On Approximate Parameterized String Matching and Related Problems

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

DOCTOR OF PHILOSOPHY

by

Shibsankar Das
(Roll Number: 10612301)

to the

Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati - 781 039, India

August 2016

"If we knew what it was we were doing,
it would not be called research, would it?"

- Albert Einstein

Dedicated
To
My Family

"Are great things ever done smoothly?
Time, patience, and indomitable will must show."

- Swami Vivekananda

Contents

DECLARATION

It is certified that the work contained in the thesis entitled **"On Approximate Parameterized String Matching and Related Problems"** has been done by me, a student in the Department of Mathematics, Indian Institute of Technology Guwahati under the guidance of **Dr. Kalpesh Kapoor** for the award of Doctor of Philosophy and that this work has not been submitted elsewhere for a degree.

August 2016

Shibsankar Das
Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati - 781 039, India

CERTIFICATE

It is certified that the work contained in the thesis entitled **"On Approximate Parameterized String Matching and Related Problems"** by **Shibsankar Das**, a student in the Department of Mathematics, Indian Institute of Technology Guwahati for the award of the degree of Doctor of Philosophy has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

August 2016

Dr. Kalpesh Kapoor
Associate Professor
Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati - 781 039, India

Acknowledgements

This thesis is a result of five years of work carried out at the Department of Mathematics, Indian Institute of Technology Guwahati (IITG), India and Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University (CTU) in Prague, Czech Republic. Many people at different levels have helped, one way or the other, in completion of my thesis and I would like to express my sincere gratitude to all of them.

First and foremost, I am highly obliged to my thesis supervisor Dr. Kalpesh Kapoor for allowing me to pursue my Ph.D. research work under his supervision. His invaluable support, friendliness, motivational and technical skills helped me throughout. I am very grateful to him for introducing me the flavors of stringology (design of string and sequence processing algorithms) and pattern matching. Under his valuable guidance and advice, I have got exposure, not only in the area related to my research work but also other perspectives.

I would like to take this opportunity to thank selection committee of HERITAGE Erasmus Mundus Action 2 Project for awarding me scholarship under Ph.D. exchange mobility, to travel and carry out part of my research work at CTU in Prague for six months. I am obliged to Heritage coordinators Prof. S. K. Bose at IITG and Ing. Volfgang Melecký at CTU in Prague for their full support.

During my work at CTU in Prague, I had the distinct honor of working with Prof. Jan Holub that I will cheer for rest of my life. His great supervision made Chapter 5 possible. Also, I would like to thank Dr. Jan Janoušek, head of the department of Theoretical Computer Science, Faculty of Information Technology for providing me a quiet office in his department during my working period at CTU in Prague.

I also want to convey my sincere gratitude to my doctoral committee members Prof. Diganta Goswami, Dr. Partha Sarathi Mandal and Dr. H. Ramesh for reviewing my research work and giving valuable suggestions which improved my research work. I am blessed to sit in some of the basic courses of Computer Science taught by Dr. R. Inkulu, Dr. Deepanjan Kesh and Prof. Sukanta Pati which inspired me to enrich my basic Computer Science knowledge and teaching skills. The interesting feedback, useful suggestions, information and support from Dr. Vinay Wagh, Dr. Siddhartha P. Chakrabarty, Dr. Shreemayee Bora, Dr. K. V. Srikanth and Dr. Anjan Kr. Chakrabarty has been invaluable to me on both academic and personal level, for which I am extremely thankful to them.

I sincerely acknowledge IITG for providing me various facilities necessary to perform my research work and to stay safe and healthy in this beautiful

campus. I am most grateful to Ministry of Human Resource Development (MHRD), Government of India, for providing me financial assistance through the institute, without which it was impossible to complete this thesis work.

I would like to thank my friends with whom I spent the magic moments during last five years at IITG. I am also thankful to all the departmental staff, and all my research scholar colleagues, specially Barun, Himadri, Punit, Saloni, Manideepa, Dinesh, Dishari, Ramesh and Ananda, for their love and cooperation during my research. I also thank Rahul, particularly for assisting me in the implementation of graph matching algorithms.

Finally, my deepest love and respect to my family, especially my parents and elder brother, for their love and support, sacrifice and consistent interest about the state of my studies over the years.

August 2016 **Shibsankar Das**

Abstract

This thesis investigates problems concerned with Approximate Parameterized String Matching (APSM) under Hamming distance error model and related problems in graph theory. We introduce a term called error class in APSM problem and explore various combinatorial properties of the error classes. We also provide a tight lower bound on the weights of Maximum Weight Bipartite Matching (MWBM) of graphs which is correlated with the counting of number of error classes in APSM problem. The problem of APSM for a pair of equal length strings under Hamming distance is computationally equivalent to MWBM problem in graph theory. Let $G = (V, E, Wt)$ be an undirected, weighted bipartite graph where V be the vertex set and E be the edge set of G with positive integer weights on the edges which are given by the weight function $Wt\colon E \rightarrow \mathbb{N}$. We fine-tune the existing decomposition theorem, originally proposed by Kao et al., for computing a MWBM of G. It is used to design a modified deterministic algorithm to compute weight of a MWBM of G in $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time, where $k(x, y) = \log x / \log(x^2/y)$ and $|E| \leq W' \leq W/g \leq W$, g denotes the GCD of the positive edges weights $\{w_1, w_2, \ldots, w_{|E|}\}$ of G. This modified decomposition technique is used as a subroutine in the following three proposed solutions. (i) An $O(n_P\, n_T\, m)$-time algorithm for all pairs approximate parameterized string matching problem with error threshold k, among two sets P and T of m-length strings over two distinct alphabets, where n_P and n_T are the cardinality of P and T, respectively. We introduce Parikh vector based filtering technique in order to preprocess the given sets of strings to avoid the comparison of non-candidate pairs. (ii) An $O(nm)$-time algorithm for APSM problem under weighted Hamming distance for a pattern $p \in \Sigma_P^m$ in a text $t \in \Sigma_T^n$. And (iii) an $O(m+k)$-time solution for parameterized string matching problem with k mismatches between a pair of m-length strings over two distinct alphabets under weighted Hamming distance. Experimental evaluation of some of the proposed techniques is also done and it is observed that the suggested techniques are efficient.

List of Abbreviations

APAPSM	All Pairs Approximate Parameterized String Matching
APSM	Approximate Parameterized String Matching
ASCII	American Standard Code for Information Interchange
ASM	Approximate String Matching
DAWG	Directed Acyclic Word Graph
DFA	Deterministic Finite Automaton
EC	Error Class
ESM	Exact String Matching
HD	Hamming Distance
KMP	Knuth-Morris-Pratt
LD	Levenshtein Distance
MCM	Maximum Cardinality Matching
MWBM	Maximum Weight Bipartite Matching
MWC	Minimum Weight Cover
NEECs	Non-trivial Empty Error Classes
NPV	Normalized Parikh Vector
PAPSM	Pairs Approximate Parameterized String Matching
PSM	Parameterized String Matching
PV	Parikh Vector
RNA	Ribo-Nucleic Acid
SCP	String Comparison Problem
smart	string matching algorithms research tool
TEECs	Trivial Empty Error Classes
WAPSM	Weighted Approximate Parameterized String Matching
WHD	Weighted Hamming Distance
WPSM	Weighted Parameterized String Matching

List of Symbols

$\mathbb{N}$	The set of natural numbers
$\mathbb{N}_0$	The set of all non-negative integers, that is, $\mathbb{N} \cup \{0\}$
$\mathbb{Q}$	The set of rational numbers
$\mathbb{Q}^+$	The set of all positive rational numbers
$\mathbb{Q}_0^+$	The set of all non-negative rational numbers, that is, $\mathbb{Q}^+ \cup \{0\}$
$\emptyset$	Empty set which contains no elements
ε	The empty string or null string
Σ^*	The set of all finite length strings over an alphabet Σ
Σ^+	A set of all non-empty strings over an alphabet Σ
Σ^m	The set of all strings of length m over an alphabet Σ
$s[i..j]$	The substring of a string s that starts and ends at the indices i and j, respectively. That is, $s[i..j] = s[i]s[i+1]\ldots s[j]$
$ESM(p,t)$	Exact string matching of a pattern $p \in \Sigma^m$ in the text $t \in \Sigma^n$
$u \mathrel{\widehat{=}} v$	The string $u = u[1..m] \in \Sigma_u^m$ is *parameterized match* or *p-match* with $v \in \Sigma_v^m$. That is, there exists a bijection π: $\Sigma_u \to \Sigma_v$ such that $\pi(u) = \pi(u[1])\pi(u[2])\ldots\pi(u[m]) = v$
$PSM(p,t)$	Parameterized string matching of a pattern $p \in \Sigma_p^m$ in the text $t \in \Sigma_t^n$
d_H: $\Sigma^* \times \Sigma^* \to \mathbb{N}_0$	Hamming distance function
$ASM(p,t)$	Approximate string matching of a pattern $p \in \Sigma^m$ in the text $t \in \Sigma^n$
$ASM(p,t,k)$	Approximate string matching of a pattern $p \in \Sigma^m$ in the text $t \in \Sigma^n$ with up to k errors
(u,v)	A pair of equal length strings where $u \in \Sigma_u$ and $v \in \Sigma_v$
π-$mismatch(u,v)$	$d_H(\pi(u), v)$

$APSMe(u, v)$	Approximate parameterized string matching between u and v, that is, $\{\pi \mid d_H(\pi(u), v)$ is minimum over all bijections π: $\Sigma_u \to \Sigma_v\}$
$cost(APSMe(u, v))$	It is equals to $d_H(\pi(u), v)$ where $\pi \in APSMe(u, v)$
$APSM(p, t)$	Approximate parameterized string matching of $p \in \Sigma_P^m$ in $t \in \Sigma_T^n$
$PSMe(u, v, k)$	PSM between a pair of equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$ with k mismatches, that is, $\{\pi \mid d_H(\pi(u), v) \leq k\}$
$PSM(p, t, k)$	Parameterized string matching with k mismatches of a pattern $p \in \Sigma_p^m$ in the text $t \in \Sigma_t^n$
D_k	An error class in APSM problem. A pair of equal length strings $(u, v) \in D_k$ if and only if $d_H(\pi(u), v) = k$ where $\pi \in APSMe(u, v)$
$\mathcal{D}$	The collection of all error classes in the APSM problem, that is, $\{D_0, D_1, D_2, \ldots\}$
$G = (V, E, Wt)$	G be an undirected, weighted bipartite graph where V the vertex set of G and E is the edge set of G with positive integer weights on the edges which are given by the weight function Wt: $E \to \mathbb{N}$
Wt	Weight function from E of G to $\mathbb{N}$
$mm(G)$	Maximum cardinality matching of the graph G
$mwm(G)$	Maximum weight matching of the graph G
W	Total weight of the graph $G = (V, E, Wt)$, that is, $W = Wt(G) = \sum_{e \in E} Wt(e)$
N	The largest weight of any edge in graph G
$K_{p,q}$	A complete bipartite graph with vertex partition size p and q.
$\mathcal{G}_{m \geq \sigma}$ or $\mathcal{G}$	$\{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = \lvert\Sigma_P\rvert = \lvert\Sigma_T\rvert, m = Wt(G), m \geq \sigma\}$
$k(x, y)$	This term is equal to $\log x / \log(x^2/y)$
$\psi(w)$	Parikh vector of a string w
$\widehat{\psi}(w)$	Normalized Parikh vector of a string w
d_{WH}: $\Sigma^* \times \Sigma^* \to \mathbb{N}_0$	Weighted Hamming distance function
π-$wmismatch(u, v)$	$d_{WH}(\pi(u), v)$
$WAPSMe(u, v)$	Weighted approximate parameterized string matching between $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$
$WAPSM(p, t)$	Weighted approximate parameterized string matching of a pattern $p \in \Sigma_P^m$ in text $t \in \Sigma_T^n$

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview and Motivation

String matching is an extensively studied topic which explores ways to find the location of a specific pattern in a given text. The term 'stringology', originally coined by Zvi Galil in 1984 [50], refers to the study of algorithms and data structures for strings and sequences. This has wide range of applications in different areas of information technology such as in text-editing programs, search engines and searching for patterns in a DNA sequence. The more details about these applications can be found in standard books [26, 34, 36, 53, 100] on string algorithms and analysis.

In last fifty years we have seen a rapid growth in the capability of computing hardware. This has also been predicted by Moore's law which states that the number of transistors on a chip doubles in every eighteen months [85–87]. As a consequence the modern computers are able to store and process a large amount of data. This and several applications of string matching have motivated researchers to continue to improve and design efficient string matching algorithms that takes advantage of improvements in the speed and storage data structures of computational devices.

Let Σ be a finite alphabet and Σ^* be the set of all strings over Σ. Further, let p and t be a pattern of length m and a text of length n in Σ^*, respectively. The basic text algorithms such as Karp-Rabin [71], Knuth-Morris-Pratt [73] and Boyer-Moore [24] discovers all text locations $i \in [1..n-m+1]$ such that $t[i..(i+m-1)] = p[1..m]$, where the notation $s[i..j]$ is used to specify the substring of a string s that starts and ends at the indices i and j, respectively. Some of the variations of string matching problems are *exact string matching* [26, 43], *approximate string matching* [90], *parameterized string matching* [17, 19, 20] and *approximate parameterized string matching* [57].

Approximate string matching of a pattern in a text considers string matching problem with errors. This is useful in a context where either the pattern, the text or both may not be exact representation of the information that they represent. For example, the pattern *aba* can be said to appear at location 1 and 3 in the text *abbba* with an error of one unit under Hamming distance.

In parameterized string matching the pattern and the text are over two distinct alphabets. The objective of parameterized string matching problem is to find the occurrence of a pattern by renaming (bijection) its alphabet in a given text [17]. Consider for instance the strings $u = 001$ and $v = aab$ over alphabets $\{0, 1\}$ and $\{a, b\}$, respectively. Let π be a bijection from $\{0, 1\}$ to $\{a, b\}$ that maps 0 and 1 to a and b, respectively. Then the two strings u and v match exactly under the map π. Given a pattern $p \in \Sigma_p^*$ and a text $t \in \Sigma_t^*$, the approximate parameterized string matching problem requires to find at each location of t, a bijection $\pi\colon \Sigma_p \to \Sigma_t$ that maximizes the number of characters that are mapped from p to the appropriate $|p|$-length substring of t, at that location. This problem has applications in software maintenance [16], image processing [13] and computational biology [5].

The string matching problem has been explored in different contexts. A broad classification of techniques is into indexed and non-indexed versions. In former case, it is allowed to preprocess string (either the pattern or the text or both) before searching for the pattern in a text. The aim of preprocessing is to improve the efficiency of a search procedure. If the pattern is given in advance, string matching algorithm uses pattern matching automata or some combinatorial properties of the given pattern to solve the problem. This kind of problem is commonly referred to as *online* string matching. In case the text is available first, a factor automata or tree index method is used to preprocess the text and solve string matching problem. In literature this kind of problem is commonly known as *offline* string matching [26, 43].

It turns out that finding a maximum weight matching in a bipartite graph is computationally equivalent to approximate parameterized string matching for a pair of pattern and text of equal length [57]. This thesis investigates problems in stringology and related problems in graph theory. In particular, following are the main goals of the thesis.

- Explore combinatorial properties of error classes in Approximate Parameterized String Matching (APSM) problem under Hamming distance.

- Provide a tight lower bound on the weights of Maximum Weight Bipartite Matching (MWBM) of bipartite graphs which is correlated with the

counting of number of error classes in APSM problem under Hamming distance.

- Improve existing deterministic algorithm for computing a MWBM of a bipartite graph.
- Present a filtering algorithm to solve all pairs approximate parameterized string matching problem.
- Design an algorithm that solves APSM problem under weighted Hamming distance model.
- Present an algorithm to solve the parameterized string matching problem with k mismatches under weighted Hamming distance, and
- Experimentally evaluate some of the designed techniques.

1.2 Organization of the Thesis

This thesis consists of seven chapters. Apart from the required introductory chapters, Chapters 3–6 contain the solution to the problems mentioned at the end of previous section. A list of abbreviations and mathematical symbols which are used in this thesis are provided on pages (xiii) and (xv), respectively. In addition, Appendix A contains some more information that is important but is not the main result of the thesis.

Chapter 2 *Basics of Approximate and Parameterized String Matching*

This chapter introduce the notations and basic definitions in the string matching which are used in the rest of the thesis. Some traditional variants of string matching problems along with the complexity analysis of the well known algorithms that solve these problems are also included in this chapter.

Chapter 3 *Combinatorics of Error Classes in Approximate Parameterized String Matching*

Let u and v be two equal length strings defined over an alphabet Σ. Then the Hamming distance between u and v (denoted by $d_H(u, v)$) is defined as the number of locations where a letter in u is different from that in v [54].

Minimizing such mismatches is one of the several criteria studied under approximate string matching. Let us consider a pair of distinct alphabets Σ_P and Σ_T where $|\Sigma_P| = |\Sigma_T| = \sigma$. So, the cardinality of each of Σ_P^m and Σ_T^m is σ^m. And hence the cardinality of all possible unique (p, t) pairs of strings in $\Sigma_P^m \times \Sigma_T^m$ is σ^{2m}, where $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$. Moreover, the total number of possible bijections from $\Sigma_P \to \Sigma_T$ is $\sigma!$.

Let π: $\Sigma_P \to \Sigma_T$ be a bijection. Given a pair $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$, APSM problem between p and t under Hamming distance requires to find a π over all bijections from Σ_P to Σ_T such that $d_H(\pi(p), t)$ is minimum. Notationally,

$$APSMe(p, t) = \{\pi \mid d_H(\pi(p), t) \text{ is minimum over all } \pi\}.$$

Let $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$ be two strings, where $\sigma = |\Sigma_P| = |\Sigma_T|$. Further let $D_k \subseteq \Sigma_P^m \times \Sigma_T^m$, where $k \in \mathbb{N}_0$. Then we say that $(p, t) \in D_k$ if there exists a π over all possible bijections from Σ_P to Σ_T such that $k = d_H(\pi(p), t)$ is minimum. Basically, for any pair (p, t) of $\Sigma_P^m \times \Sigma_T^m$ belongs to the set D_k if and only if $d_H(\pi(p), t) = k$ where $\pi \in APSMe(p, t)$. We call D_k as an *Error Class* (EC) set for APSM.

In this chapter we discuss the preliminaries of maximum matching in bipartite graph and describe the reduction between the APSM and the MWBM problems. We provide a tight lower bound on the weights of MWBM of graphs which is useful in the proof of counting the number of error classes in APSM problem. Several combinatorial properties of these classes, such as distribution of empty and non-empty ECs, exact count of non-empty ECs and divisibility property of the size of ECs in APSM are also presented.

Chapter 4 *Fine-Tuning Decomposition Theorem for Maximum Weight Bipartite Matching*

The problem of APSM for a pair of equal length strings under Hamming distance is computationally equivalent to MWBM problem in graph theory, due to both way reduction between the problems [57]. In this chapter we focus on computation of maximum weight matching in a bipartite graph using an exact algorithm. The proposed algorithm for computing MWBM will be used while solving the problems addressed later in Chapters 5 and 6.

Let $G = (V, E, Wt)$ be an undirected, weighted bipartite graph where V be the vertex set and E be the edge set of G with positive integer weights on the edges which are given by the weight function Wt: $E \to \mathbb{N}$.

We refine the existing decomposition theorem originally proposed by Kao et al. [70] for computing the weight of a maximum weight bipartite matching. We apply it to design an efficient version of the decomposition algorithm to compute the weight of a maximum weight bipartite matching of G

in $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time by employing an algorithm designed by Feder and Motwani [45] as a subroutine. The parameter W' is smaller than the total weight W of G, essentially when the largest edge weight differs by more than one from the second largest edge weight in the current working graph in any decomposition step of the algorithm. In best case $W' = |E|$ and in worst case $W' = W$, that is, $|E| \leq W' \leq W$. Further, we give a scaling property of the algorithm and a better bound of the parameter W' as $|E| \leq W' \leq \frac{W}{GCD(w_1,w_2,\ldots,w_{|E|})} \leq W$, where $GCD(w_1, w_2, \ldots, w_{|E|})$ denotes the GCD of the positive edges weights $\{w_1, w_2, \ldots, w_{|E|}\}$ of the weighted bipartite graph. The experimental evaluation of the proposed improvement on randomly generated bipartite graphs shows that the modified decomposition technique performs well in general.

Chapter 5 *All Pairs Approximate Parameterized String Matching*

This chapter deals with all pairs approximate parameterized string matching problem under Hamming distance with an error threshold, among two sets of equal length strings. Let $P = \{p_1,\ p_2, \ldots,\ p_{n_P}\} \subseteq \Sigma_P^m$ and $T = \{t_1,\ t_2, \ldots, t_{n_T}\} \subseteq \Sigma_T^m$ be two sets of strings where $|\Sigma_P| = |\Sigma_T|$. For each $p_i \in P$, we seek to find $t_j \in T$ which is approximately parameterized closest to p_i within a given threshold $k \in \mathbb{N}_0$. We give a solution with time complexity $O(n_P\, n_T\, m)$. Further, we introduce Parikh vector [92] based filtering technique, which we call as PV-filtering, in order to preprocess the given strings and avoid the comparison between the strings in non-candidate pairs for APSM. The PV-filtering does not change the asymptotic time complexity but the filter is effective for small error threshold as shown by the experiments.

Chapter 6 *Approximate Parameterized String Matching under Weighted Hamming Distance*

Here we consider the APSM problem under Weighted Hamming Distance (WHD), with a pattern $p \in \Sigma_P^m$ and a text $t \in \Sigma_T^n$ as inputs. The solution is essentially based on reduction of the APSM problem between two equal length strings under WHD to the MWBM problem. Our main result is an $O(nm)$-time algorithm for this problem, by using the MWBM algorithm proposed in Chapter 4.

Further, we investigate parameterized string matching problem with k mismatches between a pair of m-length strings under WHD. We propose an

$O(m + k)$-time solution for this problem. All the above time complexities assume constant size alphabets.

Chapter 7 *Conclusions and Future Work*

The final chapter gives an overview of the contributions of the thesis. We also state some unsolved questions in context of the problems addressed in this thesis, which might be of interest for further research in future.

Chapter 2

Basics of Approximate and Parameterized String Matching

This chapter presents the basic definitions of string matching, which are used in the rest of this thesis. It also contains some traditional variants of string matching problems along with a literature survey of the well known algorithms.

2.1 Preliminaries and Definitions

Alphabet

An *alphabet* is a non-empty finite set whose elements are called *symbols* (or *characters* or *letters*) [34]. Although an alphabet Σ could be of any collection of symbols, but usually the symbols are drawn from the well-known ASCII characters. For example: $\Sigma = \{0, 1\}$ (binary alphabet), $\Sigma = \{A, C, G, T\}$ (DNA alphabet[1]), $\Sigma = \{A, C, G, U\}$ (RNA alphabet), $\Sigma = \{a, b, \ldots, z\}$ (Roman or Latin alphabet) etc.

The alphabet can be totally ordered (so that the outcome of a comparison between two distinct symbols is "less" or "greater"), partially ordered or unordered [100].

String

A *string* over a given alphabet is a finite sequence of symbols. The *length* of a string w over a given alphabet Σ is the total number of symbols contained

[1]Recently a group of scientists led by Professor Floyd Romesberg of Romesberg Lab of Chemical Biology and Biophysics at the Scripps Research Institute in California, added two new letters to the DNA's alphabet: $X = \mathrm{d}\mathbf{NaM}$ and $Y = \mathrm{d}\mathbf{5SICS}$ [1, 84]

in w and is denoted by $|w|$. The string without any symbol over an alphabet Σ is called the *empty string* or *null string* and is denoted by ε; so $|\varepsilon| = 0$. We use Σ^* to denote the set of all finite length strings over Σ. This set is infinite and countable. A *set of all non-empty strings* over an alphabet Σ is denoted by Σ^+. Therefore, $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For any $m \in \mathbb{N}_0$, Σ^m denotes the set of all strings of length m over Σ.

Concatenation of Strings

For two strings u and v, the *concatenation* of u and v is a string formed by making a copy of u followed by a copy of v and is denoted as $u \cdot v$ (or simply as uv). This operation is associative but not commutative, that is, for any $u, v, w \in \Sigma^*$, $(uv)w = u(vw)$ and for any $u, v \in \Sigma^+$, $uv \neq vu$. Also, for any string $u \in \Sigma^*$, $u\varepsilon = \varepsilon u = u$.

Substring, Suffix, Prefix, Occurrence of a String

Let $w = xyz$ be a string where $x, y, z \in \Sigma^*$. We call y as a *substring* of the string w. If $x = \varepsilon$ then y is a *prefix* of w. If $z = \varepsilon$ then y is a *suffix* of w. The i-th symbol of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$. We denote a substring y of string w by $w[i..j]$ if y starts at the position i and ends at the position j for $1 \leq i \leq j \leq |w|$, that is, $y = w[i..j] = w[i]w[i+1] \ldots w[j]$. In that case, we also say that there is an *occurrence* of y at position i of w.

String Matching Problem

Let $t = t[1..n]$ (often called a *text*) and $p = p[1..m]$ (often called a *pattern*) be arrays of length n and m ($\ll n$), whose elements are drawn from finite alphabets Σ_t and Σ_p, respectively. The *string matching problem* is to find one, or more generally, all suitable occurrence of the pattern p in text t.

2.2 Assumptions

We assume the alphabet to be subset of ASCII codes. The standard random access machine model is used as the model of computation to study and evaluate the complexity of our algorithms. We assume that the comparison between two symbols is a constant time elementary operation. Throughout this thesis, whenever we consider a bijection from one alphabet to another alphabet, it is assumed that both the alphabets have the same number of letters.

2.3 Exact String Matching

Exact String Matching (ESM) is of fundamental importance and has direct applications in vast areas such as text, image and signal processing, speech analysis and recognition, informational retrieval, data compression, computational biology and chemistry [43]. Due to this reason researchers have made extensive study in the past decades to design efficient algorithms for solving ESM problem. The problem requires both the alphabets Σ_t and Σ_p to be the same. Let $p = p[1..m]$ and $t[1..n]$ are the pattern and text over an alphabet $\Sigma = \Sigma_t = \Sigma_p$.

Problem 2.1 (Exact String Matching)**.** *The exact string matching problem is to find all locations at which the specific pattern p occurs in the text t. The set of such locations is denoted by ESM(p, t). Formally,*

$$ESM(p,t) = \{i \mid t[i..i+m-1] = p[1..m], \text{ that is, } t[i+j-1] = p[j] \text{ for } 1 \le j \le m \text{ and } 1 \le i \le n-m+1\}.$$

Example 2.2. Let $\Sigma = \{A, C, G, T\}$ be an alphabet set. Further, consider a pattern p and a text t over the alphabet Σ as shown in Figure 2.1.

$p : AACTA$

$t : AAGTAACTAGTAACTAACTACCTAGTG$

Locations: 5 12 16

Figure 2.1: Exact string matching of a pattern p in a text t.

The pattern p occurs in t starting at locations 5, 12 and 16. Therefore, $ESM(p, t) = \{5, 12, 16\}$. Observe that occurrences of pattern in text may be overlapped such as at the locations 12 and 16 in Figure 2.1. □

Some Existing Algorithms

There are several algorithms that solve the ESM problem and each has its own advantages and disadvantages. See [26, 43] for a detailed survey of numerous ESM algorithms designed over the decades.

The naive brute force algorithm does not have a preprocessing phase. During the searching phase, the method first aligns the left end of p with the left end of t. Then it compares the characters of p and t from left to right until either two unequal characters are found, in which case we say

a mismatch has occurred, or until p is exhausted, in which case we have a complete match of p and an occurrence of p is reported. The procedure repeats the same process of comparison again after shifting p to the right until the right end of p goes beyond the right end of the text t. In worst case, the brute force algorithm finds all the occurrence of p in t in $\Theta(nm)$ time. This algorithm is inefficient because the information gained about the text for a shift is completely ignored in the next coming shifts. Such information can be very useful to increase the efficiency of searching.

Table 2.1: Efficiency comparison of some the well known exact string matching algorithms.

Algorithm	**Preprocessing Time**	**Space Required**	**Matching Time**
Naive brute-force	–	–	$O((n-m+1)m)$
Karp-Rabin [71]	$O(m)$	$O(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\Sigma\|)$	$O(m\|\Sigma\|)$	$O(n)$
Knuth-Morris-Pratt [73]	$O(m)$	$O(m)$	$O(n)$

Searching time of Karp-Rabin algorithm [71] is $O((n-m+1)m)$. Although it is not better than the naive algorithm, but in practice it works much better on average. The worst-case lower bound of this problem is $O(n)$. The first algorithm to reach that bound has been given by Morris and Pratt in [88] and later a tight analysis of the Morris-Pratt algorithm has been given by Knuth et al. [73] by improving the length of the shifts. See the Table 2.1.

The standard *string matching algorithms research tool* (also known as *smart*) [44] provide users to test, design, evaluate and understand comprehensive collections of existing string matching algorithms for exact string matching problem and it includes a wide corpus of text buffers.

2.4 Parameterized String Matching

The objective of Parameterized String Matching (PSM) is to find a parameterized occurrence of a pattern in a given text, by discovering an appropriate bijective mapping of an alphabet of a pattern to an alphabet of a text. This problem has applications in software maintenance [16, 17, 20], plagiarism detection [47], image processing [13, 101] and computational biology [5] including RNA structural matching [99] and thus it has been studied extensively over the last couple of decades [6, 10, 11, 18, 31, 39, 57, 58, 61–64, 94]. As mentioned earlier, in the following definitions we assume the alphabet sizes to be identical whenever we consider consider two distinct alphabets.

Problem 2.3 (PSM between a Pair of Equal Length Strings)**.** *Let u and v be two equal length strings over alphabets Σ_u and Σ_v, respectively. The string $u = u[1..m]$ is said to be a parameterized match or p-match with v (denoted as $u \widehat{=} v$) if there exists a bijection π: $\Sigma_u \to \Sigma_v$ such that*

$$\pi(u) = \pi(u[1])\pi(u[2]) \dots \pi(u[m]) = v,$$

that is, $\pi(u)$ is obtained by renaming each character of u using π.

Problem 2.4 (PSM for a Pattern in a Text)**.** *For a given input pattern $p \in \Sigma_p^m$ and a text $t \in \Sigma_t^n$, PSM problem is to find all locations in text t for each of which the pattern p parameterized matches to the substring of length m beginning at that text location. PSM(p, t) denotes the set of such locations. Formally,*

$$PSM(p, t) = \{i \mid p \widehat{=} t[i..i + m - 1],\ 1 \leq i \leq n - m + 1\}.$$

Example 2.5. Consider the alphabets $\Sigma_p = \{A, C, G, T\}$ and $\Sigma_t = \{1, 2, 3, 4\}$. Let $p \in \Sigma_p^*$ and $t \in \Sigma_t^*$ be the pattern and the text, respectively, as shown in Figure 2.2.

$$p : AACTA$$

$$t : 113411241341124112412241343$$

Locations: 1 5 12 16

Figure 2.2: Parameterized string matching of a pattern p in a text t.

The pattern $p = AACTA$ parameterized matches with the substring 11341 at the text location 1 under under $\pi' = \{A \to 1, C \to 3, G \to 2, T \to 4\}$ and also with the substring 11241 at the text locations 5, 12 and 16 under the bijection $\pi = \{A \to 1, C \to 2, G \to 3, T \to 4\}$. Therefore $PSM(p, t) = \{1, 5, 12, 16\}$. □

Some Existing Algorithms

The PSM problem was originally introduced by Brenda S. Baker in [17] to detect duplicate code in large software systems as an aid in software maintenance [16]. A linear time algorithm has been given in [17, 20] for a constant size alphabet. For unbounded alphabet, an optimal $O(n \log \min(m, |\Pi|))$ time exact parameterized matching algorithm has been given in [6] based on

Knuth-Morris-Pratt algorithm [73], where m is the length of the pattern, n is the length of the text and Π is a parameterized alphabet.

The parameterized matching problem has been solved in [19, 20] by constructing parameterized suffix trees. This method is also applicable for online parameterized matching. Here a parameterized suffix tree is constructed by converting a pattern into a predecessor string. The parameterized suffix tree has been further studied in [30, 80, 94].

Fredriksson and Mozgovoy [47] have proposed sublinear algorithms for one-dimensional parameterized matching using Shift-Or [14] and backward DAWG matching [33] algorithms. Further, a new sublinear algorithms has been proposed for both one-dimensional and two-dimensional parameterized string matching with q-grams [95].

Idury and Schäffer [65] considered a generalization of the standard p-string matching, namely, the dictionary matching under the parameterized pattern matching model. They have used a modified Aho-Corasick automaton [3]. Amir and Navarro [8] have investigated parameterized matching in hypertext. They have proposed an optimal algorithm for pattern matching where the text is a tree (non-linear structure), with running time $O(n \log \min(m, |\Sigma|))$ where n is the tree size, m is the length of a pattern and Σ consists of text node labels and pattern symbols.

2.5 Distance Function in String Matching

Let d: $\Sigma^* \times \Sigma^* \rightarrow \mathbb{N}_0$ be a non-negative *distance function*. The *distance* $d(x, y)$ between two strings $x = x[1..n] \in \Sigma^*$ and $y = y[1..m] \in \Sigma^*$ is the minimal cost of a sequence of edit operations that transform x into y (and ∞ if no such sequence exists). The cost of a sequence of operations is the sum of the costs of the individual operations. In general, the set of possible edit operations on a string $x[1..n]$ are:

(a) *Insertion*: Insert the character a into the string x at the i-th position, that is,

$$\mathit{insert}(x, i, a) = x[1..i-1] \cdot a \cdot x[i..n];$$

(b) *Deletion*: Delete the character from the position i of the string x, that is,

$$\mathit{delete}(x, i) = x[1..i-1] \cdot x[i+1..n];$$

(c) *Replacement* (or *Substitution*): Replace the character at the position i in x with the new character a, that is,

$$\mathit{replace}(x, i, a) = x[1..i-1] \cdot a \cdot x[i+1..n];$$

(d) *Transposition* (or *Swap*): For $x[i] \neq x[i+1]$, swap the adjacent letters $x[i]$ and $x[i+1]$. Each letter can take part in at most one transpose. That is,

$$transpose(x, i) = x[1..i-1] \cdot x[i+1] \cdot x[i] \cdot x[i+2..n].$$

Some of the standard distance functions are discussed below:

Levenshtein Distance (LD): This distance function is applicable to arbitrary length strings. The error model *Levenshtein distance* [81, 82], denoted as d_L, allows insertions, deletions and replacements, all operations cost one unit in general. Levenshtein distance is also known as *edit distance*.

