graphs constructed by above reduction, the sequence formed by the labels $z_1z_2\ldots z_m$ must be a subsequence of each string, $X_i$ in $Y$. Therefore, the size of the maximum common subgraph of the set $H$ can only be at most the size of the longest common subsequence of $Y$. These two proofs insure that the set of graphs $H$ has a maximum common subgraph of size $m$ if and only if the set of strings $Y$ has a longest common subsequence of length $m$. ■

In order to ensure that the parameterized reductions of the parameterized versions of the LCS problem to the parameterized versions of the Multi-MCS problem are linear FPT reductions, we must show that above reduction runs $O(f(s)n^{O(1)})$ time. Since above reduction does not require any parameters as input, it suffices to show that the algorithm runs in $O(n^{O(1)})$ time.

Lemma 2.5.2 The reduction of an instance of the LCS problem to an instance of the Multi-MCS problem takes $O(n^{O(1)})$ time.
Proof. In above reduction, we define the size of the LCS instance to be \( n = |Y| = \sum_{i=1}^{k} |X_i| \). Then the number of vertices in all constructed graphs is also \( n \). So the number of all edges in all graphs is at most \( O(n^2) \). Therefore, the reduction of an instance of the LCS problem to an instance of the Multi-MCS problem takes \( O(n^{O(1)}) \). 

Since the reduction of an instance of the LCS problem to an instance of the Multi-MCS problem can be done in \( O(n^{O(1)}) \), we can build a linear FPT reduction of parameterized versions of the LCS problem to parameterized versions of the Multi-MCS problem, if we can define the parameters of the Multi-MCS problem as functions of the parameters of the LCS problem.

**Theorem 2.5.3** There exists a linear FPT reduction of the \( m \)-LCS problem to the \( m \)-Multi-MCS problem; the \( k \)-LCS problem to the \( k \)-Multi-MCS problem; the \( km \)-LCS problem to the \( km \)-Multi-MCS problem; and the \( k\Gamma \)-LCS problem to the \( k\Gamma \)-Multi-MCS problem for directed graphs with vertex labels.

Proof. If \( \phi \)-LCS is one of the parameterized LCS problems given in Definitions 2.4.2 - 2.4.5, where \( \phi \in \{k, m, km, k\Gamma\} \), then \( \phi \)-LCS is linearly FPT reducible to \( \phi \)-Multi-MCS by the following linear FPT reduction. We use the algorithm given in the proof of Theorem 2.5.1 to construct an equivalent instance of the Multi-MCS problem from the instance of \( \phi \)-LCS. Then we set the values of the parameters in \( \phi \)-Multi-MCS to be equal to the values of the parameters in \( \phi \)-LCS. 

These linear FPT reductions of the parameterized versions of the LCS problem to parameterized versions of the Multi-MCS problem insure that the parameterized versions of the Multi-MCS problem are at least as hard as the parameterized versions of the LCS problem. Table 2.2 shows the hardness of each of these parameterized versions of the Multi-MCS problem for directed graphs with vertex labels.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )-Multi-MCS</td>
<td>( W[2] )-hard</td>
</tr>
<tr>
<td>( k )-Multi-MCS</td>
<td>( W[t] )-hard, ( \forall t \geq 1 )</td>
</tr>
<tr>
<td>( km )-Multi-MCS</td>
<td>( W[1] )-hard</td>
</tr>
<tr>
<td>( k\Gamma )-Multi-MCS</td>
<td>( W[t] )-hard, ( \forall t \geq 1 )</td>
</tr>
</tbody>
</table>

Table 2.2: Parameterized complexity results of the parameterized versions of the Multi-MCS problem for directed graphs with vertex labels.
Theorem 2.5.4 The complexity results in Table 2.2 also apply to the parameterized versions of the Multi-MCS problem for undirected graphs with both vertex labels and edge labels.

Proof. Since the direction of every edge in directed graph can be viewed as a label of the edge, the directed graph with vertex labels in the Theorem 2.5.3 can be viewed as an undirected graph with both vertex labels and edge labels. To do so, modify the step 3 of Lemma 2.5.1 to add an undirected edge whose label is “\(l(v_{ij}) \rightarrow l(v_{ij}')\)” where \(l(v_{ij})\) is the vertex label assigned to vertex \(v_{ij}\) in the step 3 of Lemma 2.5.1. The edge label acts as a direction on the edge between \(v_{ij}'\) and \(v_{ij}''\). Thus, the proof given for Theorem 2.5.3 holds for a reduction of the LCS problem to the Multi-MCS problem for undirected graphs with both vertex labels and edge labels. ■

2.6 Parameterized complexity of Multi-MCS for unlabeled undirected graphs via parameterized reduction with CLIQUE

In this section, we show that the \(m\)-Multi-MCS and \(km\)-Multi-MCS problems for unlabeled undirected graphs are \(W[1]\)-hard by giving a linear FPT reduction of the \(m\)-CLIQUE problem to them. we also show that when the value \(k\) is fixed, the \(m\)-Multi-MCS problem belongs to the class \(W[1]\) by showing a reduction of the problem to the \(m\)-CLIQUE problem. An algorithm for solving a problem similar to Multi-MCS for degree bounded unlabeled undirected graphs is been discussed in [11]. Here we consider the parameterized complexity of algorithms for the Multi-MCS problem on general unlabeled undirected graphs. This work is a generalization of the work in [52] for the MCS problem.

Lemma 2.6.1 For every instance of the CLIQUE problem, there is an instance of the Multi-MCS problem that has a maximum common subgraph of size \(m\) if and only if the instance of the CLIQUE problem has a maximum clique of size \(m\).

Proof. For every instance of the CLIQUE problem, \((G)\), we create an instance of the Multi-MCS problem by defining the set of graphs \(H\) as follows. If the number of graphs, \(k\), is not fixed for the Multi-MCS problem I set \(H\) equal to the set \(\{G_1, G_2\}\), where \(G_1 = G\) and \(G_2 = K_{|V(G)|}\). The graph \(K_{|V(G)|}\) is a complete graph with \(|V(G)|\) vertices. If \(k\) is fixed as a constant \(c > 2\) for the Multi-MCS problem, then we set \(H\) equal to the set \(\{G_1, G_2, \ldots, G_k\}\) where \(G_1 = G\) and \(G_i = K_{|V(G)|}\) for all \(i, 2 \leq i \leq k\).

Since \(K_{|V(G)|}\) is a complete graph, any induced subgraph of \(K_{|V(G)|}\) will also be a complete graph. If \(H\) has a maximum common induced subgraph of size \(m\), then that subgraph must be
a complete graph. Thus, $G$ must have a clique of size $m$ and the size of the maximum clique in $G$ must be at least $m$.

Every clique of size less than or equal to $|V(G)|$ is a subgraph of $K_{|V(G)|}$. If $G$ has a maximum clique of size $m$, then the set $H$ must have a common subgraph of size $m$. Thus, the size of the maximum common subgraph of $H$ must be at least $m$. These two proofs insure that the set of graphs $H$ has a maximum common subgraph of size $m$ if and only if the graph $G$ has maximum clique of size $m$. □

The time needed to construct an instance of the Multi-MCS problem according to the rules described in the proof of Lemma 2.6.1 is the time needed to construct the copies of $K_{|V(G)|}$. If we consider the size of the instance of the CLIQUE problem to be $n = |G| = |V(G)|$, then the time needed to construct a copy of $K_{|V(G)|}$ is in $O(n^2)$ because there will be $O(|V(G)|^2)$ edges in $K_{|V(G)|}$.

When $k$ is not fixed for the Multi-MCS problem, only one copy of $K_{|V(G)|}$ is needed. Thus, for an instance of the CLIQUE problem, the time complexity of the construction of an equivalent instance of the Multi-MCS problem when $k$ is not fixed is $O(n^2)$. When $k$ is fixed as a constant $c > 2$, $c - 1$ copies of $K_{|V(G)|}$ are needed. However, since $c$ is a constant, the time complexity of the construction of the instance of the Multi-MCS problem remains in $O(n^2)$ time when $k$ is fixed.

Theorem 2.6.2 The $m$-CLIQUE problem is linear FPT reducible to both the $m$-Multi-MCS problem and the $km$-Multi-MCS problem.

Proof. We construct a linear FPT reduction of the $m$-CLIQUE problem to the $m$-Multi-MCS and $km$-Multi-MCS problem as follows. We use the algorithm given in the proof of Lemma 2.6.1 to construct the new instance of the Multi-MCS problem. For the $m$-Multi-MCS problem we determine the value of the new parameter $m$ to be equal to the value of the parameter $m$ in the CLIQUE instance. For the $km$-Multi-MCS problem we determine the value of the new parameter $k$ to be equal to 2 and the new value of the parameter $m$ to be equal to the value of the parameter $m$ in the CLIQUE instance. □

The existence of a linear FPT reduction of the $m$-CLIQUE problem to both the $m$-Multi-MCS problem and the $km$-Multi-MCS problem for unlabeled undirected graphs means that these problems are $W[1]$-hard.

We would like to show that the parameterized versions of the Multi-MCS problem are linear FPT reducible to the parameterized CLIQUE problem. This would prove that these parameterized problems are in complexity class $W[1]$.

Lemma 2.6.3 For every instance of the Multi-MCS problem, there is an instance of the CLIQUE problem that has a maximum clique of size $m$ if and only if the instance of the Multi-MCS problem has a maximum common subgraph of size $m$. 

18
Proof. For every instance of the Multi-MCS problem, \( H = \{G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k)\} \), I can construct an instance of the CLIQUE problem for graph \( G \) as follows. The graph \( G \) constructed has the following properties:

1. \( V(G) = V_1 \times V_2 \times \ldots \times V_k \)
2. An edge \( (\{u_1, u_2, \ldots, u_k\}, \{v_1, v_2, \ldots, v_k\}) \) is a member of \( E(G) \) if and only if the following apply
   
   (a) For all \( i \) such that \( 1 \leq i \leq k \), \( u_i \neq v_i \)
   
   (b) One of the following two apply
      
      i. For all \( i \) such that \( 1 \leq i \leq k \), \( (u_i, v_i) \in E_i \)
   
   ii. For all \( i \) such that \( 1 \leq i \leq k \), \( (u_i, v_i) \notin E_i \)

Assume that a clique \( C = \{c_1, c_2, \ldots, c_m\} \) of size \( m \) exists in \( G \) such that there is no clique \( C' \) in \( G \) where \( |C'| > |C| \). Because \( c_j \) where \( 1 \leq j \leq m \) is a vertex in the constructed graph \( G \), \( c_j \) can also be defined as the set \( c_j = \{v_{1j}, v_{2j}, \ldots, v_{kj}\} \). The vertex \( v_{ij} \) is a vertex in the graph \( G_i \) of the set of graphs \( H \) in the given instance of Multi-MCS problem. Construct the set \( S_i \) such that \( S_i = \{v_{i1}, v_{i2}, \ldots, v_{im}\} \).

Based on the properties of the graph \( G \) constructed by reduction above I know the following. For two integers \( j' \) and \( j'' \) such that \( 1 \leq j' \leq m \) and \( 1 \leq j'' \leq m \), if an edge \( (v_{ij'}, v_{ij''}) \) exists in any graph \( G_i \) for \( i \) such that \( 1 \leq i \leq k \), then the edge \( (v_{ij'}, v_{ij''}) \) must exist in all graphs \( G_i \) for \( i \) such that \( 1 \leq i \leq k \). Conversely, if an edge \( (v_{ij'}, v_{ij''}) \) does not exist in all graphs \( G_i \) for \( i \) such that \( 1 \leq i \leq k \), then the edge \( (v_{ij'}, v_{ij''}) \) must not exist in any graph \( G_i \) for \( i \) such that \( 1 \leq i \leq k \). This insures that the subgraphs induced in the graph \( G_i \) by the vertices in the set \( S_i \) will be isomorphic to one another. Thus, the size of the maximum common subgraph of the instance of the Multi-MCS problem must be at least as large as the size of the maximum clique in the instance of CLIQUE constructed by above reduction.

Now assume that each graph \( G_i \) in the set of graphs \( H \) has an induced subgraph, \( I_i \), isomorphic to the graph \( G_{Max} = (V_{Max}, E_{Max}) \) such that there is no graph \( G' = (V', E') \) where \( |V'| > |V_{Max}| \) and \( G' \) is isomorphic to an induced subgraph of every graph \( G_i \). Because each induced subgraph \( I_i \) is isomorphic to the graph \( G_{Max} \) we know that there is a one-to-one mapping \( g_i \) between the vertices in \( V_{Max} \) and \( V(I_i) \). This insures that if the edge \( (u, v) \in E_{Max} \), then the edge \( (g_i(u), g_i(v)) \) is in the graph \( G_i \) for all \( i \) such that \( 1 \leq i \leq k \). It also insures that if the edge \( (u, v) \notin E_{Max} \), then the edge \( (g_i(u), g_i(v)) \) is not in the graph \( G_i \) for all \( i \) such that \( 1 \leq i \leq k \).

Based on the properties of the graph \( G \) constructed by above reduction I know that for every pair of vertices \( u, v \in E_{Max} \), the edge \( \{g_1(u), g_2(u), \ldots, g_k(u)\}, \{g_1(u), g_2(u), \ldots, g_k(u)\} \)
is in the graph $G$. Thus, the size of the maximum clique of the constructed instance of the \textsc{Clique} problem must be at least as large as the size of the maximum common subgraph of the instance of the Multi-MCS problem. These two proofs insure that the instance of the Multi-MCS problem has a maximum common subgraph of size $m$ if and only if the constructed instance of \textsc{Clique} has a maximum clique of size $m$. ■

Obviously, above reduction is more complex than the other reduction algorithms that we have shown in this thesis. Since above reduction does not require any parameters as input, we need the algorithm to run in $O(n^{O(1)})$ time in order for us to be able to use it as part of a linear FPT reduction. We will show that this is only possible if certain parameters of the Multi-MCS problem are fixed as constants.

**Lemma 2.6.4** The reduction of an instance of the LCS problem to an instance of the Multi-MCS problem takes $O(n^k)$ time.

**Proof.** We define the size of the Multi-MCS instance to be $n = |H| = \sum_{i=1}^{k} |G_i|$. Also, we assume that $|V_i|$ is the same for all $i$, where $1 \leq i \leq k$. This can be accomplished by padding the sets $V_i$ whose sizes are smaller. Because $|V_i|$ is the same for all $i$, where $1 \leq i \leq k$, I set $|V_i| = |V_1|$. With these assumptions $n \in O(|V_1|^2)$. In above reduction, $G$ has at most $|V_1|^k$ nodes. So it has at most $O(|V_1|^{2k})$ edges. Since $n \in O(|V_1|^2)$, the runtime of the reduction is $O(n^k)$. ■

Since the runtime of above reduction is in $O(n^k)$ the runtime cannot be in $O(n^{O(1)})$ unless $k$ is a fixed constant.

**Theorem 2.6.5** The $m$-Multi-MCS problem with a fixed value of $k$ is linear FPT reducible to the $m$-\textsc{Clique}

**Proof.** We construct a linear FPT reduction of the $m$-Multi-MCS problem with a fixed value of $k$ to the $m$-\textsc{Clique} as follows. We use the algorithm given in the proof of Lemma 2.6.3 to construct the new instance of the \textsc{Clique} problem. For the $m$-\textsc{Clique} problem we determine the value of the new parameter $m$ to be equal to the value of the parameter $m$ in the Multi-MCS instance. This reduction is linear FPT because it runs in $O(n^k)$ time. When the value of $k$ is fixed as a constant $c > 2$, $O(n^k) \in O(n^{O(1)})$. If $k$ is either a parameter or not fixed, $O(n^k) \notin O(n^{O(1)})$. Thus, the reduction of Lemma 2.6.1 is not FPT for either the $km$-Multi-MCS problem or the $m$-Multi-MCS problem where $k$ is variable. ■

The existence of a linear FPT reduction for the $m$-Multi-MCS problem where the number of input graphs $k$ is fixed for unlabeled undirected graphs to the $W[1]$-complete problem $m$-\textsc{Clique} means that the $m$-Multi-MCS problem is in the complexity class $W[1]$. Since the
The $m$-Multi-MCS problem for undirected unlabeled graphs with a fixed $k$ is known to be $W[1]$-hard by Theorem 2.6.2, the problem is $W[1]$-complete. The complexity results for all of the parameterized versions of Multi-MCS discussed in this section are presented in Table 2.3.

Table 2.3: Parameterized complexity of parameterized versions of the Multi-MCS problem for unlabeled undirected graphs obtained via linear FPT reductions to and from the $m$-CLIQUE problem

<table>
<thead>
<tr>
<th>Problem</th>
<th>$k$</th>
<th>Hardness</th>
<th>Complexity Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$-Multi-MCS</td>
<td>Not Fixed</td>
<td>$W[1]$-hard</td>
<td>Unknown</td>
</tr>
<tr>
<td>$km$-Multi-MCS</td>
<td>Parameter</td>
<td>$W[1]$-hard</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

### 2.7 Discussion

We have shown that there exists linear FPT reductions of parameterized versions of the LCS problem to parameterized versions of the Multi-MCS problem for directed graphs with vertex labels and shown its corollary for undirected graphs with vertex and edge labels. We have also shown a linear FPT reduction of the parameterized CLIQUE problem to and from parameterized versions of the Multi-MCS problem for unlabeled undirected graphs. These reductions prove the complexity results given in Table 2.4.

Table 2.4: Parameterized complexity results of the parameterized versions of the Multi-MCS problem for unlabeled undirected graphs

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Directed Graphs</td>
</tr>
<tr>
<td></td>
<td>Vertex Labeled</td>
</tr>
<tr>
<td>$k$-Multi-MCS</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
<tr>
<td>$k\Gamma$-Multi-MCS</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
</tbody>
</table>
There are still open problems about the parameterized complexity of the Multi-MCS. We would like to find linear FPT reductions of the parameterized versions of the Multi-MCS problem to other parameterized problems to prove membership in a $W[t]$ complexity class. We would like to extend the parameterized complexity results of labeled graphs to unlabeled graphs. There are also questions about whether these complexity results hold when the class of graphs is restricted to trees. The theoretical results proven in this paper will also be useful as we try to develop algorithms for solving the Multi-MCS problem and to apply these algorithms to biological problems.
Chapter 3

On the Parameterized Complexity of the Multi-MCT and Multi-MCST Problems

3.1 Introduction

Tree structures are used to represent many different types of data. A tree is called an order tree if there is an order among the children of all nodes. Ordered, vertex-labeled trees have been used to represent XML data [50] and web access logs [113], while unordered, vertex-labeled trees have been used to represent phylogenetic trees [35] and carbohydrate sugar chains [5]. Problems concerning such data can then be reduced to graph-theoretical, tree-based problems.

Two problems commonly used in the comparison of trees are the problem of finding the maximum common subtree (MCT) and the problem of finding the minimum common supertree (MCST) for two trees. A subtree in this paper refers to any vertex-induced subgraph of a tree that forms a tree. A supertree is any tree for which the original tree is a subtree. The MCT is the largest tree (by measure of the number of vertices in the tree) that is isomorphic to a subtree of both trees. The MCST is the smallest tree that is isomorphic to a supertree of both trees. The two problems MCT and MCST are closely related, since the maximum common subtree can be used to determine the minimum common supertree [90]. For both ordered and unordered, vertex-labeled trees, if a only a pair of trees is considered, polynomial-time algorithms are known to exist for the MCT problem and, by the reduction in [90], the MCST problem when either ordered or unordered vertex-labeled trees are input [1, 51].

There is also a need to find the MCT and MCST of more than just two trees. For example, multiple database schemas can be represented as XML data and therefore also as ordered, labeled trees. The MCT of these trees would represent the maximum intersection of the multiple
schemata. This is something of interest if one is trying to determine which queries can be run across multiple diverse database schemata. The MCST of the ordered, labeled trees would represent the minimum union of the multiple schemata. This MCST could be used to join the multiple database into a single database. The problem of finding the MCT and MCST for multiple trees is called the Multi-MCT and Multi-MCST problems, respectively.

Finding solutions to the Multi-MCT problem and the Multi-MCST problem has shown to be significantly more difficult than finding solutions to the MCT and MCST problems. Akutsu proved in [1] that the Multi-MCT problem for unordered, vertex-labeled graphs was NP-hard. It is also currently unknown if a solution to the Multi-MCT problem can generally be used to generate a solution to the Multi-MCST problem.

In [32] Downey and Fellows introduce parameterized complexity classes that define which problems are likely to have efficient solutions when certain parameters are fixed. In this chapter, we study the hardness of the parameterized Multi-MCT and Multi-MCST with respect to these parameterized complexity classes.

### 3.2 Definitions of Problems

In this section, we will introduce both the Multi-MCT and Multi-MCST problems, discuss their various parameterizations.

**Definition 3.2.1 (Multi-MCT)** For a given set of $k$ trees $R = \{T_1, T_2, \ldots, T_k\}$ with vertex-labels $l(v) \in \Gamma$ and an integer $m$, does there exist a tree $T_{\text{max}}$ of size greater than or equal to $m$ that is isomorphic to a subtree of every tree $T_i$ for $1 \leq i \leq k$?

We define the size of a tree as the number of vertices in the tree. The related problem of finding the minimum common supertree of multiple trees (Multi-MCST) is defined similarly and is given as follows.

**Definition 3.2.2 (Multi-MCST)** For a given set of $k$ trees $R = \{T_1, T_2, \ldots, T_k\}$ with vertex-labels $l(v) \in \Gamma$ and an integer $m$, does there exist a tree $T_{\text{min}}$ of size less than or equal to $m$ for which every tree $T_i$ for $1 \leq i \leq k$ is isomorphic to a subtree of $T_{\text{min}}$?

The following problems are defined here because parameterized versions of them will be shown to be reducible to a parameterized version of either the Multi-MCT problem or the Multi-MCST problem. LCS and CLIQUE problems are formally defined in Chapter 2. In the following, we give the definition of SCS [32].

**Definition 3.2.3 (SCS)** For a given finite alphabet $\Gamma$, a finite set $Y$ of $k$ strings $\{X_1, X_2, \ldots, X_k\}$ from $\Gamma^*$, and a positive integer $m$, is there a string $X_{\text{min}} \in \Gamma^*$ with $|X_{\text{min}}| \leq m$ such that each string $X_i \in Y$ is a subsequence of $X_{\text{min}}$?
Choosing different parameters for each of these problems leads to various parameterized problems. These parameterized problems and their corresponding parameters are shown in Table 3.1.

Table 3.1: The different parameterized versions of the Multi-MCT, Multi-MCST, LCS, CLIQUE, SCS problems and their associated parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Multi-MCT</th>
<th>LCS</th>
<th>CLIQUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$k$-Multi-MCT</td>
<td>$k$-LCS</td>
<td>N/A</td>
</tr>
<tr>
<td>$m$</td>
<td>$m$-Multi-MCT</td>
<td>$m$-LCS</td>
<td>$m$-CLIQUE</td>
</tr>
<tr>
<td>$km$</td>
<td>$km$-Multi-MCT</td>
<td>$km$-LCS</td>
<td>N/A</td>
</tr>
<tr>
<td>$k, \Gamma$</td>
<td>$k\Gamma$-Multi-MCT</td>
<td>$k\Gamma$-LCS</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The known parameterized complexity and hardness results for the parameterized versions of the LCS, CLIQUE, and SCS problems are given in Table 3.2.

Table 3.2: Parameterized complexity results of the parameterized versions of the LCS, SCS and CLIQUE problem

<table>
<thead>
<tr>
<th>Problem</th>
<th>Complexity</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-LCS</td>
<td>$W[1]$-hard, $\forall t \geq 1$</td>
<td>[15]</td>
</tr>
<tr>
<td>$k\Gamma$-LCS</td>
<td>$W[1]$-hard, $\forall t \geq 1$</td>
<td>[14]</td>
</tr>
<tr>
<td>$m$-SCS</td>
<td>$W[1]$-Complete</td>
<td>[31]</td>
</tr>
<tr>
<td>$k$-SCS</td>
<td>$W[1]$-hard, $\forall t \geq 1$</td>
<td>[53]</td>
</tr>
<tr>
<td>$km$-SCS</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>$k\Gamma$-SCS</td>
<td>$W[1]$-hard, $\forall t \geq 1$</td>
<td>[53]</td>
</tr>
</tbody>
</table>
3.3 Parameterized Complexity of the Multi-MCST Problem for Vertex-Labeled, Ordered Trees

In this section, we show that for vertex-labeled, ordered trees, the Multi-MCST problem are hard for the different parameterized complexity classes. In order to get these parameterized hardness results, we will show that the shortest common supersequence (SCS) problem is reducible to the Multi-MCST problem for vertex-labeled, ordered trees.

