MINING, INDEXING, AND SEARCH APPROACHES TO ENTITY 
AND GRAPH INFORMATION RETRIEVAL FOR 
CHEMoinFORMATICS

A Dissertation in
Computer Science and Engineering
by
Bingjun Sun

© 2008 Bingjun Sun

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2008
The dissertation of Bingjun Sun was reviewed and approved* by the following:

C. Lee Giles  
Professor of Information Sciences and Technology  
Dissertation Co-Advisor, Co-Chair of Committee

Prasenjit Mitra  
Assistant Professor of Information Sciences and Technology  
Dissertation Co-Advisor, Co-Chair of Committee

Jesse Barlow  
Professor of Computer Science and Engineering

Robert T. Collins  
Associate Professor of Computer Science and Engineering

Bing Li  
Professor of Statistics

Raj Acharya  
Professor of Computer Science and Engineering  
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.
Abstract

Traditional generic search engines using textual keyword matching do not support domain specific searches. However, different domains may have different domain specific searches. For example, Chemical research is molecule centric rather than document centric. Usually a chemical molecule can be represented in multiple ways, e.g., textual chemical entities such as chemical names and formulae, and chemical structures such as 2D graphs and 3D graphs. Thus, in Chemoinformatics, chemical entity searches and chemical structure searches are more important than simple document searches using keyword matching. In this work, we show how to build a domain specific search engine that enables both entity and 2D graph searches for chemical molecules. First of all, documents are collected from the Web, and then preprocessed using document classification and segmentation. We apply Support Vector Machines for classification and propose a novel method of text segmentation. Then chemical entities in the documents are tagged and indexed to provide fast searches. Simultaneously, chemical structure information are collected, processed, and indexed for fast graph searches.

Many issues exist to support textual chemical entity searches. Chemical names and formulae usually appear in chemical documents when corresponding molecules are mentioned, but a chemical molecule can have different textual representative ways. A simple keyword search would retrieve only the exact match and not the others. Additionally, ambiguous non-chemical terms such as “He” are retrieved. We show how chemical entity searches can improve the relevance of returned documents by avoiding those ambiguous terms. Our search engine first extracts chemical entities from text, performs novel indexing suitable for chemical names and formulae, and supports different query models that a scientist may require. We propose a model of hierarchical conditional random fields for entity tagging that considers long-term dependencies at the sentence level. Then to support efficient and effective entity searches, we propose two feature selection methods for entity
index building. One is to select frequent and discriminative subsequences from all the candidate features for chemical formula indexing. The other is to first discover subterms of chemical names with corresponding probabilities using a proposed independent frequent subsequence mining algorithm, and then segment a chemical name hierarchically into a tree structure based on discovered independent frequent subsequences. A unsupervised hierarchical text segmentation (HTS) method is proposed for this. Then subterms on the HTS tree can be indexed. Finally, query models with corresponding ranking functions are introduced for chemical entity searches. Experiments show that our approaches to chemical entity tagging, indexing, and search perform well.

In addition to text searches, massive amounts of structured data in Cheminformatics and Bioinformatics raise an issue of effective and efficient structure searches. Graphs are used to model structures for a long time. A typical type of graph search is a subgraph query that retrieves all graphs containing the query graph. To support efficient search, graph features such as subgraphs are extracted from graphs for graph indexing. Then, for a given subgraph query, graph candidates that may contain the subgraph are retrieved using the index, and subgraph isomorphism tests are performed to scan the candidates. However, since the space of all the possible subgraphs of the whole graph set is prohibitively large, feature selection for index pruning is required. Thus, one of the key issues is, given the set of all the possible subgraphs of the graph set, which subset of features is the optimal to achieve the highest precision for a given query set? To answer this question, first, we introduce a graph search process for subgraph queries. Then, we propose several novel feature selection criteria, Max-Precision, Max-Irredundant-Information, and Max-Information-Min-Redundancy, based on pointwise mutual information. We also propose a greedy feature search algorithm to find a near optimal feature set based on Max-Information-Min-Redundancy. Finally we show empirically that our proposed methods achieve a higher precision than previous methods.

Besides subgraph queries, users often execute similarity graph queries to search for graphs similar to the query graph. Previous methods usually use the maximum common edge subgraph (MCEG) to measure the similarity of two graphs. However, MCEG isomorphism is NP-hard and prohibitively slow to scan all graphs in the data set, so that it is not feasible for online searches to use MCEG in the fly to measure the similarity of retrieved graphs. We propose a novel approach to this issue from a different view by ranking search results using a weighted linear graph kernel that can avoid real time MCEG isomorphism tests, but achieve a reasonably high quality of search results. First, subgraphs features are extracted from graphs and indexed. Second, for a given graph query, graphs containing its subgraph features are retrieved and similarity scores are computed based on the indexed subgraph features. Finally, graphs are sorted and returned based on
similarity scores. Subgraph weights used by the graph kernel are learned offline from a training set generated using MCEG isomorphism. We show empirically that the proposed methods of learning to rank for similarity graph queries can achieve a reasonably high normalized discounted cumulative gain in comparison with the “gold standard” method based on MCEG isomorphism. Moreover, our method can be applied to learn other similarity metrics, such as explicit knowledge provided by domain experts or implicit knowledge from user logs.
# Table of Contents

List of Figures x
List of Tables xii
Acknowledgments xiii

Chapter 1
Introduction 1
1.1 Motivation .......................................... 1
1.2 Proposed Framework ................................. 3
1.3 Organization and Contributions .................... 6

Chapter 2
Text Classification and Segmentation 8
2.1 Background and Related Work ...................... 8
2.2 Text Classification .................................. 10
   2.2.1 Support Vector Machines ...................... 11
2.3 Text Segmentation .................................. 11
   2.3.1 Mutual Information and
   Weighted Mutual Information .................... 12
   2.3.2 Iterative Greedy Algorithm .................. 15
2.4 Experimental Evaluation .......................... 19
   2.4.1 Single-document Segmentation ................. 20
   2.4.2 Shared Topic Detection ....................... 22
   2.4.3 Multi-document Segmentation ................. 23

Chapter 3
Textual Entity Extraction 28
3.1 Background and Related Work ........................................ 28
3.2 Conditional Random Fields ........................................ 30
3.3 Imbalanced Data Classification and Tagging ...................... 33
3.4 Hierarchical Conditional Random Fields ......................... 35
3.5 Chemical Entity Extraction ......................................... 37
  3.5.1 Chemical Name Extraction ...................................... 39
  3.5.2 Chemical Formula Extraction .................................. 41
3.6 Experimental Evaluation ........................................... 42
  3.6.1 Experiment Data and Design .................................. 42
  3.6.2 Experiment results ............................................ 44

Chapter 4
Textual Entity Indexing and Searching 50
4.1 Background and Related Work ...................................... 50
4.2 Preliminaries ....................................................... 53
4.3 Segmentation-Based Indexing ...................................... 54
  4.3.1 Independent Frequent Subsequence Mining .................. 54
  4.3.2 Hierarchical Text Segmentation ............................... 57
4.4 Frequency-and-Discrimination-Based Indexing .................. 60
4.5 Chemical Formula Searches ........................................ 63
  4.5.1 Query Models .................................................. 63
  4.5.2 Ranking Functions .............................................. 64
4.6 Chemical Name Searches .......................................... 66
  4.6.1 Query Models .................................................. 66
  4.6.2 Ranking Functions .............................................. 67
4.7 Document Search ................................................... 67
4.8 Experimental Evaluation ........................................... 68
  4.8.1 Independent Frequent Subsequence Mining and Hierarchical Text Segmentation ................................. 68
    4.8.1.1 Experiment Data and Design ............................. 68
    4.8.1.2 Experiment results ..................................... 69
  4.8.2 Textual Entity Information Indexing .......................... 70
    4.8.2.1 Chemical Formula Indexing ............................... 70
    4.8.2.2 Chemical Name Indexing .................................. 72
  4.8.3 Textual Entity Information Search ............................ 73
    4.8.3.1 Chemical Formula Search ................................ 73
    4.8.3.2 Chemical Name Search .................................. 74
  4.8.4 Entity Disambiguation in Document Search .................. 75
    4.8.4.1 Experiment Data and Design ............................. 75

vii
Chapter 7
   Conclusions and Future Work 126
  7.1 Conclusions ........................................ 126
  7.2 Future Work ..................................... 128

Bibliography 131
## List of Figures

1.1 Frame of *ChemX Seer*’s Chemical Entity, Structure, and Document Search ........................................ 4

2.1 Illustration of multi-document segmentation and alignment ........ 13
2.2 Error rates for different hyper parameters of term weights. ........ 24
2.3 Term weights learned from the whole training set .................. 24
2.4 Change in (weighted) MI for $MI_l$ and $WMI_l$ ....................... 26
2.5 Time to converge for $MI_l$ and $WMI_l$ .............................. 26

3.1 Illustration of trade-off tuning between precision and recall in SVMs 34
3.2 Illustration of Hierarchical Conditional Random Fields .......... 36
3.3 Ambiguity of Chemical Formulae in Text Documents ............... 38
3.4 CRFs and HCRFs for chemical formula tagging with different feature sets and different values of feature boosting parameter $\theta$ .... 46
3.5 SVM and LASVM for chemical formula tagging with different values of threshold $t$ ............................................. 47
3.6 Running time of chemical formula tagging including feature extraction 47
3.7 CRF for chemical name tagging using different feature sets and different values of feature boosting parameter $\theta$ ..................... 48

4.1 An Example of Independent Frequent Subsequence Mining .......... 57
4.2 Illustration of Hierarchical Text Segmentation ...................... 58
4.3 Mining Independent Frequent Substrings ............................ 69
4.4 Examples of hierarchical text segmentation .......................... 70
4.5 Features and index size ratio after feature selection for formula indexing ...................................................... 70
4.6 Correlation of similarity formula search results after feature selection 71
4.7 Running time of feature selection for formula indexing ............ 72
4.8 Ratio of after v.s. before index pruning for name indexing .......... 73
4.9 An example of similarity formula search results in *ChemX Seer* ... 74
4.10 Correlation of name search results before and after index pruning . 75
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.11</td>
<td>Precision in Document Search using Ambiguous Formulae</td>
<td>76</td>
</tr>
<tr>
<td>5.1</td>
<td>A subgraph query and its support</td>
<td>80</td>
</tr>
<tr>
<td>5.2</td>
<td>Average precision of graph search for subgraph queries</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>Response time of subgraph queries for cases in Table 5.2</td>
<td>99</td>
</tr>
<tr>
<td>5.4</td>
<td>Effect of max enumerated subgraph size on response time for query</td>
<td></td>
</tr>
<tr>
<td></td>
<td>size of 20 &amp; MImR.F</td>
<td>100</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of MP, MII, and MImR in terms of feature selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>time and precision increasing rate</td>
<td>102</td>
</tr>
<tr>
<td>6.1</td>
<td>Similarity graph query and search results (MCEGs are the bold parts)</td>
<td>106</td>
</tr>
<tr>
<td>6.2</td>
<td>NDCG 1-20 for all queries</td>
<td>121</td>
</tr>
<tr>
<td>6.3</td>
<td>NDCG 1-20 for queries having no support</td>
<td>122</td>
</tr>
<tr>
<td>6.4</td>
<td>Response time of graph search using MCEG and graph kernel</td>
<td>124</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Average Error Rates of Single-document Segmentation Given Segment Numbers Known</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Single-document Segmentation: P-values of T-test on Error Rates</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Shared Topic Detection: Average Error Rates for Different Numbers of Documents in Each Subset</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Average Error Rates of Multi-document Segmentation Given Segment Numbers Known</td>
<td>23</td>
</tr>
<tr>
<td>2.5</td>
<td>Multi-document Segmentation: P-values of T-test on Error Rates for $MI_l$ and $WMI_l$</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>Multi-document Segmentation: Average Error Rate for Document Number = 5 in Each Subset with Different Number of Term Clusters</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Average accuracy of sentence tagging</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>Average accuracy of formula tagging</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>P-values of 1-sided T-test on F-measure for formula tagging</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Average accuracy of name tagging, $\theta = 1.0$</td>
<td>48</td>
</tr>
<tr>
<td>4.1</td>
<td>The most frequent subterms at each length, $Freq_{min} = 160$</td>
<td>69</td>
</tr>
<tr>
<td>5.1</td>
<td>Notations used throughout</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Average precision for feature selection methods</td>
<td>98</td>
</tr>
<tr>
<td>5.3</td>
<td>One-sided T-test for feature selection methods</td>
<td>98</td>
</tr>
<tr>
<td>6.1</td>
<td>Notations used throughout</td>
<td>107</td>
</tr>
<tr>
<td>6.2</td>
<td>Average NDCGs</td>
<td>123</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to thank my principal advisor, Professor C. Lee Giles, for his great guidance and support. He has provided me not only the technical knowledge but also a rigorous attitude towards research, for which I am very grateful. I would like to thank my associate advisor, Professor Prasenjit Mitra, for his guidance and support, especially having many discussions with me and helping me on paper writing. I appreciate their tremendous caring about students. I feel fortunate to have had them as my advisors. I would also express my gratitude to my committee, Professor Jesse Barlow, Professor Robert T. Collins, and Professor Bing Li. My gratitude also goes to Professor Hongyuan Zha and Professor John Yen for previous advice during my Ph.D.’s study.

I would like to thank the wonderful members of our research group and of Department of Computer Science and Engineering and College of Information Sciences and Technology for their help and friendship, Qiankun Zhao, Ziming Zhuang, Ding Zhou, Yang Song, Yang Sun, Huajing Li, Jian Huang, Shuyi Zheng, Qingzhao Tan, Ying Liu, Levent Bolelli, Juan Pablo Fernandez Ramirez, Saurabh Singh Kataria, Sujatha Das, Xiaonan Lu, Bi Chen, Xiao Zhang, Bo Luo, Fengjun Li, Jing Zhao, Yang Zhang. I also would like to thank all the people at the Pennsylvania State University, including faculty and staff for their help (especially Vicki Keller) during my Ph.D.’s study.

I would like to appreciate my parents Guangyou and Dongxiu for their endless support, guidance and love during each stage of my life. I also would like to thank my brother Nanfei who goes ahead with me for many years.

This work was supported partially by the National Science Foundation grant 0535656.
Chapter 1

Introduction

The burst of knowledge in the current information age exposes people to billions of data. The double-edged phenomenon, on the one hand, overwhelms people, including scientists and researchers, by an overload of information; however, on the other hand, it generates new challenges as well as opportunities for users and researchers. How to efficiently and effectively access relevant information from such a huge amount of data? This is one of the key issues for the research and applications of computer and information sciences, especially the area of information retrieval. Diverse applications and technologies, from database systems to search engines, from artificial intelligence to machine learning, from computer science to statistics, have been proposed and developed to enable people to handle the huge amount of information. Intelligence of machines to understand data is the key solution during the integration of various technologies in such systems with the goal to release the stress of massive information on researchers and simultaneously without loss of accuracy.

1.1 Motivation

Recently, massive amount of scientific data, including unstructured data such as articles, and structured data such as molecule or protein structures, are being published on the Web. In Chemoinformatics, research is molecule centric rather than document centric. Scientists, especially chemists, often desire to search chemical molecules from the published data. Two categories of methods to express such
searches: 1) using string queries, such as the chemical names or formulae, to search for molecules from the literature, and 2) using graphs representing chemical structures to search for molecules from the structure database. Current general purpose search engines do not support users searching for data using chemical formulae, names, or graphs. In this work, we propose a general framework of a chemical search engine, providing various query models of chemical molecules as well as document searches, which is one of the domain-specific information retrieval areas [1, 2].

When a scientist searches for chemical molecules in the literature using a general purpose search engine today, she can input queries of names or formulae that are tokenized into keywords, and articles are returned where the exact keyword strings are found. However, general purpose search engines searching for the exact occurrences of keywords have the following four problems for this specific domain. First, chemical names or formulae in documents and queries cannot be tokenized correctly using tokenizers of natural languages. For example, the chemical name, “Benzene, 1-methyl-4-(1-methylethyl)-”, is segmented into several tokens. Consequently the index cannot be constructed based on tokens with unique meanings, and a chemical name or formula in a query may also be tokenized into several tokens, which cannot represent the original molecule. Second, ambiguity and polysemy exist for terms in chemical articles. For example, “He” may refer “Helium” or the pronoun “he”, and C2H4O2 may refer “acetic acid”, “methyl formate”, or “ethen-1,2-diol”. Third, synonyms exist for chemical molecules, e.g., “acetic acid” can also be mentioned as “vinegar acid”, CH3COOH, or C2H4O2. Fourth, scientists may be interested in molecules with similar chemical formulae or names instead of just a particular one. To remedy these deficiencies, in the general framework of the chemical information retrieval, we propose a chemical-entity-aware search engine with various query models of chemical entities, including chemical name and formula searches. To build such a search engine, the following problems must be solved: 1) extract chemical names and formulae from text documents, 2) index chemical names and formulae, 3) design query models and ranking functions for chemical name and formula search, and 4) rewrite queries using the results of chemical entity searches for document search. Furthermore, user interaction is provided so that scientists can choose the relevant query expansions.
Besides the chemical entity searches in the literature, scientists often seek to search for published chemical molecule structures on the Web. Efficient and effective search of such kind of structured data is increasingly desired in many areas including Chemoinformatics and Bioinformatics. Graphs are widely used to model complicated structures such as chemical molecules, proteins, XML files, etc. Users often use two types of queries to search desired graphs: 1) a *subgraph query* that finds all graphs containing the query graph from the data set, and 2) a *similarity query* that finds graphs similar to a query graph. To support efficient searches, *subgraphs* are extracted from graphs and selected for graph indexing. However, since the subgraph space of the graph set is prohibitively large, subgraph features are too many to construct a graph index using all of them. Thus, feature selection is required to select an *optimal* or *near-optimal* subset of subgraph features that contains as much information as possible to index graphs. Moreover, search results, especially results for similarity graph queries, require to be ranked in terms of similarity or relevance between each retrieved graph and the query graph. Previous methods usually use the maximum common edge subgraph (MCEG) to measure the similarity of two graphs. However, two problems exist. First, the algorithm of MCEG isomorphism is prohibitively slow to be executed on the fly. Second, MCEG may not be the optimal metric to measure the similarity between two graphs in many situations. New ranking methods are desired to support effective and efficient graph searches. These kind of ranking methods should be able to 1) perform fast online and 2) learn ranking functions offline from explicit knowledge provided by domain experts or implicit knowledge from user logs.

### 1.2 Proposed Framework

As mentioned before, we propose a framework of a chemical search engine that can support chemical entity searches, document searches, and structure searches. The proposed search engine framework is illustrated in Figure 1.1, which is an integral part of *ChemX Seer* \(^1\), a digital library for chemistry. Our proposed framework mainly involves two relatively independent parts: 1) textual entity mining, indexing, and search, and 2) graph mining, indexing, and search.

\(^1\)http://chemxseer.ist.psu.edu/
For the part of textual entity search, the first step is using the focused crawler to collect all the raw data from the Web, including PDF and HTML documents. Then PDF documents are converted in text documents. After that, document pre-processing such as classification and segmentation is applied to clean and prepare the data for entity extraction, indexing, and search. Next, the whole problem of chemical entity and document searches can be addressed in three stages: 1) mining textual chemical entity information from the documents, 2) indexing textual chemical entity information and the documents, and 3) querying textual chemical entity information and the relevant documents. In the first stage, we need to tag entity terms representing textual chemical molecules. Previous research works on detecting names [3] and biological entities [4] use a broad range of techniques from rule-based methods to machine-learning-based ones. Among these approaches, the machine-learning-based approaches utilizing domain knowledge perform the best.
because they can mine implicit rules as well as utilize prior domain knowledge in the framework of statistical models to improve the overall performance. In the second stage, each chemical entity (name or formula), is then analyzed, tokenized, and indexed. Finally, online search services are provided for document and entity queries. Every time a query is accepted, the query string is analyzed. Chemical entity searches in the query string are processed and sent to search for chemical entities, and then the query is rewritten to search for documents containing the retrieved chemical entities. Both the results of chemical entity searches and document searches are ranked to enable fast searches in response to queries from end-users.

For the part of graph search, the focused crawler also collects the raw data of chemical structures in the format of graphs from the Web. Then, the collected chemical structures are mined, indexed, and searched in parallel of textual entity processing. Three stages are involved to construct a graph search engine: 1) graph feature mining, 2) graph indexing, and 3) graph search. First, subgraph features are extracted and selected from the graph set. Each subgraph is converted into a canonical string, so that each graph is mapped into a linear space of subgraph features. Second, the graph index is constructed using the canonical strings of subgraphs. Third, for a subgraph query, all the indexed subgraphs of the query are extracted, and the candidate graphs containing all the extracted subgraphs of the query are retrieved from the index. For a similarity graph query, graphs that contain any extracted subgraphs of the query are retrieved from the index. Finally, for a subgraph query, subgraph isomorphism tests are performed on the candidate set to find all graphs that really contain the query graph and the filtered graph set is returned. For a similarity graph query, a ranking function (such as the maximum common edge subgraph isomorphism or a ranking function learned offline in advance) is performed to compute the similarity score between the query graph and each retrieved graph. Results are sorted and returned by the similarity scores.
1.3 Organization and Contributions

We have stated the motivation of our research work and proposed a generic framework for a domain specific search engine of Chemistry, and showed briefly how to mine, index, and search for chemical entities and structures. The rest of this dissertation is organized as follows.

In Chapter 2, we review and introduce statistical methods of text classification and segmentation. We propose an unsupervised text topic segmentation method that segments documents into subtopics, which can be utilized by hierarchical conditional random fields, an improved model of conditional random fields that is introduced in Chapter 3. We evaluate the proposed segmentation method using both real and artificial document sets. Experimental results illustrate that it outperforms previous methods.

Chapter 3 reviews methods of entity extraction from the text documents. We show how to tag chemical names and formulae, including the feature sets used in our research. We evaluate various methods for extracting textual chemical molecule information from the literature. We introduce a method for conditional random fields to tune the trade-off between precision and recall, and also propose a model of hierarchical conditional random fields for entity tagging, which can consider the long-term dependencies at different levels of documents.

Chapter 4 describes schemes to index chemical entities, query models for chemical entity searches, and corresponding ranking functions. We introduce a new concept, independent frequent subsequences, and a corresponding algorithm that can discover subterms in chemical names. We also propose an unsupervised learning method of hierarchical text segmentation and use it for chemical name index building. For chemical formula indexing, we introduce a feature selection based on frequency and discrimination of features, which can reduce the index size significantly. Then, we present various query models for chemical entity searches with corresponding ranking functions.

In Chapter 5, we study subgraph queries for graph search. First, we review related work of graph searches and list all the key issues of graph searches for chemical structures, including the issue of subgraph feature selection. Then, we propose a subgraph query algorithm and several novel feature selection criteria, Max-
Precision, Max-Irredundant-Information, and Max-Information-Min-Redundancy, based on pointwise mutual information, with the goal to achieve a high search precision for subgraph queries. Correspondingly, we present a greedy feature selection algorithm based on Max-Information-Min-Redundancy to select a near-optimal feature set. We show empirically that the proposed feature selection criteria and the corresponding algorithm can achieve much higher precision than previous methods.

In Chapter 6, we extend our research on graph search from subgraph queries to similarity graph queries. Previous similarity graph search methods are extremely slow online, due to using maximum common edge subgraph (MCEG) isomorphism tests on the fly to measure the similarity of two graphs. Differently, we propose a novel approach by ranking retrieved graphs using a weighted linear graph kernel function. This new search method can avoid online MCEG isomorphism tests, but achieve a reasonably high quality of search results. We show empirically that the proposed method of ranking learning for similarity graph queries can achieve a reasonably high normalized discounted cumulative gain in comparison with the “gold standard method”, MCEG, but with significantly fast response time for online similarity graph queries. Moreover, the weighted linear graph kernel is learnable to approximate similarity functions other than MCEG.

Finally, Chapter 7 summaries our research work and shows some potential future research directions. In Summary, our work provides appropriate solutions for various challenging issues, especially data mining issues, in a specific domain under a generic framework, including text classification and segmentation, entity tagging, frequent pattern mining, index pruning, query models, and ranking function learning. Many proposed methods can be generalized and applied on other domains.
Text Classification and Segmentation

In this chapter, we review the background and related work of text classification and segmentation. Then we propose a new text segmentation method that can be applied both on single and multiple document segmentation. Finally we show the experimental results to evaluate the proposed method.

2.1 Background and Related Work

Text classification and clustering is an important research area of text mining, which categorizes documents into groups using supervised or unsupervised methods. In the process to construct a search engine, usually text classification is used to remove undesired documents with the goal to efficiently utilize the crawling resource and improve search quality. Text clustering usually can generate cluster labels for text documents, which can be used as features for better supervised learning, such as text classification or learning to rank. Supervised text classification requires manually labeling of each document in a training set (or knowledge from user logs), which is quite expensive. Thus, unsupervised or semi-supervised methods to cluster documents are more desired in practice. Unsupervised examples are co-clustering of documents and terms, such as Latent Semantic Analysis (LSA) [5], Probabilistic Latent Semantic Analysis (PLSA) [6], and approaches based on distances and bipartite graph partitioning [7] or maximum mutual information (MI)
Latent Dirichlet Allocation (LDA) [12] is a generative probabilistic model, which also can be used to cluster documents. It models a document as a bag of words and each topic as a mixture over an underlying set of topic probabilities. In the context of text modeling, a document can be represented as a distribution of probabilities among the topics.

Text segmentation is to segment sequential text strings into semantically close parts. For example, a text stream can be segmented into topics [13], a document can be segmented into sections at the paragraph or sentence level [14], a sentence can be segmented into phrases at the term level, a sentence can be segmented into terms at the character level [15], and a term can be segmented into subterm tokens. If natural segmentation symbols like section titles, punctuation marks, or white spaces are available, then the boundaries of segmentation can be identified easily by rule-based approaches. Otherwise machine learning methods of supervised or unsupervised learning are desired. Actually, cases that require machine learning usually also have boundaries of segmentation that can segment text into units, but some adjacent units may have closer meanings in semantics than others. For example, using punctuation marks can segment a document into units of sentences, but adjacent sentences can construct sections with closer semantic meanings. Using white spaces can segment a sentence into terms, but some adjacent terms actually form a phrase.

In this chapter, we focus on text topic segmentation at the sentence level. There are lots of research works on topic segmentation that segments a document into sections on different topics, i.e., to group adjacent sentences into sections [14]. Both supervised and unsupervised learning can be applied on text segmentation. The difference between text segmentation and text classification/clustering is that text segmentation has sequential constraints. Many researchers have worked on topic detection and tracking (TDT) [16] and topic segmentation during the past decade. Topic segmentation intends to identify the boundaries in a document with the goal to capture the latent topical structure. Topic segmentation tasks usually fall into two categories [14]: text stream segmentation where topic transition is identified, and coherent document segmentation in which documents are split into subtopics. The former category has applications in automatic speech recognition, while the latter one has more applications such as partial-text query of long documents in
information retrieval, text summary, and quality measurement of multiple documents. Previous research in connection with TDT falls into the former category, targeted on topic tracking of broadcast speech data and newswire text, while the latter category has not been studied very well.

Traditional approaches perform topic segmentation on documents one at a time [14, 17, 18]. Most of them perform badly in subtle tasks like coherent document segmentation [14]. Existing approaches to topic segmentation include supervised learning [19, 20] and unsupervised learning [21, 22, 23, 18, 14, 17, 24]. Supervised learning usually has good performance, since it learns functions from labeled training sets. However, often getting large training sets with manual labels on document sentences is prohibitively expensive, so unsupervised approaches are desired. Some models consider dependence between sentences and sections, such as Hidden Markov Models [21, 22], Maximum Entropy Markov Models [19], and Conditional Random Fields [20], while many other approaches are based on lexical cohesion or similarity of sentences [23, 18, 14, 17, 24]. Some approaches also focus on cue words as hints of topic transitions [25]. Some existing methods only consider information in single documents [18, 14], while others utilize multiple documents [26, 27]. There are not many works in the latter category, even though the performance of segmentation is expected to be better with utilization of information from multiple documents. Previous research studied methods to find shared topics [26] and topic segmentation and summarization between just a pair of documents [27]. Criteria of these approaches of text classification and clustering can be utilized in the issue of topic segmentation. Some of those methods have been extended into the area of topic segmentation, such as PLSA [23] and maximum entropy [13], but to our best knowledge, using MI for topic segmentation has not been studied.

2.2 Text Classification

As mentioned in Chapter 1 and as is shown in Figure 1.1, text classification and segmentation are applied to preprocess the crawled documents. There are many classification models that can be applied on text classification. Among them, Support Vector Machines (SVMs) [28, 29, 30, 31, 32] are widely used. From previous experiences, SVMs perform well on sparse data sets with high dimensionality, for
example, text classification. That is why we apply SVMs in our system.

### 2.2.1 Support Vector Machines

Support Vector Machines are binary classification methods which find an optimal separating hyperplane \( \{ x : w \cdot x + b = 0 \} \) to maximize the margin between two classes of training samples, which is the distance between the plus-plane \( \{ x : w \cdot x + b = 1 \} \) and the minus-plane \( \{ x : w \cdot x + b = -1 \} \). Thus, for separable noiseless data, maximizing the margin equals minimizing the objective function \( ||w||^2 \) subject to \( \forall i, w \cdot y_i (x_i + b) \geq 1 \). In the noiseless case, only the so-called support vectors, i.e., vectors closest to the optimal separating hyperplane, are useful to determine the optimal separating hyperplane. Unlike classification methods where minimizing loss functions on wrongly classified samples are affected seriously by imbalanced data, the decision hyperplane in SVMs is relatively stable. However, for inseparable noisy data, SVMs minimize the objective function: \( ||w||^2 + C \sum_{i=1}^{n} \varepsilon_i \) subject to \( \forall i, w \cdot y_i (x_i + b) \geq 1 - \varepsilon_i \), and \( \varepsilon_i \geq 0 \), where \( \varepsilon_i \) is the slack variable, which measures the degree of misclassification of a sample \( x_i \), and the regularization parameter \( C \) is a constant used to regulate the trade-off between the complexity of the model and the empirical risk of the model, i.e., if the value of \( C \) is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit, but if it is too small, we may have underfitting. This noisy objective function has included a loss function that is affected by imbalanced data.

### 2.3 Text Segmentation

There are many existing text segmentation methods. However, in this section, we propose a new unsupervised method of text segmentation that segments sentences into sections on different topics, which can be used to identify different topics in a text stream or single or multiple documents [33]. This method views a sentence as a bag of words, and then mutual information is used to measure the difference of segments.