Hamming Distance (HD): It is restricted to equal length strings. The *Hamming distance*, denoted as d_H, allows only replacements [54, 55] at unit cost.

Damerau Distance: *Damerau distance* allows insertion, deletion, replacement and transposition (of the neighbour symbols) [37]. It is also known as *Damerau-Levenshtein distance*.

Other Variants: Other variations of operations are also possible which lead to some more error models, such as *extended edit distance* (insertion, deletion, replacement and swap) [102, 103], *LCS distance*[2] (insertion, deletion) [12, 91], δ-distance [25], γ-distance [25], $\delta\gamma$-distance [25], episode distance (insertion) [38]. A more general definition associates non-negative weight functions with the operations [60].

Example 2.6. Let $x = GTATC$, $y = TGAT$, $z = GTATT$ be the strings over the DNA alphabet $\{A, C, G, T\}$. Thus $d_L(x, y) = d_L(GTATC, TGAT) = 2$ and $d_H(x, z) = d_H(GTATC, GTATT) = 1$. □

Different error models based on a metric (distance function) measure the similarity between two different strings. There are various kinds of distance measures for different dimensions which are described broadly in the second and third chapter ('The Metric Space' and 'Distance Measures' respectively) of the book entitled '*Similarity Search: The Metric Space Approach*' [105]. In [90], Gonzalo Navarro has made a survey of traditional error models for approximate string matching problem. Two of the most classical distance metrics are Levenshtein distance and Hamming distance.

[2] In molecular biology, the term *indel* is used for the same.

2.6 Approximate String Matching

In Approximate String Matching (ASM) problem, both the alphabet sets Σ_t and Σ_p of text t and pattern p, respectively, are the same. The ASM problem considers the string matching problem with errors. In general, the target is to match a pattern in a text where one or both of them have suffered from some undesirable error [90]. It has applications in several fields such as signal processing [41,81], information retrieval [15], computational biology [53], image compression [83], data mining [38].

In literature several interpretation of approximation on strings have been considered, such as *jokers* (also sometimes called *wild-card* or *don't cares*) [28, 29], *differences*, *mismatches*. Here we use the distance functions (discussed in Section 2.5) for measuring the approximation.

While considering Levenshtein distance as a measure of approximation, ASM problem can be restated as the minimal number of insertions, deletions, and replacement to transform x into y. In the literature, the search problem in many cases is called "string matching with differences". Whereas for Hamming distance, the search problem in many cases is called "string matching with mismatches". Therefore under the distance measure, the *ASM problem* becomes minimizing the total cost to transform the pattern and its occurrence in text to make them equal, and find the text positions where this cost is minimum

Since in this thesis we investigate the problems related to approximate parameterized string matching under Hamming distance and its weighted variant, so the following problem statements of ASM and its variants are considered with HD only, for equal length strings. Let $p \in \Sigma^m$, $t \in \Sigma^n$ where $m \geq n$ and $k \in \mathbb{N}_0$ be the maximum error allowed.

Problem 2.7 (Approximate String Matching without Error Threshold)**.** *ASM problem without any error threshold seeks to find all the locations in the text t, for each of which the total cost to transform the pattern p into the $|p|$-length substring beginning at that text location is minimum. Let us denote the set of such locations by $ASM(p,t)$. In other words,*

$$ASM(p,t) = \{i \mid d_H(p, t[i..i+m-1]) \text{ is minimum, where } 1 \leq i \leq n-m+1\}.$$

Problem 2.8 (Approximate String Matching with k Error Threshold)**.** *In ASM problem with error threshold k we need to find all the locations in the text t, where $|p|$-length text match the pattern with up to k errors. $ASM(p,t,k)$ denotes the set of such locations. Formally,*

$$ASM(p,t,k) = \{i \mid d_H(p, t[i..i+m-1]) \leq k,\ 1 \leq i \leq n-m+1\}.$$

In case, no such $|p|$-length substring of t is found to match with p under the k threshold then $ASM(p, t, k) = \emptyset$.

Example 2.9. Let us consider a pattern p and a text t over a given alphabet $\Sigma = \{A, C, G, T\}$ as shown in the Figure 2.3.

$p : AACTA$

$t : AAGTAACTAGTAACTAACTACCTAGTG$

Locations: 1 20

Figure 2.3: Approximate string matching of a pattern p in a text t.

Therefore, $ASM(p, t, 0) = \{5, 12, 16\}$ because at each of these locations of the text the Hamming distance between p and 5-length substring of t is minimum, which is 0 in this example. However, $ASM(p, t, 1) = \{1, 5, 12, 16, 20\}$ where the error threshold k is equal to 1; because at these text locations $d_H(p, t[i..i + 4]) \leq 1$. □

Some Existing Algorithms

One can find all the locations using the methods of Landau and Vishkin [77] or Galil and Giancarlo [51] where the pattern has at most k mismatches in $O(kn)$ time. The Abrahamson's algorithm [2] finds the number of mismatches at every location of text in $O(n\sqrt{m \log m})$ time which is independent of k. Observe that if $k < \sqrt{m \log m}$ then it is good to use former algorithms of [51, 77]; else if k is more than $\sqrt{m \log m}$, then Abrahamson's algorithm is better than [51, 77]. Further, another faster algorithm for ASM problem with at most k mismatches was presented in [7] having worst-case time complexity $O(n\sqrt{k \log k})$. Although in worst case $k = O(m)$, but in practice k might be very small. Recently, Grabowski and Fredriksson [52] refined the $O(n\lceil m \log(k)/w\rceil)$-time Shift-And algorithm [14] to $O(n\lceil m \log \log(k)/w\rceil)$ and $O(n\lceil m/w\rceil)$, where w is the number of bits in a computer word.

This problem has also been studied under edit distance. An $O(nm)$-time dynamic programming algorithm was proposed by Lowrance and Wagner [103] for extended edit distance. For the edit distance with k threshold, $O(kn)$-time algorithms are given in [76, 78]. Also, for this problem there exist $O(mn)$-time [98], $O(n + mn/\log_\sigma n)$-time [35] solutions under weighted edit distance.

2.7 Approximate Parameterized String Matching

The Approximate Parameterized String Matching (APSM) problem is a well studied problem [9, 10, 57, 79, 93] with a wide range of applications, such as in software duplication, image processing and computational biology. The problem with and without error threshold are given below.

Definition 2.10 (π-*mismatch* between $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$ [57])**.** *Given a pair of equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$ and a bijection π: $\Sigma_u \to \Sigma_v$, the π-mismatch between u and v is the Hamming distance between the image under π of u and v. Let us denote this by π-mismatch(u, v). Therefore,*

$$\pi\text{-}mismatch(u, v) = d_H(\pi(u), v).$$

Approximate Parameterized String Matching (without Error Threshold)

Problem 2.11 (APSM between a Pair of Equal Length Strings [57])**.** *The approximate parameterized string matching between a pair of equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$ is to find a π over all bijections π: $\Sigma_u \to \Sigma_v$ such that π-mismatch(u, v) is minimum. We use the notation APSMe(u, v) to denote the collection of such bijections. Formally,*

$$APSMe(u, v) = \{\pi \mid d_H(\pi(u), v) \text{ is minimum over all } \pi \text{ from } \Sigma_u \text{ to } \Sigma_v\}.$$

Note that, there are exponential number of possible bijections from Σ_u to Σ_v and such π for which $d_H(\pi(u), v)$ is minimum may not be unique. We define the *cost of APSMe*(u, v) as *cost*$(APSMe(u, v)) = d_H(\pi(u), v)$ where $\pi \in APSMe(u, v)$.

Problem 2.12 (APSM for a Pattern in a Text [57])**.** *Given a pattern $p \in \Sigma_p^*$ of length m and a text $t \in \Sigma_t^*$ of length n, the APSM for p in t consists of solving APSM between p and m-length substring of the text t, starting at each text location.*

In other words, the problem is to find bijection π: $\Sigma_p \to \Sigma_t$ at each location of t that maximizes the number of characters that are mapped from p to the appropriate $|p|$-length substring of t, at that text location. Let us denote the set of all such bijections by $APSM(p, t)$. Using notation,

$$APSM(p, t) = \{(\pi_i)_{i \in \{1,2,\ldots,n-m+1\}} \mid \pi_i \in APSMe(p, t[i..i+m-1]) \text{ where } i = 1, 2, \ldots, n-m+1\}.$$

For the case where $|p| = |t| = m$, this problem is called as *String Comparison Problem* (SCP) which is computationally equivalent to the Maximum Weight Bipartite Matching (MWBM) problem in graph. For a detailed both way reduction between the problems, please see the paper [57]. For the preliminaries of MWBM, please see the Section 3.2. The key idea of the reduction between the APSM problem for equal length strings and the MWBM problem is discussed in Section 3.3.

Parameterized String Matching with k Mismatches

Let us see the problem statement of the *parameterized string matching with k mismatches* [57]. We also call it *parameterized string matching with k threshold under Hamming distance*. The problem statement is given below.

Problem 2.13 (PSM between a Pair of Equal Length Strings with k Mismatches)**.** *Given $k \in \mathbb{N}_0$, a pair of equal strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$, the parameterized string matching with k mismatches between u and v seeks to find a bijection π: $\Sigma_u \to \Sigma_v$ such that π-mismatch$(u, v) \leq k$. We use PSMe(u, v, k) to denote the collection of such bijections. Therefore,*

$$PSMe(u, v, k) = \{\pi \mid d_H(\pi(u), v) \leq k\}.$$

In such case we say that u parameterized matches v with k threshold under HD.

In literature this problem is also known as *String Comparison Problem* (SCP) with the threshold k [57]. Observe the difference between Problems 2.11 and 2.13. In the former problem, a best π: $\Sigma_u \to \Sigma_v$ is desired; however in the later problem, any π: $\Sigma_u \to \Sigma_v$ with π-mismatch$(u, v) \leq k$ is acceptable. We define the *cost of PSMe(u, v, k)* as *cost*$(PSMe(u, v, k)) = d_H(\pi(u), v)$ where $\pi \in PSMe(u, v, k)$. If $PSMe(u, v, k) = \emptyset$ then the

$$cost(PSMe(u, v, k)) = \infty.$$

Problem 2.14 (PSM for a Pattern in a Text with k Mismatches)**.** *Given $k \in \mathbb{N}_0$, a pattern $p \in \Sigma_p^*$ of length m and text $t \in \Sigma_t^*$ of length n, the PSM with k mismatches for p in t consists of computing PSM with k mismatches between p and m-length substring of t, at each text location. The notation PSM(p, t, k) denotes the set of such bijections. Formally,*

$$PSM(p, t, k) = \{(\pi_i)_{i \in \{1,2,\ldots,n-m+1\}} \mid \pi_i \in PSMe(p, t[i..i+m-1], k) \\ where\ i = 1, 2, \ldots, n-m+1\}.$$

$p : AACTA$

$t : 1134112413411241124122413 43$

Locations: 1 5 12 20

Figure 2.4: Approximate parameterized string matching of a pattern p in a text t.

Example 2.15. Given the alphabet sets $\Sigma_p = \{A, C, G, T\}$ and $\Sigma_t = \{1, 2, 3, 4\}$, let us consider a pattern $p \in \Sigma_p^*$ and a text $t \in \Sigma_t^*$ as shown in Figure 2.4.

Let us consider the bijections $\pi = \{A \to 1, C \to 2, G \to 3, T \to 4\}$ and $\pi' = \{A \to 1, C \to 3, G \to 2, T \to 4\}$. Observe that $\pi(p) = \pi(AACTA) = 11241$ and $\pi'(p) = 11341$. Therefore, $PSM(p, t[1..10], 0) = \{(\pi', \emptyset, \emptyset, \emptyset, \pi, \emptyset)\}$. Also observe that $APSMe(p, t[1..5]) = \{\pi'\}$. Whereas, if we consider an error threshold k equal to 1, then $PSMe(p, t[1..5], 1) = \{\pi', \pi\}$. □

Some Existing Algorithms

Over the years, the PSM problem under several error distance variants have been studied, for example, edit distance [21, 93], Hamming distance [10, 57], Hamming distance for run-length encoded strings [9], LCS distance [66] and $\delta\gamma$-distance [79]. This problem is more realistic under Hamming distance as stated in [10] and can be naturally extended for the edit distance; but it is known to be NP-complete for the later [68, 72].

A solution of APSM for binary alphabets has been given in [10]. For general alphabets, when $|p| = |t| = m$, an $O(m^{1.5})$ algorithm has been presented for APSM problem by Hazay et al. in [57]. The solution uses MWBM algorithm proposed by Kao et al. [70], because of the reduction between the APSM problem for equal length strings and the MWBM problem. In the case of general text where the inputs are a text t of length n and a pattern p of length $m\,(\ll n)$, an $O(nm^{1.5})$-time solution was proposed by them to solve the APSM problem between p and t.

For the problem of PSM with k mismatches, Hazay et al. have presented an $O(m + k^{1.5})$-time algorithm when $|p| = |t| = m$, and an $O(nk^{1.5} + mk \log m)$-time algorithm where lengths of the text t and the pattern p are n and $m\,(\ll n)$, respectively.

Chapter 3

Combinatorics of Error Classes in Approximate Parameterized String Matching

In this chapter we investigate various combinatorial properties of the error classes in APSM problem for *all pairs of equal length strings*. Study of error classes in APSM problem is not addressed before in the literature of string matching.

3.1 Introduction

ASM of a pattern in a text considers string matching problem with errors where one or both of them have experienced some undesirable noise. Different error models have been considered depending on the different applications, in order to determine how erroneous or close are two strings.

Here we consider the Hamming distance [54, 55] as a measure of error, which allows only replacements with unit cost. Let $u_1, u_2 \in \Sigma_u^*$ and $v \in \Sigma_v^*$ be the equal length strings. Under this model, ASM problem between u_1 and u_2 is to minimize the total cost to transform u_1 into u_2.

The objective of the PSM problem [17,19,20] between u_1 and v is to search for a bijective mapping, π, from Σ_u to Σ_v such that $\pi(u_1) = v$. Therefore under Hamming distance error model, APSM problem between u_1 and v is to find a π over all bijections to minimize the total cost to transform $\pi(u_1)$ into v.

As stated in Problem 2.11 (see page 16), APSM problem for a pair of equal length strings under HD error model is computationally equivalent to the Maximum Weight Bipartite Matching (MWBM) problem in graph

theory, due to both way reductions between the problems. These reductions are proposed by Hazay et al. in [57].

Roadmap. Preliminaries of maximum matching in a bipartite graph is given in Section 3.2. Section 3.3 describes the reductions between the APSM problem under Hamming distance and the MWBM problem. We provide a tight lower bound on the weights of MWBM of graphs in Section 3.4 which is used in the proof of counting the number of error classes in APSM problem. In Section 3.5 we introduce the term Error Class (EC) in APSM. Several combinatorial properties of these classes, such as distribution of empty and non-empty ECs, exact count of non-empty ECs and divisibility property of ECs in APSM are presented in the Sections 3.6–3.9. An experimental result is given in Section 3.10 which enumerates the number of pairs in an EC. Finally, a summary of this chapter is given in Section 3.11.

3.2 Maximum Matching in a Bipartite Graph

Let $G = (V = V_1 \cup V_2, E, Wt)$ be an undirected, weighted bipartite graph where V_1 and V_2 are two non-empty partitions of the vertex set V of G, and E is the edge set of G with positive integer weights on the edges which are given by the weight function Wt: $E \to \mathbb{N}$. Let W denotes the total weight of G and is defined by $W = Wt(G) = \sum_{e \in E} Wt(e)$. For uniformity we treat an unweighted graph as a weighted graph having unit weight for all edges.

We use the notation $\{u, v\}$ for an edge $e \in E$ between $u \in V_1$ and $v \in V_2$, and its weight is denoted by $Wt(e) = Wt(u, v)$. We also say that $e = \{u, v\}$ is *incident* on vertices u and v; and u and v are each *incident* with e. Two vertices $u, v \in V$ of G are *adjacent* if there exists an edge $e = \{u, v\} \in E$ of G to which they are both incident. Two edges $e_1, e_2 \in E$ of G are *adjacent* if there exists a vertex $v \in V$ to which they are both incident.

A subset $M \subseteq E$ of edges is a *matching* if no two edges of M share a common vertex. A vertex $v \in V$ is said to be *covered* or *matched* by the matching M if it is incident with an edge of M; otherwise v is *unmatched* [22, 23].

A matching M of G is called a *maximum* (*cardinality*) *matching* if there does not exist any other matching of G with greater cardinality. We denote such a matching by $mm(G)$. The weight of a matching M is defined as $Wt(M) = \sum_{e \in M} Wt(e)$. A matching M of G is a *maximum weight matching*, denoted as $mwm(G)$, if $Wt(M) \geq Wt(M')$ for every other matching M' of the graph G.

Observe that, if G is an unweighted graph then $mwm(G)$ is a $mm(G)$, which we write as $mwm(G) = mm(G)$ in short and its weight is given by $Wt(mwm(G)) = |mm(G)|$. Similarly, if G is an undirected and weighted graph with $Wt(e) = c$ for all edges e in G and c is a constant then also we have $mwm(G) = mm(G)$ with weight of the matching as $Wt(mwm(G)) = c * |mm(G)|$.

In this chapter we require the idea of the reduction between the APSM problem under HD and the maximum weight matching problem in a bipartite graph.

3.3 Reductions between the APSM and MWBM Problems

Let us consider two strings $u = u_1u_2 \dots u_m$ and $v = v_1v_2 \dots v_m$ over alphabets $\Sigma_u = \{a_1, a_2, \dots, a_\sigma\}$ and $\Sigma_v = \{b_1, b_2, \dots, b_\sigma\}$, respectively. As mentioned in Problem 2.11, APSM problem between u and u under Hamming distance without any error threshold is to find a π: $\Sigma_u \to \Sigma_v$ over all bijections such that $d_H(\pi(u), v)$ is minimum. The reductions between this problem and the MWBM problem [57] is discussed below.

Reduction of APSM Problem under HD to MWBM Problem: Construct a bipartite graph $G = (U \cup V, E, Wt)$ where $U = \Sigma_u$ and $V = \Sigma_v$. We add an edge between the vertices $a_i \in U$ and $b_j \in V$ in G if and only if there is at least one alignment of a_i in string u with b_j in string v. Assign weight to this edge $\{a_i, b_j\}$ as the frequency of alignment of a_i in u with b_j in v. The following observation is the relation between the maximum weight matching in bipartite graph and minimum mismatch between strings.

Observation 3.1 ([57]). *A maximum weighted matching in G corresponds to a minimum π-mismatch, where π is defined by the edges of the matching.*

Therefore an algorithm for the MWBM problem can be used to solve the APSM problem between a pair of equal length strings. The reduction time is $O(m)$. See Example 3.2.

Reduction of MWBM Problem to APSM Problem under HD: Let $G = (U \cup V, E, Wt)$ be an undirected and weighted graph where $U = \{a_1, a_2, \dots, a_{|U|}\}$, $V = \{b_1, b_2, \dots, b_{|V|}\}$ and $E = \{e_1, e_2, \dots, e_{|E|}\}$. We create two strings u' and v' of length $Wt(G)$ as follows. For an edge $e = \{a_i, b_j\} \in E$,

we add a_i and b_j in u' and v', respectively, in any locations but maintaining the alignment of a_i and b_j for a total of $Wt(e)$ times.

Hence a bijection from Σ_u to Σ_v in the APSM problem for the strings u' and v' is a bijection from U to V which is a matching in G. The objective of the APSM problem is to minimize the mismatches, which are those edges that are not in the bijection, whereas in the MWBM problem, the objective is to maximize the number of edges in the bijection. This shows that the two objective functions are equivalent. Therefore an algorithm to solve the APSM problem can also be used for solving the MWBM problem. The reduction time is $O(|E| + |V|)$. See the following example.

Example 3.2. Let $u = a_1a_1a_2a_3a_2a_1a_1a_3a_1$ and $v = b_1b_2b_3b_3b_3b_4b_4b_3b_4$ be two strings over the alphabets $\Sigma_u = \{a_1, a_2, a_3, a_4\}$ and $\Sigma_v = \{b_1, b_2, b_3, b_4\}$, respectively. The corresponding weighted bipartite graph G is shown in Figure 3.1. In G a maximum weight matching is $mwm(G) = \{\{a_1, b_4\}, \{a_3, b_3\}\}$ and $Wt(mwm(G)) = 5$.

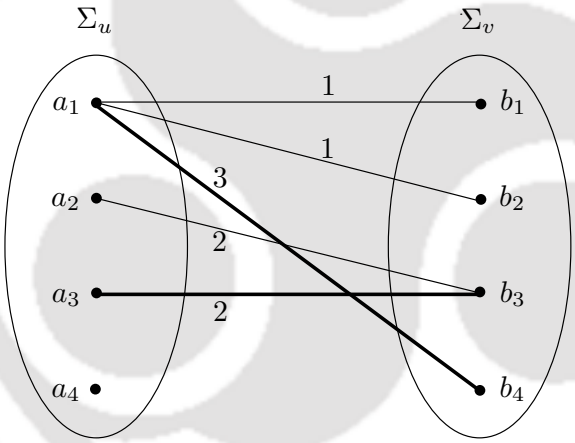

Figure 3.1: An example of reductions between the APSM and MWBM problems as mentioned in Example 3.2. In graph G the thick edges represent the edges in maximum weight matching.

A bijection for the APSM between u and v is defined by the edges of the matching. Therefore an optimal bijection is $\pi = \{a_1 \to b_4, a_2 \to b_2, a_3 \to b_3, a_4 \to b_1\} \in APSMe(u, v)$ and $d_H(\pi(u), v) = 4$ is the minimum over all bijections from Σ_u to Σ_v.

Further, as specified in the reduction that given a bipartite graph G, a pair of strings u' and v' can also be generated from the graph. They are, namely, $u' = a_1a_1a_1a_1a_1a_2a_2a_3a_3$ and $v' = b_1b_2b_4b_4b_4b_3b_3b_3b_3$. Observe that $APSMe(u, v) = APSMe(u', v')$. □

3.4 A Tight Lower Bound on the Weights of MWBM of Graphs

Let us consider a pair of distinct alphabets Σ_P and Σ_T where $|\Sigma_P| = |\Sigma_T| = \sigma$. Further, let $\mathcal{G}_{m\geq\sigma}$ be the collection of all weighted bipartite graphs $G = (\Sigma_P \cup \Sigma_T, E, Wt)$ where $\sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G)$ and $m \geq \sigma$. For notational convenience we use $\mathcal{G}$ instead of $\mathcal{G}_{m\geq\sigma}$. Therefore,

$$\mathcal{G} \equiv \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G), m \geq \sigma\}.$$

Further, let the term $\min_{G\in\mathcal{G}}\{Wt(mwm(G))\}$ denotes minimum weight among the maximum weight bipartite matchings of all the graphs in $\mathcal{G}$. Therefore, $\min_{G\in\mathcal{G}}\{Wt(mwm(G))\}$ is a lower bound of $\{Wt(mwm(G)) \mid G \in \mathcal{G}\}$.

Since $m \geq \sigma$, we can always write m as $q\sigma + r$ for some $q, r \in \mathbb{N}_0$ where $0 < r \leq \sigma$. First of all we show the existence of bipartite graph $G \in \mathcal{G}$ such that $Wt(mwm(G)) = q + 1$ in Theorem 3.3. We then prove in Theorem 3.4 that $q + 1$ is the tight lower bound of the set $\{Wt(mwm(G)) \mid G \in \mathcal{G}\}$.

Theorem 3.3. *Let $\mathcal{G} = \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G)$ and $m \geq \sigma\}$. If $m = q\sigma + r$ for some non-negative integers q and r where $0 < r \leq \sigma$, then there exists a bipartite graph $G \in \mathcal{G}$ such that $Wt(mwm(G)) = q + 1$.*

Proof. For the case $q = 0$, we have $m = q\sigma + r = r = \sigma$ as $0 < r \leq \sigma$ and $m \geq \sigma$. Figure 3.2(a) shows a graph $G' = (\Sigma_P \cup \Sigma_T, E', Wt) \in \mathcal{G}$ for this case. The weight of the graph $Wt(G') = \sigma$. In this graph G', $Wt(mwm(G')) = 1 = q + 1$.

For $q \geq 1$, total weight of the graph is $m = q\sigma + r$. We produce such a bipartite graph $G \in \mathcal{G}$ as shown in Figure 3.2(b) with $Wt(mwm(G)) = q + 1$. □

In a weighted graph G, any edge e of weight $c \in \mathbb{N}$ can be thought of as c number of overlapping unit weight edges e. Similarly, increasing the weight of G by adding weight $c \in \mathbb{N}$ is equivalent to adding c number of respective unit weight edges in G. Without loss of generality, we assume these as a convention while incrementing weight in a weighted graph.

Theorem 3.4 (Tight Lower Bound on the Weights of MWBM of the Graphs in $\mathcal{G}$ when $m \geq \sigma$)**.** *Let $\mathcal{G} = \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G)$ and $m \geq \sigma\}$. Then*

$$\min_{G\in\mathcal{G}}\{Wt(mwm(G))\} = q + 1$$

where $m = q\sigma + r$ for some non-negative integers q and r, and $0 < r \leq \sigma$.

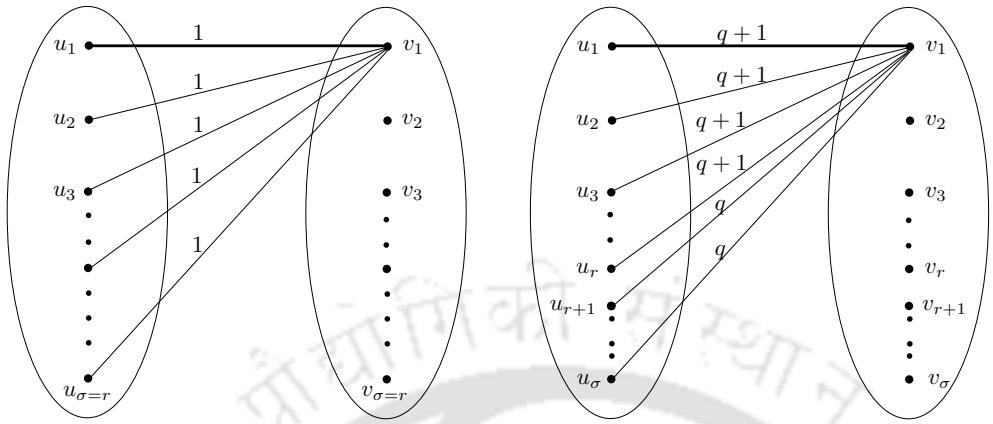

Figure 3.2: Given $m = q\sigma + r$ for some $q, r \in \mathbb{N}_0$ where $0 < r \leq \sigma$ and $\mathcal{G} = \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G),\ m \geq \sigma\}$ such that $\min_{G\in\mathcal{G}}\{Wt(mwm(G))\} = q+1$. **(a)** A bipartite graph example for the case $q = 0$. **(b)** A bipartite graph example for the $q \geq 1$ case. In both the graphs the thick edge represents maximum weight matching edges.

Proof. For $\sigma = 1$, the statement is trivially true. So we consider $\sigma \geq 2$ and prove the statement $\min_{G\in\mathcal{G}}\{Wt(mwm(G))\} = q+1$ by induction on $q \in \mathbb{N}_0$. Let $\Sigma_P = \{u_1, u_2, \ldots, u_\sigma\}$ and $\Sigma_T = \{v_1, v_2, \ldots, v_\sigma\}$ be the disjoint vertex sets of the graphs in $\mathcal{G}$. For simplicity, we denote $\mathcal{G}_{q+1} = \mathcal{G}$ when $m = q\sigma + r$ for some $q, r \in \mathbb{N}_0$ where $0 < r \leq \sigma$, that is, when $q = \lceil \frac{m-\sigma}{\sigma} \rceil$ where q is represented as a function of m and σ only.

Base Step: Let $q = 0$. Then $m = r = \sigma$ because $0 < r \leq \sigma$ and $m \geq \sigma$, and

$$\mathcal{G}_1 = \{G = (\Sigma_P \cup \Sigma_T, E,\ Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|,\ Wt(G) = \sigma\}.$$

Since for any graph $G = (\Sigma_P \cup \Sigma_T, E,\ Wt) \in \mathcal{G}_1$, $|\Sigma_P| = |\Sigma_T| = \sigma$ and $Wt(G) = \sigma$, therefore $\min_{G\in\mathcal{G}_1}\{Wt(mwm(G))\} = 1 = q+1$.

Induction Hypothesis: Assume that for $q = i$, $\min_{G\in\mathcal{G}_{i+1}}\{Wt(mwm(G))\} = i+1$, where

$$m = i\sigma + r \quad \text{and}$$

$$\mathcal{G}_{i+1} = \{G = (\Sigma_P \cup \Sigma_T, E,\ Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|,\ Wt(G) = i\sigma + r\}.$$

Let $\mathcal{G}'_{i+1} = \{G \in \mathcal{G}_{i+1} \mid Wt(mwm(G)) = i+1\}$. The set $\mathcal{G}'_{i+1}$ is non-empty by the Theorem 3.3. We use this set in the following inductive step.

Inductive Step: Let $q = i + 1$. We have to prove that

$$\min_{G \in \mathcal{G}_{i+2}} \{ Wt(mwm(G)) \} = i + 2$$

where

$$m = (i+1)\sigma + r \quad \text{and}$$
$$\mathcal{G}_{i+2} = \{ G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, \; Wt(G) = (i+1)\sigma + r \}.$$

The existence of a graph $G \in \mathcal{G}_{i+2}$ with $Wt(mwm(G)) = i+2$ is proved in the Theorem 3.3. Therefore, we only have to prove that there there does not exist any graph in $\mathcal{G}_{i+2}$ whose weight of a maximum weight matching is $i + 1$. Let us prove it by contradiction. Suppose there exists a graph $G_* \in \mathcal{G}_{i+2}$ such that $Wt(mwm(G_*)) = i + 1$.

Observe that, for any graph in $\mathcal{G}_{i+2}$, its weight is equal to $m = (i + 1)\sigma + r = (i\sigma + r) + \sigma$. Therefore, any graph in $\mathcal{G}_{i+2}$ is generated by adding a total of σ weight to the non-negative weight edges of a graph in $\mathcal{G}_{i+1}$.

Therefore, G_* can only be constructed from a graph in $\mathcal{G}'_{i+1}$ by adding a total of σ weight to the non-negative weight edges of that graph in $\mathcal{G}'_{i+1}$; because $\forall G \in \mathcal{G}_{i+1} \backslash \mathcal{G}'_{i+1}$, $Wt(mwm(G)) > i+1$. Let $\Sigma = \{e_1, e_2, e_3, \ldots\}$ be the edges, where $\sigma = \sum_{e_i \in \Sigma} Wt(e_i)$, whose weights are increased in $G \in \mathcal{G}'_{i+1}$ to build G_*.

Case 1: Let $G \in \mathcal{G}'_{i+1}$ and $M = mwm(G)$. If there exists at least one edge e in Σ such that $e \in M$ or if both the end points of e are unmatched vertices, then let $M' = M \cup \{e\}$, which is a weighted matching of G_*, not necessarily of maximum weight. Therefore

$$Wt(mwm(G_*)) \geq Wt(M') = Wt(M) + Wt(e) = i + 1 + Wt(e) > i + 1$$

which is a contradiction because we assumed that $(mwm(G_*)) = i + 1$.

Note: Hence for the rest of the cases we assume that none of the edges in Σ, which are added in $G \in \mathcal{G}'_{i+1}$ to get the $G_* \in \mathcal{G}_{i+2}$, belongs to M; or both the end points of none of the edges in Σ are unmatched vertices. Therefore if $e = \{u, v\} \in \Sigma$, then: (a) u is an unmatched vertex and v is a matched vertex or vice versa, or (b) both u and v are matched vertices, but $e \notin mwm(G)$.

Case 2: Let there exists at least one edge $e = \{u, v\} \in \Sigma$ such that $Wt(e) = w_\sigma \geq 2$. Then we have the following two sub-cases which also are shown in Figure 3.3. Let $G \in \mathcal{G}'_{i+1}$ and $M = mwm(G)$.

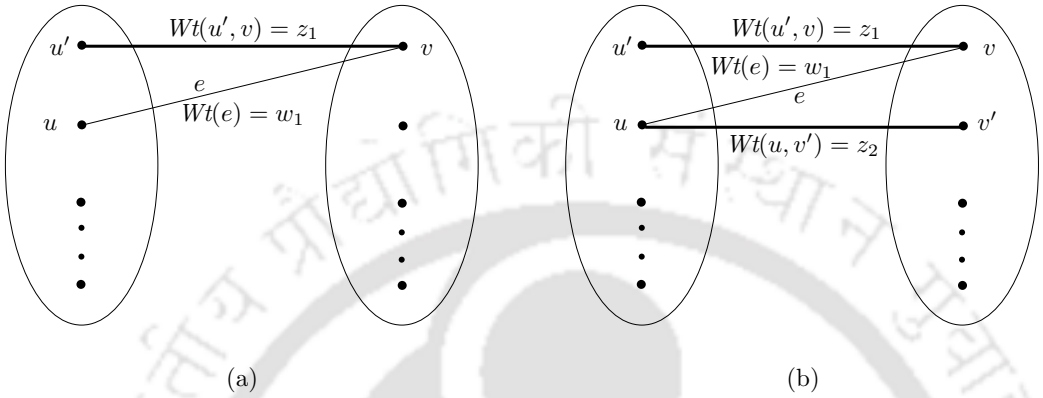

Figure 3.3: **(a)** This graph gives a pictorial representation of the Sub-case 2(a) in Theorem 3.4. **(b)** Sketch of the graph considered in Sub-case 2(b) is shown here. In both the graphs the thick edges are maximum weight matching edges.

Sub-case 2(a): Assume that u and v be the unmatched and matched vertices in $G \in \mathcal{G}'_{i+1}$, respectively. So there exists an edge $e' = \{u', v\} \in M$ which is incident on the matched vertex v. Let $Wt(e') = Wt(u', v) = z_1$ and $Wt(e) = Wt(u, v) = w_1$ in the G. Therefore $z_1 \geq w_1$. Now add the edge e in G where $Wt(e) = w_\sigma \geq 2$.

