Free Text Phrase Encoding and Information Extraction from Medical Notes

by

Jennifer Shu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author ....
Department of Electrical Engineering and Computer Science
August 16, 2005

Certified by ....
Roger G. Mark
Distinguished Professor in Health Sciences & Technology
Thesis Supervisor

Certified by ....
Peter Szolovits
Professor of Computer Science
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Free Text Phrase Encoding and Information Extraction from Medical Notes

by

Jennifer Shu

Submitted to the Department of Electrical Engineering and Computer Science
on August 16, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Laboratory for Computational Physiology is collecting a large database of patient signals and clinical data from critically ill patients in hospital intensive care units (ICUs). The data will be used as a research resource to support the development of an advanced patient monitoring system for ICUs. Important pathophysiologic events in the patient data streams must be recognized and annotated by expert clinicians in order to create a "gold standard" database for training and evaluating automated monitoring systems. Annotating the database requires, among other things, analyzing and extracting important clinical information from textual patient data such as nursing admission and progress notes, and using the data to define and document important clinical events during the patient's ICU stay. Two major text-related annotation issues are addressed in this research. First, the documented clinical events must be described in a standardized vocabulary suitable for machine analysis. Second, an advanced monitoring system would need an automated way to extract meaning from the nursing notes, as part of its decision-making process. The thesis presents and evaluates methods to code significant clinical events into standardized terminology and to automatically extract significant information from free-text medical notes.

Thesis Supervisor: Roger G. Mark
Title: Distinguished Professor in Health Sciences & Technology

Thesis Supervisor: Peter Szolovits
Title: Professor of Computer Science

Acknowledgments

I would like to thank my two thesis advisors, Dr. Mark and Prof. Szolovits, for all their help with my thesis, Gari and Bill for their guidance and support, Margaret for providing the de-identified nursing notes and helping with part of speech tagging, Neha for helping with the graph search algorithm, Tin for his help with testing, Ozlem and Tawanda for their advice, and Gari, Bill, Andrew, Brian, and Dr. Mark for all their hard work tagging data for me. This research was funded by Grant Number R01 EB001659 from the National Institute of Biomedical Imaging and Bioengineering (NIBIB).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The MIT Laboratory for Computational Physiology (LCP) and the MIT Clinical Decision Making Group are involved in a research effort to develop an advanced patient monitoring system for hospital intensive care units (ICUs). The long-term goal of the project is to construct an algorithm that can automatically extract meaning from a patient's collected clinical data, allowing clinicians to easily define and track the patient's physiologic state as a function of time. To achieve this goal, a massive, comprehensive multi-parameter database of collected patient signals and associated clinical data, MIMIC II [38, 29], is being assembled and needs to be annotated [8]. The database, once annotated, will serve as a testbed for multi-parameter algorithms that will be used to automate parts of the clinical care process.

This thesis deals specifically with two text-related facets of annotation. First, during the annotation of data, clinicians enter a free-text phrase to describe what they believe are significant clinical events (e.g., *cardiogenic shock*, *pulmonary edema*, or *hypotension*) in a patient's course. In order for the descriptions to be available in a standardized format for later machine analysis, and at the same time to allow the annotators to have expressive freedom, there must exist a method to code their free-text descriptions into an extensive standardized vocabulary. This thesis presents and evaluates an algorithm to code unstructured descriptions of clinical concepts into a structured format. This thesis also presents an automated method of extracting important information from available free text data, such as nursing admission and progress notes. Automatic extraction and coding of text not only accelerate expert annotation of a patient's medical data, but also may aid online hypothesis construction and patient course prediction, thus improving patient care and possibly improving outcomes. Important information that needs to be extracted from the nursing progress notes includes the patient's diagnoses, symptoms, medications, treatments, and laboratory tests. The extracted medical concepts may then be translated into a standardized medical terminology with the help of the coding algorithm. To test the performance of various extraction algorithms, a set of clinical nursing notes was manually tagged with three different phrase types (medications, diseases, and symptoms) and then used as a "gold standard" corpus to train statistical semantic tagging methods.

