Framework forensic examination computer systems

Gong, Ruibin

2007

Gong, R. B. (2007). Framework forensic examination computer systems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2470>

<https://doi.org/10.32657/10356/2470>

Nanyang Technological University

Downloaded on 26 Apr 2023 18:25:48 SGT

NANYANG TECHNOLOGICAL UNIVERSITY

FRAMEWORK FOR FORENSIC EXAMINATION
OF COMPUTER SYSTEMS

by

GONG RUIBIN

A thesis submitted to the
School of Computer Engineering
in conformity with the requirements for
the degree of Doctor of Philosophy

July 2007, Singapore

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by myself
and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Signature

Preface

I heard the term “Computer Forensics” for the first time when this project started in 2002. Now Computer Forensics has become a hot topic, appearing in the headlines of tones of technique reports, research papers and public newswires frequently. But to the public, computer forensics is still a mysterious domain. Even inside the computer forensics community, the following concepts need to be clarified.

Firstly, in-advance prevention is always more effective than after-event processing. Computer forensics offers methods to locate, collect and recover evidence derived from digital sources, but cannot find evidence that has been destroyed permanently. The mechanism for data recording, back up, access control and emergency response should be considered as an integrated part of computer forensics.

The information needed in computer forensics is highly selective rather than comprehensive. Although the crime investigation tries to find every tiny piece of evidence, they focus on a narrow scope of the whole scenario, due to the legal restrictions, limited resource and the nature of criminal investigation. Under such circumstances, computer forensics is expected to be able to supply highly target-oriented processing of the captured data. This is the major concern and motivation of this thesis.

Computer forensics still has a long way to go. We believe our work will add a solid brick to its development.

Abstract

Computer forensics has undergone fast growth in the past five years. As the methods on digital evidence capturing have been well established and widely accepted by court, the research focus has shifted from the evidence collecting and preservation to the evidence extraction and analysis part — from the traditional on-site forensic techniques to the research of modern information systems.

A big challenge for computer forensics is how to deal with the complicated human's behaviors in the cyberworld. Recognition, extraction and analysis of evidence from these behaviors are far beyond the abilities of the traditional information retrieval technologies used in current investigation systems. To build a systematic mechanism for knowledge collection, management, sharing, reuse, and decision support for the investigators, computational intelligence technology *should* and *could* play an active role in computer forensics.

This thesis discusses the features and requirements of a computationally intelligent computer forensic system. By introducing a novel concept, “Case-Relevance”, a computationally intelligent forensic framework has been proposed. The details of binding the “Case-Relevance” concept with the current data analysis phase in computer forensic investigation have been given in two investigation scenarios. An Email evidence extraction algorithm has been developed based on the proposed framework.

The experimental results fully demonstrated the advantage of such a computer forensic framework.

Phonetic string search is a very important technique, which has been widely used in computer forensics. In this thesis, a novel phonetic string search algorithm, Syllable Alignment Pattern Searching (SAPS), which computes the optimal pairing of the corresponding syllables in two strings, is proposed. Its major advantage is not only the high efficiency in retrieving phonetic strings, but also the flexibility to achieve optimal performance on different datasets. The time complexity of phonetic string search has also been studied. A fast algorithm called *FONS* is designed to run in linear time $O(n)$ and can be extended to search multiple phonetic strings simultaneously to cut down the search time further.

Acknowledgments

Firstly, I would like to give my sincere appreciation to my first supervisor, Associate Professor Chan Kai Yun Tony, for his remarkable and heuristic guidance throughout my Ph.D study. As a patient listener, a helpful advisor and a strong supporter, he always encourages me to explore my own ideas. In particular, I want to thank him for showing me some of the most valuable personal characteristics and right attitude.

I also would like to thank my second supervisor, Assistant Professor Shi Daming. Dr. Shi gave me a lot of useful comments and suggestion to my work, especially during the writing of this thesis.

I also would give my thanks to Mr. Mathias Gaertner, my exchange program mentor in Fraunhofer IGD, Germany. His multi-facet background in IT evidence investigation provided a lot of useful information and inspirations for my research project.

Thanks also go to the technique staff in Centre for Advanced Information Systems (CAIS) and Centre for High Performance Embedded Systems (CHiPES) for their excellent job in maintaining a comfortable environment.

At last, I would also thank my dear wife, Xiaoguang, and my parents, for all the love and support.

Contents

Preface	ii
Abstract	iii
Acknowledgments	v
Contents	vi
List of Tables	ix
List of Figures	x
1 Introduction	2
1.1 Backgrounds	2
1.2 Motivation	7
1.3 Objectives	9
1.4 Contributions	9
1.5 Organization of Thesis	10
2 Related Work	11
2.1 Frameworks for Digital Evidence Investigation	13
2.2 String Search	15
2.2.1 Approximate String Search	16
2.2.2 Multiple String Search	20
2.2.3 Phonetic String Search	21
2.3 Email Information Processing	25
3 Case-Relevance Computer Forensic Framework	32
3.1 Motivation	33
3.2 “Case-Relevance”	35
3.3 Case-Relevance Information Investigation	38

3.3.1	Case-Relevance Information Extraction	38
3.3.2	Case-Relevance Information Confirmation	43
3.4	Summary	43
4	Email Analysis for Computer Forensics	46
4.1	Motivation	47
4.2	“Case-Relevance” Email Evidence Extraction	52
4.2.1	General Idea	52
4.2.2	“Case-Relevance” Email Evidence Extraction	54
4.3	Experimental Result and Analysis	58
4.4	Summary	61
5	Phonetic String Search of Written Text	62
5.1	Motivation	62
5.2	Criteria of an Optimal Method	64
5.3	Syllable Alignment Pattern Searching	65
5.4	Adaptive Phonetic String Search	73
5.5	High Speed Phonetic String Search	76
5.5.1	Fast Online Searching Algorithm	77
5.5.2	Multiple Phonetic String Search	80
5.6	Experimental Result and Analysis	82
5.6.1	Experimental Environment	82
5.6.2	Results and Analysis	85
5.7	Summary	91
6	Conclusions	94
6.1	Conclusions	94
6.2	Future Work	95
	Bibliography	99
A	Experiment Data of SAPS	116
A.1	Query List of COMPLETE Dataset	116
A.2	Returns of SAPS with COMPLETE Dataset	117
B	Preprocessing Step in Phonix	132
B.1	Part of the C Source Code	132
C	Computer Forensics Resources	139
C.1	Information and Organizaions	139
C.2	Publications	141

C.3 Software	141
C.4 Certifications	143
Author's Publication List	145

List of Tables

2.1	Comparison between Soundex, Jaro's method and Edit Distance(ED) [126]	22
3.1	Key Distinctions between Computer Security and Forensic Computing [24]	37
4.1	Test Email Clusters	58
4.2	Result for CRE3, TF-IDF and Threading	61
5.1	Phonetic Letter Groups	77
5.2	Sample Queries and Match Lists in COMPLETE Dataset	83
5.3	Correct Matches in Top Returns	85
5.4	Overlap of the Returns (COMPLETE Dataset)	86
5.5	Overlap of FONS (COMPLETE Dataset)	89

List of Figures

1.1	Locard's Exchange Principle	3
1.2	Crime Types at Different Levels	3
2.1	Three Generations of Computer Forensic Investigation System	12
2.2	Single Tier Digital Investigations Process Framework [20]	14
2.3	Data Analysis Phase [20]	14
2.4	Email Thread Information	27
3.1	Searching/Extraction Loop	34
3.2	Degrees of Case-Relevance	36
3.3	Case-Relevance Information Extraction	39
3.4	Automatic Evidence Extraction Module	41
3.5	Case-Relevance Information Confirmation	44
4.1	An Email Evidence Extraction Scenario	50
4.2	Forensic Email Evidence Extraction	51
4.3	Automatic Email Evidence Extraction	53
4.4	Sample of IE Templates	59
5.1	The Scale of the Parameters	69
5.2	Alignment Matrix of "Stevenson" and "Stinson" and Backtracking Path	73
5.3	Preference Order of Backtracking Path	73
5.4	Training of Adaptive SAPS	75
5.5	A Superimposed NFA to search "Tom" and "Sam" simultaneously . .	81
5.6	Multiple String Search Procedure	81
5.7	<i>SAPS</i> with COMPLETE Dataset	86
5.8	<i>SAPS</i> with Zobel's A Dataset	87
5.9	<i>SAPS</i> with Zobel's B Dataset	88
5.10	<i>SAPS</i> with Zobel's C Dataset	89
5.11	<i>Adaptive SAPS</i> with COMPLETE Dataset	90
5.12	<i>Adaptive SAPS</i> with Zobel's B Dataset	91
5.13	<i>Adaptive SAPS</i> with Zobel's C Dataset	92

5.14 RGP of FONS with COMPLETE Dataset	92
5.15 Performance of Multiple Name Search	93

“The advent of computer technology has brought many kinds of opportunities and some of these, not surprisingly, are of a criminal nature.”

— Bainbridge, 1993

Chapter 1

Introduction

1.1 Backgrounds

The establishment of modern forensic science can date back to 1901. That year marked the official introduction of fingerprinting for criminal identification in England and Wales. Since then, forensic science has been well developed, covering many scientific principles or techniques that can be applied to identifying, recovering, reconstructing, or analyzing evidence during a criminal investigation.

Locard's Exchange Principle, depicted in Fig.1.1, has been cited frequently to explain the role of forensic science in a crime investigation. That is, anyone or anything entering a crime scene takes something of the scene with him and leaves something of himself behind when he departs. With these pieces of evidence, the investigators can demonstrate the strong possibility that the offender was at the crime scene.

The same principle can be applied to the investigation of cybercrime. In this thesis, the term *Cybercrime* is used loosely to refer to any crime that involves computers and networks, including those not relying heavily on computers. The Figure 1.2 from [32]

Figure 1.1: Locard's Exchange Principle

Increasing public influence ↓	Type	Local Law Enforcement Level	National Security Level
	Traffic Violations	Driving under influence (DUI), fatal/personal injury/property damage traffic accident, road rage	-
	Sex Crime	Sexual offenses, sexual assaults, child molesting	Organized prostitution
	Theft	Robbery, burglary, larceny, motor vehicle theft, stolen property	Theft of national secrets or weapon information
	Fraud	Forgery and counterfeiting, frauds, embezzlement, identity deception	Transnational money laundering, identity fraud, transnational financial fraud
	Arson	Arson on buildings, apartments	-
	Gang / drug offenses	Narcotic drug offenses (sales or possession)	Transnational drug trafficking
	Violent Crime	Criminal homicide, armed robbery, aggravated assault, other assaults	Terrorism (bioterrorism, bombing, hijacking, etc.)
	Cyber Crime	Internet frauds, illegal trading, network intrusion/hacking, virus spreading, hate crimes, cyber-piracy, cyber-pornography, cyber-terrorism, theft of confidential information	

Figure 1.2: Crime Types at Different Levels

shows the crime types at different levels, including cybercrime.

The categories of cybercrime has been studied in [94, 95, 44, 27, 96] to define the role of computers in crime and to give guidance for dealing with such computers. The US Federal Guidelines for Searching and Seizing Computers (DOJ) [44] defined a very useful set of cybercrime categories and the corresponding guidelines as,

1. Hardware as contraband or fruits of crime.
2. Hardware as an instrumentality.

3. Hardware as evidence.
4. Information as contraband or fruits of crime.
5. Information as an instrumentality.
6. Information as evidence.

Here *hardware* refers to the physical components of a computer, and *information* refers to any data in digital format stored in or transmitted by a computer or a network.

Each of these categories has unique legal procedures that must be followed [44, 28]. These categories are not mutually exclusive, i.e., a single crime can fall into more than one category and follow the procedures defined by more than one categories. For example, when a computer is instrumental in committing a crime, it is also evidence of the offense.

The last three categories that refer to information all fall into the range of *digital evidence*. Digital evidence is a type of physical evidence, and actually has several unique features over other kinds of physical evidence.

- Digital evidence is easier to duplicate and the copy can be used as if it was the original. It is common to examine a copy, thus avoiding any risk of damaging the original.
- It is easy to determine whether digital evidence has been modified or tampered by comparing with the original.
- Digital evidence is comparably difficult to destroy. It can be recovered from a “delete” operation.

There are several key aspects in the processing of digital evidence.

1. Recognition of digital evidence. The investigators have to recognize the hardware that contains digital information and be able to distinguish between the irrelevant information and the digital evidence.
2. Collecting and preserving hardware and digital evidence. Some specially designed tools and techniques are available to collect and preserve digital evidence properly in a way that will be acceptable in a court of law.
3. Documenting hardware and digital evidence. Documentation is used to demonstrate that the evidence is authentic and unaltered.
4. Classification, comparison and individualization of digital evidence. Classification finds characteristics that can be used to describe the evidence in general terms and distinguish it from similar specimens. Comparison is key when examining digital evidence. In addition to revealing class characteristics, comparing a piece of digital evidence with a control specimen can highlight unique aspects of the digital evidence (individualizing characteristics).
5. Digital evidence reconstruction. Digital evidence that has been damaged can be reconstructed through various processes. And the digital evidence can also be used to help reconstruct events surrounding a crime.

The richest category of all is *Information as Evidence*. As more digital devices and services get involved in our daily lives, there will be more chances that our actions will leave a trail of digits behind. For example, computer stores a lot of personal information of the users; the service providers (telephones companies, ISPs, banks)

keep a lot of information about their customers. All of the above information can be used directly as evidence or can help to track criminals and reconstruct the criminal scenario. *Information as Evidence* is the major interest of our work.

Here some terms used throughout this thesis require clarification: *Digital Evidence*, *Computer Forensics* and *Digital Evidence Investigation*.

Digital evidence The term *digital evidence* encompasses any and all digital data that can establish that a crime has been committed or can provide a link between a crime and its victim or a crime and its perpetrator [28].

Computer Forensics The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations [93]. *Computer Forensics* has been also referred as *Digital Forensic Science* in the literatures.

Digital Evidence Investigation The process of “identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally acceptable via the application of computer technology to the investigation of computer based crime” is called *Digital Evidence Investigation* or *Forensic Computing* [77].

1.2 Motivation

The biggest challenge for computer forensics is how to deal with the human's increasingly complex behavior in the cyberworld. The internet service is so well-developed that it almost replicates all the functions of a real society. The digital records of a person's activities consist of complex communication, relationship and behaviors. To extract and analyze the evidence from such a complex record set is far beyond the ability of current computer forensics tools, which is mainly built on keyword searching and other traditional information retrieval techniques.

The major motivation of this work is to fill the gap between the law enforcement and computer science researchers. When our work started in 2002, little work on computer forensics could be found in the publications of computer science community. After the first several months in collecting all kinds of literature, we were surprised by the fact that the lack of mutual understanding among the different parties involved in computer forensics. The law enforcement engages in the investigation procedures and knows little about the methodologies behind. In the research community, a lot of work has been done under computer security, intrusion detection, cryptology and information systems. Few of them were trying to answer the specific tasks faced by the digital evidence investigators. The same phenomenon has been observed in [53] as, "However, the people called upon to address these technological issues are typically versed exclusively in either law or technology... In the case discussed, the authors were struck repeatedly by the lack of the most basic mutual understanding of the fields of the practitioners involved." Things have changed as the community of computer forensics started to build a unified computer forensic framework, especially after the First Digital Forensics Research Workshop (DFRWS) [93]. A lot of research work has

been done on the framework. But so far, almost all these frameworks were proposed solely from the viewpoint of law enforcement. They well defined the categories of digital evidence and the procedures to prepare, retrieve, capture, analyze and present digital evidence. They established an effective schema to test and compare the devices and software tools for digital evidence investigation. But they failed to supply effective guidance to researchers and failed to build a solid ground to evaluate those cutting-edge technologies which are already widely deployed in many other computer science fields.

To face the challenges in computer forensics, new technologies need to be developed based on the cooperation of the law enforcement and computer science researchers. We believe building an open framework for the whole or part of investigation procedure based on the mutual knowledge of the law enforcement and the researchers will be highly beneficial to computer forensics — it gives explicit definition and target to researchers in computer science, while meeting all the rigid requirements by law enforcement. Thus it brings together the best of both worlds.

Another motivation of this work came from our experience with several prevalent computer forensic softwares, including Encase[112] and Forensic Toolkit [47]. Searching for personal name strings is a very common and important task in computer forensic investigation. The name searching tools in these softwares were built on some aged string search algorithms, such as edit distance and Soundex. As the effectiveness of these methods has been proven poor long ago by the researchers in phonetic string search, it is emergent to make up for the lag. On the other hand, as these algorithms were reported to be inconsistent with different datasets, we believe that they need to be re-evaluated and novel methods need to be developed to meet

the rigid requirements in computer forensics.

1.3 Objectives

The objectives of this work are:

1. To develop a framework that will support the automatic analysis and extraction of digital evidence.
2. Application of the algorithms and architecture for information extraction, criminal pattern identification, scenario reconstruction and knowledge discovery in the computer forensics environment.
3. To investigate the performance of string search algorithms applied to computer forensics with the view of introducing modifications, adaptations and innovations that will optimize the speed and precision of the algorithms.

1.4 Contributions

The contributions of our work are briefly summarized below:

- Computer Forensic Framework

This thesis discusses how to bind computational intelligence with the current computer forensic framework, especially the data analysis phase. A concept, “Case-Relevance”, has been defined to measure the importance of any information to a given case. The proposed framework fully demonstrates the benefits of computational intelligence technologies: automatic evidence extraction, knowledge reusability with great savings on human resource.

- Email Evidence Extraction

An intelligent evidence extraction algorithm from email has been developed. Based on the concept of “Case-Relevance”, the method is fully automated and very effective in extracting and clustering email information according to a given case profile.

- Phonetic String Search

A novel strategy to compare two phonetic strings based on syllable alignment has been proposed. Its major advantage is not only the high efficiency in retrieving phonetic strings, but also the flexibility to achieve optimal performance with different datasets. This is the first of this kind of algorithm as we have learnt so far.

A fast algorithm called *FONS* is designed to run in linear time $O(n)$ and can be extended to search several phonetic strings simultaneously to cut down on the search time further.

1.5 Organization of Thesis

We proceed by introducing and discussing related work in the next chapter. A framework for Case-Relevance information investigation is proposed in Chapter 3. The motivation and the analysis of the email evidence extraction is presented in Chapter 4. Chapter 5 describes our research on phonetic string search in several aspects. Chapter 6 concludes and outlines future work.

Chapter 2

Related Work

This chapter reviews the previous work in computer forensic frameworks, string search and email information processing. String search, also called string matching, pattern search and pattern matching, is widely used in computer forensic systems to search data in text format. Email is a kind of text data providing digital evidence.

The development scope of computer forensic investigation system could be divided into three generations as shown in Fig. 2.1.

Query Based System is built on traditional information retrieval techniques, relying on string search mainly. It aims to look for information relevant to the query terms from a large data collection.

In a query based system, an isolated piece of data offers little help in revealing hidden evidence, relationship analysis and decision-making support. There is no established standard method or routine for conducting a computer forensic examination. Everything depends on the skill and the knowledge of the investigator.

Figure 2.1: Three Generations of Computer Forensic Investigation System

Intelligent Evidence Extraction System Wide use of computational intelligence is the major feature of an Intelligent Evidence Extraction System. With more explicitly established and mature computer forensic investigation frameworks, standardization of investigation methods, tools and routines become possible and widely accepted by the court. With these explicit frameworks, the technologies in computational intelligence can help to build highly automated systems and to relieve the investigators from the heavy burden.

Both the investigators and law enforcement personnel will benefit from such standardized routines as the focus could move from arguing over the integrity of the investigation sequence to the evidences themselves.

Knowledge Management System This is a highly intelligent system where information is organized in a systematic manner so that high level knowledge acquisition, learning, evolution, reuse and sharing are achieved and comprehensive support for

judgement making is available. Such a knowledge management system is the ultimate goal for computer forensics. Many of its features can only be defined completely in the future.

The current computer forensic investigation systems lie somewhere between the first generation and the second generation described above. On one hand, they cope with the existing computer forensic frameworks, offering some automation in certain steps and allowing some limited information reuse and sharing. On the other hand, they still heavily rely on the traditional information retrieval technology.

2.1 Frameworks for Digital Evidence Investigation

A digital evidence investigation process must follow the “basis in existing physical crime scene investigation theory” [26]. As “almost every individual stream of digital evidence could be challenged” [114], computer forensic investigators are required to follow a rigorous process path. The work of the First Digital Forensics Research Workshop (DFRWS) [93] established a solid ground and allowed the experts and researchers from governments, law enforcement and other third parties to work together under an unified schema. Some updated work can be found in [101, 26, 80, 20, 19, 98].

A basic single tier framework (Fig. 2.2) was depicted in [20], which consists of six phases, and each phase may include several sub-phases. Phases and sub-phases are “distinct, discrete steps in the process that are usually a function of time and suggest a necessarily sequential approach” [20]. Although the phases are sequential and non-iterative in the framework, the actual investigation process is highly iterative. That is, the findings in a certain phase may be fed back to the previous phases for continuous refinement.

Figure 2.2: Single Tier Digital Investigations Process Framework [20]

Figure 2.3: Data Analysis Phase [20]

Arguably, the Data Analysis Phase is the most complex phase in the whole investigation process and draws intensive attention from the researchers. In [93, 101], it was divided into two separate phases as “examination” and “analysis”. [80] used two phases, “the live system processing and data collection” and “the analysis of secured data”, to describe the operation flow of the overall data analysis part in a network forensic environment. In [26], the data analysis part was described as a three first-tier phases, including “Survey Phase”, “Search and Collection Phase” and “Reconstruction Phase”. In this thesis, we use the three sub-phases model proposed by [20] as “Survey Sub-Phase”, “Extract Sub-Phase” and “Examine Sub-Phase” shown in Fig. 2.3. Although the exact definition and boundary of the sub-phases differ here and there in the above works, the objectives of the overall Data Analysis Phase is clear, that is, to exam, search and extract the data collected in Data Collection Phase,

to supply sufficient information for the crime scenario reconstruction and suspected activity confirmation.

2.2 String Search

String search, sometimes called string matching, is to find the position where one or several strings (also called patterns) are found within a larger string or text.

In the real world, the exact string matching algorithms are useless, when the text collections are becoming larger, more heterogeneous, and more error prone. [67] found that a non-negligible percentage of errors (7-16%) might be introduced by optical character recognition (OCR), 1-3.2% by typing and 1.5-2.5% by spelling. Experiments for typing Dutch surnames (by the Dutch) reached 38% of spelling errors. This is where approximate string search finds its place.

Approximate string searching is to find the positions where a given string occurs in a text document, allowing a limited amount of “errors” in the matches. Different applications may use different error models, which define how different two strings are.

One of the best-studied cases of this error model is the so-called *edit distance*, which allows *deletion*, *insertion* and *substitution* of single characters in both strings. The edit distance $d(x, y)$ between two strings x and y is the minimal cost of a sequence of operations that transform x into y . There are many different distance functions, and some most commonly used ones are listed here:

- Levenshtein or edit distance [71]: allows insertion, deletion and substitution. In the simplified definition, all the operations have a cost of 1.

- Hamming distance [106]: allows only substitution, with cost 1 in the simplified definition.
- Episode distance [37]: allows only insertion, with cost 1.
- Longest common subsequence distance [89, 7]: allows only insertion and deletion, both with a cost of 1.

One of the most common criteria to categorize string searching algorithms is whether the text is preprocessed or not. Thus the string searching algorithms can be divided into two categories: online string searching algorithm and index string search algorithm. Index search algorithm is based on preprocessing of the text. After building an index, for example a suffix tree or suffix array, it can find pattern quickly, using a binary search of the index. In the following review, we focus on online string search because online string search is more applicable to computer forensics, in which the cost of indexing is too high since the typical size of target data is huge (e.g., the text data in a whole hard disk).

