Mining, Indexing, and Searching for Textual Chemical Molecule Information on the Web

Bingjun Sun
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802, USA
bsun@cse.psu.edu

Prasenjit Mitra
College of Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802, USA
pmitra@ist.psu.edu

C. Lee Giles
College of Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802, USA
giles@ist.psu.edu

ABSTRACT

Current search engines do not support user searches for chemical entities (chemical names and formulae) beyond simple keyword searches. Usually a chemical molecule can be represented in multiple textual ways. A simple keyword search would retrieve only the exact match and not the others. We show how to build a search engine that enables searches for chemical entities and demonstrate empirically that it improves the relevance of returned documents. 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 chemical formula tagging that considers long-term dependencies at the sentence level. To substring searches of chemical names, a search engine must index substrings of chemical names. Indexing all possible sub-sequences is not feasible in practice. We propose an algorithm for independent frequent subsequence mining to discover sub-terms of chemical names with their probabilities. We then propose an unsupervised hierarchical text segmentation (HTS) method to represent a sequence with a tree structure based on discovered independent frequent subsequences, so that sub-terms on the HTS tree should be indexed. Query models with corresponding ranking functions are introduced for chemical name searches. Experiments show that our approaches to chemical entity tagging perform well. Furthermore, we show that index pruning can reduce the index size and query time without changing the returned ranked results significantly. Finally, experiments show that our approaches out-perform traditional methods for document search with ambiguous chemical terms.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering,Query formulation,Retrieval models,Search process*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Linguistic processing*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; J.2 [**Physical Sciences and Engineering**]: Chemistry

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

General Terms

Algorithms, Design, Experimentation

Keywords

Entity extraction, conditional random fields, independent frequent subsequence, hierarchical text segmentation, index pruning, substring search, similarity search, ranking

1. INTRODUCTION

Chemists search for chemical entities (names and formulae) on the Web [19]. We outline how a chemical entity search engine can be built to enable such searches. We conjecture that a similar approach can be utilized to construct other vertical, domain-specific search engines that need to recognize special entities.

Users search for chemical formulae using different forms or semantics of the same molecule, e.g., they may search for "CH4" or for "H4C". Furthermore, users use substrings of chemical names in their search, especially, for well-known functional groups, e.g., they may search for "ethyl" to search for the name in Figure 4. General-purpose search engines usually support exact keyword searches and do not return documents where only parts of query keywords occur. In this work, we introduce multiple query semantics to allow partial and fuzzy searches for chemical entities. To enable fast searches, we show how to build indexes on chemical entities.

Our investigation shows that end-users performing chemical entity search using a generic search engine, like Google, do not get very accurate results. One of the reasons is that general-purpose search engines do not discriminate between chemical formulae and ambiguous terms. Our search engine identifies chemical formulae and disambiguates its occurrences from those of other strings that could potentially be valid chemical formulae, e.g., "He", but are actually not. Disambiguating terms is a hard problem because it requires context-sensitive analysis.

A generic search engine does not support document retrieval using substring match query keywords. If we want to enable this kind of searches, a naive approach is to construct an index of all possible sub-terms in documents. Such an index will be prohibitively large and expensive. To support partial chemical name searches, our search engine segments a chemical name into meaningful sub-terms automatically by utilizing the occurrences of sub-terms in chemical

names. Such segmentation and indexing allow end-users to perform fuzzy searches for chemical names, including substring search and similarity search. Because the space of possible substrings of all chemical names appearing in text is extremely large, an intelligent selection needs to be made as to which substrings should be indexed. We propose to mine independent frequent patterns, use information about those patterns for chemical name segmentation, and then index sub-terms. Our empirical evaluation shows that this indexing scheme results in substantial memory savings while producing comparable query results in reasonable time. Similarly, index pruning is also applied to chemical formulae. Finally, user interaction is provided so that scientists can choose the relevant query expansions. Our experiments also show that the chemical entity search engine outperforms general purpose search engines (as expected).

Constructing an accurate domain-specific search engine is a hard problem. For example, tagging chemical entities is difficult due to noise from text recognition errors by OCR, PDF transformation, etc. Chemical lexicons are often incomplete especially with respect to newly discovered molecules. Often they do not contain all synonyms or do not contain all the variations of a chemical formula. Understanding the natural language context of the text to determine that an ambiguous string is chemical entity or not (e.g., "He" may actually refer to 'HeliumŠ as opposed to the pronoun) is also a hard problem to solve with very high precision even if dictionaries of chemical elements and formulae are used. Moreover, many chemical names are long phrases composed of several terms so that general tokenizers for natural languages may tokenize a chemical entity into several tokens. Finally, different representations of a chemical molecule (e.g., "acetic acid" is written as "CH3COOH" or "C2H4O2") have to be taken into account while indexing (or use other strategies like query expansion) to provide accurate and relevant results with a good coverage.

Our search engine has two stages: 1) mining textual chemical molecule information, and 2) indexing and searching for textual chemical molecule information. In the first stage, chemical entities are tagged from text and analyzed for indexing. Machine-learning-based approaches utilizing domain knowledge perform well, because they can mine implicit rules as well as utilize prior knowledge. In the second stage, each chemical entity is indexed. In order to answer user queries with different search semantics, we have to design ranking functions that enable these searches. Chemical entities and documents are ranked using our ranking schemes to enable fast searches in response to queries from users. Synonyms or similar chemical entities are suggested to the user, and if the user clicks on any of these alternatives, the search engine returns the corresponding documents.

Our prior work on chemical formulae in the text [25], while this paper focuses approaches to handle chemical names. We show how a domain-specific chemical entity search engine can be built that includes tasks of entity tagging, mining frequent subsequences, text segmentation, index pruning, and supporting different query models. Some previous work is in [25]. We propose several novel methods for chemical name tagging and indexing in this paper. The major contributions of this paper are as follows:

- We propose a model of hierarchical conditional random fields for entity tagging that can consider the long-term dependencies at different levels of documents.
- We introduce a new concept, *independent frequent subsequences*, and propose an algorithm to find those subsequences, with the goal to discover sub-terms with corresponding probabilities in chemical names.
- We propose an unsupervised method of hierarchical text segmentation and use it for chemical name index pruning. This approach can be extended to index any kinds of sequences, such as DNA.
- We present various query models for chemical name searches with corresponding ranking functions.

The rest of this paper is organized as follows: Section 2 reviews related works. Section 3 presents approaches of mining chemical names and formulas based on CRFs and HCRFs. We also propose algorithms for independent frequent subsequence mining and hierarchical text segmentation. Section 4 describes the index schemes, query models, and ranking functions for name and formula searches. Section 5 presents experiments and results. Conclusions and future directions are discussed in Section 6. Web service of our system is provided online [1].

