Identifying, Indexing, and Ranking Chemical Formulae and Chemical Names in Digital Documents

Bingjun Sun, Prasenjit Mitra, C. Lee Giles, Karl T. Mueller
The Pennsylvania State University

End users use chemical search engines to search for chemical formulae and chemical names. Chemical search engines identify and index chemical formulae and chemical names appearing in text documents to support efficient search and retrieval in the future. Identifying chemical formulae and chemical names in text automatically has been a hard problem that has met with varying degrees of success in the past. We propose algorithms for chemical formula and chemical name tagging using Conditional Random Fields (CRFs) and Support Vector Machines (SVMs) that achieve higher accuracy than existing (published) methods. After chemical entities have been identified in text documents, they must be indexed. In order to support user-provided search queries that require a partial match between the chemical name segment used as a keyword or a partial chemical formula, all possible (or a significant number of) sub-formulae of formulae that appear in any document and all possible sub-terms (e.g., 'methyl') of chemical names (e.g., 'methylethyl ketone') must be indexed. Indexing all possible sub-formulae and sub-terms results in an exponential increase in the storage and memory requirements, and the time taken to process the indices. We propose techniques to prune the indices significantly without reducing the quality of the returned results significantly. Finally, we propose multiple query semantics to allow users to pose different types of partial search queries for chemical entities. We demonstrate empirically that our search engines improve the relevance of the returned results for search queries involving chemical entities.

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*; H.4.0 [**Information Systems Applications**]: General; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*; J.2 [**Physical Sciences and Engineering**]: Chemistry

General Terms: Algorithms, Design, Experimentation, Documentation

Additional Key Words and Phrases: Chemical name, chemical formula, entity extraction, conditional random fields, independent frequent subsequence, hierarchical text segmentation, index pruning, query models, similarity search, ranking

Author's address: B. Sun, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, Email: sunbingjun@gmail.com. P. Mitra, College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, Email: pmitra@ist.psu.edu. C. Lee Giles, College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, Email: giles@ist.psu.edu. Karl T. Mueller, Department of Chemistry, The Pennsylvania State University, University Park, PA 16802, Email:ktm2@psu.edu

1. INTRODUCTION

End-users demand fast responses to searches for chemical entities, like chemical formulae and chemical names, over large corpuses of documents. A chemical search engine must identify the occurrences of all instances of chemical entities[1] appearing in text documents and index them in order to enable fast access. The processing and indexing of the documents are conducted off-line. However, for large document corpuses or digital libraries we still require reasonable processing and indexing times.

Tagging chemical formulae and chemical names is a hard problem because of the inherent ambiguity in natural language text. Corbett, Batchelor, and Teufel indicate that inter-annotator agreement (F-score) for a task of tagging named chemical entities in full-text chemistry papers is 93% [Corbett et al. 2007]. We show that using supervised machine-learning-based algorithms, chemical names and chemical formulae can be tagged with reasonably high accuracy. Our system tags chemical name and formulae using Conditional Random Fields(CRFs), a discriminative, probabilistic, undirected graphical model that has been widely used for labeling sequential data like text documents or biological sequences) [Lafferty et al. 2001].

Consider a researcher in environmental chemical kinetics who wishes to search for papers on the interactions of sodium acetate (also known as the sodium salt of acetic acid) with an aluminosilicate surface at different values of solution acidity (given in pH units). Many entities of chemical interest are involved, including the acetate ion, which may be protonated or deprotonated depending on pH, the sodium cations, and the aluminosilicate surface which contains (among other entities): Al-OH groups, Si-OH groups, various protonated or deprotonated versions of these sites (depending on pH), and specific Si-O-Al, Si-O-Si, and possibly Al-O-Al linkages. Chemical entity search on these species run from the simple (sodium cations) to the more complex $Si - O - H_2^+$ groups at a specific pH). In our search engine, the chemist can pose such searches using chemical names, chemical formulae and other keywords.

Chemists and other users of chemical search engines may desire to input a partial forumla or a part of a chemical name [2] They expect that the search engine

[1] In our implemented search engine, we have not handled any other chemical entity like CAS, InChI, and SMILES, but only focused on chemical names and chemical formulae although we believe that supervised-learning-based algorithms, like Conditional Random Fields (CRF) [Lafferty et al. 2001], can be trained to tag other types of chemical entities. Thus, when we use the term "chemical entity" in the rest of the paper for the sake of brevity, we refer to only "chemical names and formulae". We are not making any substantiated claims about the tagging of any other types of chemical entities.

[2] Several search engines support partial chemical name searches, e.g., the search engines associated with eChemPortal, http://webnet3.oecd.org/echemportal/Contents.aspx?ContentName=PortalInfo, ChemFinder, http://chemfinder.camsoft.com/chembiofinder/Forms/Home/ContentArea/Home.aspx, the California Department of Pesticide Regulation, http://www.cdpr.ca.gov/docs/chemical/monster2.htm, and the ChemSynthesis database, http://www.chemsynthesis.com/text-search.html, supports partial search for both chemical names and formulae.

should return documents having chemical entities that contain the partial formula or chemical name. For example, a user may search with CH_3 and expect the search engine to return a ranked set of documents with chemical formula containing the partial formula CH_3. Another example, as outlined in eChemPortal, is when a user searches for “tert-butylphenol”. The search engine should return the set of documents that contain the term “tert-butylphenol”. Search engines use indexes to enable efficient, real-time query answering [Manning et al. 2008]. In order to support such partial term searches efficiently, a chemical search engine can index partial terms or partial formulae in advance (before the query has been posed by the user). If the search engine has to index all possible sub-formulae of any formula appearing in a document or to index all possible sub-terms of a chemical name, the size of the index will be prohibitively large and building such an index will be prohibitively expensive both with respect to memory requirements and processing time requirements. To address this problem, we propose index pruning techniques that reduce the size of the index to a manageable size while avoiding significant degradation in the quality of the search results (see Section 6.3).

Users search for chemical formulae using various different forms of the same basic formula, e.g., they may search for CH_3COOH or for $C_2H_4O_2$. Although the majority of documents (278,000 as obtained from a major search engine) use the formula CH_3COOH, the formula $C_2H_4O_2$ appears in a substantial number of documents (40,000 from the same search engine). For larger chemical formulae, without accepted canonical forms, the diversity is even greater. When a user searches using one form of the formula, he/she may prefer all documents with the different forms of the formula to be returned. The search engine at ChemIndustry.com returns the “synonyms” of a chemical formula and the documents containing any of these synonyms. In order to build search engines that cater to this requirement, a search engine has to identify chemical formulae and understand the equivalence among chemical formulae. Our search engine identifies chemical formulae, disambiguates them from other non-chemical-entity-related abbreviations (e.g., “OH” may refer to the hydroxyl group or the state “Ohio”), and indexes them. We support multiple query semantics to allow for exact chemical formula and chemical name searches, partial searches, and fuzzy searches for similar chemicals. Identifying whether a term refers to a chemical formula or chemical name or neither is a hard problem because it requires context-sensitive, natural-language analysis. Because we intend to apply our techniques over documents crawled daily from the world-wide-web and have limited computational resources, our algorithms must be highly scalable. Hence, we have strived to reduce the size of our data structures, e.g., we show how our indices can be pruned to reduce memory requirements and improve the runtime without sacrificing the quality of the search results.

To support partial chemical name searches, our search engine also segments a chemical name into meaningful subterms (e.g., “ethyl” and “methyl” as opposed to “ethy” and ”lmethy”, etc.) automatically. Such segmentation and indexing allow end-users to perform partial name searches. Typically, tools, like Name=Struct [Brecher 1999], CHEMorph [Kremer et al. 2006], and OPSIN [Corbett and Murray-Rust 2006], segment a chemical name into its morphemes map the morphemes into their chemical structures, and use these structures to construct the structure of the named chemical. We believe that our segmentation algorithm can be used by name-to-structure generating software for segmenting a chemical name to its constituent morphemes. Our algorithm mines independent frequent substring patterns in the text corpus, and uses information about those substring patterns for chemical name segmentation, and then index the subterms obtained by segmenting the chemical name. The advantage of our algorithm over existing alternatives(see Section 2) is that we do not use dictionaries or lexicons, which are prone to being incomplete and to contain inaccuracies. Similarly, for chemical formula indexing, we propose an index pruning strategy based on a sequential feature selection algorithm that selects frequent and discriminative substrings of formulae as features to index.

End-users can query our system using queries having different types of semantics to obtain a set of documents that have exact, partial, or fuzzy occurrences of the query keyword or formula. Our system returns a ranked list of documents based on different ranking functions that we have proposed for each type of query. In response to a chemical formula search, our system suggests a set of related chemical formulae above the list of returned documents. A user can click on one of the suggested formulae to further refine the query and retrieve a set of documents pertaining to that particular formula.

Our experiments show that our chemical formula and name search engine outperforms general purpose search engines (as expected) and provides better results than Oscar3, [Corbett and Murray-Rust 2006] (version Alpha1, the latest version available at the time this work was done) or a rule-based search engine on our dataset. However, as the Netflix Prize[3], the KDD Cup 2009[4] challenges, and other research projects (for a tutorial see [Polikar 2006]) have shown, to obtain the best performance in any real-life annotation or classification task, one has to employ an ensemble of different classifiers. Our CRF-based tagging module can be a significant component of such an ensemble in conjunction with the methods utilized by Oscar3 and other related tools; therein lies the importance of this work.

We believe that constructing an accurate domain-specific search engine is a hard task due to a variety of problems; we only address a few of them. One source of problems is inaccurate format conversion, e.g., due to noisy PDF document to text conversion. Recovering from these errors in in future stages of processing is difficult.

Another problem is that chemical lexicons, like PubChem[5], are incomplete and contain inaccuracies. For example, PubChem lists methane and carbon as synonyms. If a search engine uses the synonym list from PubChem to expand queries, it will return documents containing the term carbon in response to an end-user query for methane. If a clean dictionary was available, our system can be easily extended to utilize the dictionary. In response to a user query, the dictionary could be consulted to obtain synonyms and the original query along with the synonyms can be looked up to obtain the result set.

Our search engine performs the following: 1) mines chemical formulae and chem-

[3]http://www.netflixprize.com/
[4]http://www.sigkdd.org/kddcup/index.php
[5]http://pubchem.ncbi.nlm.nih.gov/, retrieved on May 27th, 2009

ical names in text, 2) indexes chemical formulae and chemical names, and 3) supports users searching for information using chemical names and chemical formulae along with other keywords. In the first step, our algorithm tags chemical formulae and chemical names in text. In the second stage, each chemical formula and name is indexed. In the third stage, using our novel ranking functions, the search engine ranks the returned documents and returns them to the end-user. As an alternative to the steps our search engine takes, an argument can be made to convert all chemical names to their chemical structure and then enable search using structural similarity. However, a chemical name may have semantics that can be lost while transforming the name into a structure. For example, users searching for graphite and diamond may expect different results even though they may be mapped to having the same two-dimensional chemical structure.

The major contributions of our work are as follows:

—C1: We propose algorithms for chemical formula and chemical name tagging using CRFs, and compare them with those using other supervised learning methods like Support Vector Machines(SVMs) [Burges 1998]. We adapt CRFs by including parameters for decision-boundary tuning necessary to handle unbalanced entity distributions (chemical names and formulae occur significantly less frequently than non-chemical-entities in text). We use stacked Conditional Random Fields, to tag chemical formulae and chemical names utilizing the information about (a) the topics discussed in the document, (b) sentence- level features such as which sections of a document a sentence appears in, and (c)term-level features to tag chemical formula or chemical names more accurately (see Section 3.3 for details).

—C2: In order to support partial term querying, e.g., to enable a user's keyword query "methyl" to match a document with the term "methylethyl", our algorithm automatically segments chemical names and indexes them. We introduce a new concept, *independent frequent subsequences*, and propose an algorithm to mine these subsequences, in order to discover meaningful subterms in chemical names.

—C3: To reduce the size of the index, and thereby make the system more efficient, we propose an unsupervised method for *hierarchical text segmentation* and use it for chemical name index pruning. We utilize a sequential feature selection algorithm based on the frequency and discrimination power of features for chemical formula index pruning. We see that a search engine using the pruned index returns similar results as that with the full index, however, the memory requirements of the pruned index and the running time of the search system are improved significantly by the pruned index.

—C4: We present various query models for chemical formula or chemical name searches and propose corresponding ranking functions. Our system supports each of these query models.

Our chemical-entity-aware search engine is an integral part of $Chem_X Seer$[1], a digital library for chemistry. The proposed architecture of our chemical-entity-aware search engine is shown in Figure 1.

The rest of this paper is organized as follows: Section 2 reviews related works. Section 3 presents approaches to chemical entity tagging based on CRFs, stacked

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

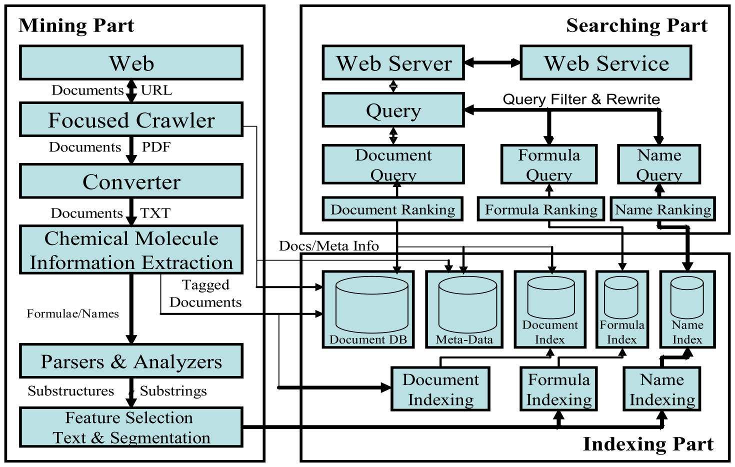


Fig. 1. Architecture of Chemical Entity Search Engine with Document Search in $Chem_X Seer$

CRFs, and SVMs. Section 4 describes the indexing schemes and related algorithms for sequential feature selection, independent frequent subsequence mining, and hierarchical text segmentation. Section 5 introduces query models, and ranking functions for name and formula searches. Section 6 presents experiments and results. Conclusions and future directions are discussed in Section 7.