2.2.1 Approximate String Search

Most of the existing online algorithms can be divided into four major categories: dynamic programming, automata, bit-parallelism and filtering.

Dynamic Programming Dynamic programming is a very old technique, and was first converted into a search algorithm in 1980 by Sellers [109]. Although the algorithm is not very efficient, it is among the most flexible ones in adapting to different distance functions. Given a string of length m and a text of length n , the time cost is $O(mn)$ in

the worst and average case. The space required is $O(\min(m, n))$ as only the previous column or row need to be stored in column-wise or row-wise processing.

In 1980, Masek and Paterson [76] proposed an algorithm base on the Four-Russians technique. Its worst case cost is $O(mn/\log_\sigma^2 n)$ and requires $O(n)$ extra space. The algorithm is only of theoretical interest, since it is not competitive for texts below 40 GB size and the huge extra space is unaffordable.

In 1983, Ukkonen [120] presented the first so-called “diagonal transition algorithm”. It is base on the fact that the diagonals of the dynamic programming matrix (running from the upper-left to the lower-right cells) increase monotonically. The algorithm computes only $O(k^2)$ positions along the diagonals. It can compute the edit distance in $O(ed(x, y)^2)$ time, or to check whether that distance was k or not in $O(k^2)$ time.

In 1985, Ukkonen developed another algorithm, called “cut-off heuristic”, to improve dynamic programming algorithm to $O(kn)$ average time and $O(m)$ space [121]. The main idea is that, since a string does not normally match in the text, the values at each column (from top to bottom) quickly reach $k + 1$ (i.e. mismatch), and that if a cell has a value larger than $k + 1$, the result of the search does not depend on its exact value.

In 1985, Landau and Vishkin [70] showed an algorithm which was $O(k^2n)$ time and $O(m)$ space, and in 1986 they obtained $O(kn)$ time and $O(n)$ space. The main idea is to adapt the Ukkonen’s diagonal transition algorithm for edit distance to text searching. The dynamic programming matrix is computed diagonal-wise instead of column-wise. But the algorithm is quite slow in practice because it is not easy to find the last common node in the suffix tree in constant time.

In 1986, Myers developed an algorithm with $O(kn)$ worst-case time and $O(n)$ extra Space [85, 86]. The algorithm shares the idea of computing the k new strokes using the previous ones, as well as the use of a suffix tree on the text for the LCA algorithm.

In 1992, Chang and Lampe [29] produced a new algorithm called “column partitioning” with an average search time of $O(kn/\sqrt{\sigma})$. This is the fastest dynamic programming algorithm in practice. But it is difficult to adapt to other distance functions, since it strongly relies on the unitary costs.

In 1998, Cole and Hariharan [35] presented an algorithm with worst case time cost of $O(n(1 + k^c/m))$, where $c = 3$ if the string is “mostly aperiodic” and $c = 4$ otherwise. The idea is that, unless a string has a lot of self-repetition, only a few diagonals of a diagonal transition algorithm need to be computed.

Algorithms Based on Automata The algorithms based on automata can achieve the best worst case time $O(n)$. However its time and space costs have exponential dependence on m and k , thus limiting its practicality.

In 1985 Ukkonen developed the first automaton in deterministic form with $O(n)$ worst case time[121]. However, the main problem becomes the construction of the deterministic finite automaton (DFA) and the cost is quite high.

An alternative solution is based on simulating the nondeterministic automaton (NFA) instead of making it deterministic. Its first implementation was developed by Wu and Manber [132]. The work in [78] added the operation of *transposition*. Then Baeza-Yates and Navarro reconsidered this problem and gave an explicit explanation in [9, 11, 15].

Bit-Parallelism Bit-parallelism is a general way to simulate simple nondeterministic automata. It has the advantage of being much simpler, faster, easier to extend in handling complex strings, and effective in any error rate. Its main disadvantage is the limitation it imposes on the size of the computer word. When coping with longer strings, it is not very efficient.

The idea was firstly introduced in the Ph.D. thesis of Baeza-Yates in 1989 [8], and was implemented to deal with common string matching [9, 10]. It takes advantage of the intrinsic parallelism of the bit operations inside a computer word and can cut down the number of operations that an algorithm performs by a factor of at most w , where w is the word width (the number of bits in a computer word). Since the word width in current computer architectures is 32 or 64, the speedup is very significant in practice, especially when short strings are involved (typical in text retrieval).

The first bit-parallel algorithm, Shift-Or was proposed by Baeza-Yates and Gonnet in 1992 [12], and has become the basis of many following works. The algorithm searches a string in a text (without errors) by parallelizing the operation of a nondeterministic finite automaton (NFA) in $O(mn/w)$ worst-case time (the original time complexity is $O(mn)$). For strings longer than the computer word width (i.e. $m > w$), the algorithm uses $\lceil m/w \rceil$ computer words for the simulation. It is easy to extend Shift-Or to handle classes of characters and search for multiple strings.

1992, Wu and Manber [132] extended the Shift-Or algorithm to handle wild cards and regular expressions. The cost of this simulation is $O(k \lceil m/w \rceil \cdot n)$ in the worst and average case. This is a perfect speedup over the serial simulation of the automaton, which would cost $O(kmn)$. The algorithm was used in one of the most successful searching softwares – Agrep [131].

In 1996, Baeza-Yates and Navarro presented a new bit-parallel algorithm able to parallelize the computation of the automaton even more [15]. The classical dynamic programming algorithm can be thought of as a column-wise “parallelization” of the automaton [11], and Wu and Manber [132] proposed a row-wise parallelization. In [15] the bit-parallel formula for a diagonal parallelization was found. In general, it has $O(\lceil k(m - k)/w \rceil \cdot n)$ worst-case time, and $O(|k^2/w| \cdot n)$ on average when using Ukkonen cut-off heuristic. The algorithm is $O(n)$ worst case time and very fast in practice if all the bits of the automaton fit into one computer word. The scheme can handle classes of characters, wild cards and different integral costs in the edit operations.

Filtering Algorithms Filtering algorithms [118, 62, 132, 15, 30, 122, 117, 31, 116, 111, 51] is to filter the text, quickly discarding text areas that do not match. Filtering algorithms are the fastest ones in practice for they need not inspect every character in a text. The major theoretical achievement is $O(n(k + \log_\sigma m)/m)$ average cost. Any filtering algorithm must be coupled with a process that verifies all the matches and report their positions. Virtually any non-filtering algorithm can be used for the verification.

The performance of filtering algorithms is very sensitive to the error rate α . Most filters work very well on low error levels and very badly otherwise.

2.2.2 Multiple String Search

Limited work has been done in multiple string search [14, 84, 65, 68, 87, 133]. Most of them are multiple extension of single string searching algorithms.

As already noted in [12], the ability of Bit-parallelism algorithms to deal with

character classes can be used to build multiple string searching algorithm [14, 68]. Assume that all the strings have the same length (otherwise, truncate them to the shortest one). Hence, all the automata have the same structure, differing only in the labels of the horizontal arrows. We can build the NFA for each one, and then “superimpose” all the NFAs.

2.2.3 Phonetic String Search

Phonetic string search of written text is used to identify and retrieve strings that may have similar pronunciation, regardless of their actual spelling. Phonetic string search is useful in Cross-Lingual/Multilingual Information Retrieval systems, Machine Translation, Transliteration, Back-Transliteration, Spell Check and especially in solving the name string searching problem. Actually, the term *Phonetic String Search* has been used with *Name Searching* or *Name String Search* alternatively in the literature. The main application of phonetic string search has been in determining whether a name in a query matches some entities in a name database, and has a wide application potential in the fields of data management of census, migration documents, Yellow Pages and Legal Cases. For example, reported by [119], 67.8%, 83.4%, and 38.8% of queries to Wall Street Journal, Los Angeles Times, and Washington Post involved name searching.

Phonetic string search is especially useful in Computer Forensics. One reason is that personal names and addresses are the most typical search target in an investigation. Another reason comes from the criminal identity deception. Criminals often provide police officers with deceptive identities to mislead police investigations, for

Table 2.1: Comparison between Soundex, Jaro’s method and Edit Distance(ED) [126]

A PAIR OF STRINGS		SOUNDEX	JARO’S	ED
JONES	JOHNSONS	0.75	0.79	0.50
MASSRY	MASSIE	1.00	0.889	0.66
SEAN	SUSAN	0.50	0.783	0.60
HARDIN	MARTINEZ	0.75	0.00	0.50
JON	JAN	1.00	0.00	0.66

example, using aliases, fabricated birth dates or addresses, etc. A case study on deceptive criminal identities records [126] found 42% of the criminals used a deceptive name having *the same or similar pronunciation*, but *spelt differently*. For example, “Cecirio” can be altered to “Cicero”. In this situation, it has been discovered in [97, 136] that phonetic string search is much more suitable than the traditional approximate string search.

Edit Distance (Levenshtein Distance), Soundex and their variants are widely used in today’s computer forensic tools to search for personal names and addresses. Because of the importance of the data they processing, it is worthwhile to have a closer look at the effectiveness of these methods. Table 2.1 from [126] shows how the results differ among different methods and a potential risk exists when an unsuitable algorithm is deployed. The table examines some name pairs using Soundex, Jaro’s method [61] and Edit Distance. The numbers are the similarity values measured by these algorithms, and higher is better.

The early work in phonetic string search dates back to early 1980’s [54]. In the following twenty years, researchers continued working on it to improve precision and recall. Except for the traditional pattern matching algorithms, such as Edit Distance, Agrep, and Q-gram, and the unsuccessful Skeleton-Key and Omission-Key methods

in [99], the different techniques developed so far in this field can be divided roughly into the following four categories:

1. *Phonetic Transformation Rules*. The algorithms in this category use predetermined transformation rules to convert a string into phonetic codes for later comparison and are based on two assumptions: (1) Consonants carry more phonetic information than vowels or diphthongs; (2) A character always has the same/similar pronunciation. Nearly all the phonetic string search algorithms belong to this category.

Soundex [54] is the best-known algorithm in this category. It uses codes based on the sound of each letter to translate a string into a canonical form of at most four characters, while preserving the first letter. Soundex is simple to deploy and fast, but is error-prone. For example, Soundex makes the error of transforming dissimilar-sounding strings, such as “Pierce” (P620) and “Price” (P620), into the same code, and of transforming similar-sounding strings into different codes, such as “Cathy” (C300) and “Kathy” (K300).

Phonix [48, 49] is a popular Soundex variant. Its major development is that it uses about 160 letter-group transformations to standardize the string before the code translation. These transformations provide context for the phonetic coding and allow, for example, ‘c’ and ‘s’ to be distinguished. There are several variants of the Phonix algorithm, such as Phonix4, Phonix8 and PhonixE, which are only different in code lengths and components.

In [57], more deliberate considerations have been given to the operations of consonant removal, vowel removal, leading characters and ending sounds in the algorithm called Celko. Another important aspect of Celko is that it produces multiple phonetic codes for every string.

In [134, 56], necessary spelling-to-sound information is extracted from a dictionary, and is used to produce the most likely phonetic codes for letters or sub strings.

Phonometric [135] uses a complex string-to-pronunciation conversion algorithm to convert a string into phonemes for later comparison. Thus, given a reliable string-to-pronunciation algorithm, we would expect Phonometric to give the best phonetic matching.

Although most transformation rules are designed for English or western European languages, they are quite flexible. By re-configuring the phonetic letter groups and the transformation rules, these algorithms can be easily transplanted to other languages using the Roman alphabet, e.g. Swedish [42].

Soundex assumes that one phonetic transformation rule works well under all situations. This over-simplification makes the overall performance of Soundex and Phonix very poor although they could fetch some good matches that are missed by other algorithms. As observed in [57, 134, 136], although these algorithms perform better by including more deliberate phonetic transformation rules, the improvements are not proportional to the added complexities.

2. *Adding Similarity Metric to Soundex.* One major weakness in Soundex is that there is no similarity metric for assessing the closeness of two strings - strings are either similar or not similar. Editex [136] adds an edit distance metric with a redefined operation function to the phonetic letter-grouping strategy used by Soundex and Phonix. In Editex, the edit operation on two characters returns 0 if the characters are identical, 1 if they both occur in the same group, and 2 otherwise. Editex is easy to deploy and its performance is quite competitive and consistent with different datasets.

3. *Assigning Different Weights to Different Positions.* It is evident that the letters in different positions should have different weights in the human judgement of phonetic string similarity, but few researchers have made use of this factor in their algorithms. An exception is Tapering [136]. It considers the human-factors property that the differences at the beginning of a string are more significant than differences at the end. In a tapered edit distance method, the maximum penalty for replacement or deletion at beginning of a string just exceeds twice the minimum penalty for replacement or deletion at the end of the string. Such an edit distance computation is effective in breaking ties: strings with the same errors are ranked according to the positions in which the errors occur. However, its experimental result was disappointing.

4. *Combination of Evidence.* It has been discovered that even the most successful methods fetch rather different sets of answers, sometimes almost without overlap [134, 136]. In Information Retrieval (IR), combining the ranked results produced by different retrieval mechanisms can improve the overall performance effectively. The same logic can be applied to phonetic matching, that is, the combination of evidence can lead to a remarkable improvement [57, 136, 97].

2.3 Email Information Processing

Email has become the most important and widely used communication tool in the modern world. According to [129], over 4 billion corporate email messages were exchanged per day in 2001, increasing to a projected 35 billion in 2005. Moreover, 97% of workers reported using email multiple times per week for a daily average of 49 minutes, and 71% of people stated that it was “essential” for their everyday work

[23, 75, 50]. Besides its success, email overload is becoming a critical problem [128]. To address the problem, much work has been done in anti-spam email, email filtering, email categorization and email task management.

Email Classification Email classification has been widely used on spam detection and automated foldering. The majority of current systems make use of text classification and machine learning techniques such as TF-IDF, Naïve Bayes, rule-based system and support vector machine. SwiftFile [108] and MailCat [107] used AIM text classifier [18], which is a TF-IDF style classifier [105] to score high classification accuracy on the task of classifying e-mail into folders in Lotus Notes. William Cohen tested TF-IDF and a rule-based classifier on a corpus of his own e-mail [34]. To simplify the training and computing cost, [79] used a query that was performed across all mail messages based upon the items which the user has placed into the category and showed the results as a relevance-ranked list.

Email classification has the potential to eliminate many of the problems of current mail filters. Studies showed text classification and machine learning algorithms achieving an accuracy of 80% or better, even when the classification task was across a wide range of classes.

Email Thread Detection Email systems is widely used to carry out conversation and group discussion. Research has revealed that threading ranges from 52% [123, 22] to 30% [43] in ordinary email. The current email tool features are lagging behind the requirements and seriously reduces individual and corporate productivity [22, 16, 17, 125, 123, 128, 127].

Before the prevalence of email, online messaging systems, such as the Usenet

```

From: Jone Smith < JoneS@ntu.edu.sg >
Subject: Re: Where is the first lecture?
Date: Sun, 23 Apr 2006 12:41:12 +0800
Lines: 11
Message-ID: <cglafa$f3o$1@dcsl-news1.ntu.edu.sg>
References: <cgl09c$b1l$1@dcsl-news1.ntu.edu.sg>
In-Reply-To: <cgl09c$b1l$1@dcsl-news1.ntu.edu.sg>

Tom Blake wrote:

> Where is the first lecture?
>
> Thanks
>
> Tom

The first lecture was in Hall 11, during the normally scheduled
time.

Jone

```

Figure 2.4: Email Thread Information

network and other bulletin systems, provide most support for threading. For example, in Usenet, the chain of unique message identifiers in the *Reference* field of the parent is copied in to the *Reference* field of the child, with the unique identifier of the parent added. And the *Subject* line of the parent is copied into the *Subject* line of the child, adding a prefix *Re:* typically. Some Usenet news readers use the structural links from the *Reference* field to organize and display a thread. The other provide a threaded display around *subject* line which are identical or have the same prefix.

The organization of thread in email systems can use the same methods as long as the system inserts some kind of thread structural information into the reply. For example, in the online email system [103], the reply command copied the Message-ID field into the In-Reply-To field of the child, and copied the Subject line into the Subject field of the child, prefixed by *Re:* (as shown in Fig. 2.4). Email systems try to support thread since early time. For example, the VM [63] mail reader allowed to group messages by the same subject line text, the same author or the same recipient. The hypermail [59] archived emails by matching the message ID in the In-Reply-To

field, the date string in the In-Reply-To field, and the Subject line.

However, in current email systems, the use of Message-ID and In-Reply-To fields are optional and subject lines may contain arbitrary texts. Moreover the use of *Re:* tag is inconsistent and the manual editing of Subject lines is very common in these systems. Due to these reasons, the error rates of the above system are considerable. [72] figured out that pushing tighter standards for specifying threads is not applicable. They used a TF-IDF classifier to match e-mail messages with their “parents” instead by Subject line as well as the message content. Thus email thread detection returns to the Email classification.

Email Thread Summarization Email thread summarization is to extract and re-organize the key information from email threads (not single emails as in [83]) and allow a fast view for the users. Email thread summarization is different from summarizing other types of written communications as it has an inherent dialog.

Work by [90] generated extractive summaries for the documents at the first two levels of the discussion, and allowed the user to decide which threads to read without browsing the actual content of the thread. Their approach works best when the subject of the root email best describes the issue of the thread and the root email does not discuss more than one issue. [91] addressed the problem of summarizing archived discussion. The messages were clustered into topic groups, and the summaries were extracted for each topic group using a scoring metric based on sentence position, lexical similarity of a sentence to cluster centroid and a feature base on quotation. [100] used the sentence extraction technique and added certain email-specific features in identifying relevant sentences for extraction.

Email-based Activity Management Email-based Activity Management (also called Email Task Management) tries to capture not only the threads of messages but also the threads of activities. It differs from collation systems [60] that track threads as it goes beyond system-defined threads to encompass user-defined task-centric collections. The work in [58, 115] have attempted to discover activities in a user's inbox using clustering technology.

Most of activity-centric systems [82, 36, 38, 81] focused on the cross application nature of activities, gathering together an activity's emails, people, and all the other related information into a single view. [69] examined email activities that represented structured business processes and reverse-engineered the structure of processes by learning them from representative email transactions. [64] considered the problem of automatically learning an activity's structure. They formalized activities as finite-state automata and inferred relationships among messages in less-structured activities. [39] did a study on the problem of classifying emails into activities, in order to automatically populate activities with the emails related to them. Their method compared the people involved in an activity against the recipients of each incoming message and used Iterative Residual Rescaling (IRR) method [6] to classify emails into activities using similarity based on message contents.

At the same time, email has become the primary communications mechanism in collaborative work [128, 40, 129], and some work have explored activity-oriented systems that help users manage their workload more effectively. The Taskmaster system [40, 41] enhanced an email client to function as a task management system. [22] allowed the user to split a thread into multiple *thrasks* and combine *technically unrelated threads*. The Activity Explorer system [82] enabled people to collaborate

around shared artifacts, but did not integrate collaboration with email.

Email Datasets The common Email databases from real data are not easy to obtain because of the privacy concerns. Before the Enron Email Dataset [2], the lack of common data set to evaluate the performance of email classification is a big problem for the researchers. Many previous researches used personal email data or corpus from newsgroups, for example 20 Newsgroups[1]. The work in this thesis also used a dataset collected from personal email.

The Enron Email dataset was made public during the legal investigation concerning the Enron corporation. The raw Enron corpus contains 619,446 messages belonging to 158 users. The dataset still had a lot of integrity issues. It had many duplicate and corrupt messages. The work of [66, 110] cleaned and corrected the dataset, and some work has been done to produce some useful subsets [21, 4]. For example, in the cleaned Enron corpus by [66], there are a total of 200,399 messages belonging to 158 users with an average of 757 messages per user.

With the large number of users, messages, threads and folders, the Enron Email dataset is a suitable corpus for evaluation of email classification methods. The research illustrated that the Enron dataset is consistent with many of the assumptions made about email classification. The research on Enron dataset also helped to reveal many Email features. For example, it was found that out of the total 200,399 messages, 30,091 threads were detected in the corpus, consisting of 123,501 emails [66]. In other words, 61.63% of emails in the corpus are in threads.

For computer forensic research, Enron Email dataset supplies a suitable corpus for understanding the behavior of email communication in a large business company. Besides Email processing, it can also be used for research in link analysis and social

CHAPTER 2. RELATED WORK

31

network analysis, which are very important in Computer Forensics. The work done in this thesis used a personal Email dataset. We plan to apply and test our methods on Enron Email dataset in the future.

Chapter 3

Case-Relevance Computer Forensic Framework

As indicated in the previous chapters that one of the biggest challenges for computer forensics is how to deal with the human's increasingly complex behavior in the cyberworld. The digital records of a person's activities consist of all kinds of communication, relationship and behaviors about the person. To extract and analyze the evidence from such a complex record set is far beyond the ability of current computer forensics tools, which are mainly built on keyword searching and information retrieval techniques. To solve this problem, we believe that computational intelligence technology should be introduced to build a knowledge level evidence extraction and analysis model for computer forensics.

Computational intelligence is "a branch of artificial intelligence. As an alternative to GOFAI it rather relies on heuristic algorithms such as in fuzzy systems, neural networks and evolutionary computation. In addition, computational intelligence also

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 33

embraces techniques that use Swarm intelligence, Fractals and Chaos Theory, Artificial immune systems, Wavelets, etc.” [130]

In this thesis we define a very important concept, “Case-Relevance”, and discuss how to understand the unique features and requirements of computer forensic systems based on the concept. We also build two typical investigation scenarios to see how to bind the new concept of “Case-Relevance” with the current framework of digital evidence investigation and how it benefits the current investigation procedure with higher automation, effectiveness and better knowledge reuse.

3.1 Motivation

The current computer forensic investigation system lies somewhere between the first generation and the second generation. To design our own method, we must have a close look at the actual methods used in existing systems.

Today’s computer forensic investigators make use of some computer software [45] to carry out their tasks more effectively and efficiently. Some multi-functional systems, such as Encase[112], Forensic Toolkit [47] and Vogon [113] Forensic Software, offer an integrated environment for data capturing, imaging, searching, filtering and analyzing. For example, eScript, a simple script language used in Encase, allows the investigators to write their own programs for customized data searching and filtering or perform a sequence of search operations. Users can download, share and exchange the script codes, which bring great flexibility, reusability, automation and efficiency to the investigation process.

But by taking an in depth look into all these tools and systems, we find that they are mainly built on the aged keyword searching tools — the computational intelligence

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 34

Figure 3.1: Searching/Extraction Loop

techniques prevalent in many other modern information systems play very little role so far. Consider a typical evidence searching procedure: the first round search starting with some initial keywords is quite loose and usually returns dozens to hundreds of hits. An investigator checks the returned hits, gets rid of the irrelevant items and refines the query terms if necessary. Once a piece of evidence is found, the electronic file or the stream containing the evidence will be analyzed immediately to bring other clues to the light. These new findings could be used to build a new keyword list for the next round of search. So the whole procedure is bootstrapping and highly iterative, trying to capture as much evidence as possible for the scenario reconstruction later, as shown in Fig. 3.1. As all the information search, extraction and analysis work is still left to the human, the data analysis phase is tedious, time-consuming and requires significant expertise on the part of the investigator.

