Chapter 3

Advancements in Text Mining Algorithms and Software

Svetlana Y. Mironova*, Michael W. Berry*,
Scott Atchley*, Micah Beck*, Tianhao Wu†,
Lars E. Holzman‡, William M. Pottenger‡, Daniel J. Phelps‡

* Department of Computer Science, University of Tennessee, Knoxville, TN, 37996-3450; [mironova,berry,atchley,mbeck]@cs.utk.edu
† Department of Computer Science and Engineering, Lehigh University, Bethelem, PA, 18015; [tiw2,leh7,billp]@lehigh.edu
‡ Eastman Kodak Company, 1999 Lake Avenue, Rochester, NY 14650-2138; daniel.phelps@kodak.com

August 11, 2003

Abstract:
In this chapter, we present two advancements in the development of algorithms and software for the mining of textual information. For large-scale indexing needs, we present the General Text Parser (GTP) software environment with network storage capability. This object-oriented software (C++, Java) is designed to provide information retrieval (IR) and data mining specialists the ability to parse and index large text collections. GTP utilizes Latent Semantic Indexing (or LSI) for the construction of a vector space IR model. Users can choose to store the files generated by GTP on a remote network in order to overcome local storage restrictions and facilitate file sharing. For
feature extraction from textual information, we present a supervised learning algorithm 
for the discovery of finite state automata, in the form of regular expressions, in textual 
data. The automata generate languages that consist of various representations of fea-
tures useful in information extraction. This learning technique has been used in the 
extraction of textual features from police incident reports and patent data.

Keywords: finite state automata, information retrieval, latent semantic indexing, logis-
tical networking, supervised learning, text mining.

3.1 Introduction

The amount of textual-based information stored electronically, whether on our own 
computers or on the Web, is rapidly accumulating. Any desktop or laptop computer 
can accommodate huge amounts of data due to the advances in hardware storage de-
vices. Accumulating information is easy, finding relevant information on demand can 
be difficult. Constructing data structures (indices) to facilitate the retrieval of rele-
vant information becomes problematic as the size of collections continue to escalate. 
Equally important is the ability to extract specific patterns or features to meet partic-
ular information needs. In this chapter we discuss novel developments in the design 
of software for large-scale index creation and algorithms for feature extraction from 
textual media. A more comprehensive survey of the field of text mining is available 
in [Ber03].

3.1.1 Software Advancement for Information Retrieval

Software companies develop products that may require megabytes of hard drive space. 
Without upgrading computers every few years, one cannot download favorite music, 
movies or play the most recent (popular) computer games. Researchers and scientists 
involved in data mining and information retrieval are facing the same reality – an enor-
mous amount of storage may be needed to run simulations and store their outputs. In 
creating the General Text Parser (GTP) with network storage capability, we are trying 
to address the needs of experts in information retrieval and modeling who deal with 
large text corpora on a daily basis but are subject to limited storage capabilities. This 
software allows a user to parse a large collection of documents and create a vector 
space information retrieval model for subsequent concept-based query processing. 

GTP utilizes latent semantic indexing (LSI) for its information retrieval (IR) mod-
eling [BB90, BDO95, BDJ99]. The user has the option of storing the model outputs 
on one of the available Internet Backplane Protocol (IBP) servers or depots [BBF+02, 
PBB+01] so that disk or memory space is shared over the network.

IBP is the foundation of the Logistical Networking Testbed developed at the Logis-
tical Computing and Internetworking (LoCI) Lab at the University of Tennessee. This 
infrastructure provides a scalably sharable storage service as a network resource for 
distributed applications [BMP02].
3.1.2 Advancement in Algorithms for Feature Extraction

We also developed an information extraction (IE) algorithm, which can make use of GTP storage. The algorithm uses reduced regular expressions as IE patterns. Regular expressions can be used as patterns to extract features from semi-structured and narrative text [Sod99]. A study of hundreds of police incident reports and patents indicates that regular expressions can be readily employed to express patterns of features. For example, in a police incident report a suspect’s height might be recorded as “{CD} feet {CD} inches tall”, where {CD} is the part of speech tag for a numeric value. Alternatively, if a sentence in a patent matches the regular expression “invention .* (to)? .* improve”, then it likely contains information about the particular solution that addresses an insufficiency in prior art. An algorithm for the automatic discovery of regular expressions of this nature has been developed.

At Lehigh University, co-authors Wu, Holzman, and Pottenger are conducting information extraction research in collaboration with the Information Mining Group at Eastman Kodak Company and Lockheed Martin M&DS in conjunction with the Pennsylvania State Police. In work with Lockheed Martin for the Pennsylvania State Police, they are also developing a system that extracts features related to criminal modus operandi and physical descriptions for suspects as recorded in narrative incident reports. Their results in [WP03b] demonstrate good performance on ten features important in homeland defense.

In collaboration with the Eastman Kodak Company (co-author Phelps), the Lehigh group is developing technologies capable of automatically extracting sentences in patents that identify the problem that a given patent addresses. Sentences of this nature are referred to as problem solved identifiers (or PSIs). This is an important domain given the commercial value of information automatically extracted from patents [HCSH99].