1.1 The MIMIC II Database

The MIMIC II database includes physiologic signals, laboratory tests, nursing flow charts, clinical progress notes, and other data collected from patients in the ICUs of the Beth Israel Deaconess Medical Center (BIDMC). Expert clinicians are currently reviewing each case and annotating clinically significant events, which include, but are not limited to, diseases (e.g., *gastrointestinal bleed*, *septic shock*, or *hemorrhage*), symptoms (e.g., *chest pain* or *nausea*), significant medication changes, vital sign changes (e.g., *tachycardia* or *hypotension*), waveform abnormalities (e.g., *arrhythmias* or *ST elevation*), and abnormal laboratory values. The annotations will be used to train and test future algorithms that automatically detect significant clinical events, given a patient's recorded data.

The nursing admission and progress notes used in this research are typed in free text (i.e., natural language without a well-defined formal structure) by the nurses at the end of each shift. The notes contain such information as symptoms, physical findings, procedures performed, medications and dosages given to the patient, interpretations of laboratory test results, and social and medical history. While some other hospitals currently use structured input (such as dropdown lists and checkboxes) to enter clinical notes, the BIDMC currently uses a free-text computer entry system to record nursing notes. There are both advantages and disadvantages of using a free-text system. Although having more structured input for nursing notes would facilitate subsequent machine analysis of the notes, it is often convenient for nurses to be able to type patient notes in free text instead of being constrained to using a formal vocabulary or structure. Detail may also be lost when nurses are limited to using pre-selected lists to describe patient progress.

1.2 Annotation Process

During the process of annotating the database, annotators review a patient's discharge summary, progress notes, time series of vital signs, laboratory tests, fluid balance, medications, and other data, along with waveforms collected from beside monitors, and mark what they believe to be the points on the timeline where significant clinical events occur. At each of those important points in the timeline, they attach a state annotation, labeled with a description of the patient's state (e.g., *myocardial infarction*). The annotators also attach to each state annotation one or more flag annotations, each of which is a piece of evidence (e.g., *chest pain* or *shortness of breath*) that supports the state annotation. See [8, 9] for a fuller description of the Annotation Station and the annotation process. An algorithm was developed to code each of the state and flag annotation labels with one or more clinical concepts. The aim is to eventually create an annotated database of patient data where each of the state annotations and flag annotations is labeled with a clinical concept code.

1.3 Medical Vocabulary

Free-text coding is needed to translate the free-text descriptions or phrases into codes from a medical vocabulary, providing a standardized way of describing the clinical concepts. The medical vocabulary that is being used for annotating MIMIC II data is a subset of the 2004AA version of the National Library of Medicine's Unified Medical

Language System (UMLS) [33], a freely available collection of over one hundred medical vocabularies that identify diseases, symptoms, and other clinical concepts. Each unique clinical concept is assigned a concept code (a unique alpha-numeric identifier), and the concept generally has several different synonyms. For example, *heart attack* and *myocardial infarction* represent the same concept, and both strings are mapped to the same unique UMLS concept code.

The UMLS was designed to help facilitate the development of automated computer programs that can understand clinical text [31], and its knowledge sources are widely used in biomedical and health-related research. In addition to the information included in all of the source vocabularies (referred to as the Metathesaurus), the UMLS contains additional semantic and syntactic information to aid in natural language processing (NLP). The SPECIALIST Lexicon is a collection of syntactic, morphological, and orthographic information for commonly used English and medical terms. It includes commonly used abbreviations and spelling variants for words, as well as their parts of speech. The Semantic Network categorizes each concept and links multiple concepts together through various types of relationships [33, 25].

1.4 Free-Text Coding

The free-text coding component of this research focuses on the development of an interactive algorithm that converts free-text descriptions or phrases into one or more UMLS codes. A graphical user interface has been developed to incorporate this algorithm into the annotation software [9]. The program is invoked when an annotation label needs to be coded, thereby making the MIMIC II annotations useful for later machine analysis.

There are several challenges to translating free-text phrases into standardized terminology. The search for concept codes must be accurate and rapid enough that annotators do not lose patience. Annotators are also prone to making spelling mistakes and often use abbreviations that may have more than one meaning. Furthermore, the same UMLS concept may be described in various different ways, or annotators might wish to code a concept that simply does not exist in the UMLS. Sometimes the annotator might not be satisfied with the level of specificity of codes returned and may want to look at related concepts. These issues are addressed and comparisons of accuracy and search times are made for a variety of medical phrases.