Another problem is *knowledge reuse*. That is, the selection of the initial keywords need extra background knowledge. An experienced investigator will maintain a collection of keyword lists from his previous cases. For a new case, he could build the keyword list based on the collection or even re-uses the one from a similar case

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 35

directly. Some software such as Encase allows the user to import and export the keyword list for a quick startup. The keyword lists may be shared and distributed within a restricted community. In this way, it can be considered as some kind of primary information reuse, but far from enough. How could someone in his first case learn to search not only the word “bomb” but also “primer” and other jargon at the same time? What we expect is a systematic mechanism for knowledge collection, management, sharing and reuse, offering decision support for the investigators.

To solve the problems mentioned above, we believe that computational intelligence technology *should* and *could* play an active role in the data analysis phase of computer forensics.

3.2 “Case-Relevance”

Definition. *Case-Relevance:* *the property of any piece of information, which is used to measure its importance to reconstruct the criminal scenario for a given criminal case. Criminal case refers to a lawsuit brought by a prosecutor employed by the federal, state or local government that charges a person with the commission of a crime.*

From the definition of “Case-Relevance”, we can assign degrees to “Case-Relevance”. *Absolutely irrelevant* refers to definitely no signal of the criminal. *Provably Case-Relevant* means that the information is undoubtedly critical to the criminal scenario reconstruction. Actually the degree of Case-Relevance covers a continuous spectrum from absolutely irrelevant to provably Case-Relevant, rather than simply relevant or irrelevant. But for convenient general discussions, it can be defined as a discrete set

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 36

Figure 3.2: Degrees of Case-Relevance

of degrees. In Fig. 3.2, we use *possible* and *probable* to describe the increasing levels of Case-Relevance or irrelevance.

Deploying the concept of “Case-Relevance” in the existing computer forensic framework will build a Case-Relevance computer forensic framework. In this framework, the evidence collection, identification and analysis can all be guided and measured by the “Case-Relevance” of the information involved. In the following section, It will be discussed how to deploy the concept of “Case-Relevance” into the current data analysis phase framework to deal with two common scenarios in digital evidence investigation.

Currently many general purposed computer security tools are used in computer forensic investigation. Despite the fact that computer forensics and computer security share a lot of tools and knowledge, there are significant differences between them, which have been discussed intensively in the literature. Part of them are summarized in Table. 3.1 [24]. The concept of “Case-Relevance” is designed to answer the special requirements of computer forensic system.

Firstly, computer forensics is strictly case specific. While computer security looks out for any possible harmful behaviors, computer forensic investigators are only interested in a very narrow scope — information that could be used to reconstruct the criminal scenario. All other data, even an obvious alarm information for computer

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 37

Table 3.1: Key Distinctions between Computer Security and Forensic Computing [24]

Security	Forensic Computing
Protects the system against attack	Does not protect the system against attack
Usually in real time	Post mortem
Conducted by computer specialists	Can be conducted by computer specialists, but often this not the case
Restricted environments for presentation of developments, issues	Evidence is nearly always presented to non-IT/IS personnel
Can be bypassed by trusted individuals/users	Integrity of the evidence is most important

security, will be considered as “noise” for computer forensics. In this situation, the concept of “Case-Relevance” is an effective criterion to filter the noise data and locate the useful ones effectively.

Secondly, every computer forensic investigation starts with, more or less, “a preliminary analysis of the crime notification (notitia criminis) which leads to the formulation of some initial hypotheses that drive the evidence discovery process” [25]. The initial information can be used to build the “Case-Relevance” judgement and give an explicit direction for the later steps. On the other hand, the computer security must be prepared for any possibilities as their enemies are “willing to invest a lot more time looking for weaknesses to exploit than most of us are willing to spend implementing good security” [92].

Thirdly, computer forensics faces a lot of time pressure. The investigators must be able to prioritize the evidences according to their importance to the criminal scenario reconstruction. The degree of Case-Relevance also provides a framework for analyzing

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 38

cost versus exhaustive search.

Last but not the least, the ultimate goal of an ideal computer forensic system is to provide direct information for the criminal scenario reconstruction. Normally, the results coming from computer security systems are only intermediate data for computer forensics. For example, the intrusion traces caught by an intrusion detection system still need to be passed to a computer forensic system to measure its relationship with the case.

3.3 Case-Relevance Information Investigation

3.3.1 Case-Relevance Information Extraction

This is the most common scenario in digital evidence investigation –an exhaustive keyword searching procedure to capture relevant information as much as possible. An abstract flowchart is shown in Fig. 3.3. Following the current Data Analysis Phase framework, the flowchart is also divided into three sub-phases.

1. *Survey Sub-Phase.* Firstly, an experienced investigator studies the initial case information carefully to work out a Case Profile. Some intuitive user interfaces may be used to offer some kinds of assistance. The Case Profile will be sent to an Expert System, behind which are a case database consisting all or selected previous case records and a complicated rule-based system.

The keywords produced will be used for the first round search in the Extraction Sub-Phase. That is why they are called “Seed Keywords”.

2. *Extraction Sub-Phase.* The Automatic Evidence Extraction Module is a highly iterative bootstrapping procedure. It starts from the few Seed Keywords and

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 39

Figure 3.3: Case-Relevance Information Extraction

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 40

will dig more and more Case-Relevance information in the repeated procedures until it is believed that all evidences have been extracted (a threshold value will be met).

It is a fully automatic procedure, thus no human intervention is required. The detailed function blocks will be explained in the later part of this subsection.

3. *Examine Sub-Phase.* The extracted information in the previous step will be examined by a veteran investigator. The investigator can confirm or reject the findings, or return to the previous Automatic Evidence Extraction Module for a refined search. The refined search could be achieved by automatic query expansion from the confirmed and rejected findings, or even by manually edit the extraction rules. The finally confirmed result will be organized and documented for the later reporting step, and it will also be added to the Case Database and the Expert System with other necessary case information. Thus the knowledge is accumulated and developed.

Automatic Evidence Extraction Module. The Automatic Evidence Extraction Module is the core in the above Case-Relevance Information Extraction scenario. Here we propose a hybrid architecture of Information Retrieval (IR), Information Extraction (IE) and computational intelligence function blocks. The diagram is shown in Fig. 3.4, and the different blocks are distinguished by the different legends.

1. *Information Retrieval Block.* Firstly, the Seed Keywords are sent to the Query Expansion&Refinement Module, which is borrowed from traditional IR techniques. A predetermined thesaurus list is used to automatically add synonyms to the original query. A concept-based thesaurus categorizes the query terms

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 41

Figure 3.4: Automatic Evidence Extraction Module

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 42

and their synonyms by their semantic concept. It can control the number of query terms and improve the precision of information retrieved effectively.

Secondly, the query terms are sent to a Multi-Pattern Searching Engine. The previous expansion step may increase the number of keywords dramatically and thus add to the payload of the searching engine. A fast multiple patterns searching algorithm is preferred, and hardware-base architecture attracts enough attention by its intrinsic parallelism and speed advantage.

2. *Case-Relevance Judgement Block.* Computational intelligence is used in the Case-Relevance Judgement module to scan the target data, make decision on the given Case Profile and return case relevant data only. The data is ranked according to their degrees of “Case-Relevance” as defined in Sect. 3.2.

3. *Information Extraction Block.* The IE block can be divided into two parts: Template Building and Keyword Extraction.

Template is a set of predefined slots for extracting specific pieces of information from unstructured or semi-structured textual documents. In Template Building, the event templates are automatically created based on the given Case Profile, and will be used in Keyword Extraction.

Keyword Extraction is performed by three functional modules. The first one is a semantic-level document filter, which permits the identification of relationships rather than the purely Boolean. Hence, topics and documents can be matched not only by whether the specified keywords occur in both, but also by whether they occur in the same (or similar) relationship in both topic and document. The non-relevant documents are abandoned even they have the same keywords. The second one is the keyword extraction module, which fills the Event Templates

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 43

with the key information that may be used as new query terms, e.g. personal name, time and locations. The third module is a semantic tagger. It is based on the semantic-level analysis to disambiguate the concepts of the words or phrases in the given context and add corresponding tags to the selected keywords. Then the new keywords list will be sent to the IR block to start the next round of search. The added tags can be processed by the Concept-based thesaurus to produce more accurate query terms.

3.3.2 Case-Relevance Information Confirmation

It is another common situation in a real case that the investigator is asked to confirm or reject one activity. Instead of searching and bringing everything involved to the court, the investigator is expected to do a comparably narrow and highly target-oriented examination on the captured data. The three sub-phases working flowchart is shown in Fig. 3.5, in which an Activity-Relevant Judgement module is used to scan the target data and return highly selective information for the activity confirmation only.

3.4 Summary

As computers are increasingly involved in crime, computer forensics will play a greater role in criminal investigation.

The work done in this chapter is trying to answer the problems we raise. To relieve the investigators from the tedious and iterative procedures and for a better knowledge reuse and sharing, computational intelligence is expected to play a more active role in

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 44

Figure 3.5: Case-Relevance Information Confirmation

CHAPTER 3. CASE-RELEVANCE COMPUTER FORENSIC FRAMEWORK 45

Computer Forensics. Towards this goal, a novel concept, “Case-Relevance”, is defined to measure the importance of any information to a given criminal scenario reconstruction. And how to build an intelligence data analysis phase around “Case-Relevance” is demonstrated through the detailed discussion of the “case-relevance information extraction” and “case-relevance information confirmation” scenarios. The study fully shows the possibility and the potential of binding computational intelligence with current Computer Forensic framework.

Chapter 4

Email Analysis for Computer Forensics

Email is an important topic for computer forensics for the following reasons,

1. Email has become the primary communication mechanism, thus having a high probability to carry and store important computer forensic information.
2. Email has many unique characteristics: it is less-structured; its grammar is quite loose with the use of abbreviations and phrases; it facilitates collaborative tasks and should not be understood and processed individually. All of these require some deliberate thinking besides the general text processing technologies.
3. Many email information processing technologies have been developed for years in information processing field. Making use of these technologies will bring direct benefits to computer forensics. On the other hand, the special requirements and characteristics of computer forensics will raise new demands and promote the research in information processing field.

This chapter developed a novel email evidence extraction method. By deploying the “Case-Relevance” information investigation framework, the method is very effective in extracting relevant information from a email corpus for a given case profile.

The method consists of two major parts: an information extraction (IE) module and an email classification module. The IE module is used to extract information from email messages with a predefined IE template. The template is determined by the initial knowledge of a given case. Then the extracted information is sent to the email classification module to calculate the degree of “Case-Relevance” of the messages. The email classification module classifies email messages not only by their statistic features but also by their degree of “Case-Relevance” (Sect. 3.2) to the given case. In this way, the method extracts highly selective information from the email corpus as evidence for the computer forensic investigators.

From the definition of computer forensic framework, the above procedure belongs to extraction sub-phase (Fig. 2.3). That is why the method is named as “Case-Relevance” Email Evidence Extraction.

4.1 Motivation

As we have reviewed in section 2.3 that many technologies have been developed to manage the large volume of personal email corpus, they can be a great help for computer forensic investigation. A close look at their potential and limitations will help to understand the motivation of our work better.

Email Foldering Almost all email readers now support sorting emails messages into folders, and often allow the rules to be defined to do the sort automatically.

When an investigator checks the suspicious email data, he can use the same software (or another software tool which is able to interpret the folder's information) to read the folder's structure. As people usually manage their folders according to certain topics or relationships (For example, it has been found that the average number of folders containing the messages of a thread is 1.37 in Enron Email dataset [66]. That is, for messages in an average size thread, the messages in that thread are distributed across only 1.37 folders), it makes it easier for the investigators to understand the emails and search for useful information.

The limitation of email foldering is its redundancy of messages in the same folders. For example, it is quite often that emails from the same sender are put into the same folder. Sometime this kind of folder may consist of hundreds or thousands of messages between the two persons. Although the classification (to a certain sender) itself is quite successful, it gives little hint on what was actually going on.

The problem with the above systems is that they process email messages individually. Email thread detection and task management technologies are developed to handle the thread of emails. A thread is a series of topically related messages and replies. The thread indicates relations and common underlying activities among messages.

Email Thread Email thread detection tries to capture the *In-Reply-To* links between the “parent” message and the “child” messages. Email thread is an idea structural unit to represent and understand the conversation and group discussion carried out by Email messages. For computer forensics, an investigation tool that understands thread can save investigators considerable effort. For example, some software packages allow the users to view a summary of the entire thread, manage or delete

the entire thread, greatly reducing the number of emails that the users must inspect.

Detecting email thread is not difficult when the email server software strictly obeys the *re: subject line text in the original email* structure. But it has been suggested that the *In-Reply-To* links does not always reflect the relationship inside a thread because

1. The reply command is often used to avoid retyping an email address, rather than to continue a conversation. That is, new topics and new sub-topics often arise in the same thread.
2. Further users may disagree about what is on-topic in a thread, and off-topic responses can easily spawn sub-discussions.
3. On-topic contributors to a discussion may simply send a fresh message rather than using the reply *Re:*.

So one thread may contain more than one topic, and one topic may be distributed to more than one email thread.

Email-based Activity Management Email-based Activity Management is more ambitious as it tries to capture not only the threads of messages but also the threads of activities. For example, [22] allowed the user to split a thread and combine *technically unrelated threads*. So an email-based activity management system goes beyond system-defined threads to encompass user-defined task-centric collections.

As the communications among criminals or between criminal and victim can be considered as email activities obviously, email thread and activity management are very useful to computer forensic investigators, especially for the investigators to view and read the findings from the search result.

Figure 4.1: An Email Evidence Extraction Scenario

Figure 4.2: Forensic Email Evidence Extraction

To understand the computer forensic requirement on email evidence extraction, an investigation scenario is depicted in Fig. 4.1. It is assumed that the investigators already have initial case information (for example, the category of the criminal such as network hijack, pornography and sensitive data leakage or some original email messages) before looking at the email data. The investigators are assumed to know what to search and how to search — they build a keyword list to start search. A keyword search module returns some results. The investigators examine the result and extract relevant information manually. The relevant information may be used to refine the keyword list. Then the next round search starts. Such an evidence extraction procedure is time consuming and really excruciating. The drawbacks have been discussed in details in Chap. 3.

To release the investigators from the tedious and iterative procedures, an automatic email evidence search module is desirable. A simple flowchart is depicted in Fig. 4.2. The automatic email evidence extraction module takes the initial case information as input and is able to extract further case relevant data incrementally and automatically. Finally it will return the investigator all the potential evidence. The remainder of this chapter will study the design of an automatic email evidence extraction module with the concept of “Case-Relevance” developed in Chap. 3.

4.2 “Case-Relevance” Email Evidence Extraction

4.2.1 General Idea

Our purpose is to design a “Case-Relevance” Email Evidence Extraction system, which mimics the human’s iterative and bootstrapping evidence search procedure. The system starts from a limited amount of information, then does an incremental search to bring more and more relevant information. The system measures the relationship of information according to the predetermined case profiles and ranks the search results by their degree of “Case-Relevance” (Fig. 3.2). Such a system is able to give investigators straightforward support and assistance. The processing flow of the system is shown in Fig. 4.3.

The system still needs some human intervention at the start. The investigators check the captured email data and figure out some *provably case-relevant* email messages. These emails messages will be used as seeds for the following incremental search step. The investigators are also required to select case templates from a case database, according to the initial case information and the initial finding in these seed email messages. The investigators can use the default templates or do some customization work as they will. These case templates contain the knowledge regarding this kind of criminal case from the viewpoint of an intelligent information system, such as what kind of information should be searched and how they are represented in the email messages. Then all the remaining work can be accomplished by the automatic “Case-Relevance” Email Evidence Extraction system.

Figure 4.3: Automatic Email Evidence Extraction

4.2.2 “Case-Relevance” Email Evidence Extraction

The email extraction module goes through the whole email corpus searching for relevant messages to the seed email messages. The relevance is measured in the degrees of “Case-Relevance” defined in sub-section 3.2. Although a message is not labelled with an explicit tag such as “*Provably Case-Relevant*”, “*Probably Case-Relevant*” and “*Possibly Case-Relevant*”, the returned messages are ranked and listed in an order from “*Provably Case-Relevant*” to “*Absolutely Irrelevant*”. The top N returns of the given seed messages will be added into the queries for the next round search. So the module is an incremental search, which starts from the limited amount of seed messages and mine more and more case-relevant messages from the whole corpus.

The relevance judgement can be implemented by any methods as long as they can handle the Case-Relevance measurement. The method proposed in the chapter has two modules: an IE module and an email classification module. The IE module extracts “Case-Relevance” information from email messages according to predefined templates. The templates are determined from the initial information of the criminal case and the seed messages. The email classification module uses these “Case-Relevance” information along with the statistic method to calculate the degree of “Case-Relevance” for a given email message, and groups the relevant email messages together into a Case-Relevance email cluster.

Definition. *Case-Relevance Email Cluster:* the collection of email messages relevant to a given case, in which the degree of “Case-Relevance” of each message is not smaller than a cut-off threshold value γ .

With the above definition, the task of the “Case-Relevance” Email Evidence Extraction algorithm can be explained as following,

With the email corpus E , the seed email messages $M_1, \dots, M_k$ (k is the number of seed messages) and the case template P , try to find the Case-Relevance Email Cluster C which includes the seed email messages $M_1, \dots, M_k$.

Here, for the case template P , we use the information extraction template derived from the initial case information and the seed messages. The “Case-Relevance” Email Evidence Extraction algorithm starts with the initial Case-Relevance Email Cluster, which is exactly the set of the seed email messages $M_1, \dots, M_k$. The calculation is between the cluster C and the remaining email messages. Each message in cluster C will go through the IE module with the template P . Every word extracted by the IE module is labelled as *case relevant keyword*.

For the IE module, MinorThird [33] is used to extract entities from the email text. MinorThird is a toolkit for storing text, annotating text, and learning to extract entities and categorize text. It was written primarily by William W. Cohen in the Center for Automated Learning and Discovery, Carnegie Mellon University. It is open-source, and available for both commercial and research purposes.

For the email classification module, the method used is built based upon the TF-IDF method. TF-IDF [104] is a vector-based term weighting algorithm, which has been widely used in common text classifiers. As we have mentioned, TF-IDF achieves a high accuracy for classifying e-mail. Previous results [34] have shown that TF-IDF is an effective Machine Learning algorithm which achieves high accuracy rates for classifying e-mail. The actual algorithm we used is developed from the AIM text classifier algorithm. The AIM [18] text classifier is a TF-IDF style classifier developed at IBMs Almaden research laboratory. The algorithm of AIM is simpler

and easier to implement.

Each email message M is represented as a word-frequency vector $F(M)$, in which each component $F(M, w)$ represents the total number of times the word w appears in M , but if the word w has been labelled as *case relevant keyword*, the number will be multiplied by a case-relevance ratio σ . The cluster C is represented using a weighted word-frequency vector $W(C, w)$. Several steps are involved in computing $W(C, w)$.

First, the centroid vector $F(C, w)$ of the cluster C is computed by summing the word-frequency vectors for each message in the cluster:

$$F(C, w) = \sum_{M \in C} F(M, w) \quad (4.1)$$

The cluster centroid vector is then converted to a weighted word-frequency vector using the TF-IDF principle: the weight assigned to a word is proportional to its frequency in the cluster and inversely proportional to its frequency in other clusters. $FF(C, w)$ is defined to be the fractional frequency of word w among messages contained within cluster C , or the number of times word w occurs in cluster C divided by the total number of words in C :

$$FF(C, w) = \frac{F(C, w)}{\sum_{w' \in C} F(C, w')} \quad (4.2)$$

The definition of term frequency $TF(C, w)$ is,

$$TF(C, w) = \frac{FF(C, w)}{FF(E, w)} \quad (4.3)$$

where E is the entire email corpus. The document frequency $DF(w)$ is defined to be the fraction of clusters in which the word w appears at least once. The definition

of inverse document frequency $IDF(w)$ is:

$$IDF(w) = \frac{1}{DF(w)^2} \quad (4.4)$$

Finally, we can get the weight for word w in cluster C by combining the above two formulas to define :

$$W(C, w) = TF(C, w) \times IDF(w) \quad (4.5)$$

The weight vectors for each cluster are used to classify each new message. When a message M arrives to be classified, it is converted into a word-frequency vector $F(M)$ firstly. Then, the similarity between the message vector M and the weighted word-frequency vectors for the cluster C , is calculated by using a variation of the cosine distance called $SIM4$ [105]:

$$SIM4(M, C) = \frac{\sum_{w \in M} F(M, w)W(C, w)}{\min(\sum_{w \in M} F(M, w), \sum_{w \in M} W(C, w))} \quad (4.6)$$

The similarities of all remaining messages in E with the cluster C are calculated with Equation (4.6). The messages with a similarity value greater than a cut-off threshold value γ are considered to be *case relevant* and put into the cluster C . The original text of the newly added email messages go through the IE module with the Template P . Any words captured by the Email Information Extraction module will be labelled. The weighted word-frequency vectors for the cluster C will be recalculated with Equations (4.1) to (4.5). Then it starts the next round calculation on the remaining messages within the cluster. The calculation stops when no changes in

Table 4.1: Test Email Clusters

Cluster ID	Number of Email Messages
Student Organization	73
Sport Activity A	56
Facility Booking	37
Toastmaster Club	29
Reading Group	27
Sport Activity A	23
Conference A	17
Online Transaction A	17
Sport Activity B	13
New Year Party A	13
Conference B	12
Online Transaction B	12

the members of the cluster C will be made.

4.3 Experimental Result and Analysis

Experiment Environment

The email dataset used in the experiment was collected from personal emails. The dataset consists of 1,260 emails. The dataset spanned a period of two years, consisting of many communications on university activities such as sports events, job interview and student organization's issues. A thorough manual check over the email corpus was made and 12 email clusters were classified, whose sizes ranged from 12 to 73 messages. These email clusters are listed in Table 4.1.

The MinorThird toolkit [33] was used to extract entities from the email text. 100 email messages from the email corpus were randomly selected and manually labelled to train the MinorThird. The IE templates for each email cluster was worked out

Figure 4.4: Sample of IE Templates

manually. Some template with sample fill-in data is shown in Fig. 4.4.

Before calculating the term frequencies and the inverse document frequencies, we parsed, removed the common words with a stop-list and did stemming with the Porter algorithm [3].

For each cluster, five email messages from each cluster were chosen randomly as the *Seed Email Messages*. The three largest email clusters in Table 4.1, “Student Organization”, “Sport Activity A” and “Facility Booking”, were chosen as testing data. The other clusters were used as training data to determine the value of the cut-off value γ and the case-relevance ratio σ . To simplify the training, the case-relevance ratio σ was constrained to be integer. To classify email for the whole email corpus, different combinations of γ and σ were run with the seed email messages and templates of the training data. The results were compared with the manually determined clusters and analyzed to locate the optimal values of γ and σ . The final values were determined as $\gamma = 0.78$ and $\sigma = 3$.

Experiment Result and Analysis

Our “Case-Relevance” Email Evidence Extraction (*CRE3*) method was compared with two baseline approaches: *Threading* and a *TF-IDF* clustering methods. The threading approach uses the in-reply-to field in message headers to group together messages into a thread. We think *Threading* is closer to the actions that a human investigator takes in a real case. In current computer forensic systems, the investigators often check the email threads to find related information of the initial messages. The *TF-IDF* clustering method used the same cut-off value as $\gamma = 0.78$.

Precision and Recall were used to measure the performance of the three methods. Precision is the proportion of retrieved matches that are correct. Recall is the proportion of correct matches actually retrieved. The result is shown in Table 4.2. It is not surprising that *Threading* is leading in Precision in all three test groups. It reflects the user’s behavior that the user stucked to the same topic when using *Re:* tag. On the other hand, the Recall of *Threading* is the worst in all three. It exhibits one common feature in email communication that on-topic contributors to a discussion may simply send a fresh message rather than using the reply *Re:*. For example, we had a check at the cluster of “Facility Booking” and found the messages are distributed in 8 threads actually.