2. RELATED WORK

Banville has provided a high-level overview on mining chemical structure information from the literature [12]. *Hidden Markov Models* (HMMs) [9] are one of the common methods for entity tagging. HMMs have strong independence assumptions and suffer from the label-bias problem. Other methods such as Maximum Entropy Markov Models (MEMMs) [16] and Conditional Random Fields (CRFs) [11] have been proposed. CRFs are undirected graph models that can avoid the label-bias problem and relax the independence assumption. CRFs have been used in many applications, such as detecting names [17], proteins [22], genes [18], and chemical formulae [25]. Other methods based on lexicons and Bayesian classification have been used to recognize chemical nomenclature [26]. The most common method used to search for a chemical molecule is *substructure search* [27], which retrieves all molecules with the query substructure. However, users require sufficient knowledge to select substructures to characterize the desired molecules for substring search, so *similarity search*[27, 29, 23, 21] is desired by users to bypass the substructure selection. Other related work involves mining for frequent substructures [8, 10] or for frequent sub-patterns [28, 30]. Some previous work has addressed how to handle entity-oriented search [6, 15]. Chemical markup languages have been proposed [20] but are still not in wide use. None of the previous work focuses on the same problem as ours. They are mainly about biological entity tagging or chemical structure search. We have shown how to extract and search for chemical formulae in text [25], but we propose several new methods to extract and search for chemical names in text documents.

3. MOLECULE INFORMATION MINING

In this section, we discuss three mining issues before index construction: how to tag chemical entities from text, how to mine frequent sub-terms from chemical names, and how to hierarchically segment chemical names into sub-terms that can be used for indexing.

[1] http://chemxseer.ist.psu.edu/

3.1 Conditional Random Fields

Because we are building a chemical entity search engine, accurately tagging entities in text is important. CRFs [11] is a powerful method for tagging sequential data. We also demonstrate how the accuracy can be improved by considering long-term dependency in the surrounding context at different levels using hierarchical models of CRFs.

Conditional Random Fields

Suppose we have a training set S of labelled graphs. Each graph in S is an i.i.d. sample [11]. CRFs model each graph as an undirected graph $G = (V, E)$. 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 two labels $y_v, y_{v'}$. For each sample, the conditional probability $p(\mathbf{y}|\mathbf{x}, \lambda)$, where $\mathbf{x}$ is the observation vector of all vertices in G, $\mathbf{y}$ is the label vector, and λ is the parameter vector of the model, represents the probability of $\mathbf{y}$ given $\mathbf{x}$. An exponential probabilistic model based on feature functions is used to model the conditional probability,

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

where $F_j(\mathbf{y}, \mathbf{x})$ is a feature function that extracts a feature from $\mathbf{y}$ and $\mathbf{x}$. $Z(\mathbf{x})$ is a normalization factor.

For sequential data, usually chain-structured CRF models are applied, where only the labels (y_{i-1} and y_i) of neighbors in a sequence are dependent. Moreover, usually only binary features are considered. There are two types of features, state features $F_j = S_j(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} s_j(y_i, \mathbf{x}, i)$ to consider only the label (y_i) of a single vertex and transition features $F_j = T_j(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} t_j(y_{i-1}, y_i, \mathbf{x}, i)$ to consider mutual dependence of vertex labels (y_{i-1} and y_i) for each edge e in G. State features include two types: single-vertex features obtained from the observation of a single vertex and overlapping features obtained from the observations of adjacent vertices. Transition features are combinations of vertex labels and state features. Each feature has a weight λ_j to specify if the corresponding feature is favored. The weight λ_j should be highly positive if feature j tends to be "on" for the training data, and highly negative if it tends to be "off".

The log-likelihood for the whole training set S is given by

$$L(\lambda) = \sum_{\mathbf{x} \in S} log(p(\mathbf{y}|\mathbf{x}, \lambda)), \quad (2)$$

where the conditional probability of each sample can be estimated from the data set. Maximize log-likelihood is applied to estimate parameters λ during training. To predict labels of $\mathbf{x}$, the probabilities of all possible $\mathbf{y}$ are computed using Equation 1, and $\mathbf{y}$ with the largest probability is the best estimation. We introduce a feature boosting parameter θ as weights for the *true* classes during the testing process to tune the trade-off between precision and recall as in [25].

Hierarchical CRFs

Although CRFs can model multiple and long-term dependencies on the graph and may have a better performance [14] than without considering those dependencies, in practice only short-term dependencies of each vertex (i.e. a word-occurrence) are considered due to the following reasons: 1) usually we do not know what kind of long-term dependencies exist, 2) too many features will 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.

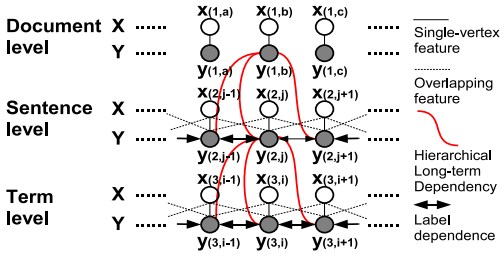


Figure 1: Illustration of Hierarchical CRFs

However, long-term dependencies at high levels may be useful to improve the accuracy of tagging tasks. For example, at the document level, biological articles have much smaller probabilities of containing chemical names and formulae. At the sentence level, sentences in different sections have different probabilities and feature frequencies of the occurrence of chemical names and formulae, e.g., references seldom contain chemical formulae. Based on these observations, we extend CRFs to introduce Hierarchical Conditional Random Fields (HCRFs) as illustrated in Figure 1. HCRFs start from the highest level to the lowest level of granularity, tag each vertex (e.g. document, sentence, or term) with labels, and use labels as features and generates new features as the interaction with low-level features. At each level, either unsupervised or supervised learning can be used.

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

$$p(\mathbf{y}_1|\mathbf{x}, \lambda_1) = \frac{1}{Z(\mathbf{x})} e^{(\sum_j \lambda_j^{(1)} F_j(\mathbf{y}_1, \mathbf{x}))}, \ ....., \quad (3)$$

$$p(\mathbf{y}_m|\mathbf{y}_1, ..., \mathbf{y}_{m-1}, \mathbf{x}, \lambda_m) = \frac{e^{\sum_j \lambda_j^{(m)} F_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x})}}{Z(\mathbf{x})}, \quad (4)$$

where $\mathbf{y}_1, ..., \mathbf{y}_{m-1}$ are the labels at the level of $1, ..., m-1$, and $F_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x})$ is the feature function that extracts a feature from the label sequences of each level $\mathbf{y}_1, ..., \mathbf{y}_m$ and the observation sequence $\mathbf{x}$. During training, the parameter λ is estimated, while during testing, $\mathbf{y}_1, ..., \mathbf{y}_{m-1}$ are estimated before $\mathbf{y}_m$ is estimated at the level m. For the level m, besides the normal features

$S_j(\mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} s_j(y_i^{(m)}, \mathbf{x}, i)$, and