1.5 Extraction of Significant Concepts from Notes

As the other main component of this research, algorithms were developed to automatically find a subset of significant phrases in a nursing note. Such algorithms will be a part of the long-term plan to have a machine use collected patient data to automatically make inferences about the patient's physiologic state over time. Given a progress note as input, these algorithms output a list of the patient's diagnoses, symptoms, medications, treatments, and tests, which may further be coded into UMLS concepts using the free text phrase encoding algorithm.

Unstructured nursing notes are difficult to parse and analyze using automatic algorithms because they often contain spelling errors and improper grammar and punctuation, as well as many medical and non-medical abbreviations. Furthermore, nurses have different writing habits and may use their own abbreviations and formatting. Natural language analysis can be helpful in creating a method to automatically find places in the notes where important or relevant medical information is most likely to exist. For example, rule-based or statistical tagging methods can be used to assign a part of speech (e.g., *noun* or *verb*) or other type of categorization (e.g., *disease or symptom*) to each word in a text. The tagged words can then be grouped together to form larger structures, such as noun phrases or semantic phrases. Tagging a representative group of texts, and then forming new grammatical or semantic assumptions from them (e.g., a disease is most likely to be a noun phrase, or a medication is most likely preceded by a number), helps to identify places in the text that contain words of interest. Such methods are explored and evaluated in this research.

1.6 Related Work

Over the past several decades, many projects have been undertaken in the biomedical and natural language communities to analyze medical notes and extract meaning from them using computers. One such project is the Medical Language Extraction and Encoding System (MedLEE) [16, 14, 15], created by Carol Friedman at Columbia University. The system uses natural language processing to extract clinical information from unstructured clinical documents, and then structures and encodes the information into a standardized terminology. Although MedLEE is designed for specific types of medical documents, such as discharge summaries, radiology reports, and mammography reports, the current online demo version [30] generally performs well on the BIDMC nursing notes. It is able to extract phrases such as problems, medications, and procedures, along with their UMLS codes. However, it does make some mistakes, such as not recognizing certain abbreviations (e.g., "CP" for *chest pain*, "pulm" for *pulmonary*, and "levo," which can stand for a number of different drug names). The system also gives some anomalous results, such as the word "drinks" in the sentence "eating full diet and supplemental drinks" being coded into a problem, *drinks alone.* Furthermore, the demo version of MedLEE does not recognize words that have spelling errors. Although the system can be run via a web interface, the source code for their tools is not readily accessible, nor is the most recent and comprehensive version of MedLEE available online.

Another relevant project is Naomi Sager's Linguistic String Project, the goal of which is to use natural language processing to analyze various types of texts, including medical notes. The group has done work in defining *sublanguage grammars* to characterize free-text medical documents and using them to extract the information from the documents into a structured database [39]. However, their source code is also not currently available.

The Link Grammar Parser [26] is another such tool that attempts to assign syntactic structure to sentences, although it was not designed specifically to analyze medical notes. The parser uses a lexicon and grammar rules to assign parts of speech to words in a sentence and syntactic structure to the phrases in the sentence. However, currently, the parser's grammatical rules are too strict and cannot handle phrases or "ungrammatical" sentences such as those in the nursing notes. Some work has been done to expand the Link Parser to work with medical notes [41, 12]; however, the use of a medical lexicon was not found to significantly improve the performance of the parser.

Zou's IndexFinder [7] is a program designed to quickly retrieve potential UMLS codes from free text phrases and sentences. It uses in-memory tables to quickly index concepts based on their normalized string representations and the number of words in the normalized phrase. The authors argue that IndexFinder is able to find a greater number of specific concepts and perform faster than NLP-based approaches, because it does not limit itself to noun phrases and does not have the high overhead of NLP approaches. IndexFinder is available in the form of a web interface [2] that allows users to enter free text and apply various types of semantic and syntactic filtering. Although IndexFinder is very fast, its shortcomings, such as missing some common nursing abbreviations such as "mi" and not correcting spelling mistakes, are similar to those of MedLEE. As of this writing, their source code was not publicly available. However, IndexFinder's approaches are useful for efficient coding and are explored in this research.

The National Library of Medicine has various open source UMLS coding tools available that perform natural language processing and part-of-speech tagging [25]. Although some of these tools are still in development and have not been released, the tools that are available may be helpful in both coding free text and analyzing nursing notes. MetaMap Transfer (MMTx) [3, 10] is a suite of software tools that the NLM has created to help parse text into phrases and code the phrases into the UMLS concepts that best cover the text. MetaMap has some problems similar to those of previously mentioned applications, in that it does not recognize many nursing abbreviations and by default does not spell check words. Nevertheless, because the tools are both free and open source, and are accessible through a Java API, it is easy to adapt their tools and integrate them into other programs. MetaMap and other NLM tools are utilized in this research and their performance is evaluated.