The remainder of the chapter is organized as follows: in Section 3.2, we provide an overview of the General Text Parser (GTP) and the GTPQUERY process. In Section 3.3, we describe the Network Storage Stack and its components which are used to implement the network storage utility within GTP. Section 3.4 details the GTP network storage implementation and usage. Section 3.5 discusses future work in the use of network storage for indexing. In Section 3.6, we summarize related work in regular expression matching in textual data, followed by a list of definitions in Section 3.7. In Section 3.8, a regular expression discovery algorithm is presented followed by a discussion of preliminary experimental results obtained for patent data and police incident reports [WP03b] in Section 3.9. Concluding remarks for the chapter are provided in Section 3.10, followed by acknowledgements in Section 3.11.

3.2 GTP

General Text Parser (GTP) is a software package developed at the University of Tennessee to facilitate text/document processing and parsing and to implement an underlying IR model based on sparse matrix data structures. GTP has the ability to parse any document: raw text, HTML document or any other tag separated document collection via tag filters. During the parsing process GTP creates a vector-space model in which
the documents and queries are represented as vectors of the terms parsed. A term-by-document matrix is used to define the relationships between the documents in the collection and the parsed terms or keywords. The elements of the matrix are typically weighted/unweighted frequencies of terms (rows) with respect to their corresponding documents (columns) [GWB03].

The underlying vector-space model exploited by the GTP is Latent Semantic Indexing (LSI). LSI is an efficient IR technique that uses statistically derived conceptual indices rather than individual words to encode documents. Specifically, LSI uses the singular value decomposition (SVD) or semi-discrete decomposition (SDD) of the large sparse term-by-document matrix mentioned above to build a conceptual vector space [BB90, BDO95]. A lower-rank approximation to the original term-by-document matrix is used to derive vector encodings for both terms and documents in the same $k$-dimensional subspace. The clustering of term or document vectors in this subspace suggests an underlying (latent) semantic structure in the usage of terms within the documents.

### 3.2.1 Evolution of GTP

GTP is public domain software available at http://www.cs.utk.edu/~lsi. The original version was developed in C++ for both Solaris and Linux platforms. There also exists a parallel version of the SVD components used in GTP which is written in C++/MPI (Message Passing Interface) [SOHL$^+$95]. The C++ version was recently ported to Java to utilize more object-oriented features. The Java version has certain limitations compared to its C++ counterpart: it is slower and it does not accept custom filters, but it does provide an internal HTML filter.

### 3.2.2 Capabilities

GTP is characterized by the numerous options provided for the user. The options allow both novice and expert users to tune the software to their needs. Recently a graphical user interface has been developed to help keep track of all the options. Some options let the user change thresholds for document and global frequencies, specify custom filters and local/global weighting schemes, and indicate new document delimiters. For more detailed overview of the options see [GWB03].

Depending on the collection, the size of the above mentioned files (when deploying the SVD) can be very large varying from kilobytes to gigabytes. For 8-byte double precision storage, a $k$-dimensional LSI model requires $8k(t + d)$ bytes, where $t$ and $d$ are the number of terms and documents in the collection, respectively. Hence, a 300-dimensional vector space model for a text collection comprising 100,000 terms and 10,000 documents would require well over 250MB. The user must also consider that in the course of their research he/she might need to repeat the process of parsing several times to achieve the desired IR model.

Using the Internet Backplane Protocol (IBP) as described in Section 3.4, one can successfully eliminate the storage bottleneck.
3.2.3 The GTPQUERY Process

GTP is not only capable of creating an index; it provides users with a GTPQUERY module for querying, i.e., determining the similarities between a query and all documents in the collection. This query processing module requires several of the output files generated by GTP, namely key, output, and LAST_RUN. A cosine similarity measure between the query vector and document vectors is used to determine the relevance of any/all documents to the query. Query vectors are constructed as pseudo-document vectors thus allowing their projection into the original term-document vector space.

The result of the query process consists of files (one per query) with document ID and corresponding cosine similarity pairs ranked from the most relevant to the least relevant. The entire GTP and GTPQUERY process is summarized in Figure 3.1.

![Figure 3.1: Flowchart of GTP and GTPQUERY processing; output files are listed in parentheses.](image)

3.3 Network Storage Stack

The Network Storage Stack has been developed by the Logistical Computing and Internetworking Lab (LoCI) at the University of Tennessee [CwLL03]. As discussed
in [BBF+02], the Network Storage Stack is modeled after the Internet Protocol (IP) Stack, and is designed to add storage resources to the Internet in a sharable, scalable manner. Figure 3.2 shows the organization of the Network Storage Stack.

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistical File System</td>
</tr>
<tr>
<td>Logistical Tools</td>
</tr>
<tr>
<td>L-Bone</td>
</tr>
<tr>
<td>exNode</td>
</tr>
<tr>
<td>IBP</td>
</tr>
<tr>
<td>Local Access</td>
</tr>
<tr>
<td>Physical</td>
</tr>
</tbody>
</table>

Figure 3.2: Network Storage Stack.

### 3.3.1 IBP

The Internet Backplane Protocol (IBP) is an essential part of the Network Storage Stack. IBP’s purpose is to allow users to share storage resources across networks. Its design echoes the major advantages of Internet Protocol (IP): abstraction of the datagram delivery process, scalability, simple fault detection (faulty datagrams are dropped), and ease of access. These factors allow any participant in an IBP network to use any local storage resource available regardless of who owns it [BBF+02]. Using IP networking to access IBP storage creates a global storage service.