$T_j(\mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} t_j(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{x}, i)$,

there are two types of features regarding the higher levels of label sequences $\mathbf{y}_1, ..., \mathbf{y}_{m-1}$, *non-interactive features*:

$S'_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y_m}|} s'_j(y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1})$ and

$T'_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y_m}|} t'_j(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1})$

which have no interaction with the observation sequence $\mathbf{x}$, and *interactive features*:

$S''_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y_m}|} s''_j(y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1}, \mathbf{x}, i)$,

$T''_j(\mathbf{y}_1, ..., \mathbf{y}_m, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y_m}|} t''_j(y_{i-1}^{(m)}, y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1}, \mathbf{x}, i)$

that interact with the observation $\mathbf{x}$. Interactive features are generated by the combination of the non-interactive features and the normal features at the level m. For example for the vertex i at the level m, $\mathbf{s}''(y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1}, \mathbf{x}, i) = \mathbf{s}'(y_i^{(m)}, \mathbf{y}_1, ..., \mathbf{y}_{m-1})\mathbf{s}^T(y_i^{(m)}, \mathbf{x}, i)$, where $\mathbf{s}'$ and $\mathbf{s}$ are state feature vectors for each vertex with sizes of $|\mathbf{s}'|$ and $|\mathbf{s}|$, and $\mathbf{s}''$ is a $|\mathbf{s}'|$ by $|\mathbf{s}|$ matrix of features.

Feature Set

For entity tagging, our algorithm extracts two categories of state features from sequences of terms: single-term features, and overlapping features from adjacent terms. Single-term features are of two types: surficial features and advanced features. Surficial features can be observed directly from the term, e.g. InitialCapital, AllCapitals, HasDigit, HasDash, HasDot, HasBrackets, HasSuperscripts, IsTermLong. Advanced features are generated by complex domain knowledge or other data mining methods, e.g. POS tags learned by natural language parsers, if a term follows the syntax of chemical formulae, etc. For chemical name tagging, we use a lexicon of chemical names collected online by ourselves and WordNet [4]. We support fuzzy matching on the Levenshtein Distance [13]. Furthermore, we check if a term has sub-terms learned from the chemical lexicon based the method in Section 3.5. For sentence tagging, feature examples are: ContainTermInList, ContainTermPattern, ContainSequentialTermPattern. For example, a list of journals, a list of names, and string patterns in reference, are used for sentence tagging. Overlapping features of adjacent terms are extracted. We used -1, 0, 1 as the window of tokens, so that for each token, all overlapping features of last token and next token are included in the feature set. For example, for *He* in "*... . He is ...*", feature($term_{n-1}$ = "." $\wedge$ $term_n$ = *initialCapital*)=*true*, and feature($term_n$ = *initialCapital* $\wedge$ $term_{n+1}$ = $isPOSTagVBZ$)=*true*, where $isPOSTagVBZ$ means that "is" is a "present tense verb, 3rd person singular". The "*He*" is likely to be an English word instead of *Helium*.

3.2 Independent Frequent Subsequence Mining

In the last subsection, we use sub-terms as features for chemical name tagging. Moreover, our system supports searches for keywords that occur as parts of terms in documents, e.g., documents with the term "methylethyl" should be returned in response to an user search for "methyl". Instead of manually discovering those chemical sub-terms by domain experts, we propose unsupervised algorithms to find them automatically from a chemical lexicon and documents.

To support such searches, our system maintains an index of sub-terms. Before it can index sub-terms, the system must segment terms into meaningful sub-terms. Indexing all possible subsequences of all chemical names would make the index too large resulting in slower query processing. Furthermore, users will not query using all possible sub-terms of queries, e.g., a query using keyword "hyl" is highly unlikely if not completely improbable, while searches for "methyl" can be expected. So, indexing "hyl" is a waste. Therefore, "methyl" should be in the index, while "hyl" should not. However, our problem has additional subtlety. While "hyl" should not be indexed, another sub-term "ethyl" that occurs independently of "methyl" should be indexed. Also, when the user searches for "ethyl", we should not return the documents that contain "methyl". Due to this subtlety, simply using the concepts of closed frequent subsequences and maximal frequent subsequences as defined in [28, 30] is not enough. We introduce the important concept of *independent frequent subsequence*. Our system attempts to identify independent frequent subsequences and index them.

In the rest of this paper, we use"sub-term" to refer to a substring in a term that appears frequently. Before our system can index the sub-terms, it must segment chemical names like "methylethyl" automatically into "methyl" and "ethyl". The independent frequent sub-terms and their frequencies can be used in hierarchical segmentation of chemical names (explained in details in the next subsection).

First, we introduce some notations as follows:

Definition 1. **Sequence, Subsequence, Occurrence:** A sequence $s =< t_1, t_2, ..., t_m >$, is an ordered list, where each token t_i is an item, a pair, or another sequence. L_s is the length of 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., $Occur_{s' \preceq s,i,j} =< t_i, ..., t_j >$, is an instance of s' in s. We say that in s, $Occur_{s' \preceq s,i,j}$ and $Occur_{s'' \preceq s,i',j'}$ overlap, i.e. $Occur_{s' \preceq s,i,j} \cap Occur_{s'' \preceq s,i',j'} \neq \emptyset$, iff $\exists n, i \leq n \leq j \wedge i' \leq n \leq j'$. *Unique* occurrences are those without overlapping.

Definition 2. **Support:** Given a 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 3. **Frequent Subsequence:** $Freq_{s' \preceq s}$ is the frequency of s' in s, i.e. the count of all unique $Occur_{s' \preceq s}$. A subsequence s' is in the set of *frequent* subsequences FS, i.e. $s' \in FS$, if $\sum_{s \in D_{s'}} Freq_{s' \preceq s} \geq Freq_{min}$, where $Freq_{min}$ is a threshold of the minimal frequency.

However, usually a sub-term is a frequent subsequence, but not for inverse, since all subsequences of a frequent subsequence are frequent, e.g. "methyl" (-CH3) is a meaningful sub-term, but "methy" is not, although it is frequent too. Thus, mining frequent subsequences results in much redundant information. We extend two concepts from previous work [28, 30] to remove redundant information:

Definition 4. **Closed Frequent Subsequence:** A frequent subsequence s is in the set of *closed* frequent subsequences CS, iff there does not exist a frequent supersequence s' of s such that all the frequencies of s in $s'' \in D$ is the same as that of s' in s'' for all $s'' \in D$. That is, $s \in CS = \{s | s \in FS \; and \; \nexists s' \in FS \; such \; that \; s \prec s' \wedge \forall s'' \in D, Freq_{s' \preceq s''} = Freq_{s \preceq s''}\}$.

Definition 5. **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. $s \in MS = \{s | s \in FS \; and \; \nexists s' \in FS \; such \; that \; s \prec s'\}$.