The Clinical Decision Making Group has projects in progress to automatically extract various types of information from both nursing notes and more formally-written discharge summaries [27]. Currently, some methods have been developed for tokenizing and recognizing sections of the nursing notes using pattern matching and UMLS resources. Additionally, algorithms have been developed to extract diagnoses and procedures from discharge summaries. This thesis is intended to contribute to the work being done in these projects.

1.7 Thesis Outline

In this thesis, a semi-automated coding technique, along with its user interface, is presented. The coding algorithm makes use of abbreviation lists and spelling dictionaries, and proceeds through several stages of searching in order to present the most likely UMLS concepts to the user. Additionally, different methodologies for medical phrase extraction are compared. In order to create a gold standard corpus to be used to train and test statistical algorithms, an exhaustive search method was first used to initially tag diseases, medications, and symptoms in a corpus of nursing notes. Then, several people manually made any necessary corrections to the tags, creating a gold standard corpus that was used for training and testing. The clinical phrases were then extracted using the statistical training data and a medical lexicon. Comparisons are made between the exhaustive search method, automated method, and gold standard.

Chapter 2 presents and evaluates an algorithm for coding free-text phrases into a standardized terminology. Chapter 3 details the creation of the gold standard corpus of tagged nursing notes, and Chapter 4 describes methods to automatically extract significant clinical terms from the notes. Finally, conclusions and future work are presented in Chapter 5.

Chapter 2

Automatic Coding of Free-Text Clinical Phrases

A method of coding free-text clinical phrases was developed both to help in labelling MIMIC II annotations and to be used as a general resource for coding medical free text. The system can be run both through a graphical user interface and through a command-line interface. The graphical version of the coding application has been integrated into the Annotation Station software [9], and it can also be run standalone. Additionally, the algorithm can be run via an interactive command-line interface, or it can be imbedded into other software applications (for example, to perform batch encoding of text without manual intervention).

As outlined in the previous chapter, there are many difficulties that occur in the process of coding free-text phrases, including spelling mistakes, ambiguous abbreviations, and combinations of events that cannot be described with a single UMLS code. Furthermore, because annotators will spend many hours analyzing and annotating the data from each patient, the free-text coding stage must not be a bottleneck; it is desirable that the retrieval of code candidates not take more than a few seconds. Results should be returned on the first try if possible, with the more relevant results at the top. The following sections describe the search procedure and resources used in the coding algorithm, as well as the user interface for the application that has been developed.

2.1 SNOMED-CT Vocabulary

The medical terminology used for coding MIMIC II annotations was limited to the subset of the UMLS containing the SNOMED-CT [18, 19] source vocabulary. SNOMED-CT is a hierarchical medical nomenclature formed by merging the College of American Pathologists' Systematized Nomenclature of Medicine (SNOMED) with the UK National Health Service's Read Clinical Terms (CT). SNOMED-CT contains a collection of concepts, descriptions, and relationships and is rapidly becoming an international standard for coding medical concepts. Each concept in the vocabulary represents a clinical concept, such as a disease, symptom, intervention, or body part. Each unique concept is assigned a unique numeric identifier or code, and can be described by one or more terms or synonyms. In addition, there are many types of relationships that link the different concepts, including hierarchical (*is-a*) relationships and attribute relationships (such as a body part being the *finding site* of a certain disease). Because of the comprehensiveness and widespread use of the SNOMED-CT vocabulary in the international healthcare industry, this terminology was chosen to represent the MIMIC II annotation labels.

The 2004AA version of the UMLS contains over 1 million distinct concepts, with over 277,000 of these concepts coming from the SNOMED-CT (January 2004) source vocabulary. The UMLS captures all of the information contained in SNOMED-CT, but is stored within a different database structure. The NLM has mapped each of the unique SNOMED-CT concept identifiers into a corresponding UMLS code. Because the free-text coding application presented in this research was designed to work with the UMLS database structure, other UMLS source vocabularies (or even the entire UMLS) can be substituted for the SNOMED-CT subset without needing to modify the application's source code.