If $z_1 < w_1 + w_\sigma$, then let

$$M' = M \setminus \{e'\} \cup \{e\}$$

which is a weighted matching of G_*. Hence

$$Wt(mwm(G_*)) \geq Wt(M') = Wt(M) - z_1 + w_1 + w_\sigma > i + 1$$

which is a contradiction.

Or else,

$$z_1 \geq w_1 + w_\sigma \Leftrightarrow z_1 - 1 \geq w_1 + (w_\sigma - 1).$$

Therefore we can construct a new graph G' from G by decreasing one unit weight of the edge $e' = \{u', v\} \in M$ and increasing the weight of the edge $e = \{u, v\} \notin M$ by one unit in G. As a consequence, the

weight of G' remains the same as that of $G \in \mathcal{G}'_{i+1}$ and so $G' \in \mathcal{G}'_{i+1}$. But

$$Wt(mwm(G')) = i < Wt(M) = i + 1$$

which contradicts the definition of the set $\mathcal{G}'_{i+1}$.

Sub-case 2(b): Suppose both u and v are matched vertices but $e \notin M$. So there exist two edges $e' = \{u', v\} \in M$ and $e'' = \{u, v'\} \in M$ which are incident on the matched vertices v and u, respectively. Let $Wt(e') = z_1$, $Wt(e'') = z_2$ and $Wt(e) = w_1$ in $G \in \mathcal{G}'_{i+1}$.

$$\therefore\ z_1 + z_2 \geq w_1 \quad \text{in } G.$$

Now after adding the edge e in G with $Wt(e) = w_\sigma \geq 2$, if

$$z_1 + z_2 < w_1 + w_\sigma,$$

then let

$$M' = M \setminus \{e', e''\} \cup \{e\}$$

which is a weighted matching of G_*. Hence

$$Wt(mwm(G_*)) \geq Wt(M') = Wt(M) - z_1 - z_2 + w_1 + w_\sigma > i + 1$$

which is a contradiction.

Or else,

$$z_1 + z_2 \geq w_1 + w_\sigma \Leftrightarrow (z_1 - 1) + z_2 \geq w_1 + (w_\sigma - 1).$$

Therefore we can construct a new graph G' from G by reducing one unit weight of the edge $e' = (u', v) \in M$ and adding one unit weight to the edge $e = (u, v) \notin M$ of G. As a consequence, the weight of G' is the same as that of $G \in \mathcal{G}'_{i+1}$ and so $G' \in \mathcal{G}'_{i+1}$. But

$$Wt(mwm(G')) = i < Wt(M) = i + 1$$

which contradicts the definition of the set $\mathcal{G}'_{i+1}$.

Case 3: Let for each edge $e \in \Sigma$, $Wt(e) = 1$. Consider $\Sigma = \{e_1 = \{u_1, v_1\}, e_2 = \{u_2, v_2\}, \ldots, e_\sigma = \{u_\sigma, v_\sigma\}\}$ and their respective weights in $G \in \mathcal{G}'_{i+1}$ are given by $\{w_1, w_2, \ldots, w_\sigma\}$. We add these σ edges of Σ in $G \in \mathcal{G}'_{i+1}$ to get G_*. Further let $M = mwm(G)$.

Therefore, there must exist two edges in Σ which are not adjacent. Because if not, then all the edges of Σ are adjacent to one vertex. Without loss of generality, suppose $u_1 = u_2 = \cdots = u_\sigma$. See Figure 3.4 and consider the following two possibilities.

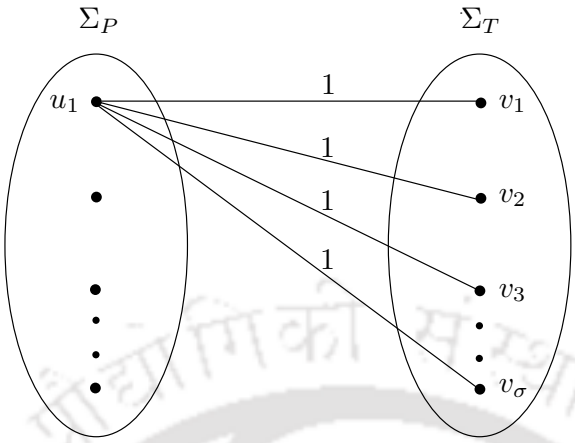

Figure 3.4: There must exists two edges $e_1, e_2 \in \Sigma$ such that e_1 and e_2 are not adjacent. This kind of graph does not arise in Case 3 of Theorem 3.4.

(a) If $u_1 \in \Sigma_P$ is an unmatched vertex in $G \in \mathcal{G}'_{i+1}$, then there must be another unmatched vertex in Σ_T of the graph G, because $\sigma = |\Sigma_P| = |\Sigma_T|$. Say the unmatched vertex is $v_1 \in \Sigma_T$. If we add an edge $\{u_1, v_1\} \in \Sigma$ in G, then this kind of graph is already addressed in Case 1. Therefore, at most $\sigma - 1$ edges of unit weight can be added in G. But in Σ we have σ number of unit weight distinct edges which are to be added in G to built the G_*. This is a contradiction.

(b) Similarly, if $u_1 \in \Sigma_P$ is a matched vertex in $G \in \mathcal{G}'_{i+1}$, then there must be another matched vertex in Σ_T of the graph G. The rest of the argument is similar to previous unmatched case.

So we assume the two edges be $e_1, e_2 \in \Sigma$ such that e_1 and e_2 are not adjacent. Then a maximum of four edges in M are adjacent to edges $e_1, e_2 \in \Sigma$. Let e'_1, e'_2, e'_3, e'_4 be the such edges and z_1, z_2, z_3, z_4 be their corresponding weights in $G \in \mathcal{G}'_{i+1}$, respectively.

$$\therefore \; z_1 + z_2 + z_3 + z_4 \geq w_1 + w_2 \quad \text{in } G.$$

Now after adding σ edges of Σ in G, if

$$z_1 + z_2 + z_3 + z_4 < w_1 + w_2 + 2,$$

then let

$$M' = M \setminus \{e'_1, e'_2, e'_3, e'_4\} \cup \{e_1, e_2\}$$

which is a weighted matching of G_*. Hence

$$\begin{aligned} Wt(mwm(G_*)) &\geq Wt(M') \\ &= Wt(M) - (z_1 + z_2 + z_3 + z_4) + (w_1 + w_2 + 2) \\ &> i + 1 \end{aligned}$$

which is a contradiction.

Or else,

$$\begin{aligned} & z_1 + z_2 + z_3 + z_4 \geq w_1 + w_2 + 2 \\ \Leftrightarrow\ & z_1 + z_2 + z_3 + z_4 - 1 \geq w_1 + w_2 + 1. \end{aligned}$$

As a consequence, by similar argument as in Sub-case 2(b), we can construct a new graph G' whose weight is same as that of $G \in \mathcal{G}'_{i+1}$ and so $G' \in \mathcal{G}'_{i+1}$. But

$$Wt(mwm(G')) = i < Wt(M) = i + 1$$

which contradicts the definition of the set $\mathcal{G}'_{i+1}$.

This completes the proof. □

An equivalent statement of the Theorem 3.4 is the following.

Corollary 3.5. *Let* $\mathcal{G} = \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G)$ *and* $m \geq \sigma\}$, *then*

$$\min_{G \in \mathcal{G}} \{ Wt(mwm(G)) \} = \left\lceil \frac{m - \sigma}{\sigma} \right\rceil + 1.$$

Proof. Since $m \geq \sigma$, we can always write m as $q\sigma + r$ for some $q, r \in \mathbb{N}_0$ where $0 < r \leq \sigma$. Then the term $\lceil \frac{m-\sigma}{\sigma} \rceil$ can be written as

$$\left\lceil \frac{m - \sigma}{\sigma} \right\rceil = \left\lceil \frac{q\sigma + r - \sigma}{\sigma} \right\rceil = \left\lceil \frac{(q-1)\sigma + r}{\sigma} \right\rceil = (q-1) + 1 = q.$$

Hence the statement in this corollary is equivalent to the same in Theorem 3.4. □

3.5 Error Classes in APSM

Let us consider a pair of distinct alphabets Σ_P and Σ_T where $|\Sigma_P| = |\Sigma_T| = \sigma$. So, the cardinality of each of Σ_P^m and Σ_T^m is σ^m. And hence the cardinality of all possible unique (p, t) pairs of strings in $\Sigma_P^m \times \Sigma_T^m$ for APSM problem is σ^{2m}, where $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$.

The total number of possible bijections from $\Sigma_P \to \Sigma_T$ is $\sigma!$. Let π: $\Sigma_P \to \Sigma_T$ be a bijection. Given a pair $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$, APSM problem between p and t under Hamming distance, requires to find a π over all bijections such that $\pi\text{-}mismatch(p, t) = d_H(\pi(p), t)$ is minimum.

Definition 3.6 (An Error Class in APSM)**.** *Let $p \in \Sigma_P^m$ and $t \in \Sigma_T^m$ be two strings, where $\sigma = |\Sigma_P| = |\Sigma_T|$. Further let $D_k \subseteq \Sigma_P^m \times \Sigma_T^m$, where $k \in \mathbb{N}_0$. Then we say that $(p, t) \in D_k$ if there exists a π over all possible bijections from Σ_P to Σ_T such that $k = d_H(\pi(p), t)$ is minimum.*

For any pair (p, t) of $\Sigma_P^m \times \Sigma_T^m$ belongs to the set D_k if and only if $\pi \in APSMe(p, t)$ and $\pi\text{-}mismatch(p, t) = d_H(\pi(p), t) = k$. We call the set D_k as an *Error Class* (EC) in APSM. Let $\mathcal{D} = \{D_0, D_1, D_2, \ldots\}$ be the collection of all ECs.

Example 3.7. Let us consider $\Sigma_P = \{a, b\}$ and $\Sigma_T = \{0, 1\}$, and all the 2-length strings over them. Therefore, $\Sigma_P^2 = \{aa, ab, ba, bb\}$ and $\Sigma_T^2 = \{00, 01, 10, 11\}$, and the cardinality of distinct (p, t) pairs in $\Sigma_P^2 \times \Sigma_T^2$ is 16, where $p \in \Sigma_P^2$ and $t \in \Sigma_T^2$. According to the Definition 3.6, these pairs are partitioned as follows.

$$D_0 = \{(aa, 00), (aa, 11), (ab, 01), (ab, 10), (ba, 01), (ba, 10), (bb, 00), (bb, 11)\},$$

$$D_1 = \{(aa, 01), (aa, 10), (ab, 00), (ab, 11), (ba, 00), (ba, 11), (bb, 01), (bb, 10)\}$$

$$\text{and} \quad D_i = \emptyset \ \ \forall i \geq 2.$$

In this example, the number of non-empty ECs in $\mathcal{D}$ is 2; and the cardinality of each of D_0 and D_1 is 8. □

3.6 Non-Empty Error Classes

Let D_i and D_j be any two error classes where $i, j \in \mathbb{N}_0$ and $i \neq j$. Then D_i and D_j are disjoint, that is, any string pair $(p, t) \in \Sigma_P^m \times \Sigma_T^m$, it must belongs to exactly one D_i for some $i \in \mathbb{N}_0$. There are two parameters for APSM problem: one is the alphabet size which is denoted by σ and the other one is the length of the pattern (and the text) which is denoted by m.

Lemma 3.8. *For any string pair (p,t) of $\Sigma_P^m \times \Sigma_T^m$, (p,t) must belongs to exactly one D_i, where $i < m$, and $D_i = \emptyset$ for $i \geq m$.*

Proof. For any string pair $(p,t) \in \Sigma_P^m \times \Sigma_T^m$, it must belongs to exactly one D_i for some $i \in \mathbb{N}_0$. Since the length of each string is m, a bijection for APSM between p and t will match at least one symbol of Σ_P in p to another symbol of Σ_T in t. Therefore there does not exist any pair of strings $(p,t) \in \Sigma_P^m \times \Sigma_T^m$ such that $d_H(\pi(p),t) \geq m$ where $\pi \in APSMe(p,t)$. Hence, $D_i = \emptyset$ for $i \geq m$. □

Thus there are at most m non-empty ECs in APSM problem, namely, $D_0, D_1, \ldots, D_{m-1}$, for the elements in $\Sigma_P^m \times \Sigma_T^m$.

Definition 3.9 (Trivial and Non-Trivial Empty Error Classes)**.** *Since the error classes $D_m, D_{m+1}, D_{m+2}, \ldots$ are always empty for APSM problem, we call them as the Trivial Empty Error Classes (TEECs) of APSM problem for $\Sigma_P^m \times \Sigma_T^m$. We call the rest of the empty classes, if exist, of APSM problem as Non-Trivial Empty Error Classes (NEECs).*

Theorem 3.10. *If $m \leq \sigma$, then the error class D_{m-1} is non-empty.*

Proof. It is enough to show that there exists at least one pair $(p,t) \in \Sigma_P^m \times \Sigma_T^m$ such that $(p,t) \in D_{m-1}$, which implies that π-*mismatch*$(p,t) = m-1$ where $\pi \in APSMe(p,t)$. Let us take $\Sigma_P = \{a_1, a_2, \ldots, a_\sigma\}$ and $\Sigma_T = \{1, 2, \ldots, \sigma\}$.

We construct a pair $(p,t) \in \Sigma_P^m \times \Sigma_T^m$ in such a way so as to generate the maximum error while computing $APSMe(p,t)$ as follows:

$$\begin{array}{lccccl} p = & a_i & a_i & \ldots & a_i & \text{for some fixed } a_i \in \Sigma_P, \text{ and} \\ t = & 1 & 2 & \ldots & m. & \end{array}$$

Such a string t exists because $m \leq \sigma$. Observe that any bijection π with $a_i \to j$, $j \in \{1, 2, \ldots, m\}$ will result in π-*mismatch*$(p,t) = m-1$ where $\pi \in APSMe(p,t)$. Hence $D_{m-1} \neq \emptyset$ for $m \leq \sigma$. □

Theorem 3.11. *The error class D_{m-1} is empty if and only if $m > \sigma$.*

Proof. Let D_{m-1} is empty, then we need to show that $m > \sigma$. This is a contrapositive statement of the Theorem 3.10.

Conversely, let $m > \sigma$. Consider a pair (p,t) which belongs to an error class with maximum error. That is, $d_H(\pi(p),t) \geq d_H(\pi'(p'),t')$ for any $(p',t') \in \Sigma_P^m \times \Sigma_T^m$ where $\pi \in APSMe(p,t)$ and $\pi' \in APSMe(p',t')$. Since $m > \sigma$, we must have at least one symbol of Σ_P and Σ_T repeating in both p and t, respectively. Let these symbols be $s_p \in \Sigma_P$ and $s_t \in \Sigma_T$ in p and t,

respectively. If both s_p and s_t are aligned in at least two places in p and t respectively, then the maximum error will be less than $m-1$.

If both s_p and s_t are not aligned in at least two places in p and t respectively, then the structure of p and t will be as shown below:

$$\begin{aligned} p &= \ldots \quad s_p \quad \ldots \quad s_p \quad \ldots \quad s'_p \quad \ldots \\ t &= \ldots \quad s_t \quad \ldots \quad s'_t \quad \ldots \quad s_t \quad \ldots \end{aligned}$$

where $s'_p \neq s_p$ and $s'_t \neq s_t$. Then we can always choose a π, not necessarily a best one, that maps $s_p \rightarrow s'_t$ and $s'_p \rightarrow s_t$ which will result in mismatch error of at most $m-2$. This proves that $D_{m-1} = \emptyset$. □

3.7 Distribution of Non-Empty and Empty Error Classes

Now we investigate how the non-empty and empty ECs are distributed in $\mathcal{D}$ of APSM problem. Is it possible that $D_k \neq \emptyset$ and $D_{k-1} = \emptyset$ for some $k \in \mathbb{N}$? The following theorem answers the same.

Theorem 3.12 (Distribution of ECs in $\mathcal{D}$)**.** *For the collection of ECs $\mathcal{D}$ of APSM problem the following two properties hold always.*

(a) if for some $k \in \mathbb{N}_0$ $D_k \neq \emptyset$, then $D_i \neq \emptyset$ $\forall i \leq k$ where $i \in \mathbb{N}_0$, and

(b) if for some $k \in \mathbb{N}_0$ $D_k = \emptyset$, then $D_j = \emptyset$ $\forall j \geq k$ where $j \in \mathbb{N}_0$.

Proof. The justification for these two properties are given below.

(a) It is enough to prove that if D_k is non-empty where $k \geq 1$, then there exists a string pair in D_{k-1}. Let us consider a pair $(p,t) \in \Sigma_P^m \times \Sigma_T^m$ which belongs to D_k. It means that π-*mismatch*$(p,t) = k$ where $\pi \in$ *APSMe*(p,t). Therefore $d_H(\pi(p),t) = k$ and there does not exist any π': $\Sigma_P \rightarrow \Sigma_T$ such that $d_H(\pi'(p),t) < k$. Let

$$\begin{aligned} p &= p_1 \quad p_2 \quad \ldots \quad p_{k-1} \quad p_k \quad p_{k+1} \quad \ldots \quad p_m \\ t &= t_1 \quad t_2 \quad \ldots \quad t_{k-1} \quad t_k \quad t_{k+1} \quad \ldots \quad t_m. \end{aligned}$$

Hence there exists exactly k locations in the strings for which $d_H(\pi(p_i),t_i) = 1$ where $i \in \{1,2,\ldots,m\}$.

Without loss of generality, let us assume that the k locations in p and t are the first k indexes of the same, for which the errors have occurred. Therefore $\pi(p_i) \neq t_i$ $\forall i \in \{1,2,\ldots,k\}$. So, we transform t to t' as follows and get another pair $(p,t') \in \Sigma_P^m \times \Sigma_T^m$ which belongs to D_{k-1}:

$$\begin{array}{lllllllll} t = t_1 & t_2 & \dots & t_{k-1} & t_k & t_{k+1} & \dots & t_m & \text{to} \\ t' = t_1 & t_2 & \dots & t_{k-1} & \pi(p_k) & t_{k+1} & \dots & t_m. & \end{array}$$

See that we have $d_H(\pi(p), t') = k - 1$.

Further, $k-1$ is the minimum error possible by any bijection from Σ_P to Σ_T. Because if not, then suppose that there exists an arbitrary bijection π_1 from Σ_P to Σ_T such that $d_H(\pi_1(p), t') = k - 2$. Note that the k-th symbol of t' is $\pi(p_k)$ and $d_H(\pi(p_k), t_k) = 1$ where $\pi \in APSMe(p, t)$. Now the following four cases may arise while computing the error for the k-th symbols in (p, t') and (p, t) under π_1, that is, $d_H(\pi_1(p_k), \pi(p_k))$ and $d_H(\pi_1(p_k), t_k)$:

$$\begin{aligned} &(i) \text{ if } d_H(\pi_1(p_k), \pi(p_k)) = 0 \text{ and } d_H(\pi_1(p_k), t_k) = 0 < d_H(\pi(p_k), t_k) \\ &\quad \Rightarrow d_H(\pi_1(p), t) = k - 2, \\ &(ii) \text{ if } d_H(\pi_1(p_k), \pi(p_k)) = 0 \text{ and } d_H(\pi_1(p_k), t_k) = 1 = d_H(\pi(p_k), t_k) \\ &\quad \Rightarrow d_H(\pi_1(p), t) = (k - 2) + 1 = k - 1, \\ &(iii) \text{ if } d_H(\pi_1(p_k), \pi(p_k)) = 1 \text{ and } d_H(\pi_1(p_k), t_k) = 0 < d_H(\pi(p_k), t_k) \\ &\quad \Rightarrow d_H(\pi_1(p), t) = (k - 2) - 1 = k - 3, \\ &(iv) \text{ if } d_H(\pi_1(p_k), \pi(p_k)) = 1 \text{ and } d_H(\pi_1(p_k), t_k) = 1 = d_H(\pi(p_k), t_k) \\ &\quad \Rightarrow d_H(\pi_1(p), t) = k - 2. \end{aligned}$$

Each of these four cases (i),(ii),(iii) and (iv) leads to a contradiction, because we assumed that $(p, t) \in D_k$ which implies $d_H(\pi(p), t) = k$ where $\pi \in APSMe(p, t)$. Hence $(p, t') \in D_{k-1}$, that is, D_{k-1} is non-empty. This completes the proof.

(b) This part is a contrapositive statement of (a), that is,

$$\begin{aligned} &D_k = \emptyset \Rightarrow D_{k+1} = \emptyset \\ &\equiv D_{k+1} \neq \emptyset \Rightarrow D_k \neq \emptyset. \end{aligned}$$

Hence the proof. □

3.8 Exact Count of Non-Empty Error Classes

Up to this point we know that if $m \leq \sigma$ then $D_{m-1} \neq \emptyset$ (see Theorem 3.10) and $D_m = \emptyset$ (see Lemma 3.8), and hence the number of non-empty ECs of

APSM problem is m and they are $D_0, D_1, \ldots, D_{m-1}$ (by Theorem 3.12). On the other hand, if $m > \sigma$ then $D_{m-1} = \emptyset$ (see Theorem 3.11). Therefore while $m > \sigma$, the number of non-empty ECs of APSM is strictly less than m and the set of non-empty ECs must be a subset of $\{D_0, D_1, \ldots, D_{m-2}\}$.

Let us first see the following lemma which is useful in proving a theorem which gives general formulas to find the non-empty and empty ECs exactly for the APSM problem when $0 < \sigma \leq m$.

Recall that in Section 3.3 where we have described that the APSM problem for equal length strings is computationally equivalent to the MWBM problem [57]. Let $\mathcal{G}$ denotes the set of all weighted bipartite graphs which are generated from all pairs of strings $(p, t) \in \Sigma_P^m \times \Sigma_T^m$, where $\sigma = |\Sigma_P| = |\Sigma_T|$ and $m \geq \sigma$. So,

$$\mathcal{G} = \{G = (\Sigma_P \cup \Sigma_T, E, Wt) \mid \sigma = |\Sigma_P| = |\Sigma_T|, m = Wt(G), m \geq \sigma\}.$$

Lemma 3.13. *Let $m \geq \sigma$ and $m = q\sigma + r$ for some $q, r \in \mathbb{N}_0$ where $0 < r \leq \sigma$. Then $D_{m-q-1} \neq \emptyset$, $D_{m-q} = \emptyset$ if and only if $\min_{G \in \mathcal{G}}\{Wt(mwm(G))\} = q+1$.*

Proof. Given $D_{m-q-1} \neq \emptyset$ and $D_{m-q} = \emptyset$

$\Leftrightarrow \max_{(p,t) \in \Sigma_P^m \times \Sigma_T^m}\{cost(APSMe(p,t))\} = m - q - 1$ (see Problem 2.11 for the term $cost(APSMe(p,t))$)

$\Leftrightarrow \max_{(p,t) \in \Sigma_P^m \times \Sigma_T^m}\{d_H(\pi(p), t) \mid \pi \in APSMe(p,t)\} = m - q - 1.$

$\Leftrightarrow \max_{(p,t) \in \Sigma_P^m \times \Sigma_T^m}\{\text{minimum } \pi\text{-}mismatch(p,t) \text{ over all } \pi\text{: } \Sigma_P^m \to \Sigma_T^m\} = m - q - 1.$

$\Leftrightarrow \min_{G \in \mathcal{G}}\{Wt(mwm(G))\} = m - (m - q - 1) = q + 1$ (see the reduction idea in Section 3.3 including the Observation 3.1). □

Although for the case $m \leq \sigma$, the exact number of non-empty ECs is already found to be m, but the same is still unanswered for the case $0 < \sigma < m$. We have only an upper bound information for the later case which is equal to $m - 1$ due to the Theorems 3.11 and 3.12. Below we present general formulas which state the non-empty and empty ECs exactly for APSM problem when $0 < \sigma \leq m$. As a consequence we can compute an exact number of non-empty error classes for the case $0 < \sigma \leq m$. These formulas depend on the parameters m and σ.

Theorem 3.14. *Let $0 < \sigma \leq m$ where $\sigma = |\Sigma_P| = |\Sigma_T|$ and m is the length of each of the strings belongs to Σ_P^m and Σ_T^m. Then*

$$(a) \quad D_i \neq \emptyset \quad \text{if} \quad 0 \leq i < m - \lceil \tfrac{m-\sigma}{\sigma} \rceil \quad \text{and}$$

$$(b) \quad D_i = \emptyset \quad \text{if} \quad i \geq m - \lceil \tfrac{m-\sigma}{\sigma} \rceil.$$

Proof. Since $m \geq \sigma$, m can be written as $m = q\sigma + r$ for some non-negative integers q and r where $0 < r \leq \sigma$. Therefore $\left\lceil \frac{m-\sigma}{\sigma} \right\rceil = q$. By Theorem 3.12, it is enough to prove that $D_{m-q-1} \neq \emptyset$, $D_{m-q} = \emptyset$. And it is true because of Lemma 3.13 and Theorem 3.4. □

Corollary 3.15. *For an APSM problem over the strings Σ_P^m and Σ_T^m, where $\sigma = |\Sigma_P| = |\Sigma_T|$, the total number of non-trivial empty ECs is given by*

$$|NEECs| = \begin{cases} 0 & \text{if} \quad m \leq \sigma \\ \lceil \frac{m-\sigma}{\sigma} \rceil & \text{if} \quad m > \sigma. \end{cases}$$

Proof. We know that $D_m = \emptyset$ always (by Lemma 3.8). Further, the Theorem 3.10 states that the $D_{m-1} \neq \emptyset$ when $m \leq \sigma$. Therefore by the Definition 3.9 (on page 31), $|NEECs| = 0$ if $m \leq \sigma$.

Further, suppose $k = m - \lceil \frac{m-\sigma}{\sigma} \rceil$. When $m > \sigma$, by Theorem 3.14 $D_{k-1} \neq \emptyset$ and $D_k = \emptyset$. Therefore, $|NEECs| = m - k = \lceil \frac{m-\sigma}{\sigma} \rceil$. □

3.9 Divisibility Property of the Size of an Error Class

Below we present some more combinatorial properties of ECs of APSM. The following theorems are true for any error class of APSM problem for arbitrary σ and m.

Theorem 3.16. *Let APSM problem is considered over the alphabets of cardinality σ each. The lower bound of size of any non-empty error class D_i is σ^2, that is, $|D_i| \geq \sigma^2$ where D_i is non-empty and $i \in \mathbb{N}_0$.*

Proof. Since D_i is non-empty, let a string pair $(p, t) \in \Sigma_P^m \times \Sigma_T^m$ belongs to it. This means $d_H(\pi(p), t) = i$ where $\pi \in APSMe(p, t)$. We use a Theorem 5.4 (on page 64) of Chapter 5 which states that if p ($\in \Sigma_P^m$) is approximate parameterized matched with t ($\in \Sigma_T^m$) and $t \mathrel{\widehat{=}} t'$ where $t' \in \Sigma_T^m$, than p is approximate parameterized matched with t' and $cost(APSMe(p, t')) = cost(APSMe(p, t)) = d_H(\pi(p), t) = i$. That is, if $(p, t) \in D_i$ for some $i \in \mathbb{N}_0$ and $t \mathrel{\widehat{=}} t'$ where $t' \in \Sigma_T^m$, then $(p, t') \in D_i$.

Now observe an important point that, given any string $p \in \Sigma_P^m$, there must be at least σ number of its distinct parameterized strings in Σ_P^m, including p. The same holds for any string $t \in \Sigma_T^m$. For example, of $\Sigma_P = \{a, b, c\}$ and $m = 4$, then the string *aaaa* is exactly parameterized with *bbbb*, *cccc* and itself.

As a consequence, for any $(p,t) \in \Sigma_P^m \times \Sigma_T^m$, there are at least σ^2 string pairs in $\Sigma_P^m \times \Sigma_T^m$ which are parameterized matched with (p,t) including itself. Hence, if D_i is non-empty for any $i \in \mathbb{N}_0$ then $|D_i| \geq \sigma^2$. □

Theorem 3.17. *Let APSM problem is considered over the alphabets of cardinality σ. The cardinality of any D_i is divisible by σ^2 where $i \in \mathbb{N}_0$.*

Proof. An mentioned in the proof of previous theorem that for any string $p \in \Sigma_P^m$, there must exist at least σ number of its distinct parameterized strings in Σ_P^m, including p. More importantly, for any string $p \in \Sigma_P^m$, there must exist exactly $c\sigma$ number of its distinct parameterized strings in Σ_P^m including p, for some constant $c \in \mathbb{N}$. This is also visible if we write all the strings in Σ_P^m in lexicographical order. Therefore the cardinality of the set of all parameterized strings of p in Σ_P^m must be divisible by σ. The same holds for any string $t \in \Sigma_T^m$. That is, for any string $t \in \Sigma_T^m$, there must exists exactly $c'\sigma$ number of its distinct parameterized strings in Σ_T^m including t, for some constant $c' \in \mathbb{N}$.

So, for any $(p,t) \in D_i$, $i \in \mathbb{N}_0$, there are $cc'\sigma^2$ number of its parameterized string pairs in D_i including (p,t), for some constant $c, c' \in \mathbb{N}$. Hence each $|D_i|$ must be divisible by σ^2. □

3.10 Enumeration of Pairs in an Error Class

Let us first see a naive algorithm to enumerate the cardinality of an EC of APSM for given σ and m. We then use the above mentioned divisibility property to improve the time complexity of the naive algorithm by $O(\sigma^2)$.

Naive Algorithm. Find all possible string pairs of $\Sigma_P^m \times \Sigma_T^m$, solve APSM for each of the pair and map them to the respective error classes. As mentioned earlier that the cardinality of each of Σ_P^m and Σ_T^m is σ^m. The total number of string pairs in $\Sigma_P^m \times \Sigma_T^m$ is σ^{2m}. Solving APSM for each pair takes $O(m\sqrt{\sigma})$ time (please refer to a Lemma 6.1 on page 76) using the MWBM algorithm. A fine-tuned MWBM algorithm is discussed in the next Chapter 4. Therefore an $O(\sigma^{2m}\, m\sqrt{\sigma})$ time is required to classify all the error classes.

Experiment 3.18. Let us see another example with $\Sigma_P = \{a,b,c\}$ and $\Sigma_T = \{0,1,2\}$, and all the m-length strings over them, simultaneously for $m = 1,2,\ldots,9$. Here we count the number of non-empty ECs for different length strings and fixed size alphabets. The cardinality of distinct (p,t) pairs of strings in $\Sigma_P^i \times \Sigma_T^i$ for APSM problem is 3^{2i}, where $p \in \Sigma_P^i$, $t \in \Sigma_T^i$ and $i \in \{1,2,\ldots,9\}$.

Table 3.1: Count of each of the error classes of APSM for different length strings for $|\Sigma_P| = |\Sigma_T| = 3$ as given in Experiment 3.18.

$m \downarrow$	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	$\cdots$	Total (p,t) pairs
1	9											9
2	45	36										81
3	153	540	36									729
4	477	3168	2916	0								6561
5	1449	13500	36000	8100	0							59049
6	4365	50976	216000	252000	8100	0						531441
7	13113	181692	1016064	2469600	1102500	0	0					4782969
8	39357	627264	4261824	14789376	20506500	2822400	0	0				43046721
9	118089	2122524	16770240	73325952	171206784	121054500	2822400	0	0			387420489

We have implemented a naive algorithm in *MATLAB Version 7.8.0.347 (R2009a)* in order to count the number of non-empty error classes. This is done by enumerating and counting the cardinality of each error classes, simultaneously for $m = 1, 2, \ldots, 9$ length strings, where sizes of both the alphabets Σ_P and Σ_T are 3.

Though the rotated Table 3.1 is self explanatory, but for the sake of completeness here goes a brief description of the table. First column represents the string length for $m = 1, 2, \ldots, 9$.

For a fixed σ and any particular m, the collection of all ECs in the APSM problem is $\mathcal{D} = \{D_0, D_1, D_2, \ldots\}$ and it is shown in the first row. And the last column adds up the total number of $(p, t) \in \Sigma_P^i \times \Sigma_T^i$ which are distributed to the respective ECs, where $i = m$. As stated in the Definition 3.9, for a fixed σ and m, the empty cells with respect to the ECs $D_0, D_1, D_2, \ldots$ are called as the trivial empty error class in the APSM Problem. For example, the meaning of fifth row is that in APSM problem for $m = 4$ with $\sigma = 3$, $|D_0| = 447$, $|D_1| = 3168$, $|D_2| = 2916$, $|D_3| = 0$ and rest are trivial empty ECs. Therefore for APSM problem with $m = 4$ and $\sigma = 3$, the number of non-empty ECs is equal to 3 and the number of non-trivial empty ECs, that is, $|NEECs|$ is equal to 1.

Similarly, looking at the Table 3.1 one can also see all the results such as distribution of empty and non-empty ECs, exact count of non-empty ECs and divisibility property of the size of an EC in APSM, which are presented in the Sections 3.6–3.9. □

3.11 Conclusions

We have explored various combinatorial properties of the error classes in APSM problems under Hamming distance error model. We have given several results on distribution of empty and non-empty ECs in $\mathcal{D}$, exact count of non-empty ECs and divisibility property of the size of ECs in APSM. Furthermore, we have provided a tight lower bound on the weights of MWBM of graphs which is correlated with the counting of number of error classes in APSM problem.

In general, the set of possible edit operations are: *insertion*, *deletion*, *substitution* or *replacement* and *transposition* [57]. Other variation of operations are also possible which lead to some more error models, such as Damerau distance [37], LCS distance [12], episode distance [38] etc. The study of combinatorial properties of the error class in APSM problem is open for different error models.

Chapter 4

Fine-Tuning Decomposition Theorem for Maximum Weight Bipartite Matching

In this chapter we focus on computation of maximum weight matching in a bipartite graph using an exact algorithm. We propose a modified decomposition algorithm to compute a MWBM of a given bipartite graph. The solutions to the problems addressed in Chapters 5 and 6 use this algorithm as a subroutine.

4.1 Introduction

Let $G = (V = V_1 \cup V_2, E, Wt)$ be an undirected, weighted bipartite graph having V_1 and V_2 as partition of vertex set V, with positive integer weights on the edges of G. For preliminaries of maximum matching in a bipartite graph, please refer to the Section 3.2 (page 20). Throughout this chapter, we use the symbols N and W to denote the largest weight of any edge and total weight of G which is equal to $\sum_{e \in E} Wt(e)$, respectively.