Our goal is to segment documents and align the segments across documents (Figure 2.1). Let \( T \) be the set of terms \( \{ t_1, t_2, ..., t_l \} \), which appear in the una-
belled set of documents \( D = \{d_1, d_2, ..., d_m\} \). Let \( S_d \) be the set of sentences for the document \( d \in D \), i.e. \( \{s_1, s_2, ..., s_{n_d}\} \). We have a 3D matrix of term frequency, in which the three dimensions are random variables of \( D \), \( S_d \), and \( T \). The term frequency can be used to estimate the joint probability distribution \( P(D, S_d, T) \), which is \( p(t, d, s) = \frac{T(t, d, s)}{N_D} \), where \( T(t, d, s) \) is the number of \( t \) in \( d \)’s sentence \( s \) and \( N_D \) is the total number of terms in \( D \). \( \hat{S} \) represents the set of segments \( \{\hat{s}_1, \hat{s}_2, ..., \hat{s}_p\} \) after segmentation where the number of segments \( |\hat{S}| = p \). A segment \( \hat{s}_i \) of the document \( d \) is a sequence of adjacent sentences in \( d \). For different documents \( s_i \) may discuss different subtopics. Our goal is to cluster adjacent sentences in each document into segments, and align similar segments among documents, so that for different documents \( \hat{s}_i \) is about the same subtopic. The goal is to find the optimal topic segmentation and alignment mapping

\[
Seg_d(s_i) : \{s_1, s_2, ..., s_{n_d}\} \rightarrow \{\hat{s}'_1, \hat{s}'_2, ..., \hat{s}'_p\}
\]

and \( Ali_d(\hat{s}'_i) : \{\hat{s}'_1, \hat{s}'_2, ..., \hat{s}'_p\} \rightarrow \{\hat{s}_1, \hat{s}_2, ..., \hat{s}_p\} \), for all \( d \in D \), where \( \hat{s}_i \) is the \( i^{th} \) segment with the constraint that only adjacent sentences can be mapped to the same segment, i.e. for \( d \), \( \{s_i, s_{i+1}, ..., s_j\} \rightarrow \{\hat{s}'_q\} \), where \( q \in \{1, ..., p\} \), where \( p \) is the segment number, and if \( i > j \), then for \( d \), \( \hat{s}_q \) is missing.

Term co-clustering is a technique that has been employed [9] to improve the accuracy of document clustering. We evaluate the effect of it for topic segmentation. A term \( t \) is mapped to exactly one term cluster. Term co-clustering involves simultaneously finding the optimal term clustering mapping \( Clu(t) : \{t_1, t_2, ..., t_l\} \rightarrow \{\hat{t}_1, \hat{t}_2, ..., \hat{t}_k\} \), where \( k \leq l \), \( l \) is the total number of words in all the documents, and \( k \) is the number of clusters.

We now describe a novel algorithm which can handle single-document segmentation, shared topic detection, and multi-document segmentation and alignment based on MI or WMI.

### 2.3.1 Mutual Information and Weighted Mutual Information

MI \( I(X; Y) \) is a quantity to measure the amount of information which is contained in two or more random variables [34, 9]. For the case of two random variables, we
have
\[
I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)},
\] (2.1)

Obviously, when random variables $X$ and $Y$ are independent, $I(X;Y) = 0$. Thus, intuitively, the value of MI depends on how random variables are dependent on each other. The optimal co-clustering is the mapping $Clu_x : X \rightarrow \hat{X}$ and $Clu_y : Y \rightarrow \hat{Y}$ that minimizes the loss: $I(X;Y) - I(\hat{X};\hat{Y})$, which is equal to maximizing $I(\hat{X};\hat{Y})$. This is the criterion of MI for clustering.

In the case of topic segmentation, the two random variables are the term variable $T$ and the segment variable $S$, and each sample is an occurrence of a term $T = t$ in a particular segment $S = s$. $I(T;S)$ is used to measure how dependent $T$ and $S$ are. However, $I(T;S)$ cannot be computed for documents before segmentation, since we do not have a set of $S$ due to the fact that sentences of Document $d$, $s_i \in S_d$, is not aligned with other documents. Thus, instead of minimizing the loss of MI, we can maximize MI after topic segmentation, computed as:
\[
I(\hat{T};\hat{S}) = \sum_{\hat{t}\in\hat{T}} \sum_{\hat{s}\in\hat{S}} p(\hat{t}, \hat{s}) \log \frac{p(\hat{t}, \hat{s})}{p(\hat{t})p(\hat{s})},
\] (2.2)

where $p(\hat{t}, \hat{s})$ are estimated by the term frequency tf of Term Cluster $\hat{t}$ and Segment $\hat{s}$ in the training set $D$. Note that here a segment $\hat{s}$ includes sentences about the the same topic among all documents, we have $P(D, S_d, T)$. The optimal solution is the mapping $Clu_t : T \rightarrow \hat{T}$, $Seg_d : S_d \rightarrow \hat{S}'$, and $Ali_d : \hat{S}' \rightarrow \hat{S}$, which maximizes $I(\hat{T};\hat{S})$.

In topic segmentation and alignment of multiple documents, based on the
marginal distributions $P(D|T)$ and $P(\hat{S}|T)$ for each term $t \in T$, we can define four types of terms in the training set:

- **Common stop words** are common both along the dimensions of documents and segments.
- **Document-dependent stop words** that depends on the personal writing style are common only along the dimension of segments for some documents.
- **Cue words** are the most important elements for segmentation. They are common along the dimension of documents only for the same segment, and they are not common along the dimensions of segments.
- **Noisy words** are other words which are not common along both dimensions.

Entropy based on $P(D|T)$ and $P(\hat{S}|T)$ can be used to identify different types of terms. To reinforce the contribution of cue words in the MI computation, and simultaneously reduce the effect of the other three types of words, similar to the idea of the tf-idf weight [35], we introduce term weights (or term cluster weights)

$$w_t = \left( \frac{E_D(\hat{t})}{\max_{t' \in \hat{T}}(E_D(t'))} \right)^a \left( 1 - \frac{E_S(\hat{t})}{\max_{t' \in \hat{T}}(E_S(t'))} \right)^b,$$

(2.3)

where $E_D(\hat{t}) = \sum_{d \in D} p(d|\hat{t}) \frac{1}{\log |D|/p(d|\hat{t})}$, $E_S(\hat{t}) = \sum_{\hat{s} \in S} p(\hat{s}|\hat{t}) \log |S|/p(\hat{s}|\hat{t})$, and $a > 0$ and $b > 0$ are powers to adjust term weights, usually $a = 1$ and $b = 1$ as default. Term cluster weights are used to adjust $p(\hat{t}, \hat{s})$,

$$p_w(\hat{t}, \hat{s}) = \frac{w_t p(\hat{t}, \hat{s})}{\sum_{i \in \hat{T}, \hat{s} \in \hat{S}} w_t p(\hat{t}, \hat{s})},$$

(2.4)

and

$$I_w(\hat{T}; \hat{S}) = \sum_{i \in \hat{T}} \sum_{\hat{s} \in \hat{S}} p_w(\hat{t}, \hat{s}) \log \frac{p_w(\hat{t}, \hat{s})}{p_w(\hat{t}) p_w(\hat{s})},$$

(2.5)

where $p_w(\hat{t})$ and $p_w(\hat{s})$ are marginal distributions of $p_w(\hat{t}, \hat{s})$.

However, since we do not know either the term weights or the real topic segmentation, we need to estimate them, but $w_t$ depends on $p(\hat{s}|t)$ and $\hat{S}$, while $\hat{S}$ and $p(\hat{s}|t)$ also depend on $w_t$ that is still unknown. Thus, an iterative algorithm
is required to estimate term weights \( w_i \) and optimize the objective function \( I_w \) concurrently. After a document is segmented into sentences and each sentence is segmented into words, each word is stemmed. Then the joint probability distribution \( P(D, S_d, T) \) can be estimated. Finally, this distribution can be used to compute MI in our algorithm.

### 2.3.2 Iterative Greedy Algorithm

Our goal is to maximize the objective function, \( I(\hat{T}; \hat{S}) \) or \( I_w(\hat{T}; \hat{S}) \), which can measure the dependence of term occurrences in different segments. Generally, first we do not know the estimate term weights, which depend on the optimal topic segmentation and alignment, and term clusters. Moreover, this problem is NP-hard [9], even though if we know the term weights. Thus, an iterative greedy algorithm is desired to find the best solution, even though probably only local maxima are reached. We present the iterative greedy algorithm in Algorithm 1 to find a local maximum of \( I(\hat{T}; \hat{S}) \) or \( I_w(\hat{T}; \hat{S}) \) with simultaneous term weight estimation. This algorithm is iterative and greedy for multi-document cases or single-document cases with term weight estimation and/or term co-clustering. Otherwise, since it is just a one step algorithm to solve the task of single-document segmentation [18, 14, 17], the global maximum of MI is guaranteed. We show later that term co-clustering reduces the accuracy of the results and is not necessary, and for single-document segmentation, term weights are also not required.

In Step 0, the initial term clustering \( Clu_t \) and topic segmentation and alignment \( Seg_d \) and \( Ali_d \) are important to avoid local maxima and reduce the number of iterations. First, a good guess of term weights can be made by using the distributions of term frequency along sentences for each document and averaging them to get the initial values of \( w_i \):

\[
w_t = \left( \frac{E_D(t)}{\max_{t' \in T}(E_D(t'))} \right) \left( 1 - \frac{E_S(t)}{\max_{t' \in T}(E_S(t'))} \right),
\]

where

\[
E_S(t) = \frac{1}{|D_t|} \sum_{d \in D_t} \left( 1 - \sum_{s \in S_d} p(s|t) \log |S_d| \left( \frac{1}{p(s|t)} \right) \right),
\]

where \( D_t \) is the set of documents which contain Term \( t \). Then, for the initial


**Algorithm 1** Topic segmentation and alignment based on MI or WMI

**Input:**
Joint probability distribution $P(D, S_d, T)$,
number of text segments $p \in \{2, 3, ..., \max(s_d)\}$,
number of term clusters $k \in \{2, 3, ..., l\}$ (if $k = l$, no term co-clustering required), and
weight type $w \in \{0, 1\}$, indicating to use $I$ or $I_w$, respectively.

**Output:**
Mapping $Clu$, $Seg$, $Ali$, and term weights $w_i$.

**Initialization:**
0. $i = 0$. Initialize $Clu^{(0)}$, $Seg^{(0)}$, and $Ali^{(0)}$; Initialize $w_i^{(0)}$ using Equation 2.6 if $w = 1$

**Stage 1:**
1. If $|D| = 1$, $k = l$, and $w = 0$, check all sequential segmentations of $d$ into $p$ segments and find the best one

$$Seg_d(s) = \arg\max_s I(\hat{T}; \hat{S}),$$

and return $Seg_d$; otherwise, if $w = 1$ and $k = l$, go to 3.1

**Stage 2:**
2.1 If $k < l$, for each term $t$, find the best cluster $\hat{t}$ as

$$Clu^{(i)}(t) = \arg\max_{\hat{t}} I(\hat{T}; \hat{S}^{(i)})$$

based on $Seg^{(i)}$ and $Ali^{(i)}$

2.2 For each $d$, check all sequential segmentations of $d$ into $p$ segments with mapping $s \to \hat{s}' \to \hat{s}$, and find the best one

$$Ali^{(i+1)}_d(Seg^{(i+1)}_d(s)) = \arg\max_{\hat{t}} I(\hat{T}^{(i+1)}; \hat{S})$$

based on $Clu^{(i+1)}(t)$ if $k < l$ or $Clu^{(0)}(t)$ if $k = l$

2.3 $i++$. If $Clu$, $Seg$, or $Ali$ changed, go to 2.1; otherwise, if $w = 0$, return $Clu^{(i)}$, $Seg^{(i)}$, and $Ali^{(i)}$; else $j = 0$, go to 3.1

**Stage 3:**
3.1 Update $w_i^{(i+j+1)}$ based on $Seg^{(i+j)}$, $Ali^{(i+j)}$, and $Clu^{(i)}$ using Equation 2.3

3.2 For each $d$, check all sequential segmentations of $d$ into $p$ segments with mapping $s \to \hat{s}' \to \hat{s}$, and find the best one

$$Ali^{(i+j+1)}_d(Seg^{(i+j+1)}_d(s)) = \arg\max_{\hat{t}} I_w(\hat{T}^{(i)}; \hat{S})$$

based on $Clu^{(i)}$ and $w_i^{(i+j+1)}$

3.3 $j++$. If $I_w(\hat{T}; \hat{S})$ changes, go to step 6; otherwise, stop and return $Clu^{(i)}$, $Seg^{(i+j)}$, $Ali^{(i+j)}$, and $w_i^{(i+j)}$. 
segmentation $Seg^{(0)}$, we can simply segment documents equally by sentences. Or we can find the optimal segmentation just for each document $d$ which maximizes the WMI, $Seg_d^{(0)} = \arg\max_I w(T; \hat{S})$, where $w = w_t^{(0)}$. For the initial alignment $Ali^{(0)}$, we can first assume that the order of segments for each $d$ is the same. For the initial term clustering $Clu^{(0)}$, first cluster labels can be set randomly, and after the first time of Step 3, a good initial term clustering is obtained.

After initialization, there are three stages for different cases. Totally there are eight cases, $|D| = 1$ or $|D| > 1$, $k = l$ or $k < l$, $w = 0$ or $w = 1$. Single document segmentation without term clustering and term weight estimation ($|D| = 1, k = l, w = 0$) only requires Stage 1 (Step 1). If term clustering is required ($k < l$), Stage 2 (Step 2.1, 2.2, and 2.3) is executed iteratively. If term weight estimation is required ($w = 1$), Stage 3 (Step 3.1, 3.2, and 3.3) is executed iteratively. If both are required ($k < l, w = 1$), Stage 2 and 3 run one after the other. For multi-document segmentation without term clustering and term weight estimation ($|D| > 1, k = l, w = 0$), only iteration of Step 2.2 and 2.3 are required.

At Stage 1, the global maximum value can be found based on $I(\hat{T}; \hat{S})$ using dynamic programming (shown below). Simultaneously finding a good term clustering and estimated term weights is impossible, since when moving a term to a new term cluster to maximize $I_w(\hat{T}; \hat{S})$, we do not know that the weight of this term should be the one of the new cluster or the old cluster. Thus, we first do term clustering at Stage 2, and then estimate term weights at Stage 3.

At Stage 2, Step 2.1 is to find the best term clustering and Step 2.2 is to find the best segmentation. This cycle is repeated to find a local maximum based on MI $I$ until it converges. The two steps are: (1) based on current term clustering $Clu^i$, for each document $d$, the algorithm segments all the sentences $S_d$ into $p$ segments sequentially (some segments may be empty), and put them into the $p$ segments $\hat{S}$ of the whole training set $D$ (all possible cases of different segmentation $Seg_d$ and alignment $Ali_d$ are checked) to find the optimal case, and (2) based on the current segmentation and alignment, for each term $t$, the algorithm finds the best term cluster of $t$ based on the current segmentation $Seg_d$ and alignment $Ali_d$. After finding a good term clustering, term weights are estimated if $w = 1$.

At Stage 3, similar to Stage 2, Step 3.1 is term weight re-estimation and Step 3.2 is to find a better segmentation. They are repeated to find a local maximum.
based on WMI $I_w$ until it converges. However, if the term clustering in Stage 2 is not accurate, then the term weight estimation at Stage 3 may have a bad result. Finally, at Step 3.3, this algorithm converges and return the output. This algorithm can handle both single-document and multi-document segmentation. It also can detect shared topics among documents by checking the proportion of overlapped sentences on the same topics, as described in Sec 5.2.

In many previous works on segmentation, dynamic programming is a technique used to maximize the objective function. Similarly, at Step 1, 2.2, and 3.2 of our algorithm, we can use dynamic programming. For Stage 1, using dynamic programming can still find the global optimum, but for Stage 2 and Stage 3, we can only find the optimum for each step of topic segmentation and alignment of a document. Here we only show the dynamic programming for Step 3.2 using WMI (Step 1 and 2.2 are similar but they can use either $I$ or $I_w$). There are two cases that are not shown in Algorithm 1: (a) single-document segmentation or multi-document segmentation with the same sequential order of segments, where alignment is not required, and (b) multi-document segmentation with different sequential orders of segments, where alignment is necessary. The alignment mapping function of the former case is simply just $Ali_d(s'_i) = s_i$, while for the latter one’s alignment mapping function $Ali_d(s'_i) = s_j$, $i$ and $j$ may be different. The computational steps for the two cases are listed below:

**Case 1 (no alignment):**

For each document $d$:

1. Compute $p_w(\hat{t})$, partial $p_w(\hat{t}, \hat{s})$ and partial $p_w(\hat{s})$ without counting sentences from $d$. Then put sentences from $i$ to $j$ into Part $k$, and compute partial WMI

$$PI_w(\hat{T}; \hat{s}_k(s_i, s_{i+1}, \ldots, s_j)) \sum_{\hat{t} \in \hat{T}} p_w(\hat{t}, \hat{s}_k) \log \frac{p_w(\hat{t}, \hat{s}_k)}{p_w(\hat{t}) p_w(\hat{s}_k)},$$

where $Ali_d(s_i, s_{i+1}, \ldots, s_j) = k$, $k \in \{1, 2, \ldots, p\}$, $1 \leq i \leq j \leq n_d$, and $Seg_d(s_q) = \hat{s}_k$ for all $i \leq q \leq j$.

2. Let $M(s_m, 1) = PI_w(\hat{T}; \hat{s}_1(s_1, s_2, \ldots, s_m))$. Then

$$M(s_m, L) = \max_i [M(s_{i-1}, L - 1) + PI_w(\hat{T}; \hat{s}_L(s_i, \ldots, s_m))],$$
where $0 \leq m \leq n_d$, $1 < L < p$, $1 \leq i \leq m + 1$, and when $i > m$, no sentences are put into $\hat{s}_k$ when compute $PI_w$ (note $PI_w(\hat{T}; \hat{s}(s_i, ..., s_m)) = 0$ for single-document segmentation).

(3) Finally $M(s_{nd}, p) = \max_i [M(s_{i-1}, p - 1) + PI_w(\hat{T}; \hat{s}_p(s_i, ..., s_{nd}))]$, where $1 \leq i \leq n_d + 1$. The optimal $I_w$ is found and the corresponding segmentation is the best.

**Case 2** (alignment required):

For each document $d$:

(1) Compute $p_w(\hat{t})$, partial $p_w(\hat{t}, \hat{s})$, partial $p_w(\hat{s})$, and $PI_w(\hat{T}; \hat{s}_k(s_i, s_{i+1}, ..., s_j))$ similarly as Case 1.

(2) Let $M(s_m, 1, k) = PI_w(\hat{T}; \hat{s}_k(s_1, s_2, ..., s_m))$, where $k \in \{1, 2, ..., p\}$. Then $M(s_m, L, k_L) = \max_{i,j} [M(s_{i-1}, L - 1, k_{L/j}) + PI_w(\hat{T}; \hat{s}_{Ali_d}(s_L') = j(s_i, s_{i+1}, ..., s_m))]$, where $0 \leq m \leq n_d$, $1 < L < p$, $1 \leq i \leq m + 1$, $k_L \in \text{Set}(p, L)$, which is the set of all $\frac{p!}{L!(p-L)!}$ combinations of $L$ segments chosen from all $p$ segments, $j \in k_L$, the set of $L$ segments chosen from all $p$ segments, and $k_{L/j}$ is the combination of $L - 1$ segments in $k_L$ except Segment $j$.

(3) Finally, $M(s_{nd}, p, k_p) = \max_{i,j} [M(s_{i-1}, p - 1, k_{p/j}) + PI_w(\hat{T}; \hat{s}_{Ali_d}(s_L') = j(s_i, s_{i+1}, ..., s_{nd}))]$, where $k_p$ is just the combination of all $p$ segments and $1 \leq i \leq n_d + 1$, which is the optimal $I_w$ and the corresponding segmentation is the best.

The steps of Case 1 and 2 are similar, except in Case 2, alignment is considered in addition to segmentation. First, basic items of probability for computing $I_w$ are computed excluding Doc $d$, and then partial WMI by putting every possible sequential segment (including empty segment) of $d$ into every segment of the set. Second, the optimal sum of $PI_w$ for $L$ segments and the leftmost $m$ sentences, $M(s_m, L)$, is found. Finally, the maximal WMI is found among different sums of $M(s_m, p - 1)$ and $PI_w$ for Segment $p$.

### 2.4 Experimental Evaluation

In this section, single-document segmentation, shared topic detection, and multi-document segmentation are tested using our proposed topic segmentation method. Different hyper parameters of our method are studied. For convenience, we refer
to the method using $I$ as $MI_k$ if $w = 0$, and $I_w$ as $WMI_k$ if $w = 2$ or as $WMI'_k$ if $w = 1$, where $k$ is the number of term clusters, and if $k = l$, where $l$ is the total number of terms, then no term clustering is required, i.e. $MI_l$ and $WMI_l$.

### 2.4.1 Single-document Segmentation

The first data set we test is a synthetic one used in previous research [18, 14, 17] and many other papers. It has 700 samples. Each is a concatenation of ten segments. Each segment is the first $n$ sentence selected randomly from the Brown corpus, which is supposed to have a different topic from each other. Currently, the best results on this data set is achieved by Ji et.al. [14]. To compare the performance of our methods, the criterion used widely in previous research is applied, instead of the unbiased criterion introduced in [36]. It chooses a pair of words randomly. If they are in different segments (different) for the real segmentation (real), but predicted (pred) as in the same segment, it is a miss. If they are in the same segment (same), but predicted as in different segments, it is a false alarm. Thus, the error rate is computed using the following equation:

$$p(\text{err}|\text{real}, \text{pred}) = p(\text{miss}|\text{real}, \text{pred}, \text{diff})p(\text{diff}|\text{real})$$

$$+ p(\text{false alarm}|\text{real}, \text{pred}, \text{same})p(\text{same}|\text{real}).$$

We test the case when the number of segments is known. Table 2.1 shows the results of our methods with different hyper parameter values and three previous approaches, C99[17], U00[18], and ADDP03[14], on this data set when the segment number is known. In $WMI$ for single-document segmentation, the term weights are computed as follows,

$$w_i = 1 - E_\hat{s}(\hat{i})/\max_{\hat{p} \in T}(E_\hat{s}(\hat{p})).$$  

For this case, our methods $MI_l$ and $WMI_l$ both outperform all the previous approaches. We compared the proposed methods with ADDP03 using one-sample one-sided t-test and p-values are shown in Table 2.2. From the p-values, we can see that mostly the differences are very significant. We also compare the error
Table 2.1. Average Error Rates of Single-document Segmentation Given Segment Numbers Known

<table>
<thead>
<tr>
<th>Range of $n$</th>
<th>3-11</th>
<th>3-5</th>
<th>6-8</th>
<th>9-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample size</td>
<td>400</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>C99</td>
<td>12%</td>
<td>11%</td>
<td>10%</td>
<td>9%</td>
</tr>
<tr>
<td>U00</td>
<td>10%</td>
<td>9%</td>
<td>7%</td>
<td>5%</td>
</tr>
<tr>
<td>ADDP03</td>
<td>6.0%</td>
<td>6.8%</td>
<td>5.2%</td>
<td>4.3%</td>
</tr>
<tr>
<td>$MI_{l}$</td>
<td>4.68%</td>
<td>5.57%</td>
<td>2.59%</td>
<td>1.59%</td>
</tr>
<tr>
<td>$WMI_{l}$</td>
<td>4.94%</td>
<td>6.33%</td>
<td>2.76%</td>
<td>1.62%</td>
</tr>
<tr>
<td>$MI_{100}$</td>
<td>9.62%</td>
<td>12.92%</td>
<td>8.66%</td>
<td>6.67%</td>
</tr>
</tbody>
</table>

Table 2.2. Single-document Segmentation: P-values of T-test on Error Rates

<table>
<thead>
<tr>
<th>Range of $n$</th>
<th>3-11</th>
<th>3-5</th>
<th>6-8</th>
<th>9-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDP03, $MI_{l}$</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>ADDP03, $WMI_{l}$</td>
<td>0.000</td>
<td>0.0099</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>$MI_{l}$, $WMI_{l}$</td>
<td>0.061</td>
<td>0.132</td>
<td>0.526</td>
<td>0.898</td>
</tr>
</tbody>
</table>

rates between our two methods using two-sample two-sided t-test to check the hypothesis that they are equal. We cannot reject the hypothesis that they are equal, so the difference is not significant, even though all the error rates for $MI_{l}$ are smaller than $WMI_{l}$. However, we can conclude that term weights contribute little in single-document segmentation. The results also show that $MI$ using term co-clustering ($k = 100$) decreases the performance. We test different number of term clusters, and found that the performance becomes better when the cluster number increases to reach $l$. $WMI_{k<l}$ has similar results that we do not show in the table.

As mentioned before, using MI may be inconsistent on optimal boundaries given different numbers of segments. This situation occurs especially when the similarities among segments are quite different, i.e. some transitions are very obvious, while others are not. This is because usually a document is a hierarchical structure instead of only a sequential structure. When the segments are not at the same level, this situation may occur. Thus, a hierarchical topic segmentation approach is desired, and the structure highly depends on the number of segments for each internal node and the stop criteria of splitting. For this data set of single-document segmentation, since it is just a synthetic set, which is just a concatenation of several segments about different topics, it is reasonable that approaches simply based
on term frequency have a good performance. Usually for the tasks of segmenting coherent documents for subtopics, the effectiveness decreases much.

2.4.2 Shared Topic Detection

The second data set contains 80 news articles from Google News. There are eight topics and each has 10 articles. We randomly split the set into subsets with different document numbers and each subset has all eight topics. We compare the proposed approach $MI_t$ and $WMI_t$ with LDA [12]. LDA treats a document in the data set as a bag of words, finds its distribution on topics, and its major topic. $MI_t$ and $WMI_t$ views each sentence as a bag of words and tag it with a topic label. Then for each pair of documents, LDA determines if they are on the same topic, while $MI_t$ and $WMI_t$ check whether the proportion overlapped sentences on the same topic is larger than the adjustable threshold $\theta$. That is, in $MI_t$ and $WMI_t$, for a pair of documents $d$, $d'$, if $\sum_{s \in S_d, s' \in S_{d'}} I_{\text{topic}_s = \text{topic}_{s'}} / \min(\|S_d\|, \|S'_{d'}\|) > \theta$, where $S_d$ is the set of sentences of $d$, and $|S_d|$ is the number of sentences of $d$, then $d$ and $d'$ have the shared topic.

For a pair of documents selected randomly, the error rate is computed using the following equation:

$$p(\text{err}|\text{real},\text{pred}) = p(\text{miss}|\text{real},\text{pred},\text{same})p(\text{same}|\text{real})$$

$$+ p(\text{false alarm}|\text{real},\text{pred},\text{diff})p(\text{diff}|\text{real})$$

(2.9)

where a miss means if they have the same topic (same) for the real case (real), but predicted (pred) as on the same topic. If they are on different topics (diff), but predicted as on the same topic, it is a false alarm.

The results are shown in Table 2.3. If most documents have different topics, in

### Table 2.3. Shared Topic Detection: Average Error Rates for Different Numbers of Documents in Each Subset

<table>
<thead>
<tr>
<th>#Doc</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>8.89%</td>
<td>16.33%</td>
<td>1.35%</td>
<td>0.60%</td>
</tr>
<tr>
<td>$MI_t, \theta = 0.6$</td>
<td>4.17%</td>
<td>1.71%</td>
<td>1.47%</td>
<td>0.0%</td>
</tr>
<tr>
<td>$WMI_t, \theta = 0.8$</td>
<td>18.6%</td>
<td>3.16%</td>
<td>1.92%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
Table 2.4. Average Error Rates of Multi-document Segmentation Given Segment Numbers Known

<table>
<thead>
<tr>
<th>#Doc</th>
<th>$MI_l$</th>
<th>$WMI_l$</th>
<th>$k$</th>
<th>$MI_k$</th>
<th>$WMI_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>3.14%</td>
<td>2.78%</td>
<td>300</td>
<td>4.68%</td>
<td>6.58%</td>
</tr>
<tr>
<td>51</td>
<td>4.17%</td>
<td>3.63%</td>
<td>300</td>
<td>17.83%</td>
<td>22.84%</td>
</tr>
<tr>
<td>34</td>
<td>5.06%</td>
<td>4.12%</td>
<td>300</td>
<td>18.75%</td>
<td>20.95%</td>
</tr>
<tr>
<td>20</td>
<td>7.08%</td>
<td>5.42%</td>
<td>250</td>
<td>20.40%</td>
<td>21.83%</td>
</tr>
<tr>
<td>10</td>
<td>10.38%</td>
<td>7.89%</td>
<td>250</td>
<td>21.42%</td>
<td>21.91%</td>
</tr>
<tr>
<td>5</td>
<td>15.77%</td>
<td>11.64%</td>
<td>250</td>
<td>21.89%</td>
<td>22.59%</td>
</tr>
<tr>
<td>2</td>
<td>25.90%</td>
<td>23.18%</td>
<td>50</td>
<td>25.44%</td>
<td>25.49%</td>
</tr>
<tr>
<td>1</td>
<td>23.90%</td>
<td>24.82%</td>
<td>25</td>
<td>25.75%</td>
<td>26.15%</td>
</tr>
</tbody>
</table>

Table 2.5. Multi-document Segmentation: P-values of T-test on Error Rates for $MI_l$ and $WMI_l$

<table>
<thead>
<tr>
<th>#Doc</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>0.19</td>
</tr>
<tr>
<td>34</td>
<td>0.101</td>
</tr>
<tr>
<td>20</td>
<td>0.025</td>
</tr>
<tr>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.002</td>
</tr>
</tbody>
</table>

$WMI_l$, the estimation of term weights in Equation 2.3 is not correct. Thus, $WMI_l$ is not expected to have a better performance than $MI_l$, when most documents have different topics. When there are fewer documents in a subset with the same number of topics, more documents have different topics, so $WMI_l$ is more worse than $MI_l$. We can see that for most cases $MI_l$ has a better (or at least similar) performance than LDA. After shared topic detection, multi-document segmentation of documents with the shared topics is able to be executed.

2.4.3 Multi-document Segmentation

For multi-document segmentation and alignment, our goal is to identify these segments about the same topic among multiple similar documents with shared topics. Using $I_w$ is expected to perform better than $I$, since without term weights the result is affected seriously by document-dependent stop words and noisy words which depends on the personal writing style. It is more likely to treat the same segments of different documents as different segments under the effect of document-dependent stop words and noisy words. Term weights can reduce the effect of document-dependent stop words and noisy words by giving cue terms more weights.

The data set for multi-document segmentation and alignment has 102 samples and 2264 sentences totally. Each is the introduction part of a lab report selected
from the course of Biol 240W, Pennsylvania State University. Each sample has two segments, introduction of plant hormones and the content in the lab. The length range of samples is from two to 56 sentences. Some samples only have one part and some have a reverse order the these two segments. It is not hard to identify the boundary between two segments for humans. We label each sentence manually for evaluation. The criterion of evaluation is just using the proportion of the number of sentences with wrong predicted segment labels in the total number of sentences in the whole training set as the error rate:

\[
p(error|predicted, real) = \frac{\sum_{d \in D} \sum_{s \in S_d} 1(predicted_s \neq real_s)}{\sum_{d \in D} n_d}.
\]

In order to show the benefits of multi-document segmentation and alignment, we compare the proposed method with different parameters on different partitions of the same training set. Except the cases that the number of documents is 102 and one (they are special cases of using the whole set and the pure single-document segmentation), we randomly divide the training set into \(m\) partitions, and each has 51, 34, 20, 10, 5, and 2 document samples. Then we apply the proposed methods on each partition and calculate the error rate of the whole training set. Each case is repeated for 10 times for computing the average error rates. For different partitions of the training set, different \(k\) values are used, since the number of terms increases when the document number in each partition increases.