2.2 Resources Used

The Java-based application that has been developed encodes significant clinical events by retrieving the clinical concepts that most closely match a free-text input phrase. To address the common coding issues mentioned above, the system makes use of an open-source spell-checker, a large list of commonly used medical abbreviations, and a custom abbreviation list, as well as normalized word tables created from UMLS data. This section describes these features in detail.

2.2.1 Medical Abbreviations

One of the most obvious difficulties with trying to match a free text phrase with terms from a standardized vocabulary is that users tend to use shorthand or abbreviations to save time typing. It is often difficult to figure out what an abbreviation stands for because it is either ambiguous or does not exist in the knowledge base. The UMLS contains a table of abbreviations and acronyms and their expansions [32], but the table is not adequate for a clinical event coding algorithm because it lacks many abbreviations that an annotator might use, and at the same time contains many irrelevant (non-medical) abbreviations. Therefore, a new abbreviation list was created by merging the UMLS abbreviations with an open source list of pathology abbreviations and acronyms [11], and then manually filtering the list to remove redundant abbreviations (i.e., ones with expansions consisting of variants of the same words) and abbreviations that would likely not be crucial to the meaning of a nursing note (e.g., names of societies and associations or complex chemical and bacteria names). The final list is a text file containing the abbreviations and their expansions.

2.2.2 Custom Abbreviations

When reviewing a patient's medical record, annotators often wish to code the same clinical concept multiple times. Thus, a feature was added to give users the option to link a free-text term, phrase, or abbreviation directly to one or more UMLS concept codes, which are saved in a text file and available in later concept searches. For example, the annotator can link the abbreviation *mi* to the concept code $C1$, the identifier for *myocardial infarction.* On a subsequent attempt to code *mi*, the custom abbreviation list is consulted, and *myocardial infarction* is guaranteed to be one of the top concepts returned. The user can also link a phrase such as *tan sxns* to both *tan* and *secretions.* This feature also addresses the fact that the common medical abbreviation list sometimes does not contain abbreviations that annotators use.

2.2.3 Normalized Phrase Tables

Many coding algorithms convert free-text input phrases into their *normalized* forms before searching for the words in a terminology database. The NLM Lexical Systems Group's [22] Norm [23] tool (which is included in the Lexical Tools package) is a configurable program with a Java API that takes a text string and translates it to a normalized form. It is used by the NLM to generate the UMLS normalized string table, MRXNS_ENG. The program removes genitives, punctuation, stop words, and diacritics, splits ligatures, converts all words to lowercase, uninflects the words, ignores spelling variants (e.g., *color* and *colour* are both normalized to *color*), and alphabetizes the words [23]. A stop word is defined as a frequently occurring word that does not contribute much to the meaning of a sentence. The default stop words that are removed by the Norm program are *of*, *and*, *with*, *for*, *nos*, *to*, *in*, *by*, *on*, *the*, and *(non mesh)*.

Normalization is useful in free-text coding programs because of the many different forms that words and phrases can take on. For example, *lower leg swelling* can also be expressed as *swelling of the lower legs* and *swollen lower legs.* Normalizing any of those phrases would create a phrase such as *leg low swell*, which can then be searched for in MRXNS_ENG, which consists of all UMLS concepts in normalized form. A problem with searching for a phrase in the normalized string table, however, is that sometimes only part of the phrase will exist as a concept in the UMLS. Thus, to search for all partial matches of *leg low swell*, up to 7 different searches might have to be performed (*leg low swell*, *leg low*, *low swell*, *leg swell*, *leg*, *low*, and *swell*). In general, for a phrase of n words, $2^n - 1$ searches would have to be performed.

Table 2.1: The structure of the INDEXED_NSTR table, which contains all of the unique normalized strings from the UMLS MRXNS_ENG table, sorted by the number of words in each phrase and each row given a unique row identifier.

row_id	cuis	nstr	numwords
132	C7	leg	1
224	C1,C2,C3	low	1
301	C4,C5	swell	1
631	C7	leg low	2
632	C6	leg swell	2
789	C8	leg low swell	3

Two new database tables were created to improve the efficiency of normalized string searches. Based on IndexFinder's *Phrase table* [37], a table was created by extracting all of the unique normalized strings (*nstr*s), with repeated words stripped, and their corresponding concept codes (*cui*s) from the MRXNS_ENG table. As shown in Table 2.1, this table, called INDEXED_NSTR, contains a row for each unique *nstr*, mapped to the list of *cui*s with that particular normalized string representation. The two additional columns specify the number of words in the normalized string and a unique row identifier that is used to reference the row. The rows are sorted according to the number of words in each phrase, such that every row contains at least as many words as all of the rows that come before it. The one-to-many mapping in each row from *row_id* to *cuis* exists for simplicity, allowing a comma-separated list of all *cui*s in a specific row to be retrieved at once. If desired, the table could also have been implemented using a one-to-one mapping from *row_id* to *cui*, as in a traditional relational database.