2. RELATED WORK

Related work falls into two categories: 1) entity extraction from the text, and 2) indexing and searching for chemical structure information. Banville has provided a high-level overview for mining chemical structure information from the literature [Banville 2006].

2.1 Entity Extraction

Hidden Markov Models (*HMMs*) [Baum and Petrie 1966; Baum et al. 1970] are commonly used to label or segment sequences. HMMs have a conditional independence assumption where given the hidden state, observations are independent [Lafferty et al. 2001]. Thus, an HMM cannot capture the interactions between adjacent tokens. Another category of entity extraction methods is based on *Maximum Entropy* (ME) [Borthwick 1999], which introduces an exponential probabilistic model based on binary features extracted from sequences and estimate parameters using maximum likelihood. Maximum Entropy Markov Models (*MEMM*s) [McCallum et al. 2000] are also exponential probabilistic models that take the observation features as input, and output a distribution over possible next states, but they suffer from the label-bias problem where states with low entropy next-state distributions effectively ignore observations when conditioning on data. Different from directed graph-based models like HMM and MEMM, CRF [Lafferty et al. 2001] uses an undirected graphical model, which can relax the conditional independence assumption of HMMs and avoid the label-bias problem of MEMMs. It follows the maximum entropy principle [Berger et al. 1996] as in MEMMs, using exponential probabilistic models and relaxing the independence assumption to involve multiple interactions among adjacent words in text and also long-range dependencies. Models based on linear-chain CRFs have been applied for labeling sequences in many applications, such as named-entity recognition [McCallum and Li 2003], detecting biological entities, such as proteins [Settles 2005] or genes[McDonald and Pereira 2005], etc.

2.1.1 *Chemical Entity Extraction.* Although methods for converting a chemical name to chemical structure have been proposed since 1962 [Garfield 1962], automatic recognition of chemical names in natural language and an enhanced chemical-name identification algorithm were first presented by Hodge, at the American Chemical Society meeting in 1989. Unfortunately, we were unable to find any subsequent article describing these methods such that we could implement them and compare them to our methods. [Hodge et al. 1989; Hodge 1991].

Kemp and Lynch discuss a method to identify specific chemical names and do *not* extract general chemical names like "alkoxycarbonyl", "halide", "hydrocarbon", etc. from text [Kemp and Lynch 1998]. As evidenced by the most-popular-search list on chemindustry.com, a popular hub for chemical information on the web, the general "chemical name", *nylon* is the second most popular search term and another general chemical name, *silicone* is the fourth most popular term[6]. Our system tags and allows for the retrieval of general chemical names too. Their method requires substantial manual culling; we present automatic methods in this paper.

Wilbur, et al., have evaluated three methods for chemical name tagging [Wilbur et al. 1999]. The first method segments a chemical name into its morphemes and checks if the morphemes appear in a dictionary. They also evaluate two methods based on Bayesian classification schemes using n-grams. They show that one of their Bayesian classification methods using n-grams outperforms the segmentation-based method and the other Bayesian methods.

Narayanaswamy, Ravikumar, and Vijay-Shanker proposed a rule-based algorithm for identifying chemical names in text documents [Narayanaswamy et al. 2003]. Their paper does not provide the entire set of rules and features on which their system is based, and thus we cannot reproduce their system for a direct comparison. The features they mention in the paper, such as the existence of capital letters and numbers, have been used as features in our system. Our system uses additional features beyond those mentioned in their paper. We have observed that these additional features improve the precision and recall of the named entity tagging task.

Vasserman examined Wilbur's method and found that their n-gram-based approach shows "significantly lower performance" on his data [Vasserman 2004]. Vasserman's methods are completely unsupervised, and he shows that the best performance is obtained by using n-grams and Naiive-Bayes models; their precision/recall numbers, while high, are lower than that shown to be achievable by our methods. Both Wilbur's and Vasserman's methods attempt to classify only predetermined tokens unlike ours.

Other previous attempts at chemical entity tagging use both machine learning

[6]As obtained from http://www.chemindustry.com/mostpop/cpb_cont.html on July 19th, 2010

approaches [Wren 2006] and rule-based approaches[7]. Corbett and Murray-Rust proposed Oscar3, an improvement on Oscar2 [Corbett and Murray-Rust 2006]. Besides its other features, Oscar3 provides chemical entity tagging. Their unsupervised chemical name recognition algorithm is also based on an n-gram-based method like Wilbur's and Vasserman's but uses modified Knesser-Ney smoothing [Kneser and Ney 1995]. Oscar3 reports precision and recall numbers ranging from 60% to 80%. In 2008, Corbett and Copestake published a method based on using "character-based n-grams, Maximum Entropy Markov Models(MEMM) [McCallum et al. 2000], and rescoring to recognize chemical names and other such entities" [Corbett and Copestake 2008]. Their algorithm was deployed by subsequent versions of Oscar3. Corbett and Copestake report a maximum F-score of 80.7(precision of 78.7% and recall 82.9% on chemistry journal papers and a maximum F-score of 83.2% (85% precision and 81.6% recall) on PubMed abstracts. Corbett and Copestake acknowledge that CRFs suffer from a label-bias problem but they nevertheless advocate the use of MEMMs because of "advantages such as shorter training cycles". While training MEMMs are undoubtedly faster than CRFs, their testing times are comparable. Training these classifiers is done offline and with today's faster machines, training a CRF can be achieved in reasonable times as shown in this work. We advocate using a CRF because it avoids the label bias problem and improves the classification accuracy.

Our supervised method requires more human effort in training the system by needing between 100-200 tagged documents, but, in turn, we observed that on our dataset it provides better precision and recall numbers and higher accuracy with respect to the chemical entity tagging task. Oscar3 is a more mature tool than ours and can search across different representations of the same compound. Our focus is not as broad as that of Oscar3. Instead, we focus primarily on chemical formula and name tagging, indexing, and search algorithms. Our efforts will not replace Oscar3 but are complementary. In many existing problems, an ensemble classifier using multiple base technique outperforms any single classifier [Polikar 2006]. We believe that our algorithm can be used in conjunction with Oscar3 to form such an ensemble classifier to provide improved chemical formula and name tagging.

After our initial work [Sun et al. 2007; Sun et al. 2008], Klinger, et al., [Klinger et al. 2008] also obtained similar results as ours using Conditional Random Fields to detect IUPAC and IUPAC-like chemical names, however, they did not address the tagging of chemical formulae or common chemical names.

Classifiers such as SVMs [8] [Joachims 1999; Bordes et al. 2005] can be applied to tag chemical formulae. Entity tagging in text is usually an asymmetric binary classification problem on imbalanced data, where there are many more *false* samples than *true* samples, but the *precision* and *recall* of the *true* class are more important than the overall accuracy. In this case, the decision boundary may be dominated by the false samples. Several methods such as cost-sensitive classification and decision-threshold tuning have been studied for imbalanced data [Shanahan and Roma 2003]. We have observed that CRFs suffer from this problem too, since in previous work based on CRFs [McDonald and Pereira 2005; Sha and Pereira 2003], *recall* is usually

[7]GATE: http://gate.ac.uk/ and Oscar2: http://wwmm.ch.cam.ac.uk/wikis/wwmm
[8]SVM light: http://svmlight.joachims.org/

lower than *precision*. To the best of our knowledge, no methods to tune the decision boundary for CRFs exist. In this work, we address this issue.

2.2 Chemical Name Segmentation

Our search engine has to segment chemical names automatically because of the following reasons.

(1) The search engine uses the segments as features to identify chemical name segments and thereby tag a chemical entity as a chemical name.
(2) The search engine indexes the chemical name segments to enable search.

To understand why we need chemical name segmentation for chemical name search, consider, for example, the chemical name *acetaldoxime*. Our segmentation algorithm segments it into the segments *acet* and *aldoxime*. Thereafter, aldoxime is segmented into *ald* and *oxime*. Consequently, if the end-user searches for the term *aldoxime* or *oxime*, our system can return the documents referring to *acetaldoxime*. Like all search engines, we use a ranking function to determine the rank of a document containing the term "acetaldoxime". The rank of a document containing *acetaldoxime* would vary depending upon the query term (i.e., whether the query is for *acetaldoxime* or *aldoxime* or *oxime*).

We believe that the benefits of our automated segmentation method over existing methods are:

(1) Most existing methods listed above require a dictionary or list of morphemes and a set of regular expressions created manually in order to segment chemical names. Our algorithm derives the morphemes by examining chemical literature and examining the frequencies of terms and sub-terms. (For details, see the algorithm in Section 4.2.1.) Whenever new chemicals are identified that use new morphemes, these lists and dictionaries will need to be updated manually. Our algorithm is automated and eliminates the need for such manual efforts and human-input domain knowledge by using the occurrence statistics of chemical names and their morphemes in a text corpora.
(2) We generate a hierarchical segmentation. In the example above, we first segment the chemical name to *aldoxime* and then segment the *aldoxime* to *oxime*. The existing methods construct a single segmentation of the term *acetaldoxime*. For example, the CAS method segments the term to *acetal*, *d*, and *oxime*. If the document with *acetaldoxime* is indexed using this segmentation a search for *aldoxime* will not retrieve it. Our segmentation, however, allows the retrieval of *acetaldoxime* in response to the query *aldoxime*.

We now review the existing work in this area and contrast them with ours. In his seminal work, Garfield proposes a technique to construct the chemical formula from a chemical name by breaking down the chemical name into its morphemes [Garfield 1962]. Garfield's method uses a dictionary of morphemes and a method that shows how the data can be examined to identify morphemes. His method considers the right-most eight characters in a chemical name, and looks up the chemical name in a dictionary. For example, *but* in *butane* is a morpheme, while *but* in *nembutal* is not. Garfield's method identifies morphemes that can be substituted for each other and checks if the substituted chemical name occurs in the data to determine whether a string in a term is a morpheme. The string *but* in *butane* can be replaced by another morpheme *hex* to create a term that occurs in the literature like *hexane*. However, for the term *nembutal*, the corresponding term *nemhexal* does not appear in the literature. Therefore, *but* is considered a morpheme in *butane* but not in *nembutal*. Our problem of segmenting chemical names is the same as Garfield's problem of detecting morphemes from chemicals. Garfield's method suffers from the following problem. Since his algorithm attempts to match the longest string from the right to left, it would segment tribromethanol as tri-bro-methanol or tribro-methanol, whereas, the term should be segmented as tri-brom-ethanol.

Van der Stouw, et al., proposed a method to convert chemical names to chemical formula [Vander Stouw et al. 1967] used by the Chemical Abstract Service in 1967 that processes chemical names from left to right. However, their description of the algorithm remains anecdotal. They do not provide enough details for us to re-implement their algorithm. They claim that their algorithm works in about 60% of the cases and with additional studies can be enhanced to cover "80 to 85% of names of carbon-containing compounds" [Vander Stouw et al. 1967]. The Chemical Abstract Service has also published the Registry File Basic Name Segment Dictionary in 1993 [ACS 1993]. The document provides a left to right chemical name segmentation algorithm.

Cooke-Fox, Kirby, and Rayner, proposed context-free grammars for various types of organic chemical compounds and a parser based on the grammar to segment chemical names [Cooke-Fox et al. 1989]. Brecher indicated that a rigid grammar-based method does not work because people use chemical names that do not conform to the formalized rules that were proposed by Cooke-Fox, et al. [Brecher 1999]. He validated this claim by examining "chemical sites on the Internet" and "catalogs produced by commercial chemical vendors". Brecher proposed a practical system "Name=Struct". However, again the details provided in their article do not allow a full reconstruction of their method. Since the work was done at CambridgeSoft, his software is also not available freely. Corbett and Murray-Rust have proposed the OPSIN sub-system as part of the OSCAR3 system [Corbett and Murray-Rust 2006]. OPSIN uses a finite-state grammar, which is "less expressive but more tractable" than the context-free grammars used in previous methods along with a set of informal rules. Their tokenization is based on "a list of multi-character tokens" and "a set of regular expressions", both created manually.

If a user posed a query by drawing a chemical structure, that could be converted to a name and matched against chemical names in text documents. AutoNom, a tool that converts a chemical structure into a chemical name, claimed to be the "first general program for" such conversion [Wisniewski 1990]. The ACD/Name "systematic nomenclature software" is said to be "a current industry standard" for converting drawn chemical structures into chemical names[9].

Apart from chemical name segmentation, automatic text segmentation techniques have been used in other contexts. Zhao, Mahmud and Ramakrishnan proposed a method for segmenting a text string into structured records [Zhao et al. 2008]. Their technique is useful for segmenting text containing relational attributes like

[9]http://207.176.233.196/products/name_lab/name/

addresses, citations, etc. and segments a text string into its 'n' attributes given 'n'. The number of segments in different chemical names varies and thus their method cannot be applied for our purposes.

3. CHEMICAL ENTITY EXTRACTION

In this section, we discuss how chemical entities in text are tagged.