There are some limitations that arise from two underlying network problems. The first problem concerns a vulnerability of IP networks to Denial of Use (DoU). The free sharing of communication within a routed IP network leaves every local network open to being overwhelmed by traffic from the wide area network. A second concern lies in that the storage service is based on processor-attached storage, which implies strong semantics: near-perfect reliability and availability. IBP is almost impossible to implement on the scale of the wide area networks [BBF+02]. These issues are resolved as follows:

- IBP storage is time limited. When the time expires the resources can be reused. An IBP allocation can also be refused by a storage facility (depot) if the user’s request demands more resources than available.

- IBP is a best effort storage service [BBF+02]. The semantics of IBP storage are weaker than the typical storage service. Since there are so many unpredictable and uncontrollable factors involved, network access to storage may become permanently unavailable (if a depot decides to withdraw from the pool, for example).

IBP storage is managed by depots or servers used by a client to perform storage operations such as Allocate, Load, Store, and Copy. See [Mir03] for details on these and other storage operations.
3.3.2 ExNode

The management of several IBP capabilities can be complicated. The exNode library was created to help the user in this task and to automate most of the work. The exNode data structure is somewhat similar to the Unix inode, but at the same time it is fundamentally different.

The exNode makes it possible for the user to chain IBP allocations into a logical entity that resembles a network file [BBM01]. Current IBP allocations have a limit of 2 GB; the exNode though allows the user to chain 2 billion IBP allocations, which equals 4 Exabytes \((2^{62})\) [BBM01].

The exNode consists of two major components: arbitrary metadata and mappings. The exNode library allows the user to create an exNode, attach a mapping to it, store IBP capabilities into the mapping and add metadata to the mapping. The exNode can also be serialized to XML, so that exNodes created on one platform can be recognized on other supported platforms. Each exNode can have multiple copies of the allocation, which provides better fault-tolerance. If a depot becomes unavailable for some reason, the user can still retrieve data from the copies stored on other depots.

3.3.3 L-Bone

The Logistical Backbone (L-Bone) is a resource discovery service that maintains a list of public depots and metadata about those depots [BBF+.02, BMP02]. The L-Bone also uses the Network Weather Service (NWS) [WSH99] to monitor throughput between depots. As of March 2003, the L-Bone provides service of over 150 depots on five continents. Figure 3.3 shows the locations of the available IBP depots.

![Figure 3.3: Over 150 worldwide depots of the L-Bone [CwLL03].](image)

3.3.4 LoRS

The next and final layer of the Network Storage Stack (see Figure 3.2) which we briefly mention is the Logistical Runtime System, or LoRS. The LoRS layer consists of a C
API and a command line interface tool set that automate the finding of IBP depots via the L-Bone, creating and using IBP capabilities and creating and managing exNodes [CwLL03]. The LoRS library also provides flexible tools to deal with the lower levels of the Network Storage Stack. Sample network file-based functions [BBF02] include: Upload, Download, Augment, Trim, Refresh, and List. LoRS supports checksums to ensure end-to-end correctness, multiple encryption algorithms since IBP depots are public, untrusted servers, and compression to reduce the amount of data transferred and stored.

3.4 GTP with Network Storage

In the process of developing GTP, we realized that parsing large collections generates numerous large files that take up a lot of valuable disk space. The Logistical Networking Testbed developed at LoCI facilitated the temporary storage of these files on a remote network (Internet) along with immediate retrieval when needed.

In the course of creating an index for a document collection, new documents may get added to the collection or some documents may be deleted. In any case, before the final collection is created, several revisions are usually done and the user may need to parse the collection multiple times. In some cases the collection is dynamic, as is the case with webpages (HTML), so that parsing is done on a regular basis in order to monitor updates. If the user keeps all the files generated by GTP and GTPQUERY after each parsing, the subsequent output files will take up an excessive amount of local disk storage. Fortunately, the concept of network storage can alleviate this burden: the user can clean up his/her hard drive and store the information produced by the parser on a remote network. Since the storage provided by the Internet Backplane Protocol (IBP) is temporary, if the user is not satisfied with the results, he will not choose to extend the time on the files stored on the IBP and the storage will be automatically reused. If, on the other hand, the user wants to store the results of the parser permanently, he can either make sure that the time limits do not expire or he can download the files back to his personal machine and then write them to other media, e.g., a CD-ROM.

3.4.1 Overview

The execution of GTP (see Figure 3.1) creates two large files: keys (the database of the terms parsed) and output (a binary file, containing vector encodings generated by the SVD). These files are essential to the GTP and GTPQUERY. If the user chooses to use network storage, after the files keys and output are generated, they are automatically uploaded to IBP depot(s). When the user wants to query into the collection that has been created, these files are downloaded back to the user’s space. The LoRS tools are used to facilitate upload and download processing.

3.4.2 GTP and Upload

The upload process requires as little or as much information from the user as he/she is willing to provide. This information helps to optimize the performance of the tools.
Here is a list of the fields the user can specify:

**Location** allows the user to enter keyword and value pairs to determine where they want storage and minimum environmental criteria. The user may specify as many or as few keyword/value pairs as the user wants. They may even leave location pointer equal to NULL if location and environment are unimportant. One can specify hostname, zip, state, city, country and airport.