The example in Figure 3 demonstrates the set of *Closed Frequent Subsequences*, CS, and *Maximal Frequent Subsequences*, MS. Given the collection D=*{abcde, abcdf, aba, abd, bca}* and $Freq_{min}$=2, support D_{abcd}=*{abcde, abcdf}*, D_{ab}=*{abcde, abcdf, aba, abd}*, etc. The set of frequent subsequence *FS={abcd, abc, bcd, ab, bc, cd}* has much redundant information. *CS={abcd, ab, bc}* removes some redundant information, e.g. *abc/bcd/cd* only appears in *abcd*. *MS={abcd}* removes all redundant information as well as useful information, e.g., *ab* and *bc* have occurrences excluding those in *abcd*. Thus, we need to determine if *ab* and *bc* are still frequent excluding occurrences in *abcd*. 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 $Freq_{min}$. If a subsequence s has more than one frequent super-sequences, then all the occurrences of those super-sequences are excluded to count the independent frequency of s, $IFreq_s$.

Algorithm: IFSM(D,$D_{\forall s}$,$O_{\forall s}$,$Freq_{min}$,L_{min}):
Input:
Candidate set of sequences D,
set $D_{\forall s}$ including the support D_s for each subsequence s,
set $O_{\forall s}$ including all $Occur_{s \in D}$,
minimal threshold value of frequency $Freq_{min}$, and
minimal length of subsequence L_{min}.
Output:
Set of Independent Frequent Subsequences IS, and
Independent Frequency $IFreq_s$ of each $s \in IS$.
1. **Initialization**: $IS = \{\emptyset\}$, length $l = max_s(L_s)$.
2. **while** $l \geq L_{min}$, do
3. put all $s \in D$, $L_s = l$,
$\sum_{s' \in D_s} Freq_{s \preceq s'} \geq Freq_{min}$ into Set S;
4. **while** $\exists s \in S, \sum_{s' \in D_s} Freq_{s \preceq s'} \geq Freq_{min}$, do
5. move s with the largest $Freq_{s \preceq s'}$ from S to IS;
6. **for each** $s' \in D_s$
7. **for each** $Occur_{s'' \preceq s'} \cap (\cup_{s \preceq s'} Occur_{s \preceq s'})^* \neq \emptyset$,
8. $Freq_{s'' \preceq s'} - -$;
9. $l - -$;
10.**return** IS and $IFreq_{s \in IS} = Freq_s$;
$^*\cup_{s \preceq s'} Occur_{s \preceq s'}$ is the range of all $Occur_{s \preceq s'}$ in s', except $Occur_{s \preceq s'} \cap Occur_{t \preceq s'} \neq \emptyset \wedge t \in IS$.

Figure 2: Algorithm 1: Independent Frequent Subsequences Mining

Input Sequences: $D = \{abcde, abcdf, aba, abd, bca\}$,
Parameters: $Freq_{min} = 2$, $L_{min} = 2$
$\mathbf{l = 5}$, $Freq_{abcde} = Freq_{abcdf} = 1$, $IS = \emptyset$;
$\mathbf{l = 4}$, $Freq_{abcd} = 2$, $IS = \{abcd\}$; now for $s = abc/bcd/ab/bc/cd$, $Freq_{s \prec abcde} - - = 0$, $Freq_{s \prec abcdf} - - = 0$;
for $s = cde/de$, $Freq_{s \preceq abcde} - - = 0$;
for $s = cdf/df$, $Freq_{s \preceq abcdf} - - = 0$;
$\mathbf{l = 3}$, all $Freq < 2$, $IS = \{abcd\}$;
$\mathbf{l = 2}$, $Freq_{ab} = Freq_{ab \prec aba} + Freq_{ab \prec abd} = 2$, but $Freq_{bc} = Freq_{bc \preceq bca} = 1$, so $IS = \{abcd, ab\}$;
Return: $IS = \{abcd, ab\}$, $IFreq_{abcd} = 2$ & $IFreq_{ab} = 2$.

Figure 3: Illustration of Independent Frequent Subsequence Mining

Thus, in D, $abcd$ is frequent and ab is frequent independently, but bc is not, since ab occurs twice independently, but bc only once independently. Thus, we get a new set of *independent frequent subsequences* $\{abcd, ab\}$. We define independent frequent subsequences below:

Definition 6. **Independent Frequent Subsequence:** A frequent subsequence s is in the set of *independent* frequent subsequences IS, iff the *independent frequency* of s, $IFreq_s$, i.e. the total frequency of s, excluding all the occurrences of its independent frequent super-sequences $s' \in IS$, is at least $Freq_{min}$. That is $s \in IS = \{s | s \in FS$ and $IFreq_s \geq Freq_{min}\}$, where $IFreq_s = \sum_{s'' \in D} \#Occur_{s \preceq s''}, \forall s' \in IS$, $\forall s'' \in D, \nexists Occur_{s' \preceq s''} \cap Occur_{s \preceq s''} \neq \emptyset$ and $\#Occur_{s \preceq s''}$ is the number of unique occurrences of s in s''.

Why is computing CS or MS not sufficient for our application? Consider the case of the sub-terms "methyl" and "ethyl". Both are independent frequent subsequences in chemical texts, but not a closed or maximally frequent subsequence. For example, for the chemical name in Figure 4, "methyl" occurs twice and "ethyl" occurs once *independently*. Assume in the collection of names D, "methyl" occurs 100 times, while "ethyl" occurs 80 times independently. In this case, "ethyl" is not discovered in MS since it has a frequent super-sequence "methyl". In CS, "ethyl" occurs 180 times, since for each occurrence of "methyl", a "ethyl" occurs. Thus, CS over-estimates the probability of "ethyl". If a bias exists while estimating the probability of sub-terms, we cannot expect a good text segmentation result using the probability (next section).

Based on these observations, we propose an algorithm for *Independent Frequent Subsequence Mining* (IFSM) from a collection of sequences in Figure 2 with an example in Figure 3. This algorithm considers sequences from the longest sequence s to the shortest sequence, checking if s is frequent. If $Freq_s \geq Freq_{min}$, 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 left occurrences of a subsequence s' still make s' frequent, then put s' into IS. After mining independent frequent sub-terms, the discovered sub-terms can used as features in CRFs, and the independent frequencies $IFreq$ can be used to estimate their probabilities for hierarchical text segmentation in next section.