3.1 Chemical Entities

In this work, we address the problem of tagging the following entities in text: chemical formulae (e.g., CH_4), trivial chemical names (e.g., *water*; including common or trade names like viagara, paraquat, etc.), systematic chemical names, such as those conforming to IUPAC nomenclature (e.g., *2-acetoxybenzoic acid*), and semi-systematic organic chemical names (e.g., *calciferol*).

A chemical formula can represent different compounds, and a compound may have different formulae and name representations. A partial formula is also a formula, by definition, but may not be a meaningful formula. For example, $CH_3(CH_2)_2OH$ is a chemical formula, and C, CH_2, $(CH_2)_2$, and $CH_3(CH_2)_2OH$ all are partial formulae. Currently, our system does not tag abbreviations, e.g., EtOAc, M(acac)$_3$, or wildcards, e.g., R in ROH, X in CH_3CH_2X as chemical formulae. We believe that extending our methods to address chemical formulae that have place-holders for elements should be straight-forward. For example, adding terms like *acac*, the wildcard R or X into our vocabulary should easily extend our method to handle most abbreviations and wildcards. However, processing abbreviations, like EtOAc, would still remain harder to tag and will be handled in future work.

Chemical names are harder to define precisely; we provide some discussion here regarding the types of names our system was trained to detect. We tag any term that refers to the name of a chemical compound as a *chemical name*, e.g., *methane*. Corbett, Batchelor and Teufel have classified chemical entities as "chemical compound", "chemical reaction", "chemical adjective", "enzyme", and, "chemical prefix" [Corbett et al. 2007]. Our system does not tag "chemical reactions" since our focus is on the narrower class of "chemical names". The system does not tag "chemical adjective"s separately but does tag the chemical adjective followed by the noun as a chemical name, e.g., "ascorbic acid". Similarly, it does not detect and tag "chemical prefix"es separately. Furthermore, separating "chemical compound"s from "enzymes" or biological macro-molecules is beyond the scope of this work. Thus, we tag each of them as *chemical name*s. We do not tag InChIs or CAS numbers as *chemical formula* or *chemical name*. Corbett and Batchelor have proposed an annotation manual that runs over 31 pages and contains 91 rules (obtained via personal communication). Our instructions to the annotator was as indicated above and thus at a higher-level than that in the detailed specification proposed by Corbett and Batchelor.

We believe that machine learning methods to mine textual chemical molecule information utilizing domain knowledge are desired due to the following reasons: 1) Two types of ambiguity exist for tagging chemical formulae. First, even though the string pattern may be similar to a formula at first glance, in reality, it may be an abbreviation, e.g., *NIH*. Second, even though a string appears to be a chemical formula, it may be a word, e.g., I(Iodine) versus the pronoun I, He(Helium) versus the pronoun *He*, *In*(Indium) versus the preposition *In*. 2) Because a chemical name can be a long phrase in the text, segmenting it from the context is also challenging, especially when there are ambiguous terms or other chemical names surrounding it. 3) If a system matches a term to terms occurring in a chemistry lexicon, then it will miss newly coined names and names not present in an incomplete lexicon. Furthermore, it cannot handle noisy strings, e.g., those that occur because of small misspellings. We present the following text fragments to show the two types of ambiguities of chemical formulae and the segmentation issue of chemical names in Figure 2.

Non-Formula or Name:
*"... This work was funded under **NIH** grants ..."*
*"... YSI 5301, Yellow Springs, **OH**, USA ..."*
*"... action and disease. **He** has published over ..."*
Chemical Formula:
*"... such as hydroxyl radical **OH**, superoxide O2- ..."*
*"... and the other **He** emissions scarcely changed ..."*
Chemical Name:
*"... studies for **Acetic acid, 1-methylethyl ester** and **2,4-Dichlorophenoxyacetic acid, 1-methylethyl ester** on all organism groups ..."*

Fig. 2. Example of Chemical Formulae, Chemical Names, and Ambiguous Terms

In order to tag chemical entities, we use supervised machine-learning algorithms, namely, Conditional Random Fields and Support Vector Machines, as discussed below.

3.2 Conditional Random Fields

CRFs[Lafferty et al. 2001] are a powerful method for tagging sequential data. Suppose we have a training set S of labeled graphs. Each graph in S is an identical independent distribution (i.i.d.) sample with an internal structure [Lafferty et al. 2001]. CRFs model each graph as an undirected one $G = (V, E)$. 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, representing the probability of $\mathbf{y}$ given $\mathbf{x}$. An exponential probabilistic model based on feature functions is graph as an undirected one $G = (V, E)$. 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, representing the probability of $\mathbf{y}$ given $\mathbf{x}$. An exponential probabilistic model based on feature functions is used to model the conditional probability,

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

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, chain-structured CRF models are usually applied, where only the labels (y_{i-1} and y_i) of neighbors in a sequence are dependent. Moreover, only binary features are usually considered for sequential data. There are two types of features, *state features* $F_j = S_j(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{|\mathbf{y}|} s_j(y_i, \mathbf{x}, i)$ that 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. We use two types of state features: *single-vertex features* obtained from the observation of a single vertex and *overlapping features* obtained from the observations of adjacent vertices. Transition features are co-occurrence of vertex labels and state features. Each feature has a weight λ_j to specify how much 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

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

where the conditional probability of each sample can be estimated from the data set. Maximum log-likelihood is applied to estimate λ. 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.

3.3 Stacked Conditional Random Fields

Although CRFs can model multiple and long-term dependencies on and may have better performance than models that do not consider those dependencies [Li and McCallum 2005], 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 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.

However, dependencies across 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 occurrences of chemical names and formulae, e.g., references seldom contain chemical formulae. Based on these observations, we use stacked CRFs as illustrated in Figure 3. We use classifiers from the highest level to the lowest level of granularity, tag each vertex (e.g., document, sentence, or term) with labels, and use the labels in a higher level as features in a lower level. In principle, at each level, CRFs could be replaced by other unsupervised or supervised classifiers.

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

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

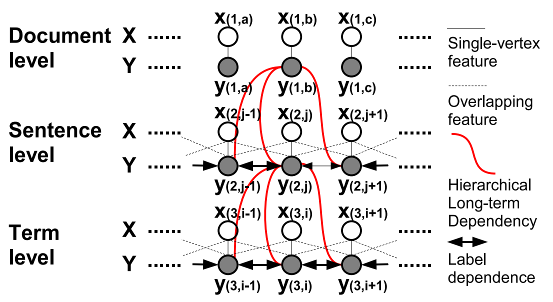


Fig. 3. Illustration of Stacked Conditional Random Fields

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

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*

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

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

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

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 as defined above at the level m. For example, for 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.

3.4 Support Vector Machines

For chemical formula tagging, since each chemical formula is a single term instead of a phrase of several terms, the sequential dependence is not as strong as chemical name tagging, where each chemical name may be a phrase of several terms. Thus, traditional binary classifiers, such as Support Vector Machines (SVMs), can be applied for formula tagging.

3.5 Decision Boundary Tuning for Imbalanced Data

We tune our algorithms to handle imbalanced data as follows. An SVM can be tuned by adjusting the classification threshold value t. When $t < 0$, *recall* is improved but *precision* decreases. When $t > 0$, the converse is expected.

To address the problem of imbalanced training datasets, while using CRFs, we use a weight parameter θ to boost features corresponding to the *true* class during the testing process. Like the classification threshold t in SVMs, θ can tune the trade-off between *recall* and *precision*, and may be able to improve the overall performance, because 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

$$F_j(\mathbf{y}, \mathbf{x}, \theta_{\mathbf{y}}) = \sum_{i=1}^{|\mathbf{x}|} \theta_{y_i} s_j(y_i, \mathbf{x}, i), or \sum_{i=1}^{|\mathbf{x}|} \theta_{y_i} t_j(y_{i-1}, yi, \mathbf{x}, i),$$

and $\theta_{\mathbf{y}}$ is a vector with $\theta_{y_i} = \theta$ when y_i =true classes, or $\theta_{y_i} = 1$ when y_i =false class, and λ_j are parameters learned while training.

3.6 Feature Set

Our algorithm uses two categories of state features from sequences of terms in the text: 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 a term, such as *word* or *word prefix* and *suffix* features, *orthographic* features, or lists of specific terms. The lists of specific terms we used are comprised of lists of chemical symbols, e.g., 'C', 'H', etc., lists of abbreviations that are not chemical terms, e.g., 'HSF','NIH', etc., lists of abbrevations of countries and U.S. states that are not chemical terms, like, 'US', 'UK', 'OH', 'CA'. Note that some features are associated with positive examples and others are features of negative examples. The classifier will learn the weights of these features.

Additionally, our algorithms use a set of parts-of-speech(POS) tagging features extracted using and open source natural language processing tool, like OpenNLP[10]. We use *overlapping features*, (described below in more detail) e.g., we use a feature that checks if a word begins with a capital letter and the following word has a *POS*-tag of *verb*. We also use rule-based features that match string patterns using domain knowledge from chemistry. For chemical name tagging, we use WordNet [11] and downloaded a lexicon of chemical names available online [12]. We use both to extract features of a term that we want to tag. For example, a chemical term is more likely to appear in the chemical lexicon. A non-chemical term is more likely to appear in WordNet but not in the chemical lexicon. We use Levenshtein Distance [Levenshtein 1966] to allow for some fuzzy matching between between the term the algorithm is tagging and a term in the chemical lexicon or a term in WordNet as features. Furthermore, we check if a term has subterms (i.e., prefix,

[10]http://opennlp.sourceforge.net/
[11]http://wordnet.princeton.edu/
[12]http://webbook.nist.gov/chemistry/

Table I. Features applied on chemical entity tagging

Name	Meaning
InitialCapital	Is the first letter of current word capital?
AllCapitals	Are all letters of current word capital?
HasPunctuation	Does current word contain punctuation?
IsAbbreviation	Is current word in the list of common abbreviations?
IsTermLong	Is current word longer than a threshold?
IsFormulaPattern	Does current word follow a string pattern of a chemical formula?
IsPOSTagNN	Is the part-of-speech tag of current word NN ($< noun >$)?
NotInWordNet	Is not current word in the WordNet lexicon?
HasSimTermInChemLex	Is current word similar to a term in the chemical lexicon?
HasSubterm	Does current word contain a subterm in a list?
LastHasPunctuation and CurrentInitialCapital	Does last word have punctuation and is the first letter of current word capital?

infix, and suffix) learned from the chemical lexicon based on the method discussed in Section 4.2.1.

Overlapping features of adjacent terms are extracted from text sequences. For each term, all overlapping features of the last term and the next term are included in the feature set. For example, for the term *He* in "*... . He is ...*", feature($term_{n-1}$ = "**.**" $\wedge$ $term_n$ = $initialCapital$)=*true*, and feature($term_n$ = $initialCapital \wedge term_{n+1} = isPOSTagVBZ$)=*true*. Consequently, "*He*" is likely to be an English word instead of *Helium*. We list some of the features used by our algorithms in Table I.

4. CHEMICAL ENTITY INDEXING

In this section, we discuss the schemes for chemical formula and name indexing. We focus on how to extract and select tokens for indexing. We discuss index pruning for both chemical name and formula indexing.

Since the number of all possible partial formulae of the set of all chemical formulae is quite large and many of them have redundant information, indexing every possible partial formula is prohibitively expensive. Previous research has shown that small indices that fit into main memory usually have much better search response times [de Moura et al. 2005; Buttcher and Clarke 2006]. We proposed an index pruning algorithm in our previous work [Sun et al. 2007] to sequentially select features of partial formulae (a subsequence in a chemical formula) that are frequent and discriminative. The algorithm considers the shortest partial formulae first and then proceeds to consider longer partial formulae.

Our system does not index all possible substrings of chemical names because that would make the index extremely large. Instead, our system hierarchically segments chemical names into "meaningful" substrings and indexes them. For example, for a chemical name string "methylethyl", indexing "methyl" and "ethyl" is enough, while "hyleth" is not necessary.

4.1 Chemical Formula Indexing

In this subsection, we discuss how to analyze a chemical formula and select features for indexing, which is important for subsequence search and similarity search. Since the full graphical structure information of a molecule is unavailable, we use partial formulae as features for indexing and search. The same chemical molecule may have different formula strings mentioned in text, e.g., "acetic acid" may have been listed as CH_3COOH or $C_2H_4O_2$. The same formula can also represent different molecules, e.g., $C_2H_4O_2$ can be "acetic acid" (CH_3COOH) or "methyl formate" (CH_3OCHO).

Algorithm 1 Sequential Feature Selection: SFS(C,$Freq_{\forall s}$,$D_{\forall s}$,$Freq_{min}$,α_{min})

```
Input:
Candidate Feature Set $C$,
set $Freq_{\forall s}$ of all frequencies $Freq_s$ for each subsequence $s$,
set $D_{\forall s}$ of all support $D_s$ for each subsequence $s$,
minimal threshold value of frequency $Freq_{min}$, and
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 Eq (3) ($\alpha_s^{(0)} = \frac{|D|}{|D_s|}$, 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 $F$;
```

Again, indexing all possible sub-formulae of every formula is prohibitively expensive. For example, the sub-formulae of CH_3OH are C, $H3$, O, H, CH_3, H_3O OH, CH_3O H_3OH and CH_3OH. Ideally, an analysis of a query log would reveal which sub-formulae are cost-effective to index. However, in the absence of a query log, we assume that infrequent sub-formulae will also not be frequently queried and thus, use an indexing scheme that does not index infrequent sub-formulae. We use a similar idea and notations about feature selection as those proposed by Yan, et al. [Yan et al. 2004]

Definition 4.1. **Support:** Given a data set D of sequences s, $D_{s'}$, the *support* of subsequence s' is the set of all sequences s containing s', i.e., $s' \preceq s$. $|D_s|$ is the number of sequences in D_s.