Lemma 3.3.1 There is a linear FPT reduction from SCS to Multi-MCST for labeled ordered trees.

Proof. Given an instance of SCS, \((X_1, \ldots, X_k, m)\), we construct an instance of Multi-MCST as follows. Assume that \(X_i = A_{i,1} \ldots A_{i,r_i}\), where \(r_i\) is the length of \(X_i\). For each \(X_i\), we construct a labeled order trees \(T_i\): it has one root node \(v_{i,0}\) with label \(\text{ROOT}\), and it has \(r_i\) ordered children, \(v_{i,1}, \ldots, v_{i,r_i}\), with labels, \(A_{i,1}, \ldots, A_{i,r_i}\), respectively.

Thus, if a sequence \(X = A_1 \ldots A_r\) is the supersequence of \(X_i\), then the tree with a root node labeled \(\text{ROOT}\) and \(r\) ordered children labeled \(A_1, \ldots, A_r\) is a supertree of \(T_i\).

Conversely, assume that the tree \(S\) is a minimum common supertree of the trees in the set constructed previously, \(R = \{T_1, \ldots, T_k\}\). Then \(S\) will be structured as a star tree whose hub is labeled \(\text{ROOT}\). Because of the structure of the trees in \(R\), each vertex in \(S\) must be either have the label \(\text{ROOT}\) or be connected to a vertex with the label \(\text{ROOT}\). Now assume that there are two vertices, \(v_1\) and \(v_2\), in \(S\) that are labeled \(\text{ROOT}\). We can construct a tree by removing \(v_2\) and connecting all of the children of \(v_2\) to \(v_1\) that will be a common supertree of \(R\) but with \(|S| - 1\) vertices. This would contradict the definition of \(S\). Therefore, \(S\) must be a star tree in which the hub and only the hub is labeled \(\text{ROOT}\). Thus, the ordered supertrees \(S\) must consists of a root node \(\text{ROOT}\) and \(r\) children nodes \(A_1, \ldots, A_r\). Hence \(S\) correspond to a supersequence \(X = A_1 \ldots A_r\) of \(X_i\).

Hence, the minimum common supersequence of \(X_1, \ldots, X_k\) is of size \(r\) if and only if the minimum common supertree of \(T_1, \ldots, T_k\) of size \(r + 1\).

The proof that the reduction is FPT is left to the reader. ■

Thus, by the known parameterized complexity results of the SCS problem, we get the parameterized hardness results of the Multi-MCST problem. This is presented in Table 3.3.

3.4 Parameterized Complexity of the Multi-MCT Problem for Vertex-Labeled, Ordered Trees

In this section, we show that for vertex-labeled, ordered trees, the different parameterized version of the Multi-MCT problem are hard. In order to get these parameterized hardness
Table 3.3: Parameterized complexity results of the parameterized versions of the Multi-MCST problem for vertex-labeled, ordered trees

<table>
<thead>
<tr>
<th>Problem</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$-Multi-MCST</td>
<td>N/A</td>
</tr>
<tr>
<td>$k$-Multi-MCST</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
<tr>
<td>$km$-Multi-MCST</td>
<td>N/A</td>
</tr>
<tr>
<td>$k\Gamma$-Multi-MCST</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
</tbody>
</table>

results, we will show that the longest common subsequence (LCS) problem is reducible to the Multi-MCT problem for vertex-labeled, ordered trees.

**Lemma 3.4.1** There is a linear FPT reduction from LCS to Multi-MCT for labeled order trees.

**Proof.** Given an instance of LCS, $(X_1, \ldots, X_k, m)$, we construct an instance of Multi-MCT as follows. Assume that $X_i = A_{i,1} \cdots A_{i,r_i}$ where $r_i$ is the length of $X_i$. For each $X_i$, we construct a labeled order trees $T_i$: it has one root node $v_{i,0}$ with label $\text{ROOT}$, and it has $r_i$ ordered children, $v_{i,1}, \ldots, v_{i,r_i}$ with labels, $A_{i,1}, \ldots, A_{i,r_i}$, respectively.

Thus, if a sequence $X = A_1 \cdots A_r$ is the subsequence of $X_i$, then there is an induced ordered subtree by the vertices labeled $A_1, \ldots, A_r$ and the root node labeled $\text{ROOT}$ in $T_i$. Conversely, an ordered subtree induced by the vertices labeled $A_1, \ldots, A_r$ and the root node $\text{ROOT}$ in $T_i$ correspond to the subsequence $A_1 \cdots A_r$ of $X_i$.

Hence, a subsequence $X = A_1 \cdots A_r$ is the common subsequence of $X_1, \ldots, X_k$ if and only if there is an order subtree consisting of a root node labeled $\text{ROOT}$ and children vertices labeled $A_1, \cdots, A_r$ is a common ordered subtree of $T_1, \ldots, T_k$.

The proof that the reduction is FPT is left to the reader. □

Thus, by the known parameterized complexity results of the LCS problem, we get the parameterized hardness results of the Multi-MCT problem. This is presented in Table 3.4.

### 3.5 Parameterized Complexity of the Multi-MCT Problem for Vertex-Labeled, Unordered Trees

In this section, we show that for vertex-labeled, unordered trees, the parameterized version of the Multi-MCT problem named $m$-Multi-MCT is $W[1]$-hard. In order to prove this, we will show that the maximum clique (CLIQUE) problem is reducible to the Multi-MCT problem for vertex-labeled, unordered trees. The reduction is motivated by the L-reduction in [2], which
Table 3.4: Parameterized hardness results of the parameterized versions of the Multi-MCT problem for vertex-labeled, ordered trees

<table>
<thead>
<tr>
<th>Problem</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>m-Multi-MCT</td>
<td>$W[2]$-hard</td>
</tr>
<tr>
<td>k-Multi-MCT</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
<tr>
<td>km-Multi-MCT</td>
<td>$W[1]$-hard</td>
</tr>
<tr>
<td>kT-Multi-MCT</td>
<td>$W[t]$-hard, $\forall t \geq 1$</td>
</tr>
</tbody>
</table>

reduces the maximum independent set problem to Multi-MCT problem in order to show the hardness of approximation algorithms. We give some modifications.

Lemma 3.5.1 There is a linear FPT reduction from CLIQUE to Multi-MCT for labeled unordered trees.

Proof. Given a graph $G = (V, E)$ with $V = (v_1, \ldots, v_n)$, we construct $n+1$ trees: $T_1, \ldots, T_{n+1}$. For any $i \leq n$, the $T_i$ have a root labeled by 0 and have two children nodes $c_i, d_i$ which are labeled by 0. For each node $c_i$, it has $n$ children nodes $c_{i,1}, \ldots, c_{i,n}$. Every node $c_{i,i}$ is labeled by $i$. For every not $c_{i,j}$ where ($i \neq j$), it is labeled by $j$ if $(v_i, v_j) \in E$, or it is labeled by $-1$ if $(v_i, v_j) \notin E$. Intuitively, the labels of vertices $c_{i,j}$ encode the adjacency list of graph vertex $v_i$.

For each node $d_i$, it has $n$ children nodes $d_{i,1}, \ldots, d_{i,n}$. For every node $d_{i,j}$, it is labeled by $j$ if $j \neq i$, or it is labeled by $-1$ if $j = i$.

The tree $T_{n+1}$ has a root node labeled by 0 and has $n$ children nodes $t_1, \ldots, t_n$. For each node $t_j$, it is labeled by $j$.

Assume that there is a clique $C = \{v_{j_1}, \ldots, v_{j_k}\}$. Then there is common subtree $T$ of size $k+1$, of the set $T_1, \ldots, T_{n+1}$ which consists of one root node labeled by 0 and $k$ children nodes which are labeled by $\{j_1, \ldots, j_k\}$.

On the other hand, suppose that $T$ is a common subtree of size $k+1$. Then $T$ must include the root of $T_{n+1}$ and some children nodes. These children nodes will have the labels $j_1, \ldots, j_k$. The vertices $v_{j_1}, \ldots, v_{j_k}$ describe exactly a clique in the graph $G$. Thus, given a common subtree of size $k+1$ we obtain a clique of size $k$.

The proof that the reduction is FPT is left to the reader.

Since $m$-CLIQUE is $W[1]$-hard, we get the following conclusion.

Theorem 3.5.2 For labeled unordered trees, $m$-Multi-MCT is $W[1]$-hard.
3.6 Conclusion

In this chapter, we have proven that various parameterized versions of the Multi-MCT and Multi-MCST problems are hard for various parameterized complexity classes. The overview of our results are given in Table 3.5. Since all of the problems considered here are at least $W[1]$-hard, it is unlikely that the parameterized versions of the Multi-MCT and Multi-MCST can be solved in $O(f(k) \cdot n^{O(1)})$ time.

Table 3.5: Parameterized complexity class hardness results for parameterized Multi-MCT and Multi-MCST problems

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Multi-MCT Ordered Trees</th>
<th>Multi-MCT Unordered Trees</th>
<th>Multi-MCST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$W[t]$-hard $\forall t \geq 1$</td>
<td>N/A</td>
<td>$W[t]$-hard $\forall t \geq 1$</td>
</tr>
<tr>
<td>$m$</td>
<td>$W[2]$-hard</td>
<td>$W[1]$-hard</td>
<td>N/A</td>
</tr>
<tr>
<td>$k, m$</td>
<td>$W[1]$-hard</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$k, \Gamma$</td>
<td>$W[t]$-hard $\forall t \geq 1$</td>
<td>N/A</td>
<td>$W[t]$-hard $\forall t \geq 1$</td>
</tr>
</tbody>
</table>
Chapter 4

The Greedy Algorithm for Maximum Weight Induced Subgraph Problem

4.1 Introduction

In 1999, Hartwell et al. introduced the concept of functional modules as a way of attributing a specific biological function to a set of interacting proteins as opposed to a single protein [55]. Early approaches to detecting these functional modules [33, 85, 97, 98] relied on the fact that functional modules tend to form statistically significant densely connected subnetworks in protein interaction networks [98]. Recently, this model has been augmented by the hypothesis that sets of evolutionarily conserved proteins with conserved patterns of interactions are likely to correspond to conserved functional modules [71]. This has led to the development of network alignment algorithms to find these evolutionarily conserved functional modules.

Network alignment algorithms such as PathBLAST [71], NetworkBLAST [93], NetworkBLAST-M [63], Graemlin [40, 39], IsoRank [96], and MaWISh [79] identify sets of proteins such that the interactions between the proteins in two different sets are likely to be conserved across multiple species. The sets of proteins found by the network algorithms are called protein alignments, while a network alignment refers to the set of all of the protein alignments.

There are a number of ways in which network alignment algorithms can differ. Some alignment algorithms are limited to aligning proteins from two species, while others can align more than two. Given a set of proteins from multiple species, alignment algorithms can either attempt to identify protein alignments for all of the proteins (global network aligners), or it can attempt to find protein alignments for only a subset of the proteins (local network aligners) [96]. To identify the candidate protein alignments, an alignment algorithm can either use a brute
Table 4.1: Comparison of some previous network alignment algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number</th>
<th>Alignment</th>
<th>Align. Generate</th>
<th>Use GAN?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Algorithm</td>
<td>2</td>
<td>Local</td>
<td>BLAST E-Values</td>
<td>Yes</td>
</tr>
<tr>
<td>MaWISh [79]</td>
<td>2</td>
<td>Local</td>
<td>BLAST E-Values</td>
<td>Yes</td>
</tr>
<tr>
<td>IsoRank [96]</td>
<td>2</td>
<td>Global</td>
<td>Brute Force</td>
<td>No</td>
</tr>
<tr>
<td>PathBLAST [71]</td>
<td>2</td>
<td>Local</td>
<td>BLAST E-Values</td>
<td>Yes</td>
</tr>
<tr>
<td>NetworkBLAST [93]</td>
<td>2-3</td>
<td>Local</td>
<td>BLAST E-Values</td>
<td>Yes</td>
</tr>
<tr>
<td>NetworkBLAST-M [63]</td>
<td>≥ 2</td>
<td>Local</td>
<td>BLAST E-Values</td>
<td>No</td>
</tr>
<tr>
<td>Graemlin [40]</td>
<td>≥ 2</td>
<td>Global &amp; Local</td>
<td>BLAST E-Values</td>
<td>No</td>
</tr>
<tr>
<td>Graemlin 2.0 [39]</td>
<td>≥ 2</td>
<td>Global</td>
<td>BLAST E-Values</td>
<td>No</td>
</tr>
</tbody>
</table>

force method and consider every possible set of proteins as a candidate protein alignment, or it can use tools such as BLAST [3] to limit candidate proteins alignments to those sets of proteins that are functionally similar or homologous. An alignment algorithm can also choose to either explicitly represent the degree of interaction conservation as a global alignment network (GAN) or not. A GAN is a network whose nodes are candidate protein alignments, and whose edges represent the degree of interaction conservation between the proteins in the candidate protein alignments. Table 4.1 gives a brief overview of the characteristics of some previous network alignment algorithms.

If a GAN is used by a network alignment algorithm, the algorithm identifies conserved functional modules through the identification of specific graph structures in the GAN. For instance, PathBLAST searches for paths in the network, and MaWISh searches for subgraphs with sufficiently large edge weight sums [71, 79]. The choice of the target graph structures determines the type of functional modules the alignment algorithm will find.

Since network alignment algorithms such as MaWISh tend to identify functional modules consisting of only a few proteins (Table 4.2), larger functional modules may be missed by these alignment algorithms. As the graph in Figure 4.1 shows, there are a large number of functional modules whose size may preclude algorithms such as MaWISh from identifying them. The graph shows the likelihood that a known functional module, in this case defined as generic KEGG pathway [64], contains at least the number of proteins given on the x-axis. Over 50% of the generic KEGG pathways contain at least 30 proteins. It remains a goal of network alignment algorithms to identify larger functional modules like these.
4.1.1 Contribution

In this Chapter, we propose a new formulation for the problem of finding significant subnetworks in a pairwise global alignment network. We also introduce an algorithm that heuristically attempts to solve this new optimization problem. We evaluate the performance of our algorithm using real-world protein interaction data. We find that our algorithm has a comparable precision and recall rates for identifying functional modules as the MaWISH algorithm. However, our algorithm has a greater chance of finding larger local alignment networks that correspond to large functional modules not identified by previous search methods.

4.2 Previous methods of finding high scoring local alignment networks

Koyutürk et al. developed a heuristic algorithm to find conserved functional protein modules by identifying locally maximal alignment networks in the global alignment network [79]. The motivation for the algorithm comes from the fact that a protein in a given functional module
interacts with all other proteins in the module either directly or through a module hub. If a functional module is conserved between two species, then edges connecting pairs of proteins from this conserved functional module will have a large, non-negative weight. This happens because the weight of the edge connecting the two pairs of proteins is positive when interaction is conserved between two pairs of proteins, and its magnitude is proportional to the combined similarity of the two pairs of proteins [79].

Finding subgraphs of the global alignment network that have a large sum of edge-weights is equivalent to solving the Maximum Weight Induced Subgraph Problem given in Definition 4.2.1.

**Definition 4.2.1** Given a graph $G(V, E)$ and a constant $\epsilon$, find a subset of nodes, $\tilde{V} \subseteq V$ such that the sum of the weights of the edges in the subgraph induced by $\tilde{V}$ is at least $\epsilon$, i.e., $W(\tilde{V}) = \sum_{v, v' \in \tilde{V}} w(vv') \geq \epsilon$.

The algorithm presented of Koyutürk et al. heuristically finds the subgraphs defined by Definition 4.2.1. The algorithm “grows” a locally maximal subgraph. The subgraph is seeded by starting with a vertex in the global alignment network that has a large number of non-negative weight edges. A large number of non-negative weight edges incident to a vertex in the global alignment network means that the aligned pair of proteins represented by the vertex has a large number of conserved interactions. This means that the vertex could likely be a module hub. Once the subgraph is seeded, the algorithm proceeds by iteratively adding to the subgraph the vertex that has the largest sum of edge-weights of the edges connecting it to the subgraph. This process stops once there are no vertices whose sum of edge-weights of the edges connecting it to the subgraph is greater than zero. This stopping criteria is based on the assumption that proteins in a functional module only loosely interact with proteins not in the functional module [79]. Thus, aligned proteins in a conserved functional module are unlikely to have a large number of conserved interactions with aligned proteins that are not in the conserved functional module.

The heuristic algorithm presented in [79] tends to produce relatively small local alignment networks. The bottom-up method of growing subgraphs utilized by the algorithm makes it difficult to grow larger subgraphs. The criteria that the vertex being added to the subgraph must be connected to the current subgraph by edges whose sum is greater than zero is a tight restriction. It requires the aligned pair of proteins represented by a vertex in the subgraph to have a large number of conserved interactions. According to [79], a conserved interaction happens only when an interaction exists in both species' protein interaction networks. This means that proteins in the conserved functional module found by this algorithm must interact with the vast majority of other proteins in the functional module. If the functional module contains a module hub, or if the protein-protein interaction data contains a great deal of noise,
then the size of the local alignment networks will be limited to relatively small numbers of proteins.

4.3 Our approach to finding high scoring local alignment networks

Here, we will present a heuristic algorithm for finding local alignment networks in a global alignment network by heuristically attempting to find solutions for the optimization version of the maximum weight induced subgraph problem given in Definition 4.3.1. The solution to the optimization version of the problem will be, by definition, larger than the solutions to the the search problem given in Definition 4.2.1. Therefore, the local alignment network determined by the solution will be more likely to correspond to an actual conserved functional module.

Definition 4.3.1 Given a graph $G(V, E)$, find a subset of nodes, $\tilde{V} \in V$ such that the sum of the weights of the edges in the subgraph induced by $\tilde{V}$ is greater than the sum of the weights of the edges in the subgraph induced by any other subset of nodes $S \subseteq V$.

Like the search version of the problem, the optimization version of the problem is $NP$-complete. Therefore, an efficient algorithm for finding exact solutions to the optimization problem is unlikely to exist. Our algorithm finds heuristic solutions to the optimization problem in Definition 4.3.1. However, related algorithms in [19] and [37] have been shown to work well for finding solutions to the densest subgraph and dense $k$-subgraphs problems.

The pseudocode for our algorithm is given in Algorithm 4.3.1. The algorithm finds subgraphs with large sums of edge weights by initially considering the entire graph. It calculates a weight for the graph as the sum of the edge-weights in the graph. The algorithm then iteratively removes the vertex that has the lowest sum of edge-weights for its incident edges and associates the sum of the edge-weights remaining in the graph as the weight of the new graph. Once every vertex has been removed it finds the subgraph that had the maximum weight and outputs it as the maximum weight induced subgraph for the given graph.

The runtime complexity of our algorithm is as follows. Lines 1 and 5 must sum over all of the edges in the graph. Therefore, the complexity of lines 1–5 is $O(|E|)$. Every iteration of lines 6–13 of our algorithm involves finding and removing the minimum weight vertex as well as updating the weights of the remaining vertices. This can be done in $O(|V|)$ time. Since the loop on line 9 must run $|V|$ times, the complexity of lines 6–13 is $O(|V|^2)$. The complexity of line 14 is $O(|V|)$. Therefore, our algorithm will run in $O(|E| + |V|^2) \in O(|V|^2)$ time. This is an improvement over the runtime of the algorithm presented in Section 4.2. That algorithm had a worst-case runtime of $O(|V||E|) = O(|V|^3)$ time [79].
A graph $G(V, E, w)$

Output: A set of vertices that induce a subgraph with nearly maximum weight

foreach $v \in V$ do
  $w(v) \leftarrow \sum_{u \in N(v)} w(u, v)$
end

$V_{|V|} \leftarrow V$

$W(V_{|V|}) \leftarrow \sum_{(u,v) \in E} w(u, v)$

for $i$ from $|V|$ to 1 do
  $\nu' \leftarrow \arg\min_{v \in V_i} w(v)$

  $V_{i-1} \leftarrow V_i \setminus \nu'$

  foreach $u \in V_{i-1}$ do
    $w(u) \leftarrow w(u) - w(u, \nu')$
  end

  $W(V_{i-1}) \leftarrow W(V_i) - w(\nu')$
end

$i \leftarrow \arg\min_{0 \leq i \leq |V|} W(V_i)$

return $V_i$

Algorithm 4.3.1: Pseudocode for algorithm based on maximum induced subgraph optimization problem

4.4 Experimental Results

In this section, the benefits of our algorithm for finding local alignment networks (LANs) will be examined by experimentally comparing our algorithm’s performance to that of the original MaWISh algorithm. Both algorithms use as input a global alignment network (GAN). The method for constructing the GAN is presented in full in [79]. However, a brief overview of their method is given here for clarity.

Two protein interaction networks, $G_1$ and $G_2$, are needed to construct a GAN. In a protein interaction network, the vertices represent proteins in an organism and edges between the vertices represent interactions between the two proteins. Each vertex in the GAN represents one protein from $G_1$ and one protein from $G_2$. In addition, the vertices of the GAN are restricted to only those pairs of proteins across the two PPI networks that have significantly similar amino acid sequences. This is determined by the BLAST $E$-values of the two proteins being less than a given threshold, $T$. Edges exist between every possible pair of vertices in the GAN. The weights of these edges is determined by the likelihood that the two pairs of vertices conserve an interaction across the two species. For an edge between the vertices $\{u_1, u_2\}$ and $\{v_1, v_2\}$, the edge-weight is positive if there is a likely interaction between both $u_1$ and $v_1$ in $G_1$ and $u_2$ and $v_2$ in $G_2$. If there is a likely interaction between the two proteins in $G_1$ but not in $G_2$ or vice-versa, then the edge-weight is negative. The magnitude of the edge-weight is determined...
Table 4.2: Comparison of local alignment networks produced and runtime for 7 pairs of PPI networks

<table>
<thead>
<tr>
<th>PPI Pair</th>
<th>Algorithm</th>
<th>LANs</th>
<th>Median Size</th>
<th>Max. Weight</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. cerevisiae vs. C. elegans</td>
<td>Our Algorithm</td>
<td>13</td>
<td>24</td>
<td>31.01</td>
<td>0.02s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>111</td>
<td>3</td>
<td>36.14</td>
<td>0.08s ± 0.01s</td>
</tr>
<tr>
<td>S. cerevisiae vs. H. sapiens</td>
<td>Our Algorithm</td>
<td>32</td>
<td>35</td>
<td>81.26</td>
<td>0.91s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>459</td>
<td>3</td>
<td>15.29</td>
<td>5.10s ± 0.03s</td>
</tr>
<tr>
<td>C. elegans vs. D. melanogaster</td>
<td>Our Algorithm</td>
<td>14</td>
<td>35</td>
<td>81.76</td>
<td>0.29s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>183</td>
<td>3</td>
<td>26.75</td>
<td>1.21s ± 0.01s</td>
</tr>
<tr>
<td>C. elegans vs. H. sapiens</td>
<td>Our Algorithm</td>
<td>23</td>
<td>30</td>
<td>42.57</td>
<td>0.89s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>262</td>
<td>3</td>
<td>14.27</td>
<td>3.98s ± 0.03s</td>
</tr>
<tr>
<td>D. melanogaster vs. H. sapiens</td>
<td>Our Algorithm</td>
<td>24</td>
<td>43</td>
<td>180.04</td>
<td>1.88s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>385</td>
<td>3</td>
<td>22.73</td>
<td>12.47s ± 0.05s</td>
</tr>
<tr>
<td>E. coli vs. V. cholerae</td>
<td>Our Algorithm</td>
<td>25</td>
<td>28</td>
<td>546</td>
<td>0.02s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>228</td>
<td>4</td>
<td>169</td>
<td>0.15s ± 0.01s</td>
</tr>
<tr>
<td>E. coli vs. C. crescentus</td>
<td>Our Algorithm</td>
<td>14</td>
<td>15</td>
<td>45</td>
<td>0.00s ± 0.01s</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>93</td>
<td>2</td>
<td>19</td>
<td>0.02s ± 0.01s</td>
</tr>
</tbody>
</table>

with respect to the sequence similarity between $u_1, u_2$ and $v_1, v_2$. The more similar these two pairs of proteins are, the greater the magnitude of the edge weight is.