From the experiment results in Table 2.4, we can see the following observations: (1) When the number of documents increases, all methods have better performances. Only from one to two documents, \(MI_t\) has decrease a little. We
can observe this from Figure 2.2 at the point of document number = 2. Most curves even have the worst results at this point. There are two reasons. First, because samples vote for the best multi-document segmentation and alignment, but if only two documents are compared with each other, the one with missing segments or a totally different sequence affects the correct segmentation and alignment of the other. Second, as noted at the beginning of this section, if two documents have more document-dependent stop words or noisy words than cue words, then the algorithm may view them as two different segments and the other segment is missing. Generally, we can only expect a better performance when the number of documents is larger than the number of segments. (2) Except single-document segmentation, \( WMI_l \) is always better than \( MI_l \), and when the number of documents is reaching one or increases to a very large number, their performances become closer. Table 2.5 shows p-values of two-sample one-sided t-test between \( MI_l \) and \( WMI_l \). We also can see this trend from p-values. When document number = 5, we reach the smallest p-value and the largest difference between error rates of \( MI_l \) and \( WMI_l \). For single-document segmentation, \( WMI_l \) is even a little bit worse than \( MI_l \), which is similar to the results of the single-document segmentation on the first data set. The reason is that for single-document segmentation, we cannot estimate term weights accurately, since multiple documents are unavailable. (3) Using term clustering usually gets worse results than \( MI_l \) and \( WMI_l \). (4) Using term clustering in \( WMI_k \) is even worse than in \( MI_k \), since in \( WMI_k \) term clusters are found first using \( I \) before using \( I_w \). If the term clusters are not correct, then the term weights are estimated worse, which may mislead the algorithm to reach even worse results. From the results we also found that in multi-document segmentation and alignment, most documents with missing segments and a reverse order are identified correctly.

Table 2.6 illustrates the experiment results for the case of 20 partitions (each has five document samples) of the training set and topic segmentation and alignment using \( MI_k \) with different numbers of term clusters \( k \). Notice that when the number of term clusters increases, the error rate becomes smaller. Without term clustering, we have the best result. We do not show results for \( WMI_k \) with term clustering, but the results are similar.

We also test \( WMI_l \) with different hyper parameters of \( a \) and \( b \) to adjust term
weights. The results are presented in Figure 2.2. It was shown that the default case $WMI_l : a = 1, b = 1$ gave the best results for different partitions of the training set. We can see the trend that when the document number is very small or large, the difference between $MI_l : a = 0, b = 0$ and $WMI_l : a = 1, b = 1$ becomes quite small. When the document number is not large (about from 2 to 10), all the cases using term weights have better performances than $MI_l : a = 0, b = 0$ without term weights, but when the document number becomes larger, the cases $WMI_l : a = 1, b = 0$ and $WMI_l : a = 2, b = 1$ become worse than $MI_l : a = 0, b = 0$. When the document number becomes very large, they are even worse than cases with small document numbers. This means that a proper way to estimate term weights for the criterion of WMI is very important. Figure 2.3 shows the term weights learned from the whole training set. Four types of words are categorized roughly even though the transition among them are subtle. Figure 2.4 illustrates the change in (weighted) mutual information for $MI_l$ and $WMI_l$. As expected, mutual information for $MI_l$ increases monotonically with the number of steps, while $WMI_l$ does not. Finally, $MI_l$ and $WMI_l$ are scalable, with computational complexity shown in Figure 2.5.

One advantage for our approach based on MI is that removing stop words is
not required. Another important advantage is that there are no necessary hyper parameters to adjust. In single-document segmentation, the performance based on MI is even better for that based on WMI, so no extra hyper parameter is required. In multi-document segmentation, we show in the experiment, $a = 1$ and $b = 1$ is the best. Our method gives more weights to cue terms. However, usually cue terms or sentences appear at the beginning of a segment, while the end of the segment may be much noisy. One possible solution is giving more weights to terms at the beginning of each segment. Moreover, when the length of segments are quite different, long segments have much higher term frequencies, so they may dominate the segmentation boundaries. Normalization of term frequencies versus the segment length may be useful.
Chapter 3

Textual Entity Extraction

In the last chapter, we have discussed text document preprocessing. In this chapter, we discuss methods of entity tagging from sequential data, like text, i.e., how to extract textual entities from text. Those methods are called shallow parsers that only detect useful entities from sequences. First, we briefly review related work, and then describe a probabilistic model considering the label dependency among terms, Conditional Random Fields (CRFs). Second, we review various methods to tune the trade-off between precision and recall for traditional classifiers in the case of imbalanced data sets. Then we introduce a similar trade-off tuning method to CRFs. Third, we propose a hierarchical model of CRFs (HCRFs) to consider long-term dependency of documents. Finally, we show the experimental results to evaluate various methods, including SVMs, CRFs, and HCRFs.

3.1 Background and Related Work

Sequence labeling is a task that assigns labels to sequences of observations, e.g., labeling Part of Speech (POS) tags and entity extraction. Labeling POS tags represents a sentence with a full tree structure and labels each term with a POS tag, while shallow parsers [37] are used to extract entities. Methods used for labeling sequences are different from those that are used for traditional classification, which only considers independent samples. Hidden Markov Models (HMM) [38] are one of the common methods used to label or segment sequences. HMM has a conditional independence assumption where given the hidden state, observations are
independent [20]. Thus, HMM cannot present the observation interaction of adjacent tokens. Another category of entity extraction methods is based on *Maximum Entropy* (ME) [3], which introduces an exponential probabilistic model based on binary features extracted from sequences and estimate parameters using maximum likelihood. *MEMM* [19] is another exponential probabilistic model that takes the observation features as input, and outputs a distribution over possible next states, but it suffers from the label-bias problem [20]. Different from directed graph-based models like HMM and MEMM, CRF [20] uses an undirected graphic model, which can relax the conditional independence assumption of HMM and avoid the label-bias problem of MEMM. It follows the maximum entropy principle [39] as ME, using exponential probabilistic models and relaxing the independence assumption to involve multiple interaction and long-range dependencies. Models based on linear-chain CRF have been applied for labeling sequences in many applications, such as named-entity recognition [40], detecting biological entities, like proteins [41] or genes [4], etc.

Chemical entity tagging differs from them in the tokenizing process and the feature set used due to different domain knowledge. Banville has provided a high-level overview for mining chemical structure information from the literature [42]. In the community of Chemoinformatics, methods based on lexicons and Bayesian classification have been used to recognize chemical nomenclature [43], but not to tag entities from the text. Previous works on chemical entity tagging use both machine learning approaches [44] and rule-based approaches. We show empirically that our methods outperform existing methods. Chemical formula tagging can also be viewed as a classification problem where classification approaches such as SVMs [31, 32] are applicable. However, usually entity tagging is an asymmetric binary classification problem on imbalanced data, where there are much more false samples than true samples, but precision and recall of the true class are more important than the overall accuracy. In this case, the decision boundary may be dominated by false samples. Several methods such as cost-sensitive classification and decision-threshold tuning are studied for imbalanced data [45]. We have observed that CRFs suffer from this problem too, since in previous work based on

---

1GATE: http://gate.ac.uk/ and Oscar3: http://wwmm.ch.cam.ac.uk/wikis/wwmm
2SVM light: http://svmlight.joachims.org/
CRFs [4, 37], usually recall is lower than precision. To the best of our knowledge, no methods to tune the decision boundary for CRFs exist.

Entity extraction can also be viewed as a classification problem where approaches such as SVMs [31, 32] are applicable, since information about dependence in the text context of terms can be represented as overlapping features between adjacent terms. However, usually entity extraction is an asymmetric binary classification problem on imbalanced data, where there are many more false samples than true samples, but precision and recall of the true class is more important than the overall accuracy. In this case, the decision boundary may be dominated by false samples. Several methods such as cost-sensitive classification and decision-threshold tuning are studied for imbalanced data [45]. We have observed that CRFs suffer from this problem too, since in previous work based on CRFs [4, 37], usually recall is lower than precision. To the best of our knowledge, no methods to tune the trade-off between precision and recall exist.

### 3.2 Conditional Random Fields

Suppose we have a training set $S$ of labeled graphs, where each graph is an independently and identically distributed sample, but each sample has an internal dependent structure. For example, in a document, adjacent terms have strong dependence, where a document is a graph, each term is a node, and the edges on the graph represent the dependency of terms. Could we find a method for each sample to represent the conditional probability $p(y|x, \lambda)$, where $x$ is the sequence of observations, $y$ is the sequence of labels, and $\lambda$ is the parameter vector of the model, and consider the dependent structure in each sample? Maximum Likelihood can be used to learn the parameter $\lambda$ and find the best $y$.

In this scenario we just describe, CRFs model each graph sample in the data set as an undirected graph $G = (V, E)$ [20]. Each vertex $v \in V$ has a label $y_v$ and an observation $x_v$. Each edge $e = \{v, v'\} \in E$ represents the mutual dependence of a pair of labels $y_v, y_{v'}$. For each sample, the conditional probability $p(y|x, \lambda)$, where $x$ is the observation vector of all vertices in $G$, $y$ is the label vector, and $\lambda$ is the parameter vector of the model, represents the probability of the sample of $y$ given $x$. 
To model the conditional probability \( p(y|\mathbf{x}, \lambda) \), we need to find a probability model that can consider not only the probability of each vertex, but also the joint probability of each pair of vertices. Then if the labels or observations of a pair of vertices have some changes, the probability of this model changes too. An exponential probabilistic model for each graph based on feature functions is applied in CRFs to model the conditional probability,

\[
p(y|\mathbf{x}, \lambda) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_j \lambda_j F_j(y, \mathbf{x})\right),
\]

where \( F_j(y, \mathbf{x}) \) is a feature function which extracts a real-valued (or binary) feature from the label vector \( y \) and the observation vector \( \mathbf{x} \), and \( Z(\mathbf{x}) \) is a normalization factor for each sample of the observation vector \( \mathbf{x} \) in \( S \).

Even though the structure of \( G \) may be arbitrary, for sequential data like text documents, usually chain-structured models of CRFs are applied, where only the labels \( y_i \) and \( y_{i+1} \) of neighbors in a sequence are dependent. Moreover, usually only binary features are considered for sequential data. There are two types of features, state features \( F_j = S_j(y, \mathbf{x}) = \sum_{i=1}^{y_i} s_j(y_i, \mathbf{x}, i) \) to consider only the label of a single vertex and transition features \( F_j = T_j(y, \mathbf{x}) = \sum_{i=1}^{y_i} t_j(y_{i-1}, y_i, \mathbf{x}, i) \) to consider mutual dependence of the vertex labels for each edge \( e \) in \( G \). State features include two kinds of features: single-unit features from a single vertex and overlapping features from adjacent vertices. Transition features actually are combinations of vertex labels and state features. Each function has a weight \( \lambda_j \), which specifies whether the corresponding feature is favored or not. \( \lambda_j \) for the feature \( j \) should be highly positive if this feature tends to be on for the training data, and highly negative if it tends to be off.

Once we have \( p(y|\mathbf{x}, \lambda) \), the log-likelihood for the whole train set \( S \) is given by

\[
L(\lambda) = \sum_{k=1}^{\vert S \vert} \log(p(y^{(k)}|\mathbf{x}^{(k)}, \lambda))
\]

The goal is to maximize this log-likelihood, which has been proved to be a smooth and concave function, and to estimate parameters \( \lambda \). To avoid the overfitting problem, \textit{regularization} may be used; that is, a penalty \( -\sum_{j=1}^{\vert \lambda \vert} \frac{\lambda_j^2}{2\sigma^2} \) is
added to the log-likelihood function (3), where \( \sigma \) is a parameter which determines how much to penalize \( \lambda \).

Differentiating the log-likelihood function (3) with respect to \( \lambda_j \), setting the derivative to zero and solving for \( \lambda \) does not have a closed form solution. Numerical optimization algorithms can be applied to solve this problem [20, 46, 37].

Differentiating the log-likelihood function (3) with respect to \( \lambda_j \) gives

\[
\frac{\partial L(\lambda)}{\partial \lambda_j} = \sum_{k=1}^{[S]} [F_j(y^{(k)}, x^{(k)}) - E_{p\lambda(y|x^{(k)})} F_j(Y, x^{(k)})].
\] (3.3)

However, setting the derivative to zero and solving for \( \lambda \) does not have a closed form solution. Numerical optimization algorithms can be applied to solve this problem [20, 46, 37].

Iterative scaling algorithms [20, 46] are simple but quite slow. Other methods such as Steepest Ascent, Conjugate Gradient, Newton’s methods, and quasi-Newton methods are studied to solve the numerical optimization problem in CRFs [37]. At each step of the numerical optimization, usually one needs to compute \( E_{p\lambda(y|x^{(k)})} F_j(Y, x^{(k)}) \) efficiently given current parameters \( \lambda \). A dynamic programming method, like the forward-backward algorithm for HMM, is used to compute this expectation. Let \( Y \) be the alphabet from which labels \( y \) and \( y' \) are drawn. For a sequence, a set of \( n+1 \) matrices are defined, where each \( M_i(x) \) is a \( |Y \times Y| \) matrix with each element

\[
M_i(y, y'|x) = \exp(\sum_j \lambda_j f_j(y, y', x, i)),
\]

where \( f_j \) is a feature function. Then for chain-structured CRFs,

\[
E_{p\lambda(y|x^{(k)})} F_j(Y, x^{(k)}) = \sum_y p\lambda(y|x^{(k)}) F_j(y, x^{(k)})
\]

\[
= \sum_{i=1}^{[x]} \sum_{y, y'} \alpha_{i-1}(y|x) M_i(x) \beta^T_i(y|x) f_j(y', y, x^{(k)}) \frac{Z(x)}{Z(x)}.
\]
where $\alpha_i$ and $\beta_i$ are the forward and backward vectors,

$$
\alpha_0(y|x) = \begin{cases} 
1 & y = \text{start} \\
0 & \text{otherwise} 
\end{cases}
$$

$$
\beta_{n+1}^T(y|x) = \begin{cases} 
1 & y = \text{end} \\
0 & \text{otherwise} 
\end{cases}
$$

$$
\alpha_i = \alpha_{i-1} M_i, \quad \beta_i^T = M_{i+1} \beta_{i+1}^T,
$$

and $Z(x) = [\prod_{i=1}^{n+1} M_i(x)]_{\text{start,end}}$.

Finally all features combine with labels. However, there are too many features and most occur rarely. We apply an approach to feature induction for CRFs proposed in [47] to score candidate features using their log-likelihood gain:

$$
\Delta L_G(f) = L(S)_{F\cup\{f\}} - L(S)_{F},
$$

where $F$ is the current feature set, $L(S)_F$ is the log-likelihood of the training set using $F$, and $L(S)_{F\cup\{f\}}$ is the log-likelihood of the training set adding the feature $f$. Thus, more useful features are selected.

In practice, all features extracted from the observations $x$ combine with labels $y$ to generate a full feature set. However, there are too many features and most occur rarely. We apply an approach to feature induction for CRFs proposed in [47] to score candidate features using their log-likelihood gain:

$$
\Delta L_G(f) = L(S)_{F\cup\{f\}} - L(S)_F,
$$

where $F$ is the current feature set, $L(S)_F$ is the log-likelihood of the training set using $F$, and $L(S)_{F\cup\{f\}}$ is the log-likelihood of the training set adding the feature $f$. Thus, more useful features are selected.

### 3.3 Imbalanced Data Classification and Tagging

In real world problems, classification tasks of imbalanced data (i.e., data with a skewed class distribution) are very common. Usually there are much less true samples (positive samples). From the results of previous research, we observe that usually for imbalanced data, recall is lower than precision for the true class. However, usually extraction of true samples is more important than the overall accuracy, and sometimes recall is more important than precision, especially in information retrieval.

Usually parameter tuning can improve recall with some loss of precision. Those
classification approaches mainly fall into two categories, a) tuning in the training process and b) tuning in the testing process. Cross validation is used to estimate the best parameter [45]. The former approach over-samples the minority class, or under-samples the majority class, or gives different weights for two classes, or gives different penalties (costs of risk) to wrong classifications, every time during training. For example, if $\text{Cost}(\text{predicted} = \text{true}|\text{real} = \text{false}) < \text{Cost}(\text{predicted} = \text{false}|\text{real} = \text{true})$ for each sample, recall is more important than precision. These asymmetric cost values affect the loss function, and finally change the decision boundary of the classes. The latter approach adjusts and finds the best cut-off classification threshold $t$ instead of the symmetric value for the output, which actually only translates the decision boundary [45], but is more efficient because only one training process is required (Figure 3.1). For example, to increase the importance of recall in SVMs, a cut-off classification threshold value $t < 0$ should be selected. In methods with outputs of class probability $[0, 1]$, then a threshold value $t < 0.5$ should be chosen. As noted before, for noiseless data, SVMs are stable, but for noisy data, SVMs are affected much by imbalanced support vectors. In our work, the latter approach is applied for SVMs, i.e., when $t < 0$, recall is to be improved but precision decreases. When $t > 0$, a reverse change is expected.

However, those approaches we just mentioned are in the range of traditional classification tasks of independent samples. In tasks of entity tagging in sequential data, samples are dependent. Those approaches cannot be applied directly. Thus, in CRFs we introduce a weighting parameter $\theta$ to boost features related to the true class during the testing process. Similar to the classification threshold $t$ in

Figure 3.1. Illustration of trade-off tuning between precision and recall in SVMs
SVMs, \( \theta \) can tune the trade-off between recall and precision, and may be able to improve the overall performance, since the probability of the true class increases. During the testing process, the sequence of labels \( \mathbf{y} \) is determined by maximizing the probability model 

\[
p(\mathbf{y}|\mathbf{x}, \lambda) = \frac{1}{Z(\mathbf{x})} \exp(\sum_j \lambda_j F_j(\mathbf{y}, \mathbf{x}, \theta_\mathbf{y}))
\]

where 

\[
F_j(\mathbf{y}, \mathbf{x}, \theta_\mathbf{y}) = \sum_{i=1}^{\mathbf{y}_j} \theta_{\mathbf{y}_i} s_j(\mathbf{y}_i, \mathbf{x}, i) \text{ or } \sum_{i=1}^{\mathbf{y}_j} \theta_{\mathbf{y}_i} t_j(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i)
\]

\( \theta_\mathbf{y} \) is a vector with \( \theta_{\mathbf{y}_i} = \theta \) when \( \mathbf{y}_i = \text{true} \), or \( \theta_{\mathbf{y}_i} = 1 \) when \( \mathbf{y}_i = \text{false} \), and \( \lambda_j \) are parameters learned while training.

### 3.4 Hierarchical Conditional Random Fields

Although CRFs can model multiple and long-term dependencies on the graph and may have better performance than models that do not consider those dependencies [48], in practice only short-term dependencies and features of neighbors of each vertex (i.e., a word-occurrence) are considered due to the following reasons: 1) we usually do not know what kinds of long-term dependencies exist, 2) too many features can be extracted if all kinds of long-term features are considered, and 3) most long-term features are too sparse and specific to be useful.

However, long-term dependencies at high levels may be useful to improve the accuracy of tagging tasks. For example, at the document level, non-chemical articles have much smaller probabilities of containing chemical formulae and names. At the sentence level, sentences in different chapters have different probability and/or features of chemical formulae.

Based on those observations, we propose a model of Hierarchical Conditional Random Fields (HCRFs) illustrated in Figure 3.2. HCRFs start from the highest level to the lowest level of granularity, tags each unit (e.g., document or sentence or term) with labels, and then use them as features and generate new features by interacting with other features at the lower level. At different levels, either unsupervised or supervised learning can be applied. It is hierarchical semi-supervised learning if the mixture of them are used.

The probability models of HCRFs from the highest level to the level \( m \) for a
sequence are defined as

\[
p(y_1|x, \lambda_1) = \frac{1}{Z(x)} \exp(\sum_j \lambda_{(1,j)} F_j(y_1, x)), \quad \ldots, \quad \nonumber
\]

\[
p(y_m|y_1, \ldots, y_{m-1}, x, \lambda_m) = \frac{\exp(\sum_j \lambda_{m,j} F_j(y_1, \ldots, y_m, x))}{Z(x)}, \quad (3.4)
\]

where \(y_1, \ldots, y_{m-1}\) are the labels at the level of \(1, \ldots, m-1\), and \(F_j(y_1, \ldots, y_m, x)\) is a feature function which extracts a feature from the label sequences of each level \(y_1, \ldots, y_m\) and the observation sequence \(x\). In the training process, the parameter \(\lambda\) is estimated, while in the testing process, label sequences \(y_1, \ldots, y_{m-1}\) are estimated before \(y_m\) is estimated at the level \(m\). For the level \(m\), besides the normal features

\[
S_j(y_m, x) = \sum_{i=1}^{\mid y \mid} s_j(y_{(m,i)}, x, i), \quad (3.5)
\]

and

\[
T_j(y_m, x) = \sum_{i=1}^{\mid y \mid} t_j(y_{(m,i-1)}, y_{(m,i)}, x, i), \quad (3.6)
\]

there are two types of features at each level of sequences regarding the higher level of label sequences \(y_1, \ldots, y_{m-1}\), non-interactive features
\[
S_j^0(y_1, ..., y_m, x) = \sum_{i=1}^{[y_m]} S_j^0(y_{(m,i)}, y_1, ..., y_{m-1}),
\]
and
\[
T_j^0(y_1, ..., y_m, x) = \sum_{i=1}^{[y_m]} T_j^0(y_{(m,i-1)}, y_{(m,i)}, y_1, ..., y_{m-1}),
\]
which have no interaction with the observation sequence \(x\), and interactive features
\[
S_j^{00}(y_1, ..., y_m, x) = \sum_{i=1}^{[y_m]} S_j^{00}(y_{(m,i)}, y_1, ..., y_{m-1}, x, i),
\]
and
\[
T_j^{00}(y_1, ..., y_m, x) = \sum_{i=1}^{[y_m]} T_j^{00}(y_{(m,i-1)}, y_{(m,i)}, y_1, ..., y_{m-1}, x, i),
\]
which have interaction with the observation sequence \(x\). The interactive features are generated by all the combinations of the non-interactive features and the normal features. For example, for each unit \(i\) at the level \(m\),
\[
s''_j(y_{(m,i)}, y_1, ..., y_{m-1}, x, i) = s'_j(y_{(m,i)}, y_1, ..., y_{m-1})s^T(y_{(m,i)}, x, i),
\]
where \(s\) and \(s'\) are vectors of features for each unit with sizes of \(|s|\) and \(|s'|\), and \(s''\) is a matrix with a size of \(|s|\) by \(|s'|\).

### 3.5 Chemical Entity Extraction

There are two ways to represent chemical molecules in the documents: texts and figures. A textual chemical entity, such as chemical formulae (e.g. \(CH_4\)), common chemical names (e.g. water), IUPAC nomenclature (e.g. 2,3-Butanedione), is a sequential way to represent chemical molecules. However, it is difficult to mine textual chemical molecule information from the literature, due to the following problems:

- The data have noise from text recognition errors by OCR, PDF transforma-
Non-formula
“... This work was funded under NIH grants ...”
“... YSI 5301, Yellow Springs, OH, USA ...”
“... action and disease. He has published over ...”

Formula
“... such as hydroxyl radical OH, superoxide O2- ...”
“... and the other He emissions scarcely changed ...”

Figure 3.3. Ambiguity of Chemical Formulae in Text Documents

tion, special characters, and different representative ways of molecules. A small noise may make a chemical entity to represent another structure.

- General tokenizers of natural languages may segment a chemical token into several tokens, and a chemical name may be a phrase of several terms.

- There are many standards to represent chemical structure information in text, and each has some variations.

- We do not have a lexicon including most of the chemicals occurring in the literature, especially those new discovered molecules.

An obvious method to detect chemical names is to use an lexicon, while a rule-based string pattern match approach can identify chemical formulae. However, machine learning methods to mine chemical molecule information utilizing domain knowledge like lexicons and string patterns are desired due to the following reasons: 1) Two types of ambiguity exist for tagging chemical formula. First, even though the string pattern seems like a formula, it may be just an abbreviation, e.g., NIH. Second, even though the matched term appears like a chemical formula string, it may be a word, e.g., I(Iodine) v.s. I, He(Helium) v.s. He, In(Indium) v.s. In. 2) If we match a term to terms occurring in a chemistry lexicon, then we miss newly coined names, names due to the incompleteness of the lexicon, and cannot handle noisy strings, e.g., those that occur because of small misspellings.

Tagging chemical names and tagging formulae are slightly different. The difference is that tagging chemical formulae from text documents can be viewed as a simple classification problem where the text is classified into two classes: a) chemical formulae and b) other text. However, tagging chemical names is not a simple
classification task. This is because a chemical formula usually is just a single term that is a token detected by a tokenizer, while a chemical name may be composed of several terms. Thus, the hidden labels of a formula term and a non-formula term do not have strong dependency on each other, while those of a beginning name term and a continuing name term in a single chemical name have strong dependency, e.g., a continuing name term never follows a non-name term. Hence, to tag chemical names, approaches considering dependency such as Hidden Markov Models [38] or Conditional Random Fields [20] are the best approaches, while to chemical formulae, both of these methods and traditional classification methods are potential approaches.

Then after tagging chemical names and formulae, they are analyzed and indexed to enable fast searches in response to queries defined by end-users. Each chemical formula is transformed into a canonical form (e.g. ND4 into \(2H4N\)). For chemical names, both data mining methods and rule-based methods can be applied to analyze them.

### 3.5.1 Chemical Name Extraction

A chemical name is a string sequence. Since various ways of naming a compound exist, no formal definition of chemical names here. Chemical names can be single terms, or phrases segmented by white spaces or punctuation marks. Thus, during the tokenizing process, a chemical name may be tokenized into several term tokens. Extracting chemical names is just a task of term entity tagging, where we tag the first term in the chemical name as the beginning term of a chemical name (B-name-type), and tag the following terms as continuing terms of a name (I-name-type), while any other terms are tagged as non-name terms (O). Thus, CRFs are used for chemical name extraction, and here we discuss features that are used in our work.

Generally, two categories of state features are extracted from sequences of terms: single-term features from a single term, and overlapping features from adjacent terms. There are two types of single-term features: surificial features and advanced features. Surificial features are those that can be observed directly from the term, such as word or word prefix and suffix features, orthographic features, or lists of specific terms. Advanced features are those generated by complex domain
knowledge or other data mining methods. Usually advanced features are more powerful than surfi cal features, but more expensive and sometimes infeasible to obtain. If an advanced feature has a very high correlation with the real hidden labels, then a high accuracy is expected. The classification uses linguistic features like POS tags. We use a natural language processing tool, OpenNLP \(^1\), to generate tags such as \(<\text{noun}>\) or \(<\text{proper-noun}>\). Those features are useful especially when they are used to generate overlapping features in the context of a word, i.e., if a word has initial capital and next word has a POS tag of \(<\text{verb}>\).

We summarize features used for name tagging. Note that all the features based on observations are combined with the state labels of tokens to construct transition features.

**Summary of features**

*Surfi cal features:* InitialCapital, AllCapitals, OneCapital, HasDigit, HasDash, HasPunctuation, HasDot, HasBrackets, IsChemicalElementName, IsTermLong, HasPartialStringPattern, character-\(n\)-gram features.

*Advanced features:* For chemical name tagging, we use lexicons to generate some advanced features. One lexicon includes chemical names collected online and the other is WordNet \(^2\). We use the values of Levenshtein Distance \([49]\) as features. Furthermore, we check if a term has subterms (i.e., prefix, infix, and sufl x) learned from the chemical lexicon using independent frequent subsequence mining described in the next chapter. Advanced feature examples are: NotInWordNet, FuzzyMatchChemLexiconWithEditDistance, HasSubterm.

*Overlapping features:* Overlapping features of adjacent terms are extracted. We used -1, 0, 1 as the window of features, so that for each token, all overlapping features about the last token and the next token are included in the feature set. For instance, for \(He\) in “... He is ...”, feature(\(term_{n-1} = “.” \land term_n = initial\text{Capital}\))=true, and feature(\(term_n = initial\text{Capital} \land term_{n+1} = is\text{POSTagVBZ}\))=true. This “He” is likely to be an English word instead of Helium.

\(^1\)http://opennlp.sourceforge.net/
\(^2\)http://wordnet.princeton.edu/
3.5.2 Chemical Formula Extraction

We define some preliminary concepts of chemical formulae first.

**Definition 3.5.1. Formula and Partial Formula:** Given a vocabulary of chemical elements, $E$, a chemical formula $f$ is a sequence of pairs of a partial formula and the corresponding frequency $< s_i, \text{Freq}_{s_i}>$, where each $s_i$ is a chemical element $e \in E$ or another chemical formula $f'$. A partial formula is viewed as a substructure of $f$, denoted as $s \leq f$, is a subsequence of $f$, or a subsequence of a partial formula $s_i$ in $< s_i, \text{Freq}_{s_i}>$ of $f$, or a subsequence of a partial formula $s'$ of $f$, so that if $s' \leq f \land s \leq s'$, then $s \leq f$. The length of a formula $L_f$ or a partial formula $L_s$ is defined as the number of pairs in the sequence.

A formula and a partial formula are sequences and subsequences, respectively. A partial formula is also a formula. For example, $\text{CH}_3\text{OH}$ is a chemical formula, and $C, \text{CH}_3, \text{OH}$, and $\text{CH}_3\text{OH}$ all are partial formulae. Similar to chemical names, CRFs can be applied to tag chemical formulae. Different from chemical names, chemical formulae usually are not separated by white spaces, so that mostly a formula is only one term, except cases where a formula is partitioned into more than one terms by the noise such as that from PDF conversion. Thus, extracting chemical formulae from text documents is also a binary classification problem where the text is classified into two classes: a) chemical formulae and b) other text. Even though there are label dependencies between adjacent term tags, we still can use classification methods such as SVMs, if the label dependencies are not very strong.

Similar to name tagging, we have single-term features (surficial features and advance features) and overlapping features. In our work, a rule-based approach using string pattern matching is applied to generate a set of features. Since we do not have a dictionary of all chemical molecules, and the formula of a molecule may have different string representations, we consider features of co-occurrence of two chemical elements in a formula to measure whether a matched string is a formula. For example, “C” and “O” co-occur frequently, but an element of the noble gases, e.g., “He”, and a metal element, e.g., “Cu”, are impossible to appear together in a formula.

As mentioned before, we need to distinguish formula terms from English words.
or personal names. Linguistic features like POS tags are used based on natural language processing, such as noun or proper noun. Those features are useful especially when combined with overlapping features in the context of a token. All the features that are used by our algorithms are summarized here.

**Summary of features**

**Surficial features:** InitialCapital, AllCapitals, OneCapital, HasDigit, HasDash, HasPunctuation, HasDot, HasBrackets, IsChemicalElementName, HasSuperscript, IsAmbiguousEnglishWord, IsAmbiguousPersonalName, IsAbbreviation, character-n-gram features. For features like IsChemicalElementName and IsAbbreviation, we have lists of names of chemical elements and common abbreviations, e.g. NIH.