For the set $D = \{CH_4, CH_2Cl_2, CHCl_3\}$, D_{CH_2} the support of subsequence CH_2 is the set $D_{CH2} = \{CH_4, CH_2Cl_2\}$ and $|D_{CH_2}| = 2$.

We propose two sequential criteria to select the set of features of subsequences F. The feature selected should be 1) frequent, and, 2) its support should not overlap too much with the intersection of the supports of its selected subsequences in F.

After the algorithm extracts all chemical formulae from documents, it generates the set of all partial formulae and records their frequencies. For the second criterion, we define a discriminative score for each feature candidate with respect to F. Similar to the definitions proposed by Yan, et al., [Yan et al. 2004], a subsequence s is *redundant* with respect to the selected feature set F, if $|D_s| \approx |\cap_{s' \in F \wedge s' \preceq s} D_{s'}|$. For example, consider the set $D = \{CH_4, CH_3Cl, CHCl_3\}$, and the feature set $F = \{CH, CH_2\}$. The set $D_{CH} = D$, and $D_{CH_2} = \{CH_4, CH_3Cl\}$. Consider the sequence CH_3 whose subsequences are CH_2 and CH. The intersection of D_{CH} and D_{CH2} has the same number of elements (namely, two) as that in D_{CH_3}. Hence the sequence CH_3 is redundant with respect to F. A subformula s is *discriminative* with respect to F, if $|D_s| << |\cap_{s' \in F \wedge s' \preceq s} D_f|$. For example, consider the set $D = \{CH_4, CH_3Cl, CHCl_3, CH_2Cl_2, CCl_4\}$, the feature set $F = \{C, H\}$, and the sequence $s = CH_4$. $D_{CH_4} = \{CH_4\}$ and has only one element whereas the intersection of D_C and D_H have four elements in them. Since, $D_{CH_4} << (D_C \cap D_H)$, the sequence CH_4 is considered discriminative with respect to the feature set F. The discriminative score for each subformula candidate s with respect to F is defined as:

$$\alpha_s = |\cap_{s' \in F \wedge s' \preceq s} D_{s'}|/|D_s|. \quad (3)$$

In the last example above, $\alpha_{CH_4} = 4/1 = 4$.

Algorithm 1 shows the sequential feature selection algorithm. The algorithm starts with an empty set F of selected features, scans each subsequence from the length $l = 1$ to $l = L(s)_{max}$.

4.2 Chemical Name Indexing

Before a chemical name is indexed, it has to be segmented into its subterms (or morphemes). A chemical name is first segmented using the punctuations that appear in it, e.g., 10-Hydroxy-trans-3-oxadecalin will be first segmented into its subterms *10, hydroxy, trans, 3, and oxydecalin*. Then, these terms individually will be segmented into their morphemes, e.g., *oxydecalin* will be segmented into the subterms *oxy* and *decalin*. Similarly, the chemical name (1R*, 3S*)-1-Bromo-3-chlorocyclohexane will be segmented using the parenthesis, the comma, and the hyphens as delimiters and then further broken down into the morphemes of the term *chlorocyclohexane*. The subterms and their positions in the name are indexed. However, because we do not interpret the chemical name into a structure, we do not try to interpret the locant information, Greek letters, say representing phases, or stereochemical identifiers. We treat them as subterms in a sequence of terms making up a chemical name. If the end-user wants a "cis" or a "trans" isomer and this term appears in the name of the chemical compound, the chemical name representing the exact isomer will be ranked first by virtue of matching both the chemical-name-term and the isomer-term followed by those documents where the chemical-name-term matches but the isomer-term does not match.

In this subsection, we have shown how to mine frequent subterms from chemical names, and then how to hierarchically segment chemical names into those discovered subterms that can be used for index construction.

4.2.1 *Independent Frequent Subsequence Mining.* In the Section 3.6, we used subterms as features for chemical name tagging. We mined these subterms from the data set. Our search engine supports subterm-based search. However, the search algorithm must have some additional subtlety; when the user searches for "ethyl", we should not return the documents that contain "methyl" before the documents that contain the term "ethyl" not as a substring of methyl. Due to this subtlety, simply using maximal frequent subsequences as defined by Yang [Yang 2004] is not enough. We define the concept of *independent frequent subsequence* below to address this subtlety. Our system attempts to identify independent frequent subsequences and index them. In the rest of this paper, we use "subterm" to refer to a substring in a term that appears frequently. Independent frequent subterms and their frequencies can be used in hierarchical segmentation of chemical names (explained in detail in the next subsection). First, we introduce some notations:

Definition 4.2. **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.

For example, the term "methylene" can be thought to have the subsequences, "meth", "yl", and "ene". We say $Occur_(ylene \preceq methylene, 2, 3) =< yl, ene >$ and $Occur(ylene \preceq methylene, 2, 3)$ overlaps with $Occur(methyl \preceq methylene, 1, 2)$.

Definition 4.3. **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.

For example, the frequency of the subsequence "thyl" in the sequence "methylethyl" is two. All frequent subsequences may not be meaningful subterms. All subsequences of a frequent subsequence are frequent, e.g., "methyl" $(-CH_3)$ is a meaningful subterm, but "methy" is not, although it is frequent too. Thus, simply mining frequent subsequences results in much redundant information. Previous work gives the following concepts [Yan and Han 2003; Yang 2004] to remove redundant information among discovered frequent subsequences (we modify the two definitions to consider subsequence frequencies):

Definition 4.4. **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' of s such that for all $s'' \in D$, the frequency of occurrence of s in s'' is the same as that of s' in s''. The set of closed frequent subsequences is $CS = \{s | s \in FS \textit{ and } \nexists s' \in FS \textit{ such that } s \prec s' \wedge \forall s'' \in D, Freq_{s' \preceq s''} = Freq_{s \preceq s''}\}$.

For example, in a corpus we used, the term "methy" occurs only as part of the term "methyl" and both terms are frequent. The term "methy" is not in the set of closed frequent subsequences CS for that corpus because there exists a super-sequence "methyl" with the same frequency.

Definition 4.5. **Maximal Frequent Subsequence:** A frequent subsequence s is in the set of *maximal* frequent subsequences MS, iff it has no frequent super-sequences. The set of maximal frequent subsequences is $MS = \{s | s \in FS \; and \; \nexists s' \in FS \; such \; that \; s \prec s'\}$.

For example, if “methyl” and “ethyl” are both frequent subsequences in the corpus, then only *methyl* belongs to the maximal frequent subsequences set MS.

Input Sequences: $D = \{methy, metha, met, men, etm\}$,
Parameters: $Freq_{min} = 2$, $L_{min} = 2$
$\mathbf{l = 5}$, $Freq_{methy} = Freq_{metha} = 1$, $IS = \emptyset$;
$\mathbf{l = 4}$, $Freq_{meth} = 2$, $IS = \{meth\}$; now for $s = met/eth/me/et/th$, $Freq_{s \preceq methy} - - = 0$, $Freq_{s \preceq metha} - - = 0$;
for $s = the/de$, $Freq_{s \preceq methy} - - = 0$;
for $s = thf/df$, $Freq_{s \preceq metha} - - = 0$;
$\mathbf{l = 3}$, all $Freq < 2$, $IS = \{meth\}$;
$\mathbf{l = 2}$, $Freq_{me} = Freq_{me \preceq met} + Freq_{me \preceq men} = 2$, but $Freq_{et} = Freq_{et \preceq etm} = 1$, so $IS = \{meth, me\}$;
Return: $IS = \{meth, me\}$, $IFreq_{meth} = 2$ & $IFreq_{me} = 2$.

Fig. 4. Illustration of Independent Frequent Subsequence Mining

The example in Figure 4 demonstrates the set of *Closed Frequent Subsequences*, CS, and *Maximal Frequent Subsequences*, MS. Given $D = \{methy, metha, met, men, etm\}$ and $Freq_{min} = 2$, e.g., support $D_{meth} = \{methy, metha\}$, $D_{me} = \{methy, metha, met, men\}$, etc. The set of frequent subsequences $FS = \{meth, met, eth, me, et, th\}$ has subterms that are unlikely to be useful as index keys. The set $CS = \{meth, me, et\}$ removes some redundant information, while the set $MS = \{meth\}$ removes all redundant information as well as potentially useful information, e.g., *me* and *et* have occurrences excluding those in *meth*. In order to decide if *me* or *et* are useful to index, we need to determine if *me* and *et* are frequent even after excluding their occurrences in *meth*. 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-sequence, then all the occurrences of those super-sequences are excluded to count the independent frequency of s. Thus, in D, *meth* is frequent and *me* is frequent independently, but *et* is not, because *me* occurs twice independently, but *et* only once independently, and $Freq_{min} = 2$. Thus, the set of independent frequent subsequences is $\{meth, me\}$. We define independent frequent subsequences as follows:

Definition 4.6. **Independent Frequent Subsequence:** The set of *independent frequent subsequences* IS contains all frequent subsequences s 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 $IS = \{s | s \in FS \text{ and } IFreq_s \geq Freq_{min}\}$, 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 the CS or MS not sufficient for our application? Consider the case of the subterms “methyl” and “ethyl” (-C2H5). Both are independent frequent subsequences in chemical texts, but not a closed or maximally frequent

Algorithm 2 Independent Frequent Subsequences Mining:

```
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.    $l - -$;
11.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 $Occur_{s \preceq s'} \cap Occur_{t \preceq s'} \neq \emptyset \wedge t \in IS$.

subsequence. For example, for the chemical name in Figure 5, "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", an "ethyl" occurs. Thus, CS over-estimates the probability of "ethyl" that is used for hierarchical text segmentation (described in the next subsection). If a bias exists while estimating the probability of subterms, the quality of the segmentation result suffers.

Based on these observations, we propose an algorithm (Algorithm 2, IFSM) that mines independent frequent subsequences from a collection of sequences with an example in Figure 4. This algorithm scans from the longest to the shortest sequence s , checking if s is frequent. If $Freq_s \geq Freq_{min}$, the algorithm puts s in IS, removes all occurrences of its subsequences that are in any occurrences of s, and removes all occurrences overlapping with any occurrences of s. If the remaining occurrences of a sequence s' still make s' frequent, then the algorithm puts s' into IS and repeats the removal process. After mining independent frequent subsequences, the independent frequencies of the subsequences can be used to estimate their probabilities for hierarchical text segmentation in the next subsection. For example, for "trimethyl", the correct segmentation is "tri" and "methyl". If we over-estimated the probability of "ethyl", the algorithm is likely to segment it into "trim" and "ethyl", because the frequency of "trim" is also not very low.

4.2.2 *Hierarchical Text Segmentation.* We propose an unsupervised hierarchical text segmentation method that segments chemical names into terms (Algorithm 3, HTS), and then utilizes the independent frequent subsequences discovered for further segmentation into subterms (Algorithm 4, DynSeg). DynSeg finds the best segmentation with the maximal probability that is the product of probabilities of each subterm. DynSeg estimates the probability of subterms as:

$$P(s) = \frac{IFreq_s}{\sum_{s' \in IS} IFreq_{s'}} \propto IFreq_s. \tag{4}$$

For each term t with m tokens, a segmentation

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

is to cluster adjacent tokens into n subsequences, where $n = 2$ for recursive segmentation. The probability of segmentation is

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

and the corresponding log-likelihood is

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

Thus, maximum (log) likelihood is used to find the best segmentation,

$$seg(t) = argmax_{seg(t)} \sum_{i \in [1,n]} log(P(s_i)). \tag{5}$$

DynSeg uses dynamic programming for text segmentation [Sun et al. 2007] (Figure 4). In practice, instead of segmenting text into n parts directly, we use hierarchical segmentation of text and at each level, a text string is segmented into two parts. We opt for hierarchical segmentation because of two reasons: 1) determining the appropriate n is difficult, and 2) a text string usually has hierarchically semantic meanings. For example, "methylethyl" is segmented into "methyl" 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".

Hence, after hierarchical text segmentation, we need to index substrings at each node on the segmentation tree. If only strings at the high levels are indexed, then nothing is returned when searching for strings at lower levels. If only strings at the lowest levels are indexed, too many candidates are returned for verification. Since the number of strings in the segmentation tree is no more than twice the number of leaves, indexing all of the strings is a reasonable approach, resulting in a reasonable index size.

5. CHEMICAL ENTITY SEARCH

Users may search for one or more chemical formulae or parts of chemical formulae in a search engine. The search engine returns documents that contain the chemical formulae in the query. A user may also enter substrings of chemical names and the

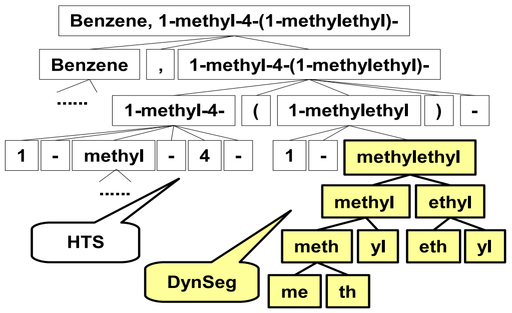


Fig. 5. Illustration of Hierarchical Text Segmentation

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;
```

search engine should return documents with chemical names containing the search term, e.g., a user querying for *aldoxime* should be able to retrieve documents with the term *acetaldoxime* (a document with the term *acetaldoxime* may be ranked after documents with the exact term *aldoxime*). Our search engine calculates a relevance score of a document in response to a user query by weighting each matched indexed feature based on its length, frequency in entities, and distribution among entities.