The result of the new “Case-Relevance” Email Evidence Extraction algorithm is satisfying. The improvement in Precision means that it is more efficient than the standard *TF-IDF* method in returning relevant email messages. It also achieved obvious improvement in Recall. It proved that our incrementally searching strategy with case information is very effective in capturing new evidence not included in the initial information. As computer forensic investigation pays more attention to the

Table 4.2: Result for CRE3, TF-IDF and Threading

	Facility Booking		Sport Activity A		Student Organization	
	Precision	Recall	Precision	Recall	Precision	Recall
CRE3	0.67	0.86	0.82	0.72	0.79	0.78
TF-IDF	0.56	0.76	0.77	0.54	0.73	0.51
Threading	0.88	0.22	1.00	0.48	0.93	0.37

exhaustive search, it will be a great help to the investigators.

4.4 Summary

The Case-Relevance Email Evidence Extraction algorithm is an implementation of the “Case-Relevance” computer forensic framework developed in Chap. 3. By using predefined IE templates and an IE module, the method succeeds in introducing “Case-Relevance” information to an email classification procedure, which brings up email messages directly relevant to the criminal case. Experimental results showed that the new algorithm is highly efficient and is able to do great help to computer forensic investigators.

Chapter 5

Phonetic String Search of Written Text

Phonetic string search of written text is used to identify and retrieve strings that may have similar pronunciation, regardless of their actual spelling. Phonetic string search is very useful in computer forensics to solve the name string searching problem. A novel phonetic string searching method, Syllable Alignment Pattern Searching algorithm (SAPS), has been proposed in this chapter. While most of the existing methods convert a string into intermediate codes with phonetic transformation rules, SAPS calculates the optimal pairing of the corresponding syllables in two strings and uses the score to rank the similarity of strings.

5.1 Motivation

There are two major difficulties in building and evaluating a phonetic string searching algorithm. One is the inconsistencies in human judgement. Researchers tried building

a “blind” test, that is, the designers know little about the assessors’ preference and the assessors do not have any information about the algorithm. They have done perfectly in the data collection step[97, 136, 134, 57]: the string databases came from a variety of media resources and the test queries were selected randomly. Regrettably, the relevance judgement steps were bias-prone. For example, in [136] it was found that the assessors had a high probability to be influenced by the readers via intonation or body language so that “where two teams of assessors judged the same queries and both identified a good number of relevant words, only one or two of the words were in common” .

Another difficulty is the evaluation and comparison of certain mechanisms in a real test. These mechanisms are designed based on a thorough research of the target language and the abstraction of the human’s perception behavior. But again, the fuzzy nature of human judgement may easily ruin the designer’s effort in setting up a delicate test environment to regulate the weights of all kinds of factors involved in making the judgement. In a real world test environment, one or two factors (varying from different assessors) are much more important than others especially with the judgement on short strings. It is not strange to observe that “The performance of tapering is disappointing, despite good indications from our initial experiments and strong positive results from work with users of a commercial system” [136].

For the above reasons, it is difficult to compare different methods reliably while they were built and tested on different datasets. We are not going to discuss which dataset is more credible - all reflect the relevant judgement of a certain assessor group to a certain dataset in some way. But we must realize that a successful algorithm may fail with a different dataset.

5.2 Criteria of an Optimal Method

In this work, four criteria are set up to evaluate the effectiveness of the different phonetic string methods:

1. Precision and Recall. Precision is the proportion of retrieved matches that are correct; Recall is the proportion of correct matches actually retrieved. Recall and Precision can be used together or separately depending on the requirements of a particular system.
2. The overlap of the output with those of other methods. As stated earlier, the overall performance can be effectively improved by combining the results obtained from different methods. If the return of an algorithm has least overlap with those of other methods, the combination should be most effective [57, 136, 97].
3. The ability to adapt to a new dataset. A successful algorithm is expected to be able to modify its parameters effectively and achieve optimal performance with different datasets on average.
4. The balance between Performance and Cost. The cost refers to computing complexity and can be measured by time complexity (cost) and space complexity (cost). There is a trade-off when developing a phonetic string search algorithm between including adequate rules to provide an acceptable quality and maintaining an acceptable cost.

This chapter proposes a novel strategy to compare two phonetic strings based on syllable alignment. That is, the strings are segmented into syllables, and then a sequence alignment matrix is calculated to find the optimal pairing of the corresponding

syllables in the two strings. An efficient phonetic string search algorithm based on this strategy, called *SAPS*, has been developed.

Section 5.5 addresses the execution time problem raised by our fourth criterion by proposing a fast online name searching algorithm running in linear time. The algorithm can be easily extended to a multiple version, which is able to search multiple name strings simultaneously and accelerate the overall processing speed dramatically.

5.3 Syllable Alignment Pattern Searching

Our Syllable Alignment Pattern Searching (*SAPS*) algorithm captures the phonetic features of a string and is effective in similarity measurement of phonetic strings. It is based on the following assumptions about how human beings recognize phonetic strings,

1. Strings are compared ***syllable-by-syllable*** rather than letter-by-letter. The Syllable is “a unit of spoken language consisting of a single uninterrupted sound formed by a vowel, diphthong, or syllabic consonant alone, or by any of these sounds preceded, followed, or surrounded by one or more consonants.” [5].
2. As pitch and stress always fall on the beginning of a syllable, the first letter of the syllable carries the major phonetic information.
3. The Corresponding Syllable Pairs in two strings are located and compared with each other. The Corresponding Syllable Pair have similar spelling/pronunciation, tone and coupled positions in the two strings, e.g., the syllable “son” in the string “Stinson” will be compared with the “son” in “Stevenson”, not with “ven” as in the sequence order.

The basic idea of *SAPS* is to segment two phonetic strings into syllables and find the optimal pairing of their corresponding syllables. Then the alignment score is used to measure the similarity of two strings. Hence the algorithm is called “Syllable Alignment”. For example, given the following name strings (the ‘^’ indicates the first letter of a syllable), the preferred alignment is:

$$\begin{array}{cccccccccccc} \hat{S} & \hat{t} & e & \hat{v} & e & n & \hat{s} & o & n & \hat{S} & \hat{t} & e & \hat{v} & e & n & \hat{s} & o & n \\ \hat{S} & \hat{t} & e & \hat{v} & e & n & \hat{s} & - & - & \hat{S} & \hat{t} & i & - & - & n & \hat{s} & o & n \end{array}$$

SAPS consists of three steps: a preprocessing step, a syllable segmentation step and an alignment and similarity-calculation step.

Preprocessing

The preprocessing step deals with some special letter combinations and simplifies the following segmentation step. The preprocessing rule is quite flexible, and can be easily modified to fit different target datasets. In *SAPS* currently, the preprocessing is exactly the same as what happens in Phonix [48, 49]. Phonix is a popular Soundex [54] variant. It adds a preprocessing step before converting a string into phonetic codes. The preprocessing uses about 160 letter-group transformations to standardize the string. For example, it maps “tjV” (where V is any vowel) to “chV” at the start of a string, “x” to “ecs” and “ph” to “f”, etc. Please refer to Appendix B for the details of the transformations, where the C source code of the preprocessing step has been listed.

However the aim of the preprocessing step of *SAPS* is different from that in Phonix: in Phonix, the procedure is used to get more accurate phonetic codes; in *SAPS*, it is used to obtain an accurate syllable segmentation.

Segmentation

The segmentation step converts the string into a concatenation of syllables. For example, “Stevens” $\Rightarrow$ “S|te|ven|s”, where ‘|’ represents the boundary between two consecutive syllables. The rules to segment a string in SAPS are based on the definition of Syllable in [5]. That is “a unit of spoken language consisting of a single uninterrupted sound formed by a vowel, diphthong, or syllabic consonant alone, or by any of these sounds preceded, followed, or surrounded by one or more consonants.” The rules can be summarized as the following,

1. A syllable starts from a consonant, followed by a vowel or a diphthong, and ends before the first letter of the next syllable. For example “S|**te**|ven|s”.
2. A syllable may end with a consonant if and only if the consonant follows a vowel or a diphthong in the syllable and the consonant precedes another consonant, e.g. “S|te|**ven**|s”
3. A single consonant can be a syllable, e.g. “**S**|te|ven|s”.
4. At the beginning (and only at the beginning) of a string, a syllable can start with a vowel, e.g. “**Au**|ty”. That is, the beginning of the string is always the start letter of a syllable.

Alignment and Similarity Calculation

A scoring scheme, considering all possible alignments between the first letter of a syllable, normal letter and gap, has been defined as (the ‘^’ indicates the first letter of a syllable) :

- Substitution Function

$$\left\{ \begin{array}{l} S(x, y) = \begin{cases} s1, & x = y \\ s2, & x \neq y \end{cases} \\ S(\hat{x}, y) = s3, \\ S(\hat{x}, \hat{y}) = \begin{cases} s4, & \hat{x} = \hat{y} \\ s5, & \hat{x} \neq \hat{y} \end{cases} \end{array} \right. \quad (5.1)$$

- Gap Penalty Function

$$\left\{ \begin{array}{l} g(x, -) = g1 \\ g(\hat{x}, -) = g2 \end{array} \right. \quad (5.2)$$

The Substitution Function, $s(x, y)$, indicates the score of aligning a character x with a character y . The Gap Penalty Function, $g(x, -)$ and $g(-, x)$, indicate the cost of aligning a single character x with a gap and aligning a gap with character x . ‘=’ is used to represent two characters that are identical; ‘ $\neq$ ’ to represent two characters that are different (e.g., ‘a’=‘a’, ‘d’ $\neq$ ‘t’). “ $S(x, y) = s1, x = y$ ” represents that when two normal letters x and y are aligned, if they are identical, the alignment score is $s1$. Both the Substitution Function and the Gap Penalty Function are symmetric, i.e., $s(x, y) = s(y, x)$ and $g(x, -) = g(-, x)$.

The values of the parameters $s1$ – $s5$ and $g1$ – $g2$ in Eq.5.1 and Eq.5.2 depend on how the similarity of the two strings and how the different weights for the alignment are defined. In *SAPS*, these parameters were determined by experiments. The alignment of normal letters were set as $s1 = 1, s2 = -1$ and $g1 = -1$ first. Considering the typical length of English surname is 4 to 9, the ranges of the other parameters were set as $s4 \in [+1, +6]$ and $s3, s5, g2 \in [-6, -1]$. Their actual values were determined

Figure 5.1: The Scale of the Parameters

by experiments.

To build the experiment dataset, an English surname dataset was collected from the Waite Genealogy Research Forum Surname List [46]. 30 surnames randomly collected from the public news wires were selected as the query terms and their matches measured by the similarity in pronunciation in the surname dataset were determined manually. The alignment of each query term and their matches were calculated with all possible parameter combinations. These alignments were compared to find the optimal combination of parameters. The values of the parameters were finally determined as,

$$\begin{cases} s1 = 1; s2 = -1; s3 = -4; s4 = 6; s5 = -2 \\ g1 = -1; g2 = -3 \end{cases} \quad (5.3)$$

The scale is shown in Fig. 5.1. Although there is no explicit relationships among these parameters, how SAPS weights different alignment operations could be understood. For example, given $|s1| = |s2| = |g1| = 1$, $s4 = 6$ means the alignment of two first letters of a syllable has the same weight as the alignment of six consecutive normal letters.

For two strings of length m and n , a $(m+1) \times (n+1)$ matrix M can be constructed to hold the distance values. The element of the matrix, $M[i, j]$, represents the operation score to match x_{1-i} to y_{1-j} , where $i \in [1, m], j \in [1, n]$. $M[i, 0]$ and

$M[0, j]$ represent the operation score between a string of length i or j and the empty string. And the value of $M[m, n]$ is the final similarity score of the two strings. The larger the score, the more similar the two string are. The score can be used to rank and sort all returned matches for a given query.

$$M[i, j] = \max \begin{cases} M[i-1, j-1] + s(S_1[i], S_2[j]) \\ M[i-1, j] + g(S_1[i], -) \\ M[i, j-1] + g(S_2[j], -) \end{cases} \quad (5.4)$$

With the initial conditions:

$$\begin{cases} M[0, 0] = 0 \\ M[i, 0] = M[i-1, 0] + g(S_1[i], -) \\ M[0, j] = M[0, j-1] + g(S_2[j], -) \end{cases} \quad (5.5)$$

The matrix can be constructed from $M[0, 0]$, from top to bottom and left to right, as it is shown in Algorithm 1. The time complexity is $O(m \cdot n)$. The space complexity is also $O(m \cdot n)$, and can be reduced to $O(\min(m, n))$, since it only needs to record one column or one row in each loop.

For example, given the two name strings “Stevenson” and “Stinson”, their alignment matrix by *SAPS* is built as shown in Fig. 5.2, and the final similarity score is 16, the value of $M[m, n]$.

Backtracking

To get the actual string alignment(which may be needed in examining the returned matches), a *Backtracking* procedure is taken from $M[m, n]$ to $M[0, 0]$ in the matrix M to find the path that led to the score. The work can be done during the construction of the alignment matrix. That is, a $(M+1) \times (N+1)$ matrix P is used to keep the path information, in which each cell saves a pointer to its parent cell. Then the

Algorithm 1 Syllable Alignment of S_1 and S_2

Input: Strings S_1 and S_2 of length m and n , Substitution Function s , Gap Penalty Function g
Output: Score of the syllable alignment of S_1 and S_2
 $M[0, 0] = 0$
for $i = 1$ to m **do**
 $M[i, 0] = M[i - 1, 0] + g(S_1[i], -)$
end for
for $j = 1$ to n **do**
 $M[0, j] = M[0, j - 1] + g(S_2[j], -)$
end for
for $i = 1$ to m **do**
 for $j = 1$ to n **do**

$$M[i, j] = \max \begin{cases} M[i - 1, j - 1] + s(S_1[i], S_2[j]) \\ M[i - 1, j] + g(S_1[i], -) \\ M[i, j - 1] + g(S_2[j], -) \end{cases}$$

 end for
end for
return $M[m, n]$

path from final cell $P[m, n]$ back to $P[0, 0]$ indicates the alignment. The detailed procedures of the alignment calculating with the backtracking matrix are given in Algorithm 2, in which ‘ $\nwarrow$ ’ represents the pointer from $M[i, j]$ to $M[i - 1, j - 1]$, ‘ $\leftarrow$ ’ represents the pointer from $M[i, j]$ to $M[i - 1, j]$ and ‘ $\uparrow$ ’ represents the pointer from $M[i, j]$ to $M[i, j - 1]$.

Several optimal alignments may exist for a pair of sequences. These alignments can be kept all as long as the backtracking keeps all available tracks. In case only one of the optimal alignments is needed, predetermined preference ordering over paths can be used to simplify the backtracking (as shown in Fig. 5.3 where 1 refers to the highest priority). Algorithm 2 deploys the High-Road order.

In the previous example, the backtracking path is indicated by the arrows in Fig. 5.2. The final alignment of the two strings is:

Algorithm 2 Syllable Alignment with Backtracking Matrix

Input: Strings S_1 and S_2 of length m and n , Substitution Function s , Gap Penalty Function g

Output: Score of the syllable alignment of S_1 and S_2

$M[0, 0] = 0$

$P[0, 0] = NULL$

for $i = 1$ to m **do**

$M[i, 0] = M[i - 1, 0] + g(S_1[i], -)$

$P[i, 0] = \leftarrow$

end for

for $j = 1$ to n **do**

$M[0, j] = M[0, j - 1] + g(S_2[j], -)$

$P[0, j] = \uparrow$

end for

for $i = 1$ to m **do**

for $j = 1$ to n **do**

$M[i, j] = \max \begin{cases} M[i - 1, j - 1] + s(S_1[i], S_2[j]) \\ M[i - 1, j] + g(S_1[i], -) \\ M[i, j - 1] + g(S_2[j], -) \end{cases}$

if $M[i, j] = M[i, j - 1] + g(S_2[j], -)$ **then**

$P[i, j] = \uparrow$

else if $M[i, j] = M[i - 1, j - 1] + s(S_1[i], S_2[j])$ **then**

$P[i, j] = \nwarrow$

else if $M[i, j] = M[i - 1, j] + g(S_1[i], -)$ **then**

$P[i, j] = \leftarrow$

end if

end for

end for

return $M[m, n]$

		S	t	e	v	e	n	s	o	n
	0	-3	-6	-7	-10	-11	-12	-15	-16	-17
S	-3	6	3	2	-1	-2	-3	-6	-7	-8
t	-6	3	12	11	8	7	6	3	2	1
i	-7	2	11	11	8	7	6	3	2	1
n	-8	1	10	10	7	7	8	5	4	3
s	-11	-2	7	7	8	7	6	14	13	12
o	-12	-3	6	6	7	7	6	13	15	14
n	-13	-4	5	5	6	6	8	12	14	16

Figure 5.2: Alignment Matrix of “Stevenson” and “Stinson” and Backtracking Path

Figure 5.3: Preference Order of Backtracking Path

$$\begin{array}{ccccccc} \hat{S} & \hat{t} & e & \hat{v} & e & n & \hat{s} & o & n \\ \hat{S} & \hat{t} & i & - & - & n & \hat{s} & o & n \end{array}$$

5.4 Adaptive Phonetic String Search

As discussed in Sect. 5.1, rather than attempting to design a method that runs well in all environments (it is too expensive since there are too many factors to cover), a more pragmatic solution is to design an adaptive algorithm to achieve an optimal performance with different datasets. Such an algorithm should be comparatively simple but flexible. There are two possible ways,

1. Modify the Phonetic Transformation Rules. For example, the phonetic letter groups of Soundex has been modified as in [136]; the Transformation rules in the preprocessing step of Phonix have been changed in [57, 134]. The drawbacks

of this method are the high demand for knowledge of the target language, the difficulty in balancing efficiency and complexity, and the limited applicability to certain datasets only.

2. Modify the Weights of Operations. This method requires the algorithm itself to be sufficiently flexible. [88] defined a cost model that allows the user to change the cost of all operations of every character, thus tweaking the model to different scenarios or languages. [102] provided a complex stochastic model which learns the primitive edit costs from a training corpus. The problem for the above methods is the balance between flexibility and the training cost. An algorithm with many parameters may be more accurate but requires more training data and more complicated training procedure. Our Syllable Alignment Pattern Searching model offers a very effective way to adapt to different datasets and is easy to train, which will be shown by the experimental results in sect. 5.6.2.

The original *SAPS* algorithm can be extended to *Adaptive SAPS* by adapting its parameters $s1-s5$ and $g1-g2$ in Eqs. 5.1 and 5.2 to a certain dataset. *Adaptive SAPS* sets $s1 = 1$, $s2 = -1$ and $g1 = -1$ as the basic operation scores, and allow $s4$ to vary between $[+1, +6]$ and $s3, s5, g2$ between $[-6, -1]$. The boundaries consider the length of typical English Surname as 4 to 9.

For a given dataset, a small subset and its relevant judgement can be built as the training data, as shown in Fig. 5.4. In the training procedure, the *Adaptive SAPS* with all its possible parameter combinations runs on the training data, and the results are compared in the Recall-Precision-Graphs. The parameters of the best result is the trained setting for that dataset.

Figure 5.4: Training of Adaptive SAPS

5.5 High Speed Phonetic String Search

While the previous sections focus on accuracy, this section looks at another important issue, speed. SAPS runs in $O(mn)$ with dynamic programming. Although the modern approximate string search methods, such as Bit-parallelism [8, 9, 10], can achieve linear time complexity $O(n)$, most of them are designed for the single edit distance problem only, in which the operation costs of insertion, deletion and substitution all equals 1. The non-unitary operation cost (that the operations on first letter cost more than 1) in SAPS obstructs it from being implemented by these fast searching methods.

Another effective way to accelerate the overall speed is to search several phonetic strings simultaneously. In multiple string search, the overhead of re-verification must be considered seriously. The multiple string searching algorithms in [133, 84, 14, 87, 65, 68] all are very sensitive to the error ratio. When the error ratio increases, the re-verification cost will rise sharply, which moderates the overall speed-up of simultaneous multiple searching. A high error ratio is unavoidable in SAPS because of the non-unitary operation cost. So it is difficult to extend SAPS to a multiple search version.

In this section, we develop a fast online phonetic search algorithm without compromising much accuracy. It can be easily implemented by Bit-parallelism in linear time $O(n)$. The extension of the algorithm to handle multiple phonetic string search has also been discussed.

Table 5.1: Phonetic Letter Groups

Groups	0	1	2	3	4	5	6	7	8	9
Letters	aeiouhwy	bp	ckq	dt	lr	mn	gj	fpv	szx	csz

5.5.1 Fast Online Searching Algorithm

First, we would like to make some definitions and symbol usages in the following discussion explicit:

- *edit operation*: *insertion*, *deletion* and *substitution* of single character;
- *error*: cost of the edit operations defined by the distance function;
- m : the length of the string;
- n : the length of the text;
- k : the maximal number of errors allowed;
- α : *Error Ratio* $\alpha = k/m$;
- w : the bit length of a single computer word;
- Σ : the alphabet set of size σ .

The Fast Online Name Searching (FONS) algorithm was previously reported in [52]. FONS is a phonetic distance measure that combines the properties of edit distance with the letter-grouping strategy used by Soundex and Phonix. Table 5.1 shows the phonetic letter groups used in FONS, which is similar to Phonix, but allows the overlap of letter groups like Editex.

The detailed distance function of FONS can be represented by the formula shown in Eq. 5.6. (Note that insertion in string x equals to deletion in string y .)

$$\left\{ \begin{array}{ll} edit(0,0) = 0; & \\ edit(1,0) = 2; & \text{(deletion on first letter costs 2)} \\ edit(0,1) = 2; & \text{(insertion on first letter costs 2)} \\ edit(i,0) = edit(i-1,0) + 1; & i \geq 2 \\ edit(0,j) = edit(0,j-1) + 1; & j \geq 2 \\ edit(i,j) = \min[edit(i-1,j) + 1, & \\ \quad edit(i,j-1) + 1, & i \geq 1, j \geq 1 \\ \quad edit(i-1,j-1) + r(s_i, t_j)]. & \end{array} \right. \quad (5.6)$$

Where

$$r(a,b) = \left\{ \begin{array}{ll} 0 & a=b \\ 1 & a,b \text{ in the same group} \\ 2 & \text{otherwise} \end{array} \right\} \text{ both } a,b \text{ are first letters} \quad (5.7)$$

$$\left\{ \begin{array}{ll} 0 & a=b \text{ or } a, b \text{ in the same group} \\ 1 & \text{otherwise} \end{array} \right\} \text{ other positions}$$

And a replacement cost function $r(a,b)$ (Eq. 5.7) is defined as following, where a and b are single characters in the two strings.

1. If a and b are the first letters of two strings: $r(a,b)$ returns 0 when a and b are identical, 1 when a and b are in the same letter group, otherwise returns 2; all the insertion and deletion operations on first letter cost 2.

2. If a and b are at other locations: $r(a, b)$ returns 0 when a and b are identical or in the same letter group, returns 1 otherwise; all the insertion and deletion operations have a cost of 1.

It can be noticed that FONS deploys a special treatment on the first letter in a string, that is, the operations on the first letter have twice distance cost as those in the consequent locations. This consideration comes from the observation that the first letter carries more pronunciation information than the consequent letters, e.g. the edit distances between “Karl”, “Karn” and “Karl”, “Barl” are equal, but the first pair is usually considered as closer by human judgement. Experimental results showed that special treatment could improve the performance by up to 37%.