**Duration** is the maximum number of days that the user will need the space. The user can even specify partial day amounts. For example, if 0.5 is entered, data will be stored on the network for 12 hours. Each depot has the maximum number of days the data will be stored for. This information can be obtained from the L-Bone (see http://loci.cs.utk.edu/lbone). If a longer time period is required, the user is currently responsible for extending the time of the allocation. A set of tools that will do this automatically is in development.

**Fragments** allows the user to subdivide a file into partitions of equal size and to store those partitions on different depots. Available depot space is used more efficiently and the performance of the download can be greatly improved.

**Copies** allows the user to specify how many copies of the original file to store. Users are encouraged to store several copies of the data. As was mentioned in Section 3.3.2, there is always a possibility that the data could be lost due to numerous uncontrollable circumstances. Subdividing the file into several fragments and storing multiple copies of the file can prevent an undesired loss of data. If during the download process some fragments cannot be found, LoRS tools will automatically check for all the copies of this fragment and will deliver the first available one.

If the upload is successful, LoRS tools will return to the user a file with .xnd extension. This file contains XML encoded information needed by the user and IBP to keep track of the file, retrieve the file and perform LoRS operations described in Section 3.3.4. GTP will store XML files (one per uploaded file) in a directory and will automatically delete the files being uploaded in order to conserve local disk storage. If on the other hand, the upload has failed, the files will be saved on the user’s machine and the user will be notified of the failure.

### 3.4.3 Download and GTPQUERY

If IBP was used to store the GTP-generated index, a query into the document collection requires that the files keys and output be downloaded from the network. The download process solely depends on the XML files produced during the upload process. The .xnd files hold the key to the location of the user’s data within IBP. If those files do not exist, download will fail and the recovery of the data will be impossible. The LoRS download tool uses multiple threads to retrieve small blocks of data and then it reassembles the blocks into the complete file at the client. LoRS uses an adaptive algorithm that retrieves more blocks from faster depots (depots with higher throughput to the client).
If some depots are much slower than others, the download tool can automatically try getting lagging blocks from the other depots that have the same data [PADB02]. The download tool is capable of starting from a specified offset and can process a prescribed bytectcount of data. All GTP output files, however, are downloaded in their entirety. After the download process is complete, the user will have the files necessary to perform any query on the collection. The progress of any upload or download is monitored by a special panel (“Network Storage Panel”) provided in the GTP graphical user interface (GUI) [Mir03].

### 3.4.4 Performance

Current benchmarks (see [Mir03]) on a FBIS (Foreign Broadcast Information Service) sub-collection from TREC-5 [HV97] (size: 63MB, 20,000 documents, 46,488 terms, output – 28MB, keys – 5.8MB) indicate that the additional time/overhead for upload is not significant compared to the total elapsed time. Table 3.1 shows the timing results for a GTP upload to France (FR), California (CA) and Tennessee (TN) with the server located in TN. The time of the upload depends on multiple factors: how far the location of the upload is from the user’s location, network bandwidth, size of the file to be uploaded and the number of copies requested.

The GTP download process, on the other hand, is almost instantaneous. All the preprocessing is done by GTP during the parsing and construction of the model, so that the GTPQUERY process simply projects the query into the original term-document space. Table 3.1 demonstrates that most of the time of the download and query processing (GTPQUERY) is taken by the download. The three batch-mode ad hoc queries used to demonstrate this benchmark were: Yugoslavia Croatia, Russia embassy FIS, and Nissan Motor [Mir03].

Table 3.1: GTP Upload and Download benchmarks for the FBIS subcollection (20,000 documents and 3 queries). FR=France, CA=California, and TN=Tennessee.

<table>
<thead>
<tr>
<th>Upload Performance (seconds)</th>
<th>FR</th>
<th>CA</th>
<th>TN</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTP</td>
<td>320</td>
<td>320</td>
<td>320</td>
</tr>
<tr>
<td>Upload/IBP</td>
<td>270</td>
<td>23</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Download Performance (seconds)</th>
<th>FR</th>
<th>CA</th>
<th>TN</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTPQUERY</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>Download/IBP</td>
<td>113</td>
<td>38</td>
<td>34</td>
</tr>
</tbody>
</table>
3.5 Future Software Development

The network storage option implemented for the Java version of GTP presented some challenges, since all IBP and LoRS tools were originally implemented in C. The merge was possible due to a special LoRS server. Each time the IBP or LoRS tools were updated, GTP had to be tuned to adjust to the changes. Work to integrate network storage into the C++ and parallel versions of GTP is in progress.

In collaboration with LoCI Lab [CwLL03], refinements of the network storage procedure itself are underway. Upgrades include adding interactive maps and utilities to allow the user to see the information about the files stored on IBP, extending storage time with a click of a button, and the possibility of streaming the data directly from a LoRS Java (or C) client to IBP depots as it gets generated. Currently, streaming can only be performed using the LoRS C library or the UNIX command line tools. This would eliminate local file generation and greatly improve the overall performance of GTP.