We propose four basic types of queries for chemical formula search: *exact formula search*, *frequency formula search*, *subsequence formula search*, and *similarity formula search*, and three basic types of queries for chemical name search: *exact name search*, *substring name search*, *similarity name search.*

Features based on subsequences (substrings in names and partial formulae in formulae) are used as tokens for search and ranking. Our search engine uses a scoring scheme based on the Vector Space Model [Baeza-Yates and Ribeiro-Neto 1999] to rank retrieved chemical entities. We define subsequence frequency and inverse entity frequency and use them to rank documents as described below.

Algorithm 4 Sequence Segmentation by 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 $log(IFreq_{s_i}) = log(IFreq_{<t_j,t_{j+1},...t_k>}), 1 \leq j < k \leq m$,
   where $s_i =< t_j, t_{j+1}, ...t_k >$ is a subsequence of $t$. The tree root $r$ with a tree
representation of $s$.
1. if $L_s = 1$ return;
2. Compute all $log(IFreq_{s_i}) = log(IFreq_{<t_j,t_{j+1},...t_k>}), 1 \leq j < k \leq m$,
   where $s_i =< t_j, t_{j+1}, ...t_k >$ is a subsequence of $t$.
3. Let $M(l, 1) = log(IFreq_{<t_1,t_2,...,t_l>})$, where $0 \leq l \leq m$.
   Then $M(l, L) = max_d(M(d, L-1) + log(IFreq_{<t_{d+1},t_{d+2}...,t_l>}))$,
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($s'$,$IF$,$r'$,$n$);
```

Definition 5.1. **SF.IEF:** 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 $s \preceq e$, which is the total number of occurrences of s in e, and $IEF(s)$ is the *inverse entity frequency* of s in C. They are defined as

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

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 $|\{e|s \preceq e\}|$ is the number of entities that contain subsequence s.

For example, consider the term "methylethyl" and say the terms "methyl" and "ethyl" are indexed. Thus $freq(methyl, methylethyl) = 1$ and $freq(ethyl, methylethyl) = 2$ because the sequence "ethyl" appears twice in "methylethyl". Therefore, $|methylethyl| = 1+2 = 3$ and $SF(ethyl, methylethyl) = 2/3$. Now, let there be 10 entities in C and five of them contain the subsequence $ethyl$. Therefore, $IEF(ethyl) = log(10/5) = log2$.

5.1 Chemical Formula Search

In this subsection, we introduce four search semantics for chemical formula search. *Frequency formula search* allows end-users to list the elements that should be in the formula, and *subsequence formula search* allows end-users to search by listing functional groups. If the end-user wants an exact match of the term as written, then they can use the *exact search*. The last type of formula search is *similarity search*. Similarity search could be performed by converting the chemical formula to a chemical structure [Garfield 1962] and then using chemical structure matching [Willett et al. 1998; Yan et al. 2006; Shasha et al. 2002; Raymond et al. 2002] to rank documents containing that chemical formula. However, because subgraph isomorphism involved in structure matching is a costly operation, we propose a fast similarity search method based on the appearance of substrings in the user query and the chemical entity terms in a document. We do not claim that these are the only formula search semantics that are necessary, but believe that these methods are interesting and useful. The utility of our methods and usefulness of our proposed semantics need to be validated by a large scale user study or by examining real query logs of a popular system where these options are made available in the future.

Definition 5.2. **Formula Query and Frequency Range:** A *formula query* q is a sequence of pairs of a partial formula and the corresponding *frequency range* $< s_i, range_{s_i} >$, where token s_i is a chemical element $e \in E$ or another chemical formula f', and $range_{s_i} = \cup_k [low_k, upper_k], upper_k \geq low_k \geq 0$.

For example, the formula query $< CH, 1-2 >< Cl, 1 >$ returns a chemical compound that has one or two CH pairs and one chlorine atom.

Exact formula search

In this case, a user specifying a formula query gets back documents having formulae that match the query exactly, i.e., the elements in the formulae appear in the exact sequence as that specified in the query and the frequency of the elements fall in the range specified in the query. For instance, the query *C1-2H4-6* matches CH_4 and C_2H_6, but not H_4C or H_6C_2. Exact formula search may at first appear to be too rigorous a category since, for example, a chemical researcher searching for CH_4 would surely want to have her search return results for H_4C. However, in more complex searches a researcher may have prior knowledge that one particular form of a formula is usually used for their compound of interest. For example acetic acid is often written as CH_3COOH, whereas methyl formate as CH_3OCOH, both of which have the same general formula $H_4C_2O_4$. A researcher interested in results for only one of these compounds could use the exact formula search in an attempt to discriminate among these forms. **Frequency formula searches**

Most current chemistry databases support *frequency searches* as the only query models for formula searches. Our system supports two types of frequency searches: *full frequency search* 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., C_2H_4, and returns formulae with two *C*, four to six *H*, and any numbers of other atoms for *partial frequency search*, e.g., C_2H_4 and C_2H_4O.

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

$$score(q, f) = \frac{\sum_{e \preceq q} SF(e, f) IEF(e)^2}{\sqrt{|f|} \times \sqrt{\sum_{e \preceq q} (IEF(e))^2}}, \tag{6}$$

where e is a chemical element, $|f| = \sum_{e \preceq f} freq(e, f)$ 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, and $1/\sqrt{\sum_{e \preceq q} (IEF(e))^2}$ is a factor to normalize the score to one atom, so that scores are comparable between different queries. Normalization does not affect the rank of retrieved formulae for a specific formula query, but affects the rank of retrieved documents when one query contains more than two formula searches. Without this factor, documents containing more occurrences of the longer query formula would receive higher scores. Equation 6 considers f as a bag of atoms, where $e \preceq f$ is a chemical element.

Subsequence formula search

In this case, the system returns documents with formulae that contain the formula query as a subsequence. We allow for three types of matches as shown using the following example. For the query *COOH*, *COOH* is an exact match (high score), *HOOC* is a reverse match (medium score), and CHO_2 is a parsed match (low score).

The scoring function for the subsequence formula search is given as

$$score(q, f) = W_{mat(q,f)}.SF(q, f).IEF(q)/\sqrt{|f|}, \tag{7}$$

where $W_{mat(q,f)}$ is the weight for different matching types, e.g., 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 can be tuned by an expert.

Similarity formula search

Similarity searches return documents with chemical formulae that are similar to the query formula. We have not used an edit distance to measure the similarity of two formulae because of two reasons: 1) Formulae with more similar strings or substrings may have a large edit distance. For example, H_2CO_3 can also be written as *HC(O)OOH*, but the edit distance of them is larger than that of H_2CO_3 and HNO_3 ($6 > 2$). Using our partial-formula-based similarity search (Equation 8), for the query of H_2CO_3, $HC(O)OOH$ has a higher ranking score than HNO_3. 2) Computing edit distances of the query formula and all the formulae in the data set is expensive, so a method based on indexed features of partial formulae is much faster and feasible in practice. Our approach is feature-based similarity search that is based on selected features of partial formulae.

A scoring function such as a sequence kernel [Haussler 1999] is designed to measure the similarity between formulae for similarity searches. It maps a query formula into a vector space where each dimension is an indexed partial formula. For instance, the query CH_3OH is mapped into dimensions of *C*, H_3, *O*, *H*, CH_3, and *OH*, if only these six partial formulae are indexed. Then formulae with those partial formulae (including reverse or parsed matched partial formulae) are retrieved, and scores are computed cumulatively for each substring. Larger partial formulae are given more weight for scoring, and scores of long formulae are normalized by their total frequency of partial formulae. The scoring function of similarity search is given as

$$score(q, f) = \frac{\sum_{s \preceq q} W_{mat(s,f)} W(s) SF(s, q) SF(s, f) IFF(s)}{\sqrt{|f|}}, \tag{8}$$

where $W(s)$ is the weight of the partial formula s, which is defined as the total atom frequency of s and $W_m at$ is as defined above.

5.2 Chemical Name Search

In this subsection, we introduce the different semantics allowed by our system for chemical name search. Exact name search is used if the user wants an exact match between the query string and the term in the document. A user uses substring name search when the user wants the query term (e.g., aldoxime) to match part of a term (e.g., acetaldoxime). A similarity name search searches for substrings that are common between a query term and a term in the document. We propose a subterm based similarity computation that runs faster than an algorithm that tries to identify the structure from a chemical name and then matches based on structural similarity. For cases where the chemical names are similarly written, we expect the subterms to match. However, in cases where the same chemical can be named in completely different ways (using completely different sub-terms), our similarity search may not work effectively and a structure-based search will work better. However, in a large digital library, we expect precision to be more important than recall and thus even if our similarity name search misses synonyms of the query term, it may produce a reasonable number of matches to satisfy the end-user. The cost-benefit analysis of the different querying models and their popularity is possible to obtain only when a real system provides these search capabilities to the end-user and analyzes their query logs or performs user studies to poll their satisfaction; currently, we do not have any means to perform such a large-scale unbiased user study.

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 a ranked list of documents containing chemical names that contain the user-provided keyword as a substring. If the query string is indexed, the results are retrieved directly. Otherwise, the query string is segmented hierarchically. The algorithm looks up the substrings in the index. If an entry exists, the entry is retrieved. Otherwise, the substring is further segmented and looked up. Finally, documents that appear in all the index entries retrieved are verified to check whether they contain the query keywords. The ranking function of substring name searches for a query q and a name string e is given as

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

Similarity name search