3.3 Hierarchical Text Segmentation

We propose an unsupervised hierarchical text segmentation method that first uses segmentation symbols to segment chemical names into terms (HTS in Figure 5), and then utilizes the independent frequent substrings discovered for further segmentation into sub-terms (DynSeg in Figure 6). DynSeg finds the best segmentation with the maximal probability that is the product of probabilities of each substring under the assumption that all substrings are independent. After mining the independent frequent substrings to estimate the independent frequency of each substring s, the algorithm estimates its probability by

$$P(s) = \frac{IFreq_s}{\sum_{s' \in IS} IFreq_{s'}} \propto IFreq_s. \quad (5)$$

For each sub-term t with $L_t = m$, a segmentation

$$seg(t) =< t_1, t_2..., t_m > \rightarrow < s_1, s_2..., s_n > \quad (6)$$

is to cluster adjacent tokens into n subsequences, where $n = 2$ for hierarchical segmentation. This is the same as the case that the number of two is applied for hierarchical clustering without prior knowledge of cluster numbers at each level [31]. The probability of segmentation is

$$P(seg(t)) = \prod_{i \in [1,n]} P(s_i), \quad (7)$$

and the corresponding log-likelihood is

$$L(seg(t)) = \sum_{i \in [1,n]} log(P(s_i))). \quad (8)$$

Maximum likelihood is used to find the best segmentation,

$$seg(t) = argmax_{seg(t)} \sum_{i \in [1,n]} log(P(s_i)). \quad (9)$$

DynSeg uses dynamic programming in text segmentation [24] (Figure 6) for optimization to maximize the log-likelihood.

In practice, instead of segmenting text into n parts directly, usually hierarchical segmentation of text is utilized and at each level a text string is segmented into two parts. This is due to two reasons: 1) It is difficult to find a proper n, and 2) A text string usually has hierarchical semantic meanings. For example, "methylethyl" is segmented into "methyl"

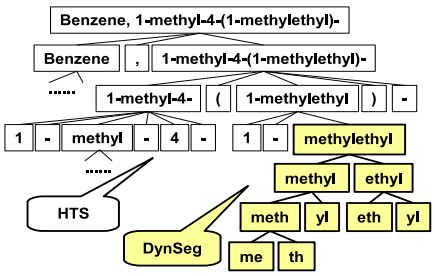


Figure 4: An Example of Hierarchical Text Segmentation

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**;

Figure 5: Algorithm 2: Hierarchical Text Segmentation

and "ethyl", and then "methyl" into "meth" and "yl", "ethyl" into "eth" and "yl", where "meth" means "one", "eth" means "two", and "yl" means "alkyl".

4. MOLECULE INFORMATION INDEXING AND SEARCH

We discuss two issues in this section: indexing schemes and query models with corresponding ranking functions. For indexing schemes, we focus on how to select tokens for indexing instead of indexing algorithms.

4.1 Index Construction and Pruning

Previous work has shown that small indexes that fit into the main memory have much better query response times [7]. We propose two methods of index pruning for the chemical name and formula indexing respectively. For chemical formula indexing, we proposed the strategy in [25].

For chemical name indexing, a naive way of index construction for substring search is to index each character and its position, and during substring search, like phrase search in information retrieval, chemical names containing all the characters in the query string are retrieved and verified whether the query substring appears in the returned names. Even though the index size is quite small using this approach, it is not used due to two reasons: 1) Each character has too many matched results to return for verification, which increases the response time of queries; 2) a matched substring may not be meaningful after text segmentation,

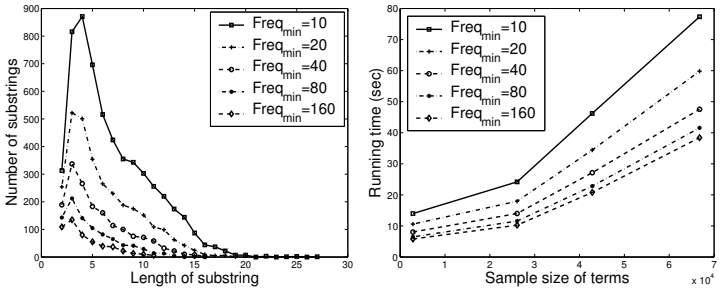
Algorithm: DynSeg(t,IF,r,n):
Input:
A sequence $t =< t_1, t_2..., t_m >$,
a set of independent frequent strings IF with corresponding independent frequency $IFreq_{s' \in IF}$,
the tree root r, and
the number of segments n.
Output:
The tree root r with a tree representation of s.
1. **if** $L_s = 1$ **return**;
2. Compute all $log(IFreq_{s_i}) = log(IFreq_{<t_j, t_{j+1}, ...t_k>})$, $1 \le j < k \le m$, where $s_i =< t_j, t_{j+1}, ...t_k >$ is a subsequence of t.
3. Let $M(l,1) = log(IFreq_{<t_1,t_2,...,t_l>})$, where $0 \le l \le m$. Then $M(l,L) = max_d(M(d, L-1) + log(IFreq_{<t_{d+1}, t_{d+2}..., t_l>}))$, where $L < n$ and $0 \le d \le l$. Note $log(IFreq_{<t_j, t_{j+1}..., t_k>}) = 0$, if $j > k$.
4. $M(m,n) = max_d(M(d, n-1) + log(IFreq_{<t_{d+1}, t_{d+2}..., t_l>}))$, where $1 \le d \le l$.
5. segment s into subsequences $< s_1, s_2..., s_n >$ using the corresponding $seg(t)$ for $M(m,n)$.
6. **if** only one $s' \in \{s_1, s_2..., s_n\} \neq \emptyset$ **return**;
7. put each $s' \in \{s_1, s_2..., s_n\} \wedge s' \neq \emptyset$ 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);

Figure 6: Algorithm 3: Text Segmentation by Dynamic Programming

e.g. "methyl" (-CH3) should not be returned when searching for chemical names containing "ethyl" (-C2H5). These indexed tokens can support similarity searches and most meaningful substring searches as shown in the experiment results in Section 5.4. As mentioned in Subsection 3.3, typically, users use meaningful substring searches. Thus, for a chemical name string, "methylethyl", indexing "methyl" and "ethyl" is enough, while "hyleth" is not necessary. Hence, after hierarchical text segmentation, the algorithm needs to index substrings at each node on the segmentation tree. This reduces the index size tremendously (in comparison with indexing all possible substrings), and our index scheme allows the retrieval of most of the meaningful substrings. If only strings at the high levels are indexed, then nothing is returned when searching for string at lower levels. If only strings at the lowest levels are indexed, too many candidates are returned for verification. Thus, at least, we need to index strings at several appropriate levels. Since the number of all the strings on the segmentation tree is no more than twice of the number of leaves, indexing all the strings is a reasonable approach.

4.2 Query Models and Ranking Functions

Features based on selected subsequences (substrings in names and partial formulae in formulae) should be used as tokens for search and ranking. Extending our previous work [25], we propose three basic types of queries for chemical name search: *exact name search*, *substring name search*, and *similarity name search*. Usually only exact name search and substring name search are supported by current chemistry databases [2]. We have designed and implemented novel ranking schemes by adapting *term frequency and inverse document frequency*, i.e. *TF.IDF*, to rank retrieved chemical entities, i.e., names and formulae.