The unitary operation cost (except for the first letter) allows FONS to be realized by fast pattern searching algorithms easily. In our experiment, we chose the Bit-parallelism simulation of Non-Deterministic Finite Automata (NFA) [11, 13, 15] for two reasons: it is one of the fastest solutions in practice, and it can be easily extended to a multiple version.

The Bit-parallelism simulation of NFA takes advantage of the intrinsic parallelism of the bit operations inside a computer word. The latest result [15] has a time complexity of $O(\lceil k(m - k)/w \rceil n)$ in worst-case and $O(\lceil k^2/w \rceil n)$ on average with a space cost $O(km)$, that is, it runs in linear time $O(n)$ when all bits of the automata can fit into a single computer word. Considering the typical length of English surname (4–9 letters), moderate error ratio ($\alpha < 0.5$) and the current 32/64 bits word length used in most computer systems, the FONS algorithm can run in linear time $O(n)$ in both the worst and the average case in most current computer systems.

5.5.2 Multiple Phonetic String Search

A formal definition of *Multiple Phonetic String Search* can be represented as:

Given a text of length n and r phonetic strings of length $m : S_1, \dots, S_r$, we want to find all occurrences of any one of the strings in the text, with maximal number k of errors allowed. All the text and the strings sequences over an alphabet Σ of size σ .

As indicated in [14, 68], the ability of Bit-parallelism algorithms to deal with character classes can be used to build multiple pattern searching algorithm by superimposing all NFAs. It requires all strings have the same length, otherwise it needs to truncate the longer ones. This method is very suitable for name string search. Most of English names have a length of 4 to 9 letters. So given a large number of surnames to be searched, it is easy to group name strings according to their lengths.

FONS can be easily extended to search for multiple names. Assuming we can put r names with the same length m into one group, all the NFA have the same structure, differing only in the labels of the horizontal arrows. Then all the NFAs can be superimposed into a single one. The superimposed NFA is able to search r patterns simultaneously. Fig. 5.5 shows a superimposed NFA for searching name strings “Tom” and “Sam”.

In the ideal situation, the superimposed NFA can reduce the total search time by r times. But in most cases, it accepts some relaxed strings as a match within k errors, e.g., the NFA in Fig. 5.5 also accepts “Tam” with zero errors. So each occurrence reported by the NFA has to be verified for all patterns involved. As the number of patterns increases, the verification cost in the multiple searching becomes non-negligible compared with the searching time. In this case, the set of patterns

Figure 5.5: A Superimposed NFA to search “Tom” and “Sam” simultaneously

Figure 5.6: Multiple String Search Procedure

is partitioned into groups of r' patterns each, where r' is small enough to make the cost of verifications negligible and the r' always exists, since for $r' = 1$ it is the same as the single pattern searching and no verification is needed. Then the NFA of each group is built and to perform $\lceil r/r' \rceil$ independent searches. The total searching cost is $O(rn/r')$, when the cost of a sequential search is $O(rn)$. The whole procedure is shown as Fig. 5.6.

As mentioned in the previous paragraph, the size r' of the groups should be as large as possible to increase the degree of parallelism, but small enough for the verifications to be not significant. The optimal value for r' and the complexity of the search on random text has been analyzed in [14, 15, 68] as Eq. 5.8 and Eq. 5.9.

$$r' = \frac{\sigma}{(2 \begin{bmatrix} m+k \\ m \end{bmatrix} (m+k)(m-1))^{\frac{1}{m}}} \quad (5.8)$$

and the total search time becomes

$$\frac{nr}{\sigma} \frac{m}{m-1} \left(\begin{bmatrix} m+k \\ m \end{bmatrix} 2(m+k)(m-1) \right)^{\frac{1}{m}} \quad (5.9)$$

We deploy the hierarchical verification method proposed in [68] to decrease the average verification cost. The idea of the hierarchical verification method is to try to quickly determine that the match of the small piece is not in fact part of a complete match. For typical English surnames with lengths varying from 4 to 9 and error ratio less than 40% (it is not a high value actually since it only allows 2 different letters for a 5 character-long string), a speedup ratio of 2 to 4 over sequential searching can be expected.

5.6 Experimental Result and Analysis

5.6.1 Experimental Environment

Name string (personal name, agent name, address and some technical terms) search is a typical phonetic string searching problem, and has been widely used in phonetic string search evaluation. In our experiment, we use two name datasets:

1. The *COMPLETE* surname corpus from Pfeifer [97]. The corpus contains 14,972 distinct surnames collected from AP newswire, ACM abstracts, Federal Register, Wall Street Journal, ZIFF-Davis Publishing, University of Dortmund phone

Table 5.2: Sample Queries and Match Lists in COMPLETE Dataset

Query	Match List
Bloch	Blache, Black, Blake, Bleck, Bloch, Block, Blog, Blough, Boch, Bolch
Schiller	Scheel, Scheele, Scheler, Scheller, Schellhardt, Schiller, Schilling, Schueler, Schueller, Schuller, Schuyler, Shalla, Shellhardt, Shlaer, Siler, Stiller

book and a bibliographic database. The source includes 90 randomly selected names used as queries. Each query has a manually judged set of relevant names. There are 1,187 relevant names in the dataset for the 90 queries. Some samples are shown in Table 5.2.

2. *Zobel's* dataset [136] has over 30,000 distinct surnames collected from Internet news. 100 queries were randomly collected from the Melbourne White Pages telephone directory. Three sets of judgements have been initialized, set A on 25 queries, set B on 50 queries and set C on 50 queries. Relevance judgements for set A and B were determined by a team, and set C was conducted by a single assessor. Set A has 26.5 correct matches per query, set B has 5.4, and set C has 8.7.

Assuming all returned matches to a given query Q are distributed in total k ranks and sorted accordingly as shown below. Within one rank, there may be more than one match term, and they are in random order (*weak-ordering*). It is assumed that the user examines the result ranking, randomly selecting a term from the currently topmost rank which has not been read completely yet.

$$Q \rightarrow \overbrace{\underbrace{T_r}_{1} \underbrace{T_r T_r T_i}_{2} \cdots \underbrace{T_i T_r}_{h-1} \underbrace{T_r T_i T_r T_r T_i T_i T_r T_i}_{h} \cdots}_{k \text{ ranks}}$$

Here T_r represents a relevant match term (which is similar to the query in its pronunciation), T_i represents an irrelevant match term, R_j is the number of the total relevant matches in rank j ($1 \leq j \leq k$) and I_j is the number of the total irrelevant matches in rank j .

To deal with the interpolation problem in *weak-ordering* algorithms, we use the probability of relevance (*PRR*) method proposed in [124]. For a given *recall* value, the number of the relevant matches requested is $N_R = recall \times N$ (where N is the total number of relevant terms of the query in the database). Assuming the currently picked up term is in rank h ($1 \leq h \leq k$), *PRR* can be defined as:

$$PRR(N_R) = \frac{N_R}{N_R + \sum_{j=1}^{h-1} I_j + \frac{(N_R - \sum_{j=1}^{h-1} I_j) \cdot I_h}{R_h + 1}} \quad (5.10)$$

PRR is thus an estimation of the momentary precision for the *recall* value. For the comparison of the results we use Recall-Precision-Graphs (RPG), in which the *x*-axis is the *recall* and *y*-axis is the *probability of relevance*. Precision values for recall ranging from 0.05 to 1 with 0.05 step intervals have been averaged over the results of all query terms.

In our experiments, an exhaustive relevance judgment is not necessary since only the first dozens of results make sense in a practical system. We utilized the idea of using a pool from [136] where each algorithm returned the top 30 answers for each query. (This was not applicable for Soundex since Soundex returned all matches that it considered relevant). As a confidence test, we used each algorithm to find 50 returns for each query and found only a few additional matches.

Table 5.3: Correct Matches in Top Returns

	Top 30 returns		Top 10 returns	
	Total	Average	Total	Average
<i>SAPS</i>	628	6.98	459	5.10
Editex	618	6.87	433	4.81
Edit Distance	598	6.64	380	4.22
Soundex	589	6.54	325	3.61

5.6.2 Results and Analysis

Syllable Alignment Model

Original SAPS The original *SAPS* was compared with Soundex, Edit Distance and Editex. The result with the COMPLETE dataset is shown in Fig. 5.7, in which our new algorithm, *SAPS*, shows an obvious improvement over Soundex, Edit Distance and Editex. To give a better understanding about their performance in a practical system, we examined the number of correct matches in the top 10 returns and top 30 returns. The result is shown in Table 5.3. *SAPS* returned 5.1 correct matches on average to the 90 queries in its top 10 returns and 6.98 in its top 30, the best of the four listed in the table. It was observed that although Soundex fetched 6.54 correct matches in its top 30 returns (only 6.3% worse than *SAPS*), it only retrieved 3.61 correct matches in its top 10 returns, (29.2% worse than *SAPS*). This means that while Soundex retrieved correct matches, it also returned more erroneous terms than the other methods.

The overlap of the correct returns from the four algorithms were examined and the result is shown in Table 5.4, where the correct matches shared between two methods are expressed in the format of *number(percentage value)*. It was noticed

Figure 5.7: *SAPS* with COMPLETE Dataset

Table 5.4: Overlap of the Returns (COMPLETE Dataset)

	Editex	Edit Distance	Soundex
<i>SAPS</i>	452(56.9%)	431(56.8%)	387(46.6%)
Editex		519(74.5%)	425(53.7%)
Edit Distance			379(46.9%)

that *SAPS* had only 56.8% of its correct matches in common with Edit Distance, and 46.6% with Soundex. While even *SAPS* only succeeded in fetching 628 correct matches in its top 30 returns (52.9% of the total 1,187 relevant terms in COMPLETE), the result suggested that combining *SAPS* with other methods will bring promising improvements.

SAPS has also been tested with Zobel's dataset. Unfortunately, *SAPS* performed poorly as shown in Figs. 5.8, 5.9, and 5.10. The reason of the performance variation has been explained in Sect. 5.4. Editex was designed and tweaked based on the Zobel's dataset. It is not surprising that Editex performed the best. It also helps to

Figure 5.8: *SAPS* with Zobel's A Dataset

support our idea that an optimal method should be trained for different datasets.

Adaptive SAPS This part examines *Adaptive SAPS*'s ability of learning and adapting to different datasets, especially where the original *SAPS* performed poorly.

The 90 queries in COMPLETE dataset were divided into two subsets: 30 queries acted as the training data and the remaining 60 queries as the test data. Firstly the *Adaptive SAPS* ran with the training data and *SAPS*(-1+2-4-6)—the *Adaptive SAPS* with $s_3 = -1$, $s_4 = +2$, $s_5 = -4$ and $g_2 = -6$, was determined as the trained setting. Then *SAPS*(-1+2-4-6) was compared with Edit Distance, Editex and the original *SAPS* with the test data (Soundex was removed because it was always the worst). The result showed that *Adaptive SAPS* achieved a further improvement on the original *SAPS* as shown in Fig. 5.11.

With the Zobel's dataset, only the subsets B and C were used. The subset A has only 25 queries, not enough for the training and test. The 50 queries in each

Figure 5.9: *SAPS* with Zobel's B Dataset

subset were divided into 20 training data and 30 test data. In the training session, $SAPS(-4+2-1-1)$ and $SAPS(-1+2-1-2)$ were determined as the final adaptive models for B dataset and C dataset and examined in the test session. The results showed that *Adaptive SAPS* made an obvious improvement on the original *SAPS*. And it was very close to Editex with the B dataset (Fig. 5.12) and slightly better than Editex with the C dataset (Fig. 5.13).

The experiments with Adaptive SAPS fully tested the adaptive ability of the Syllable Alignment Pattern Searching model. SAPS with adaptive parameters can be easily trained to achieve an optimal performance with different datasets/judgement criteria. On the other hand, it was noticed that SAPS had the best performance on different datasets with different parameter combinations. The result proves our previous assumption that different datasets have different judgement preferences and only an adaptive method can achieve optimal performance with different datasets on the average.

Figure 5.10: *SAPS* with Zobel's C Dataset

Table 5.5: Overlap of FONS (COMPLETE Dataset)

	Editex	Edit Distance	Soundex
FONS	520(76.2%)	470(63.0%)	432(55.7%)

Multiple String Search

The FONS algorithm was compared with Soundex, Edit Distance and Editex with COMPLETE dataset. As expected, FONS sacrifices its accuracy for speed, but still has acceptable performance. The overlap examination in Table 5.5 shows that FONS and Edit Distance have 63.0% in common – there is still enough space to improve by combining FONS and approximate string search algorithm. When they are all running in linear time $O(n)$, the overall processing speed will be much faster.

To evaluate of the time efficiency of the multiple name search algorithm, the multiple search was compared with a sequential searching process, i.e., the sequential

Figure 5.11: *Adaptive SAPS* with COMPLETE Dataset

searching process searched the name strings one by one in the database with the FONS algorithm. The comparison were run on $m = 4, 6$ and 8 , considering the length of typical English surnames is 4 to 9. The experimental result is shown in Fig. 5.15, where $Acceleration\ Rate = \frac{Processing\ Time\ of\ Sequential\ Search}{Processing\ Time\ of\ Multiple\ Search}$. The result demonstrates that the multiple search algorithm achieved an obvious speed-up, e.g., with the group of length 4, it took less than 20% of the time required by its sequential peer. It slowed down as the name string length increased. This is reasonable since the re-verification cost will rise as the length of strings increase, thus lowering the time efficiency. For a large name string searching problem, an overall speedup of 3 to 5 times can be expected.

Figure 5.12: *Adaptive SAPS* with Zobel's B Dataset

5.7 Summary

While most of the existing methods convert a string into intermediate codes with phonetic transformation rules, SAPS, a novel algorithm to calculate the optimal pairing of the corresponding syllables in two strings is proposed in this chapter. Experimental results showed that it is very effective with phonetic string search.

The most important advantage of SAPS is its flexibility to modify its corresponding parameters to the given training data easily and thus achieves optimal performance with the new requirements. The effectiveness of the SAPS algorithm with adaptive parameters has been tested and results showed that its performance is very competitive on different datasets.

The time complexity of phonetic string search has also been studied. A fast algorithm called FONS is designed to run in linear time $O(n)$. Then it has been extended to a multiple version to search several phonetic strings simultaneously, thus accelerating the overall searching speed dramatically.

Figure 5.13: *Adaptive SAPS* with Zobel's C Dataset

Figure 5.14: RGP of FONS with COMPLETE Dataset

Figure 5.15: Performance of Multiple Name Search

Chapter 6

Conclusions

6.1 Conclusions

Computer Crime is also called E-Crime, Hi-Tech Crime or Electronic Crime. In computer crimes, all kinds of computer technologies, often involving the Internet, are used to commit crimes or to escape discovery and punishment. Its countermeasure, Computer Forensics, must have not only the most skillful manpower and the highest intellect, but also the most cutting-edge technologies.

In this thesis, we have defined a novel concept, “Case-Relevance”, to measure the importance of any information to a given criminal scenario reconstruction. The procedure of building an intelligence data analysis phase around “Case-Relevance” was demonstrated through the detailed discussion of two typical digital evidence investigation scenarios. The study fully shows the possibility and the benefit of binding computational intelligence with current computer forensic frameworks.

Email processing is an important topic in computer forensics. We examined the

existing email processing methods, focusing on their potential and limitations in computer forensics. A novel algorithm, “Case-Relevance” Email Evidence Extraction, has been specially designed to give investigators straightforward support and assistance. The method starts from limited number of information, then carries on an incremental search to bring more and more relevant information automatically. The method measures the relationship of information according to the predetermined case profiles and ranks the search results by their degree of “Case-Relevance”. Our proposed “Case-Relevance” Email Evidence Extraction method is a practical implementation of the concept of “Case-Relevance”. The experimental results showed that it is very effective in extracting case related information from email corpus.

Phonetic string search of written text is another important tool for computer forensics. The major difficulty for the previous methods is the inconsistent performances with different datasets. We have proposed a novel algorithm, Syllable Alignment Pattern Searching (SAPS), which calculates the optimal pairing of the corresponding syllables in two strings. The most important advantage of SAPS is its flexibility to adapt its corresponding parameters to the given training data easily and thus is very competitive with different datasets. This is the first algorithm of its kind as far as we know. Regarding the requirement for fast search speed in computer forensics, we proposed a linear algorithm, FONS, which runs in time $O(n)$ and could be extended to a multiple string search algorithm. Thus it can shorten the search time dramatically.

6.2 Future Work

The methods we have proposed require a lot more work to see their full potential. There are some future research plans in our work schedule.

Phonetic String Search The syllable alignment pattern searching algorithm (SAPS) proposed in this thesis has been proven to be very effective in phonetic string search. We are going to further improve it. Its *Preprocessing* step deals with the letter combinations to get more accurate results and some simplification in the following segmentation step, which is very important to the overall performance. The same preprocessing step of Phonix is used in SAPS currently, which works very well. But considering its original design philosophy in Phonix is to get more accurate phonetic codes, we believe a careful check and redesign work on its rules is able to achieve more effective segmentation.

We plan to apply the SAPS algorithm to multi-language environments, especially with Asian languages. In machine translation of Asian languages, phonetic words are transformed into strings in Roman alphabet first (for example, Han Yu Pin Yin) and compared with a string search algorithm. The flexibility of SAPS is a great advantage here in that it can be trained to work with different languages. And as the phonetics of most Asian languages is more syllable-based, SAPS is expected to obtain better result than other methods previously used in machine translation.

Email Processing We used a personal email dataset in developing the “Case-Relevance” Email Evidence Extraction algorithm. We plan to explore its potential with a more complex environment and look for any further improvement. The Enron Email dataset can be used to study the complex email behaviour in a large business company. It will be an ideal dataset for our research.

Computer Forensic Framework We will do more research on computer forensic frameworks and improve the concept of “Case-Relevant”. Currently we only use the

“Case-Relevant” concept in the data analysis phase. We will look further at how to deploy this concept in other phases and how to build such a intelligent system for other investigation scenarios.

Dataset for Computer Forensic Research The interest will be to use openly available data to simulate the features or requirement of typical criminal scenarios. We believe one major obstacle for the deployment of computational intelligence in computer forensics is the lack of *Standard Test Datasets and Evaluation Criteria*. Computational intelligence technology is very expensive, requiring a large bunch of data for the training, performance tweaking and test. But the fear of breaking privacy or leakage of other sensitive information, restricts the data to law enforcement only and researchers from third parties are not able to access them. We noticed some similar obstacles in the research in Intrusion Detection. But they managed to work out some standard datasets and testbeds and are still working to improve them, such as the work in [73, 74]. These datasets are now available to everyone from the DARPA project website and Knowledge Discovery and Data Mining community. A proposal [55] has been made to formalize the test and evaluation of different computer forensic products. It is more urgent to establish a formal test dataset and evaluation environment for the Data Analysis Phase. Raw data from selected cases should be re-organized and filtered, adding some manually created events if needed; all the events should be carefully examined, located, categorized and listed; the private or sensitive information should be removed. Then the dataset could be published and made open to researchers.

Some open competitions (like taking the blind-test form at the Message Understanding Conference supported by DARPA) are also suggested to attract interest

CHAPTER 6. CONCLUSIONS

98

from experts and researchers from governments, law enforcement, industry and other third parties. All of these will boost research and implementation in both areas of computer forensics and computational intelligence.

Bibliography

- [1] 20 newsgroups. <http://people.csail.mit.edu/people/jrennie/20Newsgroups/>.
- [2] Enron email corpus. <http://www.cs.cmu.edu/enron/>.
- [3] The porter stemming algorithm. <http://www.tartarus.org/martin/PorterStemmer/index.html>.
- [4] Uc berkeley enron email analysis project. http://bailando.sims.berkeley.edu/enron_email.html.
- [5] American-Heritage-Dict. *The American Heritage Dictionary of the English Language, Fourth Edition*. Houghton Mifflin Company, 2000.
- [6] Rie Kubota Ando and Lillian Lee. Iterative residual rescalin. *ACM Press*, pages 152–162, 2001.
- [7] A. Apostolica and C. Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:315–336, 1987.
- [8] R. Baeza-Yates. *Efficient Text Searching*. PhD thesis, University of Waterloo, 1989.

- [9] R. Baeza-Yates. Some new results on approximate string matching. In *Workshop on Data Structures*, Dagstuhl, Germany, 1991.
- [10] R. Baeza-Yates. Text retrieval: Theory and practice. In J. van Leeuwen, editor, *12th IFIP World Computer Congress*, volume I of *Algorithms, Software, Architecture*, pages 465–476, Madrid, Spain, September 1992. Elsevier Science. September.
- [11] R. Baeza-Yates. A unified view of string matching algorithms. In *Proceedings of the Theory and Practice of Informatics (SOFSEM '96)*, volume 1175 of *LNCS*. Springer-Verlag, Berlin, 1996.
- [12] R. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Communications of the ACM*, (35):74–82, Oct 1992.
- [13] R. Baeza-Yates and G. Navarro. A faster algorithm for approximate string matching. In *Proceeding of Combinatorial Pattern Matching (CPM'96)*, Irvine, CA, June 1996.
- [14] R. Baeza-Yates and G. Navarro. Multiple approximate string matching. In *Proc. WADS'97*, volume 1272 of *LNCS*, pages 174–184, 1997.
- [15] R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.
- [16] O. Balter. *Electronic Mail in a Working Context*. PhD thesis, Royal Institute of Technology, Stockholm., 1998.
- [17] O. Balter. Keystroke level analysis of email message organization. In *CHI 2000*, pages 105–112. ACM Press New York, NY, USA, 2000.

- [18] Robert Barrett and Ted Selker. Aim: A new approach for meeting information needs. Technical report, IBM Almaden Research Center, Almaden, CA, 1995.
- [19] Venansius Baryamureeba and Florence Tushabe. The enhanced digital investigation process model. In *Digital Forensics Research Workshop (DFRWS)*, Baltimore, Maryland, August 2004.
- [20] Nicole Lang Beebe and Jan Guynes Clark. A hierarchical, objectives-based framework for the digital investigations process. In *Digital Forensics Research Workshop (DFRWS)*, Baltimore, Maryland, August 2004.
- [21] Ron Bekkerman. Document classification on enron email dataset. http://www.cs.umass.edu/~ronb/enron_dataset.html.
- [22] V. Bellotti, N. Duchenaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *Computer Human Interaction*. ACM Press New York, NY, USA, 2003.
- [23] Pitney Bowes. Increased use of electronic communications tools among north american and european workers. press release, August 2000.
- [24] Vlasti Broucek and Paul Turner. Forensic computing: Developing a conceptual approach for an emerging academic discipline. In *The 5th Australian Security Research Symposium*, Perth, Australia, 2001.
- [25] Danilo Bruschi, Mattia Monga, and Lorenzo Martignoni. how to reuse knowledge about forensic investigations. In *Digital Forensics Research Workshop (DFRWS)*, Baltimore, Maryland, August 2004.

- [26] Brian Carrier and Eugene H. Spafford. Getting physical with the digital investigation process. *International Journal of Digital Evidence*, 2(2):1–20, Fall 2003.
- [27] D. L. Carter. Computer crime categories, how techno-criminals operate. *FBI Law Enforcement Bulletin*, July 1995.
- [28] Eoghan Casey. *Digital Evidence And Computer Crime: Forensic Science, Computers and the Internet*. Academic Press, 2000.
- [29] W. Chang and J Lampe. Theoretical and empirical comparisons of approximate string matching algorithms. In *the 3d Annual Symposium on Combinatorial Pattern Matching (CPM '92)*, volume 644 of *LNCS*, pages 172–181. Springer-Verlag, Berlin, 1992.
- [30] W. Chang and E. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.
- [31] W. Chang and T. Marr. Approximate string matching and local similarity. In *the 5th Annual Symposium on Combinatorial Pattern Matching (CPM '94)*, volume 807 of *LNCS*, pages 259–273. Springer-Verlag, Berlin, 1994.
- [32] Hsinchun Chen, Wingyan Chung, Yi Qin, Michael Chau, Jennifer Jie Xu, Gang Wang, Rong Zheng, and Homa Atabakhsh. Crime data mining: An overview and case studies. In *The National Conference on Digital Government Research (dg.o2003)*, May 18-21 2003.