Similarity name searches return names that are similar to the query. We design a ranking function based on indexed substrings, so that the query is processed and the ranking score is computed efficiently. First, our algorithm segments a query string hierarchically. Then the algorithm looks up the substrings in the index to retrieve postings lists for each substring, and finally, scores are computed using Equation 9. Longer substrings are given higher weights while scoring, and the scores of names are normalized by the total frequency of their substrings. The ranking function is given as

$$score(q,e) = \sum_{s \preceq q} W(s) SF(s,q) SF(s,e) IEF(s) / \sqrt{|e|}, \tag{9}$$

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

5.3 Conjunctive Entity Search and Document Search

Conjunctive search

Conjunctive searches of the basic chemical entity searches are supported for filtering search results. For example, a user can search formulae that have two to four *C*, four to ten *H*, and have a subsequence of CH_2, using a conjunctive search of a full frequency search *C2-4H4-10* and a subsequence formula search of CH_2. For conjunctive chemical name searches, you can define multiple substrings in a query, so that the satisfied chemical name must contain both of them. Chemical names where both substrings appear in order are given higher priority than those in which only one appears.

Query rewriting

When a user inputs a query that contains chemical names and formulae as well as other keywords, our search engine performs the following: 1) chemical entity searches are executed to find desired names and formulae, and 2) returned entities as well as other (textual, i.e., non-chemical-formula and non-chemical-name) keywords defined by users are used to retrieve related documents. We use *TF.IDF* [Spärck Jones 1972] as the ranking function in the second stage, and the ranking scores of each returned chemical entity in the first stage are used as weights of the *TF.IDF* of each chemical entity when computing the ranking score in the second stage.

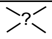
6. EXPERIMENTAL EVALUATION

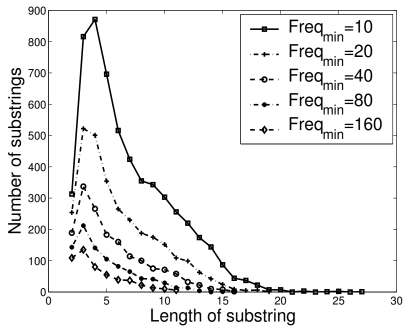
In this section, we present the results of evaluating our proposed methods empirically.

6.1 Independent Frequent Subsequence Mining and Hierarchical Text Segmentation

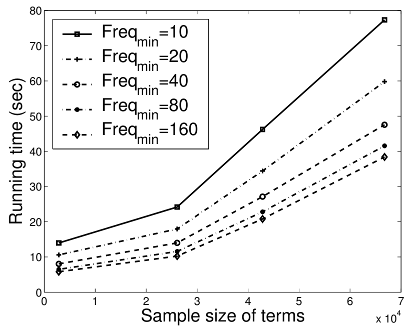
We collected 221,145 chemical names online as a lexicon to tag chemical names. We evaluate our algorithm, IFSM, with different threshold values $Freq_{min} = \{10, 20, 40, 80, 160\}$. The system first tokenizes chemical names and obtains 66,769 unique terms. Then, it discovers frequent subterms from them. The distributions of the subterms' lengths with different values of $Freq_{min}$ are presented in Figure 6(a) and the runtime of our algorithm is shown in Figure 6(b). A graduate student manually determined that most of the discovered subterms have semantic meanings in the chemistry domain. Table II shows the most frequent subterms along with their real meanings. Note that "methyl" has a higher frequency than "ethyl" because we only count the independent frequencies. After IFSM, hierarchical text segmentation is tested, and two examples of the results are shown in Figure 7. We see that most of the segmented subterms have semantic meanings. Not indexing meaningless subterms reduces the size of the index.

Table II. The most frequent subterms at each length, $Freq_{min} = 160$

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



(a) Distribution of discovered frequent substrings



(b) Algorithm running time

Fig. 6. IFSM applied on chemical names

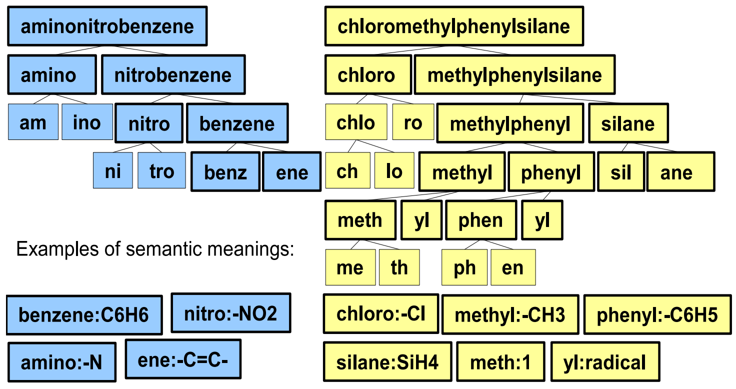


Fig. 7. Examples of HTS. Strings in bold rectangles have semantic meanings.

6.2 Chemical Entity Tagging

Table III. Formula tagging average accuracy

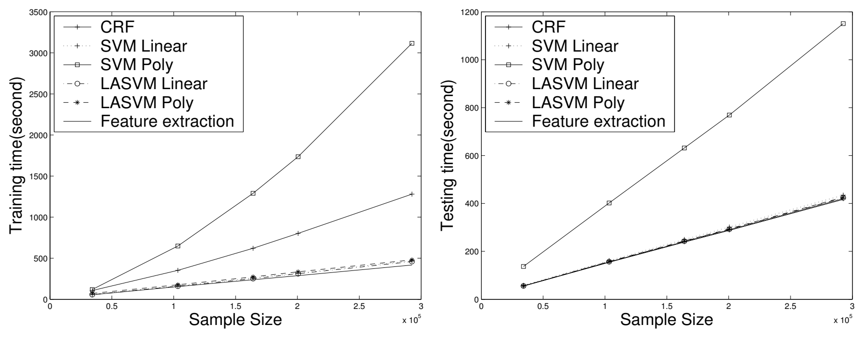
Method	Recall	Precision	F-measure
String Pattern Match	98.38%	41.70%	58.57%
CRF,$\theta = 1.0$	86.05%	96.02%	90.76%
CRF,$\theta = 1.5$	**90.92%**	**93.79%**	**92.33%**
SCRF,$\theta = 1.0$	88.63%	96.29%	92.30%
SCRF,$\theta = 1.5$	**93.09%**	**93.88%**	**93.48%**
SVM linear,$t = 0.0$	86.95%	95.59%	91.06%
SVM linear,$t = -.2$	**88.25%**	**94.23%**	**91.14%**
SVM poly,$t = 0.0$	87.69%	96.32%	91.80%
SVM poly,$t = -.4$	**90.36%**	**94.64%**	**92.45%**
LASVM linear,$t = 0.0$	83.94%	90.65%	87.17%
LASVM linear,$t = -.2$	**85.42%**	**89.55%**	**87.44%**
LASVM poly,$t = 0.0$	75.87%	93.08%	83.60%
LASVM poly,$t = -.4$	**83.86%**	**88.51%**	**86.12%**

To test the performance of our chemical entity tagging, we randomly selected publications from multiple journals crawled from the digital library of the Royal Society of Chemistry [1]. First, 200 documents are selected randomly from the data set, and a part of each document is selected randomly to construct the training set manually. This data set is very imbalanced because of the preponderance of terms that are not chemical entities. For example, only 1.59% of the tokens are chemical formulae (5203 formulae vs. 321514 non-formula tokens).

6.2.1 *Accuracy of Chemical Formula Tagging.* In our experiments to evaluate the tagging of chemical formulae, we test a 2-level stacked CRF(SCRF), where the two levels are the sentence level and the term level. At the sentence level, we construct the training set by labeling each sentence as *content* (document contents) or *meta* (document meta data, including titles, authors, references, etc.). Then the algorithm uses the sentence tags as features at the term level to tag chemical formulae. For formula tagging, we label each token as a *formula* or a *non-formula*.

We perform 10-fold cross-validation to evaluate sentence and formula tagging. For each testing set of samples obtained from 20 files, the rest of the 180 files were used to train the classifiers. For formula tagging, we evaluate several methods, including rule-based String Pattern Match, CRFs with different feature sets, SCRFs, SVMs with linear (SVM linear) and polynomial kernels (SVM poly), and SVM active learning with the linear (LASVM linear) and polynomial kernels (LASVM poly). *SVM light* [1] for batch learning and *LASVM* [Bordes et al. 2005] for active learning are used. We used the CRF tool that is available in MALLET, a Java-based package for statistical natural language processing[2]. For SVMs, we tested the linear, polynomial, RBF and Gaussian kernels. We show the results of the first two and not the latter because they resulted in worse performances and were more expensive computationally than the linear and polynomial kernels. For CRFs, to

[1]http://www.rsc.org/Publishing/index.asp
[1]http://svmlight.joachims.org/
[2]http://mallet.cs.umass.edu



(a) Training time (b) Testing time

Fig. 8. Running time of formula extraction including feature extraction

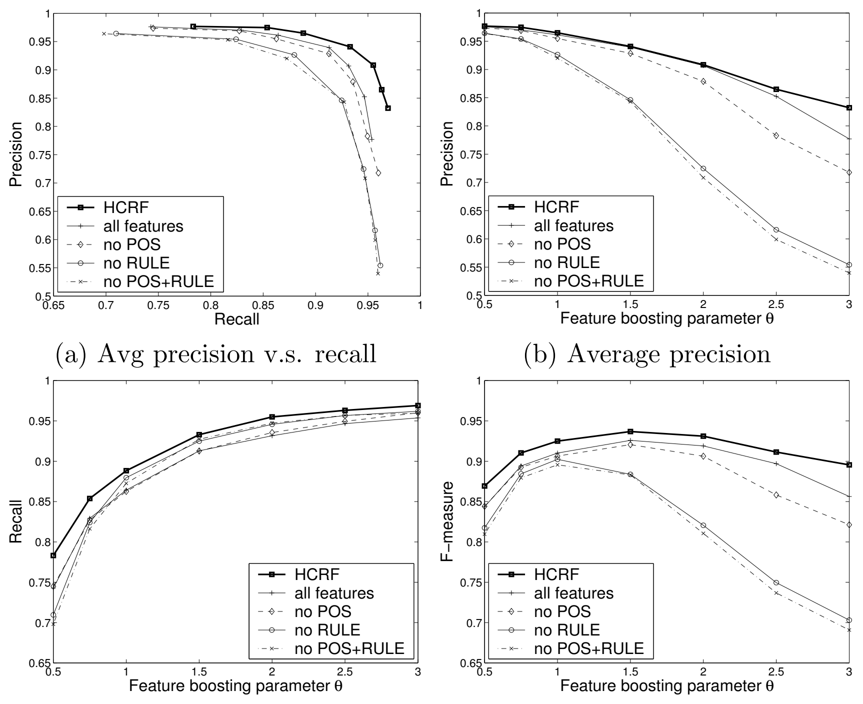
avoid the overfitting problem, *regularization* is used, with $\sigma^2 = 5.0$ [Lafferty et al. 2001]. Features are categorized into three subsets: features using rule-based string pattern match (*RULE*), features using part-of-speech tags (*POS*), and other features. Four combinations are tested: (1) all features, (2) no POS, (3) no RULE, and (4) no POS or RULE.

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 the B-name or I-name) class. Note that when $\theta = 1.0$, we have a 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-measure*, $F = 2PR/(P + R)$ [McDonald and Pereira 2005], where P is precision (i.e., the proportion of true instances among all the predicted true instances) and R is recall (i.e., the proportion of predicted true instances among all the true instances), instead of using an average error rate because the error rate is too small for imbalanced data even if we tag all formulae or chemical names wrongly. Results of average *recall*, *precision*, and *F-measure* for sentence tagging are presented in Table IV and formula tagging in Table III and Figure 9. *Figure 9 shows that using the SCRF, we can obtain a high F-measure with precision near 95% at over 90% recall.* The p-values for the t-tests of significance for formula tagging are shown in Table V. The shapes in Figure 9 and 10 show an F-measure curve with a peak. Thus, we can optimize the classifier by tuning the parameters so that we can achieve performance close to that peak if desired.

Table IV. Sentence tagging average accuracy

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

From Figure 9, we can see that the RULE features contribute more than the POS features, because the difference between curves with or without POS features is significantly smaller than that between curves with or without RULE features. Usually, the performance with more features is better than that with fewer features. We can observe that F-measure curves with fewer features have a higher curvature and are more sensitive to θ than those with more features. We have the best



(a) Avg precision v.s. recall (b) Average precision

(c) Average recall (d) Average F-measure

Fig. 9. Chemical formula tagging using different values of feature boosting parameter θ

overall performance based on *F-measure* for $\theta = 1.5$ using all features and, for this case, recall and precision are more balanced than that in the cases using other θ values. We can also see that SCRFs have the best performance. In comparison to CRFs, SCRFs only have 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 shows that long-dependence features at the sentence level have positive contributions. The downside of SCRFs is an increased runtime when compared to CRFs. However, because the entity extraction and indexing is being done offline, we believe that the extra cost can be justified by the better performance. Both for SCRFs and CRFs using all features, the best *F-measure* is reached when $\theta = 1.5$.

Based on empirical experiences of using the SVMs, we let $C = 1/\delta^2$, where $\delta = 1/n \sum_{i=1}^{n} \sqrt{ker(\mathbf{x_i}, \mathbf{x_i}) - 2 \cdot ker(\mathbf{x_i}, \mathbf{0}) + ker(\mathbf{0}, \mathbf{0})}$ for SVM light, $C = 100$ for LASVM, and we use the polynomial kernel $(\mathbf{x} \cdot \mathbf{x'} + 1)^3$. We test different decision threshold values {-0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8}. From Figure 10(a), we can see that CRF and SVM poly perform better than SVM Linear, but the difference is not statistically significant at the level of 0.05 (Table V). All of these classifiers are much better than LASVM and this conclusion is statistically significant. Moreover, we can see that CRF confers more importance to recall than to precision as opposed to SVM poly, which values precision more. When the importance of *recall* is more than that of *precision*, CRF can produce results with a better F-measure. This

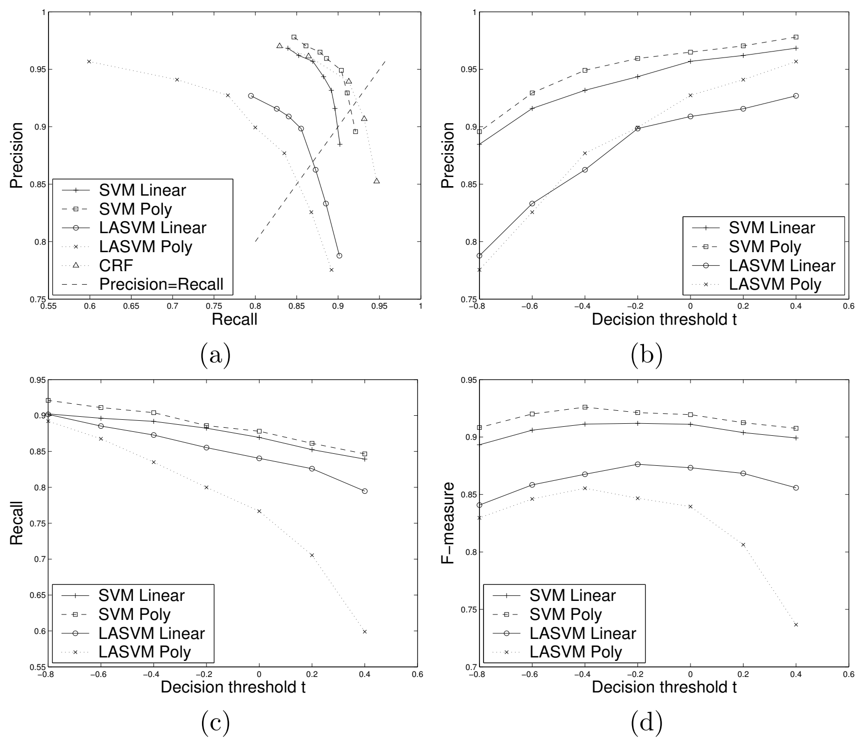


Fig. 10. SVM and LASVM with different values of threshold t

observation is important for imbalanced data.

Table V. P-values of 1-sided T-test on F-measure

Pairs of methods	F-measure
CRF,$\theta = 1.0$;CRF,$\theta = 1.5$	0.130
CRF,$\theta = 1.5$;SVM,linear,$t = -.2$	0.156
CRF,$\theta = 1.5$;SVM,poly,$t = -.4$	0.396
CRF,$\theta = 1.5$;LASVM,linear,$t = -.2$	0.002
CRF,$\theta = 1.5$;LASVM,poly,$t = -.4$	0.000
SCRF,$\theta = 1.5$;CRF,$\theta = 1.5$	0.159
SCRF,$\theta = 1.5$;LASVM,poly,$t = -.4$	0.172