**Advance features:** IsFormulaPattern, IsLongFormulaPattern, IsFormulaPatternWithCooccurrence, IsFormulaPatternWithSuperscript, IsPOSTagNN, IsFormulaPatternWithLowerCase, etc. String pattern matching and domain knowledge are used for features of formula pattern.

In our experiments of chemical formula tagging, we use two-level hierarchical conditional random fields to test the effect of the long-term dependency. The two levels are the sentence level and the term level. At the sentence level, we tag each sentence as “Meta” that includes information about authors, titles, journals, and references, or “Non-meta” that is the content of the document. For sentence tagging, feature examples are: ContainTermInList, ContainTermPattern, ContainSequentialTermPattern. Each feature here is actually a set of features in the same category. For example, a list of journals, a list of names, string patterns in reference, are used.

### 3.6 Experimental Evaluation

#### 3.6.1 Experiment Data and Design

The data set we used for testing is randomly selected from chemistry publications crawled from the website of the Royal Society of Chemistry\(^1\). First, 200 documents are selected randomly from the publication set, and a random part of each document is chosen. In our experiments of tagging chemical formulae, we test 2-level

\(^{1}\text{http://www.rsc.org/}\)
HCRF, where the two levels are the sentence level and the term level. At the sentence level, we construct the training set by manually labeling each sentence as Non-meta (content of the documents) or Meta (document meta data, including titles, authors, journal information, references). At the term level, we need to tag formulae and names manually. For formula tagging, we label each token as a Formula or a Non-formula. For name tagging, since a name may be a phrase of several terms, we label each token as a B-name (beginning of a chemical name), or I-name (continuing of a chemical name), or a Non-name. This data set is very imbalanced and most of the tokens are non-formula/non-name tokens (e.g. 5203 formulae vs. 321514 non-formula tokens).

We use 10-fold cross-validation to evaluate results of sentence tagging, 10-fold for formula tagging, and 5-fold for name tagging. Thus, for 10-fold cross-validation, each time we used a training set of samples obtained from 180 files and a testing set of samples obtained from the other 20 files. A method of CRFs is applied to tag sentences.

For formula tagging, several methods are evaluated, including rule-based String Pattern Match, SVM with the linear (SVM linear in the figures) and polynomial kernel (SVM poly in the figures), SVM active learning with the linear (LASVM linear in the figures) and polynomial kernel (LASVM poly in the figures), and CRFs with different feature sets, and HCRFs with all features. Features are categorized into three subsets: features using rule-based string pattern match (RULE), features using part-of-speech tags (POS), and the other features. Four combinations are tested: (1) all features, (2) no POS, (3) no RULE, and (4) no POS or RULE.

For chemical name tagging, we evaluate CRF with different feature sets. Features are classified into three subsets: features using frequent subterms (subterm), features using lexicons of chemical names and WordNet [50] (lexicon), and the other features. Four combinations are tested: (1) all features, (2) no subterm, (3) no lexicon, and (4) no subterm or lexicon.

SVM light [31] for batch learning and LASVM [32] for active learning are used. MALLET [51] is used for CRF. For CRFs and HCRFs, to avoid the overfitting problem, regularization is used, with $\sigma^2 = 5.0$. We tested complex kernels of RBF and Gaussian, which are not shown here due to the worse performances and more expensive computational costs than the linear and polynomial kernel.
<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF, $\theta = 1.0$</td>
<td>78.75%</td>
<td>89.12%</td>
<td>83.61%</td>
</tr>
</tbody>
</table>

We test different values \{0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0\} for the feature-boosting parameter $\theta$ for the Formula (or B-name and I-name) class. Note that when $\theta = 1.0$, it is the normal CRF, while when $\theta < 1.0$, the Non-formula (or Non-name) class gets more preference.

Based on experimental experiences of SVM, $C = 1/\delta^2$, where

$$
\delta = 1/n \sum_{i=1}^{n} \sqrt{\ker(x_i, x_i)} - 2 \cdot \ker(x_i, 0) + \ker(0, 0)
$$

for SVM light, $C = 100$ for LASVM, and polynomial kernel $(x \cdot x' + 1)^3$ are applied in experiments. We test different decision threshold values \{-0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8\} to adjust the trade-off between precision and recall.

### 3.6.2 Experiment results

To measure the overall performance, we use $F$-measure=$2PR/(P + R)$ [4], where $P$ is precision and $R$ is recall, instead of using error rate, since it is too small for imbalanced data. Results of average recall, precision, and $F$-measure for sentence tagging are presented in Table 3.1, formula tagging in Table 3.2 and 3.3 and Figure 3.4, 3.5, and 3.6, and name tagging in Table 3.4 and Figure 3.7. Note precision-recall curves here are different from the normal shape of precision-recall curves in information retrieval. The shape in Figure 3.4(a) and 3.5(a) can generate a $F$-measure curve with a peak, so that we can optimize it by parameter tuning. Moreover, if a precision-recall curve is situated towards the upper-right corner, then a better $F$-measure is expected.

First, the results in Table 3.1 are for the Meta class, and the results of sentence tagging reaches a reasonably good performance. Then for formula tagging, from Figure 3.4, we can see that the contribution of RULE features is much higher than that of POS features, since the difference between curves with or without POS features is quite smaller than that between curves with or without RULE.
Table 3.2. Average accuracy of formula tagging

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Pattern Match</td>
<td>98.38%</td>
<td>41.70%</td>
<td>58.57%</td>
</tr>
<tr>
<td>HCRF, $\theta = 1.0$</td>
<td>88.63%</td>
<td>96.29%</td>
<td>92.30%</td>
</tr>
<tr>
<td>HCRF, $\theta = 1.5$</td>
<td>93.09%</td>
<td>93.88%</td>
<td>93.48%</td>
</tr>
<tr>
<td>CRF, $\theta = 1.0$</td>
<td>86.05%</td>
<td>96.02%</td>
<td>90.76%</td>
</tr>
<tr>
<td>CRF, $\theta = 1.5$</td>
<td>90.92%</td>
<td>93.79%</td>
<td>92.33%</td>
</tr>
<tr>
<td>SVM linear, $t = 0.0$</td>
<td>86.95%</td>
<td>95.59%</td>
<td>91.06%</td>
</tr>
<tr>
<td>SVM linear, $t = -0.2$</td>
<td>88.25%</td>
<td>94.23%</td>
<td>91.14%</td>
</tr>
<tr>
<td>SVM poly, $t = 0.0$</td>
<td>87.69%</td>
<td>96.32%</td>
<td>91.80%</td>
</tr>
<tr>
<td>SVM poly, $t = -0.4$</td>
<td>90.36%</td>
<td>94.64%</td>
<td>92.45%</td>
</tr>
<tr>
<td>LASVM linear, $t = 0.0$</td>
<td>83.94%</td>
<td>90.65%</td>
<td>87.17%</td>
</tr>
<tr>
<td>LASVM linear, $t = -0.2$</td>
<td>85.42%</td>
<td>89.55%</td>
<td>87.44%</td>
</tr>
<tr>
<td>LASVM poly, $t = 0.0$</td>
<td>75.87%</td>
<td>93.08%</td>
<td>83.60%</td>
</tr>
<tr>
<td>LASVM poly, $t = -0.4$</td>
<td>83.86%</td>
<td>88.51%</td>
<td>86.12%</td>
</tr>
</tbody>
</table>

Table 3.3. P-values of 1-sided T-test on F-measure for formula tagging

<table>
<thead>
<tr>
<th>Pairs of methods</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF, $\theta = 1.0$; CRF, $\theta = 1.5$</td>
<td>0.130</td>
</tr>
<tr>
<td>CRF, $\theta = 1.5$; SVM, linear, $t = -0.2$</td>
<td>0.156</td>
</tr>
<tr>
<td>CRF, $\theta = 1.5$; SVM, poly, $t = -0.4$</td>
<td>0.396</td>
</tr>
<tr>
<td>CRF, $\theta = 1.5$; LASVM, linear, $t = -0.2$</td>
<td>0.002</td>
</tr>
<tr>
<td>CRF, $\theta = 1.5$; LASVM, poly, $t = -0.4$</td>
<td>0.000</td>
</tr>
<tr>
<td>SVM, linear, $t = 0.0$; SVM, linear, $t = -0.2$</td>
<td>0.472</td>
</tr>
<tr>
<td>SVM, poly, $t = 0.0$; SVM, poly, $t = -0.4$</td>
<td>0.231</td>
</tr>
<tr>
<td>SVM, linear, $t = -0.2$; SVM, poly, $t = -0.4$</td>
<td>0.072</td>
</tr>
<tr>
<td>SVM, linear, $t = -0.2$; LASVM, linear, $t = -0.2$</td>
<td>0.009</td>
</tr>
<tr>
<td>SVM, poly, $t = -0.4$; LASVM, poly, $t = -0.4$</td>
<td>0.000</td>
</tr>
</tbody>
</table>

features. Usually, the performance with more features is better than that with fewer features. We can observe that F-measure curves with fewer features are more peaky and sensitive to $\theta$, because both recall and precision change faster. HCRF has the best performance of precision, recall, and F-measure. This means long-dependence features at the sentence level has contribution. We also can see that both for HCRF and CRF using all features, we have the best overall performance based on F-measure when $\theta = 1.5$, and for this case, recall and precision are much more balanced.

From Figure 3.5(a), we can see that CRF and SVM poly both have a better performance curve than does SVM linear, but the difference is not statistically
significant at the level of 0.05 (Table 3.3). All of them are much better than LASVM, which is statistically significant. Moreover, we can see that CRF gives recall more preference instead of precision than does SVM poly. When recall ≥ precision, CRF can reach a better F-measure. This is important for imbalanced data.

We show the results for all approaches using all features in Table 3.2 and compare them with the String Pattern Match approach, which has very high recall but quite low precision. Its error of recall is caused mainly by wrong characters recognized from image PDF files using optical character recognition and special characters used which is not considered in the rules.

We also evaluate the time taken by these methods to run both for the training and testing process. Note that feature extraction and CRF are implemented in
Figure 3.5. SVM and LASVM for chemical formula tagging with different values of threshold $t$

Figure 3.6. Running time of chemical formula tagging including feature extraction
Figure 3.7. CRF for chemical name tagging using different feature sets and different values of feature boosting parameter $\theta$.

Table 3.4. Average accuracy of name tagging, $\theta = 1.0$

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>all features</td>
<td>76.15%</td>
<td>84.98%</td>
<td>80.32%</td>
</tr>
<tr>
<td>no subterm</td>
<td>74.82%</td>
<td>85.59%</td>
<td>79.84%</td>
</tr>
<tr>
<td>no lexicon</td>
<td>74.73%</td>
<td>84.05%</td>
<td>79.11%</td>
</tr>
<tr>
<td>no subterm+lexicon</td>
<td>73.00%</td>
<td>83.92%</td>
<td>78.08%</td>
</tr>
</tbody>
</table>

Java, while SVM and LASVM in C. Running time includes time of feature extraction and training (or testing) time, since in practice feature extraction must be counted. In Figure 3.6(a), we can see that CRF has a computational cost between SVM poly and the other methods. We also observe that LASVM is much faster than SVM, especially for complex kernels.

Based on these observations from our experiment results, we can conclude that
the boosting parameter for CRF and the threshold value for SVM can tune the
relation of precision and recall to find a desired trade-off and are able to improve
the overall F-measure, especially when recall is much lower than precision for
imbalanced data. CRF is more desired than SVM for our work, since it not only
has a high overall F-measure, but also a more balanced performance between recall
and precision. Moreover, CRF has a reasonable running time, lower than that of
SVM with complex kernels. In addition, during the testing process, the testing
cost of CRF is trivial compared with the cost of feature extraction. Moreover,
even though we can observe that HCRF has a better performance than CRF, it
required more training and testing time (sentence level and term level) than CRF
(term level only). Thus, CRF is enough for our tasks.

For name tagging, from Figure 3.7, we can see using all features has the best
recall and F-measure, and using features of frequent subterms can increase the
recall and the F-measure but decrease the precision. However, trade-off tuning
using the feature boosting parameter $\theta$ is not as obvious as formula tagging.

Previous works are the GATE Chemistry Tagger [52] and Oscar3 [53]. Since
they tag chemical names and formulae at the same time and cannot handle su-
perscripts, they are not fully comparable with our approach. We test them for
comparison of tagging chemical names and formulae, using 10 documents chosen
randomly from those 200 documents. For GATE, the experiment results are: recall
is 52.42%, precision 45.21%, and F-measure 48.55%. For Oscar3, the experiment
results are: recall is 51.41%, precision 70.08%, and F-measure 59.31%.
Textual Entity Indexing and Searching

In the last chapter, we have discussed how to extract entities from the text documents. In this chapter, we discuss two issues: entity indexing schemes and query models with corresponding ranking functions.

4.1 Background and Related Work

The inverted index is widely used in search engines and database systems. There are many previous works on how to improve the structure of inverted index and search process with the goal to reduce the query response time [54, 55]. In this chapter, rather than to improve the index structure, we discuss what the tokens are to construct an index of chemical names and formulae for fast search. Since substring and similarity searches are required for chemical entities, we cannot only index the whole entities. For example, some users use parts of chemical names in their search, especially, for well-known functional groups, e.g., they may use “ethyl” to search for the name in Figure 4.2. Or they may use “COOH” to search the chemical formula, “CH3COOH”. Thus, substrings of entities need to be indexed. In generic search engines, the indexed tokens usually are single terms and sometimes phrases, most of which have semantic meanings. However, chemical entities such as names and formulae are strings that have complicated domain specific semantic meanings. They are not simply tokenizable text.
If we want to enable substring and similarity searches, a naive approach is to construct an index of all possible substrings of entities in documents. However, such an index would be prohibitively large and expensive. Previous research has shown that small indices that can fit into main memory usually have much better response times [56, 57, 58]. Appropriate index pruning methods are required to reduce the index size without losing much information. We propose two methods of index pruning for the chemical name and formula indexing respectively.

For chemical name indexing, we segment a chemical name into meaningful subterms automatically by utilizing the frequencies of subterms in chemical names. Such segmentation and indexing allow end-users to perform a fuzzy search for matched chemical names, including substring search and similarity search. We propose a way to first mine independent frequent substring patterns, then use information about those substring patterns for chemical name segmentation, and finally index the subterms from the segmentation. Empirical evaluation shows that this indexing scheme results in substantial memory savings while producing comparable search results in a reasonable response time.

Similarly, for chemical formula indexing, an obvious method is to use the method of a bag of words and ignore the real structure of chemical molecules. We can view each chemical element as a token (just like a word) and its frequency is just the same as term frequency in traditional information retrieval. However, since there are too few chemical elements to distinguish molecules compared with using English words to distinguish text documents, substructures in formulae have to be considered as indexed tokens. Thus, we extend the frequent pattern mining method in [54] and introduce an index pruning strategy based on a sequential feature selection algorithm that selects frequent and discriminative substrings of formulae as features to index.

In summary, one of the key issues for entity indexing is frequent sequential pattern mining that selects frequent substrings of entities. The other key issue is sequence segmentation that segments entities into subterms for indexing. There are two categories of approaches to mining frequent patterns: mining sequential patterns and mining graph patterns. Most previous works on mining sequential patterns are done in the research area of database, which mainly focuses on three categories of issues: mining 1) the full set of frequent patterns, 2) closed frequent
subpatterns, i.e., those having no super-patterns with the same support [59], and 3) maximal frequent subpatterns, i.e., those having no frequent super-patterns [60].

Mining the full set of frequent subpatterns has the redundant problem, since all subpatterns of a frequent pattern are all frequent, and actually overlapping between occurrences of patterns is not considered. Thus, methods of mining closed frequent subpatterns are proposed to remove some of the redundant information. However, since only the cases of a subpattern and its super-patterns with the same support are removed, there are still some redundancies. Mining maximal frequent subpatterns only selects the largest frequent subpatterns, so that all redundant information is removed, but useful information is removed as well. That is, all subpatterns with frequent super-patterns are removed no matter how different their supports are from their super-patterns. In other words, mining closed frequent subpatterns prunes some of the redundant information, while mining maximal frequent subpatterns over-prunes the information including all redundant information and some useful information. Since each discovered frequent pattern is supposed to have a hidden semantic meaning, the discovered frequent patterns can be used for segmentation of sequential data with applications on text mining and Bioinformatics. Similarly, methods of mining graph patterns also fall into those three categories. For example, [61] mines a full set of frequent sub-graphs from chemical structures, while [62] discovers closed sub-graphs from chemical structures. Each discovered frequent pattern may have a hidden semantic meaning, and those meanings may be used to predict that it is carcinogenic or not [61].

Sequence segmentation in this chapter is different from text segmentation in Chapter 2, which focuses on topic segmentation, and also different from sequence labeling in Chapter 3. Sequence segmentation segments a sequence into subsequences hierarchically, where each subsequence occurs frequently in the data set. For example, at the term level, sequence segmentation is required to detect the boundaries of phrases, but does not care the label of each phrase. There are multiple applications of sequence segmentation such as Chinese segmentation [15], term chunking [63], and segmenting unstructured text into structured records [64]. Recognized phrases with unique or semantically close meanings can be used for index construction or storing into databases. Both unsupervised and supervised learning can be applied. Unsupervised learning [64] usually finds the best seg-
mentation that maximizes an objective function, where the probability of each segment is estimated with unsupervised learning. Term level sequence segmentation also can utilize the supervised methods of term entity extraction and tagging that are mentioned in Chapter 3, if we view the task of finding the boundaries of terms as a task of tagging terms with labels of a beginning term and a continuing term of a phrase. In this chapter, the character level sequence segmentation is applied to identify subterms from entities such as prefix, infix, and suffix, but no many previous works on this topic due to limited applications. Thus, we propose an unsupervised learning approach of textual sequence segmentation applied on chemical name segmentation. Usually unsupervised sequence segmentation has an intuition that most of the frequent subsequences have semantic meanings. That is why we also discuss methods of mining frequent subsequences in this chapter.

After entity indices are built, entity search service are provided online. Similar to but sightly different from keyword search of documents in information retrieval, we propose various query models to search chemical names and formulae, including exact search, frequency search, substring search, and similarity search. Usually only exact search, frequency search, and substring search are supported by current chemistry information systems [65]. Furthermore, to measure the relevance of the search results to the query, corresponding ranking functions are introduced for each query model. Traditional ranking schemes based on the Vector Space Model [35, 66] in information retrieval and features of subsequences are utilized to rank retrieved names and formulae.

4.2 Preliminaries

Before proposing our algorithms of mining frequent subsequences and automatic text segmentation, we first define some preliminary concepts.

**Definition 4.2.1. Sequence, Subsequence, Occurrence:** A sequence \( s = < t_1, t_2, ..., t_m > \), is an ordered list, where each token \( t_i \) can be an item, or a pair, or another sequence. \( L_s \) is the length of a sequence \( s \). A subsequence \( s' \preceq s \) is an adjacent part of \( s \), \( < t_i, t_{i+1}, ..., t_j >, 1 \leq i \leq j \leq m \). An occurrence of \( s' \) in \( s \), i.e. \( \text{Occur}_{s \preceq s', i, j} = < t_i, ..., t_j > \), is an instance of \( s' \) in \( s \). We say that in a sequence \( s \) two occurrences \( \text{Occur}_{s \preceq s, i, j} \) and \( \text{Occur}_{s'' \preceq s, i', j'} \) are overlapped,
i.e. \( \text{Occur}_{s',i,j} \cap \text{Occur}_{s',i',j'} \neq \emptyset \), iff \( \exists n, i \leq n \leq j \land i' \leq n \leq j' \). We say \( \text{Occur}_{s',i,j} \neq \text{Occur}_{s',i',j'} \), iff they are not overlapped.

**Definition 4.2.2. Support:** Given a whole data set \( D \) of sequences \( s \), \( D_s \) is the support of subsequence \( s' \), which is the set of all sequences \( s \) containing \( s' \), i.e. \( s' \preceq s \). \( |D_s| \) is the number of sequences in \( D_s \).

**Definition 4.2.3. Frequent Subsequence:** \( \text{Freq}_{s} \) is the frequency of \( s' \) that occurs in \( s \), that is, the count of all unique \( \text{Occur}_{s',i,j} \) without overlapping. A subsequence \( s' \) is in the set of frequent subsequences \( FS \), that is, \( s' \in FS \), if \( \sum_{s \in D_s} \text{Freq}_{s} \geq \text{Freq}_{\min} \), where \( \text{Freq}_{\min} \) is a threshold of the minimal frequency.

### 4.3 Segmentation-Based Indexing

For chemical name indexing, the intuition is that actually if we can segment a chemical name into meaningful tokens (like terms in document search), then we can index them instead of all possible substrings. These indexed tokens can support similarity searches and most meaningful substring searches, which are shown in the experiment results in Section 4.8. Meaningful substring searches refer to queries using meaningful substrings instead of arbitrary substrings. That is, when a user submits a query to search a name with a substring, she usually uses a meaningful substring, e.g., to search methylethyl, methyl or ethyl is used, instead of hyleth. Thus, for methylethyl, indexing methyl and ethyl is enough, while hyleth is not necessary. Hence, after HTS, we only need to index substrings at each node, which reduces the index size tremendously, and our index scheme is able to maintain most of the meaningful information.

#### 4.3.1 Independent Frequent Subsequence Mining

Mining frequent subsequences is a classical data mining issue. However, previous work mainly focus on algorithms instead of semantic meanings. In many real world problems, mining semantic meanings is the eventual goal. For example, in East Asian Language processing such as Chinese text segmentation, frequent subsequences of characters are usually meaningful phrases. In our task of segmenting chemical names, frequent subterms usually are prefixes, infixes, and suffixes of
chemical names. Slightly different from previous work [67, 68, 62, 69, 60], we give
the following definitions:

**Definition 4.3.1. Closed Frequent Subsequence:** A frequent subsequence $s$ is
in the set of closed frequent subsequences $CS$, iff there does not exist a frequent
super-sequence $s'$ such that all the frequencies of $s$ in $s'' \in D$ is the same as that
of $s'$ in $s''$. That is $s \in CS = \{s|s \in FS$ and $\nexists s' \in FS$ such that $s < s' \land \forall s'' \in D, \text{Freq}_{s' \leq s''} = \text{Freq}_{s \leq s''}\}$.

**Definition 4.3.2. Maximal Frequent Subsequence:** A frequent subsequence
$s$ is in the set of maximal frequent subsequences $MS$, iff it has no frequent super-
sequences, i.e. $s \in MS = \{s|s \in FS$ and $\nexists s' \in FS$ such that $s < s'\}$.

**Algorithm 2** Independent Frequent Subsequence Mining

**Algorithm:** IFSM($C, D_{all}, O_{all}, \text{Freq}_{\text{min}}, L_{min}$):

**Input:**
Candidate set $C$,
set $D_{all}$ including the support $D_s$ for each subsequence $s$,
set $O_{all}$ including all $Occur_{s \in C}$,
minimal threshold value of frequency $\text{Freq}_{\text{min}}$, and
minimal length of subsequence $L_{min}$.

**Output:**
Set of Independent Frequent Subsequences $IS$ and Independent Frequency $IFreq_s$
of $s \in IS$.

1. **Initialization:** $IS = \{\emptyset\}$, length $l = \max_s(L_s)$.
2. **while** $l \geq L_{min}$, do
3. put all $s \in C$, $L_s = l$,
   $\sum_{s' \in D_s} \text{Freq}_{s' \leq s} \geq \text{Freq}_{\text{min}}$ into Set $S$;
4. **while** $\exists s \in S, \sum_{s' \in D_s} \text{Freq}_{s' \leq s} \geq \text{Freq}_{\text{min}}$, do
5. move $s$ with the largest $\text{Freq}_{s' \leq s}$ from $S$ to $IS$;
6. **for each** $s' \in D_s$
7. **for each** $Occur_{s'' \leq s'} \cap (\cup_{s' \leq s} Occur_{s \leq s'}) \neq \emptyset$
8. $\text{Freq}_{s'' \leq s'} = -$;
9. $l = -$;
10. **return** $IS$ and $IFreq_{s \in IS} = \text{Freq}_s$;
* $\cup_{s' \leq s} Occur_{s \leq s'}$ is the range of all $Occur_{s \leq s'}$ in $s'$, except $Occur_{s \leq s'} \cap Occur_{t \leq s'} \neq \emptyset \land t \in IS$.

Note that our definitions are different from those in previous works [67, 62, 60],
which ignore the frequency of a subsequence in its super-sequences. They also
did not consider overlapping of subsequences. We consider each occurrence of a subsequence, since its super-sequence may have more than one occurrence of the same subsequence. However, those concepts are not enough to define a subterm, which is a frequent subsequence. For example, in Figure 4.1, $\text{Freq}_{\text{min}} = 2$, $\text{FS} = \{abcd, abc, bcd, ab, bc, cd\}$ has much redundant information. $\text{CS} = \{abcd, ab, bc\}$ removes some redundant information, e.g., $abc/bcd/cd$ only appears in $abcd$. $\text{MS} = \{abcd\}$ removes all redundant information as well as useful information, e.g., $ab/bc$ has occurrences excluding those in $abcd$. Thus, we need to determine if $ab/bc$ is still frequent excluding occurrences in $abcd$. The intuition is that, for a frequent sequence $s'$, its subsequence $s \prec s'$ is also frequent independently, only if the number of all the occurrences of $s$ not in any occurrences of $s'$ is larger than the minimal threshold $\text{Freq}_{\text{min}}$. If a subsequence $s$ has more than one frequent super-sequences, then all the occurrences of those super-sequences are excluded to count the total frequency of $s$ to judge if $s$ is frequent independently. Thus, in Figure 4.1, $abcd$ is frequent, and $ab$ is frequent independently, but $bc$ is not, since $ab$ occurs twice independently without $abcd$ but $bc$ only once independently. Thus, we get a new set $\{abcd, ab\}$ without redundant information and with all useful information. Extending previous work, we give the following definition of \textit{Independent Frequent Subsequence}:

\textbf{Definition 4.3.3. Independent Frequent Subsequence:} A frequent subsequence $s$ is in the set of independent frequent subsequences $\text{IS}$, iff the independent frequency of $s$, $\text{IFreq}_s$, i.e. the total frequency of $s$, excluding all the occurrences of its independent frequent super-sequences $s' \in \text{IS}$, is at least $\text{Freq}_{\text{min}}$. That is $s \in \text{IS} = \{s | s \in \text{FS} and \text{IFreq}_s \geq \text{Freq}_{\text{min}}\}$, where $\text{IFreq}_s = \sum_{s'' \in D} \#\text{Occur}_{s \leq s''}$, $\forall s' \in \text{IS}$, $\forall s'' \in D$, $\#\text{Occur}_{s' \leq s''} \cap \text{Occur}_{s \leq s''} \neq \emptyset$ and $\#\text{Occur}_{s \leq s''}$ is the number of unique occurrences of $s$ in $s''$.

A subterm of a chemical name or a substructure of a chemical formula is equal to an independent frequent subsequence, not a closed or maximal frequent subsequence. These three concepts are different from each other. For example, if a frequent subsequence $s \in \text{FS}$ appears mostly in its super-sequence $s' \succ s$ but not all, then from the definitions, $s$ is a closed frequent subsequence. However, since $s$ occurs alone infrequently, it is not an independent frequent subsequence. If an
Input Sequences: abcde, abcd, aba, abd, bca,
Parameters: \( F_{\text{min}} = 2, \ L_{\text{min}} = 2 \)
1 = 5, \( F_{\text{abcd}} = F_{\text{abcdf}} = 1, IS = \emptyset \); 
1 = 4, \( F_{\text{abcd}} = 2, IS = \{abcd\} \); now for \( s = \)
\( abc/\text{bcd}/ab/bc/\text{cd} \), \( F_{\text{abcdef}} = 0 \); \( F_{\text{abcde}} = 0 \);
for \( s = \text{cde/de} \), \( F_{\text{abcdef}} = 0 \);
for \( s = \text{cdf/df} \), \( F_{\text{abcdef}} = 0 \);
1 = 3, all \( F_{\text{ab}} < 2 \), \( IS = \{abcd\} \);
1 = 2, \( F_{\text{ab}} = F_{\text{ab} \leq \text{abc}} + F_{\text{ab} \leq \text{abd}} = 2 \), but \( F_{\text{bc}} = F_{\text{bc} \leq \text{bca}} = 1 \),
so \( IS = \{abcd, ab\} \);
Return: \( IS = \{abcd, ab\} \) with \( IF_{\text{abcd}} = 2 \) & \( IF_{\text{ab}} = 2 \).

**Figure 4.1.** An Example of Independent Frequent Subsequence Mining

An independent frequent subsequence \( s \) has a super-sequence \( s' \), \( s \prec s' \land s' \in FS \), \( s \) is
not a maximal frequent subsequence, but it may be a meaningful subterm. Based
on these observations, we propose an algorithm of IFSM from a training set in
Algorithm 2 with an example in Figure 4.1. This algorithm starts from the longest
sequence \( s \) to the shortest sequence, checking if \( s \) is frequent. If \( F_{\text{min}} \geq F_{\text{min}},
\) put \( s \) in \( IS \), remove all occurrences of its subsequences that are in any occurrences
of \( s \), and remove all occurrences overlapping with any occurrences of \( s \). If the re-
main ing occurrences of a subsequence \( s' \) still make \( s' \) frequent, then put \( s' \) into \( IS \).
After mining independent frequent subterms, the discovered subterms can used as
features in CRFs, and their independent frequencies can be used to estimate their
probabilities for hierarchical text segmentation.

### 4.3.2 Hierarchical Text Segmentation

As mentioned before, textual sequence segmentation can be applied to segment
text into tokens for index construction. In many cases like Chinese segmentation
[15] or chemical entity segmentation [70], when there is no natural boundaries
of segmentation symbols, like white spaces or punctuation marks, data mining
methods have to be used. In our work, a chemical name following the IUPAC
nomenclature may be complex with structural information (Figure 4.2).

However, not all the substrings are meaningful. Segmentation symbols can be
used to segment a name into terms, but for a term like *methylene*, a data mining
method is required to segment it into *methyl* and *ethyl*. Thus, we propose an
Algorithm 3 Hierarchical Text Segmentation

Algorithm: HTS($s, IF, P, r$):

Input:
A sequence $s$,
a set of independent frequent strings $IF$ with corresponding independent frequency $IFreq_{s' \in IF}$,
a set of natural segmentation symbols with priorities $P$, and
the tree root $r$.

Output: The tree root $r$ with a tree representation of $s$.

1. if $s$ has natural segmentation symbols $c \in P$
2. segment $s$ into subsequences $< s_1, s_2, ..., s_n >$
   using $c$ with the highest priority;
3. put each $s' \in \{ s_1, s_2, ..., s_n \}$ in a child node $r' \in \{ r_1, r_2, ..., r_n \}$ of $r$;
4. for each subsequence $s'$ in $r'$ do
5.    HTS($s', IF, P, r'$);
6. else if $L_s > 1$
7.    DynSeg($s, IF, r, 2$);
8. else return;

unsupervised hierarchical text segmentation method, which first uses segmentation
symbols to segment chemical names into terms (HTS in Algorithm 3), and then
utilizes the independent frequent substrings discovered by Algorithm IFSM for
further segmentation into subterms (DynSeg in Algorithm 4). The algorithm of
DynSeg to segment terms works like this: After mining the independent frequent
substrings to estimate independent frequency of each substring $s$, we can estimate
Algorithm 4: Text Segmentation using Dynamic Programming

Algorithm: DynSeg($t, IF, r, n$):

Input:
A sequence $t = < t_1, t_2, ..., t_m >$,
a set of independent frequent strings $IF$ with corresponding independent frequency $IFreq_{s' \in IF}$,
the tree root $r$, and
the number of segments $n$.

