JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

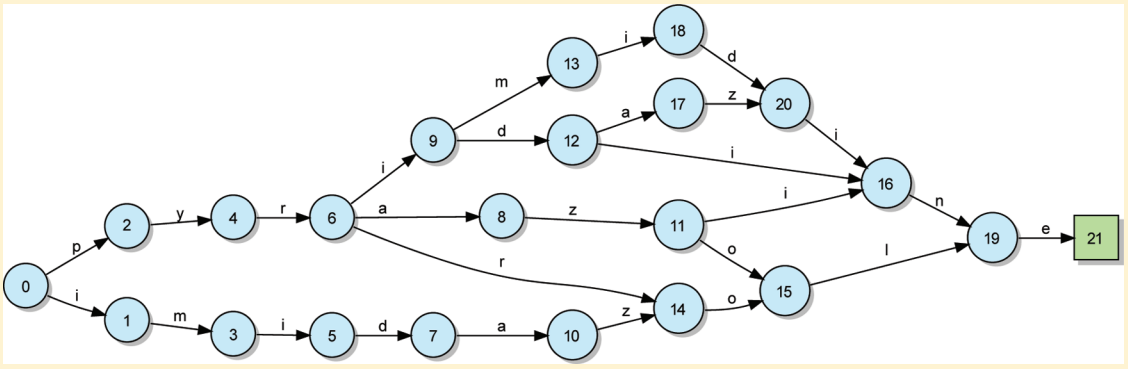
Improved Chemical Text Mining of Patents with Infinite Dictionaries and Automatic Spelling Correction

Roger Sayle,[†] Paul Hongxing Xie,[‡] and Sorel Muresan*[,‡]

[†]NextMove Software, Cambridge, United Kingdom

[‡]Discovery Sciences, Computational Sciences, AstraZeneca R&D Mölndal, S-431 83 Mölndal, Sweden

Supporting Information



ABSTRACT: The text mining of patents of pharmaceutical interest poses a number of unique challenges not encountered in other fields of text mining. Unlike fields, such as bioinformatics, where the number of terms of interest is enumerable and essentially static, systematic chemical nomenclature can describe an infinite number of molecules. Hence, the dictionary- and ontology-based techniques that are commonly used for gene names, diseases, species, etc., have limited utility when searching for novel therapeutic compounds in patents. Additionally, the length and the composition of IUPAC-like names make them more susceptible to typographic problems: OCR failures, human spelling errors, and hyphenation and line breaking issues. This work describes a novel technique, called CaffeineFix, designed to efficiently identify chemical names in free text, even in the presence of typographical errors. Corrected chemical names are generated as input for name-to-structure software. This forms a preprocessing pass, independent of the name-to-structure software used, and is shown to greatly improve the results of chemical text mining in our study.

INTRODUCTION

The ever faster publication rate of chemical information in scientific journals and patents places an increasing burden on researchers to stay current with the literature. One possible solution is the use of indexing, classification and abstraction to identify pertinent documents. Traditionally, this has been the domain of manual curators with companies, such as Chemical Abstracts Service (CAS)[1] and Thompson Reuters,[2] employing a large number of experts to extract the necessary information by hand. However, to combat the exponential growth in scientific publication rate, automated methods are increasingly being used to reduce the lag between publication and abstraction.[3]

The need to track patent applications is particularly important for the pharmaceutical industry.[4] Not only does a granted patent affect a company's freedom to operate but many scientific results are first disclosed in patents, often several years before they appear in the scientific literature as a peer-reviewed article, and for many results, a patent filing is the only place they are published. As an example, consider Merck's "Novel, Acyclic Cannabinoid-1 Receptor Inverse Agonist for the Treatment of Obesity" paper that appeared in the Journal of Medicinal Chemistry in 2006.[5] The corresponding patent US20040058820 was granted two and a half years earlier in March, 2004, and this patent's application was filed 18 months earlier still in November, 2002.[6] In the field of competitive intelligence, alerting researchers to this result rapidly after its initial disclosure in 2002 would provide a significant advantage over waiting for its academic publication three years later.

The task of automatically identifying the molecules mentioned in a document is commonly referred to as chemical named entity extraction or chemical named entity recognition (CNER). While much of the task of CNER bears similarity to other applications of text mining and text analytics, it differs in a few important aspects. The first and most important is chemistry's use of systematic nomenclatures for concisely describing molecular structures. Whereas in many fields the number of terms of interest is known and finite, and can conveniently be represented as a dictionary, the number of

Received: September 30, 2011
Published: December 13, 2011

molecules that can be described by International Union of Pure and Applied Chemists (IUPAC)[7] or CAS[8] naming is infinite. Hence, although well-established techniques, like hash-table based dictionary lookup, can be applied to common, trade, and scientific names, in much the same way as they are used for gene names or diseases, they are inappropriate for systematic nomenclature.[9] Indeed, in many applications of chemical text mining, but particularly pharmaceutical patent analysis, it is the set of names specified by systematic IUPAC-like nomenclature that are of most interest, as these contain the novel compounds. A second significant difference to regular text analytics is that the novel chemical structures reported in a document are often the most relevant. Normally, document clustering schemes identify similar documents based on the co-occurrence of terms. The more terms shared between two documents, the more similar they are. For pharmaceutical patents, however, it is the terms that are unique to a document that are most relevant, and it is the similarity between these singletons that is typically used to identify related patents. Commonly occurring chemical terms, such as "water" and "acetone", on the other hand, tend to be reagents of little significance. Finally, the third major difference is that the lexical morphology of systematic chemical names is unlike that of English or any human language. Whereas English text can usually be tokenized by punctuation, chemical names can legitimately contain whitespaces, hyphens, commas, parentheses, brackets, braces, apostrophes, superscripts, Greek characters, digits, and periods. This makes the task of determining where an IUPAC name begins and ends in a document significantly more complicated than for regular English text.

Previous Work. An excellent recent review of the various approaches to identifying chemical names in free text is given by Klinger et al.[10] The traditional process of CNER is presented graphically in Figure 1, where the task is split up

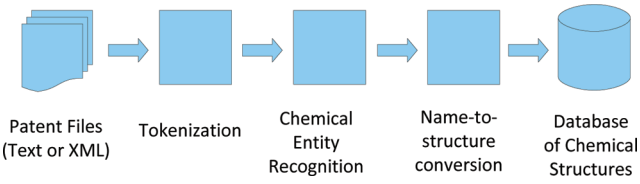


Figure 1. Traditional (flawed) chemical text mining pipeline.

into a number of sequential steps. The first step is "tokenization" where the input documents, typically in text or XML format, are segmented into individual words or entities. The second step is chemical entity recognition, where each token is analyzed to determine whether it represents a chemical name or not. And finally, the third step is to pass each of the entities determined to be a chemical name through name-to-structure software to generate a connection table, a SMILES string[11] or an InChI[12] for the molecule, which can be used to spot synonyms, and to index and retrieve this document.

Much of the previous research work on chemical text mining can be described in terms of this model. For example, the University of Cambridge's OSCAR program implements exactly this architecture,[13] tokenizing on whitespace,[14] using a Bayesian classifier trained on 4-grams to select chemical names, and then looking up the resulting entities in a dictionary, or passing them to their OPSIN program, for conversion to a chemical structure.[15]

The approaches of ProMiner[16] and EbiMed[17] restrict themselves to considering only common and drug names, allowing the use of dictionary-based look-up methods. However, to handle systematic names, typically not encountered in a training set, most approaches use probabilistic classification or machine learning techniques. Narayanaswamy et al.[18] and Kemp and Lynch[19] make use of hand crafted rules for identifying chemical names. IBM's system described by Wu et al.[20] uses *n*-gram analysis much like OSCAR,[13] while Klinger et al.[10] make use of conditional random field (CRF) models.