(a) Discovered Sub-terms (b) Algorithm Running Time

Figure 7: Independent Frequent Subsequence Mining for chemical names

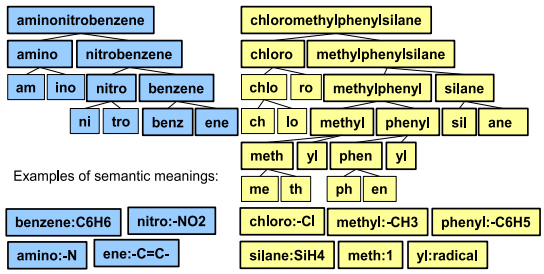


Figure 8: Examples of Hierarchical Text Segmentation. Strings in rectangles with bold boundary lines have semantic meanings.

Definition 7. **SF.IEF:** Given the collection of entity C, a query q and an entity $e \in C$, $SF(s,e)$ is the *subsequence frequency* for each subsequence $s \preceq e$, 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

$$SF(s,e) = \frac{freq(s,e)}{|e|}, IEF(s) = log\frac{|C|}{|\{e|s \preceq e\}|}, \quad (10)$$

where $freq(s,e)$ is the frequency of s in e, $|e| = \sum_k freq(s_k, e)$ is the total frequency of all selected subsequences in e, $|C|$ is the total number of entities in C, and $|\{e|s \preceq e\}|$ is the number of entities that contain subsequence s. An entity refers to a chemical name or formula.

Exact name search

An *exact name search* query returns chemical names with documents where the exact keyword appears.

Substring name search

Substring name searches return chemical names that contain the user-provided keyword as a substring with documents containing those names. If the query string is indexed, the results are retrieved directly. Otherwise, the query string is segmented hierarchically, then substrings at each node are used to retrieve names in the collection, and the intersection of returned results are verified to check if the query are contained. This happens rarely since most frequent substrings are indexed. The ranking function of substring name search regarding a query q and a name string e is given as

$$score(q,e) = SF(q,e)IEF(q)/\sqrt{|e|}. \quad (11)$$

Similarity name search

Similarity name searches return names that are similar to the query. However, the edit distance for similarity measurement is not used for two reasons: 1) Computing edit distances of the query and all the names in the data set is computationally expensive, so a method based on indexed features of substrings is much faster and feasible in practice. 2) Chemical names with similar structures may have a large edit distance. Our approach is feature-based similarity search, where substring features are used to measure the similarity. We design a ranking function based on indexed substrings, so that the query is processed and the ranking score is computed efficiently. Similar to substring search, first a query string is segmented hierarchically, then substrings at each node are used to retrieve names in the collection, and scores are computed and summed up. Longer substrings are given more weight for scoring, and scores of names are normalized by their total frequency of substrings.

Table 1: The most frequent sub-terms at each length, $Freq_{min} = 160$

String	Freq	Meaning	String	Freq	Meaning
tetramethyl	295		hydroxy	803	?-OH
tetrahydro	285		methyl	1744	?-CH3
trimethyl	441		ethyl	1269	?-CH2-CH3
dimethyl	922	H3C-?-CH3	thio	811	?-S-?
			tri	2597	three
			di	4154	two

The ranking function is given as

$$score(q,e) = \sum_{s \preceq q} W(s)SF(s,q)SF(s,e)IEF(s)/\sqrt{|e|}, \quad (12)$$

where $W(s)$ is the weight of s, defined as the length of s.

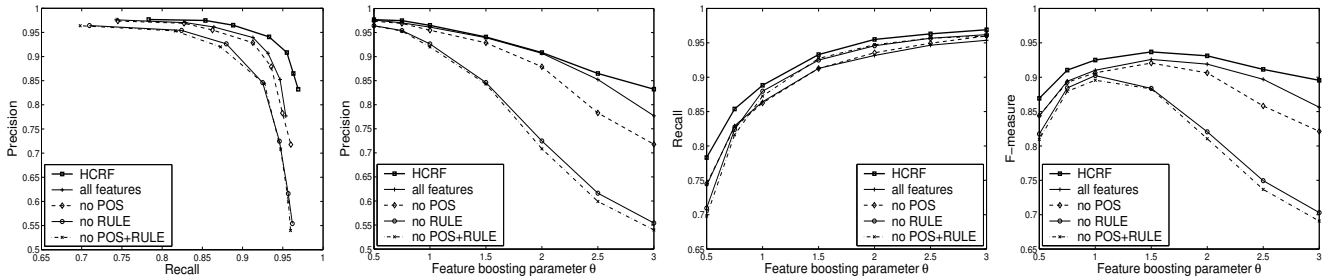
The chemical formula searches are similar and presented in our prior work [25]. *Conjunctive searches* of the basic name/formula searches are supported, so that users can define various constraints to search for desired names or formulae. When a user inputs a query that contains chemical name and formula searches as well as other keywords, the whole process involves two stages: 1) name/formula searches are executed to find desired names and formulae, and 2) returned names and formulae as well as other keywords are used to retrieve related documents. We use *TF.IDF* as the ranking function in the second stage, and the ranking scores of each returned chemical name/formula in the first stage are used as term weights to multiple the *TF.IDF* of each term when computing the ranking score in the second stage.

5. EXPERIMENTAL EVALUATION

In this section, our proposed methods are examined. We present and discuss corresponding experimental results.

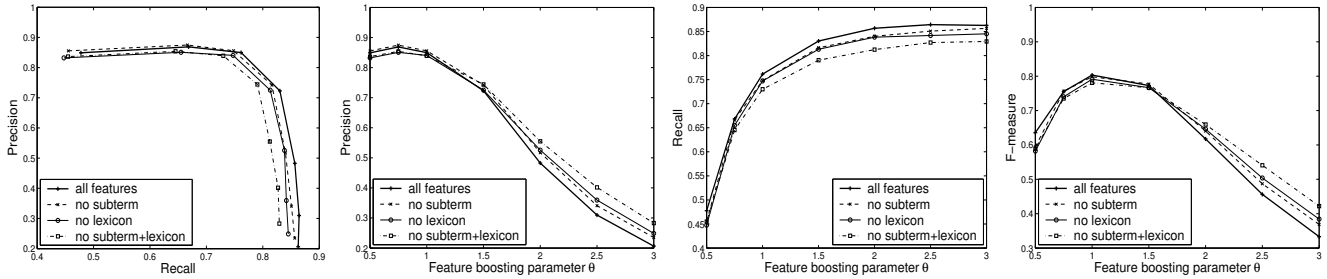
5.1 Independent Frequent Subsequence Mining and Hierarchical Text Segmentation

Since our chemical name tagging uses data mining with features of domain knowledge. Some features are based on a chemical lexicon. To construct a lexicon of chemical names, we collected 221,145 chemical names from multiple chemical webpages and online databases. We evaluate our algorithm of IFSM using this lexicon with different threshold values $Freq_{min} = \{10, 20, 40, 80, 160\}$. We first tokenize chemical names and get 66769 unique terms. Then we discover frequent sub-terms from them. The distribution of sub-term's