- [33] William W. Cohen. Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data. <http://minorthird.sourceforge.net>, 2004.
- [34] W.W. Cohen. Learning rules that classify email. In *the 1996 AAAI Spring Symposium on Machine Learning and Information Access*, pages 115–123, San Francisco, CA, 1996. Morgan Kaufmann Publishers Inc.
- [35] R. Cole and R. Hariharan. Approximate string matching: a simpler faster algorithm. In *the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, pages 463–472, 1998.
- [36] Alex Cozzi, Tom Moran, and Clemens Drews. The shared checklist: Reorganizing the user experience around unified activities. In *10th Intl Conf on Human-Computer Interaction (INTERACT 2005)*, September 2005.
- [37] G. Das, R. Fleisher, L. Gasieniek, D. Gunopulos, and J. Karkainen. Episode matching. In *the 8th Annual Symposium on Combinatorial Pattern Matching (CPM '97)*, volume 1264 of *LNCS*, pages 12–27. Springer-Verlag, Berlin, 1997.
- [38] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proc. of 10th intl. conf. on Intelligent User Interfaces*, pages 75–82. ACM Press New York, NY, USA, 2005.
- [39] Mark Dredze, Tessa Lau, and Nicholas Kushmerick. Automatically classifying emails into activities. In *IUI '06: Proc. of the 11th intl. conf. on Intelligent User Interfaces*, 2006.

- [40] N. Duchenaut and V. Bellotti. Email as habitat. volume 8, pages 30–38. ACM Press New York, NY, USA, 2001.
- [41] N. Duchenaut and V. Bellotti. A study of email work activities in three organizations (working paper): Parc, inc. 2001.
- [42] Klas Erikson. Approximate swedish name matching - survey and test of different algorithms. Master’s thesis, School of Engineering Physics, 1997. Nada report TRITA-NA-E9721.
- [43] D. Fisher and P. Moody. Studies of automated collection of email records. Technical Report UCI-ISR-02-4., University of California Irvine, 2001.
- [44] US Department fo Justice (DOJ). Federal guidelines for searching and seizing computers. http://www.usdoj.gov/criminal/cybercrime/search_docs/toc.htm, 1994.
- [45] Forensic Focus. Computer forensics software, an introduction. <http://www.forensicfocus.com/computer-forensics-software-intro.php>, September 2004.
- [46] WaiteGenealogy Research Forum. *Waite Genealogy Research Forum Surname List*. <http://www.waitegenealogy.org/TheDatabases/names.htm>.
- [47] AccessData Forensic Toolkit (FTK). <http://www.accessdata.com>.
- [48] T. N. Gadd. ‘fishing fore werds’: Phonetic retrieval of written text in information systems. *Program: automated library and information systems*, 22(3):222–237, 1988.

- [49] T. N. Gadd. Phonix: The algorithm. *Program: automated library and information systems*, 24(4):363–366, 1990.
- [50] Gartner. Trends in email usage, 2001.
- [51] R. Giegerich, S. Kurtz, F. Hischke, and E. Ohlebusch. A general technique to improve filter algorithms for approximate string matching. In *the 4th South American Workshop on String Processing (WSP '97)*, pages 38–52. Carleton Univ. Press, 1997.
- [52] Ruibin Gong and Tony Kai Yun Chan. A fast algorithm for online multiple name searching. In *Proceedings of the 2nd IASTED International Conference on Communications, Internet & Information Technology (CIIT2003)*, Scottsdale, AZ, Nov. 2003.
- [53] Gregory A. Hall and Wilbon P. Davis. Toward defining the intersection of forensics and information technology. *International Journal of Digital Evidence*, 4(1), Spring 2005.
- [54] P.A.V. Hall and G.R. Dowling. Approximate string matching. *Computing Surveys*, 12(4):381–402, 1980.
- [55] Mark Hirsh. Proposal to formalize test and evaluation activities within the forensic and law enforcement communities. In *Digital Forensics Research Workshop (DFRWS)*, Baltimore, Maryland, August 2004.
- [56] Victoria J. Hodge and Jim Austin. An evaluation of phonetic spell checkers. Technical report, Department of Computer Science, University of York, 2001.

- [57] David Holmes and M. Catherine McCabe. Improving precision and recall for soundex retrieval. In *Proceedings of International Conference on Information Technology: Coding and Computing*, Las Vegas, Nevada, April 08-10 2002.
- [58] Y. Huang, D. Govindaraju, T. Mitchell, V. Rocha, de Carvalho, and W. Cohen. Inferring ongoing activities of workstation users by clustering email. In *Proc. of the 1st Conf. on Email and Anti-Spam*, July 2004.
- [59] Kevin Hughes. Hypermail 1.02. <http://www.landfield.com/hypermail/>, 1995.
- [60] Cadiz J. J., Dabbish L., Gupta A., and Venolia G. D. Supporting email workflow. Technical Report MSR-TR-2001-88, Microsoft Research, 2001.
- [61] M.A. Jaro. Unimatch: A record linkage system: User's manual. Technical report, U.S. Bureau of the Census, Washington, DC, 1976.
- [62] Tarhio J. Jokinen, P. and E. Ukkonen. A comparison of approximate string matching algorithms. *Software Practice Exper.*, 26(12):1439–1458, 1996.
- [63] K. E. Jones. Vm user's manual. second edition, vm version 5. ftp.uu.net in the networking/mail/vm/ directory, 1991.
- [64] R. Khoussainov and N. Kushmerick. Email task management: An iterative relational learning approach. In *Proc. Conf. Email and Anti-Spam*, 2005.
- [65] Sun Kim and Yanggon Kim. A fast multiple string-pattern matching algorithm. In *Proc. of 17th AoM/IAoM Conference on Computer Science*, August 1999.

- [66] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, Pisa, Italy, 2004.
- [67] K. Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, 1992.
- [68] Josu Kuri and Gonzalo Navarro. Fast multipattern search algorithms for intrusion detection. In *7th International Symposium on String Processing and Information Retrieval(SPIRE'00)*, 1999.
- [69] N. Kushmerick and T. Lau. Automated email activity management: an unsupervised learning approach. In *IUI '05: Proc. of the 10th intl. conf. on Intelligent User Interfaces*, pages 67–74. ACM Press, 2005.
- [70] G. Landau and Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, (37):63–78, 1988.
- [71] V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Probl. Inf. Transmission*, 1:8–17, 1965.
- [72] David D. Lewis and Kimberly A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 33(2):209–217, 1997.
- [73] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation.

- In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, volume 2, 2000.
- [74] Richard P. Lippmann, Joshua W. Haines and David J. Fried, Jonathan Korba, and Kumar Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, Lecture Notes In Computer Science, pages 162–182, 2000.
- [75] Levitt M. Email usage forecast and analysis, 2000-2005. Technical Report IDC Report W23011., 2000.
- [76] W. Masek and M. Paterson. A faster algorithm for computing string edit distances. *J.Comput. Syst. Sci.*, (20):18–31, 1980.
- [77] R. McKemmish. What is forensic computing. *Trends and Issues in Crime and Criminal Justice*, 118, 1999.
- [78] B. Melichar. String matching with k differences by finite automata. In *the International Congress on Pattern Recognition (ICPR '96)*, pages 256–260, Silver Spring, MD., 1996. IEEE CS Press.
- [79] Kenrick Mock. Dynamic email organization via relevance categories. In *the 11th IEEE International Conference on Tools with Artificial Intelligence*, page 399. IEEE Computer Society, 1999.
- [80] George Mohay, Alison Anderson, Byron Collie, Oliview de Vel, and Rodney McKemmish. *Computer and Intrusion Forensics*. Artech House, Boston, 2003.

- [81] Thomas P. Moran, Alex Cozzi, and Stephen P. Farrell. Unified activity management: Supporting people in ebusiness. *Communications of the ACM*, 2005.
- [82] M. J. Muller, W. Geyer, B. Brownholtz, E. Wilcox, and D. R. Millen. One-hundred days in an activity-centric collaboration environment based on shared objects. In *CHI04: Proc. of the SIGCHI conference on Human factors in computing systems*, pages 375–382. ACM Press , NY, USA, 2004.
- [83] S. Muresan, E. Tzoukerman, and J. Klavans. Combining linguistic and machine learning techniques for email summarization. In *CoNLL Workshop at ACL-EACL 2001*, 2001.
- [84] R. Muth and U. Manber. Approximate multiple string search. In *Proc. Of CPM' 96*, pages 75–86, 1996.
- [85] G. Myers. Incremental alignment algorithms and their applications. Technical Report 86-22, Dept. of Computer Science, Univ. of Arizona, 1986.
- [86] G. Myers. An $o(nd)$ difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.
- [87] Gonzalo Navarro. Multiple approximate string matching by counting. In *Proceedings of the 4th South American Workshop on String Processing*, 1997.
- [88] Gonzalo Navarro, Ricardo Baeza-Yates, and Joao Marcelo Azevedo Arcoverde. Matchsimile: a flexible approximate matching tool for searching proper names. *Journal of the American Society for Information Science and Technology*, 54(1):3–15, 2003.

- [89] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48:444–453, 1970.
- [90] Ani Nenkova and Amit Bagga. Facilitating email thread access by extractive summary generation. In *RANLP*, pages 287–296, 2003.
- [91] Paula S. Newman and John C. Blitzer. Summarizing archived discussions: a beginning. In *Intelligent User Interfaces*, pages 273–276, 2003.
- [92] Stephen Northcutt. *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing, 1999.
- [93] Gary Palmer. A road map for digital forensics research - report from the first digital forensics research workshop (dfrws). Technical Report DTR-T001-01 Final, Air Force Research Laboratory, Rome Research Site, 2001.
- [94] D. Parker. *Crime by Computer*. NY: Charles Scribners' and Sons, New York, 1976.
- [95] D. Parker. *Fighting Computer Crime*. NY: Charles Scribners' and Sons, New York, 1983.
- [96] D. Parker. *Fighting Computer Crime: A New Framework for Protecting Information*. NY: John Wiley & Sons, New York, 1998.
- [97] Ulrich Pfeiffer, Thomas Poersch, and Norbert Fuhr. Retrieval effectiveness of proper name search methods. *Information Processing & Management*, 32:667–679, 1996.

- [98] Mark Pollitt. A framework for digital forensic science. In *Digital Forensics Research Workshop (DFRWS)*, Baltimore, Maryland, August 2004.
- [99] J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27:358–368, 1984.
- [100] Owen Rambow, Lokesh Shrestha, John Chen, and Christy Laurdisen. Summarizing email threads. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Short Papers*, pages 105–108, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.
- [101] Mark Reith, Clint Carr, and Gregg Gunsch. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3):1–12, 2002.
- [102] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. Research Report CS-TR-532-96, Department of Computer Science, Princeton University, October 1996.
- [103] M. T. Rose. *The Internet Message: Closing the Book with Electronic Mail*. Prentice Hall, Eaglewood cliffs, NJ, 1993.
- [104] Gerard Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.
- [105] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. NY: McGraw-Hill Book Company, 1983.
- [106] D. Sankoff and J. Kruskal. *TimeWarps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.

- [107] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, Volume 2 , Issue 2 (May 1999)(ISSN:1094-9224):159 – 176, 1999.
- [108] Richard Segal and Jeffrey O. Kephart. Incremental learning in swiftfile. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 863–870, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [109] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *J. Algor.*, 1:359– 373, 1980.
- [110] Jitesh Shetty. Enron dataset. <http://www.isi.edu/adibi/Enron/Enron.htm>.
- [111] F. Shi. Fast approximate string matching with q-blocks sequences. In *the 3rd South American Workshop on String Processing (WSP'96)*, pages 257–271. Carleton Univ. Press, 1996.
- [112] Encase Forensic Software. <http://www.encase.com>.
- [113] Vogon Forensic Software. <http://www.vogon-international.com>.
- [114] Peter Sommer. Intrusion detection systems as evidence. In *Recent Advances in Intrusion Detection - RAID'98*, Louvain-la-Neuve, Belgium, 1998.
- [115] A. Surendran, J. Platt, and E. Renshaw. Automatic discovery of personal topics to organize email. In *Proc. of the 2nd Conf. on Email and Anti-Spam*, July 2005.

- [116] J. Sutinen, E. and Tarhio. On using q-gram locations in approximate string matching. In *the 3rd Annual European Symposium on Algorithms (ESA '95)*, volume 979 of *LNCS*, pages 327–340. Springer-Verlag, Berlin, 1995.
- [117] T. Takaoka. Approximate pattern matching with samples. In *ISAAC '94*, volume 834 of *LNCS*, pages 234–242. Springer-Verlag, Berlin, 1994.
- [118] J. Tarhio and E. Ukkonen. Approximate boyer-moore string matching. *SIAM J. Comput.*, 22(2):243–260, 1993.
- [119] Paul Thompson and Christopher C. Dozier. Name searching and information retrieval. In *Proceedings of Second Conference on Empirical Methods in Natural Language Processing*, Providence, Rhode Island, 1997.
- [120] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, (64):100–118, 1985.
- [121] E. Ukkonen. Finding approximate patterns in strings. *J. Algor.*, 6:132–137, 1985.
- [122] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 1:191–211, 1992.
- [123] G. Venolia, A. Gupta, J.J. Cadiz, and L. Dabbish. Supporting email workflow. Technical Report MSR-TR-2001-88, Microsoft Technical Report, 2001.
- [124] V. V. Raghavan, P. Raghavan, and G. S. Jung. Retrieval system evaluation using recall and precision: Problems and answers. In N. Bellin; C. J. Van Rijsbergen., editor, *Proceedings of the Twelfth Annual International ACM SIGIR Conference*

- on Research and Development in Information Retrieval*, pages 59–68, New York, 1989. ACM.
- [125] Mackay W. More than just a communication system: Diversity in the use of electronic mail. In *Computer Supported Cooperative Work*, pages 344–353. ACM Press New York, NY, USA, 1988.
- [126] Gang Wang, Hsinchun Chen, and Homa Atabakhsh. Automatically detecting deceptive criminal identities. *Communications of the ACM*, Volume 47 Issue 3(0001-0782):70 – 76, March 2004.
- [127] Q. Whittaker, S. Jones and L. Terveen. Persistence and conversation stream management: Conversation and contact management. In *HICCS02*. IEEE Press, New York., 2002.
- [128] S. Whittaker and C. Sidner. Email overload: Exploring personal information management of email. In *ACM CHI96 Conference on Human Factors in Computing Systems*, pages 276–283, 1996.
- [129] Steve Whittaker. Supporting collaborative task management in email. *Human-Computer Interaction*, 1&2(20), 2005.
- [130] the free encyclopedia Wikipedia. <http://en.wikipedia.org/wiki/>.
- [131] S. Wu and U. Manber. Agrep-a fast approximate pattern-matching tool. In *USENIX Technical Conference*, number 10, pages 153–162, Berkeley, CA, USA, 1992. USENIX Association.
- [132] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.

- [133] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical report, Computer Science Department, University of Arizona, 1994.
- [134] Liam H. Wugshile and Jos Leubnitz. Using a pronunciation dictionary and phonetic rules for name matching applications. Technical report, Computing Science, RMIT University, 2001.
- [135] Justin Zobel and Philip Dart. Fnetik: An integrated system for phonetic matching. Technical Report 96-66, Department of Computer Science, RMIT, 1996.
- [136] Justin Zobel and Philip Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th International Conference on Research and Development in Information Retrieval*, 1996.

Appendix A

Experiment Data of SAPS

A.1 Query List of COMPLETE Dataset

ABRAHAM, ADAMS, AHRENS, AIGNER, ANDERSEN, BAYER, BECKER, BENNETT, BERG, BLOCH, BLOOD, BOWDEN, BOYCE, BRANDT, BROWN, BUCKLEY, CARTER, CHIANG, CHOW, CHRISTIAN, CONRAD, DAHL, DAMEROW, DANIELL, DAVIDSON, DIXON, ECCLES, ENGEL, FORMAN, FRAENKEL, GEHRMANN, GEISSLER, GOLDEN, GREENBERGER, GROSS, GRUEN, HOFFMANN, JACOBSON, JENSEN, JOHNSON, KLEIN, KOCH, KRAMER, KREMERS, LAMPE, LEHMANN, LEISSNER, LEWIS, MARKS, MCCULLOUGH, MCDOWELL, MEYER, MUELLER, NEUMANN, OBRIAN, PEDERSEN, PERL, PHILIPSON, PICKARD, PRINZ, REECE, REINER, REYNA, RICE, ROGERS, ROSENKRANZ, SCHAEFER, SCHILLER, SCHMIDT, SCHNEIDER, SCHROEDER, SMITH, STEFFENS, STINE, STRASSMANN, TANNENBAUM, TAYLOR, THIELE, VANRIJSBERGEN, VOIGT, WAGNER, WALLACE, WANG, WEBER, WEISS, WELCH, WHITE, WIRTH, YANG, ZWEIG

A.2 Returns of SAPS with COMPLETE Dataset

Format: *#Query Term, Rank Number+return1, Rank Number+return2,.....*

1. #ABRAHAM, 0abraham, 3astrahan, 3abrahams, 5wareham, 5allaham, 5abrash, 6ingraham, 7ibramsha, 7agrawal, 7acree, 8hollihan, 8holahan, 8aldrich, 9ulbrich, 9ozkarahan, 9olbrich, 9eckbreth, 9abramson
2. #ADAMS, 0adams, 2adensam, 2adamson, 3edens, 4anders, 4adelson, 4addison, 6andersson, 6anderson, 6andersen, 7edison, 7edelson, 7eddison, 8waltons, 8ostenso, 8hestenes, 8ammons, 8agins, 8addink, 9siddons, 9mydans, 9hastings, 9dinse, 9dance, 9atkins, 9alonso, 9ahrens, 9adrams, 9adam, 9abrams
3. #AHRENS, 0ahrens, 2adrams, 2abrams, 4aronson, 4abramson, 5irons, 5harendza, 5arazi, 5andrews, 6henricks, 6alonso, 6ahlers, 6abrahams, 7willians, 7williams, 7wellings, 7horacek, 7hendricks, 7aretz, 7andriesen, 7ambrosio, 8willetts, 8willekes, 8wilenzick, 8verenes, 8ollins, 8henriksen, 8heinrichs, 8alex, 8ahring
4. #AIGNER, 0aigner, 3wegner, 3wagner, 3axner, 4wiegnier, 4weigner, 4arnow, 4agnew, 4abnor, 4abdnor, 5wossner, 5wisner, 5winner, 5winer, 5wilner, 5wiener, 5wesner, 5wenner, 5weiner, 5wayner, 5hubner, 5hofner, 5hiner, 5hilner, 5hehner, 5hafner, 5eissner, 5eisner, 5aymar, 5awoniyi
5. #ANDERSEN, 0andersson, 0anderson, 0andersen, 4henderson, 4anders, 6adensam, 6adelson, 6addison, 6adamson, 8watterson, 9sanderson, 9gunderson, 9edison, 9edelson, 9eddison
6. #BAYER, 0buyer, 0boyer, 0beyer, 0bayer, 0bauer, 0baier, 2bier, 2berr, 2beer, 2baur, 2barr, 2baer, 2baehr, 3poore, 3peier, 3burney, 3burdea, 3buerry, 3broeer, 3briere, 3breuer, 3brauer, 3bowyer, 3bowler, 3bowker, 3borrow, 3booker, 3boemer, 3boeger, 3boecker

7. #BECKER, 0becker, 0bakker, 0baker, 1bunker, 1bowker, 1booker, 1bonker, 1boecker, 1bobker, 1bieker, 1banker, 1baecker, 2beenker, 2barker, 3peker, 3packer, 3blecker, 3binkow, 3beckert, 4picouet, 4nebeker, 4bulkeley, 4buckle, 4bucken, 4brooker, 4brinker, 4boku, 4bocoex, 4bican, 4bhasker
8. #BENNETT, 0bonnette, 0bennett, 0benett, 0banet, 0banat, 1boinet, 1benoit, 2bynun, 2bynum, 2bynon, 2burnett, 2bunnin, 2bunnell, 2bouknight, 2bono, 2bona, 2benac, 2barnett, 2barnat, 2baney, 3pannett, 3bunner, 3bucknall, 3brunet, 3boznak, 3bosnak, 3bonner, 3boner, 3bicknell, 3bannar
9. #BERG, 0burge, 0burdge, 0berg, 1bircg, 2burgin, 2bergan, 3burgoon, 3burgess, 3burger, 3bergius, 3berger, 4perng, 4osberg, 4buesing, 4buerger, 4bowling, 4bourgein, 4bosarge, 4boecking, 4bettge, 4beegan, 4auberg, 5wedberg, 5suuberg, 5silberg, 5rowberg, 5romberg, 5rettberg, 5regberg, 5ramberg
10. #BLOCH, 0bloch, 0blache, 1blanche, 1blacha, 2blue, 2beelich, 3brush, 3blaha, 3baligh, 4polash, 4broksch, 4briesch, 4brauch, 4bracchi, 4blumhoff, 4blokesch, 4blachier, 4barash, 4bailie, 5braasch, 5blaszczak, 5blackwell, 6plueschau, 6brockhouse, 6brecher, 6brayshaw, 6brasher, 7prucha, 7preusch, 7pawlyschyn
11. #BLOOD, 0blood, 0bleed, 2blough, 2bloom, 2bliss, 2bley, 2blau, 2blatt, 3bulow, 3bulaon, 3broad, 3breed, 3braid, 3bloor, 3bloess, 3blodgett, 3blair, 3biller, 3below, 3bellow, 3beller, 3belien, 3ballow, 3ballou, 3ballay, 4buttler, 4butler, 4buelow, 4buckley, 4buckler
12. #BOWDEN, 0bowden, 0boudin, 0bawden, 1burden, 1bowlden, 1bouldin, 1borden, 1beaudin, 1barden, 2bodin, 2boden, 2baudet, 3burdick, 3burdette, 3burdea, 3burda, 3broyden, 3boyton, 3bouton, 3borodin, 3berdick, 3beaudette, 3bauder, 3bardack, 4purdum, 4purdum, 4burton, 4bundy, 4budea, 4buckemden