For the results in this section, we used protein interaction data from 7 different species (*Saccharomyces cerevisiae, Caenorhabditis elegans, Homo sapiens, Drosophila melanogaster, Escherichia coli K-12, Vibrio cholerae*, and *Caulobacter crescentus*) to generate 7 different GANs. The protein interaction network data was obtained from the Database of Interacting Proteins (DIP) [110], and the GANs were constructed by the previously mentioned method. The two algorithms were run on these GANs to generate the LANs. Figure 4.2 shows that the size of the LANs generated by our algorithm tended to be larger than the alignments generated by the MaWISh algorithm. This is in line with our hypothesis, and likely due to the fact that our algorithm heuristically finds the maximum weight induced subgraph in the GAN (Definition 4.3.1) and the MaWISh algorithm heuristically finds induced subgraphs whose weights are greater than some threshold (Definition 4.2.1).

The effectiveness of the two algorithms was measured in two ways. First, we compared the two algorithms on a strictly computational basis. This included comparing the number of LANs found, the size of the different LANs, the weight of maximum weight subgraph found, and the total runtime of the algorithms. The results of this comparison are shown in Table 4.2. Our
algorithm tended to find fewer LANs, but the size and weight of the LANs tended to be larger than those of MaWISH. Our algorithm also tended to have shorter runtimes than the MaWISH algorithm.

We have also evaluated the alignment algorithms according to the biological relevance of the LANs produced. Our assumption is that the functional modules identified by our algorithm are functionally homogeneous. We evaluate each alignment’s functional homogeneity as follows. First, we followed the approach outlined by Flannick et al. in [40]. We defined as a true positive any alignment that could be annotated with a statistically significant Gene Ontology (GO) term [9]. This was determined by using the online GO::TermFinder application [17] to annotate the LANs and to determine the statistical significance of the possible annotations through the calculation of each annotation’s associated p-values. If the GO::TermFinder application annotated the LAN with any GO term with a p-value of less than 0.05, then we counted the LAN as a true positive. To use the GO::TermFinder application, we had to filter out any protein in the LANs that did not have an NCBI-GI number. Under this evaluation method, the precision of the algorithm was the number of true positives divided by the total number of
LANs that did not have all of their proteins filtered out.

We developed our own related method of evaluating an alignment’s functional homogeneity using KEGG pathways [64] to avoid any bias introduced from using GO terms in the analysis. In this method, we defined as a true positive any alignment that could be annotated with a statistically significant KEGG pathway ID. The statistical significance of annotating a LAN with a KEGG pathway ID was calculated as its hypergeometric probability, which is the probability that a random sample of the same size as the LAN would have included at least as many proteins in the annotated KEGG Pathway. This is similar to the way that GO::TermFinder calculates the $p$-value of a given GO term annotation [17]. To calculate the hypergeometric probability of a KEGG pathway ID, we first had to filter out any protein from the LAN that did not have a KEGG gene ID and at least one associated KEGG pathway. A LAN was determined to be a true positive if it had at least one KEGG Pathway annotation with a hypergeometric probability of less than 0.05.

We also measured each algorithm’s ability to find LANs that cover (contains the proteins of) as many functional modules as possible. Ideally, if a conserved functional module exists across two species we would like the alignment algorithm to find a LAN that covers the module. We once again adopted a technique from the authors of [40] to evaluate each algorithm’s ability to identify conserved modules of functional homogeneity using KEGG pathways [64]. A pathway in a given species was defined to be ”covered” if at least three of its proteins belonged to the same LAN found by the algorithm. If a conserved KEGG pathway was covered in both species by the same alignment, then it was counted as a covered pathway. We report the number of covered pathways, in addition to the two measures of precision in Table 4.3. We were unable to use KEGG data to evaluate the biological relevance of the LANs produced when using the *E. coli* PPI network as input because of an inconsistency between our PPI network data and the KEGG pathway data for *E. coli*.

### 4.5 Concluding Remarks

We have presented a novel approach to finding local alignment networks that correspond to conserved functional modules between two species. The proposed formulation of the maximum weight induced subgraph optimization problem allows us to design an algorithm that can find larger local alignment networks by iteratively removing the minimum weight vertex from a global alignment network. The worst-case runtime of the algorithm is $O(|E| + |V|^2)$ is an improvement over the previous MaWISH algorithm that had a worst-case runtime of $O(|E||V|^2)$. In addition to this worst-case complexity improvement, experimental observations have shown that our algorithm has an order of magnitude runtime improvement over MaWISH for several real-world examples. The local alignment networks found by our algorithm also offer
improvements in precision and recall values over the previous MaWISh algorithm.
Table 4.3: Comparison of precision-recall results for 7 pairs of PPI networks.

<table>
<thead>
<tr>
<th>PPI Pair</th>
<th>Algorithm</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GO Terms</td>
<td>KEGG Pathways</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. cerevisiae</td>
<td>Our Algorithm</td>
<td>73%</td>
<td>92%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>89%</td>
<td>83%</td>
</tr>
<tr>
<td>C. elegans</td>
<td>Our Algorithm</td>
<td>82%</td>
<td>93%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>61%</td>
<td>90%</td>
</tr>
<tr>
<td>S. cerevisiae</td>
<td>Our Algorithm</td>
<td>100%</td>
<td>93%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>100%</td>
<td>66%</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>Our Algorithm</td>
<td>92%</td>
<td>78%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>75%</td>
<td>88%</td>
</tr>
<tr>
<td>C. elegans</td>
<td>Our Algorithm</td>
<td>60%</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>45%</td>
<td>83%</td>
</tr>
<tr>
<td>D. melanogaster</td>
<td>Our Algorithm</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>77%</td>
<td>87%</td>
</tr>
<tr>
<td>C. elegans</td>
<td>Our Algorithm</td>
<td>70%</td>
<td>94%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>42%</td>
<td>90%</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>Our Algorithm</td>
<td>85%</td>
<td>95%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>64%</td>
<td>88%</td>
</tr>
<tr>
<td>D. melanogaster</td>
<td>Our Algorithm</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>87%</td>
<td>95%</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>Our Algorithm</td>
<td>96%</td>
<td>95%</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>68%</td>
<td>87%</td>
</tr>
<tr>
<td>E. coli</td>
<td>Our Algorithm</td>
<td>84%</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>74%</td>
<td>N/A</td>
</tr>
<tr>
<td>V. cholerae</td>
<td>Our Algorithm</td>
<td>58%</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>71%</td>
<td>N/A</td>
</tr>
<tr>
<td>E. coli</td>
<td>Our Algorithm</td>
<td>67%</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>41%</td>
<td>N/A</td>
</tr>
<tr>
<td>C. crescentus</td>
<td>Our Algorithm</td>
<td>67%</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>MaWISh</td>
<td>48%</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Chapter 5

On Finding Dense Subgraphs with Size Constraints

5.1 Introduction

Given an undirected graph with \( n \) vertices, the density of one of its subgraph is the sum of the subgraph’s edge weights divided by the number of vertices in the subgraph. Finding the densest subgraph with some size constraints is an important problem in different contexts, such as search auction graphs, blog link graphs, and protein-protein interaction graphs [44, 61, 72, 73].

To meet this need, Andersen [4] introduced the densest at-least-\( k \)-subgraph problem (DalkS) and the densest at-most-\( k \)-subgraph problem (DamkS) in [4]. The densest at-least-\( k \)-subgraph problem (DalkS) is the problem of finding an induced subgraph of maximum density among all subgraphs with at least \( k \) vertices. The densest at-most-\( k \)-subgraph problem (DamkS) is the problem of finding an induced subgraph of maximum density among all subgraphs with at most \( k \) vertices. In this paper, we consider algorithms for effectively solving the DalkS and DamkS problems.

The DalkS and DamkS problems are closely related to the densest subgraph problem [19] and the dense \( k \)-subgraph problem [37]. The densest subgraph problem is the problem of finding an induced subgraph of maximum density. In recent years, the densest subgraph problem has been studied extensively, as it plays an important role in network analysis [45, 80]. The densest subgraph can also be used to facilitate functional discovery in massive biological networks [61]. A number of polynomial time algorithms were designed to find the densest subgraph of a given graph—for example, in [42, 48], polynomial time algorithms based on the maximum flow problem were given. There are approximation algorithms with significantly better time bounds. For instance, in [19], Charikar gave a 2-approximation algorithm that runs in linear time. In [66], Kannan et al gave an \( O(\log n) \)-ratio approximation algorithm for finding the
densest subgraph in directed graphs.

The dense $k$-subgraph problem is the parameterized version of the densest subgraph problem, which is to find the densest induced subgraph of exactly $k$ vertices. The dense $k$-subgraph problem is $NP$-hard [38], which means that there is no polynomial time algorithm for this problem unless $P = NP$; however, there are several approximation algorithms. Kortsarz et al presented a polynomial time approximation algorithm with the approximation ratio of $O(n^{0.3885})$ [76]. In [37], Feige et al gave an approximation algorithm with ratio $O(n^\delta)$, for some $\delta < 1/3$. In [38], Feige and Seltser proposed an approximation algorithm based on semi-definite programming (SDP) with an approximation ratio of $n/k$. Siravastav et al independently developed another SDP-based approximation algorithm with the same approximation ratio [100]. In [54], Han et al proposed an approximation algorithm based on linear programming (LP) and an approximation algorithm based on SDP. The LP-based approximation algorithm had an approximation ratio of $n/k$, but the approximation ratio of the SDP-based algorithm was greater than $n/k$ for many values of $k$.

For DalkS, Andersen gave a 3-approximation algorithm that runs in time $O(m + n \log n)$ and a 2-approximation algorithm that runs in polynomial time [4]. The 3-approximation algorithm is an extension of Charikar’s greedy algorithm for the densest subgraph problem [19] and the 2-approximation algorithm is based on the parametric flow algorithm of Gallo et al [42].

Andersen gave a reduction from DkS to DamkS where a polynomial time approximation algorithm for DamkS with ratio $\gamma$ can produce a polynomial time approximation algorithm for DkS with ratio $4(\gamma^2 + \gamma)$. The author also showed that DamkS is $NP$-complete, but no approximation algorithms for DamkS were provided.

In [4], Andersen proposed the following open problem: “It is natural to ask whether there is a polynomial time algorithm for the densest at-least-$k$-subgraph problem. We do not know of such an algorithm, nor have we proved that DalkS is $NP$-complete.”

Our Results

In this Chapter, we introduce the definition of a minimum cut with at least $k$ vertices and give a polynomial time algorithm for it when $k$ is bounded by some constant $c$. Based on the algorithm for the minimum cut with at least $k$ vertices, we propose one polynomial time algorithms for DalkS when $k$ is bounded by some constant $c$. We still propose an LP-based polynomial time algorithms for DalkS when $k$ is bounded by some constant $c$. We also present two approximation algorithms for DamkS. The first approximation algorithm gives an approximation ratio of $\frac{n-1}{k-1}$, and the second has ratio $O(n^\delta)$ for some $\delta < 1/3$.

The remainder of the Chapter is organized as follows. Section 5.2 reviews some basic definitions. In section 5.3, we introduce the problem of finding a minimum $s$-$t$ cut with at least $k$ vertices and give a polynomial time algorithm for solving it when $k$ is bounded by a constant. In section 5.4, we give an algorithm for solving DalkS based on a minimum $s$-$t$ cut with at least...
Section 5.5, we design an LP-based polynomial time algorithm for DalkS when \( k \) is bounded by a constant. Section 5.6 describes our two approximation algorithms for DamkS. Finally, Section 5.7 discusses these results as well as some open problems inspired by these findings.

5.2 Definitions

The following basic definitions come from [4]. Let \( G = (V, E) \) be an undirected graph in which every edge is assigned a positive weight by a weight function \( w : E \rightarrow \mathbb{R}^+ \). Also, define the weighted degree of a vertex \( v \) in \( G \), \( w(v, G) \), to be the sum of the weights of the edges incident with \( v \), and let the total weight of \( G \), \( W(G) \), be the sum of the weights of all of the edges in \( G \).

**Definition 5.2.1** For any induced subgraph \( H \) of \( G \), we define the density of \( H \) to be \( d(H) = \frac{W(H)}{|H|} \).

**Definition 5.2.2** For an undirected graph \( G \), we define the following quantities:

- \( \text{dal}(G,k) := \text{the maximum density of an induced subgraph on at least } k \text{ vertices} \),
- \( \text{dam}(G,k) := \text{the maximum density of an induced subgraph on at most } k \text{ vertices} \),
- \( \text{dex}(G,k) := \text{the maximum density of an induced subgraph on exactly } k \text{ vertices} \), and
- \( \text{dmax}(G) := \text{the maximum density of any induced subgraph} \).

**Definition 5.2.3** The densest at-least-k-subgraph problem (DalkS) is the problem of finding an induced subgraph of at least \( k \) vertices with density \( \text{dal}(G,k) \). Similarly, DamkS, DkS, and the densest subgraph problem are the problems of finding an appropriate induced subgraph with density \( \text{dam}(G,k) \), \( \text{dex}(G,k) \), or \( \text{dmax}(G) \), respectively.

Now, we give a formal definition for an approximation algorithm of DamkS. Approximation algorithms for DalkS, DkS, and the densest subgraph problem are defined similarly.

**Definition 5.2.4** An algorithm \( A(G,k) \) is a \( \gamma \)-approximation algorithm for the densest at-most-k-subgraph problem if for any graph \( G \) and integer \( k \), it returns an induced subgraph \( H \) of \( G \) with at most \( k \) vertices such that its density \( d(H) \) satisfies \( \frac{\text{dam}(G,k)}{d(H)} \leq \gamma \).
5.3 The Minimum $s$-$t$ Cut

In this section, we introduce the definition of a minimum cut with at least $k$ vertices, a minimum cut with at most $k$ vertices and a minimum cut with exact $k$ vertices. We show that the minimum cut with at least $k$ vertices problem is polynomial time solvable when $k$ is bounded by a constant $c$.

Let $G = (V, E)$ be an undirected graph with vertex set $V$ and edge set $E$, where $|V| = n$ and $|E| = m$ and each edge is assigned a positive capacity value.

**Definition 5.3.1** A cut is a partition of the vertex set $V$ into two sets $V_1$ and $V_2$. A cut edge set $C = \{(v_1, v_2) \in E : v_1 \in V_1, v_2 \in V_2\}$ is associated with every cut. The capacity of a cut is the sum of the capacities of the cut edges.

**Definition 5.3.2** Given vertices $s, t \in V$, an $s$-$t$ cut is a cut $(V_1, V_2)$ such that $s \in V_1$ and $t \in V_2$. An $s$-$t$ cut with at least $k$ vertices is an $s$-$t$ cut $(V_1, V_2)$ such that $|V_1 \setminus \{s\}| \geq k$. An $s$-$t$ cut with at most $k$ or exactly $k$ vertices is defined similarly.

**Definition 5.3.3** For an undirected graph $G$, we define the following quantities:

- $Cal(G, k) :=$ the minimum capacity value of an $s$-$t$ cut with at least $k$ vertices,
- $Cam(G, k) :=$ the minimum capacity value of an $s$-$t$ cut with at most $k$ vertices,
- $Cex(G, k) :=$ the minimum capacity value of an $s$-$t$ cut with exactly $k$ vertices

It is a well-known result that finding a minimum cut is polynomial time solvable. In the following, we show that finding a minimum cut with at least $k$ vertices is polynomial time solvable when $k$ is bounded by some constant $c$.

First, we show that finding a minimum cut with at least $k$ vertices is polynomial time solvable when $k = 1$.

**Lemma 5.3.4** When $k = 1$, finding a minimum cut with at least $k$ vertices is polynomial time solvable.

**Proof.** Let $v_1, \ldots, v_{n-2}$ be all vertices in $V \setminus \{s, t\}$. We construct $n - 2$ graphs $G_1, \ldots, G_{n-2}$ as follows.

First, we describe the notion of contracting graph vertices [67]. When two vertices $v$ and $v'$ are contracted, they are replaced by a vertex $u$ where the capacities of the edges incident on $u$ are the sum of the capacities of edges incident on $v$ and $v'$. Formally, we delete $v$, $v'$, and all edges incident on them and add a new vertex $u$ with edges $(u, w)$ for all $w$ incident
to \(v\) or \(v'\), where \(\text{capacity}(u, w) = \text{capacity}(v, w) + \text{capacity}(v', w)\). The rest of the graph is unchanged. We will use \(G/\{v, v'\}\) to denote the graph formed by contracting vertices \(v\) and \(v'\) in \(G\). Extending this definition, we will let \(G/F\) denote the graph produced by contracting all vertices in a vertex set \(F\). Due to the commutative and associative properties of addition, it does not matter in what order the associated contractions are performed.

Let \(G_1 = G/\{s, v_1\}, G_2 = G/\{s, v_2\}, \ldots, G_{n-2} = G/\{s, v_{n-2}\}\), and let \(s_1, s_2, \ldots, s_{n-2}\) denote the new, contracted vertices of \(G_1, G_2, \ldots, G_{n-2}\), respectively.

For every \(G_i\) \((1 \leq i \leq n - 2)\), we find a minimum \(s_i\)-\(t\) cut and compute its capacity value \(c_i\). Let \(m = \min_{1 \leq i \leq n-2} c_i\). We now show that \(m\) is the value of a minimum \(s\)-\(t\) cut in \(G\) with at least 1 vertex.

Suppose a minimum cut \(s\)-\(t\) cut in \(G\) with at least 1 vertex is \((V_1, V_2)\). Let \(m' = \text{capacity}(V_1, V_2)\). Since \(|V_1 \setminus \{s\}| \geq 1\), there exists \(v_j \in V_1\). We can contract \(s\) and \(v_j\) to form the graph \(G_j\), with contracted vertex \(s_j\). By the definition of contraction, \(\text{capacity}(s_j, w) = \text{capacity}(s, w) + \text{capacity}(v_j, w)\) for any \(w \in V_2\), so the capacity of the cut \((V_1 \setminus \{s, v_j\} \cup \{s_j\}, V_2)\) in \(G_j\) equals the capacity \(m'\) of cut \((V_1, V_2)\) in \(G\). Thus, a minimum cut of \(G/\{s, v_j\}\) \(\leq c'\), so \(m \leq m'\).

On the other hand, suppose \(m = c_t\) is the minimum value of \(\{c_1, \ldots, c_{n-2}\}\) and corresponds to cut \((V_1, V_2)\) in \(G_t\). Then \((V_1 \setminus \{s_t\} \cup \{s, v_j\}, V_2)\) is an \(s\)-\(t\) cut in \(G\) with at least 1 vertex \((v_j)\). Again, by the definition of contraction, the capacity \(c_j\) of cut \((V_1, V_2)\) in \(G_j\) is the capacity of \((V_1 \setminus \{s_j\} \cup \{s, v_j\}, V_2)\) in \(G\). So \(m' \leq c_t = m\). Hence, \(m = m'\).

The problem of finding a minimum \(s\)-\(t\) cut in a graph with \(n\) vertices is solvable in time \(O(n^3)\) \([70]\)\(^1\). Since every \(G_i\) has \(n - 1\) vertices, the time of finding each minimum \(s_i\)-\(t\) cut is \(O(n^3)\), so the time of finding a minimum cut in \(G\) with at least 1 vertex is at most \((n - 2)O(n^3) = O(n^4)\).

**Lemma 5.3.5** When \(k\) is bounded by a constant value \(c\), finding a minimum cut with at least \(k\) vertices is polynomial time solvable.

**Proof.** Let \(v_1, \ldots, v_{n-2}\) be the vertices of \(V \setminus \{s, t\}\), and let \(p = \binom{n-2}{k}\).

Let \(F_1, \ldots, F_p\) be all of the subsets of \(\{v_1, \ldots, v_{n-2}\}\) with \(k\) vertices. We construct \(p\) graphs \(G_1, \ldots, G_p\) as follows: let \(G_1 = G/\{s, F_1\}, G_2 = G/\{s, F_2\}, \ldots, G_p = G/\{s, F_p\}\), and let \(s_1, \ldots, s_p\) denote the new, contracted vertices of their respective graphs.

For every \(G_i\) \((1 \leq i \leq p)\), we find a minimum \(s_i\)-\(t\) cut and compute its capacity value \(c_i\). Let \(c' = \min_{1 \leq i \leq p} c_i\). Similar to our proof of Lemma 5.3.4, \(c'\) will be the capacity of a minimum \(s\)-\(t\) cut in \(G\) with at least \(k\) vertices.

The problem of finding a minimum \(s\)-\(t\) cut in a graph with \(n\) vertices is solvable in time \(O(n^3)\) \([70]\). Since every \(G_i\) has \(n - k\) vertices, the time of finding a minimum \(s_i\)-\(t\) cut is

\(^1\)After Karzanov’s algorithm, there have been many time-bound improved algorithms for minimum \(s\)-\(t\) cut problems. In [68, 67, 101], there are comprehensive surveys.
So the time of finding a minimum cut with at least $k$ vertex in $G$ is at most $pO(n^3) = \binom{n-2}{k}O(n^3) = O(n^k)O(n^3) = O(n^{k+3})$, and since $k$ is bounded by a constant $c$, this is an $O(n^{c+3})$-time algorithm, which is polynomial.

5.4 An algorithm for DalkS based on the minimum $s$-$t$ cut with at least $k$ vertices

In [48], Goldberg proposes a polynomial time algorithm for the densest subgraph problem based on the minimum $s$-$t$ cut problem. In this section, we generalize Goldberg’s algorithm to find the densest subgraph with at least $k$ vertices based on the algorithm of finding a minimum $s$-$t$ cut with at least $k$ vertices. In section 3, we showed that when $k$ is bounded by a constant, a minimum $s$-$t$ cut with at least $k$ vertices can be found in polynomial time. Thus, DalkS will also be solvable in polynomial time when $k$ is bounded by a constant.

The basic idea of the algorithm is as follows. At each stage, we guess a value $g$ for $dal(G, k)$. Then, we construct a new graph and compute a minimum $s$-$t$ cut with at least $k$ vertices that enables us to decide whether $g \leq dal(G, k)$ or $g > dal(G, k)$. Thus, we search for $dal(G, k)$ in a binary fashion.

As in [48], we first consider the simplest case, in which $G$ is an unweighted, undirected graph—this is equivalent to a weighted graph in which all weights are either 1 or 0.

Let $G = (V, E)$ be an unweighted, undirected graph, and let $n = |V|$ and $m = |E|$. Let $d_i$ be the degree of vertex $i$ of $G$. Given a “guess” $g$, we construct another graph $N = (V_N, E_N)$ as follows [48].

We add source $s$ and sink $t$ to the set of vertices of $G$; connect source $s$ to every node $i$ of $G$ by an edge of capacity $m$; and connect every node $i$ of $G$ to the sink $t$ by an edge of capacity $(m + 2g - d_i)$. Each edge of $G$ is assigned capacity 1. Since $d_i \leq m$ for all $i$, all capacities are positive. Whereas Goldberg constructs a directed graph, we construct an undirected graph. This change makes no difference, though.

Let $(S, T)$ be an $s$-$t$ cut in $N$, and let $V_1 = S \setminus \{s\}$ and $V_2 = T \setminus \{t\}$. In [48], Goldberg proved the following result:

**Lemma 5.4.1** The capacity of the cut $(S, T) = m|V| + 2|V_1|(g - D_1)$, where $D_1$ is the density of the subgraph of $G$ generated by $V_1$.

In order to decide whether $g \leq dal(G, k)$ or $g > dal(G, k)$, we generalize Theorem 1 in [48] to the following result.

**Theorem 5.4.2** Let $R$ be the value of a minimum $s$-$t$ cut $(S, T)$ with at least $k$ vertices. If $R \leq m|V|$, then $g \leq dal(G, k)$. If $R > m|V|$, then $g > dal(G, k)$. 

46
**Proof.** Let $V_1 = S \setminus \{s\}$. Since $(S,T)$ is an $s$-$t$ cut with at least $k$ vertices, $|V_1| \geq k$, and $R = m|V| + 2|V_1|(g - D_1)$ by Lemma 5.4.1.

If $R \leq m|V|$, then $m|V| + 2|V_1|(g - D_1) \leq m|V|$, so $2|V_1|(g - D_1) \leq 0$. Hence, $g \leq D_1 \leq dal(G,k)$.

Let $H$ be a subgraph such that $d(H) = dal(G,k)$ and $|H| \geq k$. Let $S = H \cup \{s\}$ and $T = (V \setminus H) \cup \{t\}$. So, $(S,T)$ is a cut with at least $k$ vertices, and the capacity of cut $(S,T)$ equals $m|V| + 2|H|(g - d(H))$. Since $R$ is the minimum capacity of an $s$-$t$ cut with at least $k$ vertices, $m|V| + 2|H|(g - d(H)) \geq R$. If $R > m|V|$, then $m|V| + 2|H|(g - d(H)) > m|V|$, so $g > d(H) = dal(G,k)$. ■

As shown in [48], the smallest distance between two different possible subgraph densities for undirected graphs is $\frac{1}{n(n-1)}$. Thus, we have the following theorem.