In [69,70], Kao et al. proposed a decomposition theorem and algorithm for computing weight of a MWBM of the bipartite graph G. Our contribution in this chapter is a revised version of the existing decomposition theorem and use it efficiently to design an improved version of the decomposition algorithm to estimate the weight of a MWBM of G in time $O(\sqrt{|V|}W'/k(|V|, W'/N))$ by taking algorithm designed by Feder and Motwani [45] as base algorithm, where $k(x, y) = \log x / \log(x^2/y)$.

This algorithm bridges a gap between the best known time complexity of computing a Maximum Cardinality Matching (MCM) and that of

computing a MWBM of a bipartite graph. In best case, computation of weight of a MWBM takes $O(\sqrt{|V|}|E|/k(|V|,|E|))$ time which is the same as the complexity of the Feder and Motwani's algorithm [45] for computing MCM of unweighted bipartite graph; whereas in worst case it takes $O(\sqrt{|V|}W/k(|V|,W/N))$, that is, $|E| \leq W' \leq W$. Further, we provide an interesting scaling property of the algorithm and a better bound of the parameter W'. However, it seems to be a challenging problem to get rid of W or N from the complexity.

The modified algorithm works well for general W, but is best known for $W' = o(|E|\log(|V|N))$. We also design a revised algorithm to construct minimum weight cover of a bipartite graph in time $O(\sqrt{|V|}W'/k(|V|,W'/N))$ to identify the edges involved in maximum weight bipartite matching. It is also possible to use other algorithms as a subroutine, for example, algorithms given by Hopcroft and Karp [59] and Alt et al. [4] in which case the running times of our algorithm will be $O(\sqrt{|V|}W')$ and $O((|V|/\log|V|)^{1/2}W')$, respectively. An experimental evaluation on randomly generated bipartite graphs shows that the proposed improvement is significant in general.

Roadmap. In Section 4.2, we give a detailed summary of existing maximum matching algorithms and their complexities for unweighted and weighted bipartite graphs. Section 4.3 describes modified decomposition theorem and an algorithm to compute the weight of a MWBM. The complexity analysis of the algorithm is discussed in Section 4.4. The algorithm to compute minimum weight cover of a bipartite graph is given in Section 4.5, which is used to find the edges of a MWBM. Section 4.6 provides the experimental comparisons between the modified algorithm and Kao et al.'s algorithm for randomly generated bipartite graphs. We summarize the results in Section 4.7.

4.2 Survey of Maximum Matching in Bipartite Graph

The problem of computing maximum matching in a given graph is one of the fundamental algorithmic problem that has played an important role in the development of combinatorial optimization and algorithmics. A survey of some of the well known existing maximum (cardinality) matching and maximum weight matching algorithms for bipartite graph are summarized in Table 4.1 and Table 4.2, respectively. The algorithms with best asymptotic bound are indicated by "$*$" in these tables. A more detailed and technical discussion of the algorithms can be found in textbooks [74, 97, 104].

Maximum Cardinality Matching

For unweighted bipartite graphs, Hopcroft-Karp [59] algorithm, which is based on augmenting path technique, offers the best known performance for finding maximum matching in time $O(|E|\sqrt{|V|})$. In case of dense unweighted bipartite graphs, that is with $|E| = \Theta(|V|^2)$, slightly better algorithms exist. An algorithm by Alt et al. [4] obtains a maximum matching in $O(|V|^{1.5}\sqrt{|E|/\log|V|})$ time. In case of $|E| = \Theta(|V|^2)$, this becomes $O(|E|\sqrt{|V|/\log|V|})$ and is also $\sqrt{\log|V|}$-factor faster than Hopcroft-Karp algorithm. This speed up is obtained by an application of the fast adjacency matrix scanning technique of Cheriyan, Hagerup and Mehlhorn [27]. The algorithm proposed by Feder-Motwani [45] has the time complexity $O(|E|\sqrt{|V|}/k(|V|,|E|))$, where $k(x,y) = \log x/\log(x^2/y)$.

Table 4.1: Complexity survey of maximum unweighted bipartite matching algorithms.

Year	Author(s)	Complexity
1973 *	Hopcroft and Karp [59]	$O(\|E\|\sqrt{\|V\|})$
1991	Alt, Blum, Mehlhorn and Paul [4]	$O(\|V\|^{1.5}\sqrt{\|E\|/\log\|V\|})$
1995 *	Feder and Motwani [45]	$O(\|E\|\sqrt{\|V\|}/k(\|V\|,\|E\|))$

Maximum Weight Bipartite Matching

Several algorithms have also been proposed for computing maximum weight bipartite matching, improving both theoretical and practical running times. The well known Hungarian method, the first polynomial time algorithm, was introduced by Kuhn [75] and Munkres [89]. Fredman and Tarjan [46] improved this with running time $O(|V|(|E| + |V|\log|V|))$ for sparse graph by using Fibonacci heaps. An $O(|V|^{3/4}|E|\log N)$-time scaling algorithm was proposed by Gabow [48] under the assumption that edge weights are integers. A different and faster scaling algorithm was given by Gabow and Tarjan [49] with running time $O(\sqrt{|V|}|E|\log(|V|N))$. Kao et al. [70] proposed an $O(\sqrt{|V|}W/k(|V|,W/N))$-time decomposition technique under the assumptions that weights on the edges are positive and $W = o(|E|\log(|V|N))$.

In addition to the above exact algorithms, several randomized and approximate algorithms are also proposed, see for example [42, 96].

Table 4.2: Complexity survey of maximum weight bipartite matching algorithms.

Year(s)	Author(s)	Complexity
1955, 1957	Kuhn [75], Munkres [89]	$O(\|V\|^4)$ (Hungarian method)
1960	Iri [67, 97]	$O(\|V\|^2\|E\|)$
1969	Dinic and Kronrod [40, 97]	$O(\|V\|^3)$
1984, 1987 *	Fredman and Tarjan [46]	$O(\|V\|(\|E\| + \|V\| \log \|V\|))$
1985	Gabow [48]	$O(\|V\|^{3/4}\|E\| \log N)$
1989 *	Gabow and Tarjan [49]	$O(\sqrt{\|V\|}\|E\| \log(\|V\|N))$
1999	Kao, Lam, Sung and Ting [69]	$O(\sqrt{\|V\|}W)$
2001 *	Kao, Lam, Sung and Ting [70]	$O(\sqrt{\|V\|}W/k(\|V\|, W/N))$
2014 * (This work)		$O(\sqrt{\|V\|}W')$
		$O((\|V\|/\log \|V\|)^{1/2}W')$
		$O(\sqrt{\|V\|}W'/k(\|V\|, W'/N))$

4.3 Refined Decomposition Theorem for MWBM

We now propose a modified decomposition theorem which is a generalization of the existing decomposition theorem originally proposed by Kao et al. [70] and use it to develop a revised version of the decomposition algorithm to decrease the number of iterations and speed up the computation of the weight of a MWBM. Let $G = (V = V_1 \cup V_2, E, Wt)$ be an undirected, weighted bipartite graph having V_1 and V_2 as partition of vertex set V. Further, let $E = \{e_1, e_2, \ldots, e_{|E|}\}$ be set of edges of G with weights $Wt(e_i) = w_i$ for $1 \leq i \leq |E|$, where $w_1, w_2, \ldots, w_{|E|}$ are not necessarily distinct. As defined earlier, let N be the maximum edge weight, that is, for all $i \in \{1, 2, \ldots, |E|\}$, $0 \leq w_i \leq N$, and $W = \sum_{1 \leq i \leq |E|} w_i$ be the total weight of G.

Our algorithm considers several intermediate graphs with lighter edge weights. During this process it is possible that weights of some of the edges may be zero. An edge $e \in E$ is said to be *active* if its weight $Wt(e) > 0$, otherwise it is said to be *inactive*, that is when $Wt(e) = 0$. Let there be m' ($\leq |E|$) distinct edge weights in current working graph where $w_1 < w_2 < \cdots < w_{m'-1} < w_{m'}$. We denote the first two distinct maximum edge weights in current working graph by H_1 and H_2 ($< H_1$), respectively. Assign $H_2 = 0$ in case $m' = 1$.

We first build two new graphs referred to as G_h and G_h^{Δ} from a given

weighted bipartite graph G. For any integer $h \in [1, N]$, we decompose the graph G into two lighter weighted bipartite graph G_h and G_h^{Δ} as proposed by Kao et al. [69,70]. A minimum weight cover is a dual of maximum weight matching [70]. A *cover* of G is a function C: $V_1 \cup V_2 \rightarrow \mathbb{N}_0$ such that $C(v_1) + C(v_2) \geq Wt(v_1, v_2)$ $\forall v_1 \in V_1$ and $v_2 \in V_2$. Let $Wt(C) = \sum_{x \in V_1 \cup V_2} C(x)$. A cover C is *minimum weight cover* if $Wt(C)$ is minimum.

Formation of G_h from G: The graph G_h is formed by including those edges $\{u, v\}$ of G whose weights $Wt(u, v)$ lie in the range $[N-h+1, N]$. Each edge $\{u, v\}$ in graph G_h is assigned weight $Wt(u, v) - (N - h)$. For illustration, G_1 is constructed by the maximum weight edges of G and assigned unit weight to each edge.

Formation of G_h^{Δ} from G: Let C_h be the minimum weight cover of G_h. The graph G_h^{Δ} is formed by including every edge $\{u, v\}$ of G whose weight satisfies the condition

$$Wt(u, v) - C_h(u) - C_h(v) > 0.$$

The weight assigned to such an edge is $Wt(u, v) - C_h(u) - C_h(v)$.

Theorem 4.1 (The Decomposition Theorem [70])**.** *Let G be an undirected, weighted bipartite graph. Then*

(a) for any integer $h \in [1, N]$,

$$Wt(mwm(G)) = Wt(mwm(G_h)) + Wt(mwm(G_h^{\Delta})),$$

(b) in particular (trivial), for $h = 1$,

$$Wt(mwm(G)) = Wt(mm(G_1)) + Wt(mwm(G_1^{\Delta})).$$

Note that the Theorem 4.1(b) is derived from Theorem 4.1(a), since for $h = 1$, we have

$$mwm(G_1) = mm(G_1)$$

and

$$Wt(mwm(G_1)) = Wt(mm(G_1)) = |mm(G_1)|.$$

The Theorem 4.1(b) is used recursively in the Algorithm 4.1 [70], to compute the weight of a maximum weight matching of the graph G.

Algorithm 4.1 Kao et al.'s algorithm [70] to compute weight of a MWBM.

Input: A weighted, undirected, complete bipartite graph G with positive integer weights on the edges.

Output: Weight of a maximum weight matching of G, that is, $Wt(mwm(G))$.

Compute-MWM(G)
1: Construct G_1 from G.
2: Compute $mm(G_1)$ and find a minimum weight cover C_1 of G_1.
3: Construct G_1^{Δ} from G and C_1.
4: **if** G_1^{Δ} is empty,
5: **then return** $Wt(mm(G_1))$;
6: **else return** $Wt(mm(G_1))$+Compute-MWM(G_1^{Δ}).

Remark 4.2. *A graph G may not have all edge weights distinct. Consider the set of distinct edge weights of G. The Algorithm 4.1 works efficiently only when the largest edge weight differs by exactly one from the second largest edge weight of the current graph during an invocation of Theorem 4.1(b) in an iteration.*

Remark 4.3. *Observe that for arbitrary $h \in [1, N]$, $mwm(G_h)$ need not be equal to $mm(G_h)$, that is, we cannot always conclude that $mwm(G_h) = mm(G_h)$.*

One of our objective is to investigate those values of h for which $mwm(G_h)$ is equal to $mm(G_h)$ apart from the trivial value of h as 1 in each iteration of the Algorithm 4.1 to generate G_h having all its edge weights as 1.

In order to get the speed up whenever possible, by decreasing the number of iterations whenever possible, we revise the Theorem 4.1(b) and propose Theorem 4.4 which gives a domain of $h \in [1, N]$ where $mwm(G_h) = mm(G_h)$ and as a consequence of that we can write

$$Wt(mwm(G_h)) = Wt(mm(G_h)) = h * |mm(G_h)|.$$

It works for $h = 1$ and performs well especially when the largest edge weight differs by more than one from the second largest edge weight in the current graph in a decomposition step during an iteration.

Theorem 4.4 (The Modified Decomposition Theorem)**.** *The following equalities hold for any integer $h \in [1, H_1 - H_2]$ where H_1 and H_2 ($< H_1$) are the first two distinct maximum edge weights of graph G, respectively. We assign $H_2 = 0$ in case all edge weights are equal.*

(a) $mwm(G_h) = mm(G_h)$,

(b) $Wt(mwm(G)) = h * Wt(mm(G_h)) + Wt(mwm(G_h^{\Delta}))$.

Proof. The proof of the above statements are based on the construction of new graphs G_h and G_h^{Δ} from G and Theorem 4.1(a).

(a) To prove that for any integer h where $1 \leq h \leq H_1 - H_2$, $mwm(G_h) = mm(G_h)$ holds true, it is enough to prove the same for the maximum value[1] of h, that is, for $h = H_1 - H_2$. As specified earlier, the construction of G_h is done by choosing those edges $\{u, v\}$ of G that have weight

$$Wt(u, v) \in [N - h + 1, N] = [H_1 - (H_1 - H_2) + 1, H_1] = [H_2 + 1, H_1].$$

Since $H_1 \in [H_2 + 1, H_1]$, G_h has only the heaviest edges of G and each such edge is assigned the same weight. Thus, $mwm(G_h) = mm(G_h)$ for $h = H_1 - H_2$.

(b) Observe that $h \in [1, H_1 - H_2]$ and $[1, H_1 - H_2] \subseteq [1, N]$. So, by using Theorem 4.1(a) we have, $\forall h \in [1, H_1 - H_2]$,

$$Wt(mwm(G)) = Wt(mwm(G_h)) + Wt(mwm(G_h^{\Delta})).$$

Also by using the Theorem 4.4(a), $mwm(G_h) = mm(G_h)$ for all $h \in [1, H_1 - H_2]$. Weight of each edge[2] $\{u, v\}$ in G_h is exactly $Wt(u, v) - (N - h) = H_1 - (H_1 - h) = h$. Therefore,

$$Wt(mwm(G_h)) = h * Wt(mm(G_h)) = h * |mm(G_h)|.$$

Hence for any integer $h \in [1, H_1 - H_2]$,

$$\begin{aligned} Wt(mwm(G)) &= Wt(mwm(G_h)) + Wt(mwm(G_h^{\Delta})) \\ &= h * Wt(mm(G_h)) + Wt(mwm(G_h^{\Delta})). \end{aligned}$$

This completes the proof. □

Remark 4.5. *The equality $mwm(G_h) = mm(G_h)$ in Theorem 4.4(a) is not true for $h > H_1 - H_2$ and $h \leq N$.*

[1] For illustration, consider $h = c$ where $1 \leq c \leq H_1 - H_2$. Then as per the formation of G_h from G, G_c is built by choosing those edges of G that have weight $Wt(u, v) \in [N - (c - 1), N]$. Since, $c - 1 \geq 0$ and $N \in [N - (c - 1), N]$ for any $c \in [1, H_1 - H_2]$, G_c has only the heaviest edges of G. For optimization, choose $h = H_1 - H_2$, the maximum possible value of h.

[2] Only maximum weight edges of G are included in G_h.

To show that for any $h \in [H_1 - H_2 + 1, N]$ the statement $mwm(G_h) = mm(G_h)$ is not true, it is enough to show the same essentially for $h = H_1 - H_2 + 1$. Observe that $h = H_1 - H_2 + 1 \geq 2$, since $H_1 > H_2$. According to the construction of G_h, it is formed by edges $\{u, v\}$ of G whose weights $Wt(u, v) \in [N - h + 1, N] = [H_1 - (H_1 - H_2 + 1) + 1, H_1] = [H_2, H_1]$, that is, G_h is built with the maximum weight edges and second maximum weight edges of G, because $\{H_1, H_2\} \in [H_2, H_1]$. The weight of each heaviest edge $\{u, v\}$ of G in G_h is exactly

$$Wt(u, v) - (N - h) = H_1 - (H_1 - h) = h$$

which is greater than or equal to 2 and that of each second heaviest edge $\{u, v\}$ of G in G_h is exactly

$$Wt(u, v) - (N - h) = H_2 - (H_1 - h) = (H_2 - H_1) + h = (1 - h) + h = 1.$$

Hence $mwm(G_h) \neq mm(G_h)$ for such a value of h.

Example 4.6. Consider the graph shown in the Figure 4.1(a). Let $h = H_1 - H_2 + 1$. So, $h = H_1 - H_2 + 1 = 9 - 4 + 1 = 6$. As shown in the Figure 4.1(b), G_h is formed by the edges $\{u, v\}$ whose weights $Wt(u, v) \in [N - h + 1, N] = [9 - 6 + 1, 9] = [4, 9]$ and their respective calculated weights are 6 and 1. Hence $mwm(G_h) \neq mm(G_h)$. □

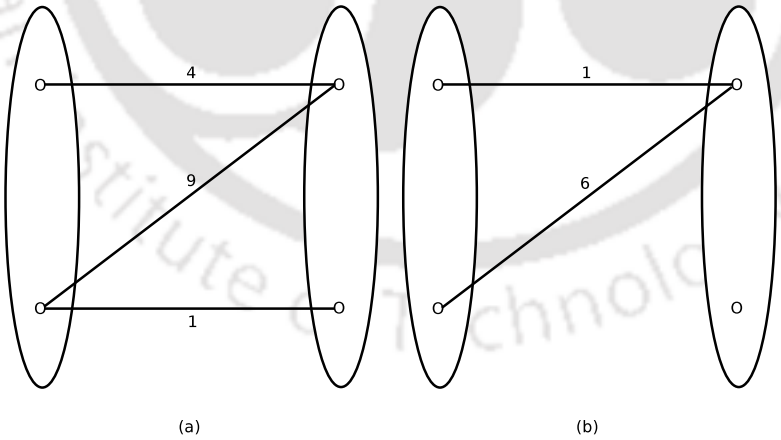

Figure 4.1: **(a)** An undirected bipartite graph G with positive integer weights on the edges. **(b)** Considering $h = H_1 - H_2 + 1 = 6$, G_h is extracted, but $mwm(G_h) \neq mm(G_h)$.

We use the modified decomposition Theorem 4.4 to design a recursive Algorithm 4.2 to compute the weight of a $mwm(G)$.

Algorithm 4.2 Compute weight of a maximum weight matching of G.

Input: A weighted, undirected, complete bipartite graph G with positive integer weights on the edges.

Output: Weight of a maximum weight matching of G, that is, $Wt(mwm(G))$.

WT-MWBM(G)
1: Assume that initially $Wt(mwm(G)) = 0$.
2: Find $h = H_1 - H_2$ from the current working graph G.
3: Construct G_h from G.
4: Compute $mm(G_h)$.
5: Find minimum weight cover C_h of G_h.
6: Construct G_h^{Δ} from G and C_h.
7: **if** G_h^{Δ} is empty (that is, G_h^{Δ} has no active edge)
8: **then return** $h * |mm(G_h)|$;
9: **else return** $h * |mm(G_h)|$ + WT-MWBM(G_h^{Δ}).

Example 4.7. Consider the bipartite graph shown in Figure 4.2(a). The Algorithm 4.2 finds the weight of a MWBM in just two iterations, as the algorithm is designed for the best h in every invocation of WT-MWBM( ), whereas algorithm by Kao et al. [70] requires 500 iterations because it considers $h = 1$ in every invocation of Compute-MWM( ). □

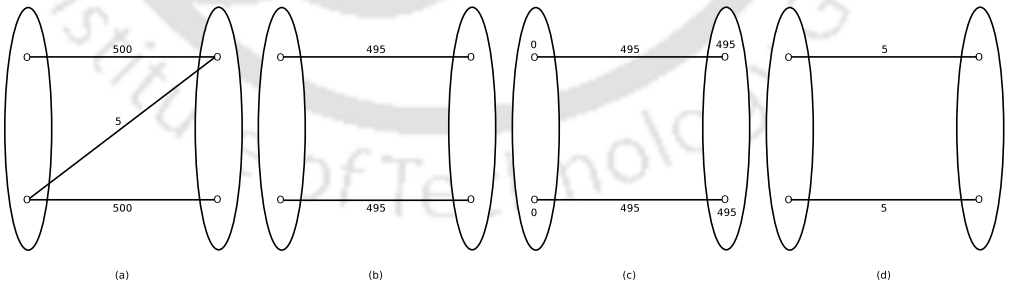

Figure 4.2: **(a)** An undirected, weighted bipartite graph G with positive integer weights on the edges. In the current graph G, $h = 495$. **(b)** G_h is extracted. **(c)** C_h is the weighted cover of G_h. **(d)** G_h^{Δ} is formed from G_h and C_h. Compute WT-MWBM(G_h^{Δ}).

Correctness of the algorithm follows from the construction of G_h and G_h^{Δ} and the modified decomposition Theorem 4.4.

4.4 Complexity of the Modified Algorithm

Let $G = (V = V_1 \cup V_2, E, Wt)$ be the initial input graph and N denotes the maximum edge weight of G, that is, for all $i \in \{1, 2, \ldots, |E|\}$, $0 \leq w_i \leq N$ and $W = \sum_{1 \leq i \leq |E|} w_i$ is the total weight of G. Further, let $\{w_1, \ldots, w_{m'}\}$ be the set of distinct edge weights of G, where $m' \leq |E|$.

Based on the constructions of G_h and G_h^{Δ}, the modified decomposition Theorem 4.4 and the Algorithm 4.2, we can easily observe that in worst case the maximum number of possible iterations of Wt-Mwbm( ) is N, when $h = 1$ in each iteration in the current working graph. Whereas in the best case, all the edge weights of G are the same and so we will have $h = N$ for the present decomposition. As a consequence the algorithm will terminate in the first iteration itself.

As the complexity analysis of the Algorithm 4.2 is almost similar to that presented elsewhere [70], the details are available in Appendix A (see page 99). The algorithm takes $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time to compute the weight of a $mwm(G)$ by using the algorithm by Feder and Motwani [45], as a subroutine.

Let L_i consists of edges of remaining G (after $i-1$-th iteration) whose weights reduce in G_h^{Δ} in i-th iteration. Also let there be p iterations, $l_i = |L_i|$ for $i = 1, 2, \ldots, p \leq N$ and $h_i = H_{i1} - H_{i2}$ in the i-th iteration, where H_{i1} and H_{i2} $(< H_{i1})$ are the first two distinct maximum edge weights of the remaining graph G after the $i-1$-th iteration.

From the detailed complexity analysis we have, $l_1h_1 + l_2h_2 + \cdots + l_ph_p = W$. Let $l_1 + l_2 + \cdots + l_p = W'$. Observe that, in worst case, if $h_i = 1$ for all $i \in [1, p]$, then $W' = \sum_{i=1}^{p} l_i = W$. And in best case, if $h_1 = N$, then $W' = |E|$. Moreover, the parameter W' is smaller than W, essentially when the largest edge weight differs by more than one from the second largest edge weight in the current working graph in decomposition step during an iteration of the algorithm. Therefore in best case[3], it requires $O(\sqrt{|V|}|E|/k(|V|, |E|))$ time and in worst case $O(\sqrt{|V|}W/k(|V|, W/N))$, to compute weight of a maximum weight matching. That is, $|E| \leq W' \leq W$.

This time complexity bridges a gap between the best known time complexity for computing a Maximum Cardinality Matching (MCM) of unweighted bipartite graph and that of computing a MWBM of a weighted bipartite

[3] In best case, all the edge weights of G are the same. So, the algorithm terminates in just one iteration and hence $W' = O(|E|)$.

graph. In best case, for computation of weight of a MWBM, the Algorithm 4.2 takes $O(\sqrt{|V|}|E|/k(|V|,|E|))$ time which is the same as the complexity of the Feder and Motwani's algorithm [45] for computing MCM of unweighted bipartite graph; whereas in worst case it (Algorithm 4.2) takes $O(\sqrt{|V|}W/k(|V|,W/N))$ time which is the same as the complexity of the Kao et al.'s algorithm [70]. However, it is very difficult and challenging to get rid of W or N from the complexity. This modified algorithm works well for general W, but is best known for $W' = o(|E|\log(|V|N))$.

Some other advantages of the modified decomposition algorithm is stated by the following propositions. Let $G = (V, E, Wt)$ be an undirected, weighted bipartite graph, $E = \{e_1, e_2, \ldots, e_{|E|}\}$ be the set of positive weight edges with weights $Wt(e_i) = w_i > 0$ (where $1 \leq i \leq |E|$), N be the maximum edge weight and $W = \sum_{1\leq i\leq |E|} w_i$ be the total weight of G. The modified decomposition Algorithm 4.2 computes weight of a maximum weight bipartite matching of G in $O(\sqrt{|V|}W'/k(|V|,W'/N))$ time, where $|E| \leq W' \leq W$.

Proposition 4.8 (Multiplicative Scaling Property)**.** *Let $\widehat{G}$ be a new weighted bipartite graph constructed by multiplying a large constant $\alpha \in \mathbb{N}$ to each w_i of G. Then for both G and $\widehat{G}$, the complexity of the Algorithm 4.2 remains $O(\sqrt{|V|}W'/k(|V|,W'/N))$ where $|E| \leq W' \leq W$; whereas for the graph $\widehat{G}$, the complexity of the Algorithm 4.1 becomes $O(\sqrt{|V|}\alpha W/k(|V|,\alpha W/N))$.*

Proof. As mentioned in the detailed complexity analysis of the Algorithm 4.2 (described in Appendix A, page 99), let L_i consists of edges of remaining graph G (left after $i-1$-th iteration), whose weights reduce in G_h^{Δ} in the i-th iteration of Wt-Mwbm( ). Assume that there be p iterations for the Algorithm 4.2, $l_i = |L_i|$ for $i = 1, 2, \ldots, p \leq N$ and $h_i = H_{i1} - H_{i2}$ in the i-th iteration, where H_{i1} and $H_{i2}\,(< H_{i1})$ are the first two distinct maximum edge weights of the remaining graph G after $i-1$-th iteration. From the detailed complexity analysis we have,

$$l_1h_1 + l_2h_2 + \cdots + l_ph_p = W \quad \text{and} \quad l_1 + l_2 + \cdots + l_p = W'.$$

Observe that for the new graph $\widehat{G}$, the number of iterations in Algorithm Wt-Mwbm($\widehat{G}$) still remains p and in the computation of Wt-Mwbm($\widehat{G}$), L_i consists of l_i number of edges of the remaining graph $\widehat{G}$ (after $i-1$-th iteration), whose weights reduce in $\widehat{G}_h^{\Delta}$ in i-th iteration of Wt-Mwbm( ). In this case, if $h'_i = H'_{i1} - H'_{i2}$ in the i-th iteration, where H'_{i1} and $H'_{i2}\,(< H'_{i1})$ are the first two distinct maximum edge weights of the remaining graph $\widehat{G}$ after $i-1$-th iteration, respectively, then

$$h'_i = H'_{i1} - H'_{i2} = \alpha * H_{i1} - \alpha * H_{i2} = \alpha * h_i \quad \text{where} \quad i = 1, 2, \ldots, p,$$

and

$$l_1h'_1 + l_2h'_2 + \cdots + l_ph'_p = \alpha W \quad \text{and} \quad l_1 + l_2 + \cdots + l_p = W'.$$

Therefore, the modified Algorithm 4.2 will take $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time to compute the weight of a $mwm(\widehat{G})$ by using the algorithm by Feder and Motwani [45] as a subroutine; whereas Kao et. al.'s Algorithm 4.1 requires $O(\sqrt{|V|}\alpha W/k(|V|, \alpha W/N))$ time. □

That is, multiplication by an integer constant to all the weight of edges of a weighted bipartite graph G does not affect the time complexity of the modified decomposition algorithm for computing the weight of a MWBM of the bipartite graph. The following remark talks about a conditional scaling down property of the algorithm for the graph G.

Remark 4.9. *Let we scale down each edge weights of G by multiplying a factor of $\frac{1}{\alpha}$ and get a new graph $\widehat{G}$, where α is the Greatest Common Divisor (GCD) of the positive edge weights of G. Then the time complexity of the Algorithm 4.2 for computing a MWBM of both the graphs G and $\widehat{G}$ remains same.*

Though during the complexity calculation of Algorithm 4.2 we have stated a bound for W' as: $|E| \leq W' \leq W$, but the following proposition gives a more better bound of the parameter W'.

Proposition 4.10 (GCD Property)**.** *Let $G = (V, E, Wt)$ be an undirected, weighted bipartite graph and $E = \{e_1, e_2, \ldots, e_{|E|}\}$ be the set of positive weight edges with weights $Wt(e_i) = w_i > 0$ for $1 \leq i \leq |E|$. Further, let the GCD of the positive edge weights of G is denoted by $GCD(w_1, w_2, \ldots, w_{|E|})$, then*

$$|E| \leq W' \leq \frac{W}{GCD(w_1, w_2, \ldots, w_{|E|})} \leq W.$$

Proof. Without going into more detailed and repeated writing, as mentioned in the previous Proposition 4.8, we have:

$$l_1h_1 + l_2h_2 + \cdots + l_ph_p = W, \text{ where } h_i = H_{i1} - H_{i2} \quad \text{and}$$
$$l_1 + l_2 + \cdots + l_p = W'.$$

Let $g = \text{GCD}(w_1, w_2, \ldots, w_{|E|})$. Observer that, for any iteration i (where $i = 1, 2, \ldots, p$) both H_{i1} and H_{i2} are divisible by g. Hence, according to the

definition of h_is, each $h_i = H_{i1} - H_{i2}$ is also divisible by the factor g.

$$\begin{aligned}\therefore \quad W' &= l_1 + l_2 + \cdots + l_p \\ &\leq l_1(h_1/g) + l_2(h_2/g) + \cdots + l_p(h_p/g) \\ &\leq (l_1h_1 + l_2h_2 + \cdots + l_ph_p)/g = \frac{W}{g} \leq W.\end{aligned}$$

This completes the proof. □

We also analyze the complexity of the Algorithm 4.2 by considering the Hopcroft-Karp algorithm [59] and Alt-Blum-Mehlhorn-Paul algorithm [4] as base algorithms.

With Respect to the Hopcroft-Karp Algorithm: Hopcroft-Karp algorithm [59] presents the best known worst-case performance for getting a maximum matching in a bipartite graph with runtime of $O(\sqrt{|V|}|E|)$. Hence the recurrence relation for running time of the Algorithm 4.2 with respect to Hopcroft-Karp algorithm is

$$\begin{aligned} & T(|V|, W', N) = O(\sqrt{|V|}l_1) + T(|V|, W'', N'') \\ \text{and} \quad & T(|V|, 0, 0) = 0 \end{aligned}$$

$$\begin{aligned}\therefore \; T(|V|, W', N) &= O(\sqrt{|V|}l_1) + O(\sqrt{|V|}l_2) + \cdots + O(\sqrt{|V|}l_p) \\ &= O\left(\sqrt{|V|}\sum_{i=1}^{p} l_i\right) = O(\sqrt{|V|}W').\end{aligned}$$

With Respect to the Alt-Blum-Mehlhorn-Paul Algorithm: A bit better algorithm for dense bipartite graph is Alt-Blum-Mehlhorn-Paul algorithm [4] which is $(\log|V|)^{1/2}$-factor faster than Hopcroft-Karp algorithm for maximum bipartite matching. Hence the time complexity, with respect to Alt-Blum-Mehlhorn-Paul algorithm as a base algorithm, is $O((|V|/\log|V|)^{1/2}W')$ and it is $(\log|V|)^{1/2}$-factor faster than the above case.

4.5 Finding a Maximum Weight Matching

The Algorithm 4.2 computes only the weight of a $mwm(G)$ of a given graph G. To find the edges of a $mwm(G)$, we give a revised algorithm for constructing a Minimum Weight Cover (MWC) of G which is a dual of maximum weight matching. As mentioned before, a *cover* of G is a function C: $V_1 \cup V_2 \to \mathbb{N}_0$ such that $C(v_1) + C(v_2) \geq Wt(v_1, v_2) \;\; \forall v_1 \in V_1$ and $v_2 \in V_2$. Let $Wt(C) = \sum_{x \in V_1 \cup V_2} C(x)$. We say C is *minimum weight cover* if $Wt(C)$ is minimum. Let C be a MWC of a graph G.

Lemma 4.11 ([70]). *Let C_h^Δ be any minimum weight cover of G_h^Δ. If C is a function on $V(G)$ such that for every $u \in V(G)$, $C(u) = C_h(u) + C_h^\Delta(u)$, then C is minimum weight cover of G.*

Using this lemma we design an $O(\sqrt{|V|}W'/k(|V|, W'/N))$-time revised algorithm to compute a MWC of G. The correctness of this algorithm is clear from the Lemma 4.11 and the time complexity analysis is similar to that given in the previous section.

Algorithm 4.3 Calculate a MWC C of G.

Input: A weighted, undirected, complete bipartite graph G with positive integer weights on the edges.

Output: A minimum weight cover C of G.

MWC(G)
1: Assume that initially $Wt(mwm(G)) = 0$.
2: Find $h \leftarrow H_1 - H_2$ from the current working graph G.
3: Construct G_h from G.
4: Compute $mm(G_h)$.
5: Find minimum weight cover C_h of G_h.
6: Construct G_h^Δ from G and C_h.
7: **if** G_h^Δ is empty (that is, G_h^Δ has no active edge)
8: **then return** C_h;
9: **else**
10: $C_h^\Delta \leftarrow$ MWC(G_h^Δ);
11: **return** C, where $C(u) = C_h(u) + C_h^\Delta(u)$ for all nodes u in G.

Now as deduced by Kao et al. in [70], finding a maximum weight matching by using the given vertex cover takes $O(\sqrt{|V|}|E|/k(|V|, |E|))$ time. Since $|E| \leq W' \leq W$, so altogether $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time requires to find a MWBM of G.

4.6 Experimental Evaluation

The Algorithm 4.2 is efficient because of the modified decomposition Theorem 4.4. In order to understand the practical importance of the Algorithm 4.2, we report experimental evaluations of the same for the randomly generated bipartite graphs.

Implementation and Experimental Environments

We have implemented both Kao et al.'s algorithm [70] and Algorithm 4.2 in `C++` and compiled them using `g++` *4.8.2-19ubuntu1* compiler. All the experiments have been performed on a Desktop PC with an *Intel® Xeon®(E5620 @ 2.40 GHz) Processor*, *32.00 GB RAM* and *1200 GB Hard Disk*, running the *Ubuntu 14.04.1 LTS* (*Trusty Tahr*) *64-bit Operating System*.