In addition to the University of Cambridge's OPSIN described above, numerous commercial solutions for name-to-structure software also exist. Examples of these include OpenEye Scientific Software's Lexichem,[21] CambridgeSoft's name=struct,[22] ACD Labs' ACD/Name,[23] ChemAxon's name-to-structure,[24] Bio-Rad's IUPAC DrawIt,[25] ChemInnovation's NameExpert,[26] and InfoChem's ICN2S.[27]

Alas, the major drawback of this traditional model of chemical text mining pipeline is precisely its modularization and decomposition into independent tasks. In practice, these steps are actually interdependent. Attempts to localize chemistry-specific knowledge to the later name-to-structure processing hinder the quality of the earlier name segmentation and putative chemical identification stages. A significant complication in OPSIN's name-to-structure algorithms is the need to address the name segmentation imposed by OSCAR. Ultimately, if the measure of success of chemical text mining is the fraction of names converted into chemical structures, why have a recognition step at all? There is no benefit for identifying something as potentially a chemical name, if it cannot be interpreted as a structure and if the filtering effects of failing to recognize an entity that could be converted to a structure only decreases the recall. Hence, with the availability of modern robust name-to-structure software that can process many thousands of names per second, the solution proposed here is to make the tokenization and entity recognition significantly more chemically aware, making use of the parsing techniques used in name-to-structure software.

A significant point worth mentioning is the dramatic improvements in the availability and quality of chemical name-to-structure software in recent years. To illustrate these, the graph in Figure 2 shows the improvement in OpenEye's Lexichem between releases, as measured against three benchmark sets of chemical names: the 234 K machine-generated names for the August 2000 version of the National Cancer Institute (NCI) database,[28] the KeyOrganics 2003 catalogue,[29] and the Maybridge 2003 catalogue.[30] The use of Lexichem as an example is arbitrary, and similar improvements can be seen for almost all of the software listed previously. The important feature is that, as a class, name-to-structure software can now handle over 95% of the chemical structure/names encountered in general organic databases and compound supplier catalogues. Although vendors will continue to further improve their algorithms and compete over the remaining few percent of difficult molecules and rare chemical nomenclatures, the scope for future improvement is dwarfed by the progress already made over the past decade. In all likelihood, the improvements to or differences between name-to-structure software will no longer be observable to the average user in the not too far distant future.

Unfortunately, the high precision and recall attained by the current generation of name-to-structure software described above reflect an idealized world of processing correctly formed

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51−62

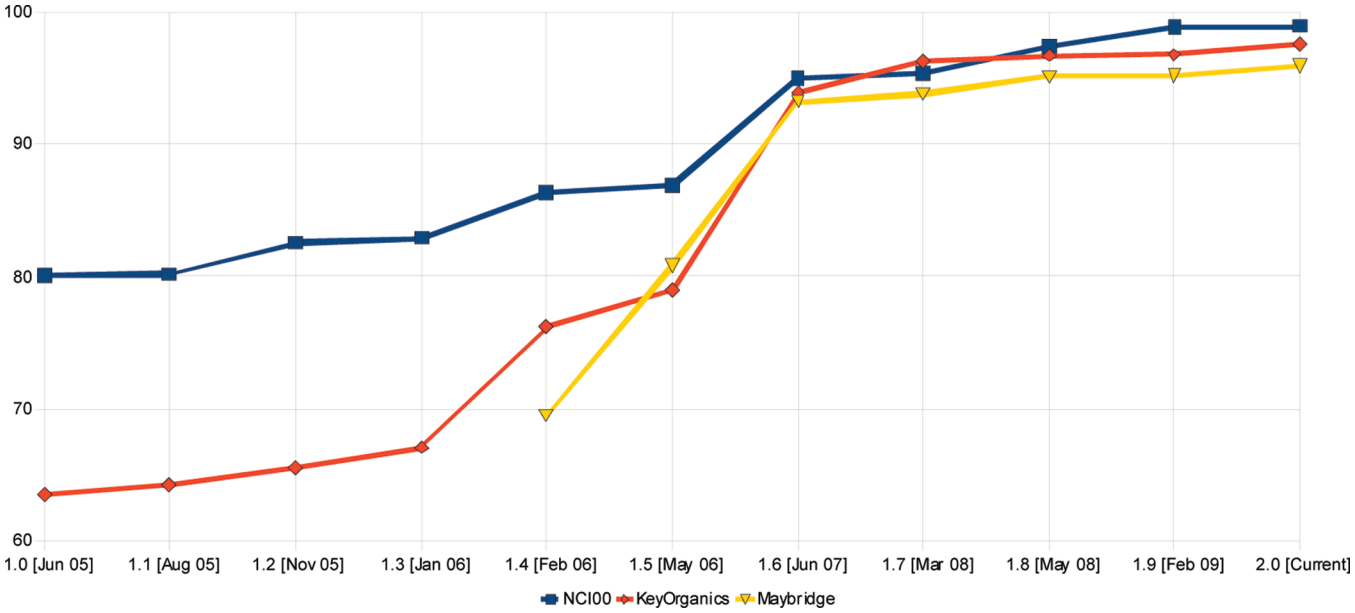


Figure 2. Historical improvement in chemical name-to-structure software (specifically, the conversion rates of OpenEye's Lexichem software on three different chemical name benchmark sets).

valid chemical names, each perfectly delimited. Under these conditions, for example, one name per line in a text file, name-to-structure software has exact conversion and round-trip rates almost as high as other chemical file formats. In practice, however, the picture is not quite so rosy. A major usage of chemical name-to-structure conversion software is in text mining and chemical named entity extraction from patents, scientific articles, and online Web documents. In these use cases, recall rates significantly less than 40% are not uncommon. However, the quality of name-to-structure conversion is not (or is no longer) the bottleneck. Instead, the major causes of missing compounds when extracting chemical entities from text are poor segmentation and the presence of typographical errors: human errors, OCR failures, hyphenation, and line breaking issues, etc.

A significant amount of computer science research has also been devoted to the problem of spelling correction. A classic review article in the field has been presented by Kukich.[31] Almost all of this work involves natural languages, such as English, though some of the work on agglutinative languages, such as Turkish and Finnish, is relevant to the lexical morphology of IUPAC names.[32] Two important published exceptions to this rule address the problem of spell checking IUPAC and IUPAC-like systematic names. The SPEEDCOP project at CAS in the early 1980s used *n*-gram analysis to identify potentially misspelled chemical names for manual correction by an abstractor.[33,34] The classic work of Cooke–Fox et al. at the University of Hull is the only publication known to us to describe automatic spelling correction of chemical nomenclature.[35] Their approach was implemented as an additional set of rules in their name-to-structure software to enable it to be more tolerant of common nomenclature mistakes. Additionally, although not described in the literature, commercial chemical name-to-structure software is increasingly aware of typographical errors. ChemAxon's name-to-structure internally makes use of textual substitutions, such as "y1" to "yl", to handle frequent typos, and CambridgeSoft's name=-struct has similar functionality extending the recognized tokens to handle cases, like "choro" and "flouro".[36] Indeed, many name-to-structure programs trivially handle spelling variants, such as British "sulfur" or German's "pyridin". While not actually correcting errors, the ability to tolerate poorly formed input is a virtue of robust software.

METHODS

Efficient Dictionaries. At its core, text mining amounts to string matching. A document, such as a pharmaceutical patent, is analyzed to identify all of the terms or keywords of interest that it contains. This step typically makes use of a large dictionary of terms to look for, and locating occurrences of these terms in large databases of text is a time-consuming computational process.

Among the many algorithms for exact string matching, those based upon finite state machines (FSMs), also called finite state automata, have demonstrated themselves to be well suited to the task.[37] Typically, the dictionary of keywords to search for, such as gene names in bioinformatics or drug names in medicinal chemistry, may be very large, often containing tens of thousands of entries or more. Brute force methods that attempt to locate each of these terms sequentially would be prohibitively expense. Fortunately, because the dictionary is known in advance and is usually relatively static, it can be preprocessed to allow for significantly more efficient matching. Indeed, methods based on FSMs or hash-tables have performance that is independent of the number of strings being searched for.[38] This allows processing with dictionaries containing many millions of keywords to be just as efficient as looking for a single term. In this work, we shall prefer FSMs to the more commonly employed hash-tables for reasons that will become apparent.

To explain the process of encoding and matching a dictionary using a FSM, we consider a small example dictionary containing the names of seven nitrogen-containing heterocycles: pyrrole, pyrazole, imidazole, pyridine, pyridazine, pyrimidine, and pyrazine.