**Theorem 5.4.3** If $H$ is a subgraph of $G$ with at least $k$ vertices, and there exists no subgraph $H'$ with at least $k$ vertices such that $d(H') \geq d(H) + \frac{1}{n(n-1)}$, then $d(H) = dal(G,k)$

The algorithm for $dal(G,k)$ is as follows:

```
Input: A unweighted graph $G = (V,E)$ with $n$ vertices, $m$ edges and an integer $k$
Output: An induced subgraph with at least $k$ vertices

1   $\ell = 0$
2   $u = m$
3   $V_1 = V$
4   while $u - \ell \geq \frac{1}{n(n-1)}$ do
5       $g = \frac{u + \ell}{2}$
6       Construct $N = (V_N,E_N)$
7       Find a minimum $s$-$t$ cut $(S,T)$ with at least $k$ vertices and compute its cut value $R$
8       if $R > m|V|$ then
9           $u = g$
10      else
11          $\ell = g$
12          $V_1 = S \setminus \{s\}$
13      end
14   end
15 return the subgraph of $G$ induced by $V_1$
```

**Algorithm 5.4.1**: The algorithm for $dal(G,k)$

In the process of the algorithm, the value of $u$ or $\ell$ changes in each loop and the value of $u - \ell$ decreases gradually. Thus, at the beginning of each loop, the value of $g$ also changes,
which influence the capacities of edges in the graph $N$. In particular, $V_1$ doesn’t influence the construction of $N$.

In the algorithm, when $R \leq m|V|$, the value of $\ell$ is changed and $V_1$ is set $S \setminus \{s\}$. Thus, by the step 7 in the algorithm, $V_1$ contains at least $k$ vertices of a subgraph of $G$. By the proof of Theorem 4.2, $V_1$ has density at least $\ell$. When the algorithm stops, we know that there is no subgraph of at least $k$ vertices with density $\ell + \frac{1}{m(n-1)}$ or greater, so the subgraph returned is a maximum density subgraph with at least $k$ vertices by the result of Theorem 4.3.

If $M(n,m)$ be the time of finding a minimum $s$-$t$ cut with at least $k$ vertices in a graph of $n$ vertices and $m$ edges, the time of Algorithm 5.4.1 is $O(M(n, n + m) \log n)$. Thus, we get the following conclusion:

**Theorem 5.4.4** If finding a minimum $s$-$t$ cut with at least $k$ vertices is solvable in polynomial time, then DalkS is polynomial time solvable.

By lemma 5.3.5, we get:

**Theorem 5.4.5** When $k$ is bounded by a constant $c$, then DalkS is polynomial time solvable in time $O(n^{c+3} \log n)$.

Using Megiddo’s technique [82], Goldberg generalizes his algorithm to weighted graphs. Similarly, our algorithm can generalized to weighted graphs. Details can be found [48], we omit them here.

### 5.5 An LP-based algorithm for DalkS when $k$ is bounded by some constant $c$

In this section, we design an LP-based polynomial time algorithm for DalkS when $k$ is bounded by some constant $c$.

First, we introduce the densest subgraph containing a vertex problem.

**Definition 5.5.1** For an undirected graph $G$ and a vertex $v_f$ of $G$, we define the following quantities:

$mdv(G, v_f) :=$ the maximum density of an induced subgraph containing vertex $v_f$.

**Definition 5.5.2** The densest subgraph containing a vertex problem: given an undirected graph $G$ and a vertex $v_f$ of $G$, find an induced subgraph containing $v_f$ with density $mdv(G, v_f)$. We denote this problem as $DS(G, v_f)$. 48
If $G$ has $n$ vertices, there are $2^{n-1}$ vertex sets in $G$ containing $v_f$. Thus, one natural computational problem is whether $DS(G, v_f)$ is $P$ or $NP$-hard. In the following, we show that $DS(G, v_f)$ is $P$ by providing an LP-based polynomial time algorithm.

In order to design an LP formulation for $DS(G, v_f)$ problem, we assign a variable $y_i$ for every vertex $v_i$ in $G$ and a variable $x_{ij}$ for every edge $(v_i, v_j) \in E$. Thus, we propose the following $LP_1$ formulation for the $DS(G, v_f)$ problem. Charikar proposed an LP formulation for the densest subgraph problem in [19].

$$\begin{align*}
\text{maximize} & \sum_{(v_i, v_j) \in E} x_{ij} \quad (LP_1) \\
\text{subject to} & 0 \leq x_{ij} \leq y_i, 0 \leq x_{ij} \leq y_j \text{ for } (v_i, v_j) \in E(G) \quad (1) \\
& \sum_{i=1}^{n} y_i \leq 1 \quad (2) \\
& 1 \geq y_f \geq y_i \text{ for } i \neq f \quad (3)
\end{align*}$$

Based on $LP_1$, we design a polynomial time algorithm for the $DS(G, v_f)$ problem.

**Input:** A graph $G = (V, E)$ and an vertex $v_f$ of $G$, where $|V| = n, |E| = m$

**Output:** An induced subgraph containing $v_f$

1. Solving $LP_1$ for $DS(G, v_f)$, get the optimum solution $y_i^*$ and $x_{ij}^*$
2. Construct $n$ vertex sets: $S_1 = \{v_i|y_i^* \geq y_1^*\}, S_2 = \{v_i|y_i^* \geq y_2^*\}, \ldots, S_n = \{v_i|y_i^* \geq y_n^*\}$
3. Compute the density $d(S_i)$ for all $i$
4. return the subgraph $S_j$ such that $d(S_j) = \max_{1 \leq i \leq n} d(S_i)$

**Algorithm 5.5.1:** The polynomial time algorithm for $DS(G, v_f)$ problem

In [19], Charikar use mathematical analysis methods to show that the optimum value of his LP formulation for the densest subgraph problem equals the maximum density of that graph. In the following, we will use algebraic methods to prove that $d(S_j)$ is equal to $mdv(G, v_f)$.

**Theorem 5.5.3** $d(S_j) = mdv(G, v_f)$

**Proof.** Since $y_1^* \geq y_j^*, v_f \in S_j$. Thus $d(S_j) \leq mdv(G, v_f)$. Let $OPT(LP_1)$ denote the optimum value of $LP_1$. We will prove that $d(S_j) \geq OPT(LP_1) \geq mdv(G, v_f)$. Suppose that $d(S_j) < OPT(LP_1)$. We will show that this assumption leads to a contradiction. In $\{y_1^*, \ldots, y_n^*\}$, let $y_i^* < \ldots < y_r^*$ be the sorted, distinct values of $y_1^*, \ldots, y_n^*$, and let $n_1, \ldots, n_r$ be the number of instances of $y_1^*, \ldots, y_r^*$ in the optimal solution. Thus, $n_1 + \ldots + n_r = n$.

Thus, $y_i^* = y_{i_1}^* + (y_{i_2}^* - y_{i_1}^*), y_{i_2}^* = y_{i_1}^* + (y_{i_2}^* - y_{i_1}^*), \ldots, y_{i_r}^* = y_{i_1}^* + (y_{i_r}^* - y_{i_1}^*) + (y_{i_{r-1}}^* - y_{i_r}^*) + \ldots + (y_{i_1}^* - y_{i_{r-1}}^*)$.

So $y_1^* + \ldots + y_n^* = n_1 y_{i_1}^* + \ldots + n_r y_{i_r}^* = n y_{i_1}^* + (n - n_1)(y_{i_2}^* - y_{i_1}^*) + \ldots + n_r (y_{i_r}^* - y_{i_{r-1}}^*)$.

Since $|S_{i_1}| = n, |S_{i_2}| = n - n_1, \ldots, |S_{i_r}| = n_r$, so $y_1^* + \ldots + y_n^* = |S_{i_1}| y_{i_1}^* + |S_{i_2}| (y_{i_2}^* - y_{i_1}^*) + \ldots + |S_{i_r}| (y_{i_r}^* - y_{i_{r-1}}^*)$. 

49
Let $E(y_i^*) = \{(v_j, v_k) \in E(G)|x_{jk}^* \geq y_i^*\}$. By constraint (1) in $LP_1$, we see that $x_{jk}^*$ is the minimum of all $x_{jk}^*$, $y_i^*$. Thus, the set of all $x_{jk}^*$ is also a bag of $y_i^*, \ldots, y_i^*$. Similarly, we get
\[
\sum_{(v_i, v_j) \in E} x_{ij}^* = |E(y_i^*)| y_i^* + |E(y_i^*)| (y_i^* - y_i^*) + \ldots + |E(y_i^*)| (y_i^* - y_i^*).
\]

Since $x_{jk}^*$ is the minimum of all $x_{jk}^*$, $y_i^*$, $y_i^*$ is the subgraph with the minimum density from $\{v_i^*, \ldots, v_i^*\}$. Similarly, a greedy approximation algorithm for the densest subgraph problem is as follows: starting from $(G, v_f)$ where $G$ has $n$ vertices, we repeatedly remove the non-$v_f$ vertex with the minimum degree in all non-$v_f$ vertices until the subgraph has only one vertex. Let $G_i$ denote each subgraph of $i$ vertices formed in the process, where $i \in \{1, \ldots, n\}$. We then output the subgraph with the maximum density from $\{G_1, \ldots, G_n\}$. It is obvious that the running time of the greedy algorithm is $O(n^2)$.

Theorem 5.5.4 The running time of Algorithm 5.5.1 is $O(n^{3.5}L)$.  

In [19], Charikar designs a greedy algorithm to compute an approximate solution of the densest subgraph. Similarly, a greedy approximation algorithm for the $DS(G, v_f)$ problem is as follows: starting from $(G, v_f)$ where $G$ has $n$ vertices, we repeatedly remove the non-$v_f$ vertex with the minimum degree in all non-$v_f$ vertices until the subgraph has only one vertex. Let $G_i$ denote each subgraph of $i$ vertices formed in the process, where $i \in \{1, \ldots, n\}$. We then output the subgraph with the maximum density from $\{G_1, \ldots, G_n\}$. It is obvious that the running time of the greedy algorithm is $O(n^2)$. 

50
In the following, we show that the greedy approximation algorithm for \( DS(G, v_f) \) has approximation ratio \( \frac{8}{3} \).

Theorem 5.5.5 The greedy approximation algorithm for \( DS(G, v_f) \) is of approximation ratio \( \frac{8}{3} \).

**Proof.** We prove the claim by induction on the vertex number \( n \) of \( G \). Basis: when \( n = 3 \), it is easy to verify the conclusion. Since there are four non-isomorphic graphs with 3 vertices, by checking all cases, we notice that the greedy algorithm finds the exact answer in all cases. (This answer may be \( mdv(G, v_f) = 0, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \) or 1, depending on \( G \) and \( v_f \).) Thus, when \( n = 3 \), the conclusion holds.

Inductive Hypothesis: We suppose the conclusion holds when \( n = x - 1 \), for some \( x \) such that \( x - 1 \geq 3 \).

We will show that the conclusion also holds for \( n = x \).

Let \( v_1 \) be the first vertex that is removed; i.e., \( v_1 \) is the vertex of minimum degree out of all non-\( v_f \) vertices. So, \( G_{x-1} = G \setminus \{v_1\} \). There are two cases for \( v_1 \): either \( v_1 \) does not appear in the vertex set of the optimum solution of \( DS(G, v_f) \) or \( v_1 \) does appear in the optimum solution of \( DS(G, v_f) \).

When \( v_1 \) does not appear in the vertex set of the optimum solution of \( DS(G, v_f) \), we get \( mdv(G_{x-1}, v_f) = mdv(G, v_f) \). Let \( A(G) \) be the output solution of greedy approximation algorithm for \( G \). Let \( A(G_{x-1}) \) be the solution produced by the greedy approximation algorithm for \( G_{x-1} \). By the induction hypothesis, \( mdv(G_{x-1}, v_f)/A(G_{x-1}) \leq \frac{8}{3} \). Since \( A(G) = \max\{d(G), A(G_{x-1})\} \), we get \( mdv(G, v_f)/A(G) \leq mdv(G_{x-1}, v_f)/A(G_{x-1}) \leq \frac{8}{3} \).

Suppose \( v_1 \) does appear in the vertex set of the optimum solution of \( DS(G, v_f) \). Let \( deg(G, v) \) denote the degree of a vertex \( v \) in a graph \( G \). We will show that \( deg(G, v_1) \geq mv(G, v_f) \). Suppose \( H = \{v_f, v_1, v_2, \ldots, v_{h-1}\} \) is a vertex set of the optimum solution of \( DS(G, v_f) \). Then \( mdv(G, v_f) = d(H) = (deg(H, v_f) + deg(H, v_1) + deg(H, v_2) + \ldots + deg(H, v_{h-1}))/2(h) \). So, \( (deg(H, v_f) + deg(H, v_1) + deg(H, v_2) + \ldots + deg(H, v_{h-1})) = 2h \cdot d(H) \).

Let \( H' = H \setminus \{v_1\} \). Then \( d(H') = (deg(H', v_f) + deg(H', v_2) + \ldots + deg(H', v_{h-1}))/2(h-1) = ((deg(H, v_f) + deg(H, v_1) + deg(H, v_2) + \ldots + deg(H, v_{h-1})) - 2deg(H, v_1))/(2(h-1)) = (2h \cdot d(H) - 2deg(H, v_1))/(2(h-1)) = \frac{2h \cdot d(H) - 2deg(H, v_1)}{2(h-1)} = \frac{d(H) - deg(H, v_1)}{h-1} \).

Since \( d(H) \geq d(H'), d(H) \geq d(H) + \frac{d(H) - deg(H, v_1)}{h-1} \). Thus, we get \( deg(H, v_1) \geq d(H) \). So \( deg(H, v_1) \geq mdv(G, v_f) \).

Thus, \( deg(G, v_1) \geq deg(H, v_1) \geq mdv(G, v_f) \). Since \( v_1 \) has the minimum degree in all non-\( v_f \) vertices in \( G \), any non-\( v_f \) vertex has degree at least \( deg(G, v_1) \). Since there are \( x - 1 \) non-\( v_f \) vertices in \( G \), the sum of degrees of all vertices in \( G \) is at least \( (x-1)deg(G, v_1) \). Thus, there are at least \( \frac{(x-1)deg(G, v_1)}{2} \) edges in \( G \); i.e., \( |E(G)| \geq \frac{(x-1)deg(G, v_1)}{2} \).

51
Hence, $d(G) = \frac{|E(G)|}{x} \geq \frac{(x-1)\deg(G,v_1)}{x} \geq \frac{mdv(G,v_f)}{2} \cdot \frac{x-1}{x}$. Since $x - 1 \geq 3$, $\frac{x-1}{x} \geq \frac{3}{4}$. Thus, $d(G) \geq \frac{mdv(G,v_f)}{2} \cdot \frac{3}{4}$. Hence, $\frac{mdv(G,v_f)}{d(G)} \leq \frac{8}{3}$.

So $mdv(G,v_f)/A(G) \leq mdv(G,v_f)/d(G) \leq \frac{8}{3}$.

Hence, the conclusion holds for $n = x$. By induction, the conclusion holds for all $n \geq 3$.

Thus, the greedy approximation algorithm for $DS(G,v_f)$ is of approximation ratio $\frac{8}{3}$.

In the following, we introduce the densest subgraph containing two vertices problem.

**Definition 5.5.6** For an undirected graph $G$ and two vertices $\{v_{f_1},v_{f_2}\}$ of $G$, we define the following quantities:

$$mdv(G,\{v_{f_1},v_{f_2}\}) := \text{the maximum density of an induced subgraph containing the two vertices } \{v_{f_1},v_{f_2}\}.$$ 

**Definition 5.5.7** The densest subgraph containing two vertices problem: given an undirected graph $G$ and two vertices $\{v_{f_1},v_{f_2}\}$ of $G$, find an induced subgraph containing $\{v_{f_1},v_{f_2}\}$ with density $mdv(G,\{v_{f_1},v_{f_2}\})$. This problem is denoted $DS(G,\{v_{f_1},v_{f_2}\})$.

If $G$ has $n$ vertices, it has $2^{n-2}$ vertex sets containing $\{v_{f_1},v_{f_2}\}$. Thus, one natural computational problem is whether $DS(G,\{v_{f_1},v_{f_2}\})$ is $P$ or $NP$-hard. In the following, we show that $DS(G,\{v_{f_1},v_{f_2}\})$ is $P$. We will design an LP-based polynomial time algorithm.

Extending $LP_1$, we design the following $LP_2$ formulation for the $DS(G,\{v_{f_1},v_{f_2}\})$ problem:

$$\begin{align*}
\text{maximize} & \quad \sum_{(v_i,v_j) \in E} x_{ij} \\
\text{subject to} & \quad 0 \leq x_{ij} \leq y_i, \quad 0 \leq x_{ij} \leq y_j \quad \text{for} \quad (v_i,v_j) \in E(G) \\
& \quad \sum_{i=1}^{n} y_i \leq 1 \\
& \quad 1 \geq y_{f_1} \geq y_i \quad \text{for} \quad i \notin \{f_1,f_2\} \\
& \quad 1 \geq y_{f_2} \geq y_i \quad \text{for} \quad i \notin \{f_1,f_2\} \\
& \quad y_{f_1} = y_{f_2}
\end{align*}$$

($LP_2$)

Based on $LP_2$, we can design a polynomial time algorithm for the $DS(G,\{v_{f_1},v_{f_2}\})$ problem similar to Algorithm 5.5.1. We also can show that the optimum value of $LP_2$ is $mdv(G,\{v_{f_1},v_{f_2}\})$. We omit the details.

Furthermore, we introduce the densest subgraph containing $k$ vertices problem.

**Definition 5.5.8** For an undirected graph $G$ and $k(k \geq 3)$ vertices $\{v_{f_1},v_{f_2},\ldots,v_{f_k}\}$ of $G$, we define the following quantities:

$$mdv(G,\{v_{f_1},v_{f_2},\ldots,v_{f_k}\}) := \text{the maximum density of an induced subgraph containing the } k \text{ vertices } \{v_{f_1},v_{f_2},\ldots,v_{f_k}\}.$$
Definition 5.5.9 The densest subgraph containing \( k \) vertices problem: given an undirected graph \( G \) and \( k \) vertices \( \{v_{f1}, v_{f2}, \ldots, v_{fk}\} \) of \( G \), find an induced subgraph containing \( \{v_{f1}, v_{f2}, \ldots, v_{fk}\} \) with density \( mdv(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \). This problem is denoted \( DS(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \).

One natural computational problem is whether \( DS(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \) is \( P \) or \( NP \)-hard. In the following, we show that \( DS(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \) is \( P \) by proposing an LP-based polynomial time algorithm.

Extending \( LP_2 \), we design the following LP formulation for the \( DS(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \) problem:

\[
\text{maximize } \sum_{(v_i, v_j) \in E} x_{ij} \quad (LP_3)
\]

subject to

1. \( 0 \leq x_{ij} \leq y_i, 0 \leq x_{ij} \leq y_j \) for \( (v_i, v_j) \in E(G) \) \hspace{1cm} (1)
2. \( \sum_{i=1}^{n} y_i \leq 1 \) \hspace{1cm} (2)
3. \( 1 \geq y_{f1} \geq y_i \) for \( i \not\in \{f1, f2, \ldots, fk\} \) \hspace{1cm} (3)
4. \( 1 \geq y_{f2} \geq y_i \) for \( i \not\in \{f1, f2, \ldots, fk\} \) \hspace{1cm} (4)
5. \( \vdots \)
6. \( 1 \geq y_{fk} \geq y_i \) for \( i \not\in \{f1, f2, \ldots, fk\} \) \hspace{1cm} (5)
7. \( y_{f1} = y_{f2} = \ldots = y_{fk} \) \hspace{1cm} (6)

Based on \( LP_3 \), we can design a polynomial time algorithm for \( DS(G, v_{f1}, \ldots, v_{fk}) \) problem similar to Algorithm 5.5.1. We also can show that the optimum value of \( LP_3 \) is \( mdv(G, v_{f1}, \ldots, v_{fk}) \). We omit the details.

Based on the polynomial time algorithm for \( DS(G, \{v_{f1}, v_{f2}, \ldots, v_{fk}\}) \), we design a polynomial time algorithm for \( DalkS \) when \( k \) is bounded by some constant \( c \).

\[\begin{align*}
\text{Input:} & \quad \text{A graph } G = (V, E) \text{ and an integer } k \ (k \text{ is bounded by some constant } c) \text{, where } |V| = n, |E| = m \\
\text{Output:} & \quad \text{An induced subgraph with at least } k \text{ vertices}
\end{align*}\]

1. Let \( S_1, \ldots, S_p \) be all vertex sets of \( k \) vertices, where \( p = \binom{n}{k} \)
2. Solve \( DS(G, S_i) \) for all \( i \leq i \leq p \) by the above LP-based polynomial time algorithm, and let \( H_i \) be the optimum solution
3. Compute the density \( d(H_i) \) for all \( i \)
4. \textbf{return} the subgraph \( H_j \) such that \( d(H_j) = \max_{1 \leq i \leq p} d(H_i) \)

Algorithm 5.5.2: The polynomial time algorithm for \( DalkS \) when \( k \) is bounded by some constant \( c \)

In the following, we show that \( d(H_j) = dal(G, k) \) and that it is a polynomial time algorithm when \( k \) is bounded by some constant \( c \).
Theorem 5.5.10 $d(H_j) = dal(G, k)$. When $k$ is bounded by some constant $c$, Algorithm 5.5.2 is a polynomial time algorithm.

Proof. Since $|H_j| \geq k$, $d(H_j) \leq dal(G, k)$. We suppose some $S \subseteq V$ and $|S| \geq k$ such that $d(S) = dal(G, k)$. Since $S$ has at least $k$ vertices and $\{S_1, \ldots, S_p\}$ contains every set of $k$ vertices, there exists some $S_h \in \{S_1, \ldots, S_p\}$ such that $S_h \subseteq S$. Let $H_h$ be an optimum solution for $DS(G, S_h)$. Thus, $d(S) \leq d(H_h) \leq d(H_j)$. So $d(H_j) = dal(G, k)$.

In Algorithm 5.5.2, the time of step (1) is $O(n^k)$. Theoretically, the best deterministic polynomial time algorithm is Karmarkar’s polynomial time algorithm for linear programming problems [65]. Its time complexity is $O(n^{3.5}L)$, where $n$ is the number of variables and $L$ is the number of bits in the input. Thus, the time of step (2) in Algorithm 5.5.2 for DalkS is also $O(n^k(n+m)^{3.5}L)$. Step (3) and step (4) can be finished in time $O(p)$, i.e $O(n^k)$. Thus, the total time of Algorithm 5.5.2 is $O(n^k) + O(n^k(n+m)^{3.5}L) + O(n^k)$, which is $O(n^d(n+m)^{3.5}L)$ when $k$ is bounded by some constant $c$. Hence, when $k$ is bounded by some constant $c$, Algorithm 5.5.2 is a polynomial time algorithm.

5.6 Approximation Algorithms for DamkS

In [4], Andersen studied the complexity of DamkS and showed that DamkS is $NP$-hard, but he didn’t study the algorithms for DamkS. In this section, we give two approximation algorithms for DamkS. The first approximation algorithm first solves the densest subgraph problem and then uses the idea of Charikar’s greedy algorithm [19] to obtain an approximate solution. The second approximation algorithm calls Feige et al.’s approximation algorithm for the dense $k$-subgraph problem [37] to find an approximation. We show that the first approximation algorithm has an approximation ratio of $\frac{n-1}{k-1}$ and the second approximation algorithm has a ratio of $O(n^\delta)$, for some $\delta < 1/3$.

In [100], Srivastav and Wolf have shown the following lemma:

Lemma 5.6.1 Let $S'$ be the subgraph obtained from a weighted graph $S$ by repeatedly removing a vertex with minimum weighted degree $w(v) = \Sigma_{vv' \in E(S)} w(vv')$. Then $w(S') \geq \frac{|S'|(|S'|-1)}{|S|(|S|-1)} w(S)$.
Thus, according to Lemma 5.6.1, we can get the following conclusion.