Input Data Description and Its Randomness

For a frame of fixed number of vertices in a partition of the vertex set and fixed weight of bipartite graph G, we have generated the random weighted G by assigning random (uniformly distributed) weight to the randomly (uniformly distributed) picked up edges of G. The outputs of these experiments for an input bipartite graph G are:

(a) the number of iterations of Wt-Mwbm( ) and Compute-Mwm( ) in the Algorithms 4.2 and 4.1, respectively, for the graph G and

(b) total time taken by the respective algorithms to compute the weight of a MWBM of G.

As mentioned earlier in Section 2.7 (see page 16) the APSM problem under Hamming distance error model is computationally equivalent to the MWBM problem in graph theory [57]. The input data relation between the above problems are:

(a) length of the pattern is equal to weight of the bipartite graph, and

(b) alphabet size of the pattern is equal to number of vertices in a partition of the vertex set of the corresponding bipartite graph.

Experimental Results

We have tested the respective algorithms with large input data sets. The details are given below. In all the graphs, the output of our Algorithm 4.2 (denoted in short by "Modified Algorithm") corresponds to the unbroken line, whereas that of for the Algorithm 4.1 (denoted in short by "Kao et al.'s Algorithm") corresponds to the dotted line.

Experiment 4.12. This experiment is done for a total of 250 pseudo-randomly generated bipartite graphs, each of its weight is fixed to 1000 unit where size of each of the partitions of the vertex set of bipartite graph varies from 2 to 26.

Table 4.3: Efficiency comparison between the Algorithms 4.2 and 4.1 for the 250 pseudo-randomly generated weighted bipartite graphs as considered in Experiment 4.12.

# Vertices in a Partition	Weight of Graph	Algorithm 4.2		Algorithm 4.1 (by Kao et al.)	
		# Iterations	Time (Sec.)	# Iterations	Time (Sec.)
2	1000	8.40	0.000050	422.60	0.002356
3	1000	8.20	0.000084	397.60	0.003769
4	1000	6.60	0.000104	402.20	0.005922
5	1000	7.20	0.000146	444.40	0.009566
6	1000	8.60	0.000214	408.60	0.010882
7	1000	7.00	0.000251	418.60	0.014451
8	1000	10.40	0.000405	455.20	0.019964
9	1000	7.20	0.000376	366.40	0.018605
10	1000	7.40	0.000411	400.20	0.024284
11	1000	8.60	0.000525	391.20	0.025592
12	1000	9.00	0.000670	391.20	0.031218
13	1000	8.40	0.000722	391.20	0.035947
14	1000	8.40	0.000811	391.20	0.039951
15	1000	10.20	0.001089	391.20	0.044228
16	1000	9.60	0.001156	391.20	0.048768
17	1000	9.40	0.001217	391.20	0.053661
18	1000	9.60	0.001411	391.20	0.058682
19	1000	11.00	0.001680	391.20	0.064456
20	1000	8.60	0.001492	391.20	0.070000
21	1000	11.80	0.002226	391.20	0.077403
22	1000	11.00	0.002284	391.20	0.083169
23	1000	13.40	0.002924	391.20	0.089702
24	1000	13.80	0.003254	391.20	0.097006
25	1000	13.00	0.003327	391.20	0.103550
26	1000	12.20	0.003342	391.20	0.110369

Each numerical row of the Table 4.3 is corresponding to 10 different random graphs, each of whose size of each partition of the vertex set and weight of the graphs are fixed. Only for this experimental result, each row reports the average output of 10 different random graphs, each of whose number of vertices and weight are fixed.

For example, the numerical row corresponding to '# Vertices in a partition' equal to 15 reports the following. For the 10 randomly generated different bipartite graphs, each of whose size of the vertex set is 30 and weight is 1000 unit. And on an average the number of iterations of Wt-Mwbm( ) and Compute-Mwm( ) in the Algorithms 4.2 and 4.1 are 10.20 and 391.20, respectively; whereas average time taken by the respective algorithms to compute the weight of a MWBM are 0.001089 and 0.044228 seconds.

Figure 4.3 shows the comparison on the number of iterations for the random graphs with different size partition of the vertices for the Algorithms 4.2 and 4.1. Similarly, Figure 4.4 gives the comparison time taken by the same algorithms for the random graphs with different size partition of the vertices. □

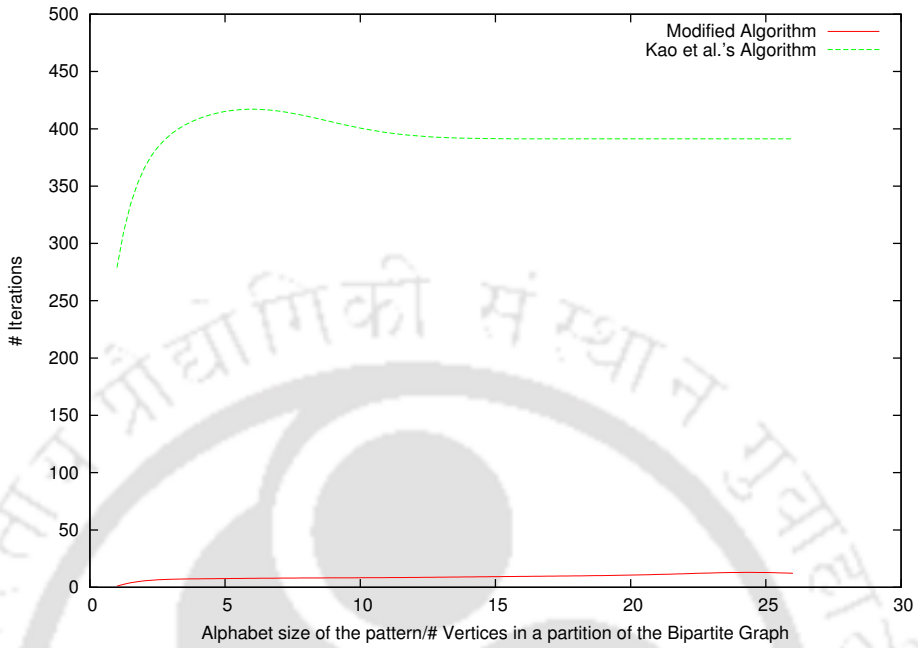

Figure 4.3: Partition size vs. Iteration graph corresponding to the Experiment 4.12. Weight of each input graph is fixed to be 1000 unit.

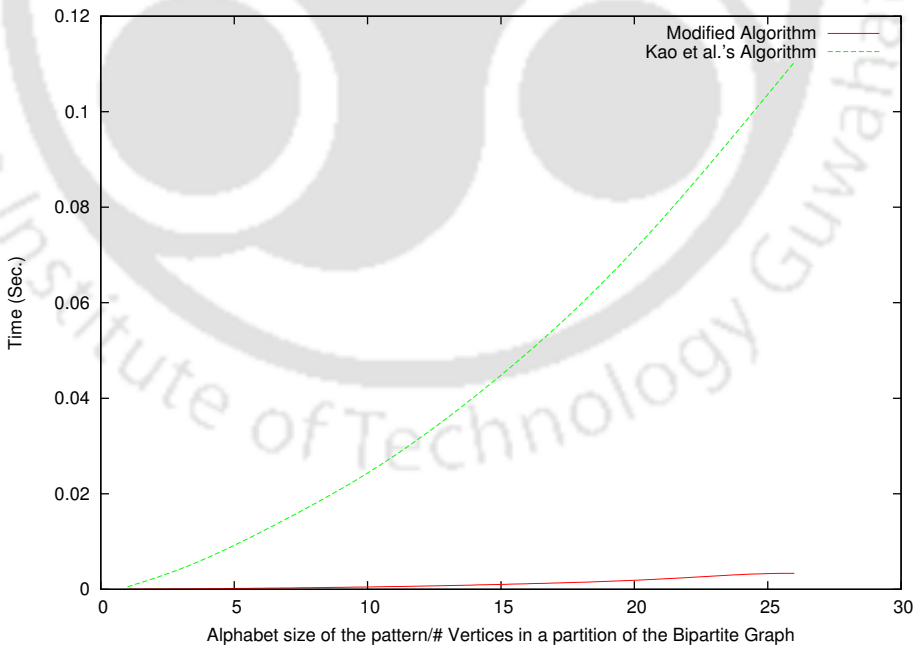

Figure 4.4: Partition size vs. Time graph corresponding to the Experiment 4.12. Weight of each input graph is fixed to be 1000 unit.

The next two experiments are done over the graphs corresponding to the randomly generated strings over the DNA alphabet $\Sigma = \{A, C, G, T\}$ of different lengths.

Experiment 4.13. In this experiment we have fixed the size of each partition of each graph to 4 and randomly generated a total of 62 bipartite graphs for 62 different weights. See Table 4.4 for more details. Unlike previous experiment, each row reports the iterations and time comparison of the Algorithms 4.2 and 4.1 on a randomly generated bipartite graph with fixed size vertex and weight. Figures 4.5 and 4.6 describe the pictorial representation of the Table 4.4. □

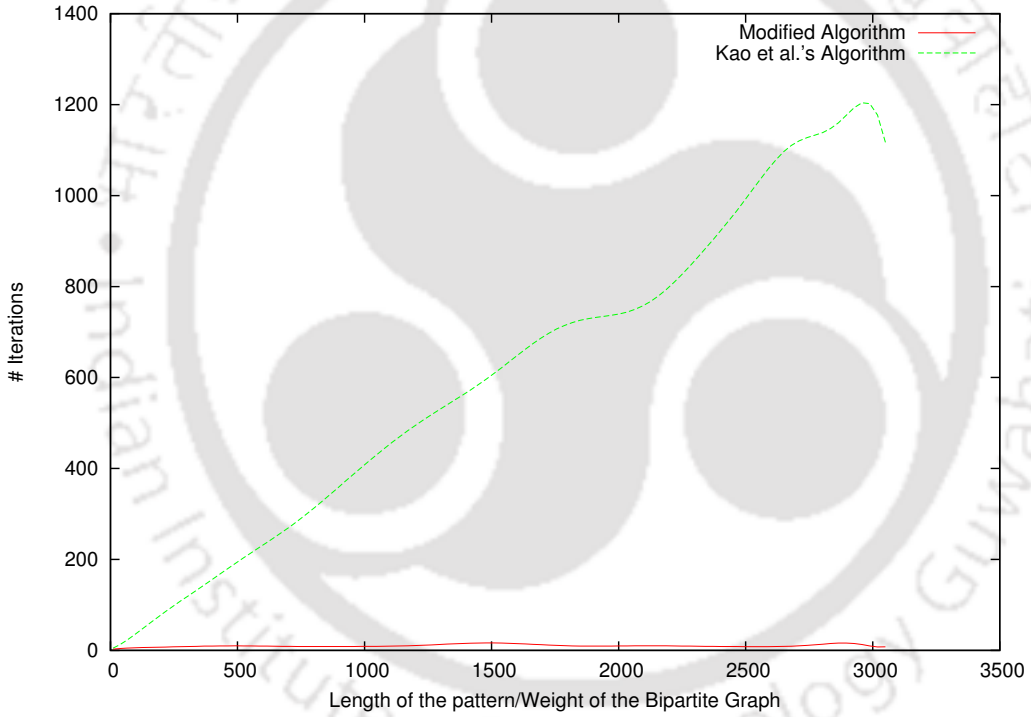

Figure 4.5: Weight vs. Iteration graph corresponding to the Experiment 4.13. The number of vertices in each partition of the vertex set is fixed to be 4.

Experiment 4.14. In the final experiment also we have fixed the size of a partition of each graph to 4 and but for a total of 71 randomly generated bipartite graphs for 71 different and large weights. See Table 4.5 for more details. □

Table 4.4: Experimental result for the 62 pseudo-randomly generated weighted bipartite graphs as considered in Experiment 4.13. The number of vertices in each partition of the vertex set of each of the graphs is fixed to be 4, but weight of the graph varies.

# Vertices in a Partition	Weight of Graph	Algorithm 4.2		Algorithm 4.1 (by Kao et al.)	
		# Iterations	Time (Sec.)	# Iterations	Time (Sec.)
4	10	3.00	0.000121	5.00	0.000152
4	50	4.00	0.000109	13.00	0.000229
4	100	9.00	0.000260	38.00	0.000965
4	150	6.00	0.000165	52.00	0.001257
4	200	5.00	0.000141	81.00	0.001641
4	250	5.00	0.000142	109.00	0.002703
4	300	18.00	0.000419	133.00	0.003255
4	350	5.00	0.000144	116.00	0.002648
4	400	6.00	0.000192	139.00	0.003666
4	450	4.00	0.000122	192.00	0.004288
4	500	31.00	0.000745	189.00	0.004863
4	550	6.00	0.000165	203.00	0.004828
4	600	6.00	0.000159	277.00	0.006564
4	650	6.00	0.000166	298.00	0.006679
4	700	9.00	0.000195	186.00	0.003468
4	750	8.00	0.000222	268.00	0.007218
4	800	7.00	0.000188	243.00	0.005828
4	850	8.00	0.000193	390.00	0.009581
4	900	8.00	0.000205	380.00	0.010258
4	950	11.00	0.000297	395.00	0.011187
4	1000	11.00	0.000248	368.00	0.009515
4	1050	12.00	0.000230	466.00	0.012499
4	1100	5.00	0.000135	535.00	0.012997
4	1150	6.00	0.000126	405.00	0.008078
4	1200	8.00	0.000205	397.00	0.008670
4	1250	6.00	0.000168	602.00	0.014516
4	1300	6.00	0.000168	648.00	0.016127
4	1350	13.00	0.000283	553.00	0.015214
4	1400	6.00	0.000158	639.00	0.016306
4	1450	10.00	0.000242	426.00	0.009439
4	1500	94.00	0.002554	456.00	0.011925
4	1550	9.00	0.000267	591.00	0.017001
4	1600	6.00	0.000162	775.00	0.016471
4	1650	7.00	0.000189	676.00	0.015674
4	1700	6.00	0.000162	665.00	0.013834
4	1750	6.00	0.000162	860.00	0.021617
4	1800	6.00	0.000163	701.00	0.017824
4	1850	6.00	0.000150	777.00	0.016954
4	1900	6.00	0.000142	827.00	0.019956
4	1950	8.00	0.000194	788.00	0.018098
4	2000	6.00	0.000168	690.00	0.016825
4	2050	6.00	0.000175	584.00	0.011463
4	2100	37.00	0.000864	727.00	0.016040
4	2150	5.00	0.000139	585.00	0.009586
4	2200	8.00	0.000186	747.00	0.016389
4	2250	6.00	0.000175	897.00	0.022177
4	2300	10.00	0.000252	1091.00	0.028346
4	2350	6.00	0.000164	740.00	0.014673
4	2400	8.00	0.000185	954.00	0.022879
4	2450	6.00	0.000168	998.00	0.023334
4	2500	18.00	0.000439	735.00	0.017273
4	2550	5.00	0.000159	1086.00	0.023379
4	2600	7.00	0.000200	1035.00	0.027333
4	2650	6.00	0.000155	1313.00	0.032712
4	2700	4.00	0.000133	1348.00	0.031344
4	2750	10.00	0.000254	922.00	0.020268
4	2800	9.00	0.000271	1030.00	0.028735
4	2850	15.00	0.000361	1206.00	0.029624
4	2900	42.00	0.000958	1144.00	0.026569
4	2950	5.00	0.000153	1266.00	0.030598
4	3000	6.00	0.000180	1249.00	0.033715
4	3050	8.00	0.000216	1117.00	0.027538

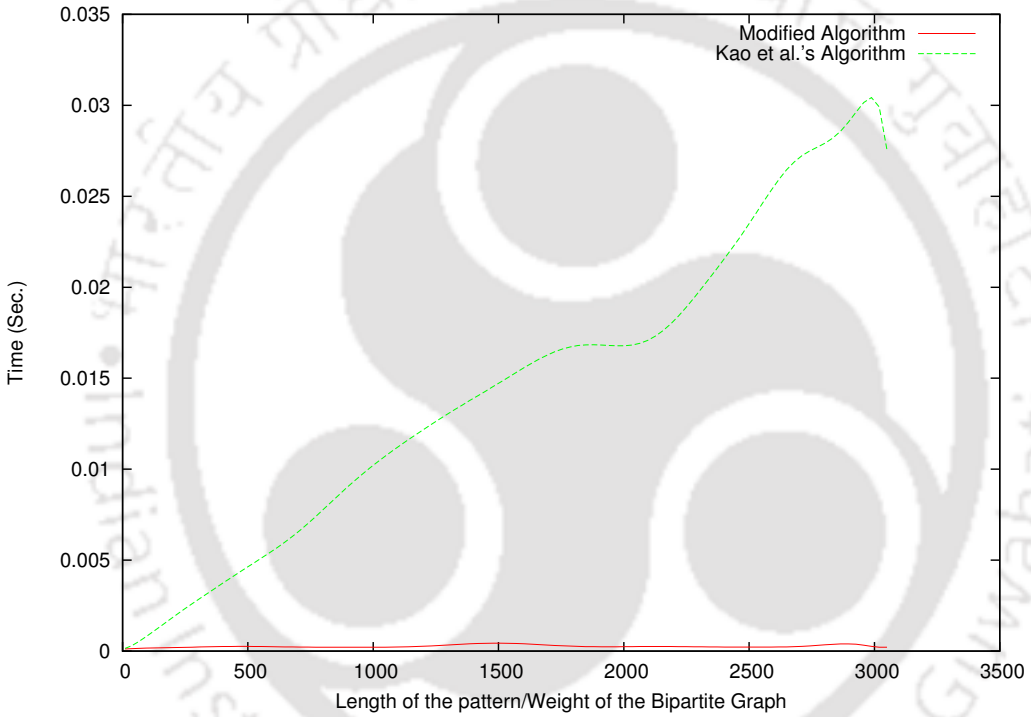

Figure 4.6: Weight vs. Time graph corresponding to the Experiment 4.13. The number of vertices in each partition of the vertex set is fixed to be 4.

Table 4.5: Experimental result for the 71 pseudo-randomly generated bipartite graphs as considered in Experiment 4.14. Cardinality of each partition of the vertex set is fixed to be 4, but weight of the graph varies largely.

# Vertices in a Partition	Weight of Graph	Algorithm 4.2		Algorithm 4.1 (by Kao et al.)	
		# Iterations	Time (Sec.)	# Iterations	Time (Sec.)
4	1000	11.00	0.000241	368.00	0.009156
4	10000	7.00	0.000196	4284.00	0.009156
4	20000	8.00	0.000193	7722.00	0.009156
4	30000	6.00	0.000161	14942.00	0.352380
4	40000	12.00	0.000326	19786.00	0.589580
4	50000	8.00	0.000213	22172.00	0.619935
4	60000	8.00	0.000207	28763.00	0.806167
4	70000	8.00	0.000215	22042.00	0.588712
4	80000	9.00	0.000190	28054.00	0.662195
4	90000	10.00	0.000240	21440.00	0.447115
4	100000	36.00	0.000970	32975.00	0.865868
4	110000	10.00	0.000222	53322.00	1.320364
4	120000	8.00	0.000199	57741.00	1.466250
4	130000	6.00	0.000156	54970.00	1.209947
4	140000	9.00	0.000204	50849.00	1.102937
4	150000	12.00	0.000221	65220.00	1.502660
4	160000	9.00	0.000183	44405.00	0.868532
4	170000	8.00	0.000188	71264.00	1.668321
4	180000	9.00	0.000212	68565.00	1.391208
4	190000	7.00	0.000182	58417.00	1.261169
4	200000	8.00	0.000199	84882.00	2.177765
4	210000	21.00	0.000411	55160.00	0.861068
4	220000	7.00	0.000171	90913.00	1.931699
4	230000	10.00	0.000247	87201.00	2.051072
4	240000	15.00	0.000315	112402.00	2.794717
4	250000	8.00	0.000180	102078.00	2.032557
4	260000	17.00	0.000353	105322.00	2.745294
4	270000	10.00	0.000251	88840.00	2.062840
4	280000	15.00	0.000368	94300.00	2.191243
4	290000	8.00	0.000191	79909.00	1.639328
4	300000	18.00	0.000364	105128.00	2.443022
4	310000	8.00	0.000188	120579.00	2.260459
4	320000	8.00	0.000216	125597.00	3.134282
4	330000	10.00	0.000236	151182.00	3.940926
4	340000	9.00	0.000196	166492.00	3.507791
4	350000	7.00	0.000181	151689.00	3.194340
4	360000	7.00	0.000186	166928.00	3.850720
4	370000	9.00	0.000212	167154.00	3.666068
4	380000	10.00	0.000162	147368.00	2.885593
4	390000	21.00	0.000388	137803.00	2.969532
4	400000	9.00	0.000202	158026.00	3.690540
4	410000	10.00	0.000256	153238.00	4.177567
4	420000	15.00	0.000350	168440.00	4.543331
4	430000	8.00	0.000174	195902.00	4.567199
4	440000	7.00	0.000192	165922.00	4.078337
4	450000	8.00	0.000199	183992.00	4.580142
4	460000	8.00	0.000190	208746.00	4.302464
4	470000	14.00	0.000234	229321.00	5.470295
4	480000	8.00	0.000210	199475.00	5.379294
4	490000	10.00	0.000266	239623.00	6.374744
4	500000	11.00	0.000238	186026.00	4.691640
4	510000	12.00	0.000298	201041.00	4.919859
4	520000	9.00	0.000218	189501.00	4.317819
4	530000	9.00	0.000205	151069.00	3.144941
4	540000	8.00	0.000219	230280.00	5.777490
4	550000	7.00	0.000164	254817.00	5.943550
4	560000	20.00	0.000433	240510.00	6.015034
4	570000	7.00	0.000178	260619.00	5.542112
4	580000	7.00	0.000177	230100.00	4.817907
4	590000	11.00	0.000239	240188.00	5.613709
4	600000	8.00	0.000202	260924.00	5.703650
4	610000	7.00	0.000179	261019.00	6.318097
4	620000	13.00	0.000295	281111.00	5.626782
4	630000	9.00	0.000227	276200.00	5.939048
4	640000	10.00	0.000227	286193.00	7.329997
4	650000	10.00	0.000213	255659.00	5.345079
4	660000	6.00	0.000165	321451.00	8.061329
4	670000	7.00	0.000170	243797.00	5.014862
4	680000	8.00	0.000173	286228.00	6.904159
4	690000	19.00	0.000396	306680.00	8.027268
4	700000	4.00	0.000134	330990.00	7.337250

4.7 Conclusions

We have fine-tuned the existing decomposition theorem originally proposed by Kao et al. in [70], in the context of maximum weight bipartite matching and applied it to design a revised version of the decomposition algorithm to compute the weight of a maximum weight bipartite matching in $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time by employing an algorithm designed by Feder and Motwani [45], as base algorithm. We have also analyzed the algorithm by using Hopcroft-Karp algorithm [59] and Alt-Blum-Mehlhorn-Paul algorithm [4] as base algorithms, respectively.

The algorithm performs well especially when the largest edge weight differs by more than one from the second largest edge weight in the current working graph during an invocation of Wt-Mwbm( ) in any iteration. Further, we have given a scaling property of the algorithm and a bound of the parameter W' as $|E| \leq W' \leq \frac{W}{GCD(w_1,w_2,\ldots,w_{|E|})} \leq W$, where $GCD(w_1, w_2, \ldots, w_{|E|})$ denotes the GCD of the positive edges weights $\{w_1, w_2, \ldots, w_{|E|}\}$ of the weighted bipartite graph. The algorithm works well for general W, but is the best known for $W' = o(|E| \log(|V|N))$. The experimental study shows that performance of the modified decomposition algorithm is satisfactory.

Chapter 5

All Pairs Approximate Parameterized String Matching

This chapter deals with all pairs approximate parameterized string matching problem with error threshold $k \in \mathbb{N}_0$, among two sets of equal length strings. An application of this problem can be in finding source code duplication. Given two source codes P and T, each one is combination of several segments of codes. The goal of the problem is for each of the code segments of P, find a code segment of T which is approximate parameterized closest to that segment of P.

Assumptions. Since in this present chapter the APAPSM problem is considered under HD, the following problems are deliberated with HD and for equal length strings. In this chapter we assume that $d = d_H$ which is a distance function for HD, for notational simplicity.

5.1 Introduction

Let us first recall the problem of APSM without error threshold for a pair of equal length strings as mentioned in Problem 2.11 (page 16). The *APSM problem* (without error threshold) between $u \in \Sigma_u^m$ and $v \in \Sigma_u^m$ is to find a π (which may not be unique always) over all bijections from Σ_u to Σ_v such that π-mismatch(u, v) is minimum. Formally,

$$APSMe(u, v) = \{\pi \mid d(\pi(u), v) \text{ is minimum over all } \pi\colon \Sigma_u \to \Sigma_v\}.$$

We have also defined *cost of* $APSMe(u, v)$ as $cost(APSMe(u, v)) = d(\pi(u), v)$ where $\pi \in APSMe(u, v)$. Let us define the APSM problem under k error threshold, as follows.

Problem 5.1 (APSM with k Error Threshold between a Pair of Equal Length Strings)**.** *APSM problem with k error threshold between $u \in \Sigma_u^m$ and $v \in \Sigma_u^m$ seeks to find a bijection π over all bijections from Σ_u to Σ_v such that $d(\pi(u), v)$ is minimum but not greater than k. We denote the collection of such bijections by $APSMe(u, v, k)$. More formally,*

$$APSMe(u, v, k) = \{\pi \mid \pi \in APSMe(u, v) \quad and \quad d(\pi(u), v) \leq k\}.$$

We define the *cost of* $APSMe(u, v, k)$ as $cost(APSMe(u, v, k)) = d(\pi(u), v)$, where $\pi \in APSMe(u, v, k)$. We consider $cost(APSMe(u, v, k)) = \infty$, if $APSMe(u, v, k) = \emptyset$. See the Example 5.3 (on page 63) which shows the difference between the Problems 2.11 and 5.1.

In this chapter we investigate *All Pairs Approximate Parameterized String Matching* (APAPSM) problem with error threshold k (under Hamming distance) among two sets of equal length strings. Let $P = \{p_1,\ p_2, \ldots, p_{n_P}\} \subseteq \Sigma_P^m$ and $T = \{t_1,\ t_2, \ldots, t_{n_T}\} \subseteq \Sigma_T^m$ be two sets of strings where $|\Sigma_P| = |\Sigma_T|$. For each $p_i \in P$, the problem is to find a $t_j \in T$ which is approximately parameterized closest to p_i under the given threshold. We give an $O(n_P\, n_T\, m)$-time solution of this problem. Further, we introduce Parikh vector filtering technique in order to preprocess the given sets of strings to avoid the comparison of non-candidate pairs. The PV-filtering does not change the asymptotic time complexity but the filter is effective for small error threshold as shown by the experiments.

Roadmap. We define the APAPSM problem and discuss a solution of it with worst case complexity $O(n_P\, n_T\, m)$, assuming a constant size alphabet, in Section 5.2. In Section 5.3, we design a filtering technique by using Parikh vector [92] in order to preprocess the given strings and reduce the number of pair comparisons for solving APSM between the strings in a non-candidate pair, with k error threshold. An algorithm using the filter for solving the APAPSM problem is given in Section 5.4. Even though the filter does not improve the asymptotic bound theoretically, but practical results in Section 5.5 show that it performs well for small error threshold. Finally, Section 5.6 summarizes the results.

5.2 APAPSM Problem

In this section we formally define and investigate APAPSM problem with k error threshold (under Hamming distance error model) among the two sets P

and T of equal length strings. The problem statement is the following along with the other required problem[1].

Problem 5.2 (Pair Approximate Parameterized String Matching (PAPSM) with k Error Threshold)**.** *Given a string $p \in \Sigma_P^m$ and $T = \{t_1, t_2, \ldots, t_{n_T}\} \subseteq \Sigma_T^m$, where $|\Sigma_P| = |\Sigma_T| = \sigma$ and $0 \leq k \leq m$. The PAPSM problem with k error threshold is to find j such that $APSMe(p, t_j, k)$ gives π_j over all bijections and $d(\pi_j(p), t_j)$ is minimum over all j where $1 \leq j \leq n_T$.*

Let us denote this problem by $PAPSM(p, T, k)$. Formally,

$$PAPSM(p, T, k) = \{j \mid \pi_j \in APSMe(p, t_j, k) \quad \text{and} \quad d(\pi_j(p), t_j) = \min_{1 \leq i \leq n_T} \{cost(APSMe(p, t_i, k))\}\}.$$

In other words, the problem is to find $t_j \in T$ which is approximately parameterized closest to p with k error threshold. We call $d(\pi_j(p), t_j)$ as the *cost of* $PAPSM(p, T, k)$ and let us denote this by $cost(PAPSM(p, T, k))$. In case $PAPSM(p, T, k) = \emptyset$, then $cost(PAPSM(p, T, k)) = \infty$.

Example 5.3. Given $p = abab \in \Sigma_P^4 = \{a, b\}^4$, $T = \{t_1 = cdcd,\ t_2 = dcdc,\ t_3 = ccdd,\ t_4 = cccd\} \subseteq \Sigma_T^4 = \{c, d\}^4$ and $k = 1$. Now,

$$\begin{aligned}
&APSMe(p, t_1, k) = \{\pi_1 = \{a \to c, b \to d\}\}, \quad d(\pi_1(p), t_1)) = 0;\\
&APSMe(p, t_2, k) = \{\pi_2 = \{a \to d, b \to c\}\}, \quad d(\pi_2(p), t_2)) = 0;\\
&APSMe(p, t_3, k) = \emptyset;\\
&APSMe(p, t_4, k) = \{\pi_4 = \{a \to c, b \to d\}\}, \quad d(\pi_4(p), t_4)) = 1.
\end{aligned}$$

Observe that, $\pi_3 = \{a \to c,\ b \to d\} \in APSMe(p, t_3)$ but $d(\pi_3(p), t_3)) = 2 > k$. So, $APSMe(p, t_3, 1) = \emptyset$. Hence, $PAPSM(p, T, 1) = \{1, 2\}$.

Also note that, if $k = 3$, then for $\pi_4' = \{a \to d, b \to c\}$, $d(\pi_4'(p), t_4) \leq k$. Hence just finding π_4' is also satisfactory to say that p parameterized matches t_4 under $k = 3$ error threshold (see Problem 2.13, page 17), that is, $PSMe(p, t_4, 3) = \{\pi_4, \pi_4'\}$; whereas $APSMe(p, t_4, 3) = \{\pi_4\}$ and $\pi_4' \notin APSMe(p, t_4, 3)$. □

Note that for the PAPSM problem it is sufficient to report a string from T which is closest to p under a given error threshold k. Also, it is possible to enumerate all $t_i \in T$ which are closest to p. Observe that, if $PAPSM(p, T, k) = \{i, j\}$ corresponding to the strings t_i and t_j in T, then $cost(PAPSM(p, T, k)) \leq k$ and more importantly, $cost(PAPSM(p, T, k)) = d(\pi_i(p), t_i) = d(\pi_j(p), t_j)$.

[1]These problems can also be extended with respect to other error models.

Theorem 5.4. *Given $p \in \Sigma_P^m$ and $T = \{t_1, t_2\} \subseteq \Sigma_T^m$. If p is an approximate parameterized matched with t_1 (without any error threshold) and $t_1 \mathrel{\widehat{=}} t_2$, then p is also approximate parameterized matched with t_2 and its cost equal to cost$(APSMe(p, t_1))$.*

Proof. The proof consists of two phases. Since, p is approximate parameterized matched with t_1 (without any error threshold), then say $\pi_1 \in APSMe(p, t_1)$. As a consequence, $cost(APSMe(p, t_1)) = d(\pi_1(p), t_1)$ and moreover it is minimum over all bijections from Σ_P to Σ_T. Also, since $t_1 \mathrel{\widehat{=}} t_2$, there exist a bijection, say π: $\Sigma_T \to \Sigma_T$ such that $\pi(t_1) = t_2$ and so $cost(APSMe(t_1, t_2)) = d(\pi(t_1), t_2) = 0$.

Let $\pi_2 = \pi \circ \pi_1$: $\Sigma_P \to \Sigma_T$ and is defined as $\pi \circ \pi_1(u) = \pi(\pi_1(u))$ where $u \in \Sigma_P^m$. It can be easily proved by contradiction that $d(\pi_2(p), t_2)$ is minimum over all bijections. So we skip it.

Now, $cost(APSMe(p, t_2)) = d(\pi_2(p), t_2) = d(\pi(\pi_1(p)), t_2) = d(\pi(\pi_1(p)), \pi(t_1))$ $= d(\pi_1(p), t_1) = cost(APSMe(p, t_1))$. Therefore, $\pi_2 = \pi \circ \pi_1 \in APSMe(p, t_2)$ and its cost equal to $cost(APSMe(p, t_1))$ unit. Hence the proof. □

The above theorem is extended for APSM problem with k error threshold.

Corollary 5.5. *Given $p \in \Sigma_P^m$ and $T = \{t_1, t_2\} \subseteq \Sigma_T^m$ and $0 \leq k \leq m$. If p is an approximate parameterized matched with t_1 under the k error threshold and $t_1 \mathrel{\widehat{=}} t_2$, then p is also approximate parameterized matched with t_2 under the k error threshold and with cost equal to cost$(APSMe(p, t_1, k))$.*

Problem 5.6 (All Pairs Approximate Parameterized String Matching (APAPSM) with k Threshold)**.** *Let $P = \{p_1,\ p_2, \ldots, p_{n_P}\} \subseteq \Sigma_P^m$ and $T = \{t_1,\ t_2, \ldots, t_{n_T}\} \subseteq \Sigma_T^m$. The problem is to find a mapping η: $[1, n_P] \to [1, n_T]$ such that sum of the cost$(APSMe(p_i, t_{\eta(i)}, k))$ over all i $(1 \leq i \leq n_P)$ is minimum.*

Let us denote this problem as $APAPSM(P, T, k)$. The problem is to search for each $p_i \in P$ $(1 \leq i \leq n_P)$, a $t_j \in T$ $(1 \leq j \leq n_T)$ which is approximately parameterized closest to p_i under k error threshold. More formally,

$$APAPSM(P, T, k) = \{(PAPSM(p_1, T, k), PAPSM(p_2, T, k), \ldots \\ \ldots, PAPSM(p_{n_P}, T, k))\}.$$

Theorem 5.7. *The above problem can be solved in $O(n_P\, n_T\, m^{1.5})$ time.*

Proof. As mentioned in Section 2.7 that APSM problem for a pair of equal length strings can be solved in $O(m^{1.5})$ time [57]. So by considering all possible pairs between the strings of P and T, an $O(n_P\, n_T\, m^{1.5})$ time is required to solve the APAPSM problem. □

5.3 Parikh Vector Based Filter for APAPSM

In general, a *filter* is a device or subroutine that processes the feasible inputs and tries to remove some undesirable component. We design a filtering technique by using Parikh vector in order to preprocess the given strings in P and T and to reduce the number of comparisons for solving APSM between the strings in the non-candidate pairs under k error threshold. We name the filter as *PV-filter* and the process of filtering the input data by PV-filter as *PV-filtering*.