A second table, INVERTED_NSTR, was then created by splitting each *nstr* from the INDEXED_NSTR table into its constituent words and mapping each unique word to all of the *row_id*s in which it appears. An example of the data contained in INVERTED_NSTR is shown in Table 2.2. Rather than storing this table in memory (as IndexFinder does), it is kept in a database on disk to avoid the time and space needed to load a large table into memory. These two new tables allow relatively efficient retrieval of potential concepts, given a normalized input phrase. For an

Table 2.2: The structure of the INVERTED_NSTR table, which contains all of the unique words extracted from INDEXED_NSTR, mapped to the list of the rows in which each word appears.

word	row_ids
leg	132,631,632,789
low	224,631,789
swell	301,632,789

input phrase of n words, n table lookups to INVERTED_NSTR are needed to find all of the different rows in which each word occurs; consequently, for each row, it is known which of the words from that row occur in the input phrase. Then, because the rows in INDEXED_NSTR are ordered by the number of words in the *nstr*, a single lookup can determine whether all of the words from the *nstr* of a given row were found in the input phrase. See Section 2.3 for further details about how these data structures are used in the coding algorithm.

2.2.4 Spell Checker

Clinicians tend to make spelling errors sometimes, due to being rushed or not knowing the spelling of a complex medical term. An open source spell checker (Jazzy) [40] is therefore incorporated into the coding process. The dictionary word list consists of the collection of word lists that is packaged with Jazzy, augmented with the words from the INVERTED_NSTR table described above. The UMLS-derived table contains some medical terms that are not in the Jazzy dictionary. Additionally, the nursing abbreviation and custom abbreviation lists mentioned above are included in the dictionary list so that they are not mistaken for misspelled words. Every time a new custom abbreviation is added, the new abbreviation is added to the dictionary list.

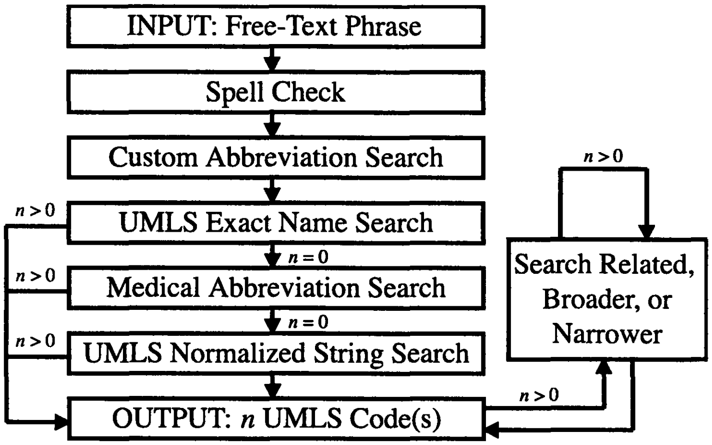

INPUT: Free-Text Phrase
Spell Check
Custom Abbreviation Search
UMLS Exact Name Search
Medical Abbreviation Search
UMLS Normalized String Search
OUTPUT: n UMLS Code(s)
Search Related, Broader, or Narrower
n>0
n>0
n>0
n>0
n>0
n=0
n=0

Figure 2-1: A flow chart of the search process, where n is the number of UMLS codes found by the algorithm at each step.

2.3 Search Procedure

The search procedure for coding is summarized in the flow diagram in Figure 2-1. The input to the program is a free-text input phrase, and the output is a collection of suggested UMLS codes. At the first step, the spell checker is run through the phrase, and if there are any unrecognized words, the user is prompted to correct them before proceeding with the search.

The next resource that is consulted is the custom abbreviation list. If the list contains a mapping from the input phrase to any pre-selected concepts, then those concepts are added to the preliminary results. Next, the UMLS concept table (MR-CONSO) is searched for a concept name that exactly matches the input phrase. To guarantee that typing a custom abbreviation or exact concept name will always return the expected results, these first two searches are always performed.