(a) Avg precision v.s. recall (b) Average precision (c) Average recall (d) Average F-measure

Figure 9: Chemical formula tagging using different values of feature boosting parameter θ



(a) Avg precision v.s. recall (b) Average precision (c) Average recall (d) Average F-measure

Figure 10: Chemical name tagging using different values of feature boosting parameter θ

Table 2: Sentence tagging average accuracy

Method	Recall	Precision	F
CRF,$\theta = 1.0$	78.75%	89.12%	83.61%

Table 3: Formula tagging average accuracy

Method	Recall	Precision	F
String Pattern Match	98.38%	41.70%	58.57%
CRF,$\theta = 1.0$	86.05%	96.02%	90.76%
CRF,$\theta = 1.5$	**90.92%**	**93.79%**	**92.33%**
HCRF,$\theta = 1.0$	88.63%	96.29%	92.30%
HCRF,$\theta = 1.5$	**93.09%**	**93.88%**	**93.48%**

lengths with different values of $Freq_{min}$ are presented in Figure 7 (a) and the run time of our algorithm is shown in Figure 7 (b). Most discovered sub-terms have semantic meanings in Chemistry. We show some of the frequent sub-terms with their meanings in Table 1. After IFSM, hierarchical text segmentation is tested using the same lexicon and tagged chemical names from online documents, and examples of the results are shown in Figure 8.

5.2 Chemical Entity Tagging

The data to test tagging chemical entities is randomly selected from publications crawled from the Royal Society of Chemistry [3]. 200 documents are selected randomly from the data set, and a part of each document is selected randomly to construct the training set manually. The training set is very imbalanced because of the preponderance of non-chemical terms. We first apply CRF and 10-fold cross-validation for sentence tagging. We manually construct the training set by labeling each sentence as *content* (content of the documents) or *meta* (document meta data, including titles, authors, references, etc.). Results for sentence tagging are presented in Table 2. Then sentence tags can be used as features for chemical entity tagging at the term level.

For chemical formula tagging, we use 10-fold cross-validation and a 2-level HCRF, where the two levels are the sentence level and the term level. The sentence tags are used as features for chemical formula tagging. At the term level, we label each token as a *formula* or a *non-formula*. Several methods are evaluated, including rule-based String Pattern Match, 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 other features. Four combinations are tested: (1) all features, (2) no POS, (3) no RULE, and (4) no POS or RULE. Results of chemical formula tagging are shown in Table 3 and Figure 9. We can see that the RULE features contribute more than the POS features and HCRF has the highest contribution. In comparison to CRFs, HCRFs only has an improvement of 1.15%. However, since the total error rate is just 7.67%, the improvement is about 15% of the total error rate of 7.67%. This means long-dependence features at the sentence level has positive contributions. HCRF has a better performance than CRF, but increases the runtime. Both for HCRF and CRF using all features, the best *F-measure* is reached when $\theta = 1.5$, and in this case recall and precision are balanced. We show the results using all features in Table 3.

For chemical name tagging, since a name may be a phrase of several terms, we label each token as a *B-name* (beginning of a name), or *I-name* (continuing of a name), or a *non-name*. We use 5-fold cross-validation and CRF with different feature sets. Features are classified into three subsets: features using frequent sub-terms (*sub-term*), features using lexicons of chemical names and WordNet [4] (*lexicon*),

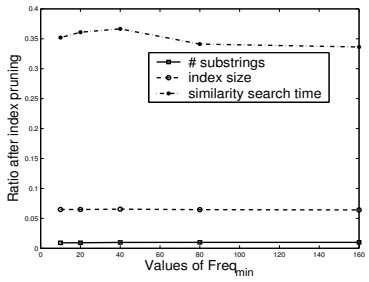
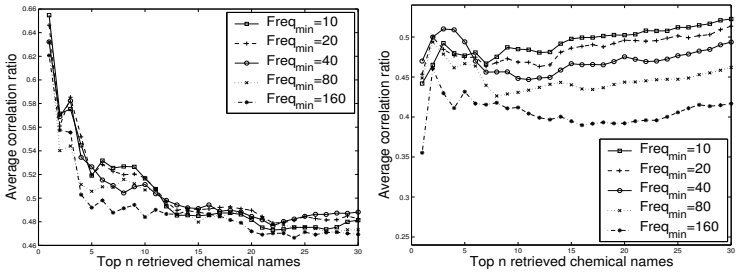


Figure 11: Ratio of after v.s. before index pruning



(a) Similarity name search (b) Substring name search

Figure 12: Correlation of name search results before and after index pruning

Table 4: Name tagging average accuracy

Method, $\theta = 1.0$	Recall	Precision	F
all features	**76.15%**	84.98%	**80.32%**
no sub-term	74.82%	**85.59%**	79.84%
no lexicon	74.73%	84.05%	79.11%
no sub-term+lexicon	73.00%	83.92%	78.08%

and other features. Four combinations are tested: (1) all features, (2) no sub-term, (3) no lexicon, and (4) no sub-term or lexicon. We test different values {0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0} for the feature boosting parameter θ for the formula (or B-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. To measure the overall performance, we use $F = 2PR/(P + R)$ [18], where P is precision and R is recall, instead of using error rate, since it is too small for imbalanced data. Results of chemical name tagging are shown in Table 4 and Figure 10. We can see using all features has the best recall and F-measure, and using features of frequent sub-terms can increase recall and F-measure but decrease precision. Our system clearly outperforms the tool for chemical entity tagging: Oscar3 [5] on the same datasets. Oscar3 has a *recall* of 70.1%, *precision* of 51.4%, and the *F-measure* is 59.3%.

We only evaluate HCRFs for chemical formula tagging, because we found that ambiguous terms, like "C", "He", "NIH", appearing frequently in parts containing document meta data, have different probabilities to be chemical formulae. For example,"C" is usually a name abbreviation in the part of authors and the references, but usually refers to "Carbon" in the body of document. Thus, HCRFs is evaluated for chemical formula tagging to check whether the long dependence information at the sentence level is useful or not. HCRFs are not applied for chemical name tagging, because we found that the major source of errors in chemical name tagging is not due to term ambiguity, but due to limitations of the chemical dictionary and incorrect text tokenizing.

5.3 Chemical Name Indexing

For chemical name indexing and search, we use the segmentation based index construction and pruning. We compare our approach with the method using all possible substrings for indexing. We use the same collection of chemical names in Section 5.1. We split the collection into two subsets. One is for index construction (37,656 chemical names), while the query names are randomly selected from the other subset. Different values of the frequency threshold $Freq_{min} = \{10, 20, 40, 80, 160\}$ to mine independent frequent substrings are tested. The experiment results in Figure 11 show that most (99%) substrings are removed after hierarchical text segmentation, so that the index size decreases correspondingly (6% of the original size left).