FSM-based string matching works by precompiling this list of words into a graph-like representation, for efficient matching.

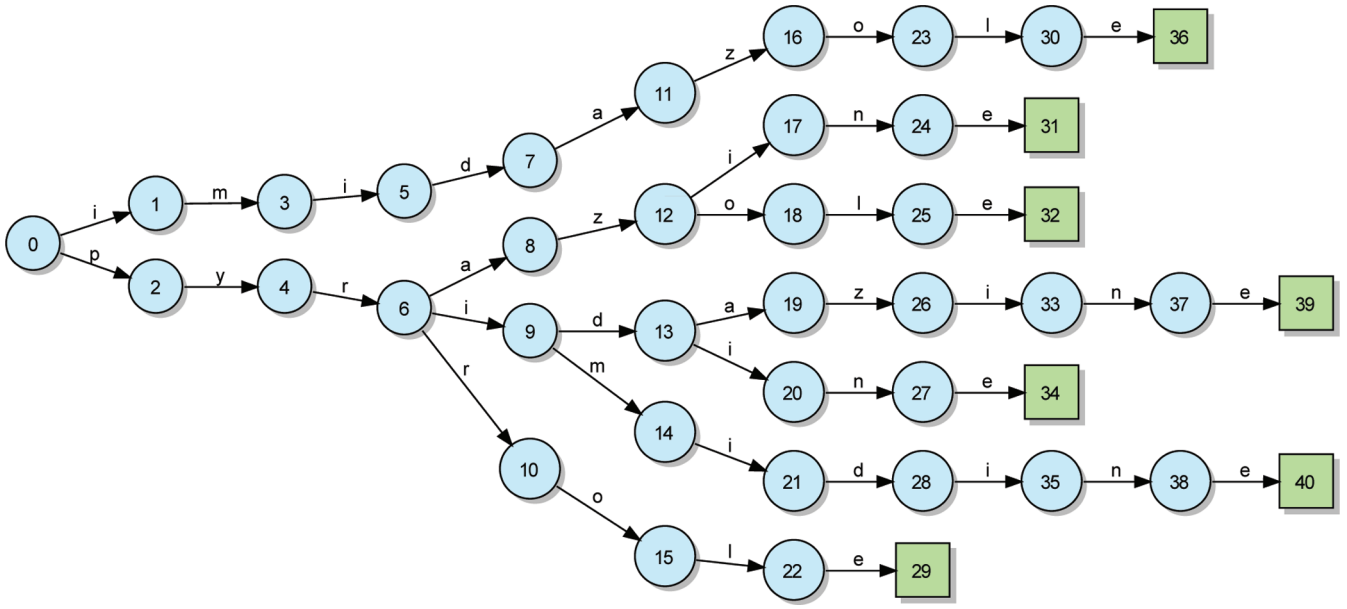


Figure 3. Trie representation of an example dictionary of seven nitrogen-containing heterocycles: pyrrole, pyrazole, imidazole, pyridine, pyridazine, pyrimidine, and pyrazine.

The simplest of these graph-like representations is known as a "trie".[39] A trie is a particular form of tree data structure where the root node denotes the empty string, each internal node represents a prefix, and each leaf node represents a full dictionary word. At each node in the trie, outgoing branches are labeled by unique characters that lead to child nodes, which represent the strings formed by adding the edge character to the current prefix. Each node has a variable number of branches, but at most the number of characters in the alphabet. The trie corresponding to our example dictionary is shown in Figure 3.

Each numbered state/node is represented by a circle or a square. The initial root state is labeled "0", and the remaining state numbers are arbitrary but unique. The circles indicate interior nodes, while the squares denote leaf nodes or accepting states. Each branch/edge is directional and labeled by a single character. Square states are accepting, indicating that if we've reached the end of the word, we match a term in the dictionary.

Conceptually, multiple string matching proceeds by starting at the root node "0". We then look at the first character of the input string and follow the matching edge, if any. If the first character is "p", we advance to state "2". If the first character is "g", there is no matching edge, and we immediately know that the input word is not in the dictionary. From state "2", we check the second character which has to be a "y" and so on. If ever we reach the end of the input string, we check whether we are currently in an accepting state, in which case we have a match, otherwise we fail. An accepting state may potentially be the ancestor of another state, indicating that a valid name is a strict prefix of another valid name.

Without loss of generality, characters may be considered to be bytes, allowing up to 255 children of a node. Typically in English text, the number of characters in the alphabet is 26, but permitting a full byte enables handling words containing upper and lowercase characters, punctuation, spaces, and even multibyte characters (such as Chinese or Japanese text) where the FSM sequentially matches each byte of the character encoding.

While conceptually simple, efficiently representing these automata/graphs in a computer poses some potential complications. The source of these complications is the arbitrary degree or fan-out of each node. At first glance this might require storing nodes as variable length records, such as an STL vector in C++, with a count field and array of outgoing edges. Traversing the graph (executing the automata) would then require the appropriate logic at each node to loop over this array, and algorithms for constructing, copying, and modifying the automata would have to be aware of the potential fragmentation issues with variable size memory allocation and deallocation. An alternative to this approach is to regularize the data structure by always allocating enough space for the maximum number (the alphabet size) of outgoing edges. Although this approach works well with small alphabets, such as nucleic acid sequences, it is wasteful when applied to sparse byte-labeled edges.

An elegant solution is to use a "linked list" of labeled outgoing edges at each state. Temporarily ignoring the "accepting state flag", each state only contains the head pointer for its list of outgoing edges. By unifying states and edges, this leads to a homogeneous "binary tree" data structure, where each node contains two pointers: a "down" edge labeled by a character, and an "across" pointer (forming the linked list's next chain). All that is required to support accepting state flags is to annotate down edges with both the character to match and a flag to indicate whether the destination is matching. The result is an efficient in-memory and on-disk representation of a dictionary, requiring about 10 bytes per node.

A suitable C/C++ representation for FSM node is given by the type declaration in Figure 4. The entire automaton/dictionary is then simply an array of these nodes. By convention, the initial state is the first element of the array.

Matching a word against a dictionary encoded in this format is remarkably simple. The entire C/C++ implementation for an exact match dictionary look up is only 10 (or so) lines long, as shown in Figure 5.

Without modifying the matching algorithm in any way, it is possible to further reduce the memory requirements of this

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51−62

```
typedef struct {
      unsigned char ch;
      unsigned char accept;
      unsigned int down;
      unsigned int across;
} FSMType;
```

Figure 4. C/C++ data structure for representing finite state machines/dictionaries.

```
int match(const FSMType *fsm, const unsigned char *ptr) {
    unsigned int state = 0;
    for (;;) {
        if (fsm[state].ch == *ptr) {
            ptr++;
            if (*ptr == '\0')
                return fsm[state].accept;
            state = fsm[state].down;
        } else state = fsm[state].across;
        if (state == 0) return 0;
    }
}
```

Figure 5. C/C++ implementation of exact match dictionary look-up.

approach, by replacing the trie with a directed acyclic graph (DAG), also called a minimal acyclic deterministic finite automata or directed acyclic word graph (DAWG). The process used to compress or minimize a trie is called "finite automata minimization", and several efficient algorithms exist in the computer science literature.[40] The result of minimizing the FSM for the example dictionary is shown in Figure 6. Although the connection back to the original dictionary word list is less obvious, following the matching process described above should convince the reader that this graph/automaton matches the exact same set of terms but requires fewer states.

Improved Tokenization. A major benefit of using FSMs for chemical text mining is improved tokenization (the splitting of free text into words/terms). Chemical names frequently contain spaces, digits, punctuation, Greek characters, and superscripts that confound traditional natural language processing (NLP) tools. A major advantage of FSMs is that in addition to being able to lookup a string in a large dictionary, they can also be used to test whether an input string is also a prefix of any word in the dictionary. As long as there remain outgoing edges from the current state, the input string may be extended to reach a terminal accepting state. Consider the chemical named entity "benzoic acid". While the word "benzoic" is not a complete IUPAC name, it is a valid prefix that may be extended by a space, enabling the tokenizer to correctly annotate the complete term. Indeed "benzoic acid" might still be the prefix of "benzoic acid methyl ester" or "benzoic acid Weinreb amide". This ability to extend tokens by recognizing prefixes allows characters, such as spaces and commas, to be delimiters in some contexts but part of the recognized entity in others.