GTP has the ability to parse and index large text collections through networking storage. Therefore, a variety of distributed data mining algorithms and data mining systems can use GTP for data collection. Co-authors at Lehigh University developed a supervised information extraction algorithm that can use GTP to store and index text collections. We present the algorithm in the following sections.

3.6 Regular Expression Matching

Although much work has been done in the field of information extraction, relatively little has focused on the automatic discovery of regular expressions. In this section, we highlight a few efforts that are related to regular expression discovery. We also touch on related work in citation analysis of patents.

Stephen Soderland developed a supervised learning algorithm, WHISK [Sod99], which uses regular expressions as patterns to extract features from semi-structured and narrative text. Eric Brill [Bri00] applied his transformation-based learning framework to learn reduced regular expressions for correction of grammatical errors in text. A crucial difference between these two approaches and the one presented in this chapter is that WHISK and Brill’s approaches require the user to identify the precise location of features for labeling while the proposed approach requires only that instances (segments) be labeled. Moreover, regular expressions are more general in that the inclusion of the logical “OR” operator is supported, while Brill’s approach does not.

Michael Chau, Jennifer J. Xu, and Hsinchun Chen have published results of research on extracting entities from narrative police reports [CXC02]. They employed a neural network to extract names, addresses, narcotic drugs, and items of personal property from these reports. Noun phrases are candidates for name entities. Although not readily apparent in [CXC02], they evidently employ a similar approach to other researchers in that feature-specific labeling is required in training set development. Their cross-validation results vary from a low of 46.8% to a high of 85.4% for various entities. In the approach discussed below, however, significantly better results have been achieved without a restriction to noun phrases. In addition, a larger number of features
can be extracted for use in various analyses, including matching on modus operandi. Much work has been done in patent citation analysis (e.g., [HJT00, BTC00]). For example, patent citation frequencies are employed to ascertain the relative importance of patents. An approach of this nature, however, does not shed light on the content of the patent. To the best of our knowledge, this effort to develop technology to extract content-based PSIs from patents is novel.

3.7 Definitions

In this section, we start with the standard definition of a regular expression, and then define a reduced regular expression as used in our algorithm. Following this, we define terms used in the remainder of the chapter.

**Regular expression:** “Given a finite alphabet , the set of regular expressions over that alphabet is defined as:

1. \( r, s \in \sum \) is a regular expression and denotes the set \( r, s \).

2. if \( r \) and \( s \) are regular expressions denoting the languages \( R \) and \( S \), respectively, then \( (r+s), (rs), \) and \( (r^*) \) are regular expressions that denote the sets \( R \cup S, RS \) and \( R^* \) respectively.” [HU79, Bri00]

**Reduced regular expression (RRE):** Our reduced regular expression is at first glance similar to that defined in [Bri00]. However, there are some significant differences. Given a finite alphabet , our reduced regular expression is defined as a set, where the star ‘*’ indicates that the character immediately to its left may be repeated any number of times, including zero, and the question mark ‘?’ indicates that the character immediately to its left may be repeated either zero times or one time.

1. \( \forall \alpha \in \sum \), \( \alpha \) is a regular expression denoting the set \( \alpha \).

2. \( \sim \alpha^* \) is a RRE denoting the Kleene closure of the set \( \sum \).

3. \( \sim \in \sum \), \( s \in \sum \), where ‘*’ is the start of a line, and \( s \) is the end of a line.

4. \( \{s, S, \ w, \ W\} \subset \sum \), where \( \{s (|t|n|r|f)\} \) is any white space, \( \{S (|t|n|r|f)\} \) is any character except white space, \( \{w (0-9a-zA-Z)\} \) is any alphanumeric character, and \( \{W (0-9a-zA-Z)\} \) is any non-alphanumeric character.

5. All words in the lexicon and all part of speech tags in the Penn tag corpus [MS00] belong to \( \sum \).

6. \( \{w\}^* \) is a RRE denoting the Kleene closure of the set \( \{w\} \).

7. \( \{w\}_{i,j} \) is a RRE denoting that \( w \) is repeated between \( i \) and \( j \) times, where \( i \geq 0 \), and \( j \geq i \).

8. \( \alpha? \) is a RRE denoting that \( \alpha \) is an optional part of the RRE.
9. if \( r \) and \( s \) are RREs denoting the languages \( R \) and \( S \) respectively, then \( (r+s) \) and \( (rs) \) are RREs that denote the sets \( R \cup S \) and \( RS \), respectively.

Some examples of regular expressions that are not RREs are: “\( \alpha \ast \)”, “\( (\alpha \beta) \ast \)”, and “\( \alpha \ast + \)”. We have not found it necessary to support such regular expressions to achieve high accuracies.

Other important terms used in the remaining sections of this chapter are briefly defined below.

**Feature** A feature is the smallest unit of information extracted. Examples include values for the attributes height, weight, age, gender, time, location, PSI, and so on.

**Segment** A segment is (a portion of) a sentence.

**Item** An item is a document from which features are extracted. In the experiments described herein this is either a police incident report or a full text patent.

**True set** If the system is learning a RRE for a feature \( f \), then the true set consists of all segments labeled \( f \) in the training set.

**False set** If the system is learning a RRE for a feature \( f \), then the false set consists of all segments that are not labeled \( f \) in the training set.