Output: The tree root $r$ with a tree representation of $s$.

1. if $L_s = 1$ return;
2. Compute all $\log(\text{IFreq}_{s_j}) = \log(\text{IFreq}_{<t_j, t_{j+1}, ..., t_k>})$,
   $1 \leq j < k \leq m$, where $s_i = < t_j, t_{j+1}, ..., t_k >$
is a subsequence of $t$.
3. Let $M(l, 1) = \log(\text{IFreq}_{<t_1, t_2, ..., t_l>})$, where $0 \leq l \leq m$.
   Then $M(l, L) = \max_d(M(d, L - 1) +
   \log(\text{IFreq}_{<t_{d+1}, t_{d+2}, ..., t_l>}))$, where $L < n$ and $0 \leq d \leq l$.
   Note $\log(\text{IFreq}_{<t_j, t_{j+1}, ..., t_k>}) = 0$, if $j > k$.
4. $M(m, n) = \max_d(M(d, n - 1) + \log(\text{IFreq}_{<t_{d+1}, t_{d+2}, ..., t_l>}))$,
   where $1 \leq d \leq l$.
5. segment $s$ into subsequences $< s_1, s_2, ..., s_n >$
   using the corresponding $\text{seg}(t)$ for $M(m, n)$.
6. if only one $s' \in \{s_1, s_2, ..., s_n\} \neq \emptyset$ return;
7. put each $s' \in \{s_1, s_2, ..., s_n\}$ in a child node
   $r' \in \{r_1, r_2, ..., r_n\}$ of $r$;
8. for each subsequence $s'$ in $r'$ do
9. DynSeg($s', IF, r', n$);

its probability by

$$ P(s) = \frac{\text{IFreq}_s}{\sum_{s' \in IS} \text{IFreq}_{s'}} \propto \text{IFreq}_s, \quad (4.1) $$

where $\text{IFreq}_s$ is the independent frequency of $s$. For each subterm $t$ with $L_t = m$,
a segmentation $\text{seg}(t) = < t_1, t_2, ..., t_m > \rightarrow < s_1, s_2, ..., s_n >$ is to cluster adjacent
tokens into $n$ subsequences, where usually $n = 2$ for recursive segmentation. The probability of the segmentation is

$$ P(\text{seg}(t)) = \prod_{i \in [1, n]} P(s_i), \quad (4.2) $$

and the corresponding log-likelihood is
Thus, maximum (log) likelihood is applied to find the optimal segmentation,

\[
\text{seg}(t) = \arg\max_{\text{seg}(t)} \sum_{i \in [1,n]} \log(P(s_i))
\]

(4.4)

Dynamic programming for text segmentation [71] can be used to find the optimal segmentation that maximizes the objective function of log-likelihood. The algorithm of hierarchical text segmentation (HTS) is presented in Algorithm 3, and text segmentation using dynamic programming (DynSeg) is shown in Algorithm 4. Once chemical names are segmented into a hierarchical tree of subterms, the index is able to be built based on those subterms.

### 4.4 Frequency-and-Discrimination-Based Indexing

For chemical formula indexing, since the number of all possible partial formulae of the formula set is quite large and many of them have redundant information, indexing every one is prohibitively expensive and not necessary. We propose an index pruning method in [1] to sequentially select features of partial formulae from the shortest to the longest for index construction, based on two criteria: the next selected feature should be 1) frequent, and 2) discriminative, i.e. its support \(D_s\) should not overlap too much with the intersection of supports \(D_{s'}\) of all its selected subsequences \(s' < s\) in the set of selected features \(F\). To measure the discrimination, a discriminative score is defined for each feature candidate with respect to \(F\). A feature \(s\) is discriminative with respect to \(F\), if \(\|D_s\| << \| \cap_{s' \in F \land s' \leq s} D_{s'} \| D_f \). Thus, the discriminative score for each candidate \(s\) with respect to \(F\) is defined as:

\[
\alpha_s = \frac{| \cap_{s' \in F \land s' \leq s} D_{s'} |}{|D_s|}.
\]

(4.5)

To support similarity search, partial formulae of each formula are useful as
Algorithm 5 Sequential Feature Selection

**Input:**
Candidate Feature Set $C$ with frequency $Freq_s$ and support $D_s$ for each substructure $s$.
Minimal threshold value of frequency $Freq_{min}$.
Minimal discriminative score $\alpha_{min}$.

**Output:**
Selected Feature Set $F$.

1. **Initialization:** $F = \{\emptyset\}$, $D_\emptyset = D$, length $l = 0$.
2. while $C$ is not empty, do
3. $l = l + 1$;
4. for each $s \in C$
5. if $Freq_s > Freq_{min}$
6. if $L_s = l$
7. compute $\alpha_s$ using Equation 4.5 ($\alpha_s^{(0)} = \frac{|D_s|}{|D|}$, since
   no $s$ satisfies $s' \preceq s \wedge s' \in F$)
8. if $\alpha_s > \alpha_{min}$
9. move $s$ from $C$ to $F$;
10. else remove $s$ from $C$;
11. else remove $s$ from $C$;
12. return $F$;

possible substructures for indexing. However, since partial formulae of a partial formula $s \preceq f$ with $L(s) > 1$ are also partial formulae of the formula $f$, the number of all partial formulae of the formula set is quite large. For instance, the candidate features of $\text{CH}_3\text{OH}$ are $\text{C}$, $\text{H}_3$, $\text{O}$, $\text{H}$, $\text{CH}_3$, $\text{H}_3\text{O}$, $\text{OH}$, $\text{CH}_3\text{O}$, $\text{H}_3\text{OH}$, and $\text{CH}_3\text{OH}$. We do not need to index every one due to redundant information. For example, two similar partial formulae may appear in the same set of formulae (e.g. $\text{CH}_3\text{CH}_2\text{COO}$ and $\text{CH}_3\text{CH}_2\text{CO}$), because they generate from the same super sequence. In this case, it is enough to index only one of them. Moreover, it is not important to index infrequent fragments. For example, a complex partial formula appearing only once in the formula set is not necessary for indexing, if its selected fragments are enough to distinguish the formula having it from others. E.g. when querying formulae having partial formulae of $\text{CH}_3, \text{CH}_2, \text{CO}, \text{OH}, \text{COOH}$, and $\text{COOH}$, if only $\text{CH}_3\text{CH}_2\text{COOH}$ is returned, then it is not necessary to index $\text{CH}_3\text{CH}_2\text{COO}$. Using a similar idea and notations about feature selection in [54], given a whole data set $D$, $D_s$ is the support of substructure $s$, the set of all formulae containing $s$, and
$|D_s|$ is the number of items in $D_s$. All substructures of a frequent substructure are frequent too. Based on these observations, two criteria may be used to sequentially select features of substructures into the set of selected features $F$. The feature selected should be 1) frequent, and, 2) its support should not overlap too much with the intersection of supports of its selected substructures in $F$.

For Criterion 1, mining frequent substructures is required in advance. After the algorithm extracts all chemical formulae from documents, it generates the set of all partial formulae and records their frequencies. Then, for Criterion 2, we define a discriminative score for each feature candidate with respect to $F$. Similar to the definitions in [54], a substructure $s$ is redundant with respect to the selected feature set $F$, if $|D_s| \approx |\cap_{s' \in F \land s' \leq s} D_{s'}|$. A substructure $s$ is discriminative with respect to $F$, if $|D_s| < |\cap_{s' \in F \land s' \leq s} D_f|$. Thus, the discriminative score for each candidate $s$ with respect to $F$ is defined as:

$$\alpha_s = |\cap_{s' \in F \land s' \leq s} D_{s'}|/|D_s|.$$ \hspace{1cm} (4.6)

The sequential feature selection algorithm is described in Algorithm 5. The algorithm starts with an empty set $F$ of selected features, scanning each substructure from the length $l = 1$ to $l = L(s)_{max}$. At each length of substructure, all frequent candidates with discriminative scores larger than the threshold are selected. This scanning sequence ensures that at each length of substructure, no scanned substructure is a substructure of another scanned one. Thus, only selected substructures at previous steps are considered to compute the discriminative scores. All substructures $s$ with $L(s) > l$ but $Freq(s) \leq Freq_{min}$ are removed directly from the candidate set $C$, because even when $L(s) = l$ after several scanning cycles to longer substructures, $Freq_s$ still has the same value, and $\alpha_s$ would decrease or remain the same. Consequently, the feature is not selected. Finally, after feature selection, all the selected subsequence features are used to index chemical formulae.
4.5 Chemical Formula Searches

4.5.1 Query Models

We propose four types of queries for chemical formula search: exact formula searches, frequency formula searches, substructure formula searches, and similarity formula searches. Different from name queries we discuss later, users need to specify element frequency ranges in a formula query, defined as follows,

Definition 4.5.1. **Formula Query and Frequency Range:** A formula query $q$ is a sequence of pairs of a partial formula and the corresponding frequency range $< s_i, \text{range}_{s_i} >$, where token $s_i$ is a chemical element $e \in E$ or another chemical formula $f'$, and $\text{range}_{s_i} = \cup_k [\text{low}_k, \text{upper}_k], \text{upper}_k \geq \text{low}_k \geq 0$.

**Exact formula search**

The answer to an exact formula search query is formulae having the same sequence of partial formulae within the frequency ranges specified in the query. Exact formula search usually is used to search exact representation of a chemical molecule. Different formula representations for the same molecule cannot be retrieved. For instance, the query $C1-2H4-6$ matches $CH4$ and $C2H6$, but not $H4C$ or $H6C2$.

**Frequency formula searches**

We say that a user runs a frequency formula search, when she specifies the elements and their frequencies. All documents with chemical formulae that have the specified elements within the specified frequency ranges are returned. As indicated before, most current chemistry databases support frequency searches as the only query models for formula search. There are two types of frequency searches: full frequency searches and partial frequency search. When a user specifies the query $C2H4-6$, the system returns documents with the chemical formulae with two $C$ and four to six $H$, and no other atoms for full frequency search, e.g., $C2H4$, and returns formulae with two $C$, four to six $H$ and any numbers of other atoms for partial frequency search, e.g., $C2H4$ and $C2H4O$.

**Substructure formula search**

Substructure formula searches find formulae that may have a query substructure defined by users. In substructure searches, the query $q$ has only one partial formula
$s_1$ with $range_{s_1} = [1, 1]$, and retrieved formulae $f$ have $freq_{s_1} \geq 1$. However, since the same substructure may have different appearances in formulae, three types of matches are considered with different ranking scores (Section 4.5.2). For example, for the query $COOH$, $COOH$ gets an exact match (high score), $HOOC$ reverse match (medium score), and $CHO2$ parsed match (low score).

**Similarity formula search**

*Similarity formula searches* return documents with chemical formulae with similar structures to the query formula, i.e., a sequence of partial formulae $s_i$ with a specific $range_{s_i}$, e.g. $CH3COOH$. The edit distance is not used for formula similarity search for the same reasons in name similarity search. For the second reason, we show an example: $H2CO3$ can also be mentioned as $HC(O)OOH$, but the edit distance of them is larger than that of $H2CO3$ and $HNO3$ ($6 > 2$). Using the partial formula based similarity search of $H2CO3$, $HC(O)OOH$ has a higher ranking score than $HNO3$ based on Equation 4.9. Our approach is feature-based similarity search, since full structure information is unavailable in formulae. The algorithm uses selected partial formulae as features. We design and present a scoring function in Section 4.5.2 based on all selected partial formulae that are selected and indexed in advance, so that the query processing and the ranking score computation is efficient. Formulae with top ranking scores are retrieved.

**Conjunctive search**

*Conjunctive searches* of the basic chemical formula (or name) searches are supported for filtering search results, so that users can define various constraints to search desired formulae. For example, a user can search formulae that have two to four $C$, four to ten $H$, and may have a substructure of $CH2$, using a conjunctive search of a full frequency search $C2-4H4-10$ and a substructure search of $CH2$.

### 4.5.2 Ranking Functions

A scoring scheme based on the Vector Space Model in information retrieval and features of partial formulae is used to rank retrieved formulae. We adapt the concepts of the term frequency $TF$ and the inverse document frequency $IDF$ to chemical entity searches.

**Definition 4.5.2.** $SF.IEF$ and Atom Frequency: *Given a collection of entities*
C, a query $q$ and an entity $e \in C$, $SF(s, e)$ is the subsequence frequency for each subsequence $s \leq e$, which is the total number of occurrences of $s$ in $e$, $IEF(s)$ is the inverse entity frequency of $s$ in $C$, and defined as

$$SF(s, e) = \frac{freq(s,e)}{|e|}, IEF(s) = \log \frac{|C|}{|\{e|s \leq e\}|}.$$  

where $freq(s,e)$ is the frequency of $s$ in $e$, $|e| = \sum_k freq(s_k, e)$ is the total frequency of all indexed subsequences in $e$, $|C|$ is the total number of entities in $C$, and $|\{e|s \leq e\}|$ is the number of entities that contain subsequence $s$. Atom frequency refers to the subsequence frequency of an chemical atom in a chemical formula.

**Frequency formula searches**

For a query formula $q$ and a formula $f \in C$, the scoring function of frequency searches is given as

$$score(q, f) = \frac{\sum_{e \subseteq q} W(e) SF(e, f) IEF(e)^2}{\sqrt{|f|} \times \sqrt{\sum_{e \subseteq q} (W(e) IEF(e))^2}},$$  \hspace{1cm} (4.7)

where $|f| = \sum_k freq(e_k, f)$ is the total atom frequency of chemical elements in $f$, $1/\sqrt{|f|}$ is a normalizing factor to give a higher score to formulae with fewer atoms, $1/\sqrt{\sum_{e \subseteq q} (W(e) IEF(e))^2}$ is a factor that makes scores comparable between different queries. It does not affect the rank of retrieved formulae for a specific formula query, but affects the rank of retrieved documents, if there are more than two formula searches embedded in the document search. Without this factor, documents containing the longer query formula get higher scores. Equation 4.7 considers $f$ as a bag of atoms, where $e \preceq f$ is a chemical element. $W(e)$ is the weight of $e$ that represents how much it contributes to the score. It can adjust the weight of each $e$ together with $IEF(e)$. Without domain knowledge $W(e) = 1$.

**Substructure formula search**

The scoring function of substructure search is given as

$$score(q, f) = W_{mat(q,f)} SF(q, f) IEF(q)/\sqrt{|f|},$$  \hspace{1cm} (4.8)

where $W_{mat(q,f)}$ is the weight for different matching types, exact match (high weight, e.g., 1), reverse match (medium weight, e.g., 0.8), and parsed match (low
weight, e.g., 0.25), which are defined by experiences.

**Similarity formula search**

A scoring function like a sequence kernel [72] is designed to measure similarity between formulae for similarity search. It maps a query formula implicitly into a vector space where each dimension is a selected partial formula using the sequential feature selection algorithm. For instance, the query \( \text{CH}_3\text{OH} \) is mapped into dimensions of \( C, H_3, O, H, \text{CH}_3 \), and \( OH \), if only these six partial formulae are selected. Then formulae with those substructures (including reverse or parsed matched substructures) are retrieved, and scores are computed cumulatively. Larger substructures are given more weight for scoring, and scores of long formulae are normalized by their total frequency of substructures. The scoring function of similarity search is given as

\[
\text{score}(q, f) = \frac{\sum_{s \subseteq q} W_{\text{mat}(s, f)} W(s)SF(s, q)SF(s, f)IEF(s)}{\sqrt{|f|}},
\]

where \( W(s) \) is the weight of the substructure \( s \), which is defined as the total atom frequency of \( s \).

### 4.6 Chemical Name Searches

#### 4.6.1 Query Models

Similar to chemical formula searches, we propose three basic types of queries for chemical name search: *exact name searches*, *substring name searches*, and *similarity name searches*. Usually only exact and substring name searches are supported by current chemistry information systems. Moreover, we also apply a similar way to rank chemical name as what we do for chemical formulae ranking.

**Exact name search**

A query of *exact name searches* is to search for exact name representations of chemical molecules.

**Substring name search**

*Substring name searches* retrieve chemical names that have a query substring defined by users. In substring name searches, the query \( q \) has only one substring
Similarity name search

Similarity name searches return documents with chemical names which are similar to the query chemical name. Similar to similarity formula searches, the edit distance is not applied on similarity name searches. Our approach is feature-based similarity search, where features of substrings are used to measure the similarity. We design a ranking function based on indexed substrings, so that the query processing and the ranking score computation is efficient.

4.6.2 Ranking Functions

Substring name search

The ranking function of substring name search regarding a query \( q \) and a name \( e \) is given as follows,

\[
\text{score}(q, e) = SF(q, e) IEF(q) / |e|^{\frac{1}{2}}. \tag{4.10}
\]

Similarity name search

For a query of similarity name search, first a query name is segmented hierarchically, and then substrings at each node is used to retrieve names in the collection, and scores are computed cumulatively. Longer substrings are given more weight for scoring, and scores of names are normalized by their total frequency of substrings. The ranking function of similarity name search is given as follows,

\[
\text{score}(q, e) = \sum_{s \subseteq q} W(s) SF(s, q) SF(s, e) IEF(s) / |e|, \tag{4.11}
\]

where \( W(s) \) is the weight of the substring \( s \), which is defined as the length of \( s \).

4.7 Document Search

Since the ultimate goal of users is to search relevant documents, the users can search using chemical names and/or formulae as well as other keywords. The search is performed in two stages. First, a query string is analyzed. All the embedded formula searches are taken out, and all possible desired formulae are retrieved.
Then, after relevant formulae are returned, the original query string is rewritten by embedding those formulae into the corresponding positions of the original query string as sub-groups with OR operators. To involve the relevance score for each retrieved formula, boosting factors with the values of relevance scores are added to each retrieved formula with a goal to rank corresponding documents. Second, the rewritten query is used to search relevant documents. For example, if a user searches documents with the term *oxygen* and the formula *CH₄*, the formula search *CH₄* is processed first and matches *CH₄* and *H₄C* with corresponding scores of 1 and 0.5. Then the query is written as “*oxygen* (CH₄^1 OR H₄C^0.5)”, where 1 and 0.5 are the corresponding boosting factors. Then documents with *CH₄* get higher scores.

4.8 Experimental Evaluation

In this section, we examine the proposed methods with experiments, and we present and discuss corresponding experimental results.

4.8.1 Independent Frequent Subsequence Mining and Hierarchical Text Segmentation

4.8.1.1 Experiment Data and Design

We collect 221,145 chemical names on the web, which can be used as a lexicon to tag chemical names. Each chemical name is a phrase of one or more terms. There are two goals for our algorithm for mining independent frequent subsequences: 1) to be features for discovering new chemical names using frequent subterms in CRFs/HCRFs, and 2) to estimate independent frequency of substrings for hierarchical text segmentation. We evaluate our algorithm with different threshold values \( Freq_{min} = \{10, 20, 40, 80, 160\} \). We first tokenize chemical names and get 66769 unique terms. Then frequent subterms are discovered from them. After independent frequent subsequence mining, hierarchical text segmentation is tested.


<table>
<thead>
<tr>
<th>String</th>
<th>Freq</th>
<th>Meaning</th>
<th>String</th>
<th>Freq</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tetramethyl</td>
<td>295</td>
<td>H₃C\text{~CH₃} \text{~CH₃}</td>
<td>hydroxy</td>
<td>803</td>
<td>\text{~OH}</td>
</tr>
<tr>
<td>tetrahydro</td>
<td>285</td>
<td>H\text{~H} \text{~H}</td>
<td>methyl</td>
<td>1744</td>
<td>\text{~CH₃}</td>
</tr>
<tr>
<td>trimethyl</td>
<td>441</td>
<td>H₃C\text{~CH₃}</td>
<td>ethyl</td>
<td>1269</td>
<td>\text{~CH₂-CH₃}</td>
</tr>
<tr>
<td>dimethyl</td>
<td>922</td>
<td>H₃C\text{~CH₃}</td>
<td>thi</td>
<td>811</td>
<td>\text{<del>S</del>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tri</td>
<td>2597</td>
<td>three</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>di</td>
<td>4154</td>
<td>two</td>
</tr>
</tbody>
</table>

Table 4.1. The most frequent subterms at each length, \( Freq_{\min} = 160 \)

(a) Discovered Independent Frequent Subterms

(b) Running time of the Algorithm IFSM

Figure 4.3. Mining Independent Frequent Substrings

4.8.1.2 Experiment results

The results of independent frequent subsequence mining is shown in Figure 4.3. The distributions of subterm’s lengths with different values of \( Freq_{\min} \) are presented in Figure 4.3(a) and the time complexity of the algorithm is shown in Figure 4.3(b). Most discovered subterms have semantic meanings in the chemical domain. We show some of the most frequent subterms with their real meanings in Table 4.1. After we discover independent frequent subsequences with their independent frequency, we can use the algorithm of hierarchical text segmentation. Most of substrings at each node in the binary segmentation tree have semantic meanings. Two examples of hierarchical text segmentation results are shown in Figure 4.4.
Figure 4.4. Examples of hierarchical text segmentation

(a) Ratio of selected features

(b) Ratio of index size

Figure 4.5. Features and index size ratio after feature selection for formula indexing

4.8.2 Textual Entity Information Indexing

For chemical formula indexing and search, we use the frequency-and-discrimination-based index construction and pruning, while for chemical name indexing and search, we use the segmentation-based index construction and pruning. We show experimental results of index construction in this section and experimental results of search in the next section.

4.8.2.1 Chemical Formula Indexing

For formula indexing and search, we test the sequential feature selection algorithm for index construction and evaluate retrieved results. We select a set of 5036 docu-
ments and extract 15853 formulae with a total of 27978 partial formulae before feature selection. Different values for the frequency threshold $Freq_{min} = \{1, 2, 3, 4, 5\}$ and the discrimination threshold $\alpha_{min} = \{0.9, 1.0, 1.2\}$ are tested, and results are shown in Figure 4.5, 4.6, and 4.7.

Note that when $\alpha_{min} = 0.9$, all frequent partial formulae are selected without considering the discriminative score $\alpha$. When $\alpha_{min} = 1.0$, each partial formula whose support can be represented by the intersection of its selected substructures’ supports is removed. We do not lose information in this case because all the information of a removed frequent structure has been represented by its selected partial formulae. When $\alpha_{min} > 1.0$, feature selection is lossy since some information is lost. After feature selection and index construction, we generate a list of 100 query formulae that are selected randomly from the set of extracted formulae and from a chemistry textbook and web pages. These formulae are used to perform similarity

Figure 4.6. Correlation of similarity formula search results after feature selection
Figure 4.7. Running time of feature selection for formula indexing

searches.

The experiment results in Figure 4.5 show that depending on different threshold values, most of the features are removed after feature selection, so that the index size decreases correspondingly. Even for the case of $Freq_{min} = 1$ and $\alpha_{min} = 0.9$, 75% of the features are removed, since they appear only once. We can also observe that from $\alpha_{min} = 0.9$ to $\alpha_{min} = 1.0$, many features are also removed, since those features have selected partial formulae with the same support $D$. When $\alpha_{min} \geq 1.0$, the selective ratio change a little. We also evaluated the runtime of the feature selection algorithm, illustrated in Figure 4.7. We can see that a larger $Freq_{min}$ can filter infrequent features directly without computing discriminative scores, which speeds up the algorithm, while the value of $\alpha_{min}$ affect the runtime little.

4.8.2.2 Chemical Name Indexing

We compare the proposed approach with the method using all possible substrings for indexing. A subset of the collection in Section 4.8.1 with 37656 chemical names is used for indexing and search. The whole data set is not used for two reasons: 1) the method using all possible substrings generates too many substrings to construct the index, and 2) names that are not indexed can be used as new names for search experiments.

To index chemical names, we first mine independent frequent substrings, then do hierarchical text segmentation for each chemical name, and finally index each name using substrings in the nodes of the hierarchical text segmentation tree. The experiment results in Figure 4.8 show that most (99%) of the substrings are
removed after hierarchical text segmentation, so that the index size decreases correspondingly (only 6% of the original size left).

4.8.3 Textual Entity Information Search

4.8.3.1 Chemical Formula Search

The most important result from our experiments is that for the same similarity search query, the search results with feature selection are similar to those without feature selection when the threshold values are reasonable. To compare the correlation between them, we use the average of the percentage of overlapping results for the top $n \in [1, 30]$ retrieved formulae, which is defined as 

$$\text{Corr}_n = \frac{|R_n \cap R'_n|}{n}, \quad n = 1, 2, 3, \ldots,$$

where $R_n$ and $R'_n$ are the search results of applying feature selection or not, correspondingly. Results are presented in Figure 4.6. As expected, when the threshold values of $Freq_{\text{min}}$ and $\alpha_{\text{min}}$ increases, the correlation curves decrease. In addition, the correlation ratio increases for more retrieved results ($n$ increases). From the retrieved results, we also find that if there is an extract matched formula, usually it is returned as the first result. This is why the correlation ratio of the top retrieved formula is not much lower than that of the top two retrieved formulae. We also can see from those curves that a low threshold value of $Freq_{\text{min}}$ can keep the curve flat and have a high correlation for smaller $n$, while a low threshold value of $\alpha_{\text{min}}$ can improve the correlation for the whole curve. For the case of $Freq_{\text{min}} = 1$ and $\alpha_{\text{min}} = 0.9$, more than 80% of the retrieved results are the same for all cases, and 75% of the features are removed,
4.8.3.2 Chemical Name Search

After index construction, for similarity name search, we generate a list of 100 queries using chemical names selected randomly half from the set of indexed chemical names and half from unindexed chemical names. These formulae are used to perform similarity searches. Moreover, for substring name search, we generate a list of 100 queries using meaningful and the most frequent subterms with the length $L_s \in \{3, 4, 5, 6, 7, 8, 9, 10\}$ discovered in Section 4.8.1.

We also evaluated the response time for each query of similarity name search, illustrated in Figure 4.8. The method using hierarchical text segmentation only requires 35% of the time for similarity name search comparing with the method using all substrings. For the time of substring name search, using hierarchical text segmentation is just a little bit better than using all substrings. However, we did not test the case that the index using all substrings requires more space than the main memory. In that case, the response time would be even longer.

We also show that for the same query of similarity name search or substring name search, the search result using segmentation-based index pruning has a strong
correlation with the result before index pruning using all substrings. To compare the correlation between them, we use the same criterion as that in chemical formula search. Results are presented in Figure 4.10. We can observe from Figure 4.10 that for similarity search, when more results are retrieved, the correlation curves decrease, while for substring search, the correlation curves increase. When $Freq_{min}$ is larger, the correlation curves decrease especially for substring search. However, this does not mean that the lower $Freq_{min}$, the better. Since when $Freq_{min} = 1$, the set of discovered independent frequent substrings is just the set of the input unique terms with $IFreq = 1$ for each, no subterms are mined.

4.8.4 Entity Disambiguation in Document Search

4.8.4.1 Experiment Data and Design

To test the capability of our approach for term entity disambiguation in document search, we index 5325 PDF documents crawled from the Web. Then we select 15 common queries of chemical formulae from tagged formulae in the documents. We categorize them into three levels based on their ambiguity, 1) hard queries ($He$, $As$, $I$, $Fe$, $Cu$), 2) medium queries ($CH_4$, $H_2O$, $O_2$, $OH$, $NH_4$), and 3) easy queries ($Fe_2O_3$, $CH_3COOH$, $NaOH$, $CO_2$, $SO_2$), each with five queries. We compare the proposed approach with the traditional approach by analyzing the precision of the returned top 20 documents. The precision is defined as the percentage of returned documents really containing the query formula. Lucene [73] is applied.
Figure 4.11. Precision in Document Search using Ambiguous Formulae

as the traditional approach. We also evaluate general search engines of Google, Yahoo, and MSN using those queries to show the ambiguity of terms. Since the indexed data sets are different, they are not comparable with the results using our approach and Lucene. Their results are just used to illustrate that the ambiguity exists and the domain-specific information retrieval is desired.

4.8.4.2 Experiment results

The experiment results are shown in Figure 4.11. From Figure 4.11, we can observe that 1) the ambiguity of terms is very serious for short chemical formulae, 2) results of Google and Yahoo are more diversified than MSN, so that chemical web pages are included into the top 20 search results, and 3) our approach over performs the traditional approach based on Lucene, especially for short formulae.
In the last three chapters, we have discussed various issues and methods related to text mining and search. However, in Chemoinformatics, an important domain specific problem is how to efficiently and effectively access structured data, such as chemical molecules. The structure of chemical molecules are represented as graphs. End-users query for chemical molecules using subgraph queries. In response to a graph query, all molecules that have a graph structure that contains the query graph should be returned. To facilitate fast response to subgraph queries, the graphs representing molecules in the database need to be indexed. In this chapter, we discuss issues related to subgraph queries in graph search, and propose a subgraph feature selection method in order to find an appropriate feature set to build a graph index.

5.1 Background

Increasingly, massive complex structured data, such as chemical molecule structures [74], DNA and protein structures [75], social networks [76], citation networks [77], and XML structures [78, 79], are being stored in databases to support fast search. Efficient and effective access of the desired structure information is crucial in those areas. Search and data mining methods on structured data can help users to quickly identify a small subset of relevant data for further study and analysis.
Graphs have been used to represent structured data for a long time. Usually a typical query to search for desired graph information is to use a user-defined graph. Then, the key issue is that what kind of related graphs should be retrieved given a graph query. A typical but simple graph query is a subgraph query that searches for the set of graphs containing exactly the query graph, i.e., the support (Figure 5.1). Graph query answering has been addressed while determining chemical structure similarity [80]. The crux of the problem lies in the complexity of subgraph isomorphism. Since subgraph isomorphism is NP-complete [54], it is prohibitively expensive to scan all graphs in real time. When the number of graphs is large, instead of isomorphism tests on the fly, we need to index the graphs offline to allow for fast graph retrieval. Thus, usually graph search is decomposed into three stages [54]: 1) graph feature mining, 2) graph indexing, and 3) graph querying. First, subgraph features are extracted and selected from the graph set, each subgraph is converted into a canonical string, and then each graph is mapped into a linear space of subgraphs. Second, the graph index is constructed using the canonical strings of subgraphs. Finally, for a given subgraph query, all the indexed subgraphs of the query are extracted, and the candidate set of graphs containing all the extracted subgraphs is retrieved from the index. Subgraph isomorphism tests are performed on this candidate set to find all graphs really containing the query graph. This candidate set must be small for the graph query if we want to retrieve the graphs in real-time. To keep the candidate set small, we need to select subgraphs judiciously for indexing. This chapter addresses the issue of determining the optimal set of subgraph features to index such that the candidate set for queries is the smallest to provide the highest precision for subgraph queries. We propose a feature selection method that can find an near-optimal feature set of subgraphs to index[81].