13. #BOYCE, 0busse, 0buss, 0boyse, 0boyce, 0bois, 0boies, 0boaz, 0bays, 0bayes, 0bause, 0baus, 0bass, 2buse, 2bui, 2buhr, 2buell, 2boyle, 2boyd, 2bose, 2bos, 2boone, 2boon, 2boom, 2booke, 2book, 2boek, 2boecke, 2biese, 2biel, 2bic
14. #BRANDT, 0brent, 0brant, 0brandt, 1bryant, 2behrendt, 2baronti, 3brecht, 3brand, 3bracht, 4parente, 4parent, 4bronster, 4bronstein, 4bristol, 4brichta, 4bretl, 4brayton, 4blount, 4berend, 4balint, 5prentice, 5brundage, 5brownstein, 5brobston, 5brittain, 5brilliant, 5brandle, 5brandl, 5brandis, 5brandin
15. #BROWN, 0browne, 0brown, 0brawn, 3bruno, 3bruenner, 4brunet, 4breznay, 4brezany, 4brennen, 4brennan, 4brajnik, 4brahmi, 5brunner, 5bruening, 5browning, 5breuning, 5brenner, 5brebner, 5beranek, 6broeckmann, 6brainerd, 6brachman, 6blatny, 7presuhn, 7brotman, 7brodman, 7brightman, 7brickman, 7brechignac, 7bratman
16. #BUCKLEY, 0buckley, 0beckley, 1buckler, 1binkley, 2bulow, 2buelow, 2bigley, 2below, 2bellow, 2barclay, 2ballow, 2ballou, 2ballay, 2bailey, 2bagley, 3buttler, 3butler, 3bulaon, 3bowler, 3biller, 3bentley, 3beller, 3belien, 3beazley, 3beasley, 3bayliss, 3bayless, 3barlow, 4burnley, 4bullock
17. #CARTER, 0carter, 0cartaya, 1courter, 2kuester, 2koetter, 2koester, 2keiter, 2coultter, 2cartiere, 3mcartor, 3courtois, 4kotiah, 4koster, 4kester, 4kelter, 4karten, 4kantor, 4kanter, 4custer, 4curtis, 4curtice, 4cotter, 4coster, 4cater, 4caster, 4carta, 4canter, 5kostiur, 5korsten, 5kirsten
18. #CHIANG, 0chuang, 0chieng, 0chiang, 0cheung, 0cheong, 0cheang, 0chaing, 2chung, 2chong, 2ching, 2cheng, 2change, 2chang, 4cushing, 5kwong, 5chassaiing, 5catching, 6crehange, 6challenger, 7quiring, 7chickering, 7chattergy, 8cuiqing, 8schaump, 9khujumtschjan, 9huang, 9chien, 9chean, 9charnes, 9chain

19. #CHOW, 0chua, 0chow, 0chou, 0choo, 0choi, 0chiu, 0chau, 0chao, 0chai, 1choate, 1chiou, 1chien, 1chess, 1cheek, 1chean, 1chait, 1chain, 2khan, 2chute, 2chun, 2chuk, 2chu, 2cho, 2chinn, 2chin, 2chi, 2chen, 2check, 2chase, 2chan
20. #CHRISTIAN, 0christian, 2crestin, 3krieglstein, 3cranston, 4krystal, 4kristol, 4kristoff, 4crostack, 4cristy, 4crater, 4christy, 4christensen, 5korostoff, 6christopher, 6christ, 7kurita, 7kraidin, 7crider, 7clinton, 7clifton, 7clayton, 7chroust, 7christie, 8kraft, 8korostynski, 8kodratoff, 8klosterman, 8kallestad, 8croft, 8cloutier
21. #CONRAD, 0konrad, 0conrad, 2conroy, 2conrow, 3kurrat, 3kulsrud, 3kendrick, 3currid, 3conrath, 3carrette, 3cantrell, 4kurek, 4kouril, 4korol, 4koren, 4korel, 4korane, 4kero, 4keery, 4karon, 4karim, 4karem, 4karam, 4cory, 4coro, 4corey, 4cockrill, 4cary, 4caron, 4carey
22. #DAHL, 0dahl, 1diehl, 1deily, 2doulis, 2dooley, 2dahlin, 3dulli, 3douglas, 3darlin, 3darley, 3daly, 3dally, 3dali, 4tayli, 4podehl, 4lodahl, 4dunlop, 4dunlap, 4dulley, 4duley, 4dudley, 4douglass, 4donnelly, 4donlin, 4donlan, 4donally, 4dolev, 4dolan, 4ditlea, 4dillon
23. #DAMEROW, 0damerow, 0damerau, 2demery, 2demari, 3dimitroff, 3demilio, 5dromerick, 5demolombe, 5demillo, 5demillo, 5demelo, 6dimitriadis, 8tenery, 8ramarao, 8pomeroy, 8komarow, 8donnelly, 8donally, 8disario, 8danuloff, 9tsesmelis, 9tanoira, 9kemmerer, 9kammerer, 9dimaio, 9demeter, 9damer
24. #DANIELL, 0danielle, 0daniell, 0daniel, 2dunnion, 2danner, 2daney, 3dowmar, 3domnisse, 3disney, 3diener, 3didner, 3dauner, 3daniels, 3dabney, 4mcdaniel, 4duna, 4dorner, 4donat, 4dishneau, 4dinan, 4dennis, 4denit, 4dashner, 4darnell, 5tunney, 5tinney, 5tenney, 5tener, 5tanner, 5tanii

25. #DAVIDSON, 0davidson, 2davison, 4davids, 8stevenson, 8devaux, 9tavares, 9donaldson
26. #DIXON, 0dixon, 1derksen, 2dickson, 2deason, 2dawson, 3diersen, 4dyson, 4doucet, 4dorsey, 4dorsett, 4dodson, 4dix, 4dietzel, 4dauncey, 5driessen, 5dreizen, 5dockser, 5dineson, 5dickinson, 5deux, 5dennison, 5denison, 5davidson, 6pederson, 6pedersen, 6dykes, 6dussa, 6donaldson, 6ditzel
27. #ECCLES, 0eccles, 6essex, 6elkins, 7erkens, 8eckerson, 9eyres, 9engels, 9echols, 9atkins, 9akers
28. #ENGEL, 0engle, 0engell, 0engel, 3engels, 3angell, 4eng, 5wige, 5wagle, 5ogle, 5engelmann, 5edgar, 5eager, 5angus, 6winget, 6wingate, 6weigle, 6unger, 6orgill, 6erenguc, 6engelen, 6egelko, 6argyle, 6angeles, 7yago, 7yagi, 7witzgall, 7winger, 7wanger, 7utgoff, 7ong
29. #FORMAN, 0furman, 0fuhrman, 0forman, 1farbman, 2forthman, 2forstmann, 2fishman, 2feeman, 3fieldman, 3farmer, 4vorsmann, 4portmann, 4perlman, 4fleishman, 4finerman, 4fermazin, 4feldmann, 4feldman, 4federman, 4fajman, 4aufermann, 5pearlman, 5furnas, 5fujimoto, 5fujimaki, 5friedmann, 5friedman, 5freshman, 5freeman, 5freedman
30. #FRAENKEL, 0fraenkel, 2frenkel, 2frankle, 2frankel, 4franco, 5farouki, 6frenck, 6fredkin, 6franke, 6frank, 8prinsky, 9pronk, 9prickett, 9prakken, 9flaak
31. #GEHRMANN, 0gorman, 0germann, 0german, 0gehrmann, 1germain, 2guthmann, 2goodman, 2geisman, 2gayman, 3ogorman, 3giesemann, 4wiggermann, 4segermann, 4jurmain, 4ingerman, 4guzman, 4gutzmann, 4guttman, 4guttman, 4guterman, 4gosmann, 4goldmann, 4goldman, 4gokmen, 4goffmann, 4goffman, 4giolma, 4giltman, 4gilman, 4giffman

32. #GEISSLER, 0giessler, 0geissler, 0geisler, 2goeller, 2gensler, 3guerler, 3gertler, 3gaengler, 4gurley, 4gittler, 4geller, 4garliepp, 4galler, 5giuliano, 5giebeler, 5ghormley, 5gehlert, 5gaylord, 5aguilar, 6guarraia, 6goulub, 6gotlieb, 6gorla, 6goley, 6goessl, 6godley, 6gilliam, 6geyelin, 6gerla, 6gawlik
33. #GOLDEN, 0golden, 2gosden, 2goodin, 2godden, 2gawdun, 3gordon, 3gold-
ing, 3gilder, 3gideon, 3gelder, 3galton, 4guida, 4gudac, 4goodeve, 4goodall, 4gold, 4gholston, 4gelston, 4geld, 4gedda, 4gazdag, 4gajda, 4gabaldon, 5gustin, 5guindi, 5guelde, 5grandine, 5gould, 5gooding, 5goldstone
34. #GREENBERGER, 0gruenberger, 0greenberger, 6greenberg, 9grundberg
35. #GROSS, 0gruss, 0grosse, 0gross, 2gruen, 2gros, 2groote, 2groom, 2groh, 2groat, 2grise, 2gries, 2grice, 2grey, 2grewe, 2greigg, 2greiff, 2greif, 2greene, 2green, 2greem, 2gray, 2grawe, 2grau, 2graeme, 2graeff, 2graefe, 2grace, 2gariess, 2gareiss, 3guryel
36. #GRUEN, 0gruen, 0groom, 0greene, 0green, 0greem, 0graeme, 2gruss, 2grosse, 2gross, 2groote, 2groh, 2groat, 2grimm, 2grey, 2grewe, 2greigg, 2greiff, 2greif, 2gray, 2grawe, 2grau, 2gramm, 2gram, 2graeff, 2graefe, 3guryel, 3grunin, 3gru-
man, 3griffin, 3grehan
37. #HOFFMANN, 0hofmann, 0hoffmann, 0hoffman, 1hoeffmann, 2hyman, 2homann, 2holman, 2hollmann, 2hohmann, 2hodgman, 2hipschman, 2hillman, 2hickman, 2heyman, 2hellmann, 2heimann, 2hehmann, 2heeman, 2heck-
man, 2hauptmann, 2hauptman, 2hamann, 2hallmann, 2hackmann, 3wopp-
man, 3shipman, 3schiffman, 3huisman, 3housman, 3holtmann
38. #JACOBSON, 0jacobson, 0jacobsen, 4jakobs, 4jacobs, 5juncosa, 7junkins, 7jinkins, 7jenkins, 9jungkontze

39. #JENSEN, 0jenssen, 0jensen, 0janzon, 0janzen, 0janssen, 0janson, 0jansen, 1johnson, 1johnsen, 2judson, 2joussen, 2jepson, 2jasen, 2jamsa, 2jackson, 3janczak, 4joseph, 4jones, 4johansson, 4johansen, 4jessup, 4james, 5jonassen, 5johns, 5jamison, 5jahns, 5gibson, 6kjeldsen, 6juncosa, 6juliussen
40. #JOHNSON, 0johnson, 0johnsen, 2joussen, 2jenssen, 2jensen, 2janzon, 2janzen, 2janssen, 2janson, 2jansen, 3johansson, 3johansen, 4judson, 4johns, 4jepson, 4jasen, 4jamsa, 4jahns, 4jackson, 5juliussen, 5josephson, 5jorgenson, 5jorgensen, 5jonassen, 5joerissen, 5janczak, 5jamison, 5jamieson, 5jacobson, 5jacobsen
41. #KLEIN, 0kleine, 0klein, 1kilian, 2klaus, 2klauk, 2klau, 2klass, 2clough, 2cloud, 2cline, 2clemm, 2clem, 2clay, 3koller, 3kliman, 3kelley, 3keller, 3kalow, 3kalau, 3cullen, 3crain, 3colomb, 3colluad, 3collom, 3collin, 3colin, 3clure, 3clear, 3classen, 3clasen
42. #KOCH, 0koch, 0cash, 1keuch, 2kuthe, 2koetsch, 2koelsch, 2kochen, 2kirsch, 2kichen, 2kee, 2karsh, 2karsch, 2coe, 2cochez, 2cashin, 2cashell, 3kunschek, 3kocher, 3kettschau, 3ketchum, 3ketcham, 3keogh, 3keith, 3kaye, 3kashyap, 3kaczor, 3cutshaw, 3clech, 4scacchi, 4nickisch
43. #KRAMER, 0kremer, 0kramer, 0crummer, 0cramer, 1kroemer, 1kreimer, 1kraemer, 1creamier, 2krechmer, 3kretschmer, 3kretchmar, 3kremers, 3klimmer, 3cremers, 3clymer, 4krummel, 4kroener, 4kreiner, 4kramek, 4crimi, 5krupman, 5karimi, 5collamore, 5caromel, 6kuramoto, 6kruessmann, 6kriegman, 6krasnow, 6creelman, 6christmann
44. #KREMERS, 0kremers, 0cremers, 4crimmins, 7clemons, 7clements, 7clemens, 9kremer, 9kramer, 9crummer, 9cramer, 9climenson, 9chalmers

45. #LAMPE, 0lumpe, 0lampe, 4lopez, 4lippitt, 4lippiis, 4lipis, 4lewpole, 4leppik, 4lepage, 4lappen, 5olympia, 5loeper, 5klimpel, 5clampett, 6libby, 6leyva, 6levy, 6levi, 6lanphar, 6lamport, 6lambiase, 6crump, 7schlaepi, 7rumpf, 7roppel, 7rappen, 7rajput, 7lyngbaek, 7lyngbaeck, 7lubove
46. #LEHMANN, 0luhmann, 0lohmann, 0lohman, 0leymann, 0lehmann, 0lehman, 0laymon, 0layman, 0lauman, 0lahman, 1liebmann, 1liebman, 1lehrman, 1laufmann, 2loomis, 2littman, 2lippmann, 2lippman, 2lipman, 2lichtman, 2lessmann, 2lemon, 2leishman, 2leemis, 2lattman, 2lammen, 3riemann, 3reimann, 3rehmann, 3raiman
47. #LEISSNER, 0leissner, 0leisner, 1luxner, 2lerner, 2leitner, 2lehner, 2laufner, 2larner, 3roesner, 3reisner, 3raisner, 3leschner, 3lechner, 3gleissner, 4loner, 4lindner, 4lessnau, 4langner, 5rosner, 5reiner, 5rayner, 5rainer, 5leonard, 5lahners, 5kleiner, 5klainer, 5gleixner, 5glassner, 5fliegner, 5flieder
48. #LEWIS, 0lewis, 2lewyn, 2lewit, 2lewin, 2lewi, 2lewan, 2lewak, 2lewack, 3luqi, 3ludwig, 3loewen, 3litwin, 3lasswell, 4lohaus, 4lockwood, 4leeuwen, 5rowen, 5rowan, 5lethaus, 5laquer, 5landwehr, 5lahey, 6roszag, 6rockwell, 6ridgway, 6radwan, 6olowacz, 6mulawa, 6luthi, 6luczak
49. #MARKS, 0marks, 1marchese, 2myers, 2morse, 2moewes, 2meeks, 2mears, 2max, 2maas, 3mooers, 3meyers, 3maddux, 4myerson, 4mohs, 4mikes, 4means, 4meads, 4maulucci, 4matthes, 4marson, 4marsan, 4marcelle, 4mairson, 5nix, 5nerses, 5minners, 5millers, 5meyerson, 5mercier, 5meisels

50. #MCCULLOUGH, 0mccullough, 0mcculley, 1mckinley, 1mckellar, 1mccoul-
lough, 1mccauley, 1mcauley, 2mccullen, 2mccollum, 2mccallum, 2mccalla,
2mcalear, 3mickela, 3mcauliffe, 4nikolai, 4nicolay, 4mccaffrey, 5mckerrow, 6ni-
colo, 6nicolas, 6nicola, 6mercurio, 6mcallister, 6mcalester, 7mcquillan, 7mc-
causland, 7mccarron, 7macallister, 8sculley, 8scalia
51. #MCDOWELL, 0mcdowell, 9mcdaniel
52. #MEYER, 0moyer, 0moore, 0meyer, 0meier, 0mayor, 0mayer, 0major, 0maire,
0maier, 2myre, 2myer, 2moir, 2mier, 2meyr, 2mayr, 2mayoh, 2marr, 2maer,
3neyer, 3nayar, 3murray, 3murley, 3muhler, 3mueller, 3morrow, 3morley,
3moorer, 3mooers, 3moeser, 3moeller
53. #MUELLER, 0muhler, 0mueller, 0moeller, 0mehler, 0mahler, 1meilleur,
1marler, 1marlar, 2mutchler, 2muller, 2moler, 2mockler, 2miller, 2millar, 2mel-
lor, 2mehlios, 3naylor, 3murley, 3morley, 3moorer, 3metzler, 3meixler, 3mau-
rer, 3marlow, 3malaiya, 3maccluer, 4muello, 4moulin, 4mottley, 4mostellor
54. #NEUMANN, 0newman, 0neumann, 0neuman, 0nauman, 1norman, 1niep-
mann, 2nordman, 2niemi, 2naman, 2nackman, 3noonan, 3naamad, 3maimone,
3maimon, 4nummy, 4nimit, 4nemec, 4neeshma, 4myrman, 5vannorman, 5nim-
mer, 5niemeyer, 5neumeyer, 5neidleman, 5needleman, 5musmann, 5mossman,
5mittman, 5mitman, 5milman
55. #OBRIAN, 0obrien, 0obrien, 0obrian, 0obrian, 4orin, 4orey, 5oregan, 5orear,
5opler, 5oldroyd, 5oblow, 5adrian, 6umbreit, 6oreagan, 6oral, 6obrecht, 6he-
brail, 6asprey, 6aspray, 6aroian, 6ambrogne, 6abraham, 7wegbreit, 7unruh,
7otromke, 7ostroff, 7osler, 7orrell, 7ororke, 7olin

56. #PEDERSEN, 0pederson, 0pedersen, 3pettersen, 3pettersen, 3peterson, 3pettersen, 3patterson, 3paterson, 4peterseim, 4pejtersen, 7peters, 8vandersypen, 8peltason, 9sanderson, 9pierson, 9petaccia, 9pearson, 9henderson, 9gunderson
57. #PERL, 0poehl, 0perl, 1pearl, 2pohl, 2pihl, 2perlis, 2parlow, 2pahl, 3proehl, 3podehl, 3pauli, 3partlow, 4powley, 4poulos, 4poulin, 4perry, 4paulus, 4paulin, 4pashley, 5purtilo, 5prahl, 5pouliot, 5polly, 5pilla, 5persily, 5perrone, 5perron, 5perrin, 5perras, 5pehler
58. #PHILIPSON, 0phillipson, 0philipson, 3felician, 4phillips, 4philips, 5faloutsos, 6flessau, 6fellows, 7verleysen, 7pilenzo, 7pallozzi, 7friesen, 7friesem, 7franken, 7felix, 8tollefson, 8rulfson, 8paoluzzi, 8kolovson, 8frentzen, 8flores, 8flecken, 9vanrossum, 9vanriessen, 9vallance, 9placey, 9palloks, 9pallats, 9frixione, 9fredriksen
59. #PICKARD, 0pickard, 4peickert, 5burkard, 6pinkerton, 6beckert, 8rickard, 8reckard, 8pollard, 8lockard, 8fukuda, 8burkart, 9seacord, 9riccardi, 9pulford, 9peker, 9pekarske, 9pandard, 9packer, 9foucault, 9buckemden, 9bickerstaff
60. #PRINZ, 0prinz, 3puritz, 3prince, 3franz, 4frantz, 5princen, 5pomeranz, 5fritz, 5behrenz, 6prausa, 6prasso, 6pomerantz, 6flanz, 6bruns, 6brons, 7verenes, 7prosis, 7prasad, 7frantz, 7franks, 7bruhns, 7brands, 8pronk, 8principi, 8presser, 8preikszas, 8prang, 8podhoretz, 8pilenzo, 8kranz
61. #REECE, 0rose, 0roes, 0ries, 0rice, 0rees, 0reece, 0ras, 0race, 1ruiz, 1rouse, 1ross, 1roos, 1risse, 1reyes, 1reis, 2wright, 2ryu, 2ryle, 2ruy, 2ruwe, 2ruppe, 2rupp, 2rule, 2ruhe, 2ruh, 2ruge, 2ruff, 2rudd, 2roy, 2rowe

62. #REINER, 0reiner, 0rayner, 0rainer, 1roesner, 1reisner, 1raisner, 2rudner, 2rovner, 2rosner, 2rooney, 2rittner, 2renner, 2redner, 2rattner, 3traynor, 3rightnour, 3riemer, 3rheiner, 3reinert, 3reiners, 3reimer, 3regener, 3lehner, 3kroener, 3kreiner, 3groener, 3greiner, 3bruenner, 4schreiner, 4rychener
63. #REYNA, 0reyna, 1rooney, 1reinig, 1reinecke, 2wrona, 2reiner, 2rayner, 2rana, 2rainer, 3triano, 3rynum, 3rone, 3romani, 3roesner, 3rodnick, 3rodini, 3resnick, 3reisner, 3raney, 3ranai, 3raisner, 3laine, 4runyon, 4runyan, 4rudner, 4rubinoff, 4rovner, 4rosner, 4rokne, 4rittner
64. #RICE, 0rose, 0roes, 0ries, 0rice, 0rees, 0reece, 0ras, 0race, 1ruiz, 1rouse, 1ross, 1roos, 1risse, 1reyes, 1reis, 2wright, 2ryu, 2ryle, 2ruy, 2ruwe, 2ruppe, 2rupp, 2rule, 2ruhe, 2ruh, 2ruge, 2ruff, 2rudd, 2roy, 2rowe
65. #ROGERS, 0rogers, 0rodgers, 2rogerson, 5regazzi, 5ragucci, 5gregerson, 8robers, 8rivers, 8rekers, 9wiegers, 9seegers, 9roberts, 9riegert, 9reiners, 9regar, 9ragozzino, 9rager, 9pagejones, 9kragenings
66. #ROSENKRANZ, 0rosenkranz, 3rosenkrance, 3rosencrans, 5rosendrance
67. #SCHAEFER, 0schiefer, 0schaeffer, 0schaefer, 2shaffer, 2shafer, 2schaffer, 2schafer, 3schaeper, 5shepherd, 5schuppar, 5schaper, 5schaffert, 6schoenfelder, 6scheepers, 7sheviak, 7sheffield, 7shalvoy, 7schofield, 7scheviak, 7scherf, 7scheffermann, 7scharf, 8swafford, 8stauffer, 8sheppard, 8shepard, 8schuyler, 8schueller, 8schueler, 8schmalhofer
68. #SCHILLER, 0schuller, 0schiller, 0scheller, 0scheler, 1shudler, 1shedler, 1schuyler, 1schuppler, 1schueller, 1schueler, 1scheffler, 2shelah, 2schuessler, 2schindler, 2scheifler, 3shipley, 3sherer, 3shapley, 3scherer, 4shelef, 4shelankow, 4shalla, 4shalit, 4schehrer, 4schalin, 4schalij, 5shirai, 5shiflett, 5shareef, 5sharah

69. #SCHMIDT, 0simmet, 0schmitt, 0schmit, 0schmidt, 0schmid, 0sammet, 0samet, 2smit, 2simone, 2simon, 2simoes, 2simi, 2simek, 2schmick, 2schmall, 2schmal, 3sussman, 3susman, 3sommer, 3smoot, 3smead, 3sigmann, 3sigman, 3siemon, 3sennett, 3seaman, 3schmitz, 3schmidke, 3schmetz, 3salmon
70. #SCHNEIDER, 0schneider, 2sneider, 3schmieder, 4snyder, 4snider, 5shneiderman, 7schmiedel, 8snediker, 8sneddon, 8shwayder, 8schroeder, 8schreoder, 8schmolders, 8schleider, 9shneier, 9schmedel
71. #SCHROEDER, 0schroeder, 0schreoder, 2schrader, 3schroeter, 3schleider, 4strader, 4seraydarian, 5streeter, 5schlender, 6schlueter, 7stritter, 8shwayder, 8schreiner, 8schreiber, 8schneider, 8schmieder, 8schluter, 8schlotter, 8schleter, 8schlatter, 8schlater, 8sarrafzadeh, 9sirota, 9shrimpton, 9sheridan, 9scoredos, 9schlageter, 9saladin
72. #SMITH, 0smyth, 0smith, 3semich, 3samarth, 4schmolze, 5snethen, 5smatloch, 6smothers, 6smithwick, 6smillie, 6smellie, 6sciomachen, 6schmitroth, 8sreenath, 8spath, 8smithheimer, 8smets, 8sheth, 8schmidtkehle, 9spruth, 9spaeth, 9snepscheut, 9smit, 9sleeth, 9seth, 9schumacher, 9schmitz, 9schmidke, 9schmetz, 9nemeth
73. #STEFFENS, 0stephens, 0steffens, 2stephenson, 3stevens, 5stevenson, 5staples, 9stephan, 9steins, 9steffen, 9stallings
74. #STINE, 0stumm, 0stone, 0stine, 1sutton, 1stein, 1steen, 1sitton, 1sittam, 1sattin, 2stowe, 2stopp, 2stoll, 2stock, 2stitt, 2stipp, 2stepp, 2stede, 2steck, 2stay, 2stal, 2staes, 2stack, 2sefton, 2santon, 2salton, 3stumpe, 3stump, 3stuewe, 3stueke, 3stout