**Lemma 5.6.2** \( w(H_i) \geq \frac{i(i-1)}{m(m-1)} w(H) \) for all \( i \), where \( m = |H| \).

The detail of the first approximation algorithm for DamkS is described as follows:

```
Input: A weighted graph \( G = (V, E) \) of \( n \) vertices and an integer \( k \)
Output: An induced subgraph with at most \( k \) vertices
1 Find the induced subgraph \( H \) with maximum density by known algorithms
2 if \( |H| \leq k \) then
3 return \( H \)
4 else
5 \( m = |H| \)
6 \( H_m = H \)
7 Compute the density \( d(H_m) = \frac{\Sigma_{e \in H_m} w(e)}{m} \)
8 Compute the weight of each vertex, \( w(v) = \Sigma_{v v' \in E} w(v v') \)
9 for \( i = m \) to 1 do
10 Let \( v \) be the vertex in \( H_i \) with minimum weight
11 Let \( H_{i-1} \) be the graph of \( H_i \) with \( v \) removed
12 Update the weight of each neighbor \( v' \) of \( v \), \( w(v') = w(v') - w(v v') \) in \( H_{i-1} \)
13 Compute the density of \( H_{i-1} \), \( d(H_{i-1}) = \frac{(w(H_i) - w(v))}{(i - 1)} \)
14 end
15 return the subgraph \( H_j \) such that \( d(H_j) = \max_{1 \leq i \leq k} d(H_i) \)
```

**Algorithm 5.6.1**: The first DamkS approximation algorithm

In the following, we show the approximation ratio.

**Theorem 5.6.3** The first DamkS approximation algorithm has an approximation ratio of \( \frac{n-1}{k-1} \).

**Proof.** Let \( H \) be a subgraph of \( G \) with density \( d_{\text{max}}(G) \), let \( m = |H| \), and let \( H_m = H \). If \( m \leq k \), then Algorithm 5.6.1 will return \( H \), for an approximation ratio of \( 1 \leq \frac{n-1}{k-1} \). Otherwise \( m > k \), and we let \( H_j \) be the subgraph returned by Algorithm 5.6.1. By Lemma 5.6.2, \( \frac{d(H_m)}{d(H_k)} \leq \frac{m-1}{k-1} \). Since \( \text{dam}(G, k) \leq d_{\text{max}}(G) = d(H) \) and \( d(H_j) = \max_{1 \leq i \leq k} d(H_i) \), this implies that \( \frac{\text{dam}(G, k)}{d(H_j)} \leq \frac{d(H_m)}{d(H_k)} \leq \frac{d(H_m)}{d(H_k)} \leq \frac{n-1}{k-1} \). \( \blacksquare \)

The following approximation algorithm for DamkS can get the approximation ratio of \( O(n^\delta) \) for some \( \delta < 1/3 \): For \( i = 1, \ldots, k \), find the subgraph \( H_i \) with exactly \( i \) vertices by Feige et al’s approximation algorithm \([37]\). And then, output the subgraph \( H_j \) such that \( d(H_j) = \max_{1 \leq i \leq k} d(H_i) \). Since Feige et al’s approximation algorithm is of approximation ratio of \( O(n^\delta) \)
for some $\delta < 1/3$, it is trivial that the approximation algorithm for DamkS is also of the same approximation ratio.

5.7 Concluding Remarks

In this chapter, we have presented two polynomial time algorithms for the densest at-least-$k$-subgraph problem (DalkS) when $k$ is bounded by a constant. We have also provided two approximation algorithms for the densest at-most-$k$-subgraph problem (DamkS), one with an approximation ratio of $\frac{n-1}{k-1}$ and another with ratio $O(n^\delta)$ for some $\delta < 1/3$. The questions of whether there exist constant factor approximation algorithms for DamkS and the dense $k$-subgraph still remain open. Another open problem suggested by our work is whether DalkS is still solvable in polynomial time when $k$ is not bounded by a constant (particularly as $k$ approaches $\frac{n}{2}$).
Chapter 6

The Multiple Alignment Algorithm for Metabolic Pathways without Abstraction

6.1 Introduction

With the development of science and technology, several different types of biological data are produced, including genomic, proteomic, and pathway data. Metabolic pathway data is one of important types of data in biology and provides interaction information on three types of entities important to organisms [10]. By analyzing this data, people can get useful information. For example, the analysis of metabolic pathway data can provide important information for drug designing [99] and comparing genomes [16, 28].

The comparative analysis of pathways is one of the most important types of pathway analysis, and the comparative analysis of different metabolic pathways can provide insight in several biological applications. Network alignment is a comparative method of analysis that seeks to identify subnetworks that are conserved across two or more species, which implies that they represent true functional modules [92]. Network alignment methods can be used to compare many different types of networks, including protein interaction networks [71], regulatory networks [112], and metabolic networks [84]. The first and still most widely-used type of network for network alignment is the metabolic network. Aligning metabolic networks requires a method of determining similarity between parts of different networks, which has been done in many different ways. Similarity has been determined using the similarity of the enzymes that catalyze the metabolic reactions in the network [71, 87] and their EC numbers [103, 78, 106]. Similarity between the topological structure of the networks has also been used [87, 111]. Recently, similarity measure that combine many different measures of similarity have been developed
Metabolic network alignment algorithms use these similarity measures to identify conserved subnetworks, which likely correspond to conserved metabolic pathways.

The problem of computing the similarity of two pathways is referred to as the pairwise pathway alignment problem [10]. As stated in [10], this problem is related to the well-known graph isomorphism and subgraph isomorphism problems. The graph isomorphism problem is the problem of deciding if two graphs have the same topological structure, and the subgraph isomorphism problem is the problem of deciding if one graph is isomorphic to a subgraph of another. In computer science, the complexity of graph isomorphism is unknown, though the subgraph isomorphism problem is known to be NP-complete [43]. At present, there are no polynomial-time exact algorithms for the graph isomorphism and subgraph isomorphism problems, and as the pairwise alignment of pairwise can be reduced to these problems, existing algorithms for the pairwise pathway alignment problem rely on heuristic methods [104, 87, 10].

In [104], Tohsato et al. propose a pairwise alignment algorithm based solely on the similarities between compounds in the pathways, and Printer et al. [87] propose a pairwise alignment algorithm based only on the similarities between enzymes. These two algorithms only consider the similarities among one type of entity and neglect the other entities, as well as the topological structure of the pathways. This neglect can result in information loss, as explained by Ferhat Ay et al. [10], which can reduce accuracy in the results and limits the utility of these algorithms in the pairwise alignment problem [10]. To overcome these restrictions, Ferhat Ay et al. propose the first algorithm for the pairwise alignment of metabolic pathways based on the similarity of enzymes, compounds, reactions, and topological structure [10]. This technique improves accuracy and is not limited to pathways with topological restrictions. However, Ferhat Ay et al.’s algorithm can only be applied to two pathways. In many cases, we need to compute the similarity between multiple pathways.

The problem of computing the similarities between multiple pathways is referred to as the problem of multiple pathway alignment. Tohsato et al. proposed an algorithm to align multiple metabolic pathways based on the similarity of enzymes [103]. This algorithm, however, does not consider similarities in topology or other entities, such as compounds and reactions. Thus, their algorithm is less accurate. To overcome this shortcoming, we propose the first algorithm for the problem of aligning multiple metabolic pathways based on the similarities between enzymes, compounds, reactions, and topological structure.

A $k$-partite matching in a $k$-partite graph is a vertex partition that partitions the vertices into different equivalence classes. Given a weighted $k$-partite graph, the maximum-weighted $k$-partite matching problem is the problem of finding the $k$-partite matching with the maximum weight. In [96], the maximum-weighted $k$-partite matching problem has been used to compute an alignment across multiple protein-protein interaction networks. Our reduction from the problem of aligning multiple metabolic pathways to the maximum-weighted $k$-partite match-
Our Contributions:

In this chapter, we propose the first algorithm for aligning multiple metabolic pathways based on the similarity of enzymes, compounds, reactions, and topological structure. We use Ferhat Ay’s graph model to represent each metabolic pathway [10]. In order to compute the alignment of multiple pathways, we create a weighted $k$-partite graph for each type of entity and reduce the problem of aligning multiple metabolic pathways to the maximum-weighted $k$-partite matching problem.

Before we describe our algorithms, we introduce the maximum-weighted $(1, r)$-matching problem in a bipartite graph and give a polynomial time algorithm for this problem. Based on the algorithm for the maximum-weighted $(1, r)$-matching problem, we will give an approximation algorithm and a heuristic algorithm for the maximum-weighted $k$-partite matching problem.

Singh et al. have proposed a greedy algorithm for the maximum-weighted $k$-partite matching problem in [96]. We compare the accuracy of Singh et al.’s greedy algorithm with our approximation heuristic algorithms, and our experimental results show that our heuristic algorithm is the most accurate based on the matching weights.

We apply our heuristic algorithm to compute an alignment of multiple metabolic pathways. Our experiments show that our algorithm can correctly identify common subnetworks among multiple pathways, which can be used to find conserved entities.

6.2 The Algorithm for the Alignment Problem of Multiple Pathways

In [10], Ferhat Ay et al. describe a graph representation of metabolic pathways for computing the pairwise alignment of pathways. The advantage of Ferhat Ay et al.’s graph representation is that it exhibits no information loss and includes the topological structure of the pathways. If the graph representation exhibited information loss, the alignment results would be less accurate [10]. Thus, in order to compute the alignment of multiple pathways without topological restrictions, we use Ferhat Ay et al.’s graph model to represent metabolic pathways.

For clarity, we reproduce some of the notations from [10].

Notations in [10]: “Let $\mathcal{P}, \mathcal{R}, \mathcal{C}, \mathcal{E}$ denote the sets of all pathways, all reactions, all compounds, and all enzymes, respectively. Let $R \subseteq \mathcal{R}$ such that $R = \{R_1, R_2, \ldots, R_{|R|}\}$ denote the reactions. Let $C \subseteq \mathcal{C}$ such that $C = \{C_1, C_2, \ldots, C_{|C|}\}$ denote the compounds. Let $E \subseteq \mathcal{E}$ such that $E = \{E_1, E_2, \ldots, E_{|E|}\}$ denote the enzymes.”
We also reproduce the definition of the graph representation for a metabolic pathway from [10].

**Definition 6.2.1** A directed graph, $G(V, I)$ for representing the metabolic pathway $P \in \mathcal{P}$, is constructed as follows [10]: The node set, $V = [R, C, E]$, is the union of reactions, compounds and enzymes of $P$. The edge set, $I$, is the set of interactions between the nodes. An interaction is represented by a directed edge that is drawn from a node $x$ to another node $y$ if and only if one of the following three conditions holds:

1) $x$ is an enzyme that catalyzes reaction $y$.
2) $x$ is an input compound of reaction $y$.
3) $x$ is a reaction that produces compound $y$.

Below, we give the definition of an alignment of multiple pathways, which is a generalization of the pairwise alignment of pathways described in [10].

**Definition 6.2.2** An alignment of $k$ metabolic pathways $P_1 = G([R_1, C_1, E_1], I_1), \ldots, P_k = G([R_k, C_k, E_k], I_k)$ is a partition over the node set $V = V_1 \cup \ldots \cup V_k$ that divides $V$ into disjoint equivalence classes. The nodes in each equivalence classes are mapped to each other and are of the same type of entity (i.e., compounds are mapped to compounds, enzymes to enzymes, and reactions to reactions).

To evaluate the degree of similarity between two alignments across multiple pathways, we define an alignment score based on compounds and enzymes, a generalization of the similarity score of pairwise pathways defined in [10]. Since the similarity score of two reactions is computed from the similarity scores of compounds and enzymes [10], the similarity scores of reactions are not included in the alignment score.

**Definition 6.2.3** Suppose $P_1 = G([R_1, C_1, E_1], I_1), \ldots, P_k = G([R_k, C_k, E_k], I_k)$ are $k$ pathways. Let $\varphi = [\varphi_R, \varphi_C, \varphi_E]$ be an alignment, where $\varphi_R, \varphi_C, \varphi_E$ are maps between the reactions, compounds, and enzymes, respectively, of $P_1, \ldots, P_k$. The alignment score of $P_1, \ldots, P_k$ is:

$$Score_\varphi(P_1, \ldots, P_k) = \frac{\beta}{|\varphi_C|} \sum_{(C_i, C_j) \in \varphi_C} SimC(C_i, C_j) + \frac{1-\beta}{|\varphi_E|} \sum_{(E_i, E_j) \in \varphi_E} SimE(E_i, E_j).$$

where $0 \leq \beta \leq 1$ is a parameter that measures the relative importance of compounds and enzymes, $|\varphi_C|$ is the number of compound mappings, and $|\varphi_E|$ is the number of enzyme mappings. $SimC(C_i, C_j)$ denotes the similarity score of $C_i$ and $C_j$, and $SimE(E_i, E_j)$ denotes the similarity score of $E_i$ and $E_j$. Different methods for calculating the similarity scores of two
compounds and two enzymes are discussed in Section 4.1 of [10]. Two enzyme similarity scoring methods are *Hierarchical enzyme similarity score* [103] and *Information content enzyme similarity score* [87], and two compound similarity scoring methods are *trivial compound similarity score* and *SIMCOMP compound similarity score* [56].

The alignment problem of multiple pathways is defined as follows:

**Definition 6.2.4** The problem of aligning the *k* pathways $P_1, \ldots, P_k$ is the problem of finding an alignment $\varphi$ of $P_1, \ldots, P_k$ with the maximum alignment score.

The maximum-weighted *k*-partite matching problem is introduced in [96], and a greedy algorithm is used to compute the global alignment of multiple protein-protein interaction networks. Similarly, we will use the solution produced by our heuristic algorithm to compute the mappings between the compounds, enzymes, and reactions of multiple pathways. In [96], there is an informal description for the maximum-weighted *k*-partite matching problem. We give a formal definition for the maximum-weighted *k*-partite matching problem.

**Definition 6.2.5** Suppose $G = (V, E, w)$ is a weighted *k*-partite graph whose *k* parts are $U_1, \ldots, U_k$. A *k*-partite matching $M$ of $G$ is a vertex partition that divides $V$ into $V_1, \ldots, V_h$ such that $V = V_1 \cup \ldots \cup V_h$ and $V_i$ contains at most $r$ vertices from $U_j$ for any $i, j$, where $r$ is a positive integer. The weight of $M$ is $\sum_{i=1}^{h} W(G(V_i))$, where $W(G(V_i)) = \sum_{e \in E(G(V_i))} w(e)$ is the weight of the subgraph $G(V_i)$ induced by $V_i$, $E(G(V_i))$ is the set of edges in $G(V_i)$, and $w(e)$ is the weight of edge $e$. When $k = 2$, this matching is called an $(r, r)$-matching.

**Definition 6.2.6** Given a weighted *k*-partite graph, the maximum-weighted *k*-partite matching problem is the problem of finding a *k*-partite matching with the maximum weight.

In the following, we give an algorithm for the alignment of multiple pathways without topological restrictions (see Algorithm 7.5.1).
Input: $k$ metabolic pathways $P_1 = G([R_1, C_1, E_1], I_1), \ldots, P_k = G([R_k, C_k, E_k], I_k)$.

Output: A global alignment of $P_1, \ldots, P_k$.

1. For each pair of pathways $P_i$ and $P_j$ ($i \neq j$), we compute a weight for each pair of compounds, each pair of reactions, and each pair of enzymes in $P_i$ and $P_j$ by the method in [10]. This method considers the similarity of the entities as well as the topology.

2. Based on these weights, we construct weighted $k$-partite graphs $H_R, H_E,$ and $H_C$ for the reactions, enzymes, and compounds, respectively. $H_R$ consists of $k$ partite sets of reactions nodes: each partite set of reaction nodes comes from one pathway. Weighted edges are constructed as follows. If $r_i$ comes from $P_i$ and $r_j$ comes from $P_j$, and the weight between $r_i$ and $r_j$ is $w_{ij}$, then there is an edge $E(r_i, r_j)$ between $r_i$ and $r_j$ iff $w_{ij} > 0$. The weight of $E(r_i, r_j)$ is $w_{ij}$. $H_E$ and $H_C$ are constructed similarly.

3. We compute approximation solutions $M_R, M_E,$ and $M_C$ for the maximum-weighted $k$-partite matching of $H_R, H_E,$ and $H_C,$ respectively, by a heuristic algorithm from the following sections.

4. Based on $M_R$ and reachability concept, we can determine if two enzymes (or compounds) are consistent. From $M_E$ and $M_C$, by pruning those non-consistent enzymes and compounds, we get $M_E'$ and $M_C'$.

5. These $M_R, M_E'$, and $M_C'$ give the consistent mapping of reactions, enzymes, and compounds, respectively, and define an consistent alignment of $P_1, \ldots, P_k$.

Algorithm 6.2.1: The Multiple Pathway Alignment Algorithm

This algorithm is a modification of Singh et al.’s algorithm for aligning multiple protein-protein interaction networks in [96]. Our novelty is as follows: we create three $k$-partite graphs, and we propose an approximation algorithm and a heuristic algorithm for the maximum-weighted $k$-partite matching problem in the following sections since the maximum weight $k$-partite matching is NP-hard [96]. In order to enforce the consistent mapping, we first get the mapping of reactions. Then, we remove those non-consistent enzymes and compounds based on the mapping of reactions and reachability concept.[The consistent alignment concept is introduced in [10].]

6.3 A Polynomial Time Algorithm for the Maximum-Weighted $(1,r)$-Matching Problem in a Bipartite Graph

In a weighted bipartite graph, a maximum-weighted matching is a matching with maximum weight. The maximum-weighted bipartite matching problem is the problem of finding the maximum-weighted matching in a weighted bipartite graph, which is a well known combinatorial optimization problem in computer science. In order to give the new algorithms for the
maximum-weighted $k$-partite matching problem, we generalize the maximum-weighted matching problem in bipartite graphs to the maximum-weighted $(1, r)$-matching problem in bipartite graphs.

In the following, we introduce the definition of the maximum-weighted $(1, r)$-matching in a bipartite graph.

**Definition 6.3.1** Let $G = (L, R)$ be a weighted bipartite graph. A $(1, r)$-matching is a vertex partition of $V$ that partitions $V$ into $V_1, \ldots, V_h$ such that $|V_i \cap L| \leq 1$ and $|V_i \cap R| \leq r$ for every $V_i$. The maximum $(1, r)$-matching problem is the problem of finding the $(1, r)$-matching with the maximum weight.

In the following, we give a polynomial-time algorithm to compute the maximum-weighted $(1, r)$-matching in a bipartite graph.

**Algorithm 6.3.1**: A polynomial algorithm $A_1$ for the maximum-weighted $(1, r)$-matching in a bipartite graph.

| Input: | A weighted bipartite graph $G = (L, R)$, where $|L| = m$ and $|R| = n$. |
| Output: | A $(1, r)$-matching of maximum weight. |
| 1 | Let $v_1, \ldots, v_{m_1}$ be the vertices in $L$. Let $d(v_i)$ denote the degree of $v_i$ for all $i$. |
| 2 | Construct $G' = (L', R')$ from $G$ as follows. For any $v_i \in L$, if $d(v_i) \geq r$, then we place $r$ copies $v_i_1, \ldots, v_i_r$ of $v_i$ into $G'$. If $d(v_i) \leq r$, then we put $d(v_i)$ copies $v_i_1, \ldots, v_i_{d(v_i)}$ of $v_i$ in $G'$, where vertex $u$ being a copy of vertex $v$ means that $(v, u') \in E(G)$ iff $(v, u') \in E(G')$. |
| 3 | for any vertex $u'$. Let $R' = R$. |
| 4 | Find the maximum-weighted matching $M'$ in $G'$ by using the known algorithms, such as the algorithm in [41]. (More references can be found in [41].) |
| 5 | Compute the $(1, r)$-matching $M$ from $M'$: for any $v_i$, if $(v_{ij_i}, w_{i1}) \in M', \ldots, (v_{ij_i}, w_{ih}) \in M'$, then $v_i$ is matched to $\{w_{i1}, \ldots, w_{ih}\}$. |

In the following, we show that above algorithm can find a maximum-weighted $(1, r)$-matching in polynomial time.

**Theorem 6.3.2** Algorithm 7.5.2 can find a maximum-weighted $(1, r)$-matching in $O(mn + n^2 \log n)$-time.

**Proof.** Steps 1 and 2 can be finished in time $O(m)$. Step 3 can be finished in time $O(mn + n^2 \log n)$ [41]. Thus, the algorithm 7.5.2 can be finished in $O(mn + n^2 \log n)$-time.

First, we show that $M$ in algorithm $A_1$ is a $(1, r)$-matching. Suppose that $v_i$ is matched to $\{w_{i1}, \ldots, w_{ih}\}$ and that $v_j$ is matched to $\{w_{j1}, \ldots, w_{j\ell}\}$. Let $M'$ be the maximum matching
found in $G'$ at step 3. From step 4, $(v_i, w_{ip}) \in M'$ and $(v_j, w_{jq}) \in M'$ for every $p \in \{1, \ldots, h\}$ and $q \in \{1, \ldots, \ell\}$. Since $M'$ is a matching in $G'$, $w_{ip} \neq w_{jq}$ for any $p \in \{1, \ldots, h\}$ and $q \in \{1, \ldots, \ell\}$. So $\{w_{i1}, \ldots, w_{ih}\} \cap \{w_{j1}, \ldots, w_{j\ell}\} = \emptyset$. Since there are at most $r$ copies of any vertex $v_i$, $h \leq r$. Thus, $M$ is a $(1, r)$-matching.

Second, we show that the weight $w(M)$ of $M$ is the maximum. Let $M_2$ be any $(1, r)$-matching in $G$. We can construct a matching $M'$ in $G'$ as follows: if $v_i$ is matched to $\{w_{i1}, \ldots, w_{ih}\}$ in $M_2$, then $(v_{ij}, w_{i1}) \in M'_2, \ldots, (v_{ij}, w_{ih}) \in M'_2$. Let $M'$ be the maximum matching found in $G'$ at step 3. Since $M'$ is a maximum matching in $G'$, $w(M') \geq w(M'_2)$. Since $w(M) = w(M')$ and $w(M_2) = w(M'_2)$, $w(M) \geq w(M_2)$, so the weight $w(M)$ of $M$ is the maximum out of all $(1, r)$-matchings. ■

### 6.4 An Approximation Algorithm for the Maximum-Weighted $k$-partite Matching Problem

The maximum-weighted $k$-partite matching problem is introduced in [96] and is used to compute the alignment of multiple protein-protein interaction networks. Each of these protein-protein interaction (PPI) network is modeled as a weighted graph, and the problem of aligning the PPI networks is reduced to the maximum-weighted $k$-partite matching problem. Singh et al. give a greedy heuristic algorithm for the maximum-weighted $k$-partite matching problem. In this section, we describe the first approximation algorithm for the maximum-weighted $k$-partite matching problem. Our approximation algorithm is a generalization of the approximation algorithm for the maximum disjoint $k$-clique problem in [58], which is a special case of maximum-weighted $k$-partite matching problem. The approximation algorithm given in [58] is based on the maximum-weighted matching in a bipartite graph and the idea of merging matched vertices. Our approximation algorithm for the maximum-weighted $k$-partite matching problem is based on the maximum-weighted $(1, r)$-matching in a bipartite graph and the idea of merging matched vertices.

First, we extend the definition of the merging operation in [58] in order to merge $(1, r)$-matching operation. For clarity, we reproduce some notations from [58] in the following definition. In our definition, the matched vertices are merged into a single node, whereas in [58], an edge is merged into a node.

**Definition 6.4.1** Suppose $G = (V, E, w)$ is a weighted $k$-partite graph with $k$ parts: $U_1, \ldots, U_k$. Let $G_{i,j} = G[U_i \cup U_j]$ be the subgraph induced by $U_i \cup U_j$, and let $M_{i,j}$ be a $(1, r)$-matching of $G_{i,j}$. For our definition, if $v$ is matched to $\{u_1, \ldots, u_r\}$ in $M_{i,j}$, then we merge $v, u_1, \ldots, u_r$ into a new vertex $u$. Let $U_{i,j}$ denote the new vertex set of $G$. For each $v' \in V$, we define edge weights to the merged vertex using $w(u, v') = w(v, v') + \sum_{i=1}^{r} w(u_i, v')$, and the loops and
edges between vertices \(v, u_1, \ldots, u_r\) are removed. Let \((G_{i(j)}, w')\) denote the new weighted graph after merging \(M_{i,j}\). Note that \((G_{i(j)}, w')\) is an edge-weighted \((k-1)\)-partite graph.