All previous approaches [82, 83, 84, 62, 54, 85] try to find an appropriate way to discover a set of subgraph features that contain as much information as possible to answer graph queries efficiently. However, no previous work proposes any criteria to measure the information contained in a set of subgraph features. Most of them only assume frequent subgraphs are more informative than infrequent subgraphs [82, 83, 84, 62]. Some of them find frequent subgraphs independently and ignore the redundancy between subgraphs [82, 83, 84]. For example, for a set of graphs
all subgraphs $G'_i$ of a frequent subgraph $G'$ are also frequent. If $G'$ occurs every time when $G'_i$ occurs, i.e., their correlation is 100%, then $G'_i$ is redundant. Selecting $G'_i$ after having selected $G'$ cannot increase the information contained in the feature set as expected. For example, each query $G_q$ that contains $G'$ must contain $G'_i$. $G'$ and $G'_i$ have the same support, so that using them together has the same pruning power as using either one of them to remove graphs containing no $G_q$. In other words, two informative features with high redundancy contain less over-all information than two informative features with low redundancy do.

Some previous works propose heuristics to avoid selecting redundant features to some extent [62, 86] (discussed in the next section). However, all of them can only remove partial redundancy instead of as much redundancy as possible to achieve the highest precision of query answers.

All previous works select features independently or sequentially, but none of them propose any criteria to measure the information contained in a set of subgraph features. Independent methods ignore redundancy between subgraph features. Sequential methods find frequent subgraphs from the smallest to the largest size and avoiding selecting both the two features for any pairs of highly correlated supergraphs and subgraphs. They do not consider highly correlated features that are not pairs of a supergraph and a subgraph. Previous efforts [62, 86] focus on two criteria, i.e., informative principle and/or irredundant principle, although they have different heuristics to estimate information and redundancy. However, none of them is shown theoretically to have an optimal or near optimal performance.

Our goal is to improve the efficiency of subgraph queries, and the most subgraph querying method is a two-staged one as follows, first to search for graph candidates using a graph index, and then performs subgraph isomorphism tests to select only graphs containing the query graph as a subgraph. Thus, one of the key issues is to improve precision of the graph candidate set at the first stage using the indexed features. We propose a feature selection criterion, Max-Precision(MP), which can directly optimize the precision of a given query set and is the theoretically optimal measurement of the over-all information of a feature set. Then, we propose a feature selection criterion, Max-Irredundant-Information (MII), based on mutual information, which is an approximation of Max-Precision and also can measure the over-all information of a feature set. However, computing the infor-
mation of all possible subset of selected features is expensive. Thus, we propose an approximation method of MII, Max-Information-Min-Redundancy (MImR), combined with a greedy algorithm that is much more computationally efficient. Thus, our approach is different from the previous work in that: 1) we optimize the evaluation criterion of precision directly, 2) we use a probabilistic model based on mutual information to approximately optimize precision, and 3) we combine the informative and irredundant principles naturally using the probabilistic model and find an approximation method to find the near-optimal subgraph feature set to index. The proposed methods are expected to outperform previous ones, because we prove theoretically that they can achieve the optimal or near-optimal precision. Furthermore, previous work [54] only considers the occurrence of subgraphs on graphs as binary features for search. We utilize the subgraph frequencies on each graph to prune more graphs that do not contain the query graph.

5.2 Related Work

Prior work on querying graph databases fall into three categories: graph pattern mining, indexing, and search. For graph pattern mining, features should be extracted from graphs to map them into a linear space for indexing. Three types of features can be used, paths only [87], trees only [88], and subgraphs [54]. Most previous approaches index subgraphs, because paths and trees are special subgraphs that lose much structural information due to their limited representative capabilities. After features of subgraphs are extracted, canonical labeling or an isomorphism test is used to determine if two graphs are isomorphic to each other.
Canonical labeling is a system to generate a unique string for each graph, so that there is a one-to-one mapping function between graphs and canonical labeling strings. Both canonical labeling and isomorphism tests are NP-complete [54, 82]. Finally, canonical strings of subgraphs based on canonical labeling are used for graph indexing and search.

Since there are too many subgraphs to index, feature selection is required. A naive idea is to select frequent subgraphs only [61]. There are several algorithms for mining frequent subgraphs, AGM [83], FSG [82], gSpan [54], FFSM [75], Gaston [84], and others [89, 90]. Comparisons are available in [91], and an overview is available in [92]. However, the full set of frequent subgraphs is still very large, because many of the frequent subgraphs are redundant. Previous works usually use the following approaches: 1) mining closed frequent subgraphs [62] where only each subgraph that is not 100% correlated with any of its supergraphs are selected, 2) mining each frequent subgraphs that has correlations with all of its supergraphs lower than a threshold [86], 3) sequentially selecting subgraphs from the smallest size to the largest size based on subgraphs’ frequency and discrimination [54], and 4) using paths composed of small basic structures, such as cycles, crosses, and chains, instead of vertices and edges [93].

For chemical structure search, previous methods fall into four categories: 1) full structure search: search the exactly matched structure as the query graph, 2) substructure search: find structures that contain the query graphs [94, 54], 3) full structure similarity search: retrieve structures that are similar to the query graph [80], and 4) substructure similarity search: find the structures that contain a substructure that is similar to the query graph [95]. The most common and simple method among them is substructure search [96], which retrieves all molecules with the query substructure. However, sufficient knowledge to select substructures to characterize the desired molecules is required, so similarity structure search is desired to bypass the substructure selection.

Generally, a chemical similarity search is to search molecules with similar structures as the query molecule. Previous methods fall into two major categories based on different criteria to measure similarity. The first one is feature-based approaches using substructure fragments [96, 97], or paths [87]. The major challenge for this category is how to select a set of features, like subgraphs or paths, to find a trade-
Table 5.1. Notations used throughout

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G, G'$</td>
<td>graph</td>
<td>$FG$</td>
<td>candidate subgraph set</td>
</tr>
<tr>
<td>$G', G'_q$</td>
<td>subgraph</td>
<td>$S$</td>
<td>selected subgraph set</td>
</tr>
<tr>
<td>$D$</td>
<td>graph set</td>
<td>$n$</td>
<td># candidate subgraphs</td>
</tr>
<tr>
<td>$D_G$</td>
<td>support of $G$</td>
<td>$m$</td>
<td># selected subgraphs</td>
</tr>
<tr>
<td>$G_q$</td>
<td>graph query</td>
<td>$F_{G' \subseteq G}$</td>
<td>frequency of $G'$ in $G$</td>
</tr>
<tr>
<td>$Q$</td>
<td>query set</td>
<td>$s$</td>
<td>${G'_q</td>
</tr>
</tbody>
</table>

off between efficiency and effectiveness and improve both of them. The second category of approaches uses the concept of Maximum Common EDGE Subgraph (MCEG) [80] to measure the similarity by finding the size of the MCEG of two graphs. The feature-based approaches are more efficient than the MCEG method, because finding MCEG is expensive and needs to be computed between the query graph and every graph in the collection. Thus, feature-based approaches based on substructure fragments are used to screen candidates before finding MCEG [97]. Hierarchical screening filters are introduced in [80], where at each level, the screening method is more expensive but more accurate.

Previous works on index pruning for information retrieval usually prune postings of irrelevant terms in each document [56, 57, 58]. Criteria in information theory are applied to measure the information of each term in each document. However, most previous works focus on selecting informative terms without considering redundancy [56, 57]. Moura, et al., consider local information of phrases to keep consistent postings of correlated terms, instead of global information for feature selection [58]. Peng, et al., propose feature selection focusing on supervised learning. The generic goal is to find the optimal subset from the set of feature candidates so that selected features are 1) correlated to the positive and negative class distribution, and 2) uncorrelated to each other [98]. We extend the idea of feature selection to graph search.
5.3 Problem Formalization

In this section, we first give preliminary notations, and then give an overview of how a subgraph query is processed. Table 5.1 lists the notations used throughout this chapter.

5.3.1 Preliminaries

We consider only connected labeled undirected (sub)graphs. Relevant notations are given as follows:

Definition 5.3.1. Labeled Undirected Graph: A labeled undirected graph is a 5-tuple, \( G = \{V, E, L_V, L_E, l\} \), where \( V \) is a set of vertices. Each \( v \in V \) is a unique ID representing a vertex, \( E \subseteq V \times V \) is a set of edges where \( e = (u, v) \in E, u, v \in V \), \( L_V \) is a set of vertex labels, \( L_E \) is a set of edge labels, and \( l : V \cup E \rightarrow L_V \cup L_E \) is a function assigning labels to vertices and edges on the graph.

Definition 5.3.2. Connected Graph: A path \( p(v_1, v_{n+1}) = (e_1, e_2, ..., e_n), e_1 = (v_1, v_2), e_2 = (v_2, v_3), ..., e_n = (v_n, v_{n+1}), e_i \in E_G, i = 1, 2, ..., n, \) on a graph \( G \) is a sequence of edges connecting two vertices \( v_1 \in V_G, v_{n+1} \in V_G \). A graph \( G \) is connected iff \( \forall u, v \in V_G, \) a path \( p(u, v) \) always exists. The size of a graph \( \text{Size}(G) \) is the number of edges in \( G \).

Definition 5.3.3. Subgraph and Connected Subgraph: A subgraph \( G' \) of a graph \( G \) is a graph, i.e., \( G' \subseteq G \), where \( V_{G'} \subseteq V_G \), and \( E_{G'} \subseteq E_G \) where \( \forall e = (v_1, v_2) \in E_{G'}, \) the two vertices \( v_1, v_2 \in V_{G'} \). \( G \) is the supergraph of \( G' \), or we say \( G \) contains \( G' \). A subgraph \( G' \) of a graph \( G \) is a connected subgraph if and only if it is a connected graph.

Note that if we change the IDs of vertices on a graph, the graph still keeps the same structure. Thus, a graph isomorphism test is required to identify whether two graphs are isomorphic (i.e., the same) to each other [54]. Another method to achieve the function of graph isomorphism tests is to use canonical labels of graphs [54]. Usually if there is a method to sequentialize all isomorphic graphs of the same graph into different strings, then the minimum or maximum string is the
canonical labeling. Two graphs are isomorphic to each other, if and only if their strings of canonical labeling are the same. Thus, strings of canonical labeling of subgraph features can be used to index graphs for fast search. We provide two definitions below:

**Definition 5.3.4. Graph Isomorphism and Subgraph Isomorphism:** A graph isomorphism between two graphs $G$ and $G'$, is a bijective function $f : V_G \rightarrow V_{G'}$ that maps each vertex $v \in V_G$ to a vertex $v' \in V_{G'}$, i.e., $v' = f(v)$, such that $\forall v \in V_G, l_G(v) = l_{G'}(f(v))$, and $\forall e = (u, v) \in E_G, (f(u), f(v)) \in E_{G'}$ and $l_G(u, v) = l_{G'}((f(u), f(v))$. Since it is a bijective function, a bijective function $f' : V_{G'} \rightarrow V_G$ also exists. A subgraph isomorphism between two graphs $G'$ and $G$ is the graph isomorphism between two graphs $G'$ and $G''$, where $G''$ is a subgraph of $G$.

**Definition 5.3.5. Canonical labeling:** A canonical labeling $CL(G)$ is a unique string to represent a graph $G$, where given two graphs $G$ and $G'$, $G$ is isomorphic to $G'$ iff $CL(G) = CL(G')$.

As mentioned before, both isomorphism tests and canonical labeling can be used to determine if two graphs are isomorphic. Usually, the strings generated by canonical labelings of selected subgraph features are indexed for graph search.

### 5.3.2 Answering Subgraph Queries

In this section, we first provide some definitions, and then introduce an algorithm to answer subgraph queries.

**Definition 5.3.6. Support, Support Graph, and Subgraph Query:** Given a data set $D$ of graphs $G$, the support of subgraph $G'$, $D_{G'}$, is the set of all graphs $G$ in $D$ that contain $G'$, i.e., $D_{G'} = \{G | \forall G \in D, G' \subseteq G\}$. Each graph in $D_{G'}$ is a support graph of $G'$. $|D_G|$ is the number of support graphs in $D_G$. A subgraph query $G_q$ seeks to find the support of $G_q$, $D_{G_q}$.

Like words are indexed to support document search, subgraphs are indexed to support graph search. Note that subgraphs may overlap with each other. We define subgraph frequency as follows,
Algorithm 6: Graph Search of Subgraph Query

**Algorithm: GSSQ(G_q, S, Index_D):**

**Input:** Query Subgraph G_q, indexed subgraph set S, and index of the graph set D, Index_D.

**Output:** Support of G_q, D_{G_q}.

1. if G_q is indexed, find D_{G_q} using Index_D; **return** D_{G_q};
2. D_{G_q} = \{\emptyset\};
   find all subgraphs of G_q, G'_q \in S with F_{G'_q \subseteq G_q};
3. if no G'_q found, D_{G_q} = D;
4. **else for all** G'_q **do**
5.   Find D_{G_q \geq G'_q} (Table 5.1) using Index_D,
   then D_{G_q} = D_{G_q} \cap D_{G'_q};
6. **for all** G \in D_{G_q} **do**
7.   if subgraphIsomorphism(G_q, G)==false, remove G;
8. **return** D_{G_q};

**Definition 5.3.7. Embedding and Subgraph Frequency:** An embedding of a subgraph G' in a graph G, i.e., Emb_{G' \subseteq G}, is an instance of G' \subseteq G. The frequency of a subgraph G' in a graph G, i.e., F_{G' \subseteq G}, is the number of embeddings of G' in G. Embeddings may overlap.

Algorithm 6 shows how a subgraph query can be answered. First, if the query graph G_q is indexed, its support is returned directly. Otherwise, it identifies all indexed subgraphs G'_q with their corresponding frequency F_{G'_q \subseteq G_q} of G_q, then finds graphs where each graph G satisfies F_{G'_{q} \subseteq G} \geq F_{G'_q \subseteq G_q}, and finally performs subgraph isomorphism tests on each graph to identify graphs containing G_q. The recall of each subgraph query G_q is always 100%, because for each selected subgraph G'_q \subseteq G_q, each support graph G of G_q must be a support graph of each G'_q and satisfy F_{G'_{q} \subseteq G} \geq F_{G'_q \subseteq G_q}. Note that besides using features of subgraph frequencies, we also can use binary features to represent if a subgraph feature occurs in a graph or not. In this case, F_{G'_q \subseteq G} = \{0, 1\}. This is because each occurrence of G'_q in G_q also occurs in each support graph of G_q, G. Thus, the intersection of the support of each G'_q with F_{G'_q \subseteq G} \geq F_{G'_q \subseteq G_q} must contain each support graph of G_q. As mentioned before, how to extract and select subgraph features for graph indexing and querying is the key issue. A subgraph G' is frequent if the size of its support |D_{G'}| \geq F_{min}, a minimum frequency threshold.
Algorithm 7 Independent Frequent Subgraph Mining

Algorithm: IFGM\((D,F_{\text{min}},F_{\text{max}},\text{Corr}_{\text{max}})\):

**Input:** Set of graphs \(D\),
minimal and maximal threshold of frequency \(F_{\text{min}}\) and \(F_{\text{max}}\),
maximal threshold of correlation \(\text{Corr}_{\text{max}}\).

**Output:** Set of Independent Frequent Subgraphs \(FG\), each subgraph has a list of support graphs with corresponding frequencies.

1. **Initialization:** \(FG = \emptyset\), and
   find frequent vertex set \(FV = \{v | F_{\text{min}} \leq F_v \leq F_{\text{max}}\}\).
2. **for all** \(v \in FV\) **do**
3.   Find the set of all one-edge extensions of \(v, L\);
4.   **searchSubgraph**\((v, path, L)\);
5.   **return** \(FG\);

**Subprocedure:** **searchSubgraph**\((G, T, L)\):

**Input:** A graph \(G\), its type \(T \in \{\text{path, tree, cyclic}\}\), and its extension set \(L\).

**Output:** \(FG\).

1. \(\text{Dep}_G = false\);
2. **for all** \(l \in L\) **do**
3.   \(G' = G + l\) and find \(T'\);
4.   Find the frequency of \(G'\) in \(D\), \(F_{G'}\);
5.   **if** \(\text{Corr}(G, G') \geq \text{Corr}_{\text{max}}\), **then** \(\text{Dep}_G = true\);
6. **if** \(\text{Dep}_G == false\), **then** put \(G\) into \(FG\);
7. **for all** \(l \in L'\) **do**
8.   **if** \(F_{\text{min}} \leq F_{G'} \leq F_{\text{max}}\)
9.   Find the set of all one-edge extensions of \(G', L'\);
10. **searchSubgraph**\((G', T', L')\);

5.4 Subgraph Mining

In this section, we describe the algorithm to mine *independent* frequent subgraphs from a set of graphs. Then we introduce three feature selection criteria, MP, MII, and MIMR, and finally we propose a greedy algorithm to select subgraph features from the discovered independent frequent subgraphs for graph indexing and search.

5.4.1 Independent Frequent Subgraph Mining

Our proposed independent frequent subgraph mining algorithm is extended from previous work [84]. Neither infrequent nor very frequent subgraphs are informative, because infrequent subgraphs usually are large subgraphs appearing infrequently
and very frequent subgraphs are just like stop words appearing everywhere. Thus, we define a lower bound and an upper bound of frequencies for frequent subgraph mining, and we remove subgraphs that have any highly correlated supergraphs.

Thus, rather than only identifying the support graphs of each subgraph, we also find the subgraph frequencies on each support graph to compute correlation of two subgraphs $G_i', G'$ where $G_i' \subseteq G'$, by using the correlation of two random variables $G_i'$ and $G'$, i.e.,

$$Corr(G_i', G') = \frac{Cov(G_i', G')}{SD(G_i')SD(G')}$$  \hspace{1cm} (5.1)

where $Cov(G_i', G')$ is the covariance of $G_i'$ and $G'$, $SD(G')$ is the standard deviation of $G'$. The random variable $G_i'$ (similar to the random variable $G_i$) is a variable representing which graph $G$ the subgraph $G'$ occurs in. Its probability distribution $p(G' = G)$ represents how likely $G'$ occurs in $G$. It is estimated using $p(G' = G) = F_{G' \subseteq G}/ \sum_{G_i \in D} F_{G' \subseteq G_i}$. Note that we use the correlation utilizing subgraph frequencies on support graphs, which is different from that used in previous work [62, 85], which only uses the number of support graphs. Our proposed approach can measure the correlation more accurately. For example, if two subgraphs $G_i' \subseteq G'$ always appear on the same graphs $G$, previous methods only select $G'$. However, if $F_{G_i' \subseteq G} \gg F_{G_i' \subseteq G'} \times F_{G' \subseteq G}$, i.e., $G_i'$ has more embeddings on $G$ other than the embeddings of $G_i'$ on all the embeddings of $G'$, $G_i'$ is still useful to prune irrelevant graphs in addition with $G'$.

### 5.4.2 Irredundant Informative Subgraph Selection

The independent frequent subgraphs discovered by the Algorithm 7 can be used for graph indexing. However, correlation in Equation (5.1) is used only to pre-filter highly correlated subgraphs. Partial redundancies between subgraphs still exist. Thus, we propose a feature selection approach for further index pruning.

We have a matrix of subgraph frequencies in all graphs. Each subgraph feature $G'$ has a list of support graphs $G$ with a frequency $F_{G' \subseteq G}$, and correspondingly a graph has a list of subgraph features $G'$ with $F_{G' \subseteq G}$. Then we can have the joint probability distribution $P(G, F)$, where $G$ is a random variable with outcomes of all the graphs $G \in D$, $F$ is a random variable with outcomes of all the subgraph features $G' \in FG$. This joint distribution is computed using $p(G, G') = F_{G' \subseteq G}/Z$, where $Z = \sum_{G' \in FG} F_{G' \subseteq G}$.
where $Z$ is a constant to normalize all subgraph frequencies into probabilities.

**Max-Precision**

As mentioned before, our search scheme can guarantee 100% recall for subgraph queries. Thus, the goal of feature selection is to optimize the precision of relevant graphs among all the retrieved candidates for all queries. Thus, given the possible user generated graph query set $Q$, we can find the support graphs of each query $G_q \in Q$, where each support graph is considered as relevant to $G_q$. Since the possible user generated graph query set is hard to obtain without user logs, we use a pseudo graph query set $Q$ for feature selection that is generated randomly from the set of all the discovered subgraphs $FG$. Thus, the Max-Precision (MP) problem to select the optimal subgraph set $S_{opt}$ is defined as follows:

$$S_{opt} = \arg \max_S \text{Prec}(S), \text{ where Prec}(S)$$

$$= \frac{1}{|Q|} \sum_{G_q \in Q} \frac{|D_{G_q}|}{\prod_{G_q' \subseteq G|G_q'} |D_{G_q'}|}$$

$$= \frac{1}{|Q|} \sum_{G_q \in Q} p(G_q \subseteq G|\forall G_q') = \frac{1}{|Q|} \sum_{G_q \in Q} \frac{p(G_q \subseteq G, \forall G_q')}{p(\forall G_q')}$$

$$\approx \left[ \prod_{G_q \in Q} \frac{p(G_q \subseteq G)}{p(\forall G_q')} \right]^{1/|Q|}, \quad (5.2)$$

where $S = \{G_1', G_2', ..., G_m'\}$, $D_{G_q' \subseteq G|G_q'} \subseteq G_q$ is the set of support graphs of $G_q'$ where $\forall G, F_{G_q' \subseteq G} \geq F_{G_q \subseteq G}$ (note that $D_{G_q \subseteq G} = \varnothing$), and in this chapter, we let

$$\forall G_q' = \forall G_q', G_q' \subseteq G_q \land G_q' \in S, F_{G_q' \subseteq G} \geq F_{G_q \subseteq G}.$$

Then $p(G_q \subseteq G|\forall G_q')$ is the conditional probability that a graph $G$ contains $G_q$ given $F_{G_q' \subseteq G} \geq F_{G_q \subseteq G}$ for all $G_q'$ such that $G_q' \in G_q, G_q' \in S$. The last term in Equation (5.2) uses the geometric mean to approximate the arithmetic mean.

However, even though we can have the possible user generated graph query set $Q$ with a probability distribution of each query, finding the optimal feature set $S$ that maximizes Prec$(S)$ is computationally expensive, since for each possible subset of features we have to compute Prec$(S)$. Even though greedy algorithms
can be used, it is still expensive (shown in Section 5.4.3 and Section 5.5.5). We desire a more time-efficient algorithm. We show below how to use an approximation method to select features.

**Max-Irredundant-Information**

As mentioned before, we need to combine the informative and irredundant principles together. In order to do so, we propose a mutual-information-based strategy as follows. The mutual information (MI) $MI(X; Y)$ is a quantity to measure the amount of information that is contained in two or more random variables [99, 33]. For the case of two random variables, we have

$$MI(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}.$$  

Obviously, when random variables $X$ and $Y$ are independent, $I(X; Y) = 0$. Thus, intuitively, the value of MI depends on how random variables are dependent on each other. The pointwise mutual information (PMI) of a pair of outcomes of discrete random variables is used to measure their dependence. In our case, the pair of outcomes are two different subgraphs $G$ and $G'$, where $G' \subset G$. Mathematically,

$$PMI(x; y) = \log \frac{p(x, y)}{p(x)p(y)}.$$  

we use PMI to measure the dependence of a subgraph feature $G_q'$ and a query graph $G_q$ in the set of retrieved graphs $G$, or dependence of a pair of subgraphs, $G_i'$ and $G_j'$. We call this scheme the Max-Irredundant-Information (MII) that has the following form:

$$S_{opt} = \arg \max_S (IrreduInfo(S)),$$

where $IrreduInfo(S) = \sum_{G_q \in Q} \log \frac{p(G_q \subseteq G, \forall G_q')}{p(G_q \subseteq G)p(\forall G_q')} = -\sum_{G_q \in Q} \log p(\forall G_q'). \quad (5.3)$

**Theorem 1.** MII is equivalent to MP with geometric mean.

**Proof.** For MP using the geometric mean,

$$S_{opt} = \arg \max_S (Prec(S)) = \arg \max_S (\log (Prec(S)))$$
\[
= \arg \max_S \sum_{G_q \in Q} [\log p(G_q \subseteq G) - \log p(\forall G'_q)].
\]
\[
= \arg \max_S \left[ - \sum_{G_q \in Q} \log p(\forall G'_q) \right],
\]
which is the same as MII. Thus, as shown above, MII is equivalent to MP with geometric mean.

Both MP and MII are computationally expensive, because for each possible subset of features we have to compute \(\text{IrreduInfo}(S)\). MP and MII both combine the informative and irredundant principles naturally, where selected features should be informative (have a high pruning power, i.e., \(|D|/|D_{G_q' \supseteq F_{G_q' \subseteq G_q}}|\)), but also each pair of features should be irredundant of each other. Thus, we decompose this problem into two subproblems: Max-Information and Min-Redundancy, with the goal to reduce the time complexity.

**Max-Information**

If all \(G' \in S\) are independent to each other, then we have

\[
\text{IrreduInfo}(S) = - \sum_{G_q \in Q} \sum_{G'_q \subseteq G_q} \log p(G'_q).
\]

Obviously, \(\text{IrreduInfo}(S)\) is the sum of each subgraph’s pointwise contribution. We propose a pointwise-pruning-power-based criterion to measure the information contained by each subgraph as follows

\[
\text{Info}(G'_q) = \sum_{G_q \in Q \land G'_q \subseteq G_q} \log \frac{p(G_q \subseteq G, G'_q)}{p(G_q \subseteq G)p(G'_q)}
\]

\[
= - \sum_{G_q \in Q \land G'_q \subseteq G_q} \log p(G'_q) = \sum_{G_q \in Q \land G'_q \subseteq G_q} \log \frac{|D|}{|D_{G'_q \supseteq F_{G'_q \subseteq G_q}}|},
\]

where we use \(p(G'_q)\) to represent \(p(G'_q, F_{G'_q \subseteq G} \supseteq F_{G_q' \subseteq G_q})\), and \(D_{G'_q \supseteq F_{G'_q \subseteq G_q}}\) is the support graph set where each support graph \(G\) has at least \(F_{G_q' \subseteq G_q}\) embeddings of \(G'_q\), i.e., \(F_{G'_q \subseteq G} \supseteq F_{G'_q \subseteq G_q}\). Thus, the first goal of feature selection is to find a subset of features \(S\) with \(m\) features \(G_i'\) that maximizes the sum of information
scores of each $G'_i$, called Max-Information (MI), is defined as follows

$$\max_S Info(S), \text{ where } Info(S) = \frac{1}{m} \sum_{G'_i \in S} Info(G'_i).$$

**Min-Redundancy**

Using PMI, we can define the dependence of a pair of subgraphs $G'_i$ and $G'_j$. Two features $G'_i$ and $G'_j$ are **positively dependent** if $p(G'_i|G'_j) > p(G'_j)$, **negatively dependent** if $p(G'_i|G'_j) < p(G'_j)$, and **independent** otherwise. They are **irrelevant** if they are negatively dependent or independent. Thus, we define redundancy of two features as follows,

$$Re{du}(G'_i; G'_j) = \sum_{G_q \in Q \wedge G'_i \subseteq G_q \wedge G'_j \subseteq G_q} \log \frac{p(G'_i, G'_j)}{p(G'_i)p(G'_j)}.$$ 

If two features are irredundant, i.e., have a low redundancy score, their integration is more informative than that of two redundant features. Thus, another goal of feature selection is to find a subset of features $S$ with $m$ features $G'_i$, which minimizes the redundancy of selected features, i.e., the sum of mutual information of each pair $G'_i$ and $G'_j$, called Min-Redundancy, defined as follows:

$$\min_S Redu(S), \text{ where } Redu(S) = \frac{2}{m(m-1)} \sum_{G'_i, G'_j \in S, i \neq j} Redu(G'_i, G'_j).$$

**Max-Information-Min-Redundancy**

We need to obtain Max-Information and Min-Redundancy in the selected feature set, but considering all the selected features together as in MP or MII is computationally expensive. Thus, we propose a global criterion that combines the two constraints, Max-Information and Min-Redundancy, and is significantly more efficient computationally, called Max-Information-Min-Redundancy (MImR), as follows

$$S_{opt} = \arg \max_S (Info(S) - Redu(S)). \quad (5.4)$$

In practice, usually feature selection algorithms using first-order incremental search can be used to find the near-optimal feature set. Suppose we have selected $k - 1$
features, then next step is to select the kth one from the rest of the features. Then
the local optimal feature $G_k'$ is selected to maximize the following function:

$$MP : \max_{G'} (Prec(S_k)),$$
$$MII : \max_{G'} (IrreduInfo(S_k)),$$
$$MImR : \max_{G'} (Info(S_k) - Redu(S_k))$$

$$= \max_{G'} (Info(G') - \frac{2}{k-1} \sum_{G_i \in S_{k-1}} Redu(G', G_i')).$$  \hspace{1cm} (5.5)

Now we show that for the first-order incremental search, MImR is an approximation
to MII. First we define pointwise incremental search, MImR is an approximation
to MII. First we define pointwise entropy $PH(x) = -\log p(x)$ and joint pointwise
entropy as $PH(x, y) = -\log p(x, y)$. It is easy to verify that

$$IrreduInfo(S) = \sum_{G_q \in Q} [PH(G_q \subseteq G) + PH(\forall G_q')$$
$$- PH(G_q \subseteq G, \forall G_q')]$$ \hspace{1cm} (5.6)

We define pointwise total correlation $PC(S)$ as follows

$$PC(S) = \sum_{G_q \in Q} \log \frac{p(\forall G_q')}{\prod_{G_q' \in G_q} p(G_q')}$$

$$= \sum_{G_q \in Q} [\sum_{G_q'} PH(G_q') - PH(\forall G_q')]$$ \hspace{1cm} (5.7)

and $PC(S, Q)$ as follows:

$$PC(S, Q) = \sum_{G_q \in Q} \log \frac{p(G_q \subseteq G, \forall G_q')}{p(G_q \subseteq G) \prod_{G_q'} p(G_q')} =$$

$$\sum_{G_q \in Q} [PH(G_q \subseteq G) + \sum_{G_q'} PH(G_q') - PH(G_q \subseteq G, \forall G_q')]$$ \hspace{1cm} (5.8)

Then by subtracting (5.7) from (5.8) and substituting the difference into (5.6) we
have

\[ \text{IrreduInfo}(S) = PC(S,Q) - PC(S). \]  

(5.9)

Thus, MII is equivalent to simultaneously maximizing the first term and minimizing the second term at the left hand side of Equation (5.9).

It is easy to show that for the first term,

\[
PC(S,Q) = \sum_{G_q \in Q} \left[ \sum_{G_q'} PH(G_q') - PH(\forall G_q' \mid G_q \subseteq G) \right] 
\]

\[
\leq \sum_{G_q \in Q} \sum_{G_q'} PH(G_q'),
\]

which is maximized only if all the variables in \( \{S, Q\} \) are the most dependent. Thus, if all \( m-1 \) features in \( S \) have been selected, the \( m \)th feature that is the most dependent on \( Q \) should be selected for MII, because it can maximize \( PC(S,Q) \).