Infinite Dictionaries (Grammars). The use of state minimized DAGs for encoding dictionaries represents the current state-of-the-art in dictionary lookup and spell checking and is the method currently used by many word processors, office productivity tools, and Internet search engines. The original contribution presented in this work is to lift the restriction on FSMs used in spelling correction to being acyclic graphs. By allowing backward edges in the dictionary FSM, a single finite state machine can represent/encode/recognize an infinite number of unique strings. By using methods similar to those used to match regular expressions,[41,42] it has been possible to construct a single large FSM encoding a significant fraction of the syntactic rules used for IUPAC naming. This dictionary effectively contains an infinite number of compound names, covering the vast majority of chemical names likely to be found in medicinal chemistry papers or pharmaceutical patents. The construction and structure of the CaffeineFix FSM bear great similarity to the parsers often used in name-to-structure conversion software.

To demonstrate the benefits and weaknesses of this approach, Figure 7 presents a short formal grammar describing a set of halogenated alkanes. The syntax used is similar to that used in Kirby et al.[35] By this example grammar, a locant is defined to by any single digit, a substituent is one of "fluoro", "chloro", or "bromo", and a parent is one of "methane", "ethane", "propane", or "butane". A name is defined to be a parent, optionally preceded by a prefix, which may be optionally separated by a hyphen. The interesting bit is that a

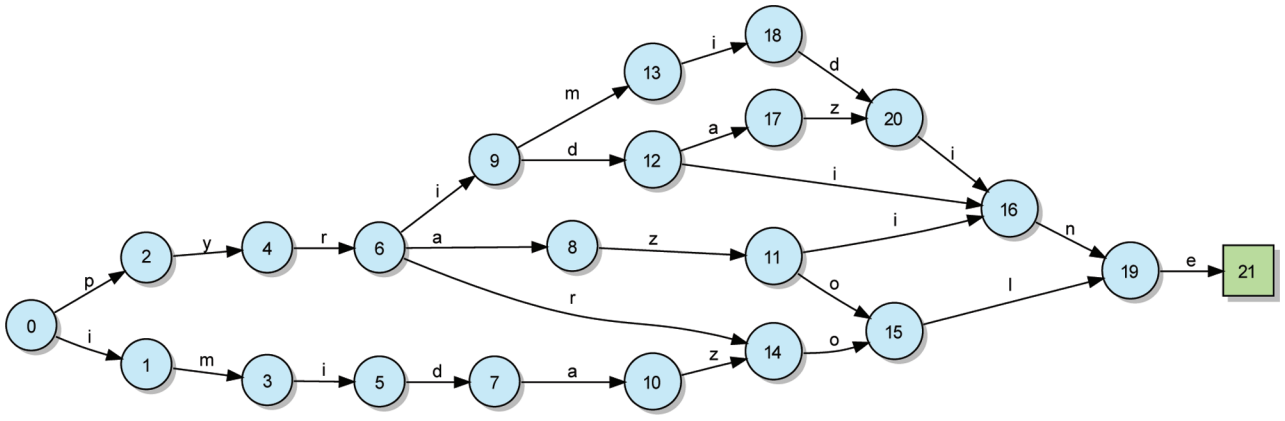


Figure 6. Minimal FSM for recognizing the example dictionary of seven nitrogen-containing heterocycles.

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51–62

```
locant          =     '#' ; /* any digit */

substituent     =     'fluoro' , 'chloro' , 'bromo' ;

alk             =     'meth' , 'eth' , 'prop' , 'but' ;

parent          =     alk 'ane' ;

prefix          =     [ prefix '-' ] [ locant '-' ] substituent ,

                      [ prefix ] substituent ;

name            =     [ prefix [ '-' ] ] parent ;
```

Figure 7. Example of a formal grammar describing a set of halogenated alkanes.

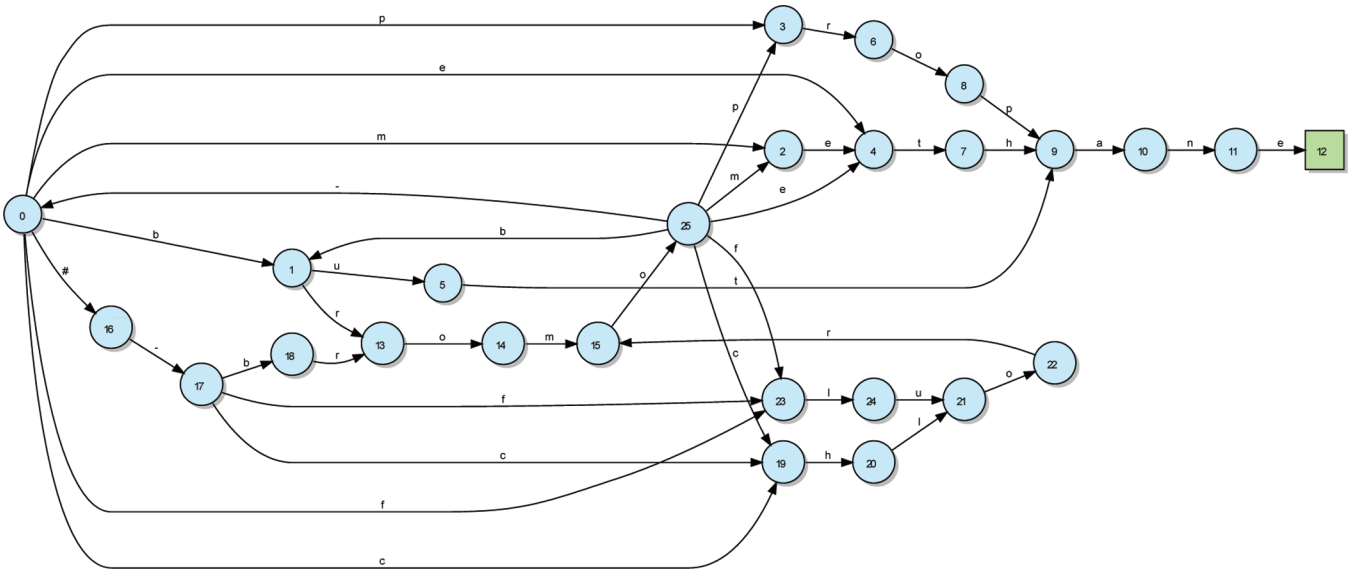


Figure 8. Finite state machine encoding an infinite dictionary of IUPAC-like names.

prefix is recursively defined in terms of itself, becoming an arbitrary long list of substituents, each optionally prefixed by a locant, and all suitably separated by hyphens.

By definition, this grammar recognizes an infinite number of strings that include "methane", "chloroethane", "2-bromopropane", "chloro-bromo-methane", "1-chloro-2-fluoro-ethane", "chlorofluoromethane", and so on. The minimal FSM that corresponds to this grammar is shown in Figure 8. This demonstrates that an infinite number of terms can be represented in a finite, even small, number of states (amount of space). The drawback is that such grammars encode only the syntax of the IUPAC naming rules but not the semantics, allowing this FSM to also accept chemically nonsense strings, such as "9-bromomethane" or "2-chloro-2-chloro-2-chloropropane". In practice, these false positives are not a problem, as the resulting extracted entities are subsequently confirmed by name-to-structure conversion.

One minor technical complication with the approach is that although FSMs can encode "regular grammars", they fail to precisely capture more complex "context-free grammars" such as required by systematic IUPAC-like nomenclature. The distinction between a regular (or Chomsky type-3) grammar and a context-free (or Chomsky type-2) grammar is a subtlety of automata theory.[43] In lay terms, the problem is that parentheses, brackets, and braces need to be balanced and nested in correctly formed IUPAC-like names. Keeping track of how many and which brackets have been opened and closed in an FSM dramatically increases its size and in the general case potentially requires an infinite number of states to allow arbitrarily deep nesting. Our technical solution is to keep track of a parentheses stack outside of the FSM, an arrangement called a "pushdown automaton". This modification simply requires that whenever an open parenthesis, bracket, or brace is encountered, it is pushed onto a stack, whenever a close parenthesis, bracket, or brace is encountered, it matches the type last pushed on the stack, and when the end of input is encountered, the name is only valid if the stack is empty.