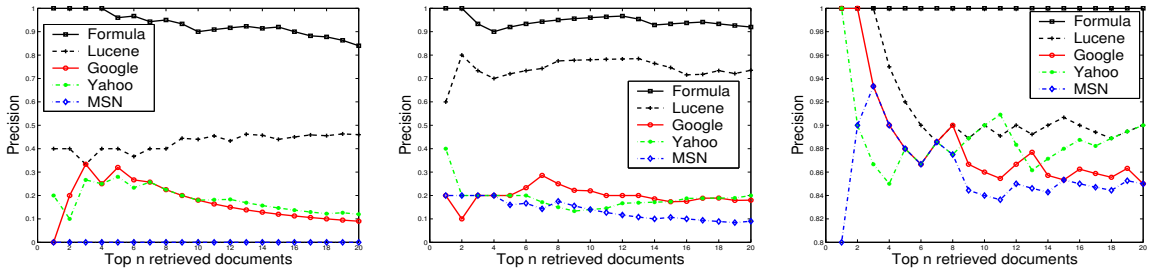
5.4 Chemical Name Search

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 sub-terms with the length 3-10 discovered in Section 5.1. We also evaluated the response time for similarity name search, illustrated in Figure 11. The method using HTS only requires 35% of the time for *similarity name search* compared with the method using all substrings. However, we did not test the case where the index using all substrings requires more space than the main memory. In that case, the response time will 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. 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 that is defined as $Corr_n = |R_n \cap R'_n|/n, n = 1, 2, 3, ...$, where R_n and R'_n are the search results of before and after index pruning, respectively. Results are presented in Figure 12. We can observe 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.

5.5 Term Disambiguation

To test the ability of our approach for term disambiguation in documents, we index 5325 PDF documents crawled online. We then design 15 queries of chemical formulae. We categorize them into three levels based on their ambiguity, 1) hard (*He, As, I, Fe, Cu*), 2) medium (*CH4, H2O, O2, OH, NH4*), and 3) easy (*Fe2O3, CH3COOH, NaOH, CO2, SO2*). We compare our 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.



(a) Hard Queries (b) Medium Queries (c) Easy Queries

Figure 13: Average Precision in Document Search using Ambiguous Formulae

Lucene [1] is used for the traditional approach. We also evaluate generic search engines, Google, Yahoo, and MSN using those queries to show the ambiguity of terms. Since the indexed data are different, they are not comparable with the results of our approach and Lucene. Their results illustrate that ambiguity exists and the domain-specific search engines are desired. From Figure 13, we can observe 1) the ambiguity of terms is very serious for short chemical formulae, 2) results of Google and Yahoo are more diversified than that of MSN, so that chemical web pages are included, and 3) our approach out-performs the traditional approach based on Lucene, especially for short formulae.

6. CONCLUSIONS AND FUTURE WORK

We evaluated CRFs for chemical entity extraction and proposed an extended model HCRFs. Experiments show that CRFs perform well and HCRFs perform better. Experiments illustrate that most of the discovered sub-terms in chemical names using our algorithm of IFSM have semantic meanings. Examples show our HTS method works well. Experiments also show that our schemes of index construction and pruning can reduce indexed tokens as well as the index size significantly. Moreover, the response time of similarity name search is considerably reduced. Retrieved ranked results of similarity and substring name search before and after segmentation-based index pruning are highly correlated. We also introduced several query models for name searches with corresponding ranking functions. Experiments show that the heuristics of the new ranking functions work well. In the future, a detailed user study is required to evaluate the results. Entity fuzzy matching and query expansion among synonyms will also be considered.

7. ACKNOWLEDGMENTS

We acknowledge partial support from NSF grant 0535656.

8. REFERENCES

[1] Apache lucene. http://lucene.apache.org/.
[2] Nist chemistry webbook. http://webbook.nist.gov/chemistry.
[3] Royal society of chemistry. http://www.rsc.org.
[4] Wordnet. http://wordnet.princeton.edu/.
[5] World wide molecular matrix. http://wwmm.ch.cam.ac.uk/wikis/wwmm.
[6] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proc. WWW*, 2007.
[7] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving web search efficiency via a locality based static pruning method. In *Proc. WWW*, 2005.
[8] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proc. SIGKDD*, 1998.
[9] D. Freitag and A. McCallum. Information extraction using hmms and shrinkage. In *AAAI Workshop on Machine Learning for Information Extraction*, 1999.
[10] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. ICDM*, 2001.
[11] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, 2001.
[12] D. L.Banville. Mining chemical structural information from the drug literature. *Drug Discovery Today*, 11(1-2):35–42, 2006.
[13] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 1966.
[14] W. Li and A. McCallum. Semi-supervised sequence modeling with syntactic topic models. In *Proc. AAAI*, 2005.
[15] G. Luo, C. Tang, and Y. li Tian. Answering relationship queries on the web. In *Proc. WWW*, 2007.
[16] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proc. ICML*, 2000.
[17] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proc. CoNLL*, 2003.
[18] R. McDonald and F. Pereira. Identifying gene and protein mentions in text using conditional random fields. *Bioinformatics*, 6(Suppl 1):S6, 2005.
[19] P. Murray-Rust, H. S. Rzepa, S. Tyrrell, and Y. Zhang. Representation and use of chemistry in the global electronic age. *Org. Biomol. Chem.*, 2004.
[20] P. Murray-Rust, H. S. Rzepa, and M. Wright. Development of chemical markup language (cml) as a system for handling complex chemical content. *New J. Chem.*, 2001.
[21] J. W. Raymond, E. J. Gardiner, and P. Willet. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002.
[22] B. Settles. Abner: an open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.
[23] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proc. PODS*, 2002.
[24] B. Sun, P. Mitra, H. Zha, C. L. Giles, and J. Yen. Topic segmentation with shared topic detection and alignment of multiple documents. In *Proc. SIGIR*, 2007.
[25] B. Sun, Q. Tan, P. Mitra, and C. L. Giles. Extraction and search of chemical formulae in text documents on the web. In *Proc. WWW*, 2007.
[26] W. J. Wilbur, G. F. Hazard, G. Divita, J. G. Mork, A. R. Aronson, and A. C. Browne. Analysis of biomedical text for chemical names: A comparison of three methods. In *Proc. AMIA Symp.*, 1999.
[27] P. Willet, J. M. Barnard, and G. M. Downs. Chemical similarity searching. *J. Chem. Inf. Comput. Sci.*, 38(6):983–996, 1998.
[28] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *Proc. SIGKDD*, 2003.
[29] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based substructure similarity search. *ACM Transactions on Database Systems*, 2006.
[30] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proc. SIGKDD*, 2004.
[31] Y. Zhao and G. Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 2005.