We now propose an approximation algorithm for the maximum-weighted \(k\)-partite matching problem (see Algorithm 6.4.1) that is the generalization of the approximation algorithm for the maximum disjoint \(k\)-clique problem in [58]. In order to explain our algorithm clearly, we present our algorithm in a modified format from the description of the approximation algorithm in [58].

| Input: A weighted \(k\)-partite graph \(G = (U_1, \ldots, U_k)\). Let \(|U_1| = c = \max\{|U_1|, |U_2|, \ldots, |U_k|\}.\) |
|-----------------------------------------------|
| Output: A \(k\)-partite matching. |
| 1 Let \(i = k\) and \(G' = G\). |
| 2 while \(i \neq 2\) do |
| 3 Let \(G'_{i,i} = G'[U_1 \cup U_i].\) |
| 4 Find a maximum-weighted \((1, r)\)-matching \(M_i\) of \(G'_{i,i}\) by Algorithm 7.5.2. Suppose that \(M_i\) matches \(v_1\) to the vertex set \(S_{1,i}\), \(v_2\) to the vertex set \(S_{2,i}\), etc. |
| 5 Merge \(M_i\) in \(G'\) to produce the \((i-1)\)-partite graph \(G'_{1(i)}\). |
| 6 Let \(i = i - 1\) and \(G' = G'_{1(i)}\). |
| 7 end |
| 8 Find a maximum-weighted \((1, r)\)-matching \(M_2\) of \(G'\) by Algorithm 7.5.2. |
| 9 Let \(V_1 = v_1 \cup S_{1}^{2} \cup S_{1}^{3} \cup \ldots \cup S_{1}^{k_1}, \ldots, V_c = v_c \cup S_{c}^{2} \cup S_{c}^{3} \cup \ldots \cup S_{c}^{k_c}.\) |
| 10 Output \(M = \{V_1, \ldots, V_c\}\). |

Algorithm 6.4.1: The approximation algorithm A2 for the maximum-weighted \(k\)-partite matching problem.

The idea behind Algorithm 6.4.1 is as follows. First, we find the maximum-weighted \((1, r)\)-matching between \(U_1\) and \(U_2\) and merge \(U_1\) and \(U_2\) into one graph, turning the \(k\)-partite graph into a \((k-1)\)-partite graph. This process is repeated until the original \(k\)-partite graph becomes the bipartite graph \(G'\). Finally, we find the maximum-weighted \((1, r)\)-matching in \(G'\) and return the associated \(k\)-partite matching.

Let \(c = \max\{|U_1|, |U_2|, \ldots, |U_k|\}\). Step 4 in algorithm 6.4.1 can be finished in time \(O(c^3)\) by Theorem 3.2, and since the while loop at step 2 can iterate at most \(k\) times, the total time for Algorithm 6.4.1 is at most \(O(kc^3)\).

In the following, we prove an approximation ratio for our algorithm.

**Theorem 6.4.2** Suppose \(G\) is a weighted complete bipartite graph where each part has \(c\) vertices. Let \(M\) be the solution output by Algorithm 6.4.1. It must be the case that \(\frac{w(G)}{w(M)} \leq c\).
Suppose \( k \) is simple to prove that every disjoint perfect matchings by induction on \( k \).

\( w \) and let \( L'' = L' \), and add \((c - 1)r \) new vertices to the set \( R' \) to form \( R'' \). We add an edge of weight 0 from every new vertex to every vertex in \( L'' \), and keep all of the edges in \( G' \). Thus, for every matching \( M'' \) in \( G'' \), \( M'' \cap G' \) is a matching in \( G' \) and \( w(M'') = w(M'' \cap G') \). Since \( G'' \) is complete bipartite graph with \( cr \) vertices in each part, it can be easily decomposed into the \( cr \) edge-disjoint perfect matching \( M''_1, \ldots, M''_{cr} \).

(\( It \) is simple to prove that every \( k \)-regular bipartite graph can be decomposed into \( k \) edge-disjoint perfect matchings by induction on \( k \). Please see exercise 3.3.4 in the book [107].) Thus, \( w(M'') + \ldots + w(M''_{cr}) = w(G'') \), so \( w(M''_1 \cap G') + \ldots + w(M''_{cr} \cap G') = w(G'') \). Since \( w(M''_i \cap G') \leq w(M') \) for any \( i \), \( cr \cdot w(M') \geq w(G'') \). As \( w(G'') = w(G') = r \cdot w(G) \) and \( w(M') = w(M), cr \cdot w(M) \geq r \cdot w(G) \), proving that \( \frac{w(G)}{w(M)} \leq c \).

Based on the proof of Theorem 7.6.1, we show that the conclusion of Theorem 2 in [58] holds for our algorithm A2 by induction.

**Theorem 6.4.3** Suppose \( G \) is a weighted complete \( k \)-partite graph where each part has \( c \) vertices. Let \( M \) be the solution output by Algorithm A2. It must be the case that \( \frac{w(G)}{w(M)} \leq c \).

**Proof.** We use induction on \( k \) to prove the claim. First, the conclusion holds for \( k = 2 \) by Theorem 7.6.1. So, suppose that the claim holds for \( k - 1 \). (We will establish that the claim also holds for \( k \).)

In step 3 in the first while loop in Algorithm A2, \( G_{1,k} = G[U_1, U_k] \) is generated, and in step 4, a maximum-weighted \((1, r)\)-matching \( M_k \) of \( G_{1,k} \) is found. By Theorem 7.6.1, \( w(G_{1,k}) \leq c \cdot w(M_k) \). After merging \( M_k \) in \( G^k \), we get a \((k - 1)\)-partite graph \( G^k_{1(k)} \). Suppose that \( M' \) is the approximate solution obtained by applying Algorithm A2 to \( G^k_{1(k)} \). By the induction hypothesis, we know that \( w(G^{k}_{1(k)}) \leq c \cdot w(M') \), so \( w(G) = w(G_{1,k}) + w(G^{k}_{1(k)}) \leq c \cdot (w(M_k) + w(M')) = c \cdot w(M) \) by step 9 in Algorithm 6.4.1.

**Corollary 6.4.4** Suppose \( G \) is a weighted complete \( k \)-partite graph where each part has \( c \) vertices. Algorithm A2 has an approximation ratio of \( c \).

**Proof.** Let \( M^* \) be the maximum-weighted \( k \)-partite matching in \( G \). So, \( \frac{w(M^*)}{w(M)} \leq \frac{w(G)}{w(M)} \leq c \).

**Corollary 6.4.5** Suppose \( G \) is a weighted complete \( k \)-partite graph whose \( k \) parts are \( U_1, \ldots, U_k \), and let \( c = \max\{|U_1|, |U_2|, \ldots, |U_k|\} \). The algorithm A2 has an approximation ratio of \( c \).

**Proof.** From \( G \), we construct a complete \( k \)-partite graph \( G' \) with \( c \) vertices in each part by adding new vertices in each part and new edges of weight 0. Let \( M' \) be the matching
produced by applying Algorithm A2 to $G'$ and $M = M' \cap G$. Since the new edges weights are 0, $w(M') = w(M)$, so $\frac{w(G)}{w(M)} \leq \frac{w(G')}{w(M')} \leq c$. Thus, $\frac{w(M^*)}{w(M)} \leq \frac{w(G)}{w(M)} \leq c$. ■

6.5 A Heuristic Algorithm for the Maximum-weighted $k$-partite Matching Problem

In this section, we propose a heuristic algorithm for the maximum-weighted $k$-partite matching problem. Note that each matched node set in the solution $M$ produced by Algorithm 6.4.1 contains at most one node from the first part $U_1$ and at most $r$ nodes from other parts, but each matched node set of a $k$-partite matching can contain up to $r$ nodes from each part. In order to allow each matched node set of the approximation to also contain up to $r$ nodes from $U_1$, we propose a novel heuristic algorithm.

Our heuristic algorithm idea is as follows. We first find an $(r, r)$-matching $M$ for the subgraph of two parts by the greedy algorithm in [96] and then perform a merge based on $M$ to get a $(k - 1)$-partite graph $G'$. (Since the definition of merging a $(r, r)$ matching is very similar to the definition of merging a $(1, r)$ matching, we omit it here.) Next, we find a matching $M'$ for the $(k - 1)$-partite graph $G'$ using our approximation algorithm, and the matchings $M$ and $M'$ are combined into an approximate solution.

The heuristic algorithm for the maximum-weighted $k$-partite matching problem appears in Algorithm 6.5.1. In order to explain our algorithm clearly, we reproduce some formatting of the approximation algorithm for the maximum disjoint $k$-clique problem given in [58].
Input: A weighted \(k\)-partite graph \(G = (U_1, \ldots, U_k)\). Suppose 
\[|U_1| = c = \max\{|U_1|, |U_2|, \ldots, |U_k|\}.\]
Output: A \(k\)-partite matching.

1. Let \(G^k = G\).
2. Let \(G_{1,k}^k = G^k[U_1 \cup U_k]\).
3. Find an approximate solution \(M_k\) of the maximum-weighted \((r, r)\)-matching of \(G_{1,k}^k\) by the greedy algorithm in [96]. Suppose that in \(M_k\), \(S_1^1\) is matched to a vertex set \(S_2^k\), \(S_2^1\) is matched to \(S_2^2\), etc.
4. Merge \(M_k\) in \(G^k\) to get the \((k - 1)\)-partite graph \(G_{1(k)}^k\).
5. Let \(i = k - 1\) and \(G^i = G_{1(k)}^k\).
6. while \(i \neq 2\) do
7.   Let \(G_{i,i}^i = G^i[U_1 \cup U_i]\).
8.   Find a maximum-weighted \((1, r)\)-matching \(M_i\) of \(G_{i,i}^i\). Suppose that in \(M_i\), \(v_1\) is matched to the vertex set \(S_1^i\), \(v_2\) is matched to \(S_2^i\), etc.
9.   Merge \(M_i\) in \(G^i\) to get the \((i - 1)\)-partite graph \(G_{1(i)}^i\).
10. Let \(i = i - 1\) and \(G^i = G_{1(i)}^i\).
11. end
12. Find a maximum-weighted \((1, r)\)-matching \(M_2\) of \(G^2\).
13. Let \(V_1 = v_1 \cup S_1^1 \cup S_1^2 \cup \ldots \cup S_{1}^{k_1}, \ldots, V_c = v_c \cup S_c^1 \cup S_c^2 \cup \ldots \cup S_{c}^{k_c}\).
14. Output \(M = \{V_1 \ldots V_c\}\).

Algorithm 6.5.1: The heuristic algorithm \(A3\) for the maximum \(k\)-partite matching problem.

6.6 Comparison of Results for Several Algorithms

In this section, we compare our approximation algorithm \(A2\), our heuristic algorithm \(A3\), and the greedy algorithm in [96] on the basis of the sum of the weights of the edges in the resulting matchings. In our experiments, we build a \(k\)-partite graph by generating random edge weights \(0 \leq w \leq 1\). We ran each algorithm on 100 random \(k\)-partite graphs with 20 vertices in each part. In the following table, we give the mean value and the standard deviation for the sum of the edge weights for the 100 matchings produced by the three algorithms, for \(k\) between 3 and 8. We choose \(r = 5\) for our algorithms as well as the greedy algorithm in [96]. We have considered other values for \(r\), but due to space limitations, we only list the results for \(r = 5\).
Table 6.1: The mean value and standard deviation for weights of matchings in 3-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>145.79</td>
<td>7.25</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>36.06</td>
<td>12.21</td>
<td>0.000005</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>165.82</td>
<td>5.07</td>
<td>0.000002</td>
</tr>
</tbody>
</table>

Table 6.2: The mean value and standard deviation for weights of matchings in 4-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>282.42</td>
<td>11.07</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>108.42</td>
<td>24.3</td>
<td>0.000008</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>336.00</td>
<td>7.58</td>
<td>0.000005</td>
</tr>
</tbody>
</table>

Table 6.3: The mean value and standard deviation for weights of matchings in 5-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>459.34</td>
<td>15.90</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>227.06</td>
<td>33.11</td>
<td>0.000013</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>557.02</td>
<td>10.57</td>
<td>0.000008</td>
</tr>
</tbody>
</table>
Table 6.4: The mean value and standard deviation for weights of matchings in 6-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time(sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>662.25</td>
<td>20.83</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>407.90</td>
<td>42.65</td>
<td>0.000015</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>829.99</td>
<td>12.20</td>
<td>0.000013</td>
</tr>
</tbody>
</table>

Table 6.5: The mean value and standard deviation for weights of matchings in 7-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time(sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>892.81</td>
<td>29.74</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>632.44</td>
<td>57.12</td>
<td>0.000151</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>1157.13</td>
<td>13.47</td>
<td>0.000014</td>
</tr>
</tbody>
</table>

Table 6.6: The mean value and standard deviation for weights of matchings in 8-partite graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
<th>Mean Time(sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Algorithm</td>
<td>1139.60</td>
<td>42.65</td>
<td>0.000001</td>
</tr>
<tr>
<td>Algorithm A2</td>
<td>913.36</td>
<td>77.73</td>
<td>0.000203</td>
</tr>
<tr>
<td>Algorithm A3</td>
<td>1537.18</td>
<td>16.91</td>
<td>0.000171</td>
</tr>
</tbody>
</table>

From these tables (Table 6.1—Table 6.6), we conclude that Algorithm A3 produces better matchings than Algorithm A2, which finds better matchings than the Greedy Algorithm. We believe that this is because Algorithm A3 considers the possibility that each matched node set of a k-partite matching can contain up to r nodes from the first part (in step 3). Elsewhere in
Algorithm $A_3$, an $(1, r)$-matching is computed exactly, and each matched node set of a $k$-partite matching is allowed to contain up to $r$ nodes from other parts.

6.7 Alignment Results of Metabolic Pathways

From the above sections, we concluded Algorithm $A_3$ is the best. So we apply Algorithm 7.5.1, using Algorithm $A_3$ in step 3, to metabolic data from the KEGG database in order to compute the alignment of multiple metabolic pathways. We used the hierarchical enzyme similarity score [103] to calculate the similarity between enzymes and trivial compound similarity score [56] for compounds. Other algorithmic parameters are as follows: $\alpha = 0.7$, $\beta = 0.5$, $\gamma_{C_{in}} = 0.3$, $\gamma_{C_{out}} = 0.3$, and $\gamma_E = 0.4$.

6.7.1 Biological Significance

Aligning metabolic pathways of multiple phenotype-related microorganisms enables identification of conserved metabolic components, such as compounds, enzymes, or reactions. By analyzing these conserved metabolic components, scientists may gain new insights into metabolic processes, structural information, and evolutionary relationships (for instance, by identifying homologous proteins) [78, 34]. In particular, biological engineers are interested in using this data to identify conserved metabolic pathways related to the production of specific microbial products, such as biological hydrogen.

To evaluate the performance of our algorithm, we aligned the well-characterized tricarboxylic acid (TCA) cycle across five aerobic microorganisms and compared the results with known microbial physiological data. We also aligned the TCA pathway in KEGG across five facultative anaerobes.

TCA-related enzymes

We aligned the tricarboxylic acid (TCA) cycle pathway (KEGG pathway 00020) across the five aerobic organisms *Bordetella bronchiseptica* RB50 (*bbr*), *Staphylococcus saprophyticus subsp. saprophyticus* ATCC 15305 (*ssp*), *Myxococcus xanthus* DK 1622 (*mxa*), *Leptospira interrogans* serovar Lai str. 56601 (*lil*), and *Helicobacter pylori* HPAG1 (*hpa*). The resulting alignments for the TCA enzymes appear in Table 6.7. The parts of resulting alignment for the TCA reactions and compounds are depicted in Table 6.8.

The alignment results generated for the TCA cycle pathway demonstrate the ability of our algorithm to identify key enzymes associated with phenotype-related metabolic pathways. In our validation experiment, all TCA enzymes were correctly aligned in the four aligned microorganisms that have complete TCA cycles (*bbr, ssp, mxa*, and *lil*). While the presence of the TCA
cycle is considered an indicator of aerobic respiration, it is not complete in all aerobic species [108]. The TCA enzymes succinate thiokinase and malate dehydrogenase were not aligned in *H. pylori*, as these two enzymes are missing from *hpa* in the KEGG data. This absence may be due to incomplete data in KEGG or the lack of a complete TCA cycle in *hpa*. In this case, studies have indicated that only parts of the TCA cycle are active in *H. pylori*, and that the active components are responsible for providing necessary biosynthetic components [18].

**Facultative respiration**

To further understand the sub-paths involved in central metabolism, we used the KEGG TCA metabolic pathway to align the five facultative organisms *Escherichia coli* (*eco*), *Shewanella oneidensis* MR-1 (*son*), *Zymomonas mobilis* (*zmo*), *Listeria innocua* (*lin*), and *Streptococcus pneumonia* (*spn*). These five organisms were selected for their potential use in industrial environmental technologies. The enzyme alignment results appear in Table 6.9. Here, we omit the reaction and compound alignment results.

Alignment results for these species reveal that only two bacterial species, *E. coli* and *S. oneidensis* contain complete TCA cycles, consistent with the ability of these organisms to thrive in a number of aerobic habitats, such as aquatic systems. As such, one would expect these organisms to use an oxidative pathway like the TCA cycle to break down simple organic matter (e.g., glucose) and generate the ATP metabolites necessary for growth [108].

Results for the alignment also indicated that the TCA cycle was incomplete in *Z. mobilis* and *L. innocua* due to the absence of two important TCA enzymes, malate dehydrogenase (MDH) and α-ketoglutarate dehydrogenase (KGDH). Unlike *E. coli* and *S. oneidensis*, these bacterial species tend to thrive in low oxygen environments, so complete TCA cycles are not necessarily optimal for them. While these organisms can utilize some TCA reactions to produce biosynthetic precursors, the absence of MDH and KGDH may play an important role in regulating overall cell growth and energy requirements.

In the case of *Z. mobilis*, a recent genomic study by Seo et al. found that *Z. mobilis* ZM4 does not contain genes that encode these two enzymes, and the lack of these enzymes is likely responsible for low cell biomass yields compared to other organisms with complete TCA cycles. In addition, Seo et al. noted that *Z. mobilis* relied on the Entner-Doudoroff (ED) pathway to metabolize simple sugars such as glucose and fructose. As such, it incorporates high quantities of glucose for the production of end-products such as ethanol, and without a complete TCA cycle, most of this energy is directed to ethanol production rather than cellular growth [91].

Similar to *Z. mobilis*, we found that *L. innocua* did not contain the two TCA enzymes KGDH and MDH. *L. innocua* is a non-pathogenic bacterium commonly found in aquatic and terrestrial environments [69]. Our findings are consistent with comparative genomic studies by Karlin et al. [69] that found *L. innocua* and other *Listeria* species to contain incomplete
TCA cycles. The advantages of having an incomplete cycle are not well understood at this time, but further alignments of metabolic pathways involved in glycolysis, pentose phosphate metabolism, and the reductive TCA cycle may shed light onto these advantages, or even lead to alternative routes for the production of biosynthetic precursors like succinyl-CoA.

6.8 Conclusion

In this Chapter, we have considered the multiple alignment problem for metabolic pathways. We’ve developed two algorithms for aligning multiple pathways based on similarities between compounds, reactions, enzymes, and topological structure. We’ve also reduced the problem of aligning multiple metabolic pathways to the maximum-weighted \( k \)-partite matching problem.

For the maximum-weighted \( k \)-partite matching problem, we presented an approximation and a heuristic algorithm. In order to design our algorithms, we introduced the maximum-weighted \((1, r)\)-matching problem and gave a polynomial-time algorithm is given for solving this problem, and we designed our \( k \)-partite matching algorithms based on an algorithm for solving the maximum-weighted \((1, r)\)-matching problem. Singh et al. proposed a greedy algorithm for the maximum-weighted \( k \)-partite matching problem in [96], and we compare this algorithm with the ones proposed in this paper. Experimental results show that our heuristic algorithm is the most accurate.

Finally, we applied our heuristic algorithm to compute the alignment of multiple metabolic pathways, and our experimental results verify that our algorithm correctly identifies the common subnetworks of multiple pathways. Moreover, alignments identified by our algorithm may help guide scientists to find novel and promising metabolic capabilities for biological engineering or industrial purposes.
Table 6.7: Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five aerobic organisms. Only the alignments for enzymes are presented in this table. Each row in the table below represents an alignment of a single enzyme across the organisms *Bordetella bronchiseptica* RB50 (*bbr*), *Staphylococcus saprophyticus* subsp. *saprophyticus* ATCC 15305 (*ssp*), *Myxococcus xanthus* DK 1622 (*mxa*), *Leptospira interrogans* serovar *Lai* str. 56601 (*lil*), and *Helicobacter pylori* HPAG1 (*hpa*). The + entries in the table indicate which enzyme was identified as being aligned for that organism, whereas ⋄ indicates that no aligned enzyme was identified for that organism. Enzymes marked with ⋆ are part of the TCA cycle.

<table>
<thead>
<tr>
<th>Enzyme commission</th>
<th>Enzyme name</th>
<th>bbr</th>
<th>ssp</th>
<th>mxa</th>
<th>lil</th>
<th>hpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1.1.1.42</td>
<td>Isocitrate dehydrogenase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E1.3.99.1</td>
<td>Succinate dehydrogenase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E2.3.3.1</td>
<td>Citrate synthase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E4.2.1.2</td>
<td>Fumarase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4.2.1.3</td>
<td>Aconitase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E1.2.4.1</td>
<td>Pyruvate dehydrogenase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.2.7.1</td>
<td>Pyruvate synthase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.2.4.2</td>
<td>Alpha ketoglutarate dehydrogenase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E1.2.7.3</td>
<td>2-oxoglutarate synthase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.1.1.37</td>
<td>Malate dehydrogenase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E1.8.1.4</td>
<td>Dihydrolipoyl dehydrogenase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E2.3.1.12</td>
<td>Dihydrolipoyllysine-residue acetyltransferase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E2.3.1.61</td>
<td>Dihydrolipoyllysine-residue succinyltransferase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E6.2.1.5</td>
<td>Succinate thiokinase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E6.4.1.1</td>
<td>Pyruvate carboxylase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E4.1.3.6</td>
<td>Citrate (pro-3S)-lyase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E4.1.1.32</td>
<td>Phosphoenolpyruvate carboxykinase (GTP)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E4.1.1.49</td>
<td>Phosphoenolpyruvate carboxykinase (ATP)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E4.1.3.6</td>
<td>Citrate (pro-3S)-lyase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.2.7.3</td>
<td>2-oxoglutarate synthase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.1.1.41</td>
<td>Isocitrate dehydrogenase (NAD+)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

74
Table 6.8: Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five aerobic organisms. The parts of alignments for reactions and compounds are presented in this table.

<table>
<thead>
<tr>
<th>Reaction or Compound ID</th>
<th>Reaction or Compound name</th>
<th>bbr</th>
<th>ssp</th>
<th>mxa</th>
<th>lil</th>
<th>hpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>R00412</td>
<td>Acceptor oxidoreductase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>R01082</td>
<td>(S)-malate hydro-lyase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>R01900</td>
<td>isocitrate hydro-lyase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>R00351</td>
<td>Oxaloacetate C-acetyltransferase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>R01899</td>
<td>NADP+ oxidoreductase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C00022</td>
<td>Pyruvate</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C00024</td>
<td>Acetyl-CoA</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C00026</td>
<td>2-Oxoglutarate*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C00036</td>
<td>Oxaloacetate</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C00042</td>
<td>Succinate</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Table 6.9: Results for aligning the metabolic components of KEGG pathway 00020 (the TCA cycle) across five facultative anaerobic organisms. Only the alignments for enzymes are presented in this table. Each row in the table below represents an alignment of a single enzyme across the organisms *Escherichia coli* (*eco*), *Shewanella oneidensis* MR-1 (*son*), *Zymomonas mobilis* (*zmo*), *Listeria innocua* (*lin*), and *Streptococcus pneumonia* (*spn*). The + entries in the table indicate which enzyme was identified as being aligned for that organism, whereas ⋄ indicates that no aligned enzyme was identified for that organism. Enzymes marked with * are part of the TCA cycle.