If there are any results found, the program returns the UMLS codes as output. From this point on, if the number of preliminary results, n, at each stage is greater than zero, the program immediately outputs the results and terminates. Terminating as soon as possible ensures that the program returns potential codes to the user quickly and does not keep searching unnecessarily for more results.

The next step is to check the common medical abbreviation list to see if the input phrase is an abbreviation that can be expanded. Currently, if the entire phrase is not found in the abbreviation list, and the phrase consists of more than two words, then the program proceeds to the next stage. Otherwise, if the phrase consists of exactly two words, then each word is looked up in the abbreviation list to see if it can be expanded. Each of the combinations of possible expansions is searched for in the custom abbreviation list and MRCONSO table. For example, if the input phrase is *pulm htn*, first the whole phrase is looked up in the medical abbreviation list. If there are no expansions for *pulm htn*, then *pulm* and *htn* are looked up separately. Say *pulm* expands to both *pulmonary* and *pulmonic*, and *htn* expands to *hypertension.* Then the phrases *pulmonary hypertension* and *pulmonic hypertension* are both searched for in the custom abbreviations and UMLS concept table.

The attempt to break up the phrase, expand each part, and re-combine them is limited to cases in which there are only two words, because the time complexity of the search can become very high if there are several abbreviations in the phrase and each of the abbreviations has several possible expansions. For example, consider a phrase $x\ y\ z$, where each of the words is an abbreviation. Say x has 3 possible expansions, y has 5 possible expansions, and z has 3 possible expansions. Then there are 3*5*3 = 45 possible combinations of phrases between them.

An alternate method of performing this step is to expand and code each word separately, instead of trying to combine the words into one concept. This method would work correctly, for example, if the input phrase was *mi and chf.* Expanding *mi* would produce *myocardial infarction* and expanding *chf* would produce *congestive heart failure.* Coding each term separately would then correctly produce the two different concepts, and this method would only require a number of searches linear in the total number of expansions. However, this method would not work as desired for phrases such as *pulm htn*, because coding *pulm* and *htn* separately would produce two different concepts (*lung structure* and *hypertensive disease*), whereas the desired result is a single concept (*pulmonary hypertension*). In an interactive coding method, users have the flexibility to do multiple searches (e.g., one for *mi* and one for *chf*), if the combination (*mi and chf*) cannot be coded. Thus, using the "combination" method of abbreviation expansion was found to be more favorable.

If there are still no concept candidates found after the medical abbreviation searches, the algorithm then normalizes the input phrase and tries to map as much of the phrase as possible into UMLS codes. Below are the steps performed during this stage:

1. Normalize the input phrase using the Norm program, to produce normalized phrase $nPhrase$.

2. For each word *word* in $nPhrase$, find all rows *row_id* in INVERTED_NSTR in which *word* occurs. Create a mapping from each *row_id* to a list of the words from that row that match a word from $nPhrase$.

3. Set $unmatchedWords$ equal to all of the words from $nPhrase$. Sort the rows by the number of matches m found in each row.

4. For each m, starting with the greatest, find all of the rows *row_id* that have m matches. Keep as candidates the rows that have exactly m words and contain at least one word from $unmatchedWords$. Also keep as candidates the rows that have excess (i.e., more than m) words but contain a word from $unmatchedWords$ that no other rows with fewer words have. Store the candidate rows in the same order in which they were found, so that rows with more matched words appear first in the results. Remove all of the words from $unmatchedWords$ that were found in the candidate rows. Until $unmatchedWords$ is empty, repeat this step using the next largest m.

5. For each candidate row, get all concepts from that row using the INDEXED_NSTR table.

In step 1, Norm may produce multiple (sometimes incorrect) normalized representations of the input string (e.g., *left ventricle* is normalized to two different forms, *left ventricle* and *leaf ventricle*). In these cases, only the first normalized representation is used, in order to keep the number of required lookups to a minimum. Furthermore,

Table 2.3: The portion of INVERTED_NSTR that is used in the normalized string search for the phrase *thick white sputum.*

word	row_ids
sputum	834,1130,1174,1441,...
thick	834,1130,1174,...
white	1441,...