The developed CaffeineFix finite state machine that encodes a significant fraction of the IUPAC naming rules for organic chemistry, including numerous CAS and Beilstein naming and traditional variants, currently has a total of 456 387 states. With the implementation described above, requiring 10 bytes per state, the entire FSM requires less than 5 megabytes of memory. This single large FSM currently recognizes 98.97% of the 234 142 Lexichem-generated names for NCI 2000 database, 90.98% of the 71 367 names in the Maybridge 2003 catalogue, and 86.78% of the names in the KeyOrganics 2003 catalogue (the same three benchmarks used in Figure 2). Although still continually improving, these numbers are competitive with the conversion rates seen with mature state-of-the-art name-to-structure tools.

Automatic Spelling Correction. One of the major advantages of using finite state machines (or pushdown automata) to encode dictionaries is the ability to perform

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51−62

"fuzzy" string matching, i.e., to return the set of strings in the dictionary that are similar to the query.[44] When performing an exact lookup, we proceed deterministically from the initial state and test whether we reach an accepting state at the end of the input string. Fuzzy string matching may be implemented by using a back-tracking search, where a limited number of mismatches are allowed between the input string and the words in the dictionary.

Consider the simplest case where, given an input string is not recognized, such as "benzine", we wish to enumerate all the valid strings that are one character substitution away from it. This is called a Hamming distance 1 search.[45] Conceptually, we could potentially enumerate all the possible candidates, by replacing each character of the word with every character in the alphabet. For "benzine", we'd start by considering "aenzine", "cenzine", ..., "zenzine" then "banzine", ..., "bbnzine", eventually reaching "benzinz". For a word of length n and an alphabet of size m, this brute force approach would consider $n*(m-1)$ candidate corrections, that could each be looked up efficiently with the FSM. For just English alphabetic characters, "benzine" would generate 175 candidates, one of which would be "benzene".

A significantly more efficient approach is to use the FSM to guide which substitutions could potentially lead to valid names. By looking at the out-going edges from the initial state, we know the exact set of characters that a term may start with. Likewise at any internal state, the outgoing transitions other than the one matching the current input character denote possible alternatives. This can be implemented efficiently with FSMs using a back-tracking or A* search.[46]

The above example considers only character substitutions and therefore implements "Hamming distance" corrections to the input string, where any suggestion found will be the same length as the input string. The same recursive backtracking approach may be extended to "Levenshtein distance",[47] which also allows insertion and deletion of characters (indels) as single edits. Naturally, this enables the suggestions returned by spelling correction to be longer or shorter than the original string. As an example, "benzene" is suggested as a correction for "benzne" by inserting the missing "e". A further refinement still that is suitable for typed text is to also allow "transpositions" of adjacent characters as a single edit, called "Damerau–Levenshtein distance".[47]

Automatic spelling correction is applied when only a single dictionary entry is found within a given edit-distance search radius. If no entries are found, the string is not recognized. If one entry is found, it is assumed to be the intended spelling. If two or more entries are found, the correction is ambiguous. When being used interactively, these multiple suggested corrections could be presented to a user for manual selection, but for chemical text mining, the need to limit human intervention when processing large volumes of data restricts spelling correction to those forms that can be fixed automatically.

In addition to the simple "benzine" and "benzne" examples above, the full CaffeineFix grammar is able to suggest a number of impressive substitutions. For "12-dichlorobenzene", the algorithm suggests "1,2-dichlorobenzene", for "didec-2-ene" the program suggests "dodec-2-ene", and for "spiro[2.2]-hexane" it suggests "spiro[2.3]hexane".

An interesting quirk of using automatic chemical spelling correction is the need to use a "white word list" when processing text documents. Although the FSM, as described, recognizes a significant fraction of chemical nomenclature, it is ignorant of the English words that also appear in a patent or research article. As a result, it also suggests "heroin" as a replacement for "herein", "uranium" as a replacement for "cranium" and "abilify" as a replacement for "ability". To silence these suggestions, a white word FSM is constructed from a dictionary of English words, and the automatic spelling correction is only applied to strings not recognized as English words or chemical terms.

A current area of investigation is the influence of the definition of edit distance, or similarity measure, on the quality of the suggestions. For example, the Damerau–Levenshtein distance described above is computationally no more powerful than Levenshtein distance, as unlike Hamming distance, both can calculate a distance between any pair of strings. The difference is that for a transposition, such as "beznene", Levenshtein distance considers this two edits away from benzene (either two substitutions or an insertion and a deletion), whereas Damerau–Levenhtein considers it only a single edit away. This difference may be significant in identifying suggestions or resolving ambiguities. The choice of which model of edit distance should also depend on the source errors found in text. In the general case, the cost of individual substitutions, transpositions, insertions, and deletions can be parametrized, potentially from statistics. Indeed, this aspect of spelling correction is very similar to bioinformatics, where sequence searching and alignment algorithms make heavy use of gap penalties and substitution matrices derived from genetic mutation rates, such as BLOSUM and Dayhoff's PAM matrices.[48] In an OCR environment, transpositions are relatively rare, but homoglyphic substitutions, such as between "1", "I", and "l" or between "O" and "0", are relatively common and could be penalized less than between nonhomoglyphic characters such as "t" and ".". For text typed by human beings, on the other hand, transpositions are common, as are phonetic substitutions, and even the distance between keys on a QWERTY keyboard may be useful in identifying "fat-finger" issues.

In the current implementation, insertion/deletion of spaces, line breaks, and the string "
" are considered without penalty, as are substitutions between "1", "l", and "I" and between "0" and "O". As an initial attempt at parametrizing an OCR model, the substitution or "rn" with "m" is considered as free, such that "...arnino..." is corrected to "...amino..." rather than the equally valid but less likely "...arsino...". Efforts are currently underway to implement and evaluate substitution scores based on the OCR frequencies found in real text.

Chemical Entity Name Classification. A useful feature of a chemical entity recognition algorithm is the ability to categorize the classes of entities it returns. In the current work, this categorization is implemented by having different FSMs for different name classes, each identified by a single letter. The main FSM, described above is labeled 'M' (for molecule), that represents a full IUPAC-like systematic name, such as "benzoic acid". A subset of this grammar, representing a functional group prefix or substituent list (equivalent to prefix in the grammar in Figure 7) is available as FSM 'P' (prefix or part), enabling the recognition of common partial names, such as "methyl", "nitro", or "4-fluorophenyl". The 'D' FSM is used for identifying trivial (dictionary) molecule names such as "aspirin" or "ranitidine". The majority of these dictionary entries used in this study were World Health Organization (WHO) International Nonproprietary Names (INNs),[49]

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51–62

United States Adopted Names (USANs),[50] and British Approved Names (BANs).[51]

As mentioned previously, in addition to traditional word-list dictionaries and IUPAC-like grammars, finite state machines can also be used to recognize regular expressions. Regular expressions are used to create FSMs for pharmaceutical registry numbers ('R'), CAS Registry Numbers ('C'), and sum formulas ('S'). Using the 'C' FSM as an example, CAS numbers can be identified as a series of two to seven digits, not starting with a zero, followed by a hyphen, followed by two digits, followed by a hyphen, and followed by a final check digit. The identifier "7732-18-5" is the CAS number for water. Given that when CAS numbers start with two digits, the first digit must be five or higher, a reasonable regular expression might be "(([1-9]\d{2,6})|([5-9]\d))-\d\d-\d" where \d matches any digit, i.e. "[0-9]". An FSM that implements this regexp requires 15 states, which corresponds to 127 nodes/edges in the flattened data structure described above.