<table>
<thead>
<tr>
<th>Enzyme commission</th>
<th>Enzyme name</th>
<th>eco</th>
<th>son</th>
<th>zmo</th>
<th>lin</th>
<th>spn</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1.2.4.1</td>
<td>Pyruvate dehydrogenase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E1.8.1.4</td>
<td>Dihydrolipoyl dehydrogenase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E2.3.1.12</td>
<td>Dihydrolipoyllysine-residue acetyltransferase</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E2.3.3.1</td>
<td>Citrate synthase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>⋄</td>
</tr>
<tr>
<td>E4.2.1.3</td>
<td>Aconitase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>⋄</td>
</tr>
<tr>
<td>E4.2.1.2</td>
<td>Fumarase*</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>⋄</td>
</tr>
<tr>
<td>E1.3.99.1</td>
<td>Succinate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6.2.1.5</td>
<td>Succinate thiokinase*</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E6.4.1.1</td>
<td>Pyruvate carboxylase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>E1.1.1.42</td>
<td>Isocitrate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E2.3.1.61</td>
<td>Dihydrolipoyllysine-residue succinyltransferase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1.3.99.1</td>
<td>Succinate dehydrogenase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>E1.3.99.1</td>
<td>Succinate dehydrogenase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>E1.1.1.42</td>
<td>Isocitrate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E4.1.3.6</td>
<td>Citrate (pro-3S)-lyase</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E1.1.1.37</td>
<td>Malate dehydrogenase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>E1.1.1.37</td>
<td>Malate dehydrogenase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>E4.2.1.2</td>
<td>Fumarase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E1.1.1.42</td>
<td>Isocitrate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E2.3.1.61</td>
<td>Dihydrolipoyllysine-residue succinyltransferase</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E4.1.1.49</td>
<td>Phosphoenolpyruvate carboxykinase (ATP)</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E4.1.1.49</td>
<td>Phosphoenolpyruvate carboxykinase (ATP)</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E1.2.4.2</td>
<td>Alpha ketoglutarate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E1.2.4.2</td>
<td>Alpha ketoglutarate dehydrogenase*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td>⋄</td>
</tr>
<tr>
<td>E1.1.1.42</td>
<td>Isocitrate dehydrogenase*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

76
Chapter 7

An Approximation Algorithm for the Maximum Duo-preservation String Mapping Problem (MPSM)

7.1 Introduction

The minimum common string partition problem (MCSP) has been well-investigated as a fundamental problem in computer science [26, 49]. Given two finite length strings over the finite letter alphabet, MCSP is to partition strings into identical substrings with the minimum number of partitions. MCSP is also viewed as the problem of finding a letter-preserving bijective mapping $\pi$ from letters in one string $A$ to letters in the other string $B$ with the minimum number of breaks, where a letter-preserving bijective mapping $\pi$ means that each letter in $A$ is mapped into the same letter in $B$ and the mapping is bijective, and a break is a pair of consecutive letters in $A$ that are mapped by $\pi$ to non-consecutive letters in $B$ [49]. In a string, a pair of consecutive letters is called a duo [49].

As an example, let us assume that there is a letter-preserving bijective mapping $\pi$ (see Figure 7.1) between two strings $A = abcab$ and $B = ababc$. From Figure 7.1, we can see that $\pi$ has only one break: $ca$ is a duo of $A$, but $\pi(c)\pi(a)$ is not a duo of $B$. However, the other three duos in $A$ (ab, bc, ab) are kept by $\pi$, each of which is called duo-preservation. So, the sum of the number of breaks and the number of duo-preservations is four, which is the length of any input string minus 1.

For a letter-preserving bijective mapping between two strings, on the one hand, the optimization goal can be to minimize the number of breaks that is known as the MCSP problem. On the other hand, the optimization goal can be to maximize the number of duo-preservations. We define the maximization version of the problem as the maximum duo-preservation string
mapping problem (MPSM), i.e. the problem of finding a letter-preservation bijective mapping $\pi$ from one string to the other string with the maximum number of duo-preservations. The restricted version of MPSM, where each letter occurs at most $k$ times in each input string, is denoted by $k$-MPSM. The MPSM problem is complementary to the MCSP problem as shown in Section 2. From this complementary relationship, it follows that MPSM is also $NP$-hard since the MCSP problem is $NP$-hard [49].

While the MCSP problem has been widely studied, to the best of our knowledge, the MPSM problem has not been addressed before. Specifically, various approximation algorithms have been proposed to solve the $k$-MCSP problem, a version of MCSP, where each letter appears at most $k$ times in any input string. These results are surveyed in Table 7.1. Motivated by the assignment of orthologous genes, Chen et al. introduced and studied the minimum common string partition problem [26]. They proposed an approximation algorithm for 2-MCSP within the approximation ratio of 1.5 based on the approximation algorithm for the minimum vertex cover problem. In [49], Goldstein et al. proved that MCSP is APX-hard and gave an approximation algorithm for 2-MCSP with the approximation ratio of 1.1037 and an approximation algorithm for 3-MCSP with the approximation ratio of 4. In [19], Chrobak et al. proposed a greedy approximation algorithm and showed that the greedy algorithm was of the approximation ratio of 3 for 2-MCSP and a low bound $\Omega(\log n)$ for 4-MCSP. Dan He designed another greedy algorithm with better average performance than Chrobak et al.’s greedy algorithm [57]. Kolman and Waleń advanced Chrobak et al.’s greedy algorithm and proved that their algorithm was of approximation bound $O(k^2)$ for $k$-MCPS [74]. Using the Hitting set, Kolman and Waleń also proposed an approximation algorithm with the approximation ratio of $4k$ for $k$-MCPS [75]. Damaschke studied the parameterized algorithm for MCSP and designed a fixed-parameter tractable (FPT) algorithm for some parameters [29].

Although there are approximation algorithms for MCSP, it is still required to design approximation algorithms for MPSM, because a pair of complementary $NP$-hard problems may have different approximation cases, i.e. an approximation algorithm for one problem sometimes cannot be used to approximate its complementary problem. For example, the minimum vertex cover problem and the maximum independent set problem are two well-known complementary
Table 7.1: The approximation ratio summary for $k$-MCSP

<table>
<thead>
<tr>
<th>Paper</th>
<th>2-MCSP</th>
<th>3-MCSP</th>
<th>$k$-MCSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[26]</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[49]</td>
<td>1.1037</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>[19]</td>
<td>3</td>
<td></td>
<td>$O(n^{0.69})$</td>
</tr>
<tr>
<td>[74]</td>
<td></td>
<td></td>
<td>$O(k^2)$</td>
</tr>
<tr>
<td>[75]</td>
<td></td>
<td></td>
<td>$4k$</td>
</tr>
</tbody>
</table>

problems in computer science [43]. For a given graph with $n$ vertices, the minimum vertex cover problem can be approximated within a ratio of 2, but the maximum independent set is $NP$-hard to approximate within a factor $n^\delta$, for some $\delta > 0$ [36, 7, 8]. Another pair of complementary problems is the Max-Satisfy problem and the Min-Unsatisfy problem [4, 6]. Both the Max-Satisfy problem and the Min-Unsatisfy problem are $NP$-hard, but their approximation cases are also different. For a system of $m$ linear equations with $n$ variables over fractional numbers $Q$, the Min-Unsatisfy problem can be approximated within a factor of $m + 1$, but the Max-Satisfy problem is $NP$-hard to approximate within a factor of $n^\delta$, for some $\delta > 0$ [4, 6].

In order to study approximation algorithms, Raghavan and Thompson introduced a randomized rounding method in [88]. Since then, the randomized rounding method has been widely used to design approximation algorithms for many $NP$-hard problems ([12, 59, 95, 81, 46, 47, 54, 62], etc.). The general idea behind the randomized rounding method is: (1) An $NP$-hard problem is first transformed into a 0-1 Integer Programming (IP) problem. (2) Then, it is relaxed to a Linear Programming (LP) problem. (3) For the optimal solution to LP, the value of each variable is rounded to 0 or 1 by some specific method. Thus, one approximation solution to some specific $NP$-hard problem can be achieved if the challenges of steps (1) and (3) can be overcome.

The contributions of the chapter are as follows:

1. We propose an approximation algorithm for MPSM based on the randomized rounding technology and prove that the approximation algorithm is of approximation ratio $k^3$ for $k$-MPSM;

2. We propose a 2-approximation algorithm for 2-MPSM, a 9-approximation algorithm for 3-MPSM, and a 16-approximation algorithm for 4-MPSM and

3. We give a 0-1 quadratic Integer Programming formulation for the MCSP problem to pave the way for new approximation algorithms of MCSP using the randomized rounding method.
7.2 Preliminaries

In this section, we first reproduce some formal notations and definitions (Definition 2.1.–Definition 2.6.) about the MCSP problem from [49], then describe the formal problem statement of the MPSM problem. Since our goal is to design a randomized approximation algorithm based on the randomized rounding technology, the definition of approximation ratio of a randomized approximation algorithm is introduced. Finally, we define the constrained maximum induced subgraph problem (CMIS) and the constrained minimum induced subgraph problem (CNIS) in order to give polynomial time reductions from MPSM and MCSP to CMIS and CNIS, respectively.

Definition 7.2.1 (Duo) A duo is an ordered pair of letters that occur consecutively in a string [49].

Definition 7.2.2 (Partition) A partition of a string $A$ is a sequence of strings $P = (P_1, P_2, \ldots, P_m)$ whose concatenation is equal to $A$, that is $P_1 \cdots P_m = A$, where the strings $P_i$ ($1 \leq i \leq m$) are called the blocks of $P$ and $m$ is called the number of blocks [49].

Definition 7.2.3 (Common Partition) Given a partition $P = (P_1, P_2, \ldots, P_m)$ of a string $A = a_1 \ldots a_n$ and a partition $Q = (Q_1, \ldots, Q_m)$ of a string $B = b_1 \ldots b_n$, we say that the pair $< P, Q >$ is a common partition $\pi$ of $A$ and $B$ if $Q$ is a permutation of $P$. The common partition $\pi$ can be naturally interpreted as a bijective mapping from $A$ to $B$, such that, for each $j$ ($1 \leq j \leq m$), the letters from $P_j$ are mapped from left to right to the corresponding letters from $Q_j$ ($1 \leq j' \leq m$) [49].

Definition 7.2.4 (Break) A break is a pair of letters that are consecutive in string $A$ but are mapped by $\pi$ to letters that are not consecutive in string $B$. Obviously, the block number of a partition is equal to its break number plus 1 [49].

Definition 7.2.5 (Letter-preserving Bijective Mapping) A letter-preserving bijective mapping is a bijective mapping $\pi$ from letters of string $A$ to letters of the other string $B$ such that any letter in $A$ is mapped into the same letter in $B$ [49]. Duo-preservation means that a duo of $A$ is kept by $\pi$ in $B$.

Problem 7.2.1 (Minimum Common String Partition Problem (MCSP)) The minimum common string partition problem (MCSP) is to find a common partition of two strings $A$ and $B$ with the minimum number of blocks. MCSP is also viewed as the problem of finding a letter-preserving bijective mapping from letters in one string to letters in the other string with the minimum number of breaks. The restricted version of MCSP, where each letter occurs at most $k$ times in each input string, is denoted by $k$-MCSP [49].
Definition 7.2.6 (Related Strings) Two strings $A$ and $B$ are related if every letter appears the same number of times in $A$ and $B$ [49].

Obviously, two strings have a common partition iff they are related [49].

Lemma 7.2.7 Two strings $A$ and $B$ have a letter-preserving bijective mapping iff they are related.

Proof. From the definition of the letter-preserving bijective mapping, there is a one-to-one correspondence between letters in $A$ and $B$. Thus, every letter appears the same number of times in $A$ and $B$. Hence, $A$ and $B$ are related. ■

Problem 7.2.2 (Maximum Duo-preservation String Mapping Problem (MPSM)) The maximum duo-preservation string mapping problem (MPSM) is the problem of finding a letter-preserving bijective mapping $\pi$ from string $A$ to string $B$ with the maximum number of duo-preservations, where $A$ and $B$ have the same length. The restricted version of MPSM, where each letter occurs at most $k$ times in each input string, is denoted by $k$-MPSM.

Note that two input strings are related in the MPSM problem.

Theorem 7.2.8 MPSM and MCSP are complementary.

Proof. Suppose $\pi$ is the letter-preserving bijective mapping between two strings $A$ and $B$ in MPSM and MCSP. Then, the number of duos of $A$ is $n - 1$. Let $n_b$ be the number of breaks and $n_d$ be the number of duo-preservations. Thus, $n_b + n_d = n - 1$. So, the MPSM problem is complementary to the MCSP problem. ■

Definition 7.2.9 (Approximation Ratio of Randomized Approximation Algorithm) Let $OPT(I)$ denote the optimum solution for an instance $I$ of the maximization problem. Let $\hat{E}(R(I))$ denote the expected value of the output solution $R(I)$ of a randomized approximation algorithm $R$. For some $r \geq 1$, if $\frac{OPT(I)}{\hat{E}(R(I))} \leq r$, for any instance $I$, then the randomized approximation algorithm $R$ is called the algorithm of approximation ratio of $r$ [54].

Problem 7.2.3 (CMIS and CNIS) Given an $m$-partite graph $G$ with $m$ parts: $M_1, \ldots, M_m$, where each $M_i$ has $n_i \times n_i$ vertices and all the vertices are put in an $n_i \times n_i$ matrix, the goal of the constrained maximum induced subgraph problem (CMIS) is to find $n_i$ vertices from each part $M_i$, where $n_i$ vertices are from different rows and different columns, such that the induced subgraph has the maximum number of edges. If all $n_i \leq k$, the restricted version is called $k$-CMIS. On the other hand, we define the minimization version as the constrained minimum induced subgraph (CNIS) problem in an $m$-partite graph.
7.3 A Polynomial Time Reduction from the MPSM Problem to the CMIS Problem

In this section, we give a polynomial time reduction from the MPSM problem to the CMIS problem.

Given two related strings \( X = x_1x_2 \ldots x_n \) and \( Y = y_1y_2 \ldots y_n \), let \( m \) be the number of different letters in \( X \) and \( S = \{a_1, \ldots, a_m\} \) be the unduplicated letter set. Let \( n_i \) be the number of appearances of letter \( a_i \) in \( X \). Thus, \( n_1 + \cdots + n_m = n \). In the following, we construct an instance \( G_{XY} \) of CMIS, which has \( m \) parts. For each \( a_i \), we construct one part \( M_i \), which has \( n_i \times n_i \) nodes. Let \( < a_i^{11}, a_i^{12}, \ldots, a_i^{n_i} > \) be all \( a_i \)'s in \( X \) by their appearance order. Let \( < a_i^{21}, a_i^{22}, \ldots, a_i^{2n_i} > \) be all \( a_i \)'s in \( Y \) by their appearance order. For each \( a_i^{1h} \) and \( a_i^{2\ell} \) (\( 1 \leq h, \ell \leq n_i \)), we construct one node \( (a_i^{1h}, a_i^{2\ell}) \) in the \( h \)-th row and \( \ell \)-th column of \( M_i \). Edges only exist between nodes from different parts. There is an edge between \( (a_i^{1h}, a_i^{2\ell}) \) and \( (a_j^{1r}, a_j^{2s}) \) iff \( a_i^{1h}a_i^{1r} \) is a duo in \( X \) and \( a_i^{2\ell}a_j^{2s} \) is a duo in \( Y \). For the graph \( G_{XY} \), the goal of the CMIS problem is to find \( n_i \) vertices at different rows and different columns from each part \( M_i \) such that the subgraph induced by the chosen \( n \) nodes \( (n = n_1 + \ldots + n_m) \) has the maximum number of edges.

The number of vertices in \( G_{XY} \) is \( n_1^2 + n_2^2 + \ldots + n_m^2 \) which is \( O(n^2) \). Thus, the number of edges in \( G_{XY} \) is at most \( O(n^4) \). So, the reduction is of polynomial time complexity.

For example, for two strings \( A = abcab \) and \( B = ababc \), we can construct a graph \( G_{AB} \) as follows (see Figure 7.2). \( G_{AB} \) has three parts \( M_1, M_2, \) and \( M_3 \). Part \( M_1 \) contains four \((a, a)\) nodes. Part \( M_2 \) contains four \((b, b)\) nodes, and part \( M_3 \) contains one \((c, c)\) node. In \( G_{AB} \), there is an edge between \((a, a)\) node at the position \((1,1)\) of \( M_1 \) and \((b, b)\) node at the position \((1,1)\) of \( M_2 \), because the first \( a \) and the first \( b \) in \( A \) form a duo \( ab \) and the first \( a \) and the first \( b \) in \( B \) also form a duo \( ab \). Other edges are similarly constructed.

![Figure 7.2: Two strings \( A = abcab \) and \( B = ababc \) are transformed into a graph \( G_{AB} \)](image-url)
In the graph $G_{AB}$, the five black nodes are chosen from different rows and different columns in each $M_i$ part, respectively, because the subgraph induced by these five nodes has the maximum edge number of 3 (dashed lines in Figure 7.2).

The following Theorem 7.3.1 shows that the MPSM problem for strings $X$ and $Y$ is related to the CMIS problem in the graph $G_{XY}$.

**Theorem 7.3.1** For the graph $G_{XY}$, an induced subgraph by $n$ nodes, of which $n_i$ nodes are chosen from different rows and different columns in each $M_i$ part respectively, has the maximum number of edges iff $X$ and $Y$ have a bijective mapping with the maximum number of duo-preservations.

**Proof.** Since each node in the graph $G_{XY}$ denotes that a letter in $X$ is mapped to a letter in $Y$, these $n$ nodes denote a bijective mapping $\pi$ from $X$ to $Y$. Since each edge in $G_{XY}$ denotes a duo-preservation, the number of edges in the induced subgraph by these $n$ nodes is the number of duo-preservations in the bijective mapping $\pi$. Thus, the maximum number of edges in the induced subgraph by these $n$ nodes is the maximum number of duo-preservations in a bijective mapping. The other direction of the theorem is trivial. $
$
Thus, by Theorem 7.3.1 an approximation algorithm for the MPSM problem can be achieved by designing an approximation algorithm for the CMIS problem with the same approximation ratio. If each letter in $X$ and $Y$ appears at most $k$ times, then each part in $G_{XY}$ has at most $k \cdot k$ nodes by the above reduction process. Thus, the $k$-MPSM problem can be reduced to the $k$-CMIS problem with the same approximation ratio.

### 7.4 An Approximation Algorithm for the Constrained Maximum Induced Subgraph (CMIS) Problem

In this section, we will design an approximation algorithm for the CMIS problem based on the Linear Programming (LP) technology.

First, we give the 0-1 quadratic Integer Programming (IP) formulation for the CMIS problem. Suppose $G$ is an $m$-partite graph with $m$ parts: $M_1, \ldots, M_m$, where the vertices in each $M_i$ are put in $n_i \times n_i$ matrix. For each node $v_{ip}^r$ ($1 \leq i \leq n_r$, $1 \leq p \leq n_r$) in $M_r$ ($1 \leq r \leq m$), where $v_{ip}^r$ is at the $i$-th row and $p$-th column, let $x_{ip}^r$ be a 0–1 decision variable, that $x_{ip}^r = 1$ means that $v_{ip}^r$ is chosen, otherwise $v_{ip}^r$ is not chosen. Thus, the 0-1 quadratic IP formulation for the CMIS problem is as follows.
Maximize \( \sum_{(v^p, v^q) \in E} x^p_i x^q_j \) \hspace{1cm} (IP_1)

subject to \( \sum_{i=1}^{n_r} x^p_i = 1, \text{ for } r = 1, \ldots, m \) \hspace{1cm} (1)

\( \sum_{p=1}^{n_r} x^p_i = 1, \text{ for } r = 1, \ldots, m \) \hspace{1cm} (2)

\( x^p_i \in \{0, 1\}, \text{ for } r = 1, \ldots, m \)

Constraints (1) and (2) guarantee that only the nodes at different rows and different columns from each part are chosen.

For example, for the graph \( G_{AB} \) in Figure 7.2, we have the following 0-1 quadratic IP formulation:

Maximize \( A_{11} B_{11} + A_{12} B_{12} + B_{12} C + A_{21} B_{21} + A_{22} B_{22} \)

subject to \( A_{11} + A_{12} = 1; A_{21} + A_{22} = 1; \)

\( A_{11} + A_{21} = 1; A_{12} + A_{22} = 1; \)

\( B_{11} + B_{12} = 1; B_{21} + B_{22} = 1; \)

\( B_{11} + B_{21} = 1; B_{12} + B_{22} = 1; C = 1; \)

\( A_{ij}, B_{ij} \in \{0, 1\}, \text{ where } i = 1 \text{ or } 2, j = 1 \text{ or } 2 \)

Where \( A_{ij} \) (\( 1 \leq i, j \leq 2 \)) is the decision variable corresponding to the node at the \( i \)-th row and \( j \)-th column in \( M_1 \), \( B_{ij} \) (\( 1 \leq i, j \leq 2 \)) is the decision variable corresponding to the node at the \( i \)-th row and \( j \)-th column in \( M_2 \), and \( C \) is the decision variable corresponding to the node in \( M_3 \).

Using the common relaxation method for quadratic IP formulation [54], we build the following LP formulation for the CMIS problem:

Maximize \( \sum_{(v^p, v^q) \in E} z_{rip, sjq} \) \hspace{1cm} (LP_1)

subject to \( z_{rip, sjq} \leq x^p_i \), for all \( r \) \hspace{1cm} (3)

\( z_{rip, sjq} \leq x^q_j \), for all \( r \) \hspace{1cm} (4)

\( \sum_{i=1}^{n_r} x^p_i = 1, \text{ for } r = 1, \ldots, m \)

\( \sum_{p=1}^{n_r} x^p_i = 1, \text{ for } r = 1, \ldots, m \)

\( 0 \leq z_{rip, sjq} \leq 1, \text{ for all } r \)

\( 0 \leq x^p_i \leq 1, \text{ for all } r \)

\( 0 \leq x^q_j \leq 1, \text{ for all } r \)

For example, the \( LP_1 \) formulation for the graph \( G_{AB} \) in Figure 7.2 is as follows:
Maximize $z_1 + z_2 + z_3 + z_4 + z_5$
subject to $z_1 \leq A_{11}, z_1 \leq B_{11};$
    $z_2 \leq A_{12}, z_2 \leq B_{12};$
    $z_3 \leq B_{12}, z_3 \leq C;$
    $z_4 \leq A_{21}, z_4 \leq B_{21};$
    $z_5 \leq A_{22}, z_5 \leq B_{22};$
    $A_{11} + A_{12} = 1; A_{21} + A_{22} = 1;$
    $A_{11} + A_{21} = 1; A_{12} + A_{22} = 1;$
    $B_{11} + B_{12} = 1; B_{21} + B_{22} = 1;$
    $B_{11} + B_{21} = 1; B_{12} + B_{22} = 1; C = 1;$
    $A_{ij}, B_{ij}, z_r \in [0, 1]$ for all $i, j, r$

In the following, we design the approximation algorithm for the CMIS problem based on its LP$_1$ formulation (see Algorithm 7.4.1).

Algorithm 7.4.1: The approximation algorithm for the CMIS problem.
It is known that Linear Programming can be solved in a polynomial time (see [27]). Thus, Algorithm 7.4.1 is of polynomial time complexity.

**Theorem 7.4.1** The Algorithm 7.4.1 is of approximation ratio $k^3$ for $k$-CMIS.

**Proof.** Let $Pr(X)$ denote the probability of the event $X$. Let $\hat{E}(X)$ denote the expected value of the event $X$. For each $r$, let $A_j$ denote the event that $x_{ij}^p = 1$ appears in the $j$-th while loop (line 5 in Algorithm 7.4.1). Since $x_{ij}^p = 1$ can occur in the first while loop, or the second while loop, or the $n_r$-th while loop, $Pr(x_{ij}^p = 1) = Pr(A_1 \lor \cdots \lor A_{n_r})$. Thus $Pr(x_{ij}^p = 1) \geq Pr(A_1)$.

Since $Pr(A_1) = \frac{\sqrt{x_{ij}^p}}{\sum_{x_{ij}^p \in X_r} \sqrt{x_{ij}^p}}$ (the probability is similar to that in [54]), we get: $Pr(x_{ij}^p = 1) \geq \frac{\sqrt{x_{ij}^p}}{n_r}$.

Since $\sum_{x_{ij}^p \in X_r} \sqrt{x_{ij}^p} \leq \sqrt{\sum_{x_{ij}^p \in X_r} x_{ij}^p} \leq \sqrt{n_r}$, we get $\sum_{x_{ij}^p \in X_r} \sqrt{x_{ij}^p} \leq n_r \sqrt{n_r}$. Thus $Pr(x_{ij}^p = 1) \geq \frac{1}{n_r} \sqrt{x_{ij}^p \sqrt{n_r}} \geq \frac{1}{k} \frac{1}{\sqrt{k}} \sqrt{x_{ij}^p}$. Similarly, we can get $Pr(x_{ij}^q = 1) \geq \frac{1}{k} \frac{1}{\sqrt{k}} \sqrt{x_{ij}^q}$.

So $Pr(z_{ij} \equiv 1) = Pr(x_{ij}^p = 1)Pr(x_{ij}^q = 1) = \frac{1}{k} \sqrt{x_{ij}^p} \sqrt{x_{ij}^q} \geq \frac{1}{k} \frac{1}{\sqrt{k}} \min \{x_{ij}^p, x_{ij}^q\}$

$= \frac{1}{k} \frac{1}{\sqrt{k}} z_{ij}^*$ (By the constraints (3) and (4)).