**Element** Words in the RRE with frequency in the true set higher than a threshold \( \varepsilon_{Word} \) and part of speech tags in the RRE with frequency in the true set higher than a threshold \( \varepsilon_{Tag} \) are termed elements of the RRE.

**Root** We term the first element found by the algorithm in a RRE the root of the RRE.

**\( R_{and} \)** The RRE learned after completion of the “AND” learning process.

**N** The number of elements in \( R_{and} \).

### 3.8 Approach to RRE Discovery

In this section, an approach to the discovery of RREs from a small set of labeled training segments is presented. The process begins with the processing of datasets. Next, a greedy algorithm is applied. Finally, RREs for the same feature are combined to form a single RRE.

#### 3.8.1 Pre-Processing

Pre-processing includes segmentation, feature identification, segment labeling, and part of speech tagging. Each item is split into segments at this stage and becomes an instance in our system. We assume that no features cross segments. This assumption is practical for a number of important features, including those listed in Table 3.2. A domain expert must identify features that will be extracted such as Height, Weight, Eye
Figure 3.4: RRE Discovery

Color, etc. Each segment is then assigned labels that correspond to the set of features present. After labeling, each feature has a true set corresponding to true positive segments and a false set corresponding to true negative segments. Finally, each word is (automatically) assigned its part of speech [MS00].

3.8.2 Learning Reduced Regular Expressions

Our focus is on the discovery of word sequences and/or part of speech tags that have high frequency in a collection of segments, while having low frequency outside the segments. The algorithm first discovers the most common element of an RRE, termed the root of the RRE. The algorithm then extends the ‘length’ of the RRE in the “AND” learning process. During the “OR” learning process, the ‘width’ of the RRE is extended. Next, optional elements are discovered during the “Optional” learning process. The algorithm then proceeds with the “NOT” learning process, and finally discovers the start and the end of the current RRE. Figure 3.4 depicts the entire learning process.

Our approach is based on a semi-supervised learning method for feature extraction from police incident reports. Instead of labeling the exact location of features in a training set, the training-set developer need only record whether a specific feature of interest occurs in a sentence segment. The rules learned by the algorithm are for exact feature extraction from police reports. Therefore, the approach is a semi-supervised learning in this case. On the other hand, the approach is a supervised learning for PSI because PSI does not require an exact match. The training-set developer tags whether a sentence is a PSI or not. The rules learned by the algorithm are also used to tag whether a sentence is a PSI or not. Thus, PSI is a supervised classification problem.

A covering algorithm is employed in this approach. After one RRE is generated, the algorithm removes all segments covered by the RRE from the true set. The remaining segments become a new true set and the steps in Figure 3.4 repeat. The learning process stops when the number of segments left in the true set is less than or equal to a threshold \( \delta \). We describe the details of the first three steps of the algorithm in what follows. [WP03a] describes the details of the other steps.

To discover the root of a RRE, the algorithm matches each word and/or part of speech tag (specified as a simple RRE) in each true set segment against all segments. The performance of each such RRE in terms of the metric (see [Rij79])

\[ F_\beta = \frac{((\beta^2 + 1)PR)}{\beta^2 P + R} \quad (3.1) \]

is considered. In this formula, \( P = \text{precision} = TP/(TP + FP) \) and \( R = \text{recall} = TP/(TP + FN) \), where \( TP \) are true positives, \( FP \) are false positives, and \( FN \) are false.
negatives. The parameter $\beta$ enables us to place greater or lesser emphasis on precision, depending on our needs. The word or part of speech tag with the highest score is chosen as the root of the RRE. In other words, the algorithm discovers the word or part of speech tag that has a high frequency of occurrence in segments with the desired feature (the true set). Meanwhile, it must also have low frequency in segments that do not contain the desired feature (the false set). Our approach places less emphasis on precision and more on recall during the root discovery process. We use the parameter $\beta_{\text{root}}$ to control this. Naturally this results in a larger set of segments that match the root. These segments, however, are not necessarily all true positives. As a result, the AND and NOT learning phases all prune false positives from the set of strings that match the root RRE. The result is both high precision and high recall.

After the root is discovered, the algorithm places new candidate elements immediately before and after the root, thereby forming two new RREs. Any word or part of speech tag (other than the root itself) can be used in this step. The RRE with the highest score will replace the previous RRE. The new RRE is then extended in the same way. Adjacency implies the use of an “AND” operator. As before, candidate words and parts of speech are inserted into all possible positions in the RRE. The algorithm measures the performance of each new RRE and the one with the highest score is selected if its score is greater than or equal to the previous best score. In this sense the algorithm is greedy. The RRE learned after this step is termed $R_{\text{and}}$. The overall complexity of this step is $O(N^2)$, where $N$ is the number of elements in $R_{\text{and}}$ [WP03a].

After the “AND” learning process is complete, the algorithm extends the RRE with words and part of speech tags using the “OR” operator. For each element in $R_{\text{and}}$, the algorithm uses the “OR” operator to combine it with other words and/or part of speech tags. If the newly discovered RRE has a better F-measure than the previous RRE, the new RRE will replace the old one. The complexity of “OR” learning process is $O(N)$. The “Optional” learning process and the “NOT” learning process are then applied. Each of them has complexity $O(N)$. Finally the algorithm discovers the start and the end of the current RRE. For details on these three steps, please refer to [WP03a].