SVM,linear,$t = 0.0$;SVM,linear,$t = -.2$	0.472
SVM,poly,$t = 0.0$;SVM,poly,$t = -.4$	0.231
SVM,linear,$t = -.2$;SVM,poly,$t = -.4$	0.072
SVM,linear,$t = -.2$;LASVM,linear,$t = -.2$	0.009
SVM,poly,$t = -.4$;LASVM,poly,$t = -.4$	0.000

We show the results for formula tagging using all approaches and all features in Table III and compare them with the String Pattern Match approach, which has very high recall but quite low precision. We compare the formula tagging results with the GATE Chemistry Tagger [1]. Since it cannot handle superscripts and can recognize names of chemical elements, e.g., oxygen, the GATE Chemistry Tagger is not fully comparable with our approach. Without counting these two cases, its *recall* is around 63.4%, *precision* 45.2%, and *F-measure* 52.8%. In contrast, the sCRF formula tagger achieves over 93% precision, recall, and F-measure demonstrating that identifying chemical formulae automatically with high accuracy is possible.

6.2.2 *Runtime of Chemical Formula Tagging.* We also evaluate the time taken to run these methods both for the training and testing processes. Note that feature extraction and CRF are implemented in Java, while SVM and LASVM are implemented using the C programming language. The reported runtime includes the time taken for feature extraction and training (or testing) time. In Figure 8(a), we can see that the CRF has a computational cost lying between that of SVM poly and other methods. We also observe that LASVM is much faster than SVM, especially for complex kernels. Nevertheless, because the runtimes of the CRF both for training and testing are reasonable and stacked CRFs produce the best results, we suggest that stacked CRFs be used for chemical formula tagging.

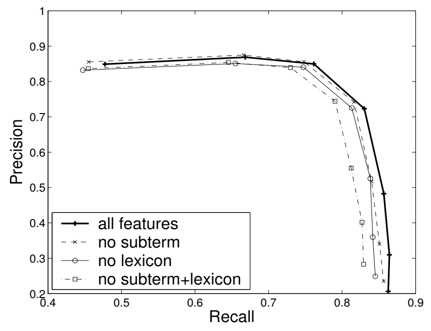
6.2.3 *Accuracy of Chemical Name Tagging.* For name tagging, because a name may be a phrase of several terms, we label each token as a *B-name* (beginning of a name), or *I-name* (continuing of a name), or a *non-name*. We use 5-fold cross-validation to evaluate our methods for name tagging. Specifically, we evaluate CRFs with different feature sets. Features are classified into three subsets: features using frequent subterms (*subterm*), features using lexicons of chemical names and WordNet (*lexicon*), and other features. Four combinations are tested: (1) all features, (2) no subterm, (3) no lexicon, and (4) no subterm or lexicon.

Table VI. Chemical name tagging average accuracy

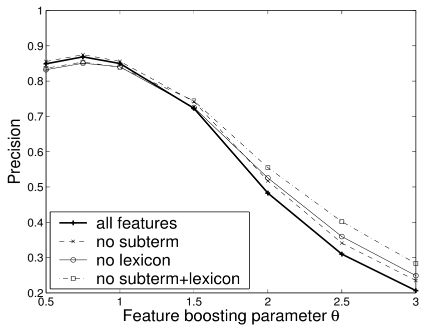
Method, $\theta = 1.0$	Recall	Precision	F
all features	**76.15%**	84.98%	**80.32%**
no subterm	74.82%	**85.59%**	79.84%
no lexicon	74.73%	84.05%	79.11%
no subterm+lexicon	73.00%	83.92%	78.08%

The precision, recall, and the f-measures of the chemical name tagging using different combinations of features are presented in Table VI and Figure 11. From Figure 11, we observe that using all features results in the best recall and F-measure, and using features of frequent subterms improves the recall and the f-measure but decreases precision. Our system outperforms the only downloadable tool for chemical name tagging: Oscar3[2] on our dataset. We used version alpha 1, which was the latest available version at the time this work was done. The experimental results for Oscar3, alpha 1, on our annotated corpus are: *recall* 70.1%, *precision* 51.4%, and *F-measure* 59.3%. These observations are worse than that reported by Oscar3 [Corbett and Murray-Rust 2006]. Hettne, et al., [Hettne et al. 2009] used Oscar3 for chemical entity extraction and reported that they obtained a *precision*

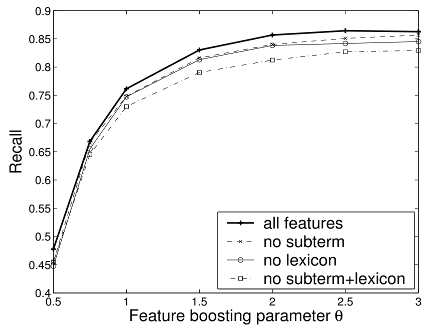
[1]http://gate.ac.uk/
[2]http://wwmm.ch.cam.ac.uk/wikis/wwmm



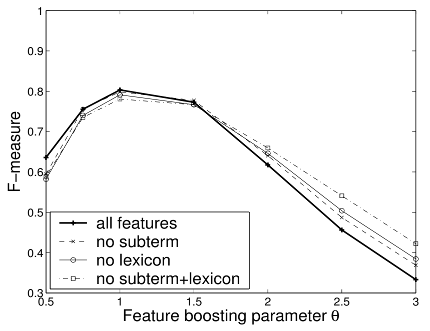
(a) Avg precision v.s. recall



(b) Average precision



(c) Average recall



(d) Average F-measure

Fig. 11. Chemical name tagging using different values of feature boosting parameter θ

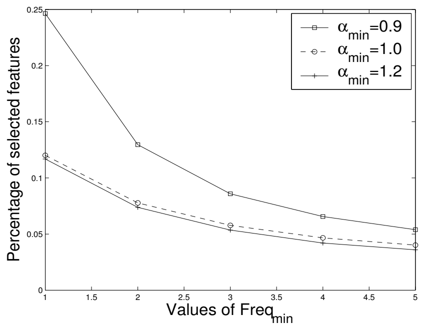
of 45%, *recall* of 82% and an *F-measure* of 58%. These numbers are similar to our observations. The variation of the performance of Oscar3 are perhaps due to the differences in characteristics of the different datasets that were used for training and testing in the three studies. The exact sources of these differences need to be better understood in the future.

As shown above, acceptable precision and recall can be achieved for the task of chemical name tagging using our CRF-based algorithm using features mentioned.

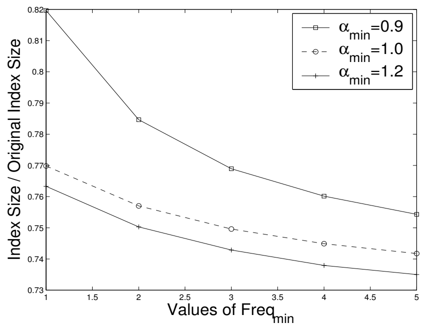
6.2.4 *Conclusions of Chemical Entity Tagging.* We posit that a CRF-based classifier is preferred, for our work, to SVM due to the following reasons: 1) It not only has a high overall F-measure, but also a more balanced tradeoff between recall and precision. 2) The CRF classifier has a reasonable running time. The testing time taken by a CRF classifier is trivial compared with the cost of feature extraction. 3) SVMs only had good performance for chemical formula tagging. For chemical name tagging, CRFs performed better than the SVMs, which is not shown here. The stacked CRFs have the best results, but were also significantly more expensive than CRFs and SVMs with respect to runtime. In summary, we have shown that chemical names and chemical formulae in text can be extracted with reasonable accuracy using the methods listed above.

6.3 Chemical Entity Indexing

We selected a set of 5036 documents and extracted 15853 formulae with a total of 27978 partial formulae before feature selection. Different values for the frequency threshold $Freq_{min} \in \{1, 2, 3, 4, 5\}$ and the discrimination threshold $\alpha_{min} \in \{0.9, 1.0, 1.2\}$ are tested. Note that when $\alpha_{min} = 0.9$, all frequent partial formulae are selected without considering the discriminative score α. When $\alpha_{min} = 1.0$, each partial formula whose support is the intersection of its selected subsequences' supports is removed. When $\alpha_{min} > 1.0$, feature selection may be lossy. In this case, say the feature f is not indexed. The intersection of the supports of the subsequences of f may contain formulae that do not contain f. 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 some webpages [13]. These formulae are used to perform similarity searches.



(a) Ratio of selected features



(b) Ratio of index size

Fig. 12. Features and index size ratio after feature selection

The experimental results (Figure 12) 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 removed, because those features have selected partial formulae with the same support D. When $\alpha_{min} \geq 1.0$, the selection ratio changes a little. We also evaluated the runtime of the feature selection algorithm (Figure 13). A larger $Freq_{min}$ can filter infrequent features directly without computing discriminative scores, which speeds up the algorithm, while the value of α_{min} affects the runtime little (note that curves with the same $Freq_{min}$ but different α_{min} almost overlap in Figure 13). Figure 13 establishes that index pruning reduces the index size by 90% and the search time by over 65%; in the next subsection, we show that the quality of the search results remains high.

For chemical name indexing and search, we use our segmentation based index construction and pruning. We compare our approach to the method using all possible substrings for indexing. We use the same collection of chemical names as

[13] http://webbook.nist.gov/chemistry/

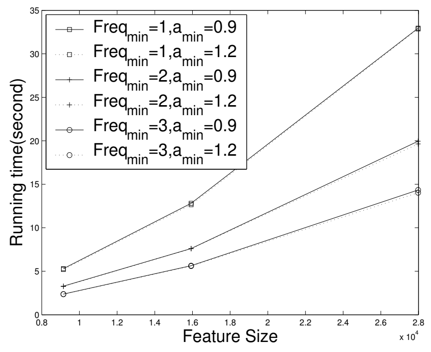


Fig. 13. Running time of feature selection

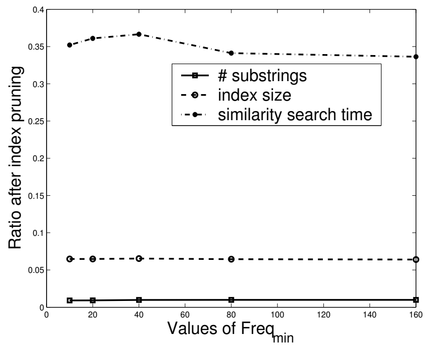


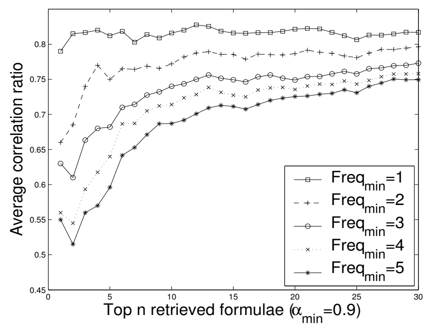
Fig. 14. Ratio of after v.s. before index pruning

mentioned in Section 6.1, and split the collection into two subsets. The first subset is used for index construction (37,656 chemical names), while the query names are randomly selected from the second subset. We test different values of the frequency threshold $Freq_{min} \in \{10, 20, 40, 80, 160\}$ to mine independent frequent substrings. The results in Figure 14 show that about 99% of the substrings are removed after hierarchical text segmentation, so that the index size decreases correspondingly (6% of the original size remains), when $Freq_{min} = 10$. Other values of $Freq_{min}$ result in similar results.

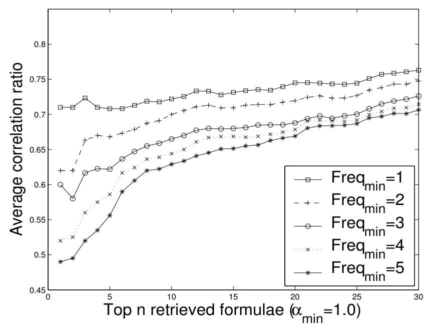
6.4 Chemical Entity and Document Search

6.4.1 *Chemical Formula Search Results After Indexing Pruning.* We observed that for the same similarity formula searches (Figure 15), the search results with feature selection are similar to those without feature selection for reasonable threshold values. We determine the average normalized overlap score, which we will refer to as the *average correlation ratio* for the top $n \in [1, 30]$ retrieved formulae, which is defined as $Over_n = |R_n \cap R'_n|/n, n = 1, 2, 3, ...$, where R_n and R'_n are the search results obtained by applying feature selection or not respectively. As expected, when the threshold values of $Freq_{min}$ and α_{min} increases, the average correlation ratio decreases. The average correlation ratio increases with an increase in n. From the retrieved results, we also find that if there is an exactly matched formula, usually it is returned as the first result. Thus the average correlation ratio for the top retrieved formula is not much lower than that of the top two retrieved formulae. Also, we can see from these curves that a low threshold value of $Freq_{min}$ can keep the curve flat and result in a high average correlation ratio for smaller n, while a low threshold value of α_{min} can increase the average correlation ratio for the whole curve. For the case of $Freq_{min} = 1$ and $\alpha_{min} = 0.9$, more than 80% of the retrieved results are the same for all cases, and 75% of the features are removed, which is both efficient and effective. Reducing the feature space results in reduced memory consumption and query processing times.

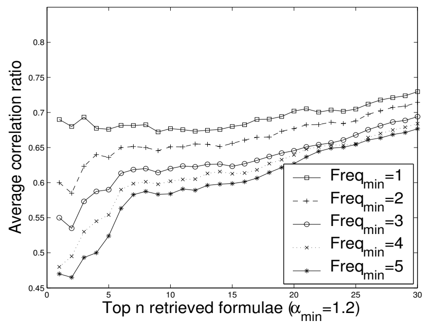
6.4.2 *Chemical Name Search Results After Index Pruning.* For similarity name searches, we generate a list of 100 queries using chemical names selected randomly:



(a) $\alpha = 0.9$



(b) $\alpha = 1.0$



(c) $\alpha = 1.2$

Fig. 15. Overlap of similarity formula search results before and after feature selection

half from the set of indexed chemical names and half from unindexed chemical names. Moreover, for substring name search, we generate a list of 10 queries using the most frequent but semantically meaningful subterms with the length 3-10 discovered in Section 6.1. We also evaluated the response times for similarity name searches, illustrated in Figure 14. 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, we believe that the response time will be even longer.

We also show in Figure 17 that for the same query of similarity name searches or substring name searches, the search results using segmentation-based index pruning has a high average correlation ratio with respect to the result before index pruning. (We use the same average correlation ratio as used for similarity formula searches as discussed in the last subsection to compare the overlap between the results in the two cases.) We observe that for similarity name searches, when more results are retrieved, the average correlation ratio decrease. Conversely, for substring name searches, the overlap between the results in the two cases increase as more results are retrieved. When $Freq_{min}$ is increased, the overlaps also decrease, especially for substring name searches.