For any instance $I$, let $A(I)$ denote the output solution of the approximation algorithm. Let $OPT(I)$ denote the optimum solution. Let $OPT(IP_1)$ denote the optimum solution of $IP_1$ formulation for $I$. Let $OPT(LP_1)$ denote the optimum solution of $LP_1$ formulation for $I$.

So $\hat{E}(A(I)) = \hat{E}(\sum_{(i,j),(s,q)\in E} z_{ij}^* s_{jq}) = \sum_{(i,j),(s,q)\in E} \hat{E}(z_{ij}^* s_{jq}) = \sum_{(i,j),(s,q)\in E} Pr(z_{ij} \equiv 1) = \frac{1}{k} \sum_{(i,j),(s,q)\in E} z_{ij}^*$

Thus $\frac{OPT(I)}{\hat{E}(A(I))} \leq \frac{OPT(LP_1)}{\hat{E}(A(I))} \leq k^3$.

Hence, the Algorithm 7.4.1 is of approximation ratio $k^3$ for $k$-CMIS. ■

Since the $k$-MPSM problem can be reduced to $k$-CMIS problem, an approximation algorithm for the $k$-MPSM problem can be achieved with the same approximation ratio as the Algorithm 7.4.1. Thus, we can get the following conclusion.
Corollary 7.4.2 There is an approximation algorithm with approximation ratio $k^3$ for $k$-MPSM.

7.5 Improved Approximation Algorithms for 2-CMIS, 3-CMIS, and 4-CMIS

In this section, we propose improved approximation algorithms for 2-CMIS, 3-CMIS, and 4-CMIS.

First, we design an improved approximation algorithm for the 2-CMIS problem based on its $LP_1$ formulation (see Algorithm 7.5.1).

By solving the Linear Programming formulation $LP_1$ for the 2-CMIS problem, we get an optimum solution $x_{rip}^r$, $z_{rip}^r s_{jq}$ for all $r$.

Randomized rounding:

```
1  for $r = 1$ to $m$ do
2      if $n_r = 2$ then
3          $X_r = \begin{pmatrix} x_r^{11} & x_r^{12} \\ x_r^{21} & x_r^{22} \end{pmatrix}$. Let $Y_1 = (x_r^{11}, x_r^{22})$ and $Y_2 = (x_r^{12}, x_r^{21})$
4          $Y_i$ is chosen with probability $\sqrt{x_r^{11}} / (\sqrt{x_r^{11}} + \sqrt{x_r^{22}})$ ($i = 1, 2$).
5          When $Y_1$ is chosen, we set $x_r^{11} = x_r^{22} = 1$ and $x_r^{12} = x_r^{21} = 0$; When $Y_2$ is chosen, we set $x_r^{12} = x_r^{21} = 1$ and $x_r^{11} = x_r^{22} = 0$.
6  end
7  if $n_r = 1$ then
8      We set $x_r^{11} = 1$.
9  end
10 end
11 Output those nodes whose variables are set to 1. Let $z_{rip}^r s_{jq}$ = $x_{rip}^r x_{jq}^s$. The edge number of induced subgraph by these output nodes is $\sum_{(ip, jq) \in E} z_{rip}^r s_{jq}$.
```

Algorithm 7.5.1: The approximation algorithm for the 2-CMIS problem.

It is known that Linear Programming can be solved in a polynomial time (see [27]). Thus, Algorithm 7.5.1 is of polynomial time complexity.

Theorem 7.5.1 The Algorithm 7.5.1 is of approximation ratio 2 for 2-CMIS.

Proof. Let $Pr(X)$ denote the probability of the event $X$. Let $\hat{E}(X)$ denote the expected value of the event $X$. For each $r$, let $A_j$ denote the event that $Y_j$ is chosen ($j = 1, 2$).
By the constraints (3) and (4), we get $z_{r_ip_{s_q}}^* = \min\{x_{r_p}^{ip}, x_{s_q}^{jq}\}$. By the constraints (3') and (4'), we get $x_{r_1}^{11s} + x_{r_2}^{12s} = 1, x_{r_1}^{11s} + x_{r_2}^{21s} = 1$ and $x_{r_1}^{12s} + x_{r_2}^{22s} = 1$. So, we get $x_{r_1}^{11s} = x_{r_2}^{22s}$ and $x_{r_2}^{12s} = x_{r_2}^{21s}$.

Thus, for any $i, p (i, p = 1$ or $2)$, when $i = p$, $Pr(x_{r_i}^{ip} = 1) = Pr(A_1) = \frac{\sqrt{x_{r_1}^{11s}}}{\sqrt{x_{r_1}^{11s} + \sqrt{x_{r_2}^{12s}}}}$; When $i \neq p$, $Pr(x_{r_i}^{ip} = 1) = Pr(A_2) = \frac{\sqrt{x_{r_2}^{12s}}}{\sqrt{x_{r_1}^{11s} + \sqrt{x_{r_2}^{12s}}}} = \frac{\sqrt{x_{r_2}^{12s}}}{\sqrt{x_{r_1}^{11s} + \sqrt{x_{r_2}^{12s}}}}$. So, in any case, $Pr(x_{r_i}^{ip} = 1) = \frac{\sqrt{x_{r_i}^{ip}}}{\sqrt{x_{r_1}^{11s} + \sqrt{x_{r_2}^{12s}}}}$.

Since $\frac{\sqrt{x_{r_1}^{11s} + \sqrt{x_{r_2}^{12s}}}}{2} \leq \sqrt{\frac{x_{r_1}^{11s} + x_{r_2}^{12s}}{2}} \leq \sqrt{\frac{1}{2}}$, we get $\sqrt{x_{r_1}^{11s}} + \sqrt{x_{r_2}^{12s}} \leq \sqrt{2}$. Thus $Pr(x_{r_i}^{ip} = 1) \geq \frac{1}{\sqrt{2}} \sqrt{x_{r_i}^{ip}}$. Similarly, we can get $Pr(x_{s_q}^{jq} = 1) \geq \frac{1}{\sqrt{2}} \sqrt{x_{s_q}^{jq}}$.

So $Pr(z_{r_ip_{s_q}}^* = 1) = Pr(x_{r_i}^{ip} x_{s_q}^{jq} = 1)$
$= Pr(x_{r_i}^{ip} = 1) Pr(x_{s_q}^{jq} = 1)$
$\geq \frac{1}{2} \sqrt{x_{r_i}^{ip}} \sqrt{x_{s_q}^{jq}}$
$\geq \frac{1}{2} \min\{x_{r_i}^{ip}, x_{s_q}^{jq}\}$
$= \frac{1}{2} \cdot z_{r_ip_{s_q}}^*.$

For any instance $I$, let $A(I)$ denote the output solution of the approximation algorithm. Let $OPT(I)$ denote the optimum solution. Let $OPT(IP_1)$ denote the optimum solution of $IP_1$ formulation for $I$. Let $OPT(LP_1)$ denote the optimum solution of $LP_1$ formulation for $I$.

So $\hat{E}(A(I)) = \hat{E}\left( \sum_{(i,j) \in E} z_{r_ip_{s_q}}^* \right)$
$= \sum_{(i,j) \in E} \hat{E}(z_{r_ip_{s_q}}^*)$
$= \sum_{(i,j) \in E} Pr(z_{r_ip_{s_q}}^* = 1)$
$\geq \frac{1}{2} \sum_{(i,j) \in E} z_{r_ip_{s_q}}^*$
$= \frac{1}{2} \cdot OPT(LP_1)$.

Thus $\frac{OPT(IP_1)}{OPT(LP_1)} \leq \frac{OPT(LP_1)}{OPT(I)} \leq 2$.

Hence, the Algorithm 7.4.1 is of approximation ratio 2 for 2-CMIS. ■

Since the 2-MPSM problem can be reduced to 2-CMIS problem, an approximation algorithm for the 2-MPSM problem can be achieved with the same approximation ratio as the Algorithm 7.5.1. Thus, we can get the following conclusion.

**Corollary 7.5.2** There is an approximation algorithm with approximation ratio 2 for 2-MPSM.

Second, we design the approximation algorithm for the 3-CMIS problem based on its $LP_1$ formulation (see Algorithm 7.5.2).
1 By solving the Linear Programming formulation \( LP_1 \) for the 3-CMIS problem, we get an optimum solution \( x_r^{ip}, z_{rip,sjq}^* \) for all \( r \).

2 Randomized rounding:

3 for \( r = 1 \) to \( m \) do

4 if \( n_r = 3 \) then

5 \[ X_r = \begin{pmatrix} x_r^{11s} & x_r^{12s} & x_r^{13s} \\ x_r^{21s} & x_r^{22s} & x_r^{23s} \\ x_r^{31s} & x_r^{32s} & x_r^{33s} \end{pmatrix}. \]

6 Let \( Y_1^r = (x_r^{11s}, x_r^{22s}, x_r^{33s}), Y_2^r = (x_r^{12s}, x_r^{23s}, x_r^{31s}) \) and \( Y_3^r = (x_r^{13s}, x_r^{21s}, x_r^{32s}). \)

7 Let \( S_1 = x_r^{11s} + x_r^{22s} + x_r^{33s}, S_2 = x_r^{12s} + x_r^{23s} + x_r^{31s} \) and \( S_3 = x_r^{13s} + x_r^{21s} + x_r^{32s}. \)

8 \( Y_i \) is chosen with probability \( \frac{\sqrt{S_i}}{\sqrt{S_1} + \sqrt{S_2} + \sqrt{S_3}} (i = 1, 2, 3). \)

9 When \( Y_1 \) is chosen, we set \( x_r^{11} = x_r^{22} = x_r^{33} = 1; \) When \( Y_2 \) is chosen, we set \( x_r^{12} = x_r^{23} = x_r^{31} = 1; \) When \( Y_3 \) is chosen, we set \( x_r^{13} = x_r^{21} = x_r^{32} = 1; \)

10 end

11 if \( n_r = 2 \) then

12 We set \( x_r^{ip} = 1 \) by the method in Algorithm 7.5.1.

13 end

14 if \( n_r = 1 \) then

15 We set \( x_r^{11} = 1. \)

16 end

17 end

18 Output those nodes whose variables are set to 1. Let \( z_{rip,sjq}' = z_{rip,sjq}^{ip,jq}. \) The edge number of induced subgraph by these output nodes is \( \sum_{(v_r^{ip}, v_r^{jq}) \in E} z_{rip,sjq}'. \)

**Algorithm 7.5.2:** The approximation algorithm for the 3-CMIS problem.

It is known that Linear Programming can be solved in a polynomial time (see [27]). Thus, Algorithm 7.5.2 is of polynomial time complexity.

**Theorem 7.5.3** The Algorithm 7.5.2 is of approximation ratio 9 for 3-CMIS.

**Proof.** Let \( Pr(X) \) denote the probability of the event \( X \). Let \( E(X) \) denote the expected value of the event \( X \). For each \( r \), let \( A_j \) denote the event that \( Y_j \) is chosen \((j = 1, 2, 3)\).

By the constraints (3) and (4), we get \( z_{rip,sjq}^* = \min\{x_r^{ip}, x_s^{jq}\}. \) By the constraints \( (3') \) and \( (4') \), we get \( S_1 + S_2 + S_3 = x_r^{11s} + x_r^{12s} + x_r^{13s} + x_r^{21s} + x_r^{22s} + x_r^{23s} + x_r^{31s} + x_r^{32s} + x_r^{33s} = 3. \)

When \( n_r = 3 \), for any \( i, p \), if \( x_r^{ip} \in Y_1^r \), then \( Pr(x_r^{ip} = 1) = Pr(A_j) = \frac{\sqrt{S_j}}{\sqrt{S_1} + \sqrt{S_2} + \sqrt{S_3}} \geq \frac{\sqrt{S_1}}{\sqrt{S_1} + \sqrt{S_2} + \sqrt{S_3}}. \) Since \( \sqrt{\frac{S_1}{3}} + \sqrt{\frac{S_2}{3}} + \sqrt{\frac{S_3}{3}} \leq \sqrt{S_1 + S_2 + S_3} = \sqrt{3} = 1 \), we get \( \sqrt{S_1 + S_2 + S_3} \leq 3. \)

89
Thus \( \Pr(x_{ip}^p = 1) \geq \frac{1}{3} \sqrt{x_{ip}^{ps}} \). When \( n_r = 2 \), by the proof of Theorem 7.5.1, for any \( i, p \), we have \( \Pr(x_{ip}^p = 1) \geq \frac{1}{\sqrt{2}} \sqrt{x_{ip}^{ps}} \geq \frac{1}{3} \sqrt{x_{ip}^{ps}} \). Thus, in any case, we get \( \Pr(x_{ip}^p = 1) \geq \frac{1}{3} \sqrt{x_{ip}^{ps}} \).

Similarly, we can get \( \Pr(x_{jq}^q = 1) \geq \frac{1}{3} \sqrt{x_{jq}^{qs}} \).

So \( \Pr(z_{rip,jsq} = 1) = \Pr(x_{ip}^p x_{jq}^q = 1) \)
\[ = \Pr(x_{ip}^p = 1) \Pr(x_{jq}^q = 1) \]
\[ \geq \frac{1}{9} \sqrt{x_{ip}^{ps}} \sqrt{x_{jq}^{qs}} \]
\[ \geq \frac{1}{9} \min\{x_{ip}^{ps}, x_{jq}^{qs}\} \]
\[ = \frac{1}{9} z_{rip,jsq}^* \]

For any instance \( I \), let \( A(I) \) denote the output solution of the approximation algorithm. Let \( OPT(I) \) denote the optimum solution. Let \( OPT(IP_1) \) denote the optimum solution of \( IP_1 \) formulation for \( I \). Let \( OPT(LP_1) \) denote the optimum solution of \( LP_1 \) formulation for \( I \).

So \( \tilde{E}(A(I)) = \tilde{E}(\sum_{(v_{ip}^p, v_{jq}^q) \in E} z_{rip,jsq}^t) \)
\[ = \sum_{(v_{ip}^p, v_{jq}^q) \in E} \tilde{E}(z_{rip,jsq}^t) \]
\[ = \sum_{(v_{ip}^p, v_{jq}^q) \in E} \Pr(z_{rip,jsq}^t = 1) \]
\[ \geq \frac{1}{9} \sum_{(v_{ip}^p, v_{jq}^q) \in E} z_{rip,jsq}^* \]
\[ = \frac{1}{9} \cdot OPT(LP_1). \]

Thus \( \frac{OPT(I)}{E(A(I))} = \frac{OPT(IP_1)}{E(A(I))} \leq \frac{OPT(LP_1)}{E(A(I))} \leq 9. \)

Hence, the Algorithm 7.5.2 is of approximation ratio 9 for 3-CMIS.

Since the 3-MPSM problem can be reduced to 3-CMIS problem, an approximation algorithm for the 3-MPSM problem can be achieved with the same approximation ratio as the Algorithm 7.5.2. Thus, we can get the following conclusion.

Corollary 7.5.4 There is an approximation algorithm with approximation ratio 9 for 3-MPSM.

Similar to Algorithm 7.5.2, we can design an approximation algorithm for 4-CMIS with approximation ratio 16. We omit the algorithm details. Thus, we get the following conclusion.

Corollary 7.5.5 There is an approximation algorithm with approximation ratio 16 for 4-MPSM.

7.6 A 0-1 Quadratic Integer Programming Formulation for the Minimum Common String Partition Problem (MCSP)

In this section, we build a 0-1 quadratic Integer Programming formulation for the MCSP problem in order to help researchers design good approximation algorithms using the randomized rounding method for this problem.
First, we give a reduction from the MCSP problem to the constrained minimum induced subgraph (CNIS) problem. The reduction idea is similar to that in Section 3.

Given two related strings \( X = x_1x_2 \ldots x_n \) and \( Y = y_1y_2 \ldots y_n \), let \( m \) be the number of different letters in \( X \) and \( S = \{ a_1, \ldots, a_m \} \) be the unduplicated letter set. Let \( n_i \) be the number of appearance of letter \( a_i \) in \( X \). Thus, \( n_1 + \ldots + n_m = n \). In the following, we construct an instance \( G_{XY} \) of CMIS, which has \( m \) parts. For each \( a_i \), we construct one part \( M_i \), which has \( n_i \times n_i \) nodes. Let \( < a_1^{11}, a_1^{12}, \ldots, a_1^{n_i} > \) be all \( a_i \) in \( X \) by their appearance order. Let \( < a_2^{11}, a_2^{12}, \ldots, a_2^{n_i} > \) be all \( a_i \) in \( Y \) by their appearance order. For each \( a_1^{1h} \) and \( a_2^{1\ell} \) (\( 1 \leq h, \ell \leq n_i \)), we construct one node \( (a_1^{1h}, a_2^{1\ell}) \) in the \( h \)-th row and \( \ell \)-th column of \( M_i \). Edges only exist between nodes from different parts. There is an edge between \((a_1^{1h}, a_2^{1\ell})\) and \((a_1^{1r}, a_2^{1s})\) iff \( a_1^{1h}a_2^{1r} \) is a duo in \( X \) and \( a_1^{1\ell}a_2^{1s} \) is not a duo in \( Y \). For the graph \( G_{XY} \), the goal of the CNIS problem is to find \( n_i \) vertices at different rows and different columns from each part \( M_i \) such that the subgraph induced by chosen total \( n \) nodes \( (n = n_1 + \ldots + n_m) \) has the minimum number of edges.

For example, for two strings \( A = abcab \) and \( B = ababc \), we can construct a graph \( \bar{G}_{AB} \) as follows (see Figure 7.3).

In the graph \( \bar{G}_{AB} \), the subgraph induced by the five black nodes has the minimum edge number of 1 (dashed line in Figure 7.3).

The following conclusion shows that the MCSP problem for strings \( X \) and \( Y \) is related to the CNIS problem in the graph \( G_{XY} \).

**Theorem 7.6.1** For the graph \( G_{XY} \), a subgraph induced by \( n \) nodes, of which \( n_i \) nodes are chosen from different rows and different columns in each \( M_i \) part respectively, has the minimum number of edges iff \( X \) and \( Y \) have a bijective mapping with the minimum number of breaks.

**Proof.** Since each node in the graph \( \bar{G}_{XY} \) denotes that a letter in \( X \) is mapped to a letter in \( Y \),
these $n$ nodes denote a bijective mapping $\pi$ from $X$ to $Y$. Since each edge in $\bar{G}_{XY}$ must produce a break, the number of edges in the induced subgraph by $n$ nodes is the number of breaks in the bijective mapping $\pi$. Thus, the minimum number of edges in the subgraph induced by $n$ nodes is the minimum number of breaks in a bijective mapping. The other direction of the theorem is trivial.

In the following, for the CNIS problem in the graph $\bar{G}_{XY}$, we give a 0-1 quadratic IP formulation. Suppose $\bar{G}_{XY}$ has $m$ parts: $M_1, \ldots, M_m$, where the vertices in each $M_i$ are put in an $n_i \times n_i$ matrix. For each node $v_{ip}^r$ ($1 \leq i \leq n_r, 1 \leq p \leq n_r$) in $M_r$ ($1 \leq r \leq m$), where $v_{ip}^r$ is at the $i$-th row and $p$-th column, let $x_{ip}^r$ be a 0-1 decision variable, that $x_{ip}^r = 1$ means that $v_{ip}^r$ is chosen, otherwise $v_{ip}^r$ is not chosen. Thus, the 0-1 quadratic IP formulation for the MCSP problem is as follows.

Minimize $\sum_{(v_{ip}^r, v_{jq}^s) \in E} x_{ip}^r x_{jq}^s$ $(IP_2)$

subject to $\sum_{i=1}^{n_r} x_{ip}^r = 1$, for $r = 1, \ldots, m$ (5)
$\sum_{p=1}^{n_r} x_{ip}^r = 1$, for $r = 1, \ldots, m$ (6)
$x_{ip}^r \in \{0, 1\}$, for $r = 1, \ldots, m$

Constraints (5) and (6) guarantee that the nodes at different rows and different columns from each part are chosen.

7.7 Conclusion

In this chapter, we have proposed an approximation algorithm for the maximum duo-preservation string mapping problem (MPSM) based on the randomized rounding technology. We have proved that the approximation algorithm is of approximation ratio $k^3$, where $k$ is the maximum number of each letter in each string. In addition, we have proposed a 2-approximation algorithm for 2-MPSM, a 9-approximation algorithm for 3-MPSM, and a 16-approximation algorithm for 4-MPSM. We have also designed a 0-1 quadratic Integer Programming formulation for the minimum common string partition problem (MCSP). We hope that the formulation can be used to design good approximation algorithms for the MCSP problem via the randomized rounding method in the future.
Chapter 8

Conclusion

In this thesis, we study the parameterized complexity and algorithms for graph problems driven by comparative analysis of large-scale biological networks. From the parameterized complexity theory perspective, we study the following problems: the Maximum Common Subgraph of Multiple Graphs (Multi-MCS), the Maximum Common Subtree of Multiple Trees (Multi-MCT) and the Minimum Common Supertree of Multiple Trees (Multi-MCST). From the efficient algorithm development perspective, we study the following problems: the Maximum Weight Induced Subgraph problem (MWIS), the Densest \textit{at-least-}k-subgraph problem (DalkS), the Densest \textit{at-most-}k-subgraph problem (DamkS), the Multiple Alignment problem for Metabolic Pathways without abstraction, and the Maximum Duo-preservation String Mapping problem (MPSM).

In Chapter 2, we have studied the parameterized complexity of Multi-MCS. Section 2.5 shows the linear FPT reductions of the parameterized versions of the Longest Common String (LCS) problem to the parameterized versions of the Multi-MCS problem for directed graphs with vertex labels. Section 2.6 proves that parameterized version of the Multi-MCS problem for unlabeled undirected graphs is $W[1]$-hard by showing linear FPT reductions of the problem from the parameterized maximum clique problem. It also shows that a special case of one of the parameterized problems is $W[1]$-complete.

In Chapter 3, we have proven that various parameterized versions of the Multi-MCT and Multi-MCST problems are hard for various parameterized complexity classes. Since all of the problems considered here are at least $W[1]$-hard, it is unlikely that the parameterized versions of the Multi-MCT and Multi-MCST can be solved in $O(f(k) \cdot n^{O(1)})$ time.

In Chapter 4, we have studied the maximum weight induced subgraph problem. We give a greedy heuristic algorithms for the maximum weight induced subgraph problem. The idea of our greedy algorithm is that it repeatedly removes the minimum weighted vertex. Our algorithm is faster than MaWish algorithm in the worst case. We also implement the greedy algorithm
and use it to find the local alignment of pairwise Protein-Protein Interaction (PPI) networks by models of evolution. We compare our results with MaWish algorithm. Experimental results show that our greedy algorithm is more accurate and has higher specificity and sensitivity ratio than MaWish. It is also an order of magnitude faster than MaWish.

In Chapter 5, we have studied the densest at-least-$k$-subgraph problem (DalkS) and the densest at-most-$k$-subgraph problem (DamkS). We have presented two polynomial time algorithms for the densest at-least-$k$-subgraph problem (DalkS) when $k$ is bounded by a constant. We have also provided two approximation algorithms for the densest at-most-$k$-subgraph problem (DamkS), one with an approximation ratio of $\frac{n-1}{k-1}$ and the other with ratio $O(n^\delta)$ for some $\delta < 1/3$.

In Chapter 6, we have considered the multiple alignment problem for metabolic pathways. We have reduced the problem of aligning multiple metabolic pathways to the maximum-weighted $k$-partite matching problem. For the maximum-weighted $k$-partite matching problem, we presented an approximation algorithm and a heuristic algorithm. Finally, we applied our heuristic algorithm to compute the alignment of multiple metabolic pathways, and our experimental results verify that our algorithm correctly identifies the common subnetworks of multiple pathways.

In Chapter 7, we have proposed an approximation algorithm for the maximum duo-preservation string mapping problem (MPSM) based on the randomized rounding technology. We have proved that the approximation algorithm is of approximation ratio $k^3$, where $k$ is the maximum number of each letter in each string. In addition, we have proposed a 2-approximation algorithm for 2-MPSM, a 9-approximation algorithm for 3-MPSM, and a 16-approximation algorithm for 4-MPSM. We have also designed a 0-1 quadratic Integer Programming formulation for the minimum common string partition problem (MCSP). We hope that the formulation can be used to design good approximation algorithms for the MCSP problem via the randomized rounding method in the future.
REFERENCES


Qiaoming Han, Yinyu Ye, and Jiawei Zhang. Approximation of dense-k subgraph.