Definition 5.8 ($\gamma(k)$-match of Strings)**.** *Let $k \in \mathbb{N}_0$. For two given strings $u = u[1..m]$, $v = v[1..m] \in \Sigma^*$ and the alphabet set $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ where each $a_i \in \mathbb{N}_0$, u is said to be $\gamma(k)$-matched with v if and only if $\sum_{i=1}^{m} |u_i - v_i| \leq k$.*

The term *$\gamma(k)$-match* is a suitably renamed version of the terminology *γ-approximate* which was prescribed in [25] and defined on strings. Similar as above, we define $\gamma(k)$-match on two equal cardinality vectors of numbers.

Definition 5.9 (γ-distance, $\gamma(k)$-match of Vectors)**.** *Given two vectors $u = (u_1, u_2, \ldots, u_m)$, $v = (v_1, v_2, \ldots, v_m)$ where $u_i, v_j \in \mathbb{N}_0, 1 \leq i, j \leq m$ and $l, k \in \mathbb{N}_0$, γ-distance between u and v is l (denoted as $\gamma(u, v) = l$) if and only if $l = \sum_{i=1}^{m} |u_i - v_i|$. We say that the string u $\gamma(k)$-matches with v if and only if $\gamma(u, v) = \sum_{i=1}^{m} |u_i - v_i| \leq k$.*

The notion of Parikh mapping or vector was introduced by R. J. Parikh in [92]. It provides numerical properties of a string in terms of vector by counting the number of occurrences of the symbols in the string. Parikh vector of a string w is denoted as $\psi(w)$.

Definition 5.10 (Parikh Vector (PV))**.** *Let $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$. Given $w \in \Sigma^*$, $\psi(w) = (f(a_1, w), f(a_2, w), \ldots, f(a_\sigma, w))$ where $f(a_i, w)$ gives the frequency of the symbol $a_i \in \Sigma$ $(1 \leq i \leq \sigma)$ in the string w.*

For example, if $\Sigma = \{c, d\}$ then $\psi(cddcc) = (3, 2)$. However, much information is lost in the transition from a string to its PV. Note that Parikh mapping is not injective as many strings over an alphabet may have the same PV and so the information of a string is reduced while changing the string into a PV. For example, the strings *ccccdddd* and *dcdcdcd* have the same Parikh vector $(3, 4)$.

Definition 5.11 (Normalized Parikh Vector (NPV))**.** *NPV of a string $w \in \Sigma^*$ is denoted by $\widehat{\psi}(w)$ and defined by $\widehat{\psi}(w) = (g_1, g_2, \ldots, g_\sigma)$ such that $\forall i,\ 1 \leq i < \sigma,\ g_i \geq g_{i+1}$ and there exists a bijective mapping ρ: $\{1..\sigma\} \rightarrow \{1..\sigma\}$ such that $g_i = f(a_{\rho(i)}, w)$.*

In other words, we sort the elements of $\psi(w)$ in non-increasing order to get the $\widehat{\psi}(w)$ of the string w. For example, $\psi(dcdcdcd) = (3, 4)$ and $\widehat{\psi}(dcdcdcd) = (4, 3)$.

Theorem 5.12. *Given a pair of equal length strings $u \in \Sigma_P^*$ and $v \in \Sigma_T^*$, if $u \mathrel{\widehat{=}} v$ then $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = 0$.*

Proof. Since $u \mathrel{\widehat{=}} v$, then there exists a bijection $\pi\colon \Sigma_P \to \Sigma_T$ such that $\pi(u) = v$, that is, $\pi(u)$ is obtained by renaming each character of u using π (see Problem 2.3, page 11). Though symbols of Σ_P are renamed by π, frequency of each symbol $a \in \Sigma_P$ in u will be same as the frequency of $\pi(a) \in \Sigma_T$ in $v = \pi(u)$. As a consequence, $\widehat{\psi}(v) = \widehat{\psi}(u)$, even though there may be the case $\psi(u) \neq \psi(v)$. □

However, the converse of the Theorem 5.12 is not always true. Example 5.13 exhibits the same.

Example 5.13. Given $p = ababa, \in \Sigma_P^* = \{a, b\}^*$ and $T = \{t_1 = cdcdd,\ t_2 = dcdcd\} \subseteq \Sigma_T^* = \{c, d\}^*$. Now,

$$\begin{aligned} \psi(p) &= (3, 2) \quad \text{and} \quad \widehat{\psi}(p) = (3, 2); \\ \psi(t_1) &= (2, 3) \quad \text{and} \quad \widehat{\psi}(t_1) = (3, 2); \\ \psi(t_2) &= (2, 3) \quad \text{and} \quad \widehat{\psi}(t_2) = (3, 2). \end{aligned}$$

As mentioned in Theorem 5.12, $p \mathrel{\widehat{=}} t_2$ and so $\gamma(\widehat{\psi}(p), \widehat{\psi}(t_2)) = 0$, even though $\psi(p) \neq \psi(t_2)$. Conversely, $\widehat{\psi}(t_1) = \widehat{\psi}(p) = \widehat{\psi}(t_2) = (3, 2)$, but $p \mathrel{\widehat{\neq}} t_1$ and $p \mathrel{\widehat{=}} t_2$. Hence, for a pair of equal length strings $u \in \Sigma_P^*$ and $v \in \Sigma_T^*$, if $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = 0$ then it is required to check if $u \mathrel{\widehat{=}} v$ or not. □

The following theorems are useful in minimizing the number of pairs comparisons for APAPSM problem to improve the solution from practical aspect. The Theorem 5.14 is applicable for ASM problem. It is extended in Theorems 5.15 and 5.16 in the context of APSM problem without and with k error threshold, respectively.

Theorem 5.14. *Let $u, v \in \Sigma^*$ be a pair of equal length strings and $k = d(u, v)$, is the Hamming distance. Then*

$$\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) \leq 2k \quad \textit{and} \quad \gamma(\psi(u), \psi(v)) \leq 2k.$$

Proof. We prove it by the principle of mathematical induction on k.

Base Case: For $u = v$, $k = d(u, v) = 0$ and $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = 0$.

Induction Hypothesis: Assume that for any k with $0 \leq k = d(u,v) \leq i$, $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) \leq 2k$.

Inductive Step: Let, after introducing one more error by replacement (symbol $a \in \Sigma$ is replaced by $b \in \Sigma$ in any position of u) operation in u we get u' such that $d(u',u) = 1$ and $k = d(u',v) = i+1$. However, while changing u to u' with $d(u',u) = 1$, there may be only other case that $k = d(u',v) = i-1$ for which also the inequality is true (by the induction hypothesis). So we have to argue only for the former case: $k = i+1$. While introducing one error by replacement, $\gamma(\widehat{\psi}(u'), \widehat{\psi}(u))$ will be increased by at most 2 as the frequency of symbol a is decreased by one and the frequency of b is increased by one. Hence, $\gamma(\widehat{\psi}(u'), \widehat{\psi}(v)) \leq \gamma(\widehat{\psi}(u'), \widehat{\psi}(u)) + \gamma(\widehat{\psi}(u), \widehat{\psi}(v)) \leq 2 + 2i = 2(i+1)$ while $k = d(u',v) = i+1$.

Hence the proof of the first inequality, by principle of mathematical induction. For the other inequality also, the proof justification is similar. □

Theorem 5.15. *Given a pair of equal length strings $u \in \Sigma_P^*$ and $v \in \Sigma_T^*$, let $k = cost(APSMe(u,v))$. Then $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) \leq 2k$.*

Proof. Let $\pi \in APSMe(u,v)$. Therefore by definition, $k = cost(APSMe(u,v)) = d(\pi(u), v)$ is minimum over all bijections. Let $\pi(u) = u' \in \Sigma_T^*$. Since $u \mathrel{\widehat{=}} u'$ under π, therefore $\widehat{\psi}(u) = \widehat{\psi}(u')$ (by Theorem 5.12). Hence $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = \gamma(\widehat{\psi}(u'), \widehat{\psi}(v))$. By using Theorem 5.14, we have

$$\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = \gamma(\widehat{\psi}(u'), \widehat{\psi}(v)) \leq 2k.$$

□

Theorem 5.16. *Given $u \in \Sigma_P^*$ and $v \in \Sigma_T^*$ where $|u| = |v|$. Let $\widehat{k} = cost(APSMe(u,v,k))$. Then $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) \leq 2\widehat{k}$ (which we call as PV-filter).*

Proof. The proof is very similar to as Theorem 5.15. Let $\pi \in APSMe(u,v,k)$. According to the Problem 5.1, there exists a bijection π: $\Sigma_P \to \Sigma_T$ such that $\widehat{k} = cost(APSMe(u,v,k)) = d(\pi(u), v)$ is minimum but not greater than k. Let $u' = \pi(u) \in \Sigma_T^*$. With similar argument as above, we have $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = \gamma(\widehat{\psi}(u'), \widehat{\psi}(v)) \leq 2\widehat{k}$. □

We use this PV-filter as a subroutine during the design of a simple algorithm to solve the APAPSM problem with error threshold k among two sets P and T of equal length strings. In worst case (that is, none of the pairs are filtered out by PV-filter), it takes $O(n_P\, n_T\, m)$.

5.4 Solving APAPSM under Error Threshold Using PV-Filter

Let $P = \{p_1,\ p_2, \ldots, p_{n_P}\} \subseteq \Sigma_P^m$ and $T = \{t_1,\ t_2, \ldots, t_{n_T}\} \subseteq \Sigma_T^m$ be two sets of strings where $|\Sigma_P| = |\Sigma_T| = \sigma$. In Algorithm 5.1, we compute APAPSM problem with error threshold $k \in \mathbb{N}_0$ among two sets P and T of equal length strings.

Algorithm 5.1 Compute $APAPSM(P, T, k)$ after using the PV-filter.

Input: The sets P, T of equal length strings and an error threshold k.

Output: $APAPSM(P, T, k)$ with respect to Hamming distance error model.

APAPSM(P, T, k)
1: **for** $i \leftarrow 1 : n_P$ **do** compute NPV of p_i.
2: **for** $i \leftarrow 1 : n_T$ **do** compute NPV of t_i.
3: **do** parameterized clustering of P and T, i.e., for any $(p_1, p_2) \in P \times P$ or $T \times T$ of a cluster, $p_1 \mathrel{\widehat{=}} p_2$.
4: **for** each parameterized cluster of P, pick a representative, say p_i
5: **for** each parameterized cluster of T, pick a representative, say t_j
6: **if** $\gamma(\widehat{\psi}(p_i), \widehat{\psi}(t_j)) \leq 2k$ (the *PV-filtering*)
7: **then** compute $APSMe(p_i, t_j, k)$.
8: **return** $APAPSM(P, T, k)$.

In Step 3, clustering is precisely recommended, if in advance it is known that there are many exact and parameterized repetition of strings in P and T. Otherwise, Step 3 can be omitted. To create the equivalence classes in P and T separately, with respect to parameterization, clustering is done based on the converse of Theorem 5.12, that is, in case for any two strings $u, v \in P$ (and T, respectively) if $\gamma(\widehat{\psi}(u), \widehat{\psi}(v)) = 0$, then and only then check for $u \mathrel{\widehat{=}} v$. If $u \mathrel{\widehat{=}} v$ holds, them put u and v into same cluster.

Complexity Analysis

Steps 1–3 of Algorithm 5.1 are the preprocessing steps for computing the $APAPSM(P, T, k)$; Steps 1–2 takes $O(m(n_P+n_T))$ and Step 3 takes $O(m(n_P^2+n_T^2))$ time, assuming a constant size alphabets. So the preprocessing time of the algorithm is $O(m(n_P^2 + n_T^2))$. But as mentioned earlier, clustering is optional, it might be skipped depending on the circumstances.

In Steps 4–7, for any pair $(p_i, t_j) \in P \times T$, computation of $APSMe(p_i, t_j, k)$ can be done by reducing the problem to MWBM problem [57]. Let $G = (V, E, Wt)$ be an undirected, weighted (positive integer weight) bipartite graph where V, E and W are the vertex set, edge set and total weight of G which is equal to $Wt(G) = \sum_{e \in E} Wt(e)$, respectively. As discussed in Section 4.4, the MWBM problem can be solved in $O(\sqrt{|V|}W')$ time, where $|E| \leq W' \leq W/g$, where g denotes the GCD of the positive edges weights $\{w_1, w_2, \ldots, w_{|E|}\}$ of G. It is a fine-tuned version of the existing decomposition solution [70]. Using the fine-tuned decomposition solution for MWBM, $APSMe(p_i, t_j, k)$ can be solved in $O(m\sqrt{\sigma})$ where $W' = O(W/g) = O(m)$ and $V = O(\sigma)$. In the worst case scenario: each of the clusters will have just a single string either from P or T and PV-filter in Step 6 does not filter out any pair $(p_i, t_j) \in P \times T$. Therefore computing time (except preprocessing time) of $APAPSM(P, T, k)$ is $O(n_P\, n_T\, m\sqrt{\sigma})$ in worst case, which is $O(n_P\, n_T\, m)$ if we assume a constant alphabet.

5.5 Experimental Evaluation

To test the efficiency of the PV-filter, we performed several experimental study, but only few are reported in this section. The Algorithm 5.1 which solves $APAPSM(P, T, k)$, is implemented in *MATLAB Version 7.8.0.347 (R2009a)*. All the experiments are conducted on a PC Laptop with an *Intel® Core™ 2 Duo (T6570 @ 2.10GHz) Processor*, *3.00 GB RAM* and *500 GB Hard Disk*, running the *Microsoft Windows 7 Ultimate 32-bit Operating System*.

Input Data Description

We generate the input data sets P and T by using the predefined `randi` function. It helps to generate uniformly distributed pseudorandom integers. The function `randi(imax,m,n)` returns an `m`-by-`n` matrix containing pseudorandom integer values drawn from the discrete uniform distribution on `1:imax`.

Efficiency of the PV-Filter

The experimental results show that the PV-filter is efficient, essentially for small error threshold k, to avoid some of the non-candidate pairs (u, v) for solving the $APSMe(u, v, k)$, where $u \in P$ and $v \in T$. According to the random experiment, very smaller threshold gives much more better filtering

of the non-candidate pairs. See the following experiments on the randomly generated data inputs.

Experiment 5.17. Consider the alphabets $\Sigma_P = \{a, b, c\}$ and $\Sigma_T = \{a', b', c'\}$; $P \in \Sigma_P^{10}$, $T \in \Sigma_T^{10}$; the cardinality of each of the sets P and T is 100, that is, $|P| = |T| = 100$ and the length of each string in P and T is $|p_i| = |t_j| = 10$ for $1 \leq i, j \leq 100$.

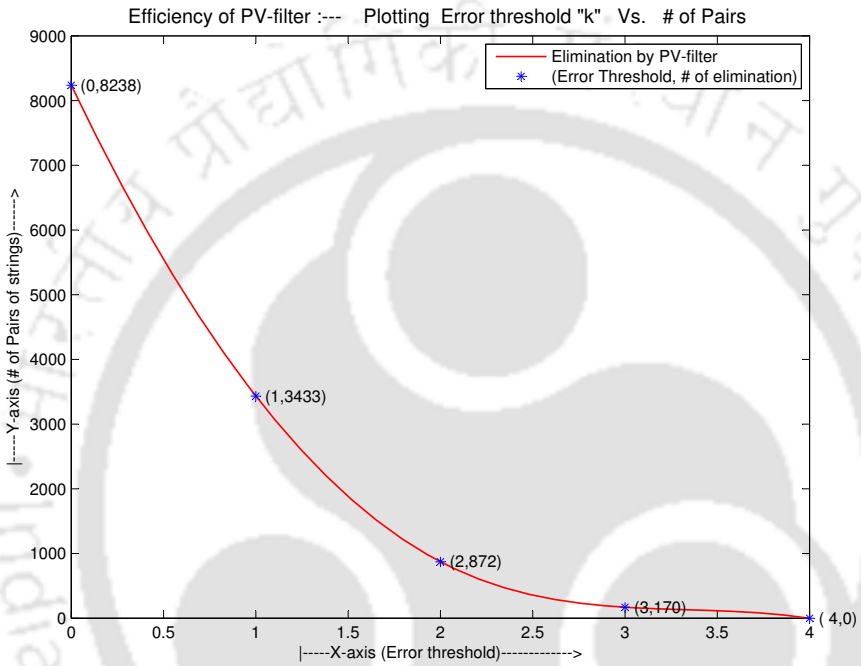

Figure 5.1: Elimination graph of pairs of strings after using the PV-filter for the input data set with $|\Sigma_P| = |\Sigma_T| = 3$, $|P| = |T| = 100$, $|p_i| = |t_j| = 10$ for $1 \leq i, j \leq 100$, as mentioned in Experiment 5.17.

Table 5.1: Experimental results showing efficiency of PV-filter for the data set considered in Experiment 5.17.

Number of Pairs $\downarrow$	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
before using PV-filter	10,000	10,000	10,000	10,000	10,000
eliminated by PV-filter	8,238	3,433	872	170	0
passed through PV-filter	1,762	6,567	9,128	9,830	10,000
whose APSM cost $\leq k$	3	28	208	1,128	4,109

According to the data set generated in Experiment 5.17, a total of 10,000 (u, v) pairs of comparisons for $APSMe(u, v, k)$ are required without PV-filtering, where $u \in P$ and $v \in T$. Figure 5.1 shows the efficiency graph

after using the filter on the input data set. Each blue "$*$" point in the graph indicates the number of elimination of pairs comparison for a given error threshold k, after using the PV-filter. For example, each point (i, j) in the Figure 5.1 represents that for $k = i$ error threshold, j number of $(u, v) \in P \times T$ pairs of strings are skipped the comparison for $APSMe(u, v, i)$.

Table 5.1 gives more light for the Experiment 5.17. Second row represents that for a given k, the total number of (u, v) pairs are to be checked for $APSMe(u, v, k)$, initially before using the PV-filter. The third row says, the number of pairs of strings are eliminated by the PV-filter for different k. Simultaneously, fourth row describes that how many string pairs are passed by the filter. And the final row mentions the number of (u, v) pairs, which are passed by the filter, for which $cost(APSMe(u, v)) \leq k$. □

Experiment 5.18. Consider, alphabet sets $\Sigma_P = \{a, b, c, d, e, f, g, h, i, j\}$ and $\Sigma_T = \{a', b', c', d', e', f', g', h', i', j'\}$; $P \in \Sigma_P^6$, $T \in \Sigma_T^6$; the cardinality of each of the sets P and T is 100, that is, $|P| = |T| = 100$ and $|p_i| = |t_j| = 6$ for $1 \leq i, j \leq |P| = |T|$. Figure 5.2 gives the elimination graph due to the PV-filter and more details are listed in Table 5.2. □

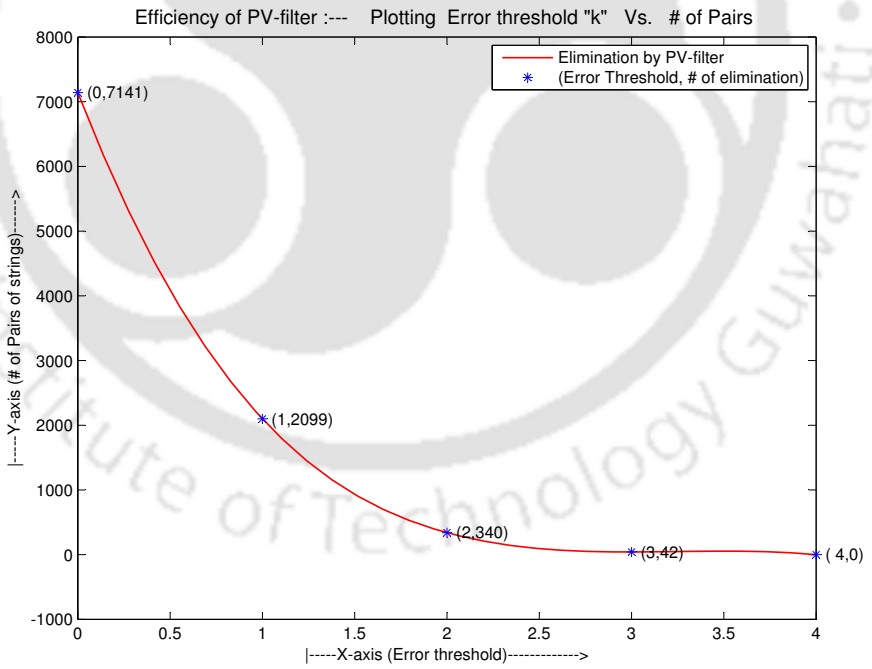

Figure 5.2: Elimination graph of pairs of strings after using the PV-filter for the input data set with $|\Sigma_P| = |\Sigma_T| = 10$, $|P| = |T| = 100$, $|p_i| = |t_j| = 6$ for $1 \leq i, j \leq |P| = |T|$, as considered in Experiment 5.18.

Table 5.2: Experimental results showing efficiency of PV-filter for the data set considered in Experiment 5.18.

Number of Pairs $\downarrow$	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
before using PV-filter	10,000	10,000	10,000	10,000	10,000
eliminated by PV-filter	7,141	2,099	340	42	0
passed through PV-filter	2,859	7,901	9,660	9,958	10,000
whose APSM cost $\leq k$	570	3,986	8,699	9,869	10,000

Experiment 5.19. Consider alphabet sets $\Sigma_P = \{a, b, c, d\}$ and $\Sigma_T = \{a', b', c', d'\}$; $P \in \Sigma_P^{2000}$, $T \in \Sigma_T^{2000}$; the cardinality of each of the sets P and T is 100, that is, $|P| = |T| = 100$ and $|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq 100$. Figure 5.3 gives the elimination graph. Big table is omitted. □

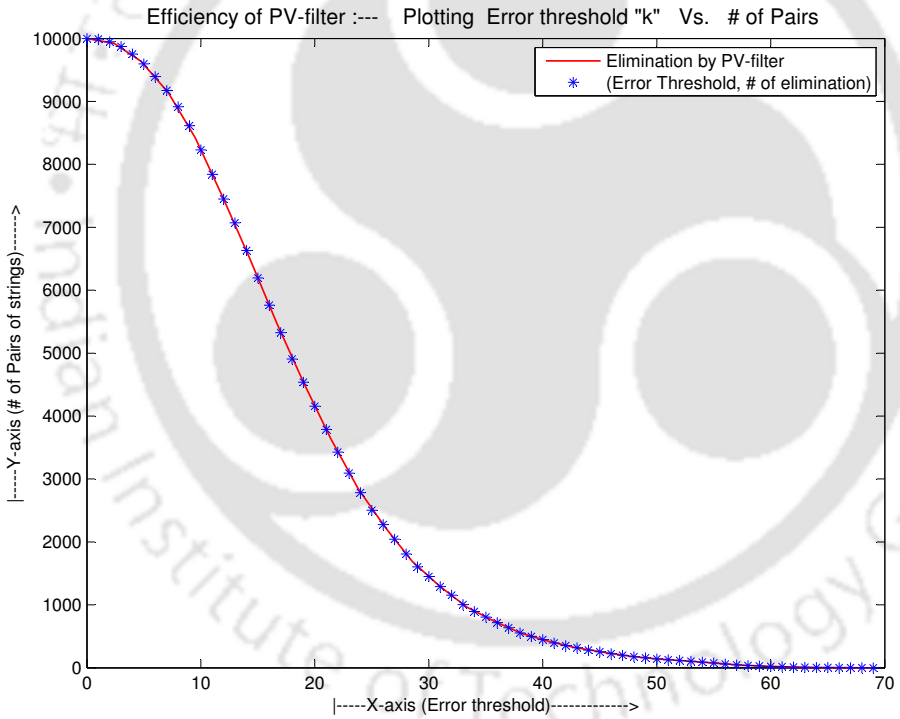

Figure 5.3: Elimination graph of pairs of strings after using the PV-filter for the input data set with $|\Sigma_P| = |\Sigma_T| = 4$, $|P| = |T| = 100$, $|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq |P| = |T|$, as mentioned in Experiment 5.19.

Experiment 5.20. Consider alphabet sets $\Sigma_P = \{a, b, c, d, e, f, g, h, i, j\}$ and $\Sigma_T = \{a', b', c', d', e', f', g', h', i', j'\}$; $P \in \Sigma_P^{2000}$, $T \in \Sigma_T^{2000}$; the cardinality of each of the sets P and T is 100, that is $|P| = |T| = 100$ and

$|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq |P| = |T|$. Figure 5.4 gives the elimination graph. Corresponding big table is omitted. □

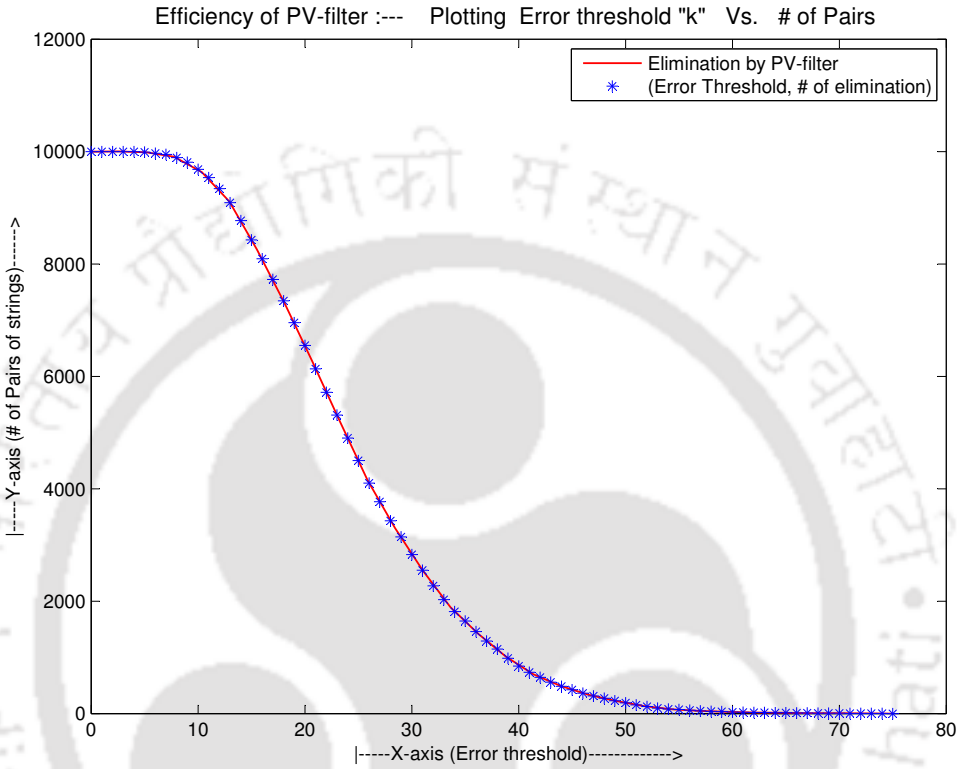

Figure 5.4: Elimination graph of pairs of strings after using the PV-filter for the input data set with $|\Sigma_P| = |\Sigma_T| = 10$, $|P| = |T| = 100$, $|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq |P| = |T|$, as stated in Experiment 5.20.

Experiment 5.21. Consider the alphabet sets $\Sigma_P = \{a, b, c, \ldots, z\}$, $\Sigma_T = \{a', b', c', \ldots, z'\}$ with $|\Sigma_P| = |\Sigma_T| = 26$; $P \in \Sigma_P^{2000}$, $T \in \Sigma_T^{2000}$; the cardinality of each of the sets P and T is 100, that is, $|P| = |T| = 100$ and $|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq |P| = |T|$. Figure 5.5 gives the elimination graph. Corresponding big table is omitted. □

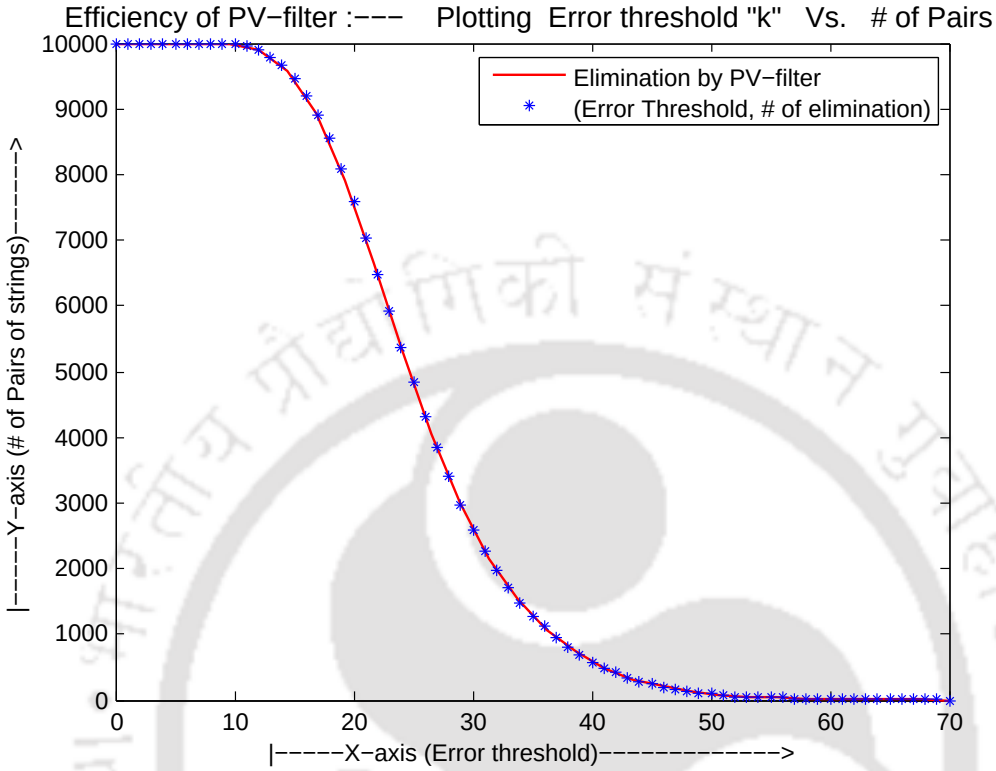

Figure 5.5: Elimination graph of pairs of strings after using PV-filter for the input data set with $|\Sigma_P| = |\Sigma_T| = 26$; $|P| = |T| = 100$; $|p_i| = |t_j| = 2000$ for $1 \leq i, j \leq |P| = |T|$, as considered in Experiment 5.21.

5.6 Conclusions

In this chapter we have explored all pairs approximate parameterized string matching problem with k Hamming distance error threshold between two sets of equal length stings. We have presented a solution with worst case complexity $O(n_P\, n_T\, m)$, assuming constant alphabet size. In order to minimize number of paired comparisons for solving APSM between the strings in the pair of strings with error threshold, we have proposed a PV-filtering technique by using Parikh vector. Although the filter does not improve the worst case asymptotic bound, but using it as a subroutine, we can avoid some of the non-candidate pairs for APAPSM. Experimental results show that the PV-filter is efficient essentially for small error threshold.

Chapter 6

Approximate Parameterized String Matching under Weighted Hamming Distance

This chapter is devoted to the study of the APSM problem (without error threshold) under *Weight Hamming Distance* (WHD) which is a generalized version of Hamming distance. Furthermore, we consider PSM problem with k mismatches under WHD where $k \in \mathbb{N}_0$.

Assumptions. We denote the distance function for HD and WHD by d_H and d_{WH}, respectively. But in this chapter we assume that $d = d_{WH}$ for notational simplicity.

6.1 Introduction

Up to now we have considered the APSM problem under Hamming distance. In Hamming distance, the cost of each replacement operation is one unit irrespective of the characters involved (see Section 2.5). Let us see a generalization of the HD. The WHD allows only replacements but associated with non-negative weights (or costs) where the cost of each operation depends on the characters involved in the corresponding replacement.

In this chapter we study *Weighted Approximate Parameterized String Matching* (WAPSM) problem for a pattern of length m in a text of length n, which considers the APSM problem under WHD error metric. The solution is essentially based on a reduction of the APSM problem between two equal length strings under WHD to the MWBM problem. Given a pattern $p \in \Sigma_P^m$ and a text $t \in \Sigma_T^n$, we present an $O(nm)$-time algorithm for the WAPSM

problem by using the MWBM algorithm proposed in Chapter 4. Furthermore, recall the Problem 2.13 described on page 17, that is, PSM problem between a pair of equal length strings under HD with k mismatches. We investigate the same problem but under WHD. *Weighted Parameterized String Matching* (WPSM) problem between a pair of m-length strings with k mismatches considers PSM problem between a pair of m-length strings under WHD with k mismatches. We propose an $O(m + k)$-time solution for this problem. All the above time complexities presume constant size alphabets.

Roadmap. The main problem of this chapter is discussed in detail in Section 6.2. Here we present a reduction of WAPSM problem to the MWBM problem in graph, an algorithm for WAPSM problem (without error threshold) and analyze its complexity. Section 6.3 talks about the WPSM problem with k threshold. Finally, we summarize the work in Section 6.4.

6.2 Weighted Approximate Parameterized String Matching

Here we discuss the main results of this chapter on WAPSM problem which, in other words, is the APSM problem that allows WHD as a measure of error. The APSM problem [9, 10, 57] is a well studied problem under Hamming distance and this problem for a pair of m-length strings can be solved in $O(m^{1.5})$ time [57] by using the decomposition algorithm for MWBM [70]. The following theorem talks about a better bound of this problem.