The “Optional” learning process and “NOT” learning process are designed especially for exact match of feature extraction from police incident reports. In PSI discovery, the algorithm only requires steps 1, 2, and 3 in Figure 3.4. Any segment (sentence) that matches the RRE generated is considered a PSI. Otherwise, it is not a PSI.

### 3.8.3 Post Processing

During each iteration in Figure 3.4, one RRE is generated. This RRE is considered a sub-pattern of the current feature. After all RREs have been discovered for the current feature (i.e., all segments labeled by the feature are covered), the system uses the “OR” operator to combine the RREs.

In this section, we have described a greedy covering algorithm that discovers a RRE for a specific feature in narrative text. The basic idea is to find high frequency patterns in segments/sentences associated with the feature. We have applied “AND”, “OR”, and “NOT” operators to find elements of a RRE that accept sub-patterns of the feature under consideration. Optional elements as well as the start and the end of a sentence...
Table 3.2: 10-fold cross-validation performance on police incident report data

<table>
<thead>
<tr>
<th>Feature</th>
<th>Average Precision %</th>
<th>Average Recall %</th>
<th>Average F-measure %</th>
<th>Average # of True Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>97.27</td>
<td>92.38</td>
<td>94.34</td>
<td>13</td>
</tr>
<tr>
<td>Date</td>
<td>100</td>
<td>94.69</td>
<td>97.13</td>
<td>8.8</td>
</tr>
<tr>
<td>Time</td>
<td>100</td>
<td>96.9</td>
<td>98.32</td>
<td>8.9</td>
</tr>
<tr>
<td>Eye Color</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Gender</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>33.6</td>
</tr>
<tr>
<td>Hair Color</td>
<td>100</td>
<td>60</td>
<td>60</td>
<td>0.8</td>
</tr>
<tr>
<td>Height</td>
<td>100</td>
<td>98</td>
<td>98.89</td>
<td>2.4</td>
</tr>
<tr>
<td>Race</td>
<td>95</td>
<td>96.67</td>
<td>94.67</td>
<td>3.3</td>
</tr>
<tr>
<td>Week day</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>9.8</td>
</tr>
<tr>
<td>Weight</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>1.9</td>
</tr>
</tbody>
</table>

are also components of the RRE. Finally, RREs for all sub-patterns are combined to form a single RRE with the “OR” operator.

### 3.9 Experimental Results

In this section, we first briefly summarize the result of feature extraction from police incident reports. Following this, we discuss how the training and testing datasets used in PSI extraction were constructed. Finally, we present the results of the widely applied technique of cross-validation to evaluate our models for PSI extraction from full text patents.

Table 3.2 summarizes the results of 10-fold cross validation based on 100 police incident reports consisting of 1404 segments. 10-fold cross validation involves splitting the training data into ten pieces. Nine of the pieces are subsequently used for training and the remaining piece for validation (testing). This process repeats nine times. Each time, each of the other nine pieces is used for validation. The average performance of the ten validation (testing) runs becomes the final result of the 10-fold cross validation. There are ten different features evaluated in Table 3.2 (first column). Eye Color, Gender and Weekday have perfect test performance (100%). The performance of Age, Date, Time, Height, Race, and Weight are also excellent (F-measure scores \( \geq 90\% \)). The performance of Hair Color is however, not as good. This is due to the lack of Hair Color segments in some folds of the training sets. However, the test performances on other folds, in which there are Hair Color segments, are perfect (100%). As a result, we conclude that the RREs discovered for these ten features are high-quality.

For PSI extraction, our datasets include 55 patents, of which 15 containing 7,723 segments (sentences) were used in cross-validation. These patents were retrieved in the focused domain of text mining to enable us to label the training data more easily. Each segment in each patent was manually tagged by a human expert, thereby creating our ground truth. We split all true segments randomly into 10 folds for cross-validation.
Each fold was given a roughly equal number of true segments (i.e., the folds were stratified). We did the same thing with the false segments.

As part of our experimental method we employed resampling to vary the ratio of true segments to false segments in the training sets. Figure 3.5 depicts how precision, recall, and F-measure score vary with the ratio of true to false segments.

From Figure 3.5 it can be seen that the F-measure reaches an optimum when the ratio of true to false segments is 1:73, which in fact happens to be the original ratio in the source patents. Naturally, we do not resample the test folds – otherwise, the 10-fold cross-validation performance would not reflect the true testing performance on data representative of the domain.

![Figure 3.5: Performance curve of false/true segment ratio.](image)

Table 3.3 depicts optimal parameter values for PSI extraction. In each case, we varied each parameter in turn while holding the others fixed in order to determine an optimal value. Figures 3.6 and 3.7 depict the results of varying as an example of this approach to optimization of parameter values. The plots in Figures 3.6 and 3.7 are drawn from the average testing results based on 10-fold cross validation.