Note that this is the same as the strategy of Max-Info. The second term \( PC(S) > 0 \) if features are positively totally dependent, \(< 0 \) if negatively totally dependent, and \( = 0 \) if all the features are independent. Thus, if all \( m-1 \) features in \( S \) has been selected, the \( m \)th feature that is the most pairwise negatively dependent on each selected features in \( S \) should be selected for MII, which can minimize \( PC(S) \).

This is same as the strategy of Min-Redu. Therefore, as a combination of Max-Info and Min-Redu, MImR is an approximation to MII.

### 5.4.3 Subgraph Selection Algorithm

As mentioned before, finding the optimal solution for the MP or MII problem is as expensive as

\[
O(\frac{n!}{m!(n-m)!}|Q| \cdot \text{avg}|s| \cdot (|D| + \text{avg}|D_{G_q \in s, \geq F_{G_q \subseteq G}}|)),
\]

where \( s = \{G_q' \mid G_q' \in S, G_q' \subseteq G_q \in Q\} \) is the set of possible subgraphs in \( G_q \) and \( \text{avg}|s| \) is the average size of \( s \).

We use forward selection in this work (Algorithm 8). Forward selection is a greedy algorithm using first-order incremental search, i.e., every time only the best feature from the rest of the candidate feature set is added to the selected feature
set. Initially, the algorithm finds the most informative feature. Then, every time
when a new feature is added, the rest of the features in the candidate set are
scanned and evaluated. The one that maximizes Equation (5.5) is selected. The
algorithm repeats this until \( m \) features are selected.

For MP and MII, computational complexity of the first-order incremental se-
lection is

\[
O(n^2|Q| \cdot \text{avg}|s| \cdot (|D| + \text{avg}|D_{G_q^s} \in E \geq F_{G_q^s} \subseteq G_q|)).
\]

For MImR, the computational complexity of the first-order incremental selec-
tion involves three parts:

1) for pre-computing information scores of all features:

\[
O(|Q| \cdot \text{avg}|s| \cdot |D|)
\]

2) for pre-computing pairwise dependence scores of all feature pairs:

\[
O(|Q| \cdot (\text{avg}|s| \cdot |D| + \text{avg}|s|^2 \cdot \text{avg}|D_{G_q^s} \in E \geq F_{G_q^s} \subseteq G_q|))
\]

and 3) for feature selection:

\[
O(n^2),
\]

which is much faster than that of MP or MII. Because \( \text{avg}(|s|) \) is a constant com-
pared with other numbers, forward feature selection based on MImR is quadratic,
while for MP or MII it is quartic. Another advantage of the first-order incremen-
tal search is that we only need to run the algorithm once to select \( m \) features, and
then we know what are the best \( k \leq m \) features instead of computing again, and
we avoid re-running the algorithm every time when the number of selected features
is changed.

5.5 Experimental Evaluation

In this section, we evaluate the proposed approaches and compare the experimental
results with two previous methods. Experimental results show that:
Algorithm 8 Irredundant Informative Subgraph Selection

Algorithm: IIGS\((FG, m)\):

**Input:**
Candidate set of subgraphs \(FG\), and
Number of features to select \(m\).

**Output:**
Set of Irredundant Informative Subgraphs \(S\).

1. **Initialization:** \(S = \emptyset\), \(k = 1\).
2. **while** \(k \geq m\), **do**
3. scan all \(G' \in FG\) and find \(G_{opt} = \arg \max_{G'} Equation(5.5)\);
4. move \(G_{opt}\) from \(FG\) to \(S\);
5. \(k++\);
7. **return** \(IIFG\);

- The proposed feature selection and graph search method can improve the precision of return candidates in comparison to previous methods by about 4%-13%.
- The algorithm based on MImR has a much lower computational cost than those based on MP and MII, and MImR can achieve a reasonably high precision that is only slightly lower than that of MP or MII.

Response time of a subgraph query involves *search time* to retrieve candidates and *verification time* to find support graphs of the query from the candidates. Search time is similar using different feature selection methods. However, verification time is proportional to the number of returned candidates. Thus, if the precision of returned candidates is higher, then verification time and response time is shorter.

### 5.5.1 Experimental Data Set

We use the same real data set and testing query set as those used by Yan, et al., [54]. It is a NCI/NIH HIV/AIDS antiviral screen data set that contains 43905 chemical structures. The experimental subset contains 10000 chemical structures randomly selected from the whole set and the query set contains 6000 randomly generated queries, i.e., 1000 queries per \(Size(G_q) = \{4, 8, 12, 16, 20, 24\}\). Although we only use chemical structures for experiments, the proposed approach is applicable to any structures that can be represented by graphs, such as DNA sequences and
5.5.2 Evaluated Feature Selection Methods

We evaluate the average precision of returned chemical structures for all queries using different feature selection methods. When we compute the average precision of returned structures, we only count queries with non-empty supports in the data set. In our experiment, we evaluate six methods. First, we evaluate three methods without considering subgraph frequencies, including two previous methods, *CloseG* and *GIndex* [62, 54], and our proposed method, *MImR*. They use binary features that only consider the occurrence of subgraphs in graphs. Each subgraph feature takes a binary value of 1 or 0 to represent the occurrence of a subgraph in a graph or not, respectively. Then we evaluate three more methods by considering subgraph frequencies, *CloseG.F*, *GIndex.F*, and *MImR.F*, which are
extensions of CloseG, GIndex, and MImR, respectively. They use numerical features of subgraph frequencies in graphs. CloseG and CloseG.F [62] just select frequent closed subgraphs independently that have $D_G \geq F_{\text{min}} = \{1000, 500, 200, 100\}$ ($m = \{460, 1795, 9846, 50625\}$ respectively) without considering redundancy. GIndex and GIndex.F [54] select subgraphs from the candidate subgraph set where $D_G \geq 100$. It scans subgraphs from the smallest to the largest avoiding to select redundant supergraphs of selected subgraphs. A discriminative score is defined and computed for each subgraph candidate to measure the redundancy. If its score is large than a threshold $D_{\text{min}} = \{7.0, 3.1, 1.09, 1\}$ ($m = \{667, 1779, 9855, 50625\}$ respectively) then the subgraph is selected. MImR.F is our proposed method in Section 4.2 and MImR is similar but uses binary features. Five variations on the number of selected features are evaluated, $m = \{460, 667, 1779, 9855, 50625\}$. MP and MII are only evaluated using the data set with 100 structures because they are forbiddingly expensive in practice. We randomly sample 30,000 subgraphs as the training query set with the same distributions as that of the testing query set, and then use the training query set for feature selection in MImR and MImR.F.

### 5.5.3 Precision of Returned Results

We show precision of returned results for the testing query set in Figure 5.2. Because each feature selection method can select different numbers of features for indexing by adjusting parameters, we show the curves of precision versus different values of the selected feature number $m$ and the index size in Figure 5.2. We also present the precision curves versus the query graph size $|G_q|$ to illustrate the effect of different query sizes in Figure 5.2. To evaluate the precision versus the index size, we first index the canonical string of each selected subgraph. If a subgraph has a frequency larger than one, we then index the frequency. Thus, using numerical features has a larger index size than using binary features given the same number of selected feature.

In Figure 5.2 (a), we can observe that given the same number of selected features, GIndex improves the average precision compared with CloseG. Our proposed approach MImR can improve the precision by about 4%-13%. This illustrates that our probabilistic model for feature selection works better than the previ-
Table 5.2. Average precision for feature selection methods

<table>
<thead>
<tr>
<th>Method</th>
<th>#Feature</th>
<th>Index size</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseG</td>
<td>1795</td>
<td>3.44MB</td>
<td>61.10%</td>
</tr>
<tr>
<td>CloseG.F</td>
<td>1795</td>
<td>4.14MB</td>
<td>69.15%</td>
</tr>
<tr>
<td>GIndex</td>
<td>1779</td>
<td>2.07MB</td>
<td>66.32%</td>
</tr>
<tr>
<td>GIndex.F</td>
<td>1779</td>
<td>2.18MB</td>
<td>71.47%</td>
</tr>
<tr>
<td>MImR</td>
<td>1779</td>
<td>2.50MB</td>
<td>74.11%</td>
</tr>
<tr>
<td>MImR.F</td>
<td>1779</td>
<td>2.73MB</td>
<td>80.03%</td>
</tr>
</tbody>
</table>

Table 5.3. One-sided T-test for feature selection methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIndex vs. CloseG</td>
<td>0.000</td>
</tr>
<tr>
<td>MImR vs. GIndex</td>
<td>0.000</td>
</tr>
<tr>
<td>GIndex.F vs. CloseG.F</td>
<td>0.001</td>
</tr>
<tr>
<td>MImR.F vs. GIndex.F</td>
<td>0.000</td>
</tr>
<tr>
<td>CloseG.F vs. CloseG</td>
<td>0.000</td>
</tr>
<tr>
<td>GIndex.F vs. GIndex</td>
<td>0.000</td>
</tr>
<tr>
<td>MImR.F vs. MImR</td>
<td>0.000</td>
</tr>
</tbody>
</table>

A previous method proposed by Yan, et al., [54]. We also can see that CloseG.F, GIndex.F, and MImR.F can improve the average precision compared with CloseG, GIndex, and MImR by using subgraph frequencies as features (Figure 5.2 (a)). This demonstrates that using subgraph frequencies for subgraph queries can improve the precision by about 4%-12%. In Figure 5.2 (b), we can see that using subgraph frequencies as features increases the index size by about 6%-30%, since more information of frequencies are indexed. We can see that given the same index size, CloseG.F, GIndex.F, and MImR.F also have higher precision than CloseG, GIndex, and MImR, even though not as much as that in the cases in Figure 5.2 (a). In Figure 5.2 (c), the curves are of the precisions vs. query sizes for the cases in Table 5.2. We can observe that CloseG and CloseG.F have higher precision for small queries, while GIndex and GIndex.F have higher precision for large queries. MImR and MImR.F are more balanced than either and always have precisions above their precisions. The p-values in Table 5.3 of one-sided T-tests show that the improvement for each pair of methods in Table 5.3 is significant with a confidence level of at least 99.9%.
Note: For combination of Methods, Features, and Query Sizes, M=MImR, G=GIndex, C=CloseG, MF=MImR.F, GF=GIndex.F, CF=CloseG.F, 4=Query size of 4. For example, M4 is MImR using binary features for queries with size of 4, and CF12 is CloseG.F using frequency features for queries with size of 12.

Figure 5.3. Response time of subgraph queries for cases in Table 5.2

5.5.4 Response Time of Subgraph Queries

In the last section, we show the precision of retrieved candidates. In this section, we show the overall response time for subgraph queries using different feature selection methods. The search process to answer subgraph queries works as follows. Every time when a subgraph query is entered, the algorithm first generated the canonical string of the query graph and check if this query graph is indexed. If
the query is indexed, the support of the query is retrieved without verification using subgraph isomorphism. If the query is not indexed, its subgraph features are enumerated, and among those subgraphs, indexed ones are used to scan the index and find candidate sets for each subgraph feature and compute the intersection for all candidate sets. Finally, subgraph isomorphism tests are performed on all retrieved candidates to prune out unsatisfied graphs.

Thus, response time of a subgraph query involves 1) **enumeration time** that is to identify if the query graph is indexed or not, and enumerate the subgraph features of the query if the query is not indexed (note that enumeration time also includes the time of finding canonical strings of subgraph features), 2) **search time** that is to retrieve candidates from the graph index, and 3) **verification time** to prune out unsatisfied graphs that are not supergraphs of the query from the candidates using subgraph isomorphism tests. Enumeration time first depends on how many queries are indexed and then depends on query sizes. Usually large queries have long enumeration time. Search time includes time to retrieve candidate sets, and time to compute the set intersection. If the query is indexed, then no set intersection is computed. Moreover, if the query is indexed, verification time is zero. Otherwise, it depends on the number of returned candidates. Thus, if the precision of returned candidates is higher, then verification time is shorter.

We illustrate average response times for the cases in Table 5.2, in Figure 5.3. We set the max enumerated subgraph size of queries as 8. From Figure 5.3, we can
observe that the average over-all response time of subgraph queries using MImR is always the best in comparison with the other two methods, GIndex and CloseG, for different query sizes, irrespective of whether binary features or frequency features are used in the index and the search process. However, the improvement of response time is significant for small queries, while for large queries, it is not significant, because the enumeration time dominates the over-all response time. We also can observe that search time is always a small part of the over-all response time, while enumeration time and verification time change much for different cases and affect the over-all response time significantly. Usually for small queries, the major part of response time is the verification time, while for large queries, the major part is enumeration time. This is because small queries usually have large supports containing more supergraphs so that as a result the candidate set is also very large to verify, although each subgraph isomorphism test for small queries is not expensive. In comparison, large queries usually have small candidates to verify, but the enumeration time is expensive for large queries, because it increases exponentially when the query size increases.

Similar to precision curves in Figure 5.2 (c), we can observe that 1) for GIndex and GIndex.F, the verification time is long for small queries but short for large queries, 2) for CloseG and CloseG.F, the verification time is short for small queries but long for large queries, 3) for MImR and MImR.F, the verification time is short for all queries. Moreover, using frequency features can achieve a shorter verification time than using binary features for all the cases. This is consistent with the precision curves in Figure 5.2 (c). Although using frequency features can achieve a shorter verification time than using binary features, it requires a longer enumeration time because the exact occurrence number of each subgraph features is needed to be identified for frequency features. Thus, the over-all response time is shorter using binary features than using frequency features for large queries, because the enumeration time dominates the response time for large queries. However, the over-all response time is shorter using frequency features than binary features for small queries, because the verification time contributes more than the enumeration time to the response time.

Because the subgraph enumeration process uses Algorithm 8, we can set the maximum subgraph size to enumerate. If we set a smaller maximum subgraph
size, we can expect a shorter enumeration time but a longer verification time, because fewer subgraphs are used to retrieve candidates. We adjust the maximum subgraph size and evaluate MImR.F with the query size of 20, and show the results in Figure 5.4. We can observe that for queries with graph size of 20, if we use MImR.F, the optimal maximum subgraph size for subgraph enumeration is 5. Below that, verification time increases significantly, while above that, enumeration time increases much. Thus, we can achieve better response times than those in Figure 5.3, if we tune and find the best maximum subgraph size for subgraph enumeration.

5.5.5 Time Complexity of Subgraph Selection Methods

To compare the time complexity for our proposed methods, MP.F, MII.F, and MImR.F, we select a data set with 100 chemical structures. We use a testing set of 100 queries with 20 queries per $Size(G_q) = \{4, 8, 12, 16, 20\}$, and a training set of 500 queries with the same distribution. We all use forward feature selection for the three methods. We show the time complexity and the average precision of query answers in Figure 5.5. The results demonstrate that MImR.F is significantly more computationally efficient than MP.F and MII.F. Furthermore, MImR.F can achieve a slightly worse result in comparison to MP.F or MII.F. Note that MP.F and MII.F may not achieve the optimal performance since we use forward feature
selection. To find the better solution using a global method of feature selection, we expect significantly more computational costs. We can observe from Figure 5.5 (b) that the precision increases while more features are selected. After a certain number of features have been selected, the precision reaches the highest value, i.e., adding more features cannot increase the information contained in the feature set for the testing query set. Thus, if the user query set is known, we can find this point and stop adding useless features to the index after this point. A better feature selection method can achieve the highest precision with a smaller number of features. Note that there is no overfitting problem for the task of subgraph queries. This is because in the subgraph query problem, given that a query graph is a subgraph of a graph, all the subgraph features of this query graph are always subgraphs of the same graph. Adding more features can always prune out more (at least zero) unsatisfied candidates and improve (or at least maintain) the precision of subgraph queries.
Chapter 6

Searching for Similar Graphs

As mentioned in the last chapter, besides subgraph queries, similarity graph queries are also one of the broadly used query model for graph searches. In this chapter, we discuss related issues about similarity graph queries. First, we review previous works on similarity graph searches that use maximum common edge subgraph isomorphism to measure the similarity between two graphs. Then we introduce a new similarity graph search algorithm and a ranking method based on a graph kernel and learned weights that can achieve a reasonably high search quality and a much faster online search time than previous works.

6.1 Background

As mentioned before, graphs have been used to represent structured data for a long time, and a typical but simple graph query is a subgraph query that searches for graphs containing exactly the query graph, i.e., the support [54]. However, sufficient knowledge to select subgraphs to characterize the desired graphs is required and sometimes no support exists, so the similarity graph query searching for all graphs similar to the query graph is desired to bypass the subgraph selection [96, 100, 101]. To measure the similarity of two graphs, previous methods [80, 95] usually use the size of the maximum common edge subgraph (MCEG) between two graphs, i.e., the number of edges in the MCEG (Figure 6.1). The crux of similarity graph search lies in the complexity of the MCEG isomorphism algorithm for similarity measurement. However, since the MCEG isomorphism algorithm is
NP-hard [54], it is prohibitively expensive to scan all graphs in real time. For a large graph data set, rather than executing MCEG isomorphism tests on the fly, we need to index the graphs offline using subgraph features to enable fast graph retrieval. Previous works [97, 80] use different filters to prune out unsatisfied graphs given a user specified minimum size of MCEGs. If users need more search results, the minimum size of MCEGs has to be reduced and more graphs are retrieved. However, previous methods are still slow due to the following reasons: 1) MCEG isomorphism tests have to be executed on the filtered graph set, which slows down the search process extremely, and 2) when users need more graphs, the minimum MCEG constraint has to be relaxed and the process has to be repeated.

The goal of using the size of MCEG to measure the similarity of two graphs is to rank retrieved graphs from the most similar one to the least, and return the retrieved graphs in that order to the end-user. Instead of using MCEG to rank retrieved results, we propose a novel approach that uses a linear graph kernel function to rank retrieved graphs using the indexed subgraph features and feature weights learned from a training set [102]. Our method generates a training set offline using MCEG isomorphism, a training query set, and a graph set. Our approach avoids online MCEG isomorphism tests and is more efficient computationally than previous methods using filters and online MCEG isomorphism tests. Experimental results also show that our method can achieve a reasonably high normalized discounted cumulative gain [103] in a significantly shorter time in comparison to existing methods. Moreover, since our method learns the ranking function from a training set, it can be applied to any other metrics of similarity, including explicit similarity scores labeled by human experts or implicit partial orders extracted from user logs.

Our similarity graph search technique involves four stages: 1) graph feature extraction, 2) graph ranking learning, 3) graph indexing, and 4) graph search and ranking. First, subgraph features are extracted and selected from the graph set, each subgraph is converted into a canonical string, so that each graph is mapped into a linear space of subgraphs. Second, a training set is generated offline and the subgraph feature weights for graph ranking computing are learned from it offline. Third, the graph index is constructed using canonical strings of subgraphs. Finally, for a given graph query, graphs are retrieved and ranked using similarity scores
6.2 Related work

Besides three categories of related work for subgraph queries in the last chapter: frequent subgraph mining, graph indexing, and graph search, there are two additional areas of work related to ours: learning to rank and graph kernels. Previous works [97, 80] use MCEG isomorphism to measure the similarity of the query and a graph, which have an extremely long response time for an online query. Although filters have been used to remove graphs with small MCEG sizes, this approach is still too slow to support fast online similarity graph searches.

Previous works about ranking learning mainly fall into two categories in terms of the training instances and the labels: 1) search results [104], where the relevance scores of each result are the labels, and 2) search result pairs [103], where the partial orders (relative order) of the two elements in the pair are the labels. For Case 1, usually users or domain experts can rate the relevance scores of entities. Based on types of score values, methods of binary classification [105], multi-class classification [104], or regression are applied [104]. For Case 2, the partial orders in pairs can be obtained both from explicit relevance scores labeled by experts or implicit relative orders from user click-through logs. Some methods [103] are proposed for training based on instances of ranked orders of entities with a goal to minimize the number of disordered pairs of search results. Generally, two metrics, maximum average precision (MAP) [105] and (normalized) discounted cumulative gain (NDCG) [103], are proposed to evaluate the performance of ranking functions. To our best knowledge, there is no work on learning to rank graphs. There are
various works about graph-based ranking that is to rank vertices on a graph, like PageRank [106]. Some previous works of graph mining also works on entity and relation graphs [107, 108], which is different from our scenario. Various types of graph kernels are proposed to measure the similarity of graphs [109], which can be applied on graph ranking during search. However, graph kernels are not learnable and not fast enough to support online graph searches. Our research addresses graph search works using learned graph kernels for ranking.

### 6.3 Preliminaries

In this section, we first give preliminary notations for graphs, ranking evaluation, and existing graph similarity measurement methods based on maximum common edge subgraph isomorphism. Table 6.1 lists the notations and their meanings used throughout this chapter. Besides the preliminaries about graphs defined in Section 5.3 of the last chapter, we introduce notations about Discounted Cumulative Gain and Maximum Common Edge Subgraph.

#### 6.3.1 Discounted Cumulative Gain

As mentioned before, discounted cumulative gain (DCG) is one of the most widely used metrics to evaluate the performance of ranking functions. Given a query $q$ and top $n$ ordered results returned and ranked by a system, the DCG score is

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_i$</td>
<td>graph</td>
</tr>
<tr>
<td>$G'_i$</td>
<td>subgraph</td>
</tr>
<tr>
<td>$G$</td>
<td>query graph</td>
</tr>
<tr>
<td>$D$</td>
<td>graph set</td>
</tr>
<tr>
<td>$D_{G_q}$</td>
<td>support of $G_q$</td>
</tr>
<tr>
<td>$T$</td>
<td>training set</td>
</tr>
<tr>
<td>$y_n$</td>
<td>similarity score</td>
</tr>
<tr>
<td>$\epsilon_n$</td>
<td>regression error</td>
</tr>
<tr>
<td>$K(G_i,G_j)$</td>
<td>kernel of $G_i$ &amp; $G_j$</td>
</tr>
<tr>
<td>$Q$</td>
<td>query set</td>
</tr>
<tr>
<td>$L$</td>
<td>loss function</td>
</tr>
<tr>
<td>$G'$</td>
<td>MCEG of $G_i$ and $G_j$</td>
</tr>
<tr>
<td>$F_{G' \subseteq G}$</td>
<td>frequency of $G'$ in $G$</td>
</tr>
<tr>
<td>$\bar{y}_n$</td>
<td>predicted $y_n$</td>
</tr>
<tr>
<td>$w(\epsilon_n)$</td>
<td>weight of $\epsilon_n$</td>
</tr>
<tr>
<td>$W(G')$</td>
<td>subgraph cluster weight</td>
</tr>
<tr>
<td>$W(Cluster(G'))$</td>
<td>subgraph cluster weight</td>
</tr>
</tbody>
</table>
computed as follows [104],
\[
DCG = \sum_{i=1}^{n} c_i f(y_i),
\]
(6.1)
where \(y_i, i = 1, ..., n\) are the real relevance scores of the top \(n\) ordered results, \(c_i\) is a non-increasing function of \(i\), typically,
\[
c_i = 1/\log(i + 1),
\]
(6.2)
and \(f(y_i)\) is a non-decreasing function of \(y_i\), typically,
\[
f(y_i) = 2^{y_i} + 1, \text{ or sometimes } f(y_i) = y_i.
\]
(6.3)
If \(y_i\) is higher, the result \(i\) is more relevant. If \(y_i \in \{0, 1\}\), only relevance and irrelevance are considered. Normalized discounted cumulative gain (NDCG) is a score that normalize the DCG score into the interval of \([0, 1]\) for each query using the maximum DCG that can be achieved. Thus, NDCG is computed as follows [104],
\[
NDCG_q = \frac{DCG_q}{DCG_{q,\text{max}}},
\]
(6.4)
where \(DCG_{q,\text{max}}\) is the maximum possible DCG score for the query \(q\). The average NDCG for the whole query set \(Q\) is defined as
\[
NDCG_Q = \frac{1}{|Q|} \sum_{q \in Q} NDCG_q.
\]
(6.5)
Usually NDCG is used to make the ranking results for different queries comparable, because if \(DCG_{\text{max}}\) is different for two queries, simply comparing their DCG cannot decide which one has a better ranking list.

### 6.3.2 Maximum Common Edge Subgraph

As mentioned before, MCEG is the traditional method to measure the similarity or relevance between a query graph and retrieved graphs especially in Chemoinformatics. A maximum common edge subgraph is defined as follows,

**Definition 6.3.1. Maximum Common Edge Subgraph:** A graph \(G'\) is a
common edge subgraph of $G_i$ and $G_j$, if $G'$ is isomorphic to subgraphs of $G_i$ and $G_j$. A common edge subgraph $G'$ of $G_i$ and $G_j$ is a maximum common edge subgraph, i.e., $MCEG(G_i, G_j)$, iff no common edge subgraph $G''$ of $G_i$ and $G_j$ exists that $|E(G'')| > |E(G')|$, i.e., the edge count on $G''$ is larger than that on $G'$. The size of a MCEG, $|MCEG(G_i, G_j)|$, is defined as its edge count.

Two examples of the MCEG between a query and a retrieved graph are shown in Figure 6.1 (note that an MCEG is not necessarily a connected graph). Obviously, the possible maximum size of MCEG is at most the size of the smaller graph between two graphs. Usually, the query graph is smaller than the retrieved graph. Thus, the possible maximum size of the MCEG is larger for larger query graphs. To make the similarity scores comparable between different sizes of query graphs in our research, we normalize the MCEG sizes into the interval $[0, 4]$ by dividing the size of the query graph, similar to the relevance score range of previous work on learning to rank [103, 104], where 4 means the query graph is a subgraph of the retrieved graph, while 0 means no edge matched. These normalized MCEG sizes are used as the similarity scores for training and test in our experiments.

### 6.4 Learn to Rank Graphs

In this section, first we describe our similarity graph search algorithm. Then we propose the weighted linear graph kernel used to measure the similarity of graphs for ranking. Finally we describe how to learn the weights of the linear graph kernel.

#### 6.4.1 Similarity Graph Search

A naive approach to similarity graph search is to scan all the graphs to find MCEGs of the query and each graph. As mentioned before, this is prohibitively expensive to be executed in real time during the search process. Efficient methods are desired. We propose a new similarity graph search algorithm shown in Algorithm 9. Usually previous methods first use filters to remove a part of graphs with lower MCEG sizes than a given threshold, and then perform the MCEG isomorphism test to compute the real MCEG sizes as the similarity scores [95, 80]. Because the problem of MCEG isomorphism is NP-hard [95], all algorithms for MCEG
isomorphism are extremely expensive. This makes online similarity graph search prohibitively slow. Our algorithm first returns all the graphs in the support of the query graph that have the maximum MCEG size, and then use a fast graph ranking function to compute a heuristic similarity score. To return the support of the query graph, subgraph isomorphism tests are required to check retrieved candidates. However, algorithms for subgraph isomorphism are significantly faster than those for MCEG isomorphism [95, 54]. Our proposed fast graph ranking function uses weighted kernel of subgraph features. It needs algorithms for canonical labeling of graphs that has a similar computational cost to algorithms of subgraph isomorphism [95, 54]. Thus, our proposed method is significantly faster for online queries in comparison with methods using MCEG (shown in Section 6.5.5).

First, we assume we have build an index of graphs in a database using subgraph features of those graphs. Subgraph features can be discovered from those graphs using any previous methods [54, 62, 84, 86]. Then, as illustrated in Algorithm 9 given a query graph $G_q$, the algorithm finds the support of $G_q$, $D_{G_q}$ (Line 1-11). If the support of a query graph is non-empty, all the graphs in the support have the maximum size MCEG. Thus, all the graphs in the support should be returned as the top-most candidates in the result list. If $G_q$ is indexed, it is simple to find $D_{G_q}$ using the index; Otherwise, candidates containing all the indexed subgraph features of $G_q$ is returned and subgraph isomorphism is performed to remove graphs that do not contain $G_q$. Algorithms of subgraph isomorphism are much more time efficient than algorithms of MCEG isomorphism, even though both of them are NP-hard. Thus, the method using subgraph isomorphism to select top candidates in advance can reduce the response time significantly with the same retrieved results as those of the method using MCEG isomorphism, if the number of graphs in the support of the query is larger than or equal to the desired number of top similar candidates.

Second, if not enough results are returned, similar graphs with lower similarity scores are returned (Line 12-19). If the MCEG size is used as the similarity score, then the similarity function is based on MCEG isomorphism. Our proposed method using the weighted kernel as the similarity function. All the graphs in the database containing at least one indexed subgraph feature of $G_q$ is returned as candidates except support graphs found at the first stage. For each graph candidate and the query graph $G_q$, a similarity score is computed using a weighted
Algorithm 9 Similarity Graph Search

Algorithm: SGS($G_q, S, \text{Index}_D, n$):

Input: Query Subgraph $G_q$,
index of the subgraph set $S$,
index of the graph set $D$, $\text{Index}_D$, and
the number of returned results, $n$.

Output: A sorted list of $n$ graphs similar to $G_q$, $\text{List}_{G_q}$.

1. if $G_q$ is indexed,
2. find all $G \supseteq G_q$ using $\text{Index}_D$, i.e., the support of $G_q$, $D_{G_q}$;
3. else
4. $D_{G_q} = \{\emptyset\}$;
5. find all subgraphs of $G_q$, $G'_q \in S$ with $F_{G'_q \subseteq G_q}$;
6. for all $G'_q$ do
7. find $D_{G'_q}$, where $\forall G \in D_{G'_q}$, $F_{G'_q \subseteq G} \geq F_{G'_q \subseteq G_q}$;
8. then $D_{G_q} = D_{G_q} \cap D_{G'_q}$;
9. for all $G \in D_{G_q}$ do
10. if subgraphIsomorphism($G_q, G$)==false, remove $G$;
11. if $|D_{G_q}| \geq n$ return $\text{List}_{G_q} = \text{top n graphs} \in D_{G_q}$;
12. $S_{G_q} = \{\emptyset\}$;
13. find all subgraphs of $G_q$, $G'_q \in S$;
14. for all $G'_q$ do
15. find $D_{G'_q}$, where $\forall G \in D_{G'_q}$, $F_{G'_q \subseteq G} \geq 0$;
16. then $D_{G_q} = D_{G_q} \cup D_{G'_q}$;
17. $S_{G_q} = S_{G_q} - D_{G_q}$;
18. for all $G \in S_{G_q}$ compute similarity($G_q, G$);
19. sort $S_{G_q}$ in terms of similarity($G_q, G$);
20. return $\text{List}_{G_q} = D_{G_q} + \text{top} (n - |D_{G_q}|)$ graphs $\in S_{G_q}$;

linear graph kernel based on the indexed subgraph features and corresponding weights. This similarity score computation is fast and it can be computed during the search process from the index. No additional post-processing is needed after the search process. Finally, graphs are sorted based on the similarity scores and the top results are returned.

6.4.2 Graph Kernels

We utilize a linear weighted graph kernel to compute similarity scores of two graphs to rank the retrieved graphs. A graph kernel is defined as follows,

Definition 6.4.1. Graph Kernel: Let $X$ be a set of graphs, $\mathcal{R}$ denotes the real
numbers, \( \times \) denotes set product, the function \( K : X \times X \to \mathcal{R} \) is a kernel on \( X \times X \) if \( K \) is symmetric, i.e. \( \forall G_i \) and \( G_j \in X, K(G_i, G_j) = K(G_j, G_i) \), and \( K \) is positive semi-definite, i.e. \( \forall N \geq 1 \) and \( \forall G_1, G_2, ..., G_N \in X \), the \( N \times N \) matrix \( K \) defined by \( K_{ij} = K(G_i, G_j) \) is positive semi-definite, i.e. \( \sum_{ij} c_i c_j K_{ij} \geq 0, \forall c_1, c_2, ..., c_N \in \mathcal{R} \). Equivalently, a symmetric matrix is positive semi-definite if all its eigenvalues are nonnegative [72].