The actual CAS number FSM implementation used in CaffeineFix is more complex, in that it confirms the final check digit matches the rest of the number. The final check digit is calculated by summing a series of terms modulo ten. The terms consist of the last digit times one, the previous digit times two, the digit before that by three, and so on. Hence, for the water CAS number, 7732-18-5, the final digit is computed as $(1 \times 8 + 2 \times 1 + 3 \times 2 + 4 \times 3 + 5 \times 7 + 6 \times 7) \bmod 10 = 5$. Fortunately, this computation can be captured by an FSM with 4779 nodes/edges, which can distinguish valid CAS numbers from those with an incorrect check digit. Interestingly, this FSM can be used in conjunction with the backtracking spell checking described above to find possible suggestions for invalid CAS numbers. For the incorrect identifier "7732-18-8", the algorithm suggests nine alternatives that are one edit away: "7732-18-5", "7732-11-8", "77328-18-8", "7733-18-8", "77342-18-8", "77392-18-8", "71732-18-8", "76732-18-8", and "97732-18-8".

In addition to chemical entity names that denote discrete compounds that can be represented by a traditional connection table, a number of chemical text mining programs also annotate chemical terms representing generic structures or compound classes. Although these entities are less useful for indexing and structure searching, CaffeineFix provides a generic ('G') FSM to annotate such terms, to simplify comparison with other chemical entity extractors. Examples of generic terms include "alkyl", "heteroaryl", "halide", and "amino acid".

A number of chemical entity terms occur so frequently in contexts outside of chemistry that they are classified as noise words, or category 'N', by CaffeineFix. These include words such as "air", "alcohol", "lead", "formal", "water", and "salt". To illustrate the obfuscation introduced by such common search terms, the phrase "lead examiner" occurs in the vast majority of US patent filings.

Finally, an analysis of the chemical terms identified by chemical text mining reveals that a disturbing fraction (often 10−20%) are very small, consisting of an element name or composed of a single heavy (nonhydrogen) atom. To simplify the analysis and filtering of these terms, the IUPAC grammar categories molecules ('M') and parts ('P') are split into two further subcategories elements ('E') and atomic parts ('A'). Examples of 'E' terms include "nitrogen" and "water", and examples of 'A' terms include "chloro", "methyl", and "hydroxyl".

To summarize, Table 1 lists the nine FSMs used in the analyses presented in the Result section below. For each category, the table lists its single character code, an example term, the number of terms encoded in the FSM, and the number of nodes/edges in the FSM. The term finite indicates a very large but finite number of terms, i.e., many millions, whereas infinite indicates a grammar based dictionary. Finally, the "Fix" column indicates whether this FSM is used with automatic spelling correction or not.

Table 1. Finite State Machine Dictionaries Used in This Work

code	category	example	words	nodes	fix
M	molecule	benzoic acid	infinite	455 341	Y
D	dictionary	ranitidine	11 196	24 985	Y
R	registry no.	GW-409544	finite	211	N
C	CAS number	7732-18-5	finite	4779	N
E	element	nitrogen	181	510	N
P	prefix	phenyl	infinite	21 517	Y
A	atom prefix	chloro	11	46	N
G	generic	alkane	115	385	N
N	noise	formal	15	61	N

RESULTS AND DISCUSSION

Recall Results. To assess the recall of the described methods, they were applied to the chemical text mining of IBM's text database of around 12 million United States, European, and World patents. Non-English abstracts were translated using OpenEye's Lexichem translation functionality.[52,53] This processing run identified 251 311 877 chemical entities in 8 336 922 patent documents, where identical lexical duplicates within a single document do not contribute to the total. This corresponds to 5 025 487 unique names, of which 4 222 612 could be converted to a SMILES by v2.0.2 of OpenEye's Lexichem to produce 3 329 224 unique SMILES.[54] A breakdown of entities by name category is given in Table 2.

Table 2. Chemical Named Entities Recognized by Category From IBM's Text Database of around 12 million United States, European, and World Patents

code	category	entities	fraction
M	molecule	84 952 033	33.8%
D	dictionary	27 352 706	10.9%
R	registry no.	313 196	0.1%
C	CAS number	324 821	0.2%
E	element	43 219 986	18.6%
P	prefix	42 288 642	18.0%
A	atom prefix	4 363 901	1.8%
G	generic	26 905 972	10.7%
N	noise	17 293 541	6.9%
	total	251 311 877	

These numbers can also be analyzed by location within a patent, showing that 0.6% occur in the title, 2.9% in the abstract, 12.5% in the claims, and the vast majority, 84.0%, in the descriptive text.

This large scale analysis produced some note-worthy observations. The molecule represented by the SMILES string "NCCO" is found as 93 different lexical forms in the 12 million patents, including "2-amino-ethanol", "2-hydroxyethylamine",

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51−62

Table 3. USPTO-50 Benchmark Set of Top-Selling Drug Patents[a]

no.	patent no.	pub. date	trade name	scientific name	MACCS Tanimoto
1	US4231938	19801104	mevacor	lovastatin	1.00
2	US4254129	19810303	allegra	fexofenadine	0.86
3	US4255431	19810310	losec	omeprazole	1.00
4	US4282233	19810804	claritin	loratadine	0.70
5	US4335121	19820615	flovent	fluticasone propionate	0.73
6	US4382938	19830510	ambien	zolpidem	0.75
7	US4503067	19850305	coreg	carvedilol	0.75
8	US4572909	19860225	norvasc	amlodipine	1.00
9	US4636505	19870113	casodex	bicalutamide	0.81
10	US4650884	19870317	celexa	citalopram	1.00
11	US4659516	19870421	faslodex	fulvestrant	0.61
12	US4659716	19870421	clarinex	desloratadine	0.72
13	US4681893	19870721	lipitor	atorvastatin	0.91
14	US4689338	19870825	aldara	imiquimod	1.00
15	US4695590	19870922	zofran	ondansetron	0.78
16	US4721723	19880126	paxil	paroxetine	1.00
17	US4738974	19880419	nexium	esomeprazole	1.00
18	US4746680	19880524	meridia	sibutramine	1.00
19	US4755534	19880705	lamisil	terbinafine	0.81
20	US4816470	19890328	imitrex	sumatriptan	1.00
21	US4847265	19890711	plavix	clopidogrel	0.68
22	US4874794	19891017	abreva	docosanol	1.00
23	US4895841	19900123	aricept	donepezil	0.81
24	US4935437	19900619	arimidex	anastrozole	1.00
25	US4978672	19901218	femara	letrozole	0.88
26	US4990517	19910205	vigamox	moxifloxacin	0.84
27	US5002953	19910326	avandia	rosiglitazone	0.93
28	US5006528	19910409	abilify	aripiprazole	1.00
29	US5006530	19910409	baycol	cerivastatin	0.91
30	US5023269	19910611	cymbalta	duloxetine	1.00
31	US5045552	19910903	aciphex	rabeprazole	1.00
32	US5116863	19920526	patanol	olopatadine	0.62
33	US5153197	19920610	cozaar	losartan	1.00
34	US5196444	19930323	atacand	candesartan	1.00
35	US5270317	19931214	avapro	irbesartan	0.81
36	US5354772	19941011	lescol	fluvastatin	0.73
37	US5360800	19941101	lotronex	alosetron	0.94
38	US5382600	19950117	detrol	tolterodine	1.00
39	US5399578	19950321	diovan	valsartan	0.98
40	US5466823	19951114	celebrex	celecoxib	1.00
41	US5521184	19960528	gleevec	imatinib	0.72
42	US5565447	19961015	avodart	dutasteride	0.64
43	US5616599	19970401	benicar	olmesartan medoxomil	0.88
44	US5633272	19970527	bextra	valdecoxib	1.00
45	US5747498	19980505	tarceva	erlotinib	1.00
46	US5770599	19980623	iressa	gefitinib	1.00
47	US5849911	19981215	reyataz	atazanavir	0.72
48	US5859006	19990112	cialis	tadalafil	0.74
49	US6410550	20020625	chantix	varenicline	0.67
50	US6566360	20030520	levitra	vardenafil	0.96

[a]The text of the 50 patents (XML) and the structures of the corresponding drug molecules (SMILES) are provided as Supporting Information.