It is interesting to note that in Figure 3.6, the higher the value of $\beta$ (see Equation (3.1)), the higher the average recall and the lower the precision. Meanwhile, the number of true positives increases as $\beta$ increases (Figure 3.7). In practice, precision and recall are inversely related [Cle72], that is, one generally has to trade off precision to increase recall and vice versa. Our intuition in PSI extraction is that it is more important to extract at least one PSI per patent than to extract several PSIs that are mixed with segments that are not PSIs. In this sense, precision is more important than recall in PSI extraction. Although this is intuitive, in this particular case we did not select 0.25 for $\beta_{Others}$ in Table 3.3 due to the fact that there were too few true positives extracted. Moreover, the F-measure metric has the best performance when $\beta_{Others}$ equals 0.5, a
value that still reflects a focus on precision but has better recall. Therefore, we chose 0.5 as the optimal value for $\beta_{Others}$ in Table 3.3.

We tuned all the other parameters in the same way. It is possible for the parameters $\varepsilon_{Word}$ and $\varepsilon_{Tag}$ to be over 100% because the same word or part of speech tag may occur more than once in a single sentence. The final performance after all parameters have been optimized is not guaranteed to be globally optimal since our method to tune parameters is a greedy approach. The parameter values used for final cross-validation tests reported in Table 3.4 are depicted in Table 3.3.

Table 3.3: Optimal Parameter Values.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta_{Root}$</td>
<td>6</td>
</tr>
<tr>
<td>$\beta_{Others}$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\varepsilon_{Word}$</td>
<td>5%</td>
</tr>
<tr>
<td>$\varepsilon_{Tag}$</td>
<td>400%</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>5</td>
</tr>
</tbody>
</table>

We performed 10-fold cross validation based on the configuration listed in Table 3.3 with a 1:73 ratio of true to false segments in the training set. The test results of the cross-validation are shown in Table 3.4.

Table 3.4: 10-fold cross-validation test performance on patent data.

<table>
<thead>
<tr>
<th>Test sets</th>
<th>Precision %</th>
<th>Recall %</th>
<th>F-measure</th>
<th># of True Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>85.71</td>
<td>60.00</td>
<td>70.59</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>57.14</td>
<td>40.00</td>
<td>47.06</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>62.50</td>
<td>50.00</td>
<td>55.56</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>66.67</td>
<td>40.00</td>
<td>50.00</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>37.50</td>
<td>30.00</td>
<td>33.33</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>42.86</td>
<td>30.00</td>
<td>35.29</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>40.00</td>
<td>8.18</td>
<td>25.00</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>50.00</td>
<td>7.27</td>
<td>35.29</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>71.43</td>
<td>5.45</td>
<td>55.56</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>44.44</td>
<td>36.36</td>
<td>40.00</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td>55.83</td>
<td>37.73</td>
<td>44.77</td>
<td>3.9</td>
</tr>
</tbody>
</table>

The average precision is 55.83%. That means over half of the sentences extracted using the RREs discovered by our algorithm contained information relevant to the problem solved by patents. Considering the complexity of natural language expressions used in patents, we consider this result promising. The average recall is 37.73%. This
Figure 3.6: Performance of $\beta_{Others}$.

Figure 3.7: True positives based on $\beta_{Others}$. 
value is acceptable because as noted previously, precision is more important than recall in this particular application. The average F-measure with $\beta = 1$ is 44.77%. This value tells us that we are currently successfully extracting PSI-related segments about half the time. Another important metric is the distribution of correctly extracted PSIs across patents. In order to assess our algorithm’s performance with regard to this metric, we measured the distribution of true positives from the 10 test folds across the 15 patents used to form the training set. Ideally, we would like to extract one or more PSIs from each patent. In this case, 80% of the original 15 patents were covered by at least one PSI. This result too is quite promising.

These experimental results provide evidence that the approach to RRE discovery can be usefully applied to extract features from police incident reports and PSIs from patents. With the former application we achieved very good test set performance, and with the latter we achieved reasonable and stable test set performance.

### 3.10 Concluding Remarks

The object-oriented software environment GTP (General Text Parser) with network storage capability has been designed to provide a scalable solution to index growing and dynamic text collections. It allows IR professionals to create, store, and share an index via a remote network. The addition of network storage capability certainly addresses the problem of inadequate storage and file sharing over the network. GTP with network storage gives users an opportunity to create a user-specific IR model, place the files (index) generated by GTP on a sharable network so that all the participants in a project, no matter where they are located, can have access to them.

In the domain of textual-based feature extraction, an algorithm for feature extraction from police incident reports and patents has been presented. The algorithm can be plugged into GTP as a component that would aid in the indexing process. The algorithm can be used to learn reduced regular expressions that are used as patterns to match and extract previously unseen features with a high degree of reliability. Experiments have demonstrated that reduced regular expressions extract information useful in criminal justice, homeland defense, and patent intelligence applications.

### 3.11 Acknowledgements

Co-authors W.M. Pottenger and T. Wu gratefully acknowledge the continuing help of their Lord and Savior, Yeshua the Messiah (Jesus Christ) in their lives. The research of S.Y. Mironova and M.W. Berry was supported in part by the National Science Foundation under grant CISE-EIA-99-72889, and that of S. Atchley and M. Beck was supported by the U.S. Department of Energy (SciDAC) under grant DE-FC02-01ER25465 and by the National Science Foundation under grant ANI-99-80293.

All the co-authors thank the anonymous referees for their helpful comments and suggestions for improving the chapter.
Bibliography


[WP03b] T. Wu and W.M. Pottenger. A semi-supervised algorithm for pattern discovery in information extraction from textual data. In The Seventh Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Seoul, Korea, April 2003.