Lemma 6.1 (A Better Upper Bound of APSM Problem under HD)**.** *Given a pair of equal length strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$, an $O(m\sqrt{\sigma})$ time is required to solve the APSM problem under HD, where $\sigma = |\Sigma_u| = |\Sigma_v|$.*

Proof. Recall that the APSM Problem 2.11 for u and v under HD and the MWBM problem are computationally equivalent [57]. Let the bipartite graph $G = (V, E, Wt)$ is generated from u and v (see the construction technique in Section 3.3) where W is the total weight of G, that is, $W = Wt(G) = \sum_{e \in E} Wt(e)$. As discussed in Section 4.4, a MWBM of G can be computed in $O(\sqrt{|V|}W')$ time by using the modified decomposition theorem (Theorem 4.4), where $|E| \leq W' \leq W/g \leq W$, g denotes the GCD of the positive edges weights of G. The advantage of using the MWBM algorithm is that it considers the whole strings at once and efficiently finds the best mapping over all bijections from Σ_u to Σ_v. Using this fine-tuned decomposition solution for the MWBM problem, $APSMe(u, v)$ can be solved in $O(m\sqrt{\sigma})$ because $W' = O(W/g) = O(W) = O(m)$ and $V = O(\sigma)$. □

Let us now focus on the WAPSM problem. According to the definition of the Hamming distance, cost of each replacement operation is one unit irrespective of the characters involved.

Definition 6.2. *Weighted Hamming Distance (WHD) allows only replacement operations but associated with non-negative weights (or costs) where the cost of each operation depends on the characters involved in the corresponding replacement.*

The WHD function is denoted by d_{WH}. Formally, the WHD between two equal length strings $u = u[1..m]$ and $v = v[1..m] \in \Sigma^*$ is a function d_{WH}: $\Sigma^* \times \Sigma^* \rightarrow \mathbb{N}_0$ and is defined as

$$d_{WH}(u, v) = \sum_{i=1}^{m} d_{WH}(u_i, v_i) = \sum_{i=1}^{m} d_H(u_i, v_i) * d_{ij},$$

where $D(\Sigma) = (d_{ij})_{|\Sigma| \times |\Sigma|}$ is a non-negative integer *cost* (or *weight*) *matrix* based on the problem under consideration. The cost matrix may not be symmetric. As mentioned in the beginning of this chapter, from now onwards we assume that $d = d_{WH}$, for notational simplicity. Let us see an example to illustrate the difference between Hamming distance and weighted Hamming distance.

Example 6.3. Let us consider the DNA alphabet $\Sigma = \{A, C, G, T\}$ and a corresponding weight or cost matrix $D(\Sigma)$ is as follows.

$D(\Sigma) =$

	A	C	G	T
A	0	1	2	1
C	1	0	1	2
G	2	1	0	1
T	1	2	1	0

Consider the strings belong to Σ^*: $u_1 = ATTAAA$,
$u_2 = ATTCGG$ and
$u_3 = ATTCCG$;

Now, by the definitions of HD and WHD,

$d_H(u_1, u_2)$
$= d_H(A, A) + d_H(T, T) + d_H(T, T) + d_H(A, C) + d_H(A, G) + d_H(A, G)$
$= 3$, but

$d(u_1, u_2)$
$= d(A, A) + d(T, T) + d(T, T) + d(A, C) + d(A, G) + d(A, G)$
$= 5$.

On the contrary,

$d_H(u_1, u_3)$
$= d_H(A, A) + d_H(T, T) + d_H(T, T) + d_H(A, C) + d_H(A, C) + d_H(A, G)$
$= 3$, but

$d(u_1, u_3)$
$= d(A, A) + d(T, T) + d(T, T) + d(A, C) + d(A, C) + d(A, G)$
$= 4$.

Observe that, as calculated above, under Hamming distance, both u_2 and u_3 are equidistant from u_1; whereas under weighted Hamming distance, u_1 is closer to u_3 than it is to u_2. □

6.2.1 APSM Problem under WHD

Given a pair of equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$, a cost matrix $D(\Sigma_v)$ and a bijection π: $\Sigma_u \to \Sigma_v$, the *weighted π-mismatch* between u and v is the WHD between the image of u under π and v, that is, $d(\pi(u), v)$. Denote it as π-*wmismatch*(u, v). Therefore,

$$\pi\text{-}wmismatch(u, v) = d(\pi(u), v).$$

Problem 6.4 (WAPSM Problem between u and v)**.** *Weighted approximate parameterized string matching between u and v is to find a π: $\Sigma_u \to \Sigma_v$ over all bijections such that π-wmismatch(u, v) is minimum.*

We denote this collection by $WAPSMe(u, v)$. Formally,

$WAPSMe(u, v) = \{\pi \mid d(\pi(u), v)$ is minimum over all bijections π: $\Sigma_u \to \Sigma_v\}$.

We define the *cost of* $WAPSMe(u, v)$ as $cost(WAPSMe(u, v)) = d(\pi(u), v)$ where $\pi \in WAPSMe(u, v)$. The following example clarifies the above definition and differentiates it from the APSM problem under HD.

Example 6.5. Let $\Sigma_u = \{a, b\}$, $\Sigma_v = \{a', b'\}$ and the cost matrix corresponding to Σ_v is given as follows.

$D(\Sigma_v)$ =

	a'	b'
a'	0	1
b'	3	0

The two possible bijections from Σ_u to Σ_v are $\pi_1 = \{a \to a', b \to b'\}$ and $\pi_2 = \{a \to b', b \to a'\}$. Given $u = aaaab \in \Sigma_u^5$ and $v = a'b'b'b'b' \in \Sigma_v^5$, APSM between u and v under HD and WHD, are computed below.

APSM between u and v under HD: By considering the bijective mapping π_1,

$$d_H(\pi_1(u), v) = d_H(a'a'a'a'b', a'b'b'b'b') = 3,$$

whereas under the mapping π_2,

$$d_H(\pi_2(u), v) = d_H(b'b'b'b'a', a'b'b'b'b') = 2.$$

Therefore, as stated in Problem 2.11 (on page 16), we have $APSMe(u, v) = \{\pi_2\}$ and $cost(APSMe(u, v)) = 2$.

APSM between u and v under WHD: By considering the mapping π_1, we have

$$d(\pi_1(u), v) = d(a'a'a'a'b', a'b'b'b'b') = d(a', a') + 3 * d(a', b') + d(b', b') = 3,$$

whereas under the bijection π_2,

$$d(\pi_2(u), v) = d(b'b'b'b'a', a'b'b'b'b') = d(b', a') + 3 * d(b', b') + d(a', b') = 4.$$

Hence, according to the Problem 6.4, we have $WAPSMe(u, v) = \{\pi_1\}$ and $cost(WAPSMe(u, v)) = 3$.

Thus, this example illustrates the difference between APSM under HD and WHD for a pair of equal length strings. □

Problem 6.6 (WAPSM Problem for a Pattern p in a Text t)**.** *Given a pattern $p \in \Sigma_P^m$, a text $t \in \Sigma_T^n$ and a cost matrix $D(\Sigma_T)$, WAPSM problem for p in t is to find a bijection π: $\Sigma_P \to \Sigma_T$ at each location i (where $1 \leq i \leq n-m+1$) of t, for which the π-wmismatch$(p, t[i..i+m-1])$ is minimum.*

Let us denote the set of all such bijections as $WAPSM(p, t)$. Formally, $WAPSM(p, t) = \{(\pi_i)_{i \in \{1,2,\ldots,n-m+1\}} \mid \pi_i \in WAPSMe(p, t[i..i+m-1]) \ \forall i \in \{1, 2, \ldots, n-m+1\}\}$.

6.2.2 Reducing WAPSM Problem to the MWBM Problem

Given two equal length strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$, we reduce the WAPSM problem (that is, APSM problem under WHD) to the MWBM problem. For the former problem, the inputs are:

(a) strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$ with $|\Sigma_u| = |\Sigma_v| = \sigma$,

(b) a cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma \times \sigma}$ where $d_{ij} \in \mathbb{N}_0$, $1 \leq i, j \leq \sigma$.

We first construct a weighted bipartite graph $G = (V = V_1 \cup V_2, E, Wt)$ from the input information given in (a), then build another weighted bipartite (error) graph G' from G based on the cost matrix given in (b), and finally construct a bipartite graph $\widehat{G}$ from G', as follows.

Construction of G: The construction of G is similar as described in Section 3.3. Let $\Sigma_u = \{a_1, a_2, \ldots, a_\sigma\}$, $\Sigma_v = \{b_1, b_2, \ldots, b_\sigma\}$. Take $V_1 = \Sigma_u$ and $V_2 = \Sigma_v$. Add an edge between the vertices $a_i \in V_1$ $(= \Sigma_u)$ and $b_j \in V_2$ $(= \Sigma_v)$ if and only if a_i in u is aligned with b_j in v at least one location of the strings, and assign a weight to the edge $\{a_i, b_j\}$ as the frequency of total such alignment of a_i in u with b_j in v. Let $F = (f_{ij})_{\sigma\times\sigma}$ is the *frequency matrix* where f_{ij}, the ij-th entry of F, is the total number of alignment of a_i in u with b_j in v.

Construction of G' from G and $D(\Sigma_v)$: The graph $G' = (V, E', Wt)$ is formed by including every edge $e_{ij} = \{a_i, b_j\}$ of G (where $i, j \in \{1, 2, \ldots, \sigma\}$), whose weight satisfies the condition:

$$\sum_{\substack{j'=1 \\ j'\neq j}}^{\sigma} f_{ij'}\, d(b_j, b_{j'}) > 0.$$

The weight assigned to such an edge e_{ij} is

$$Wt(e_{ij}) \;= \sum_{\substack{j'=1 \\ j'\neq j}}^{\sigma} f_{ij'}\, d(b_j, b_{j'}) \;= \sum_{\substack{j'=1 \\ j'\neq j}}^{\sigma} f_{ij'}\, d_{jj'}.$$

At the end, if the graph G' is not of the form of $K_{\sigma,\sigma}$, a complete bipartite graph, then we insert the missing edge and assign them weight 0.

We call G' as an *error graph*, because it is constructed by incorporating the cost matrix $D(\Sigma_v)$ in G.

Construction of $\widehat{G}$ from G': Let

$$c_{G'max} = \max\{\, Wt(e_{ij}) \mid e_{ij} \in E' \;\text{ where }\; i, j \in \{1, 2, \ldots, \sigma\}\}.$$

The graph $\widehat{G}$ is constructed by including every edge $e_{ij} = \{a_i, b_j\}$ of G' (where $i, j \in \{1, 2, \ldots, \sigma\}$), whose weight satisfies the condition

$$c_{G'max} - Wt(e_{ij}) > 0.$$

Assign weight to such an edge e_{ij} as

$$c_{G'max} - Wt(e_{ij}) = c_{G'max} - \sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d(b_j, b_{j'}) = c_{G'max} - \sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d_{jj'}.$$

To get the pictorial view of the above construction, refer to the Examples 6.8 and 6.9 (on pages 82 and 83 respectively).

Finally, we use a maximum weight matching of the bipartite graph $\widehat{G}$ to compute a minimum π-*wmismatch*(u, v) for the WAPSM problem between u and v, where π is defined by the edges of the matching. The following theorem shows the correctness of the construction.

Theorem 6.7 (Correctness of the above Construction)**.** *Given a pair of equal length strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$ with $\sigma = |\Sigma_u| = |\Sigma_v|$, and a cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma \times \sigma}$, computing a MWBM in $\widehat{G}$ is equivalent to solving the WAPSM problem between u and v.*

Proof. Let $\Sigma_u = \{a_1, a_2, \ldots, a_\sigma\}$, $\Sigma_v = \{b_1, b_2, \ldots, b_\sigma\}$ and $F = (f_{ij})_{\sigma \times \sigma}$ be the frequency matrix where f_{ij} is the number of times an alphabet a_i in u is aligned to the alphabet b_j in v. Accordingly, the corresponding weighted bipartite graph $G = (V_1 \cup V_2, E, Wt)$ is constructed from u and v by choosing $V_1 = \Sigma_u$ and $V_2 = \Sigma_v$. We claim that finding a maximum weight matching in G corresponds to finding the minimum π-*wmismatch*(u, v), where π: $V_1 \to V_2$ is defined by the edges of that matching.

Recall that a matching of a graph is a subset of the graph edges which does not contain any of its adjacent edges. In APSM between two strings u and v, only these adjacent edges contribute to total error. In fact, if the edges of a (bijective) matching in a bipartite graph is given by π: $V_1 \to V_2$, then we call this error as:

(i) π-*mismatch*(u, v) for the case of APSM problem between u and v under HD error model, and

(ii) π-*wmismatch*(u, v) for the case of APSM problem between u and v under WHD error model.

Accordingly, we construct a weighted error bipartite graph G' form G and the given cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma \times \sigma}$ as follows. For each vertex $a_i \in V_1$, if we consider an edge[1] $e_{ij} = \{a_i, b_j\}$ of G (where $j \in \{1, 2, \ldots, \sigma\}$) to be a part

[1]It may be a zero weight edge of G, in order to maintain the bijection property.

of its matching edges then all the adjacent edges[2] $e_{ij'} = \{a_i, b_{j'}\}$ of G (where $j' \in \{1, 2, \ldots, \sigma\}$ but $j' \neq j$) will not be a part of that matching. As a result, while considering the mismatches in APSM under HD, these adjacent edges will contribute an error of $\sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}$ unit in G' for the matching edge e_{ij} of G. Moreover, while considering the weighted cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma \times \sigma}$ (that is, under the WHD), the above mismatches lead to incorporate a weighted error of

$$\sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d(b_j, b_{j'}) = \sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d_{jj'}$$

unit in the weighted error graph G', corresponding to the matching edge e_{ij} of G. Likewise we complete the construction of the graph G' which is of the form $K_{\sigma,\sigma}$ and assign weight

$$Wt(e_{ij}) = \sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d_{jj'}$$

to every edge e_{ij}, where $i, j \in \{1, 2, \ldots, \sigma\}$. Observe that, in order to find a minimum π-$wmismatch(u, v)$ for the problem of APSM between u and v under WHD error model, it is enough to find a minimum weight perfect matching in the bipartite graph G', where π: $V_1 \to V_2$ is defined by the edges of the matching.

A minimum weight perfect matching in a weighted bipartite graph is equivalent to a maximum weight matching in the weighted bipartite graph obtained by replacing each edge weight $Wt(e_{i,j})$ with $\alpha - Wt(e_{i,j})$, where α is some large fixed positive constant. And hence, computing a maximum weight bipartite matching in $\widehat{G}$ corresponds to solving the WAPSM problem between u and v. □

Let us see an example with the same initial inputs as considered in the Example 6.5 (on page 78) to illustrate the above reduction and to solve WAPSM between a pair u and v. The Theorem 6.7 is further clarified with another example (Example 6.9) for 3×3 cost matrix.

Example 6.8. Let $\Sigma_u = \{a, b\}$, $\Sigma_v = \{a', b'\}$ and the cost matrix $D(\Sigma_v)$ corresponding to Σ_v is $d(a', a') = d(b', b') = 0$, $d(a', b') = 1$, $d(b', a') = 3$ as displayed in Figure 6.1(a).

[2]Note that, $a_i \in V_1$ is the vertex to which all the edges e_{ij} (where $j = 1, 2, \ldots, \sigma$) are incident.

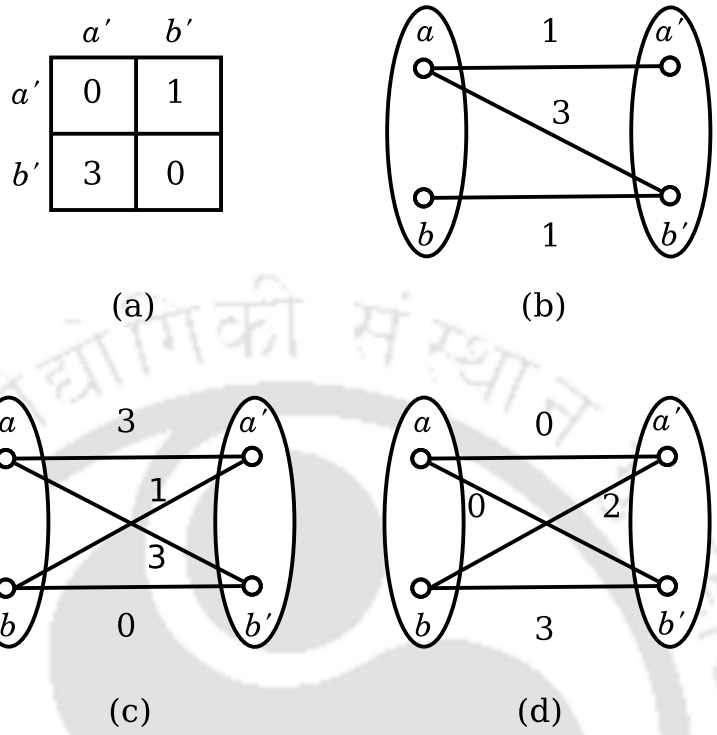

Figure 6.1: **(a)** Distance (or cost or weight) matrix $D(\Sigma_v)$ corresponding to Σ_v. **(b)** G is formed from the given strings $u = aaaab \in \Sigma_u^5$ and $v = a'b'b'b'b' \in \Sigma_v^5$. **(c)** G' is constructed from G and $D(\Sigma_v)$. **(d)** Finally, $\widehat{G}$ is formed after making required changes in G'.

For the pair of given strings $u = aaaab \in \Sigma_u^5$ and $v = a'b'b'b'b' \in \Sigma_v^5$, we construct the corresponding bipartite graph G shown in Figure 6.1(b). Figure 6.1(c) displays the error graph G' which is formed from G and $D(\Sigma_v)$. The weight of the edges of G' are computed as follows.

$$Wt(a, a') = 3 * d(a', b') = 3, \qquad Wt(a, b') = 1 * d(b', a') = 3;$$
$$Wt(b, a') = 1 * d(a', b') = 1, \qquad Wt(b, b') = 0 * d(b', a') = 0.$$

Finally, the bipartite graph $\widehat{G}$, as shown in Figure 6.1(d), is created by replacing each weight $Wt(e_{ij})$ of G' with $c_{G'max} - Wt(e_{ij})$. The edge set of a MWBM in $\widehat{G}$ is $M = \{\{a, a'\}, \{b, b'\}\}$. See that this M is also a minimum weight perfect matching in G'. Hence, $WAPSMe(u, v) = \{a \to a', b \to b'\}$ and $cost(WAPSMe(u, v)) = 3$. Observe that, the Example 6.5 explicitly computes this example and results the same. □

Example 6.9. Let us consider $\Sigma_u = \{a, b, c\}$, $\Sigma_v = \{a', b', c'\}$. And the given cost matrix $D(\Sigma_v)$ corresponding to Σ_v is mentioned in Figure 6.2(a).

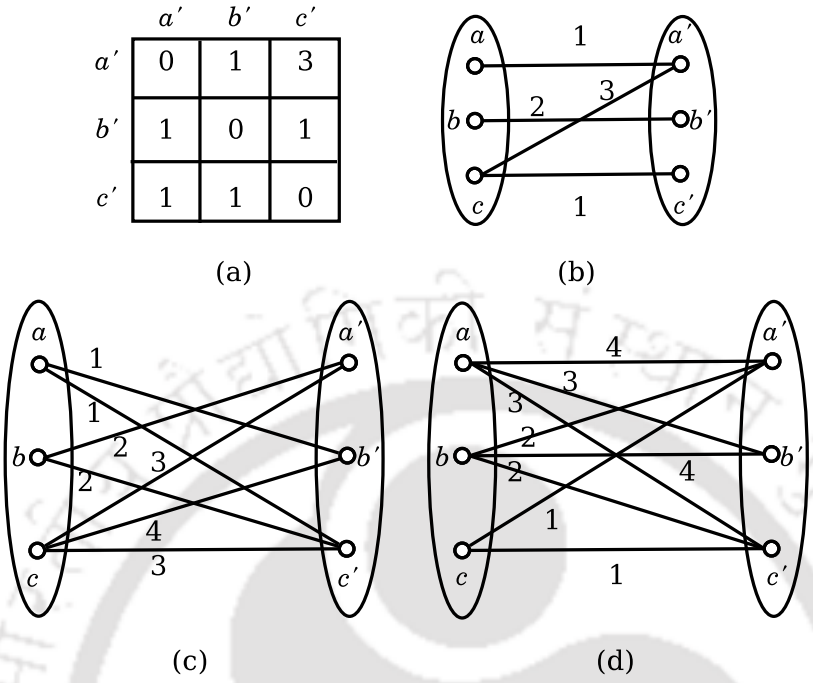

Figure 6.2: **(a)** Distance or cost matrix $D(\Sigma_v)$ corresponding to Σ_v. **(b)** G is formed from the given strings $u = abbcccc \in \Sigma_u^7$ and $v = a'b'b'a'a'a'c' \in \Sigma_v^7$. **(c)** G' is constructed from G and $D(\Sigma_v)$. **(d)** Finally, $\widehat{G}$ is generated after making required changes in G'.

For the pair of given strings $u = abbcccc \in \Sigma_u^7$ and $v = a'b'b'a'a'a'c' \in \Sigma_v^7$ we construct the corresponding bipartite graph G shown in Figure 6.2(b). The error graph G' is displayed in Figure 6.2(c) which is formed from G and $D(\Sigma_v)$. The weights of the edges of G' are calculated below:

$$Wt(a, a') = 0 * d(a', b') + 0 * d(a', c') = 0,$$
$$Wt(a, b') = 1 * d(b', a') + 0 * d(b', c') = 1,$$
$$Wt(a, c') = 1 * d(c', a') + 0 * d(c', b') = 1;$$

$$Wt(b, a') = 2 * d(a', b') + 0 * d(a', c') = 2,$$
$$Wt(b, b') = 0 * d(b', a') + 0 * d(b', c') = 0,$$
$$Wt(b, c') = 0 * d(c', a') + 2 * d(c', b') = 2;$$

$$Wt(c, a') = 0 * d(a', b') + 1 * d(a', c') = 3,$$
$$Wt(c, b') = 3 * d(b', a') + 1 * d(b', c') = 4,$$
$$Wt(c, c') = 3 * d(c', a') + 0 * d(c', b') = 3.$$

Finally, the bipartite graph $\widehat{G}$, shown in Figure 6.2(d), is generated by replacing each weight $Wt(e_{ij})$ of G' with $c_{G'max} - Wt(e_{ij})$. The edge set of a MWBM in $\widehat{G}$ is $M = \{\{a, a'\}, \{b, b'\}, \{c, c'\}\}$. Notice that this M is also a minimum weight perfect matching in G'. Hence,

$$WAPSMe(u, v) = \{\pi = \{a \to a', b \to b', c \to c'\}\} \quad \text{and}$$

$$\begin{aligned} &cost(\, WAPSMe(u, v)) \\ &\quad = d(\pi(u), v) \\ &\quad = d(a', a') + d(b', b') + d(b', b') + d(c', a') + d(c', a') + d(c', a') + d(c', c') \\ &\quad = 3. \end{aligned}$$

Observe that, this value is also the weight of a minimum weight perfect matching M in the graph G'. □

6.2.3 Algorithm for Computing $WAPSM(p, t)$

The primary idea of our algorithm for the Problem 6.6 consists of construction of the graphs G, G' and $\widehat{G}$, and use of the Theorem 6.7. As per the statement of the Problem 6.4, $WAPSMe(u, v)$ solves the APSM under WHD for a pair of equal length strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$. We extend this idea to the general case where the input strings are of different length: the pattern $p \in \Sigma_P^m$ of length m and the text $t \in \Sigma_T^n$ of length n ($\gg m$).

Algorithm 6.1 Compute WAPSM for a pattern p in the text t.

Input: (i) Alphabet sets: Σ_P and Σ_T with $\sigma = |\Sigma_P| = |\Sigma_T|$,
(ii) a pattern $p \in \Sigma_P^m$ of length m, a text $t \in \Sigma_T^n$ of length n,
(iii) a cost matrix $D(\Sigma_T) = (d_{ij})_{\sigma \times \sigma}$.

Output: APSM for p in t under WHD, that is, $WAPSM(p, t)$.

WAPSM(p, t)
1: **for** $i \leftarrow 1 : n - m + 1$
2: Construct G_i from the pair of strings p and $t[i..i + m - 1]$.
3: Create G_i' from G_i and $D(\Sigma_T)$.
4: Generate $\widehat{G_i}$ from G_i'.
5: Compute MWBM of $\widehat{G_i}$ and find the participating matching edges.
6: Solve $WAPSMe(p, t[i..i + m - 1])$ and compute
$cost(\, WAPSMe(p, t[i..i + m - 1])$.
7: **return** $WAPSM(p, t)$ and the corresponding cost at each locations.

The correctness of Algorithm WAPSM(p,t) follows from the construction of the graphs G, G' and $\widehat{G}$, and Theorem 6.7. The following theorem states the running time complexity of the Algorithm 6.1.

Theorem 6.10. *The Algorithm* WAPSM(p,t) *takes* $O(nm\sigma^{2.5}c_{Dmax})$ *time to solve the WAPSM for a pattern* $p \in \Sigma_P^m$ *in the text* $t \in \Sigma_T^n$.

Proof. The Algorithm WAPSM(p,t) solves the APSM for the pattern p of length m in the text t of length n where WHD is considered as a measure model of error. Let c_{Dmax} is the maximum element of the cost matrix $D(\Sigma_T)$, that is, $c_{Dmax} = \max\{D(\Sigma_T)\} = \max\{d_{ij} \mid i,j \in \{1,2,\ldots,\sigma\}\}$. In Step 2, for each $i \in \{1,2,\ldots,n-m+1\}$, the construction of G_i from p and $t[i..i+m-1]$ takes $O(m)$ time. Step 3 and Step 4 take at most $O(\sigma^2)$ time to create G'_i and $\widehat{G_i}$, respectively.

Let the local variables W, W' and $\widehat{W}$ are the weights of the graph G_i, G'_i and $\widehat{G_i}$, respectively. Then, as per the construction of the graphs: $W = m$ and

$$\begin{aligned} W' = \sum_{i=1}^{\sigma}\sum_{j=1}^{\sigma}\sum_{\substack{j'=1\\ j'\neq j}}^{\sigma} f_{ij'}\, d_{jj'} &= O\left(c_{Dmax}\sum_{i=1}^{\sigma}\sum_{j=1}^{\sigma}\sum_{\substack{j'=1\\ j'\neq j}}^{\sigma} f_{ij'}\right) \\ &= O\left((\sigma-1)c_{Dmax}\sum_{i=1}^{\sigma}\sum_{j=1}^{\sigma} f_{ij}\right) \\ &= O\big(W(\sigma-1)c_{Dmax}\big) \\ &= O(m\sigma c_{Dmax}). \end{aligned}$$

And the maximum possible weight of any edge in G' is

$$\begin{aligned} c_{G'max} &= \max\left\{\sum_{\substack{j'=1\\ j'\neq j}}^{\sigma} f_{ij'}\, d_{jj'} \mid i,j \in \{1,2,\ldots,\sigma\}\right\} \\ &= O\left(c_{Dmax}\sum_{i=1}^{\sigma}\sum_{j=1}^{\sigma} f_{ij}\right) \\ &= O(Wc_{Dmax}) \\ &= O(mc_{Dmax}). \end{aligned}$$

Therefore $\widehat{W} = O(mc_{Dmax})\,O(\sigma^2) = O(m\sigma^2 c_{Dmax})$. Hence in Step 5, a MWBM of $\widehat{G_i}$ can be computed in time $O(\widehat{W}\sqrt{\sigma}) = O(m\sigma^{2.5}c_{Dmax})$, by using the modified decomposition Algorithm 4.2. Once the edges of a MWBM in

$\widehat{G_i}$ are found, then computing the $WAPSMe(p, t[i..i+m-1])$ takes $O(m)$ time only, in Step 6. All together, for Steps 2–6 of each loop use $O(m) + O(\sigma^2) + O(m\sigma^{2.5}c_{Dmax}) + O(m) = O(m\sigma^{2.5}c_{Dmax})$ time. Therefore, the total time complexity of the Algorithm WAPSM(p, t) is $O(nm\sigma^{2.5}c_{Dmax})$. □

Remark 6.11. *If we assume σ and c_{Dmax} are bounded constants, then the above time complexity turns out to be $O(nm)$.*

6.2.4 Reducing MWBM Problem to the WAPSM Problem

In the Section 6.2.2, we have seen that the MWBM problem can be used to solve the APSM problem under WHD. Now we will see the sketch of the reverse reduction.

Let us consider $\widehat{G} = \{\Sigma_u \cup \Sigma_v, \widehat{E}, Wt\}$ be a weighted bipartite graph where $\Sigma_u = \{a_1, a_2, \ldots, a_\sigma\}$, $\Sigma_v = \{b_1, b_2, \ldots, b_\sigma\}$ and a cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma\times\sigma}$ (where $d_{ij} \in \mathbb{N}_0$, $1 \le i, j \le \sigma$) which depends on the working problem. Our target is to convert this graph $\widehat{G}$ to another bipartite graph G which corresponds to a pair of equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$, so that computing the APSM between u and v under WHD results to finding a MWBM in $\widehat{G}$ [57].

Let us do a reverse of the previous construction mentioned in the Section 6.2.2. We first transform the input weighted bipartite graph $\widehat{G}$ to another weighted error graph G', and then construct G from it based on the given cost matrix $D(\Sigma_v) = (d_{ij})_{\sigma\times\sigma}$, as follows.

Construction of G' from $\widehat{G}$: Let

$$c_{\widehat{G}max} = \max\{\, Wt(e_{ij}) \mid e_{ij} = \{a_i, b_j\} \in \widehat{E} \text{ where } i, j \in \{1, 2, \ldots, \sigma\}\}.$$

The graph G' is constructed by including every non-negative weighted edge $e_{ij} = \{a_i, b_j\}$ of $\widehat{G}$ (where $i, j \in \{1, 2, \ldots, \sigma\}$), and assign weight to such an edge in G' as

$$w_{ij} = c_{\widehat{G}max} - Wt(e_{ij}).$$

Construction of G from G' and $D(\Sigma_v)$: Let $f_{ij} = Wt(a_i, b_j)$ in G, where $i, j \in \{1, 2, \ldots, \sigma\}$. That is, weight of the edges incident to a vertex a_i in G (where $i \in \{1, 2, \ldots, \sigma\}$) are given by $\{f_{i1}, f_{i2}, \ldots, f_{i\sigma}\}$.

Therefore, we have the following weight equation with respect to the edge e_{ij} (where $i, j \in \{1, 2, \ldots, \sigma\}$) as:

$$\sum_{\substack{j'=1 \\ j' \neq j}}^{\sigma} f_{ij'}\, d_{jj'} = w_{ij}.$$

And hence for σ^2 number of possible edges, we now have a system of σ^2 number of linear equations in σ^2 unknowns f_{ij}, where $i, j \in \{1, 2, \ldots, \sigma\}$. These can be easily solved in $O(\sigma^{2\log_2 7})$ time [32] to find the weights f_{ij}s of the bipartite graph G.

The rest is now same as done in [57] to get the corresponding equal length strings $u \in \Sigma_u^*$ and $v \in \Sigma_v^*$. Let us consider an example (similar to the Example 6.9) to illustrate the above reduction for 3×3 cost matrix.

Example 6.12. Let $\widehat{G} = \{\Sigma_u \cup \Sigma_v, \widehat{E}, Wt\}$ be a weighted bipartite graph as shown in Figure 6.2(d) where $\Sigma_u = \{a, b, c\}$, $\Sigma_v = \{a', b', c'\}$. Further, the given cost matrix $D(\Sigma_v)$ corresponding to Σ_v is mentioned in Figure 6.2(a).

For the notational simplicity in this calculation, let us rename the graph vertices as follows:

$$a_1 = a,\ a_2 = b,\ a_3 = c;$$
$$b_1 = a',\ b_2 = b',\ b_3 = c'.$$

As shown in Figure 6.2(c), the bipartite graph G' is built from $\widehat{G}$ by replacing each weight $Wt(e_{ij}) = Wt(a_i, b_j)$ of $\widehat{G}$ with $c_{\widehat{G}max} - Wt(e_{ij})$, which we denoted as w_{ij} in the above reduction, where $i, j \in \{1, 2, 3\}$.

If we assume the variable f_{ij} as $f_{ij} = Wt(a_i, b_j)$ in G, where $i, j \in \{1, 2, 3\}$, then we have the following system of nine linear equations with nine unknowns f_{ij}s.

$$\begin{aligned}
&f_{12} * d(a', b') + f_{13} * d(a', c') = w_{11} = 0,\\
&f_{11} * d(b', a') + f_{13} * d(b', c') = w_{12} = 1,\\
&f_{11} * d(c', a') + f_{12} * d(c', b') = w_{13} = 1;\\
&f_{22} * d(a', b') + f_{23} * d(a', c') = w_{21} = 2,\\
&f_{21} * d(b', a') + f_{23} * d(b', c') = w_{22} = 0,\\
&f_{21} * d(c', a') + f_{22} * d(c', b') = w_{23} = 2;\\
&f_{32} * d(a', b') + f_{33} * d(a', c') = w_{31} = 3,\\
&f_{31} * d(b', a') + f_{33} * d(b', c') = w_{32} = 4,\\
&f_{31} * d(c', a') + f_{32} * d(c', b') = w_{33} = 3.
\end{aligned}$$

After Solving the above system of linear equations in f_{ij}s (where $i, j \in \{1, 2, 3\}$), we have the weight of the edges of G as follows:

$$f_{11} = 1,\ f_{12} = 0,\ f_{13} = 0;$$
$$f_{21} = 0,\ f_{22} = 2,\ f_{23} = 0;$$
$$f_{31} = 3,\ f_{32} = 0,\ f_{33} = 1.$$

Finally, we can generate a pair of strings $u = abbcccc \in \Sigma_u^7$ and $v = a'b'b'a'a'a'c' \in \Sigma_v^7$ from the graph G using the reduction mentioned by Hazay et al. [57]. Like previous, it can be easily proved that finding a π: $\Sigma_u \to \Sigma_v$ for the WAPSM between $u \in \Sigma_u^7$ and $v \in \Sigma_v^7$ corresponds to finding a maximum weight bipartite matching in $\widehat{G}$, where the matching edges are defined by the bijective mapping π. □

6.3 WPSM Problem with k Mismatches

PSM problem with k mismatches under HD was studied in [57]. See the Problem 2.14 described on page 17. Given $k \in \mathbb{N}_0$, $p \in \Sigma_P^m$ and $t \in \Sigma_T^n$, the problem of *PSM under HD with k mismatches* is to search a bijection π: $\Sigma_P \to \Sigma_T$ at each location i (where $1 \leq i \leq n - m + 1$) of t for which π-mismatch$(p, t[i..i + m - 1]) \leq k$. However, for each location i in the text t, any π with π-mismatch$(p, t[i..i + m - 1]) \leq k$ will be satisfactory.