"1-amino-2-ethanol", "2-amino-1-ethanol", "2-hydroxyethanamine", "1-amino-ethan-2-ol", "2-amino ethyl alcohol", "(2-hydroxyethyl)amine", and so on. Clearly, if a small molecule with only four heavy atoms can have as many as 93 synonyms, the number of possible permutations of large drug-like molecules (including parentheses and hyphenation variants) could be enormous. Such large numbers of synonyms severely limit the utility of (finite) dictionary-based methods in patent analysis. A second interesting fact is that the entity "cyclosporin A" occurs 15 639 times in a single patent, US20070015693, the largest number of times a single entity appears in a single document. At first, the authors thought this may have been a processing bug, given the coincidental similarity between the count and the last digits of the patent number, but inspection confirmed that this is a formulation patent covering multiple

dx.doi.org/10.1021/ci200463r | J. Chem. Inf. Model. 2012, 52, 51−62

clinical dosing forms, in which the term "cyclosporine A" is indeed mentioned over 15 000 times.

Precision Results. To assess the quality of structures extracted by chemical text mining, CaffeineFix was also evaluated on a pragmatic benchmark of 50 United States patents disclosing top-selling drugs. This benchmark was inspired by the test set proposed by Hattori and co-workers,[55] who describe a set of 30 European and World patents in their paper. Frustratingly, not only do data feeds from World and European patents have a much higher OCR error rate than United States patents (giving an unfair advantage to methods that incorporate spelling correction) but a significant fraction of these patents are in languages other than English. Of the 30 first-to-appear pharmaceutical patents reported by Hattori et al., 6 are in German, 1 is in French, and another is in Dutch. Taking this last one as an example, the first public disclosure of GSK's Flovent was in the Dutch patent NL8100707(A). Presumably, when patents are filed simultaneously in multiple territories, the patent offices with the lower workloads are able to process and publish the patent application faster, leading to a significant fraction of first disclosures in non-English patents.

To produce a public benchmark set that can be used to assess the precision of chemical text mining methods, 50 United States patent documents corresponding to top-selling drugs were selected. These patents are listed in Table 3 (see also Supporting Information). The objective of the benchmark is for each patent to report the highest MDL MACSS 166-bit key[56] Tanimoto value of all the chemical entities discovered in the document to the drug molecule associated with that patent. The choice of MDL MACCS 166-bit keys as a measure of similarity was based upon the wide availability of implementations, including open source options, allowing these results to be reproduced and comparable figures presented by other authors. A Tanimoto of 1.00 is interpreted as a hit where the drug was correctly identified and extracted from the text. High Tanimoto values (but less than 1.00) are treated as near misses. The values reported in the final column of Table 3 describe the results obtained with the CaffeineFix algorithm described in this work, when used in conjunction with OpenEye's Lexichem name-to-structure software.

These results, while perhaps disappointing on a chemical text mining scale, are competitive with state-of-the-art annotation tools. The 21 hits for CaffeineFix/Lexichem compare with 20 hits for IBM SIMPLE's annotation pipeline (which uses CambridgeSoft's name=struct for name-to-structure conversion) and 17 hits for the University of Cambridge's OSCAR, which uses OPSIN for name-to-structure conversion. The surprise is that all of these automated methods have only about a 40% success rate of correctly associating the drug molecule with the patent, despite the often hundreds or thousands of chemical entities discovered in these patents.

One interesting observation of the hit lists is the relatively low overlap of which drugs are "hit". Comparing the 20 hits from IBM Simple to CaffeineFix/Lexichem's hits shows 14 hits in common, indicating that there is a significant benefit in using either multiple annotation pipelines or equally likely multiple name-to-structure programs with each entity recognition engine. An example of a hit reported by IBM SIMPLE but not by CaffeineFix/Lexichem is Avandia, where the relevant systematic name ends with "...-2,4-thiazolidinedione". Inspecting the CaffeineFix results shows that the name is correctly tokenized and identified, but in this case, Lexichem fails to convert it to a structure due to the ambiguity of whether the "2,4" locants apply to the "thiazolidine" or to the "dione".

The influence on automatic spelling correction, even on relatively clean United States patent data, can be seen by the fact that only 13 drugs are hits without any form of correction, 4 more are recognized at distance 0 (i.e., deleting space and hyphens, and allowing homoglyphic substitutions), while another 4 require an edit distance of 1. In noisy data feeds, the impact of automatic spelling correction would be expected to be more pronounced.

CONCLUSIONS

The major causes of failing to annotate chemical structures in patent documents are the difficulty with correctly tokenizing systematic chemical names and the high frequency of typographical issues found in patent office data feeds. To address both of these issues, a FSM-based method has been developed that allows the efficient recognition of an infinite number of systematic IUPAC-like chemical names in free text and even in the presence of OCR and other typographical errors. This technique is shown to improve the extraction of relevant structures from pharmaceutical patents.

ASSOCIATED CONTENT

Supporting Information

Text of the 50 patents (XML) and the structures of the corresponding drug molecules (SMILES) for the USPTO-50 benchmark set of top-selling drug patents. This information is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*E-mail: sorel.muresan@astrazeneca.com.

ACKNOWLEDGMENTS

The authors would like to thank Plamen Petrov and Jon Winter at AstraZeneca for their contributions to this project.

REFERENCES

(1) *CAS Registry Numbers*; http://www.cas.org/index.html (accessed November 11, 2011).

(2) *Thomson Reuters*; http://thomsonreuters.com/products_services/science/science_products/scientific_research/drug_discovery_development/chemistry_research/ (accessed November 11, 2011).

(3) *Chemical Information Mining: Facilitating Literature-Based Discovery*; Banville, D. L., Ed.; CRC Press: Boca Raton, FL, 2009.

(4) Suriyawongkul, I.; Southan, C.; Muresan, S. The Cinderella of Biological Data Integration: Addressing Some of the Challenges of Entity and Relationship Mining from Patent Sources. In *Data Integration in the Life Sciences*; Lambrix, P., Kemp, G., Eds.; Springer: Berlin, Heidelberg, Germany, 2010; pp 106−121.

(5) Lin, L. S.; Lanza, T. J.; Jewell, J. P.; Liu, P.; Shah, S. K.; Qi, H.; Tong, X.; Wang, J.; Xu, S. S.; Fong, T. M.; Shen, C.-P.; Lao, J.; Xiao, J. C.; Shearman, L. P.; Stribling, D. S.; Rosko, K.; Strack, A.; Marsh, D. J.; Feng, Y.; Kumar, S.; Samuel, K.; Yin, W.; Van der Ploeg, L. H. T.; Goulet, M. T.; Hagmann, W. K. Discovery of N-[(1S,2S)-3-(4-Chlorophenyl)-2-(3-cyanophenyl)-1-methylpropyl]-2-methyl-2-{[5-(trifluoromethyl)pyridin-2-yl]oxy}propanamide (MK-0364), a Novel, Acyclic Cannabinoid-1 Receptor Inverse Agonist for the Treatment of Obesity. *J. Med. Chem.* **2006**, *49*, 7584−7587.

(6) Hagmann, W. K.; Lin, L. S.; Shah, S. K.; Guthikonda, R. N.; Qi, H.; Chang, L. L.; Liu, P.; Armstrong, H. M.; Jewell, J. P.; Lanza, T. J. J. *Substituted Amides*. US 2004/0058820, 2004.