Overall, we see that the pruned indexes significantly while preserving the quality

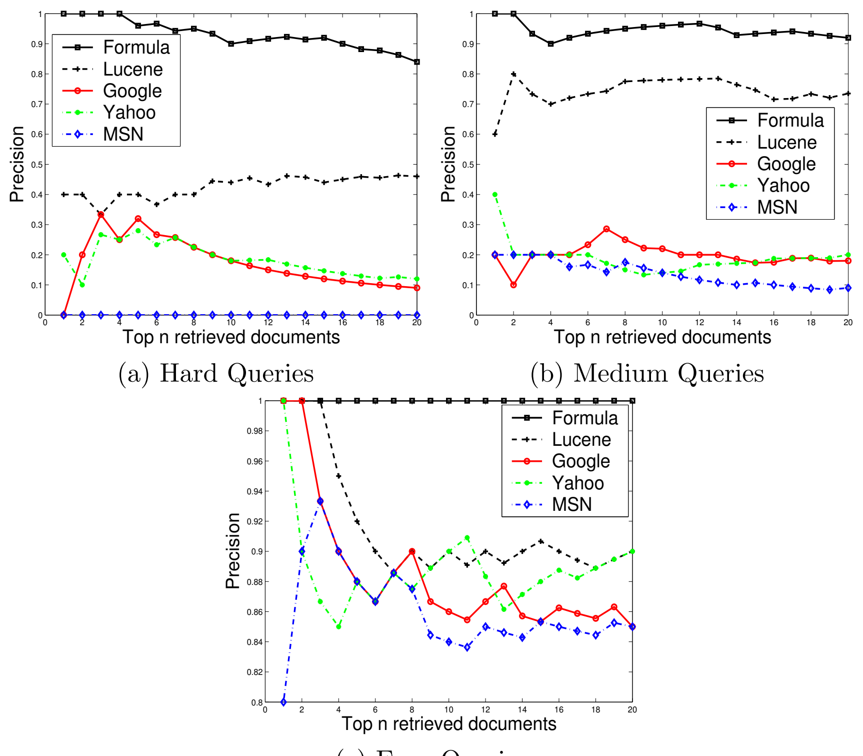


Fig. 16. Average Precision in Document Search using Ambiguous Formulae

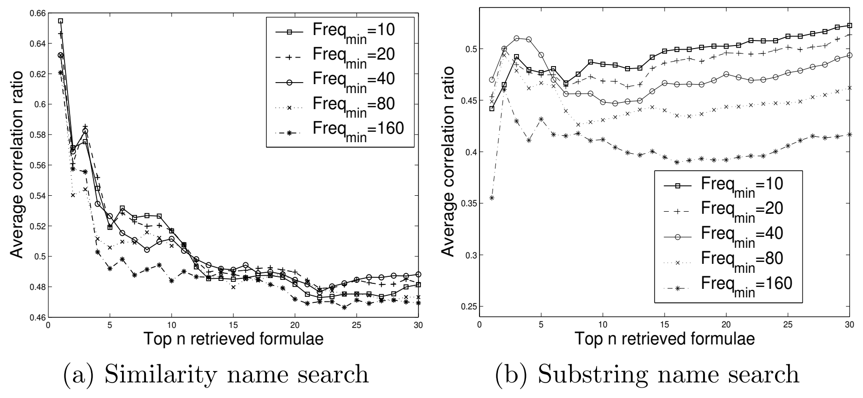


Fig. 17. Correlation of name search results before and after index pruning

of the search results.

6.4.3 *Term Disambiguation in Document Search.* To test the ability of our approach for term disambiguation in documents, we index 5325 PDF documents crawled from the digital library of the Royal Society of Chemistry [1]. Then, we 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 ($CH_4, H_2O, O_2, OH, NH_4$), and 3) easy ($Fe_2O_3, CH_3COOH, NaOH, CO_2, SO_2$). We compare our approach with the traditional approach using keyword matching by analyzing the precision of the top-20 returned documents. The precision is defined as the percentage of returned documents that really contain the query formula in the chemical sense. Lucene [1] is used for the traditional approach. We also evaluate generic search engines (Google, Yahoo, and MSN) using those queries to demonstrate the ambiguity of terms. Since the documents indexed by different search engines are different, the results of general-purpose search engines are not directly comparable with our results and that of Lucene. The results illustrate that ambiguity exists and domain-specific search engines are desired and can improve performance.

From Figure 16, we can observe 1) the ambiguity of terms is very serious for short chemical formulae, 2) results (obtained on Feb 20, 2007) from Google and Yahoo are more diversified than that obtained from MSN and 3) our approach out-performs the traditional approach based on Lucene, especially for short formulae.

7. CONCLUSIONS AND FUTURE WORK

Disambiguating chemical entities from abbreviations, acronyms, and other text is a challenging problem especially for short chemical names, such as OH, He, As, I, NIH, etc. We evaluate classification algorithms based on SVM and CRF for chemical entity tagging and use multi-level stacked CRFs to to address this task. Experiments show that the stacked CRFs perform well, and that our techniques outperform known entity extractors like GATE and Oscar3 on our dataset. We propose efficient index pruning schemes to support partial and fuzzy searches for chemical formulae and chemical names. Experiments illustrate that most of the discovered subterms in chemical names using our algorithm, IFSM, have semantic meanings. Our HTS method for automatic chemical name segmentation worked well on examples obtained from our dataset. Our experiments also show 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. We show that the retrieved ranked results of similarity and substring searches before and after index pruning are highly correlated. We introduce several query models for chemical name and formula searches with corresponding ranking functions. Entity fuzzy matching and query expansion among synonyms will be considered in the future.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0535656. We thank Qingzhao Tan for collecting part of the data and Seyda Ertekin for providing the LASVM code.

[1]http://www.rsc.org/Publishing/index.asp
[1]http://lucene.apache.org/

REFERENCES

ACS. 1993. Registry file basic name segment dictionary.

Baeza-Yates, R. and Ribeiro-Neto, B. 1999. *Modern Information Retrieval.* Addison Wesley.

Banville, D. L. 2006. Mining chemical structural information from the drug literature. *Drug Discovery Today 11(1-2)*, 35–42.

Baum, L. E. and Petrie, T. 1966. Statistical inference for probabilistic functions of finte state markov chains. *Annals of Mathematical Statistics 37*, 1554–1563.

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics 41,* 1, 164–171.

Berger, A. L., Pietra, S. D., and Pietra, V. J. D. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics 22,* 1, 39–71.

Bordes, A., Ertekin, S., Weston, J., and Bottou, L. 2005. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research 6(Sep)*, 1579–1619.

Borthwick, A. 1999. *A Maximum Entropy Approach to Named Entity Recognition.* Ph.D. thesis, New York University.

Brecher, J. 1999. Name=struct: A practical approach to the sorry state of real-life chemical nomenclature. *Journal of Chemical Information and Computer Sciences 39,* 6, 943–950.

Burges, C. J. C. 1998. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov. 2,* 2, 121–167.

Buttcher, S. and Clarke, C. L. A. 2006. A document-centric approach to static index pruning in text retrieval systems. In *Proc. CIKM.* ACM Press.

Cooke-Fox, D. I., Kirby, G. H., and Rayner, J. D. 1989. Computer translation of iupac systematic organic chemical nomenclature. 3. syntax analysis and semantic processing. *Journal of Chemical Information and Computer Sciences 29,* 2, 112–118.

Corbett, P., Batchelor, C., and Teufel, S. 2007. Annotation of chemical named entities. In *BioNLP 2007, A workshop of ACL 2007.* 57.

Corbett, P. and Copestake, A. 2008. Cascaded classifiers for confidence-based chemical named entity recognition. *BMC bioinformatics 9,* Suppl 11, S4+.

Corbett, P. and Murray-Rust, P. 2006. High-throughput identification of chemistry in life science texts. In *Computational Life Sciences II.* Springer, New York, 107–118.

de Moura, E. S., dos Santos, C. F., Fernandes, D. R., Silva, A. S., Calado, P., and Nascimento, M. A. 2005. Improving web search efficiency via a locality based static pruning method. In *Proc. WWW.* ACM Press, New York, 235–244.

Garfield, E. 1962. An algorithm for translating chemical names to molecular formulas. *Journal of Chemical Documentation 2,* 3, 177–179.

Haussler, D. 1999. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10.

Hettne, K. M., Stierum, R. H., Schuemie, M. J., Hendriksen, P. J. M., Schijvenaars, B. J. A., Mulligen, E. M. v., Kleinjans, J., and Kors, J. A. 2009. A dictionary to identify small molecules and drugs in free text. *Bioinformatics 25,* 22, 2983–2991.

Hodge, G. 1991. Enhanced chemical name identification algorithm. In *Abstracts of Papers of the American Chemical Society.* Vol. 202. P41–CINF.

Hodge, G., Nelson, T., and Vleduts-Stokolov, N. 1989. Automatic recognition of chemical names in natural-language texts. In *Abstracts of Papers of the American Chemical Society.* Vol. 197. P17–CINF.

Joachims, T. 1999. Making large-scale support vector machine learning practical. 169–184.

Kemp, N. and Lynch, M. 1998. Extraction of information from the text of chemical patents. 1. identification of specific chemical names. *Journal of Chemical Information and Computer Sciences 38,* 4, 544–551.

Klinger, R., Kolarik, C., Fluck, J., Hofmann-Apitius, M., and Friedrich, C. 2008. Detection of IUPAC and IUPAC-like chemical names. *Bioinformatics 24,* 13, i268.

Kneser, R. and Ney, H. 1995. Improved backing-off for m-gram language modeling. In *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, volume I*. 181–184.

Kremer, G., Anstein, S., and Reyle, U. 2006. Analysing and classifying names of chemical compounds with chemorph. In *Proceedings of the Second International Symposium on Semantic Mining in Biomedicine*, S. Ananiadou and J. Fluck, Eds.

Lafferty, J., McCallum, A., and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*.

Li, W. and McCallum, A. 2005. Semi-supervised sequence modeling with syntactic topic models. In *Proc. AAAI*.

Manning, C. D., Raghavan, P., and Schütze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

McCallum, A., Freitag, D., and Pereira, F. 2000. Maximum entropy markov models for information extraction and segmentation. In *Proc. ICML*.

McCallum, A. and Li, W. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proc. CoNLL*.

McDonald, R. and Pereira, F. 2005. Identifying gene and protein mentions in text using conditional random fields. *Bioinformatics 6(Suppl 1):S6*.

Narayanaswamy, M., Ravikumar, K. E., and Vijay-Shanker, K. 2003. A biological named entity recognizer. In *Pacific Symposium Biocomputing*. AU-KBC Research Centre, Chennai 600044 India., 427–438.

Polikar, R. 2006. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine 6,* 3, 21–45.

Raymond, J. W., Gardiner, E. J., and Willett, P. 2002. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal 45(6)*, 631–644.

Settles, B. 2005. Abner: an open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics 21(14)*, 3191–3192.

Sha, F. and Pereira, F. 2003. Shallow parsing with conditional random fields. In *Proc. HLT-NAACL*.

Shanahan, J. G. and Roma, N. 2003. Boosting support vector machines for text classification through parameter-free threshold relaxation. In *Proc. CIKM*.

Shasha, D., Wang, J. T. L., and Giugno, R. 2002. Algorithmics and applications of tree and graph searching. In *Proc. PODS*.

Spärck Jones, K. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*.

Sun, B., Mitra, P., and Giles, C. L. 2008. Mining, indexing, and searching for textual chemical molecule information on the web. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25*. 735–744.

Sun, B., Mitra, P., Zha, H., Giles, C. L., and Yen, J. 2007. Topic segmentation with shared topic detection and alignment of multiple documents. In *Proc. of SIGIR*.

Sun, B., Tan, Q., Mitra, P., and Giles, C. L. 2007. Extraction and search of chemical formulae in text documents on the web. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Canada*.

Vander Stouw, G. G., Naznitsky, I., and Rush, J. E. 1967. Procedures for converting systematic names of organic compounds into atom-bond connection tables. *Journal of Chemical Documentation 7,* 3, 165–169.

Vasserman, A. 2004. Identifying chemical names in biomedical text: An investigation of substring co-occurrence based approaches.

Wilbur, W. J., Hazard, G. F., Divita, G., Mork, J. G., Aronson, A. R., and Browne, A. C. 1999. Analysis of biomedical text for chemical names: A comparison of three methods. In *Proc. AMIA Symp.*

Willett, P., Barnard, J. M., and Downs, G. M. 1998. Chemical similarity searching. *J. Chem. Inf. Comput. Sci. 38(6)*, 983–996.

Wisniewski, J. 1990. AUTONOM: system for computer translation of structural diagrams into IUPAC-compatible names. 1. General design. *Journal of Chemical Information and Computer Sciences 30,* 3, 324–332.

Wren, J. D. 2006. A scalable machine-learning approach to recognize chemical names within large text databases. *BMC Bioinformatics 7,* 2.

Yan, X. and Han, J. 2003. Closegraph: Mining closed frequent graph patterns. In *Proc. SIGKDD*.

Yan, X., Yu, P. S., and Han, J. 2004. Graph indexing: A frequent structure-based approach. In *Proc. SIGMOD*.

Yan, X., Zhu, F., Yu, P. S., and Han, J. 2006. Feature-based substructure similarity search. *ACM Transactions on Database Systems*.

Yang, G. 2004. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proc. SIGKDD*.

Zhao, C., Mahmud, J., and Ramakrishnan, I. 2008. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *SDM*. 420–431.