75. #STRASSMANN, 0strassmann, 0strassman, 2strotmann, 2striemann, 2stratmann, 4shribman, 4schribman, 5stromer, 5streimer, 5schreibman, 6subramian, 7sulaiman, 7strojny, 7strohmeier, 7solomon, 7sloman, 7shoreman, 7schlotmann, 7salomon, 7salamone, 7salamon, 8stremikis, 8strathmeyer, 8slinkman, 8sielemann, 9strasheim, 9srimani, 9septrini, 9salemi, 9salameh
76. #TANNENBAUM, 0tenenbaum, 0tannenbaum, 0tanenbaum, 8dannenber
77. #TAYLOR, 0taylor, 1teschler, 2tyler, 2toller, 2tiller, 2tessler, 2tesler, 2teller, 3towsley, 4tselios, 4tolley, 4tilley, 4tazelaar, 4tayli, 4talley, 4stoeppler, 4stiegler, 5turlo, 5tinsley, 5thaler, 5teorey, 5teichroew, 5tarlin, 5talreja, 5stimler, 5stiller, 5stadler, 5dooley, 5doler, 5derfler
78. #THIELE, 0thiele, 0thiel, 0theile, 2thoene, 2thieme, 2theine, 2thau, 3toher, 3tchou, 4thome, 4thobe, 4thies, 4thayer, 4tham, 5twiss, 5tweed, 5touchiette, 5thurow, 5thurn, 5thorpe, 5thoro, 5thorne, 5thomas, 5thomann, 5thoma, 5thiesz, 5thibud, 5thelen, 5thedy, 5tharp
79. #VANRIJSBERGEN, 0vanrijsbergen
80. #VOIGT, 0voigt, 2vogt, 4vest, 4vertin, 5vechten, 5valente, 5port, 5poelt, 5forte, 5faust, 5faisst, 6vorholt, 6vinzant, 6vinton, 6vincent, 6versteeg, 6verruit, 6verity, 6vautour, 6vardi, 6vantyle, 6vantine, 6valtin, 6valiant, 6picht, 6levert, 6furst, 6fujita, 6foata, 6fichte
81. #WAGNER, 0wegner, 0wagner, 1wiegnier, 1weigner, 2wossner, 2wisner, 2winer, 2winer, 2wilner, 2wiener, 2wesner, 2wenner, 2weiner, 2wayner, 3werner, 3warner, 3wandner, 3aigner, 4wynia, 4wozniak, 4wilnai, 4wignall, 4weitzner, 5wuennnerke, 5woolnough, 5widmar, 5werthner, 5weinert, 5weimer, 5wegener

82. #WALLACE, 0wyllys, 0willis, 0wallis, 0wallace, 2wolik, 2willett, 2willet, 2wiley, 2wallick, 2wallack, 3wigley, 3wielage, 3wesley, 3weller, 3waller, 3wallach, 3hollis, 3hillis, 4woodley, 4wittler, 4winslow, 4winslett, 4winkley, 4wettler, 4weiler, 4warlick, 4wadler, 5wylie, 5wyland, 5wszola
83. #WANG, 0wong, 0wing, 0wang, 2winget, 2wingate, 2welge, 3yong, 3yang, 3winger, 3whang, 3wanger, 3kwong, 3hwang, 3hung, 3hong, 3heng, 3hang, 3ewing, 3ainge, 4young, 4wiple, 4whitng, 4welling, 4weigel, 4waring, 4wagle, 4swango, 4huang, 5wuepping, 5wooding
84. #WEBER, 0weber, 0webber, 1wember, 3wallbaum, 3walbaum, 3huber, 3haber, 4wolper, 4willborn, 4wilburn, 4wedberg, 4ichbiah, 5woodburn, 5wogsberg, 5wiebecke, 5westberg, 5weisberg, 5weinberg, 5weibel, 5washburn, 5wabuka, 5elber, 6wirbel, 6werbos, 6weisbeck, 6warbis, 6hubbard, 6hopper, 6holbein, 6hibbert
85. #WEISS, 0weiss, 2wyss, 2wohr, 2weyr, 2weise, 2wehr, 2warr, 2ware, 3wyrsh, 3weyer, 3heess, 4wyatt, 4wouk, 4wos, 4wood, 4woo, 4wise, 4wiig, 4wiese, 4wiele, 4wied, 4wey, 4werbos, 4wein, 4weil, 4weide, 4wei, 4warbis, 4waite, 4wain
86. #WELCH, 0welsh, 0welsch, 0welch, 0walz, 0walsh, 2winch, 2wasch, 2walczak, 2wache, 3wyrsh, 3wiecha, 3welcher, 3walzer, 3holze, 3holz, 4wuthe, 4winship, 4willshire, 4wijshoff, 4waychoff, 4waldoch, 4schwelz, 5worth, 5woellhaf, 5witcher, 5wirth, 5winkhaus, 5westhoff, 5werth, 5weinshall
87. #WHITE, 0whyte, 0whitt, 0white, 2whale, 3wheal, 3wahle, 4wiehle, 4whitng, 4wathen, 4eshed, 5woellhaf, 5wither, 5winship, 5wijshoff, 5wiethoff, 5wiecha, 5whitty, 5whitney, 5whitman, 5whitley, 5whang, 5weuthen, 5westhoff, 5weiher, 5waychoff, 5wasscher, 5walczak, 5uhle, 5aho, 6yuhas

APPENDIX A. EXPERIMENT DATA OF SAPS

131

88. #WIRTH, 0worth, 0wirth, 0werth, 2wyrsh, 2warthen, 4wuthe, 4wilmouth, 4wiethoff, 4weuthen, 4warshaw, 4warshall, 4hoerth, 4haworth, 5worhach, 5wiseth, 5wilmeth, 5wiecha, 5weyrich, 5wallrath, 5samworth, 5papworth, 5orth, 5neuwirth, 5hoath, 5hirsh, 5hirsch, 5herz, 5hersh, 5hersch, 5heath
89. #YANG, 0yong, 0yang, 1young, 3yago, 3yagi, 3wong, 3wing, 3wang, 3hung, 3hong, 3heng, 3hang, 3ainge, 4huang, 5yentang, 5yager, 5yaeger, 5winget, 5wingate, 5welge, 5ong, 5eng, 5ang, 6zheng, 6zhang, 6winger, 6whang, 6wanger, 6udding, 6sheng
90. #ZWEIG, 0zweig, 5zweben, 5zwaan, 5zhou, 5zhao, 5sweet, 5sweep, 5swaine, 5swain, 6zwicker, 6zachwieja, 6schweid, 7zhu, 7zerweck, 7zakhor, 7zakheim, 7swope, 7swid, 7swick, 7swed, 7swan, 7sawyer, 8zwalina, 8zurwehn, 8swoben, 8swee, 8swasy, 8swasey, 8sward, 8swamy

Appendix B

Preprocessing Step in Phonix

B.1 Part of the C Source Code

The following C code contains the preprocessing in Phonix algorithm as it was defined by [48, 49].

```
/******  
NAME : PhonixReplace1  
INPUT : int where — replace OldStr only if it occurs at this position  
char *Name — string to work  
char *OldStr — old letter group to delete  
char *NewStr — new letter group to insert  
int CondPre — condition referring to letter before OldStr  
int CondPost — condition referring to letter after OldStr  
OUTPUT : char *Name — Name with replaced letter group
```

FUNCTION: This procedure replaces the letter group OldStr with the letter group NewStr in the string Name, regarding the position of OldStr where (START, MIDDLE, END, ALL) and the conditions CondPre and CondPost (NON, VOC, CON).RT).

/*****

void PhonixReplace1 (int Where, char *Name, char *OldStr, char *NewStr, int CondPre, int CondPost)

*****/

NAME : PhonixReplace2

INPUT : int where — replace OldStr only if it occurs at this position

char *Name — string to work

char *OldStr — old letter group to delete

char *NewStr — new letter group to insert

OUTPUT : char *Name — Name with replaced letter group

FUNCTION: This procedure replaces the letter group OldStr with the letter group NewStr in the string Name, regarding the position of OldStr where (START, MIDDLE, END, ALL).

/*****

void PhonixReplace2 (int Where, char *Name, char *OldStr, char *NewStr)

*****/

NAME : PhonixPreprocess

INPUT : char *Name — string to be preprocessed

OUTPUT : char *Key — preprocessing result string

FUNCTION: PhonixPreprocess transforms the original string according to Gadd's rules

```

/*****
void PhonixPreprocess (char *Name, char *Key){
char NewName[50];
int i,j,k;
char ch;
strcpy(NewName, Name);
/* replace letter groups according to Gadd's definition */
PhonixReplace2(ALL , NewName, "DG" , "G" );
PhonixReplace2(ALL , NewName, "CO" , "KO" );
PhonixReplace2(ALL , NewName, "CA" , "KA" );
PhonixReplace2(ALL , NewName, "CU" , "KU" );
PhonixReplace2(ALL , NewName, "CY" , "SI" );
PhonixReplace2(ALL , NewName, "CI" , "SI" );
PhonixReplace2(ALL , NewName, "CE" , "SE" );
PhonixReplace1(START , NewName, "CL" , "KL" , NON, VOC);
PhonixReplace2(ALL , NewName, "CK" , "K" );
PhonixReplace2(END , NewName, "GC" , "K" );
PhonixReplace2(END , NewName, "JC" , "K" );
PhonixReplace1(START , NewName, "CHR" , "KR" , NON, VOC);
PhonixReplace1(START , NewName, "CR" , "KR" , NON, VOC);
PhonixReplace2(START , NewName, "WR" , "R" );
PhonixReplace2(ALL , NewName, "NC" , "NK" );

```

APPENDIX B. PREPROCESSING STEP IN PHONIX

135

```

PhonixReplace2(ALL , NewName, "CT" , "KT" );
PhonixReplace2(ALL , NewName, "PH" , "F" );
PhonixReplace2(ALL , NewName, "AA" , "AR" );
PhonixReplace2(ALL , NewName, "SCH" , "SH" );
PhonixReplace2(ALL , NewName, "BTL" , "TL" );
PhonixReplace2(ALL , NewName, "GHT" , "T" );
PhonixReplace2(ALL , NewName, "AUGH" , "ARF" );
PhonixReplace1(MIDDLE, NewName, "LJ" , "LD" , VOC, VOC);
PhonixReplace2(ALL , NewName, "LOUGH" , "LOW" );
PhonixReplace2(START , NewName, "Q" , "KW" );
PhonixReplace2(START , NewName, "KN" , "N" );
PhonixReplace2(END , NewName, "GN" , "N" );
PhonixReplace2(ALL , NewName, "GHN" , "N" );
PhonixReplace2(END , NewName, "GNE" , "N" );
PhonixReplace2(ALL , NewName, "GHNE" , "NE" );
PhonixReplace2(END , NewName, "GNES" , "NS" );
PhonixReplace2(START , NewName, "GN" , "N" );
PhonixReplace1(MIDDLE, NewName, "GN" , "N" , NON, CON);
PhonixReplace1(END , NewName, "GN" , "N" , NON, NON);
PhonixReplace2(START , NewName, "PS" , "S" );
PhonixReplace2(START , NewName, "PT" , "T" );
PhonixReplace2(START , NewName, "CZ" , "C" );
PhonixReplace1(MIDDLE, NewName, "WZ" , "Z" , VOC, NON);
PhonixReplace2(MIDDLE, NewName, "CZ" , "CH" );

```

APPENDIX B. PREPROCESSING STEP IN PHONIX

136

```

PhonixReplace2(ALL , NewName, "LZ" , "LSH" );
PhonixReplace2(ALL , NewName, "RZ" , "RSH" );
PhonixReplace1(MIDDLE, NewName, "Z" , "S" , NON, VOC);
PhonixReplace2(ALL , NewName, "ZZ" , "TS" );
PhonixReplace1(MIDDLE, NewName, "Z" , "TS" , CON, NON);
PhonixReplace2(ALL , NewName, "HROUG" , "REW" );
PhonixReplace2(ALL , NewName, "OUGH" , "OF" );
PhonixReplace1(MIDDLE, NewName, "Q" , "KW" , VOC, VOC);
PhonixReplace1(MIDDLE, NewName, "J" , "Y" , VOC, VOC);
PhonixReplace1(START , NewName, "YJ" , "Y" , NON, VOC);
PhonixReplace2(START , NewName, "GH" , "G" );
PhonixReplace1(END , NewName, "E" , "GH" , VOC, NON);
PhonixReplace2(START , NewName, "CY" , "S" );
PhonixReplace2(ALL , NewName, "NX" , "NKS" );
PhonixReplace2(START , NewName, "PF" , "F" );
PhonixReplace2(END , NewName, "DT" , "T" );
PhonixReplace2(END , NewName, "TL" , "TIL" );
PhonixReplace2(END , NewName, "DL" , "DIL" );
PhonixReplace2(ALL , NewName, "YTH" , "ITH" );
PhonixReplace1(START , NewName, "TJ" , "CH" , NON, VOC);
PhonixReplace1(START , NewName, "TSJ" , "CH" , NON, VOC);
PhonixReplace1(START , NewName, "TS" , "T" , NON, VOC);
PhonixReplace1(ALL , NewName, "TCH" , "CH" );
PhonixReplace1(MIDDLE, NewName, "WSK" , "VSKIE", VOC, NON);

```

APPENDIX B. PREPROCESSING STEP IN PHONIX

137

```

PhonixReplace1(END , NewName, "WSK" , "VSKIE", VOC, NON);
PhonixReplace1(START , NewName, "MN" , "N" , NON, VOC);
PhonixReplace1(START , NewName, "PN" , "N" , NON, VOC);
PhonixReplace1(MIDDLE, NewName, "STL" , "SL" , VOC, NON);
PhonixReplace1(END , NewName, "STL" , "SL" , VOC, NON);
PhonixReplace2(END , NewName, "TNT" , "ENT" );
PhonixReplace2(END , NewName, "EAUX" , "OH" );
PhonixReplace2(ALL , NewName, "EXCI" , "ECS" );
PhonixReplace2(ALL , NewName, "X" , "ECS" );
PhonixReplace2(END , NewName, "NED" , "ND" );
PhonixReplace2(ALL , NewName, "JR" , "DR" );
PhonixReplace2(END , NewName, "EE" , "EA" );
PhonixReplace2(ALL , NewName, "ZS" , "S" );
PhonixReplace1(MIDDLE, NewName, "R" , "AH" , VOC, CON);
PhonixReplace1(END , NewName, "R" , "AH" , VOC, NON);
PhonixReplace1(MIDDLE, NewName, "HR" , "AH" , VOC, CON);
PhonixReplace1(END , NewName, "HR" , "AH" , VOC, NON);
PhonixReplace1(END , NewName, "HR" , "AH" , VOC, NON);
PhonixReplace2(END , NewName, "RE" , "AR" );
PhonixReplace1(END , NewName, "R" , "AH" , VOC, NON);
PhonixReplace2(ALL , NewName, "LLE" , "LE" );
PhonixReplace1(END , NewName, "LE" , "ILE" , CON, NON);
PhonixReplace1(END , NewName, "LES" , "ILES" , CON, NON);
PhonixReplace2(END , NewName, "E" , "" );

```

APPENDIX B. PREPROCESSING STEP IN PHONIX

138

```
PhonixReplace2(END , NewName, "ES" , "S" );
PhonixReplace1(END , NewName, "SS" , "AS" , VOC, NON);
PhonixReplace1(END , NewName, "MB" , "M" , VOC, NON);
PhonixReplace2(ALL , NewName, "MPTS" , "MPS" );
PhonixReplace2(ALL , NewName, "MPS" , "MS" );
PhonixReplace2(ALL , NewName, "MPT" , "MT" );
strcpy(Key,NewName);
}
```

Appendix C

Computer Forensics Resources

C.1 Information and Organizations

1. The National Consortium for Justice Information and Statistics,
<http://www.search.org/>
2. National Center for Forensic Science, <http://ncfs.ucf.edu/>
3. National Institute of Justice, <http://www.nlectc.org/>
4. NIST Computer Forensics Tool Testing Program,
<http://www.cftt.nist.gov/>.
5. National White Collar Crime Center (NW3C), <http://www.nw3c.org/>
6. Forensic Science Service Online, UK, <http://www.forensic.gov.uk/>
7. Computer Crime and Intellectual Property Section (CCIPS) of
the Criminal Division of the U.S. Department of Justice,
<http://www.usdoj.gov/criminal/cybercrime/index.html>

APPENDIX C. COMPUTER FORENSICS RESOURCES

140

8. The National Cybercrime Training Partnership (NCTP),
<http://www.nctp.org/>
9. Carnegie Mellon's Computer Emergency Response Team
Coordination Center (CERT/CC), <http://www.cert.org/>
10. Law Enforcement Coordinating Committee (LECC)
on Information Technology - Western New York,
<http://www.wdnytech.org>
11. Computer Forensics Laboratory - Department of Defense,
<http://www.dcfll.gov>
12. Federal Computer Incident Response Center (FedCIRC),
<http://www.fedcirc.gov/index.html>
13. Scientific Working Group in Digital Evidence (SWGDE),
<http://www.for-swg.org>
14. Regional Computer Forensics Laboratory (RCFL), San Diego, CA,
<http://www.rcfl.org/>
15. National Infrastructure Protection Center (NIPC), <http://www.nipc.gov/>
16. Pittsburgh High Tech Computer Crime Task Force,
<http://www.cyber-response.org/>
17. Internet Fraud Complaint Center (IFCC), <http://www1.ifccfbi.gov>
18. International Association of Computer Investigative Specialists (IACIS),
<http://www.iacis.com/>

19. Regional Computer Forensics Laboratory, Western New York,
<http://www.rcflwny.org/>
20. Forensics Web, <http://www.forensicsweb.com>
21. FBI National Computer Crime Squad,
<http://www.emergency.com/fbi-nccs.htm>.
22. Computer Forensics, Cybercrime and Steganography Resources,
<http://www.forensix.org/links>.
23. SANS Reading Room, <http://www.sans.org/>.

C.2 Publications

1. Digital Investigation,
<http://www.compseconline.com/digitalinvestigation/>.
2. Digital Forensic Research Workshop, <http://www.dfrws.org/>.
3. International Journal of Digital Evidence (IJDE), <http://www.ijde.org/>.

C.3 Software

1. AccessData, <http://www.accessdata.com/>.
2. Computer Cop, <http://www.computercop.com/>.
3. dtSearch, <http://www.dtsearch.com/dtsoftware.html>.

4. EnCase, <http://www.guidancesoftware.com/>.
5. ILook Investigator, <http://www.ilook-forensics.org/>.
6. Legal Imager and reaSsembly Application,
<http://www.blackcat.demon.co.uk/lisa/>.
7. Mares and Company, <http://www.dmares.com/>.
8. New Technologies, Inc., <http://www.forensics-intl.com/>.
9. SMART, <http://www.asrdata.com/>.
10. SnapBack, <http://www.snapback.com/>.
11. Symantec Ghost,
<http://sea.symantec.com/content/product.cfm?productid=9>.
12. Technology Pathways, <http://www.techpathways.com/>.
13. Paraben Corporation, <http://www.paraben-forensics.com/>.
14. Vogon International, <http://www.vogon-international.us/tools>.
15. Avantstar Quick View Plus, <http://www.avantstar.com/>.
16. Digital Intelligence, <http://www.digitalintelligence.com/>.
17. DiskJockey File Viewer, <http://www.clear-simple.com/>.
18. E-mail Detective,
http://www.evestigate.com/computer_Forensic_Training.htm.
19. Hushkeeper, <http://www.hashkeeper.org/>.

20. Intelligence Computer Solutions,
http://www.icsforensic.com/products_cat_fr.cfm.
21. The Coroner's Toolkit, <http://www.porcupine.org/forensics/tct.html>.
22. The Sleuth Kit, <http://www.sleuthkit.org/sleuthkit/index.php>.
23. Tech Assist, Inc. (Byteback)
<http://www.toolsthatwork.com/computer-forensic.htm>.
24. ThumbsPlus, <http://www.cerious.com/>.
25. WinHex, <http://www.winhex.com/>.
26. X-Ways, <http://www.x-rays.net/forensics/index-m.html>.

C.4 Certifications

1. Advanced Information Security (AIS),
<http://www.securityuniversity.net/certification.php>.
2. *Certified Computer Examiner (CCE),
<http://www.certified-computer-examiner.com>.
3. Certified Cyber-Crime Expert (C³E), <http://www.ctrcglobal.com>
4. *Certified Information Forensics Investigator (CIFI),
<http://www.iisfa.org/certification/certification.asp>.
5. *Certified Computer Crime Investigator (CCCI),
<http://www.htcn.org/changes.htm>.

APPENDIX C. COMPUTER FORENSICS RESOURCES

144

6. Certified Computer Forensic Technician (CCFT), <http://www.htcn.org>.
7. *Computer Forensic Computer Examiner (CFCE),
<http://www.iacis.com/html/training.htm>.
8. Certified Information System Auditor (CISA), <http://www.isaca.org>.
9. EnCase Certified Examiner Program, <http://www.guidancesoftware.com>.
10. GIAC Certified Forensic Analyst (GCFA),
http://www.giac.org/subject_certs.php.
11. *Personal Certified Investigator (PCI),
<http://www.asisonline.org/certification/pci/pciabout.xml>.

*: *highly recognized*.

Author's Publication

1. **Gong Ruibin**, Chan Kai Yun Tony. Syllable Alignment: A Novel Model for Phonetic String Search. *IEICE Trans. on Information & Systems*, 2006 E89-D(1):332-339.
2. **Gong Ruibin**, Chan Kai Yun Tony, and Mathias Gaertner. Case-Relevance Information Investigation: Binding Computer Intelligence into Current Computer Forensic Framework. *International Journal of Digital Evidence (IJDE)*, 4(1). Spring 2005.
3. **Gong Ruibin**, Chan Kai Yun Tony. Self-Learning for Phonetic String Search. In Proceedings of *Ninth International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES2005)*, p915-912. 14 - 16 September, 2005, Hilton on the Park, Melbourne, Australia.
4. **Gong Ruibin**, Chan Kai Yun Tony. A Phonetic String Searching Algorithm based on Syllable Alignment. In Proceedings of *the 8th IASTED International Conference on Internet & Multimedia Systems & Applications (IMSA2004)*, Aug. 2004. Kauai, Hawaii, USA.
5. **Gong Ruibin**, Chan Kai Yun Tony. A Fast Algorithm for Online Multiple Name Searching. In Proceedings of *the 2nd IASTED International Conference on Communications, Internet & Information Technology (CIIT2003)*, Nov. 2003.Scottsdale, AZ.
6. Wu Horng Jyh, Paul, Ng Hong I, and **Gong Ruibin**. Efficient Methods for Multigram Compound Discovery. In Proceedings of *the 17th Pacific Asia Conference on Language, Information and Computation* . Oct. 2003. Singapore.