The MCEG sizes of two graphs is also a graph kernel, but finding the MCEG is extremely time-consuming. We require a time-efficient graph kernel function that can be used to replace the kernel function of MCEG sizes. There are many elaborate graph kernels, such as convolution kernels [72, 110] and diffusion kernels [111]. They are computationally expensive and not learnable. We define a weighted linear graph kernel based on indexed subgraph features and corresponding frequencies as follows,

\[
K(G_i, G_j) = \sum_{G' \in S} W(G') \min(F_{G' \subseteq G_i}, F_{G' \subseteq G_j}).
\] (6.6)

This linear graph kernel is significantly faster than MCEG isomorphism and other graph kernels. More importantly, it is learnable. We can give more weights to more important of subgraphs to return a better ranking list. We also show the effect of learning in Section 6.5.4. \( W(G') \) are the learnable parameters in this kernel function. Thus, our goal is to learn the kernel function to approximate a target function for ranking, but not necessarily the same as the function finding MCEG sizes.

### 6.4.3 Feature Exaction for Subgraph Clustering

As in text classification tasks, our learning task also suffers the data sparsity problem [112]. Due to the data sparseness, many features appearing in the test set may not have appeared in the training set. We introduce a way to smooth the space of subgraph features by assigning clusters to subgraphs. The data sparsity problem in graph learning is even more serious than in text classification, because there are massive subgraphs and most of them occur very infrequently. Thus, learning weights of subgraphs is difficult due to the sparsity. With the goal to make the space dense, we use a feature extraction method to generate features
from subgraphs, and cluster subgraphs with the same feature vector together into a single dimension. Let the many-to-one mapping function from a subgraph $G'$ to a subgraph cluster using the proposed feature extraction method as $\text{Cluster}(G')$. Then we can rewrite the linear graph kernel as follows,

$$K(G_i, G_j) = \sum_{G' \in S} W(\text{Cluster}(G')) \min(F_{G' \subseteq G_i}, F_{G' \subseteq G_j}). \quad (6.7)$$

We extract the following features of a subgraph: the number of edges, the number of vertices with a specific label, the number of branches, and the number of cycles.

### 6.4.4 Kernel Learning using Regression

Suppose we have a training set with $N$ instances, $T = \{G_{(q,n)}, G_n, y_n\}_{n=1}^N$, where each instance is a pair of a query graph $G_{(q,n)}$ and a retrieved graph $G_n$, and $y_n$ is the similarity score of them generated by a certain way. As mentioned before, if $y_n \in \{1, 0\}$, it represents only relevance or irrelevance between $G_{(q,n)}$ and $G_n$; otherwise, it represents the similarity between $G_{(q,n)}$ and $G_n$. This training set can be generated by arbitrary similarity functions that take in two graphs $G_{(q,n)}$ and $G_n$ as inputs and output a similarity score $y_n$. In this work, we use the normalized MCEG sizes as the “true” similarity scores, $y_n$.

Our eventual goal is to find the optimal weighted graph kernel that maximizes the NDCG function that is the metric to evaluate the ranked retrieved results. However, the objective function of NDCG cannot be represented by the parameters of the graph kernel in a closed form, so we cannot optimize the NDCG function directly and find the optimal graph kernel. Instead, we optimize a specific loss function $f(y_n)$, the non-increasing function in Equation 6.3, using regression. Previous work [104] showed that regression on $f(y_n)$ can achieve a better NDCG of the ranked search results than regression on $y_n$.

Thus, one of the key issues is to choose the loss function. Basically there are two widely used loss functions, the $L_1$ loss function,

$$L = \sum_{n=1}^N |e_n| = \sum_{n=1}^N |f(y_n) - f(\hat{y}_n)|, \quad (6.8)$$
and the $L^2$ loss function,

$$L = \sum_{n=1}^{N} (e_n)^2 = \sum_{n=1}^{N} (f(y_n) - f(\hat{y}_n))^2, \quad (6.9)$$

where $e_n$ is the error of the instance $n$ and $\hat{y}_n$ is the predicted value of $y_n$.

We use a weighted $L^2$ loss function because of two reasons: 1) linear least square is the major traditional objective function of regression that is easy to optimize, and 2) empirically it is better to maximize the NDCG values of the ranked search results than using the $L^1$ loss function. We expand on the second reason in the next section.

### 6.4.5 Weighted Loss Function and Weighted Sampling

As mentioned in the last section, we should choose a good loss function, so that when it is optimized, the NDCG values of the ranked search results can achieve a high value. Previous work [113] proposed a method based on an intuition to weight instances by their importance. Instances with higher relevance scores are considered more important, so that they have higher weights. However, no previous work determined that what the value of the instance weights should be.

First of all, we use the $L^1$ loss function instead of the $L^2$ loss function, because we need to figure out how much a loss of the DCG value an error in the $L^1$ loss function can cause. For example, the same error value in the loss function for different instances may result in different losses of the DCG value. Intuitively, we can use a weighted $L^1$ loss function, where the weight for each instance represents approximately how much of the DCG value its error in the $L^1$ loss function causes, although in practice, the value changing of the DCG function and the $L^1$ loss function may not be a simple linear relation.

Thus, the key issue for weighted $L^1$ loss function is, given an error $e_n = f(y_n) - f(\hat{y}_n)$ of the predicted similarity score $\hat{y}_n$ for an instance $n$ that ranks $n$, its real similarity score $y_n$ and its real ranking weight $c_n$, what is the expected loss of the DCG function? For example, for a positive error $e_n > 0$, we can easily find that
the loss of DCG is as follows

\[ L_{DCG}(n, 1) = (c_n - c_{n+1})(f(y_n) - f(y_{n+1})), \]  

(6.10)

where \( f(y_{n+1}) \approx f(\hat{y}_n) \) and \( c_{n+1} = \hat{c}_n \), if the error \( e_n \) reduces the rank of instance \( n \) by 1 in the ordered list, and

\[ L_{DCG}(n, k) = \sum_{i=n}^{n+k-1} (c_i - c_{i+1})(f(y_n) - f(y_{i+1})), \]  

(6.11)

where \( f(y_{n+k}) \approx f(\hat{y}_n) \) and \( c_{n+k} = \hat{c}_n \), if that reduces the rank of instance \( n \) by \( k \) in the ordered list in the ordered list. Similarly for a negative error \( e_n < 0 \) that makes \( n \) rank \( k \) places higher in the ordered list, the loss of DCG is

\[ L_{DCG}(n, k) = \sum_{i=n-1}^{n-k} (c_i - c_{i+1})(f(y_i) - f(y_n)). \]  

(6.12)

Obviously a kernel function generating perfect similarity scores can result in perfect ranking. However, imperfect kernel functions may also result in perfect ranking, if errors are small enough and the ranking order is not affected [104]. To simply the situation we assume there are infinite instances that make the function \( f(y_i) \) of \( i \) differentiable.

For a positive error \( e_n \) that makes \( n \) rank \( k \) lower, the loss of DCG in Equation 6.11 can be estimated as follows,

\[ L_{DCG}(n, k) = \int_n^{n+k} \left(-\frac{dc_i}{di}\right)(f(y_n) - f(y_i))di. \]  

(6.13)

Then the derivative of \( L_{DCG} \) with respect to \( e_n \), i.e., how much loss of DCG is caused by the same small \( e_n \), is computed as follows,

\[ \frac{dL_{DCG}(n, k)}{de_n} = \frac{d}{dk} \int_n^{n+k} \left(-\frac{dc_i}{di}\right)(f(y_n) - f(y_i))di/dk \]  

(6.14)

\[ = \frac{dc_{n+k}}{d(n+k)}(f(y_n) - f(y_{n+k})) \frac{d\hat{c}_n}{dk} \frac{f(y_n)}{dk} \approx \frac{d\hat{c}_n}{dk} \frac{f(y_n)}{dk} e_n \]  

(6.15)
Similarly, for a negative error \( e_n^- \), the loss of DCG is

\[
L_{\text{DCG}}(n, -k) = \int_{n-k}^{n} \left( -\frac{dc_i}{dt} \right) (f(y_i) - f(y_n)) di,
\]

(6.16)

and

\[
\frac{dL_{\text{DCG}}(n, -k)}{de_n^-} \approx \frac{dc_{n-k}/dk}{df(\hat{y}_n)/dk} |e_n^-| = \frac{d\hat{c}_n/dk}{df(\hat{y}_n)/dk} |e_n^-|.
\]

(6.17)

Assume the error is small, so that approximately the weight of each instance for the weighted loss function should be the derivative of \( L_{\text{DCG}} \), i.e.,

\[
w(e_n) \propto \frac{dL_{\text{DCG}}(n, k)}{de_n} \approx \frac{\hat{c}_n'}{f'(\hat{y}_n)} |f(y_n) - f(\hat{y}_n)|,
\]

(6.18)

where \( \hat{c}_n' = d\hat{c}_n/dk \) and \( f'(\hat{y}_n) = df(\hat{y}_n)/dk \neq 0 \), which can be explained as how much the error of an instance can affect the DCG values. Thus, we present a new weighted loss function for regression based on these observations as follows,

\[
L_w = \sum_{n=1}^{N} w(e_n) |f(y_n) - f(\hat{y}_n)|
\]

\[
= \sum_{n=1}^{N} \frac{\hat{c}_n'}{f'(\hat{y}_n)} (f(y_n) - f(\hat{y}_n))^2,
\]

(6.19)

which is a weighted \( L^2 \) loss function.

The difficulty is to compute \( \hat{c}_n' \) and \( f'(\hat{y}_n) \). First of all, for each query, the training set must have a full list of retrieved results. Then for each result \( n \), its predicted similarity score \( \hat{y}_n \) should be put into the sorted list of \( y_i \neq n \) and estimate \( \hat{c}_n' \) and \( f'(\hat{y}_n) \). If both of them are linear functions, then the loss function can use a normal unweighted \( L^2 \) loss function. However, both \( \hat{c}_n' \) and \( f'(\hat{y}_n) \) cannot be computed for each pair of query and result individually, and compute them by sorting all the results for each query during search is expensive. Rather than estimate the weights in Equation 6.19, empirically we define the weights as the normalized MCEG sizes (values between \([0, 4]\)). Additionally, if the normalized MCEG size is smaller than a minimum threshold, we simply set the weight as zero. That is, during the training process, we can use the normalized MCEG sizes as the instance weights to learn the subgraph weights that maximize the weighted...
L2 loss function, and use NDCG to evaluate the ranked search results.

Obviously, another option to use the weighted L2 loss function is to use the unweighted L2 loss function but a weighted sampling method to generate a training set. Using this method, we can have a smaller number of instances in the training set in comparison with the method based on uniform sampling but weighted loss function. Once we have defined the loss function for regression and have the weighted sampled training set, we apply the normal linear least square square regression to learn the subgraph feature weights in the graph kernel function in the Equation 6.7 for graph ranking.

6.5 Experimental Evaluation

In this section, we evaluate our proposed approach by comparing it with two heuristics and the method using MCEG isomorphism in terms of NDCG and response time of queries. The method using MCEG isomorphism is assumed to have the optimal ranked results. The two heuristics use the same search process of the proposed approach but different subgraph features. Our proposed approach compute the ranking scores using learned subgraph feature weights, while the two heuristics simply use uniform subgraph feature weights and subgraph sizes as feature weights.

Our experimental results show that:

- Our proposed weighted graph ranking method can achieve a reasonably high NDCG of similarity graph search results in comparison with the “perfect” similarity function of the MCEG sizes.
- The more important benefit is that the linear graph kernel to rank graphs is significantly more time efficient than the methods using the MCEG isomorphism.
- Moreover, the weighted graph ranking method is learnable to approximate other similarity functions beyond the MCEG sizes.
6.5.1 Experimental Data Set

We use the same real data set and test query set as those used by Yan, et al., [54], which is also used in the last chapter to evaluate subgraph queries. It is a NCI/NIH HIV/AIDS antiviral screen data set that contains 43905 chemical structures. The experimental subset contains 10000 chemical structures that are selected randomly from the whole data set and the query set contains 6000 randomly generated queries, i.e., 1000 queries per query size, where \( Size(G_q) = \{4, 8, 12, 16, 20, 24\} \). This query set is used for search result evaluation.

6.5.2 Training Set

In our experiment, rather than using a weighted loss function, we use a weighted sampling method to generate a training set. We generate a training set offline using MCEG isomorphism tests. We first generate 6000 queries with the same distribution of the test query set described above. Then for each query graph, we scan graphs of the 10000 chemical structures with corresponding conditional sampling probability given the normalized MCEG sizes (as mentioned before, they are normalized between \([0,4]\)) between the query and the graph. Finally we use the normalized MCEG sizes as the target similarity scores \( y_n \) for the \( n^{th} \) query-graph pair. Since we only care top 20 search results, we intentionally remove all the query-graph pairs with low normalized MCEG sizes. We also remove query-graph pairs where the query is a subgraph of the graph, because this case is solved by subgraph isomorphism in the Algorithm 9 and no learning is required. Since finding the MCEG between the query and the selected graph is time-consuming, to speed up the training instance generation, we use the following steps: 1) given a query, search all graphs using Algorithm 9, and using the similarity function of the linear graph kernel with uniform feature weights (described in Section 6.5.3), 2) pick only the top 1000 returned graphs and remove graphs among them that are supergraphs of the query, and 3) compute the normalized MCEG sizes \( y_n \) between each survived graph and the query and sample the \( n^{th} \) pair using the probability of \( (y_n/4)/10 \).

The final training set contains instances of pairs of a query and a graph with a similarity score \( y_n \), and each instance has a subgraph feature vector where each
entry is the minimum one of the subgraph frequency on the query and the subgraph frequency on the graph (shown in Equation 6.7). Note that we do not have complete ordered lists of top similar graphs for each query in our training set. Finally, in our experiment, we generate a training set with a total of 459,047 pairs of queries and graphs. In this training set, we do not extract all the possible subgraph features for each graph and query, because this worsens the data sparsity problem. Any previous subgraph feature selection methods can be applied to select a dense subset of frequent subgraphs [61, 83, 82, 54, 75, 84, 89, 90]. We use an open source ParMol\footnote{http://www2.informatik.uni-erlangen.de/Forschung/Projekte/ParMol/?language=en} to extract frequent subgraphs. After feature selection and before training and test, we cluster subgraphs using feature extraction. After subgraph clustering by feature exaction, the total dimensionality is 300 dimensions. We test subgraph clustering using about half of the training pairs, the total dimensionality is 297 dimensions. This means the total number of subgraph clusters converges and our subgraph clustering method can avoid the data sparsity problem and most of the features in the test set appear in the training set.

### 6.5.3 Evaluated Methods

In our experiment, we compare our proposed method with the previous method using MCEG both in terms of search quality using NDCG and the query response time. We also compare linear graph kernel using learned feature weights with two simple heuristic weights to show the effect of learning. Moreover, we also use two different sizes (|S| = 9855 subgraph features v.s. |S| = 50475 subgraph features) of indexed subgraph sets to show the effect of the number of the indexed subgraph set, S. In summary, we compare the following methods: 1) linear graph kernel with subgraph feature weights learned using regression on $f(y_n)$ with the $L2$ loss function and weighted sampling (learn in Figure 6.2 and 6.3), 2) linear graph kernel using subgraph sizes as feature weights (size in Figure 6.2 and 6.3), 3) linear graph kernel with uniform subgraph feature weights (uniform in Figure 6.2 and 6.3), 4) linear graph kernel using subgraph sizes as feature weights with a larger subgraph feature set (size* in Figure 6.2 and 6.3), and 5) linear graph kernel with uniform subgraph feature weights with a larger subgraph feature set.
Note that for NDCG in Figure 6.2 and 6.3, the method using MCEG always has the perfect NDCG, because it is assumed to be the gold standard. For the query response time in Figure 6.4, since the proposed method has similar online response time no matter what kind of subgraph feature weights it uses, we only evaluate the learned weights and called it graph kernel. As mentioned before, all the algorithms of the MCEG isomorphism are NP-hard. In the experiment, we apply the techniques in [80] to optimize the algorithms of the MCEG isomorphism. Thus, the response time of MCEG in Figure 6.4 has been optimized.

6.5.4 NDCG

After the training set is generated, we train and test the model. Experimental NDCG results of top 20 search results are illustrated in Figure 6.2 and 6.3. In Figure 6.2, we evaluate all queries for different query sizes together. Because many queries have their supports found in the graph set, so that graphs in the support always have the maximum MCEG size. This makes the NDCGs of the proposed method very high. To show the effect of the proposed method on “hard” queries whose optimal results have low ranking scores, we select only queries that have no support in the graph sets, and show their curves of NDCGs in Figure 6.3.

First, from the curves of size and uniform in Figure 6.2 and 6.3, we can observe that using the linear graph kernel with uniform subgraph feature weights or subgraph sizes as feature weights can achieve reasonably high NDCGs. We can see even for “hard” queries with empty supports, the average NDCGs for top 10 search results are around 72% - 83%, which is reasonable compared with NDCGs for web search [104].

Second, if we use a larger set of subgraph features, we may not expect an increase of NDCGs, i.e., in most cases, the curves of uniform and size are similar to those of uniform and size. Sometime more subgraphs cause a drop of NDCGs, e.g., when query size = 4. This may be because that the smaller subgraph set already has enough feature information to distinguish graphs. Using more features cannot increase much information and may cause more divergences in the proposed ranking method, especially for small queries. Intuitively, if the
number of features increases from 0 to 100, we can expect an increase of NDCGs, but not the same case if we already have enough subgraph features. Thus, choosing an appropriate number of subgraph features for indexing can not only control the running time, but also result in better search results.

Finally, we compare our proposed method that uses the learned weights and the above four cases. We can observe from the figures that except when query size
Figure 6.3. NDCG 1-20 for queries having no support

= 4, we have a much higher NDCGs (about 2% - 9% improvement) than methods using linear graph kernel but heuristic weights without learning. The reasons that our proposed method performs worse for the case of query size = 4 are: 1) smaller queries usually have larger supports in the graph set than larger ones, and graphs in the supports are removed from our training set, so that there are fewer training pairs for smaller queries, and 2) smaller queries have fewer subgraph features.
Table 6.2. Average NDCGs

<table>
<thead>
<tr>
<th>Method</th>
<th>NDCG 1</th>
<th>NDCG 3</th>
<th>NDCG 10</th>
<th>NDCG 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>All queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>learn</td>
<td>94.224%</td>
<td>94.842%</td>
<td>95.648%</td>
<td>96.308%</td>
</tr>
<tr>
<td>size</td>
<td>93.259%</td>
<td>93.896%</td>
<td>94.716%</td>
<td>95.336%</td>
</tr>
<tr>
<td>uniform</td>
<td>93.403%</td>
<td>94.043%</td>
<td>94.898%</td>
<td>95.570%</td>
</tr>
<tr>
<td>sizeL</td>
<td>93.140%</td>
<td>93.793%</td>
<td>94.687%</td>
<td>95.318%</td>
</tr>
<tr>
<td>uniformL</td>
<td>93.208%</td>
<td>93.872%</td>
<td>94.807%</td>
<td>95.470%</td>
</tr>
<tr>
<td>Queries having no support</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>learn</td>
<td>71.092%</td>
<td>74.255%</td>
<td>78.349%</td>
<td>81.691%</td>
</tr>
<tr>
<td>size</td>
<td>66.171%</td>
<td>69.450%</td>
<td>73.702%</td>
<td>76.937%</td>
</tr>
<tr>
<td>uniform</td>
<td>66.896%</td>
<td>70.188%</td>
<td>74.616%</td>
<td>78.108%</td>
</tr>
<tr>
<td>sizeL</td>
<td>65.575%</td>
<td>68.935%</td>
<td>73.549%</td>
<td>76.832%</td>
</tr>
<tr>
<td>uniformL</td>
<td>65.916%</td>
<td>69.332%</td>
<td>74.152%</td>
<td>77.588%</td>
</tr>
</tbody>
</table>

Thus, smaller queries have much less contributions than larger queries during the training process, so that the learned weights fit more on larger queries.

We average NDCGs of all the queries ignoring the query size in Table 6.2. We can see the over-all NDCGs are improved by about 1% for all queries and 4-5% for queries having no support. From the previous work [104], for the case of a standard deviation = 24 and a sample size = 10000, roughly speaking, the difference of two NDCGs is considered as “significant” if it is larger than 0.47%. Hence, the improvements of NDCGs after learning are roughly statistically significant for both over-all NDCGs or NDCGs for different query sizes except the case of query size = 4.

From the curves of size, uniform, sizeL, and uniformL in Figure 6.2 and 6.3, we can see that these two types of weights have similar performance. Only for large queries, using query sizes as features weights performs slightly better than using uniform feature weights. Another observation from Figure 6.3, we have the worst NDCG curves for the case of query size = 12, which means that the proposed method has the worst NDCGs for queries with medium sizes. This is different for curves in Figure 6.2, where the largest queries have the worse NDCGs. This is because in Figure 6.2, we evaluate all queries including queries with non-empty supports. Larger queries usually have smaller sizes of supports, which is the reason that the largest queries have the worse NDCGs, not due to the proposed ranking
6.5.5 Response Time

In this subsection, we compare the average online response time for a query using the proposed linear graph kernel as the ranking function and using the MCEG isomorphism algorithm. As in the proposed Algorithm 9, to return top $n$ similar graphs using the MCEG isomorphism algorithm, two cases exist: 1) If the top $n$ similar graphs all contain the query as a subgraph, the MCEG isomorphism algorithm does not need to run. In this case, only the subgraph isomorphism algorithm is executed, which is much more time efficient, and the response time of a query is the same as our proposed method. 2) If only part of or none of the top $n$ similar graphs contain the query as a subgraph, the MCEG isomorphism algorithm has to be executed to find other top similar graphs. However, applying the MCEG isomorphism algorithm to scan and rank all the graphs in the database is prohibitively expensive. As mentioned above, previous methods [95, 80] use filters to remove graphs containing smaller MCEGs than the MCEG size threshold before preforming the MCEG isomorphism algorithm. However, no previous work proposed methods to find top $n$ similar graphs containing the largest MCEG sizes.

To simplify the situation for time complexity comparison, we assume that we have a filter to return only 100 graph candidates to execute the MCEG isomorphism test. That is, the curve in Figure 6.4 is the response time that at most 100 MCEG isomorphism tests are performed. Actually for most cases, more than 100 graph
candidates are returned to perform MCEG isomorphism tests [95], which means in practice, using the MCEG isomorphism algorithm requires even a longer average response time than the cases shown in our experiments.

Figure 6.4 shows the curves of average response times of similarity graph queries using two ranking methods: *graph kernel* that uses weighted linear graph kernel where the weights are learned offline, and *MCEG* that uses the MCEG isomorphism algorithm to rank graphs. It illustrates that our proposed method *graph kernel* is significantly more efficient with respect to time than *MCEG*. We also observe that when the query size increases, both the response times of *graph kernel* and *MCEG* increase. Two factors make the response time increase in the two methods: 1) For *graph kernel*, larger query graphs have more indexed subgraph features to enumerate, search from the index, and compute the scores. 2) For *MCEG*, the sizes of supports become smaller when the query graph size increases, which causes the algorithm to trigger the MCEG isomorphism algorithm to find more similar graphs that do not contain the full query graph. Additionally, MCEG isomorphism is more expensive for larger graphs. From Figure 6.4, we can observe that when the query size increases, the response time of *MCEG* increases much faster than *graph kernel* does.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, we have presented a framework of chemical search engine, which can support various types of entity searches, document searches, and graph searches. Within the proposed framework, we have presented related issues and proposed corresponding methods and algorithms. Both theoretical and empirical evaluations illustrated that the proposed framework and methods for the multiple parts work well in practice. Many approaches to specific issues are not only able to be applied in this specific domain, but also have potential to be extended to similar issues in other domains of information retrieval.

The proposed framework mainly involves two relatively independent parts: 1) textual entity mining, indexing, and search, applied on chemical entities like chemical formulae and names, and 2) graph mining, indexing, and search, applied on chemical molecule structures. For the part of entity search, text documents first are crawled from the Web and pre-processed such as classification and segmentation. Second, textual entities are exacted from the documents. Third, extracted entities are analyzed and tokenized to build entity indices, and simultaneously text documents are indexed too. Finally, various query models are supported through the online web service to help users access both entities and documents efficiently and effectively. The part of graph search is similar. First, chemical molecules in the format of graphs are collected online. Second, frequent and irredundant subgraphs are discovered and selected to building the graph index. Then, ranking models
are learned off-line. Finally, web services of subgraph queries and similarity graph queries are provided by using graph inputs.

For text document preprocessing, we proposed a new text segmentation method based on weighted mutual information that can handle both single documents one by one and multiple documents at the same time. The proposed approach outperforms all the previous methods on single-document cases. Moreover, we also showed that doing segmentation among multiple documents can improve the performance tremendously. The experimental results also illustrated that using weighted mutual information can utilize the information of multiple documents to reach a better performance.

For entity extraction, we evaluated various methods, including a classification model, Support Vector Machines, and a sequence labeling model, Conditional Random Fields. We also extended CRFs and presented a hierarchical model, Hierarchical Conditional Random Fields (HCRFs). Experiments showed that SVMs and CRFs perform well and HCRFs perform better, and our techniques outperform existing chemical entity extractors like GATE and Oscar3. For entity indexing, we proposed efficient index pruning schemes to support substring and similarity searches for chemical formulae and names. Two indexing schemes are proposed: segmentation-based indexing for chemical names and frequency-and-discrimination-based indexing for chemical formulae. The former one involves two algorithms: independent frequent subsequence mining and hierarchical text segmentation. Experiments illustrated that most of the discovered subterms in chemical names using our algorithm, independent frequent subsequence mining, have semantic meanings. Examples showed that the proposed hierarchical text segmentation algorithm for automatic chemical name segmentation works well. Experiments also showed that our schemes of index construction and pruning can reduce the number of indexed tokens as well as the index size significantly. Moreover, the response time of similarity searches is considerably reduced. Retrieved ranked results of similarity and substring searches before and after index pruning are highly correlated. For entity search, we also introduced several query models for chemical formula search, which are different from keywords searches in traditional information retrieval. Corresponding ranking functions are designed, which extended the idea of TF-IDF, and considered sequential information in chemical
names and formulae. Experiments and examples showed that the new heuristic ranking functions work well.

For graph searches, we studied issues on subgraph queries and similarity graph queries. For subgraph queries, we proposed a novel probabilistic model applied on subgraph feature selection for index building, which can improve the precision for subgraph queries given a fixed number of indexed subgraph features. We also considered subgraph frequencies in graphs to improve the precision further. We introduced several criteria for feature selection, including Max-Precision (MP), a method that directly optimizes the precision of query answers, Max-Irredundant-Information (MII) and Max-Information-Min-Redundancy (MImR) that are based on a probabilistic model using mutual information. We showed theoretically that MImR and MII are approximations of MP. Moreover, we proposed a greedy feature selection algorithm using MImR that works well in practice. Experiments showed that the proposed approaches perform significantly better than previous methods.

Since subgraph queries have some limitations, we proposed a method to handle similarity graph queries. Correspondingly, we also proposed a fast weighted linear graph kernel to rank returned graphs for similarity graph search. Regarding the subgraph feature weights, we applied heuristics that use uniform feature weights and subgraph sizes as feature weights. We also learned feature weights using regression from a training set generated using weighted sampling. Experiments showed that the proposed approaches have reasonable high NDCGs of similarity graph search results in comparison with the “ prefect” ranking function using maximum common edge subgraph isomorphism. Moreover, experimental results showed that the proposed method is significantly more time efficient than that using MCEG isomorphism, so it is more feasible for online similarity graph search. The more important benefit of the proposed method is that it can learn other ranking functions when MCEG is not an appropriate function to measure the similarity of two graphs.

7.2 Future Work

Our chemical search engine is still at the initial stage. In the future, real data sets should be collected from domain experts or user logs to train and test the proposed
methods in our framework. Besides, there are several potential future directions of our research work worth working on.

**Identity Disambiguation, Fuzzy Matching, and Query Expansion**

Since the same chemical molecule can be mentioned using different representations, identification disambiguation is required. Additionally, because the same string sometimes can represent different chemical molecules, and entities appearing in documents are noisy, fuzzy matching of chemical entities should be considered. Moreover, to search entities and documents, query expansion among different representations of the same molecule is necessary. There are three categories of query expansion as follows, 1) inside of a standard, among synonyms of chemical names, or different representations of chemical formulae, 2) across different standards, different names, or formulae, or InChI, or SMILES, 3) to support structure search for textual entities in documents.

**Probabilistic String Pattern Match**

Usually a systematic chemical name has some certain string pattern, so that this kind of string pattern may be utilized as a feature to detecting new chemical names that are not in the lexicon. However, the rules are very complicated and noise exists. Thus, a method of probabilistic string pattern match based on Markov Models can learn the transition matrix from a training set for chemical names. To reinforce the performance, we also can construct a training set including not only true samples (chemical names), but also false samples (such as English words). The train set can be constructed by using chemical names from a lexicon as true samples and terms from documents as noisy false samples.

**Probabilistic Graph Segmentation**

Usually approaches of probabilistic pattern segmentation fall into two categories: supervised segmentation and unsupervised segmentation. Our hierarchical text segmentation is an unsupervised approach and is a special case of probabilistic pattern segmentation to handle sequential patterns. Graph pattern segmentation is an interesting and challenging issue. It is related to clustering on the graph and finding cuts on the graph. However, the difference is that probabilistic graph segmentation takes a set of i.i.d. graphs as instances, and segment them into semantic meaningful sub-graphs, while clustering on the graph or finding cuts on the graph takes each node on the graph as an instance with relations to other nodes modeled
by the edges on the graph.

**3D Structure Mining, Indexing, and Search**

Currently we represent 2D structures of chemical molecules using graphs. However, 3D structure information is lost. To support more complicated structure searches, 3D graphs with coordinates of vertices can be used to represent chemical molecules. Efficient and effective methods of 3D graph mining, indexing, and search are desired. A canonical labeling method of 3D graphs is desired. Schemes of index construction of 3D graphs are required. Different query models and corresponding ranking functions need to be designed. For example, 3D graph kernel may be introduced.
Bibliography


[62] Yan, X. and J. Han (2003) “CloseGraph: Mining Closed Frequent Graph Patterns,” in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.


Vita

Bingjun Sun

Bingjun Sun was born in Chongqing, China. He entered the Ph. D. degree in the Department of Computer Science and Engineering at the Pennsylvania State University in 2004. Prior to that, he had received a Master of Science degree in the Department of Computer Science and Engineering at the same University in 2003. He also received a B.Arch degree in 1997 and a M.Arch degree in 2000 from Tsinghua University, China. He is a student member of ACM. His main areas of research interest are information retrieval and data mining. He also has broad interests and experiences on multiple areas, such as database and information systems, multi-agent systems, artificial intelligence, architecture and urban planning, statistics.