For $|p| = |t| = m$, Hazay et al. [57] initially proposed an $O(m^{1.5})$ solution for this problem by using the MWBM algorithm [70]. The bipartite matching method views the whole string at once. However, with the intention to model an algorithm with running time dependent of k rather that m, they presented an $O(m+k^{1.5})$-time algorithm. It is based on a new method deciding whether a given location is match or mismatch before fixing a suitable bijection.

In this section we investigate the same problem but under WHD. We call it *Weighted Parameterized String Matching* (WPSM) problem between a pair of m-length strings with k mismatches, which considers PSM problem between a pair of m-length strings under the WHD with k mismatches. By adapting similar method of [57], we propose an $O(m + k)$-time solution for this problem.

Problem 6.13 (WPSM with k Mismatches)**.** *Given $k \in \mathbb{N}_0$, a pair of strings $u \in \Sigma_u^m$, $v \in \Sigma_v^m$ and a cost matrix $D(\Sigma_v)$, the WPSM with k mismatches seeks to find a bijection π: $\Sigma_u \to \Sigma_v$ such that the π-wmismatch$(u, v) \leq k$.*

We then say that u parameterized matches v with k mismatches under WHD. Like as the Problem 2.13, in this problem also any π: $\Sigma_u \to \Sigma_v$ with π-*wmismatch*$(u, v) \leq k$ is acceptable.

6.3.1 Mismatch Pairs and Their Properties under WHD

Definition 6.14 (Mismatch Pair [57]). *Given two equal length strings $u = u_1u_2 \dots u_m \in \Sigma_u^*$ and $v = v_1v_2 \dots v_m \in \Sigma_v^*$, a mismatch pair between u and v is a pair of locations (i, j) such that one of the following holds,*

(a) $u_i = u_j$ *and* $v_i \neq v_j$,

(b) $u_i \neq u_j$ *and* $v_i = v_j$.

The following lemma is inspired by the Lemma 3.3 in [57] and immediately follows from the above Definition 6.14. It is slightly extended and modified in the context of WHD.

Lemma 6.15. *Given two equal length strings $u = u_1u_2 \dots u_m \in \Sigma_u^*$ and $v = v_1v_2 \dots v_m \in \Sigma_v^*$, if (i, j) is a mismatch pair between u and v then for every bijection $\pi: \Sigma_u \to \Sigma_v$ either location i or location j, or both locations i and j are mismatches, that is, $\pi(u_i) \neq v_i$ or $\pi(u_j) \neq v_j$ or both are unequal [57, Lemma 3.3]. Let c_{Dmax} and c_{Dmin}, respectively, be the maximum and non-zero minimum entries of $D(\Sigma_v)$, then the mismatch pair (i, j) contributes*

(a) at least $\max\{d(\pi(u_i), v_i), d(\pi(u_j), v_j)\}$ *error which is greater than or equal to c_{Dmin} and*

(b) at most $d(\pi(u_i), v_i)+d(\pi(u_j), v_j)$ error which is less than or equal to $2c_{Dmax}$,

to the parameterized match between u and v under WHD.

Lemma 6.16 ([57]). *Let $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$, and $S \subseteq \{1, 2, \dots, m\}$ be a set of locations which does not contain any mismatch pair. Then there exists a bijection $\pi: \Sigma_u \to \Sigma_v$ that is parameterized match on the set of locations S, that is, for every $i \in S$, $\pi(u_i) = v_i$.*

Similar to [57, Section 3.2], the main idea of our algorithm for solving WPSM with k mismatches is to count the mismatch pairs between u and v. The Lemma 6.15 states that each mismatch pair contributes at least c_{Dmin} units of error to the parameterized match between u and v. So, if there are more than $\lfloor \frac{k}{c_{Dmin}} \rfloor$ mismatch pairs, then there exists no bijection from Σ_u to Σ_v for which u parameterized matches v with k error threshold under WHD.

Definition 6.17 (Maximal Disjoint Collection [57]). *Given two equal length strings u and v, a collection L of mismatch pairs is said to be a maximal disjoint collection if*

(a) all mismatch pairs in L are disjoint, that is, do not share a common location, and

(b) there is no mismatch pair that can be added to L without violating disjointness.

The following corollary is the extension of Corollary 3.6 in [57].

Corollary 6.18. *Let L be a maximal disjoint collection of mismatch pairs of two given strings u and v. If $|L| > \lfloor \frac{k}{c_{Dmin}} \rfloor$, then for every π: $\Sigma_u \to \Sigma_v$, the π-wmismatch$(u, v) > k$. If $|L| \leq \lfloor \frac{k}{c_{Dmax}} \rfloor/2$, then there exists π: $\Sigma_u \to \Sigma_v$ such that the π-wmismatch$(u, v) \leq k$.*

Proof. The proof follows from the Lemma 6.15, Lemma 6.16 and Definition 6.17. □

6.3.2 Algorithm for WPSM with k Mismatches under WHD

Let the number of mismatch pairs between u and v is denoted by mp, then according to the Corollary 6.18,

(a) if $mp \leq \lfloor \frac{k}{c_{Dmax}} \rfloor/2$, then u parameterized matches v, and

(b) if $mp > \lfloor \frac{k}{c_{Dmin}} \rfloor$, then u do not parameterized matches v

with k mismatches under WHD. But for the case when

$$\left\lfloor \frac{k}{c_{Dmax}} \right\rfloor/2 < mp \leq \left\lfloor \frac{k}{c_{Dmin}} \right\rfloor,$$

it is difficult to conclude whether u parameterized matches v or not with k error threshold under WHD. This is done by bipartite matching algorithm where bipartite graph needs to be constructed bit differently [57]. For simplicity we will not differentiate between the case of

$$mp \leq \left\lfloor \frac{k}{c_{Dmax}} \right\rfloor/2 \quad \text{and} \quad \left\lfloor \frac{k}{c_{Dmax}} \right\rfloor/2 < mp \leq \left\lfloor \frac{k}{c_{Dmin}} \right\rfloor,$$

and consider only the two cases:

$$mp \leq \left\lfloor \frac{k}{c_{Dmin}} \right\rfloor \quad \text{and} \quad mp > \left\lfloor \frac{k}{c_{Dmin}} \right\rfloor.$$

By using a simple stack scheme, we can find the set, say MP, of mismatch pair between u and v in $O(m)$ time and space, where $m = |u| = |v|$. See

more details in Section 3.3.1 of paper [57]. If $mp = |MP| \leq \lfloor \frac{k}{c_{Dmin}} \rfloor$, then we move to the verification stage to find the minimal π-*wmismatch* over all bijections by similar technique used to the Algorithm 6.1.

As defined in [57], let $L' \subset \{1, 2, \ldots, m\}$ be the locations that appear in a mismatch pair. We say that a symbol $a \in \Sigma_u$ (or Σ_v) is *matched* if there is a location $i \in L'$ such that $u_i = a$; otherwise $a \in \Sigma_u$ (or Σ_v) is called *free*. We construct a bipartite graph B of size $O(\lfloor \frac{k}{c_{Dmin}} \rfloor)$ in $O(\lfloor \frac{k}{c_{Dmin}} \rfloor)$ time using all the mismatch symbols belong to Σ_u and Σ_v, and some relevant free symbols. The rest is now similar as previous problem of Section 6.2. Construct $\widehat{B}$ from B based on the given $D(\Sigma_v)$ and find the weight of a MWBM of $\widehat{B}$. The whole process is framed in WPSM(u, v, k) of Algorithm 6.2, step by step.

Algorithm 6.2 Solution of WPSM with k mismatches under WHD.

Input: A pair of equal length strings $u \in \Sigma_u^m$ and $v \in \Sigma_v^m$, $D(\Sigma_v)$ and an error threshold $k \in \mathbb{N}_0$.

Output: Returns TRUE if u parameterized matches v with k threshold under WHD, else returns FALSE.

WPSM(u, v, k)

1: Find the set mismatch pairs between u and v by using simple stack technique. Let this set be MP.
2: $mp = |MP|$.
3: **if** $mp \leq \lfloor \frac{k}{c_{Dmax}} \rfloor / 2$ **then**
4: **return** TRUE.
5: **else if** $mp > \lfloor \frac{k}{c_{Dmin}} \rfloor$ **then**
6: **return** FALSE.
7: **else if** $mp \leq \lfloor \frac{k}{c_{Dmin}} \rfloor$ **then**
8: Construct a bipartite graph B using all the mismatch symbols belong to Σ_u and Σ_v, and some relevant free symbols.
9: Create a new bipartite graph $\widehat{B}$ from B as mentioned in Section 6.2.2.
10: Compute a MWBM of $\widehat{B}$.
11: **if** weight of a MWBM of $\widehat{B}$ is less than or equal to k,
12: **then return** TRUE.
13: **return** FALSE.

Theorem 6.19. *The Algorithm* WPSM(u, v, k) *takes* $O(m + \lfloor \frac{k}{c_{Dmin}} \rfloor \sqrt{\sigma})$ *time to solve the WPSM problem with* k *threshold under WHD.*

Proof. Given a pair of equal length strings u and v with $m = |u| = |v|$, $D(\Sigma_v)$ and an error threshold k, WPSM with k mismatches can be solved in $O(m + \lfloor \frac{k}{c_{Dmin}} \rfloor \sqrt{\sigma})$ time: $O(m)$ time to find mismatch pairs between u and v in Step 1, and $O(\lfloor \frac{k}{c_{Dmin}} \rfloor \sqrt{\sigma})$ to check if u approximate parameterized matches v with k threshold under WHD in Steps 8–10. □

Remark 6.20. *If we assume σ and c_{Dmin} are bounded constants, then the above time complexity turns out to be $O(m + k)$.*

6.4 Conclusions

We have explored the WAPSM matching problem of a given pattern of length m in a text of length n under weighted Hamming distance error model. An $O(nm)$ time algorithm is presented by using the MWBM algorithm as a subroutine. Further we have solved the WPSM problem with k mismatches under weighted Hamming distance in $O(m+k)$ time. Both the problems are considered under the assumption that the parameters alphabet size, c_{Dmin} and c_{Dmax} are constant.

Chapter 7

Conclusions and Future Work

In this chapter we summarize this thesis and its contributions. We also report some unsolved questions related to the addressed problems in this thesis that might be of interest for further research in future.

In this thesis we have presented the solution of problems related to approximate parameterized string matching in stringology and related problems in graph theory. In particular, following are the primary findings of the thesis. We have explored enumerative and combinatorial properties of error classes in approximate parameterized string matching under HD. An improvement of the existing deterministic algorithm for the maximum weight bipartite matching problem has proposed. We have designed a filtering technique and an algorithm for all pairs approximate parameterized string matching problem under HD. We also present a algorithm for approximate parameterized string matching problem under the weighted Hamming distance model. To check the efficiency of the suggested techniques we have evaluated some of the designated techniques experimentally.

7.1 Contributions of the Thesis

After stating overview and motivation of the thesis in Chapter 1 and the necessary fundamentals and related works in Chapter 2, the key contributions of the thesis are pointed out below, chapter wise.

Chapter 3: In this chapter, we have discovered a tight lower bound on the weights of MWBM of graphs which is useful in the proof of counting the number of error classes in APSM problem under HD. We have explored several combinatorial properties of these classes, such as distribution of empty and non-empty ECs, exact count of non-empty ECs and divisibility property of the size of ECs in APSM.

Chapter 4: We have fine-tuned the existing decomposition theorem originally proposed by Kao et al. in [70], in the context of maximum weight bipartite matching and applied it to design a revised version of the decomposition algorithm to compute the weight of a maximum weight bipartite matching in $O(\sqrt{|V|}W'/k(|V|, W'/N))$ time by employing an algorithm designed by Feder and Motwani [45] as base algorithm, where $k(x, y) = \log x/\log(x^2/y)$ and $|E| \leq W' \leq W$. The algorithm performs well especially when the largest edge weight differs by more than one from the second largest edge weight in the current working graph during an invocation of Wt-Mwbm( ) in an iteration. In best case $W' = O(|E|)$ and in worst case $W' = W$, that is, $|E| \leq W' \leq W$. Further, we have given a scaling property of the algorithm and a bound of the parameter W' as $|E| \leq W' \leq \frac{W}{GCD(w_1,w_2,\ldots,w_{|E|})} \leq W$ where $GCD(w_1, w_2, \ldots, w_{|E|})$ denotes the GCD of the positive edges weights $\{w_1, w_2, \ldots, w_{|E|}\}$ of the weighted bipartite graph. The algorithm works well for general W, but is best known for $W' = o(|E| \log(|V|N))$.

This time complexity bridges a gap between the best known time complexity for computing a Maximum Cardinality Matching (MCM) of unweighted bipartite graph and that of computing a MWBM of a weighted bipartite graph. In best case, for computation of weight of a MWBM, the Algorithm 4.2 takes $O(\sqrt{|V|}|E|/k(|V|, |E|))$ time which is the same as the complexity of the Feder and Motwani's algorithm [45] for computing MCM of unweighted bipartite graph; whereas in worst case this algorithm takes $O(\sqrt{|V|}W/k(|V|, W/N))$ time which is the same as the complexity of the Kao et al.'s algorithm [70]. However, it is very difficult and challenging to get rid of W or N from the complexity.

We have also analyzed the algorithm by using Hopcroft-Karp algorithm [59] and Alt-Blum-Mehlhorn-Paul algorithm [4] as base algorithms, respectively. We have also done experimental study to show the performance of the modified decomposition algorithm with respect to Kao et al.'s algorithm.

Chapter 5: In this chapter we have investigated all pairs approximate parameterized string matching problem with k Hamming distance error threshold between two sets of equal length stings. We have presented a solution with worst case complexity $O(n_P\, n_T\, m)$, assuming constant alphabet size. In order to minimize the number of paired comparisons for solving APSM between the strings in a pair with error threshold, we have proposed a PV-filtering technique by using Parikh vector. Although the filter does not improve the worst case asymptotic bound,

but the using it as a subroutine, we can avoid some of the non-candidate paired comparison for APSM. Experimental results show that the PV-filter is efficient for small error threshold.

Chapter 6: We have explored weighted approximate parameterized string matching problem of a given pattern of length m in a text of length n under weighted Hamming distance error model. An $O(nm)$-time algorithm is presented by using the maximum weight bipartite matching algorithm as a subroutine. Next we have solved the parameterized string matching problem with k mismatches under weighted Hamming distance in $O(m+k)$ time. Both the problems are considered under the assumption of constant parameters: size of alphabet, c_{Dmin} and c_{Dmax}.

7.2 Future Work

Some unsolved questions related to the problems addressed in this thesis is given below, which might be of interest for further research in future.

- There are some other variation of operations which lead to some more error models, such as Damerau distance [37], LCS distance [12], episode distance [38] etc. In Chapter 3 we have done the combinatorial studies of the error classes in APSM problem under Hamming distance error model only. One may investigate these combinatorial properties of the error classes for different error models.
- Analyzing average case of the Algorithm 4.2 of Chapter 4, might improve the complexity of the algorithm.
- For all pairs approximate parameterized string matching problem addressed in Chapter 5, one may provide an average case analysis and provide extensive experimentation, in order to get better complexity of the solution.
- The Algorithm 6.1 is a simple loop over and considers a subroutine (the MWBM algorithm) as a black-box. It would be better to find an incremental matching algorithm to obtain a better bound to the WAPSM problem for a pattern $p \in \Sigma_P^m$ in the text $t \in \Sigma_T^n$.
- Algorithms presented in Chapter 6 deal with weighted Hamming distance only. It would be interesting to investigate the weighted approximate parameterized string matching problem under some other error model.

Appendix A

Detailed Complexity Analysis of Algorithm 4.2

Here we give complexity analysis of the Algorithm 4.2 in general. It is almost similar as done in the paper [70]. We assume that a maximum heap [32] is used to store the distinct edge weights along with the associated edges of G.

Let the running time of Wt-Mwbm(G) be $T(|V|, W', N)$ excluding the initialization. Let L be the set of the heaviest weight edges in G. So up to Step 3, construction of G_h requires $O(|L| \log |E|)$ time. The Step 4 takes $O(\sqrt{|V|}|L|/k(|V|, |L|))$ time by using Feder and Motwani's algorithm [45] to compute $mm(G_h)$. In Step 5, C_h can be found in $O(|L|)$ time from this matching. Let L_1 be the set of edges of G adjacent to some node u with $C_h(u) > 0$, i.e., L_1 consist of edges of G whose weights reduce in G_h^{Δ}. Let $l_1 = |L_1|$. Step 6 updates every edges of L_1 in the heap in $O(l_1 \log |E|)$ time. Since $L \subseteq L_1$, Step 1 to 6 takes $O(\sqrt{|V|}l_1/k(|V|, l_1))$ time altogether. Let $l_i = |L_i|$ for $i = 1, 2, \ldots, p \leq N$ and $h_i = H_1 - H_2$ for i-th phase of the recursion, where L_i consists of edges of remaining G whose weights reduce in G_h^{Δ} on i-th iteration. Note that,

$$l_1 h_1 + l_2 h_2 + \cdots + l_p h_p = W.$$

Let $l_1 + l_2 + \cdots + l_p = W'$. Observe that if $h_i = 1$ for all $i \in [1, p]$, then $W' = \sum_{i=1}^{p} l_i = W$. Step 7 uses at most $T(|V|, W'', N'')$ time, where W'' $(< W')$ is the total weight of G_h^{Δ} and N'' $(< N)$ is the maximum edge weight of G_h^{Δ}. Hence the recurrence relation for running time is

$$T(|V|, W', N) = O(\sqrt{|V|}l_1/k(|V|, l_1)) + T(|V|, W'', N'') \quad \text{and}$$
$$T(|V|, 0, 0) = 0$$

$$\begin{aligned}
&\therefore\ T(|V|, W', N) \\
&= O\left(\frac{\sqrt{|V|}l_1}{k(|V|, l_1)}\right) + O\left(\frac{\sqrt{|V|}l_2}{k(|V|, l_2)}\right) + \cdots + O\left(\frac{\sqrt{|V|}l_p}{k(|V|, l_p)}\right) \\
&= O\left(\sqrt{|V|}\left(\frac{l_1}{k(|V|, l_1)} + \frac{l_2}{k(|V|, l_2)} + \cdots + \frac{l_p}{k(|V|, l_p)}\right)\right) \\
&= O\left(\frac{\sqrt{|V|}}{\log|V|}\left(\log|V|^2 \sum_{i=1}^{p} l_i - \sum_{i=1}^{p} l_i \log l_i\right)\right)
\end{aligned}$$

Let $f(x) = x \log x$. Note that it is a convex function, so by Jensen's inequality[1],

$$\begin{aligned}
\sum_{i=1}^{p} l_i \log l_i = \sum_{i=1}^{p} f(l_i) \geq pf\left(\frac{\sum_{i=1}^{p} l_i}{p}\right) = pf\left(\frac{W'}{p}\right) \\
= p\frac{W'}{p}\log\frac{W'}{p} = W'\log\frac{W'}{p} = O\left(W'\log\frac{W'}{N}\right)
\end{aligned}$$

This lead to the running time complexity as follows.

$$\begin{aligned}
&T(|V|, W', N) \\
&= O\left(\frac{\sqrt{|V|}}{\log|V|}\left(W'\log|V|^2 - W'\log\frac{W'}{N}\right)\right) \\
&= O\left(\frac{\sqrt{|V|}W'}{\log|V| / \log\frac{|V|^2}{W'/N}}\right) \\
&= O(\sqrt{|V|}W'/k(|V|, W'/N)).
\end{aligned}$$

This is better than the $O(\sqrt{|V|}W/k(|V|, W/N))$ time as mentioned in [70].

The parameter W' is smaller than W which is the total weight of G, essentially when the heaviest edge weight differs by more than one unit from the second heaviest edge weight in a current working graph during a decomposition in any iteration of the algorithm. In best case the algorithm takes

[1]**Jensen's Inequality** [56]. If $f(x)$ is a convex function on an interval I and $\mu_1, \mu_2, \ldots, \mu_n$ are positive weights such that $\sum_{i=0}^{n} \mu_i = 1$ then

$$f\left(\sum_{0}^{n} \mu_i x_i\right) \leq \sum_{i=0}^{n} \mu_i f(x_i).$$

$O(\sqrt{|V|}|E|/k(|V|,|E|))$ time to compute a maximum weight matching and in worst case $O(\sqrt{|V|}W/k(|V|,W/N))$, that is, $|E| \leq W' \leq W$. This time complexity bridges a gap between the best known time complexity for computing a Maximum Cardinality Matching (MCM) of unweighted bipartite graph and that of computing a MWBM of a weighted bipartite graph.

However, it is very difficult and challenging to get rid of W or N from the complexity. This modified algorithm works well for general W, but is best known for $W' = o(|E| \log(|V|N))$.

References

[1] Scientists create first living organism that transmits added letters in DNA 'alphabet'. `http://www.scripps.edu/newsandviews/e_20140512/romesberg.html`, NEWS & VIEWS at The Scripps Research Institute (TSRI) in California. 2014.

[2] Karl Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, December 1987.

[3] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, June 1975.

[4] Helmut Alt, Norbert Blum, Kurt Mehlhorn, and Markus Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Information Processing Letters*, 37(4):237–240, 1991.

[5] Amihood Amir, Yonatan Aumann, Richard Cole, Moshe Lewenstein, and Ely Porat. Function matching: Algorithms, applications, and a lower bound. In *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 929–942. Springer Berlin Heidelberg, 2003.

[6] Amihood Amir, Martin Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49(3):111–115, 1994.

[7] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004.

[8] Amihood Amir and Gonzalo Navarro. Parameterized matching on non-linear structures. *Information Processing Letters*, 109(15):864–867, July 2009.

[9] Alberto Apostolico, Péter L. Erdős, and Alpár Jüttner. Parameterized searching with mismatches for run-length encoded strings. *Theoretical Computer Science*, 454(0):23–29, 2012. Formal and Natural Computing Honoring the 80^{th} Birthday of Andrzej Ehrenfeucht.

[10] Alberto Apostolico, Péter L. Erdős, and Moshe Lewenstein. Parameterized matching with mismatches. *Journal of Discrete Algorithms*, 5(1):135–140, 2007.

[11] Alberto Apostolico and Raffaele Giancarlo. Periodicity and repetitions in parameterized strings. *Discrete Applied Mathematics*, 156(9):1389–1398, 2008. General Theory of Information Transfer and Combinatorics.

[12] Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1-4):315–336, 1987.

[13] G. Phanendra Babu, Babu M. Mehtre, and Mohan S. Kankanhalli. Color indexing for efficient image retrieval. *Multimedia Tools and Applications*, 1(4):327–348, 1995.

[14] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, October 1992.

[15] Ricardo Baeza-yates and Gonzalo Navarro. Faster approximate string matching. *Algorithmica*, 23:127–158, 1999.

[16] Brenda S. Baker. A program for identifying duplicated code. In *Computer Science and Statistics: 24^{th} Symposium on the Interface*, pages 49–57, March 1992.

[17] Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Symposium on Theory of Computing*, pages 71–80. ACM, 1993.

[18] Brenda S. Baker. Parameterized pattern matching by boyer-moore-type algorithms. In 6^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 541–550, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[19] Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996.

[20] Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26(5):1343–1362, October 1997.

[21] Brenda S. Baker. Parameterized diff. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 854–855. Society for Industrial and Applied Mathematics, 1999.

[22] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, NY, USA, 5^{th} edition, 1982.

[23] J. A. Bondy and U. S. R. Murty. *Graph Theory*, volume 244, chapter 16 Matchings, page 419. Springer, 1^{st} edition, 2008.

[24] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, October 1977.

[25] Emilios Cambouropoulos, Maxime Crochemore, Costas S. Iliopoulos, Laurent Mouchard, and Yoan J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. *International Journal of Computer Mathematics*, 79(11):1135–1148, 2002.

[26] Christian Charras and Thierry Lecroq. *Handbook of Exact String Matching Algorithms*. King's College Publications, 2004.

[27] Joseph Cheriyan, Torben Hagerup, and Kurt Mehlhorn. Can a maximum flow be computed in $o(nm)$ time? In 17^{th} *International Colloquium on Automata, Languages and Programming*, pages 235–248, New York, NY, USA, 1990. Springer-Verlag.

[28] Raphaël Clifford and Ely Porat. A filtering algorithm for k-mismatch with don't cares. *Information Processing Letters*, 110(22):1021–1025, October 2010.

[29] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In 36^{th} *Annual ACM Symposium on Theory of Computing*, pages 91–100, New York, NY, USA, 2004. ACM.

[30] Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM Journal on Computing*, 33(1):26–42, January 2004.

[31] Richard Cole, Carmit Hazay, Moshe Lewenstein, and Dekel Tsur. Two-dimensional parameterized matching. *ACM Transactions on Algorithms*, 11(2):12:1–12:30, October 2014.

[32] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3^{rd} edition, 2009.

[33] Maxime Crochemore, Artur Czumaj, Leszek Gasieniec, Stefan Jarominek, Thierry Lecroq, Wojciech Plandowski, and Wojciech Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12(4-5):247–267, 1994.

[34] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007.

[35] Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.

[36] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology: Text Algorithms*. World Scientific Press, 2002.

[37] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[38] Gautam Das, Rudolf Fleischer, Leszek Gasieniec, Dimitrios Gunopulos, and Juha Kärkkäinen. Episode matching. In 8^{th} *Annual Symposium on Combinatorial Pattern Matching*, pages 12–27, London, UK, 1997. Springer-Verlag.

[39] Satoshi Deguchi, Fumihito Higashijima, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Parameterized suffix arrays for binary strings. In *The Prague Stringology Conference, Prague, Czech Republic*, pages 84–94, 2008.

[40] E. A. Dinic and M. A. Kronrod. An algorithm for the solution of the assignment problem. *Soviet Mathematics Doklady*, 10:1324–1326, 1969.

[41] N. Rex Dixon and Thomas B. Martin, editors. *Automatic Speech and Speaker Recognition*. IEEE Press, New York, 1979.

[42] Ran Duan and Seth Pettie. Approximating maximum weight matching in near-linear time. In *Annual Symposium on Foundations of Computer Science*, pages 673–682, Washington, DC, USA, 2010. IEEE Computer Society.

[43] Simone Faro and Thierry Lecroq. The exact online string matching problem: A review of the most recent results. *ACM Computing Surveys*, 45(2):13:1–13:42, March 2013.

[44] Simone Faro and Thierry Lecroq. Smart: a string matching algorithm research tool. `http://www.dmi.unict.it/~faro/smart/`, University of Catania and University of Rouen, 2011.

[45] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, October 1995.

[46] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[47] Kimmo Fredriksson and Maxim Mozgovoy. Efficient parameterized string matching. *Information Processing Letters*, 100(3):91–96, 2006.

[48] Harold N. Gabow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148–168, September 1985.

[49] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, October 1989.

[50] Zvi Galil. Open problems in stringology. In *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI Series*, pages 1–8. Springer Berlin Heidelberg, 1985.

[51] Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, March 1986.

[52] Szymon Grabowski and Kimmo Fredriksson. Bit-parallel string matching under hamming distance in $O(n\lceil m/w \rceil)$ worst case time. *Information Processing Letters*, 105(5):182–187, February 2008.

[53] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

[54] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[55] Richard W. Hamming. *Coding and Information Theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2^{nd} edition, 1986.

[56] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 1934.

[57] Carmit Hazay, Moshe Lewenstein, and Dina Sokol. Approximate parameterized matching. *ACM Transactions on Algorithms*, 3(3), August 2007.

[58] Carmit Hazay, Moshe Lewenstein, and Dekel Tsur. Two dimensional parameterized matching. In *Combinatorial Pattern Matching*, volume 3537 of *Lecture Notes in Computer Science*, pages 266–279. Springer Berlin Heidelberg, 2005.

[59] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[60] Xiaoqiu Huang. A lower bound for the edit-distance problem under an arbitrary cost function. *Information Processing Letters*, 27(6):319–321, 1988.

[61] Tomohiro I, Satoshi Deguchi, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Lightweight parameterized suffix array construction. In 20^{th} *International Workshop on Combinatorial Algorithms*, volume 5874 of *Lecture Notes in Computer Science*, pages 312–323. Springer Berlin Heidelberg, 2009.

[62] Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Counting parameterized border arrays for a binary alphabet. In 3^{rd} *International Conference on Language and Automata Theory and Applications*, pages 422–433, Berlin, Heidelberg, 2009. Springer-Verlag.

[63] Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Verifying a parameterized border array in $O(n^{1.5})$ time. In 21^{st} *Annual Conference on Combinatorial Pattern Matching*, pages 238–250, Berlin, Heidelberg, 2010. Springer-Verlag.

[64] Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Verifying and enumerating parameterized border arrays. *Theoretical Computer Science*, 412(50):6959–6981, 2011.

[65] Ramana M. Idury and Alejandro A. Schäffer. Multiple matching of parameterized patterns. *Theoretical Computer Science*, 154(2):203–224, 1996.

[66] Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, and Tomasz Waleń. Algorithms for computing the longest parameterized common subsequence. In *Combinatorial Pattern Matching*, volume 4580 of *Lecture Notes in Computer Science*, pages 265–273. Springer Berlin Heidelberg, 2007.

[67] M. Iri. A new method for solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3:27–87, 1960.

[68] Ming Yang Kao. *Encyclopedia of Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2008.

[69] Ming Yang Kao, Tak Wah Lam, Wing Kin Sung, and Hing Fung Ting. A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees. In *Annual European Symposium on Algorithms*, pages 438–449. Springer-Verlag, 1999.

[70] Ming Yang Kao, Tak Wah Lam, Wing Kin Sung, and Hing Fung Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal on Computing*, 31(1):18–26, January 2001.

[71] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development - Mathematics and Computing*, 31(2):249–260, March 1987.

[72] Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. On the longest common parameterized subsequence. *Theoretical Computer Science*, 410:5347–5353, November 2009.

[73] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[74] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 4^{th} edition, 2007.

[75] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, March 1955.

[76] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, April 1998.

[77] Gad M. Landau and Uzi Vishkin. Efficient string matching in the presence of errors. In 26^{th} *Annual Symposium on Foundations of Computer Science*, pages 126–136, Washington, DC, USA, 1985. IEEE Computer Society.

[78] Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[79] Inbok Lee, Juan Mendivelso, and YoanJ. Pinzn. $\delta\gamma$ – parameterized matching. In *String Processing and Information Retrieval*, volume 5280 of *Lecture Notes in Computer Science*, pages 236–248. Springer Berlin Heidelberg, 2009.

[80] Taehyung Lee, Joong Chae Na, and Kunsoo Park. On-line construction of parameterized suffix trees. In *String Processing and Information Retrieval*, volume 5721 of *Lecture Notes in Computer Science*, pages 31–38. Springer Berlin Heidelberg, 2009.

[81] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, 1:8–17, 1965.

[82] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[83] Tomasz Luczak and Wojciech Szpankowski. A suboptimal lossy data compression based on approximate pattern matching. *IEEE Transactions on Information Theory*, 43(5):1439–1451, 1997.

[84] Denis A. Malyshev, Kirandeep Dhami, Thomas Lavergne, Tingjian Chen, Nan Dai, Jeremy M. Foster, Ivan R. Correa, and Floyd E. Romesberg. A semi-synthetic organism with an expanded genetic alphabet. *Nature*, 509(7500):385–388, May 2014.

[85] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[86] Gordon E. Moore. 1965 - "Moore's Law" predicts the future of integrated circuits. `http://www.computerhistory.org/semiconductor/timeline/1965-Moore.html`, Computer History Museum. 1965.

[87] Gordon E. Moore. Excerpts from a conversation with gordon Moore: Moore's Law. `http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A_Conversation_with_Gordon_Moore.pdf`, Intel Corporation. 2005.

[88] J. H. Morris and V. R. Pratt. A linear pattern matching algorithm. Technical Report 40, Computing Center, University of California, Berkeley, 1970.

[89] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[90] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33:31–88, March 2001.

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

[92] Rohit J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, October 1966.

[93] Rajesh Prasad and Suneeta Agarwal. Study of bit-parallel approximate parameterized string matching algorithms. In *Contemporary Computing*, volume 40 of *Communications in Computer and Information Science*, pages 26–36. Springer Berlin Heidelberg, 2009.

[94] S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees. In 36^{th} *Annual Symposium on Foundations of Computer Science*, pages 631–638, Washington, DC, USA, 1995. IEEE Computer Society.

[95] Leena Salmela and Jorma Tarhio. Fast parameterized matching with q-grams. *Journal of Discrete Algorithms*, 6(3):408–419, 2008.

[96] Piotr Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theoretical Computer Science*, 410(44):4480–4488, October 2009.

[97] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*, volume 24 A. Springer, 2003.

[98] Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980.

[99] Tetsuo Shibuya. Generalization of a suffix tree for rna structural pattern matching. *Algorithmica*, 39(1):1–19, January 2004.

[100] Bill Smyth. *Computing Patterns in Strings*. Pearson Addison-Wesley, New York, 2003.

[101] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, November 1991.

[102] Robert A. Wagner. On the complexity of the extended string-to-string correction problem. In 17^{th} *Annual ACM Symposium on Theory of Computing*, pages 218–223, New York, NY, USA, 1975. ACM.

[103] Robert A. Wagner and Roy Lowrance. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183, April 1975.

[104] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2^{nd} edition, September 2000.

[105] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32. Springer, 2006.

Publications of the Author

[1] **Shibsankar Das** and Kalpesh Kapoor. Fine-tuning decomposition theorem for maximum weight bipartite matching. In *Theory and Applications of Models of Computation*, volume 8402 of *Lecture Notes in Computer Science*, pages 312–322. Springer International Publishing, 2014.

[2] **Shibsankar Das**, Jan Holub, and Kalpesh Kapoor. All pairs approximate parameterized string matching. Technical Report FIT-2014-01, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic, March 2014.

[3] **Shibsankar Das** and Kalpesh Kapoor. Approximate parameterized string matching under weighted hamming distance. *AKCE International Journal of Graphs and Combinatorics*. Elsevier, (to appear) 2016.

Manuscripts Communicated / under Preparation

[4] **Shibsankar Das**, Jan Holub, and Kalpesh Kapoor. A filtering technique for all pairs approximate parameterized string matching.

[5] **Shibsankar Das** and Kalpesh Kapoor. A combinatorial study of error classes in approximate parameterized string matching.

[6] **Shibsankar Das** and Kalpesh Kapoor. Experimental evaluation of decomposition algorithm for maximum weight bipartite matching.

[7] **Shibsankar Das** and Kalpesh Kapoor. A tight lower bound for the weights of maximum weight matching in bipartite graphs. (Communicated)