(7) International Union of Pure and Applied Chemists (IUPAC). *A Guide to IUPAC Nomenclature of Organic Compounds, Recommendations 1993*; Blackwell Scientific Publications: London, U.K., 1993.
(8) Naming and Indexing of Chemical Substances for Chemical Abstracts. *Appendix IV of CA Index Guide, Chemical Abstracts Service (CAS)*; http://www.cas.org/ASSETS/58D34DD3892142D18F5C3B0A004D3A0C/indexguideapp.pdf (accessed November 11, 2011).
(9) Kolářik, C.; Klinger, R.; Friedrich, C.; Hofmann-Apitius, M.; Fluck, J. In *Chemical Names: Terminological Resources and Corpora Annotation*, LREC 2008 Workshop, May 26, 2008, Marrakech, Morocco; http://www.scai.fraunhofer.de/fileadmin/images/bio/data_mining/paper/kolarik2008.pdf.
(10) Klinger, R.; Kolářik, C.; Fluck, J.; Hofmann-Apitius, M.; Friedrich, C. M. Detection of IUPAC and IUPAC-like chemical names. *Bioinformatics* **2008**, *24*, i268−276.
(11) Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31−36.
(12) *InChI: The IUPAC Chemical Identifier*; http://www.iupac.org/inchi/ (accessed November 11, 2011).
(13) Corbett, P.; Murray-Rust, P. High-Throughput Identification of Chemistry in Life Science Texts. In *Computational Life Sciences II*; Berthold, M., Glen, R., Fischer, I., Eds.; Springer: Berlin, Heidelberg, Germany, 2006; pp 107−118.
(14) Corbett, P.; Batchelor, C.; Teufel, S. Annotation of chemical named entities. In *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*, Association for Computational Linguistics: Prague, Czech Republic, 2007; pp 57−64.
(15) Lowe, D. M.; Corbett, P. T.; Murray-Rust, P.; Glen, R. C. Chemical Name to Structure: OPSIN, an Open Source Solution. *J. Chem. Inf. Comput. Sci.* **2011**, *51*, 739−753.
(16) Hanisch, D.; Fundel, K.; Mevissen, H.-T.; Zimmer, R.; Fluck, J. ProMiner: rule-based protein and gene entity recognition. *BMC Bioinformatics* **2005**, *6* (Suppl 1), S14.
(17) Rebholz-Schuhmann, D.; Kirsch, H.; Arregui, M.; Gaudan, S.; Riethoven, M.; Stoehr, P. EBIMed—text crunching to gather facts for proteins from Medline. *Bioinformatics* **2007**, *23*, e237−e244.
(18) Narayanaswamy, M.; Ravikumar, K. E.; Vijay-Shanker, K. A biological named entity recognizer. *Pac. Symp. Biocomput.* **2003**, *8*, 427−438.
(19) Kemp, N.; Lynch, M. Extraction of Information from the Text of Chemical Patents. 1. Identification of Specific Chemical Names. *J. Chem. Inf. Comput. Sci.* **1998**, *38*, 544−551.
(20) Wu, X.; Zhang, L.; Chen, Y.; Rhodes, J.; Griffin, T. D.; Boyer, S. K.; Alba, A.; Cai, K. ChemBrowser: A Flexible Framework for Mining Chemical Documents. In *Advances in Computational Biology*; Arabnia, H. R., Ed.; Springer: New York, 2011; pp 57−64.
(21) *Lexichem*, version 2.0.2; OpenEye Scientific Software: Santa Fe, NM; http://www.eyesopen.com/lexichem-tk (accessed November 11, 2011).
(22) *Struct=Name*; CambridgeSoft, Cambridge, MA;http://www.cambridgesoft.com/software/details/?ds=5 (accessed November 11, 2011).
(23) *ACD/Name*; ACD/Labs: Toronto, Canada; http://www.acdlabs.com/products/draw_nom/nom/name/ (accessed November 11, 2011).
(24) *Name <> Structure*; ChemAxon: Budapest, Hungary; http://www.chemaxon.com/products/name-to-structure/ (accessed November 11, 2011).
(25) *IUPAC NameIt and DrawIt*; Bio-Rad Laboratories: Irvine, CA; http://www3.bio-rad.com/pages/SAD/docs/95925-IUPAC_DS.pdf#zoom=75% (accessed November 11, 2011).
(26) *NameExpert*; ChemInnovation Software Inc.: San Diego, CA; http://www.cheminnovation.com/products/nameexpert.asp (accessed November 11, 2011).
(27) *ICN2S*; InfoChem: Munich, Germany; http://infochem.de/mining/icn2s.shtml (accessed November 11, 2011).
(28) *NCI Open Database Compounds*; NCI: Bethesda, MD; http://cactus.nci.nih.gov/download/nci/ (accessed November 11, 2011).
(29) *KeyOrganics*; Key Organics: London, U.K.; http://www.keyorganics.co.uk/Downloads (accessed November 11, 2011).
(30) *Maybridge*; Thermo Fisher Scientific: Waltham, MA; http://www.maybridge.com (accessed November 11, 2011).
(31) Kukich, K. Techniques for Automatically Correcting Words in Text. *ACM Comput. Surv.* **1992**, *24*, 377−439.
(32) Oflazer, K.; Güzey, C. In *Spelling Correction in Agglutinative Languages*; The 4th ACL Conference on Applied Natural Language Processing, October 13−15, 1994, Stuttgart, Germany; Association for Computational Linguistics: Stroudsburg, PA; http://dx.doi.org/10.3115/974358.974406.
(33) Pollock, J. J.; Zamora, A. Collection and characterization of spelling errors in scientific and scholarly text. *J. Am. Soc. Inf. Sci.* **1983**, *34*, 51−58.
(34) Pollock, J. J.; Zamora, A. Spelling correction in scientific and scholarly text. *Commun. ACM* **1984**, *27*, 358−368.
(35) Kirby, G. H.; Lord, M. R.; Rayner, J. D. Computer translation of IUPAC systematic organic chemical nomenclature. 6. (Semi)automatic name correction. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 153−160.
(36) *Converting Chemical Names to Structures with Name=Struct*; http://www.cambridgesoft.com/services/DesktopSupport/Documentation/N2S/batch/index.htm (accessed November 11, 2011).
(37) Lucchesi, C. L.; Kowaltowski, T. Applications of finite automata representing large vocabularies. *Software: Pract. Exp.* **1993**, *23*, 15−30.
(38) Knuth, D. E. *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed.; Addison-Wesley: Reading, MA, 1998.
(39) Fredkin, E. Trie memory. *Commun. ACM* **1960**, *3*, 490−499.
(40) Watson, B. A Taxonomy of Algorithms for Constructing Minimal Acyclic Deterministic Finite Automata. In *Automata Implementation*; Boldt, O., Jürgensen, H., Eds.; Springer: Berlin, Heidelberg, Germany, 2001; pp 174−182.
(41) Aho, A. V.; Lam, M. S.; Sethi, R.; Ullman, J. D. *Compilers: Principles, Techniques and Tools*, 2nd ed.; Pearson Education, Inc.: Boston, MA, 1986.
(42) Friedl, J. E. F. *Mastering Regular Expressions*. O'Reilly Media Inc.: Sebastopol, CA, 1997.
(43) Chomsky, N. Three models for the description of language. *IRE Trans. Inf. Theory* **1956**, *2*, 113−124.
(44) Oflazer, K. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Comput. Linguist.* **1996**, *22*, 73−89.
(45) Damerau, F. J. A technique for computer detection and correction of spelling errors. *Commun. ACM* **1964**, *7*, 171−176.
(46) Hart, P. E.; Nilsson, N. J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100−107.
(47) Navarro, G. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)* **2001**, *33*, 31−88.
(48) Henikoff, S.; Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U.S.A.* **1992**, *89*, 10915−10919.
(49) *International Nonproprietary Names (INN)*; http://www.who.int/medicines/services/inn/en/ (accessed November 11, 2011).
(50) *United States Adopted Names*; http://www.ama-assn.org/ama/pub/physician-resources/medical-science/united-states-adopted-names-council.page (accessed November 11, 2011).
(51) *British Approved Names*; http://www.pharmacopoeia.gov.uk/publications/british-approved-names.php (accessed November 11, 2011).
(52) Sayle, R. Foreign Language Translation of Chemical Nomenclature by Computer. *J. Chem. Inf. Model.* **2009**, *49*, 519−530.
(53) Sayle, R. Preserving Nuance in Chemical Nomenclature Translation. *The ATA Chronicle (The Journal of the American Translators Association)* **2009**, *38*, 22−29.
(54) Muresan, S. Automated spelling correction to improve recall rates of name-to-structure tools for chemical text mining, ChemAxon's

European User Group Meeting, May 17−18, 2011, Budapest, Hungary; http://www.chemaxon.com/library/automated-spelling-correction-to-improve-recall-rates-of-name-to-structure-tools-for-chemical-text-mining/ (accessed November 11, 2011).
(55) Hattori, K.; Wakabayashi, H.; Tamaki, K. Predicting key example compounds in competitors' patent applications using structural information alone. *J. Chem. Inf. Model.* **2008**, *48*, 135−142.
(56) Durant, J. L.; Leland, B. A.; Henry, D. R.; Nourse, J. G. Reoptimization of MDL Keys for Use in Drug Discovery. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 1273−1280.