Table 2.4: The inverse mapping created from each row to the words from the row that occur in the input string. *matched_numwords* is the number of words from the input that were found in a particular row, and *row_numwords* is the total number of words that exist in the row, as found in INDEXED_NSTR.

row_id	matched_words	matched_numwords	row_numwords
834	sputum, thick	2	2
1130	sputum, thick	2	3
1174	sputum, thick	2	3
1441	sputum, white	2	4

the UMLS normalized string table (MRXNS_ENG), which was created using Norm, often contains separate entries for the different representations that Norm gives (e.g., the concept for *left ventricle* is linked to both normalized forms *left ventricle* and *leave ventricle*), so even if only the first normalized form is used, the correct concept can usually be found.

Step 2 finds, for each word in *nPhrase*, all of the rows from INVERTED_NSTR that the word appears in, and creates an inverted mapping from each of these rows to the words that appeared in that row. In this way, the number of words from *nPhrase* that were found in each row can be counted. Consider, for example, the phrase *thick white sputum.* After Norm converts the phrase into *sputum thick white*, each of the three words is looked up in INVERTED_NSTR to find the *row_ids* in which they exist (see Table 2.3). In Table 2.4, an inverted mapping has been created from each of the *row_ids* to the words from Table 2.3.

In Step 3, the rows are sorted according to the number of matched words, so that when going down the list, the rows with more matched words will be returned

Table 2.5: An example of the final row candidates left over after filtering. The *cuis* corresponding to each of these rows are returned as the output of the normalization stage.

row_id	cuis	nstr	numwords
834	C1	sputum thick	2
1441	C2	appearance foamy sputum white	4

first. Some rows in this list might contain extra words that are not in *nPhrase*, and some rows might contain only a subset of words in *nPhrase*. In the above example, the greatest number of words that any row has in common with the phrase *thick white sputum* is two (rows 834, 1130, and 1174 have *sputum* and *thick*, while row 1441 has *sputum* and *white*). The total number of words in row 834 (found in INDEXED_NSTR) is exactly two, whereas the other three rows have extraneous (i.e., more than two) words.

Step 4 prioritizes the rows and filters out unwanted rows. Each "round" consists of examining all of the rows that have m matching words and then deciding which rows to keep as candidates. The *unmatchedWords* list keeps track of which words from *nPhrase* have not been found before the current round, and initially contains all of the words in *nPhrase*. For each number of matched words m, the rows that contain no extraneous words are added to the candidate list first, followed by rows that have extraneous words but also have words that none of the rows with fewer extraneous words have. Ordering the candidate rows this way ensures that as many words from *nPhrase* are covered as possible, with as few extraneous words as possible. Once *unmatchedWords* is empty or there are no more rows to examine, Step 4 ends and the concepts from the candidate rows are returned as the output of the coding algorithm's normalization stage. Only one round (m=2) needs to be performed for *thick white sputum*, because all words in the phrase can be found in this round. Row 834 is kept as a candidate because it covers the words *thick* and *sputum* without having any extraneous words, but rows 1130 and 1174 are thrown out because they contain extraneous words and do not have any new words to add. Row 1441 also contains extra words, but it is kept as a candidate because it contains a word (*white*) that none of the other rows have thus far. Table 2.5 shows the two rows that are left at the end of this step. The results of the normalization stage are the two concepts, C1 and C2, found in the candidate rows.

At any of the stages of the coding algorithm where potential concepts are returned, the user has the option of searching for *related*, *broader*, or *narrower* terms. A concept $C1$ has a *broader* relationship to a concept $C2$ if $C1$ is related to $C2$ through a *parent* (*PAR*) or *broader* (*RB*) relationship, as defined in the UMLS MRREL table. Similarly, a *narrower* relationship between $C1$ and $C2$ is equivalent to the *child* (*CHD*) and *narrower* (*RN*) relationships in MRREL. These relationships allow the user to explore the UMLS hierarchy and thus are helpful for finding concepts with greater or less specificity than those presented.

2.4 Configuration Options

The free-text coding tool can be run with various configurations. For example, the name of the UMLS database and abbreviation and dictionary lists are all configurable. Below is a summary of further options that can be specified for different aspects of the coding process.

2.4.1 Spell Checking

Spell checking can either be set to *interactive* or *automatic*. The *interactive* mode is the default used in the graphical version of the software, and can also be used in the command-line version. When the mode is set to *automatic*, the user is not prompted to correct any spelling mistakes. Instead, if a word is unrecognized and there are spelling suggestions, then the word is automatically changed to the first spelling suggestion before proceeding.

2.4.2 Concept Detail

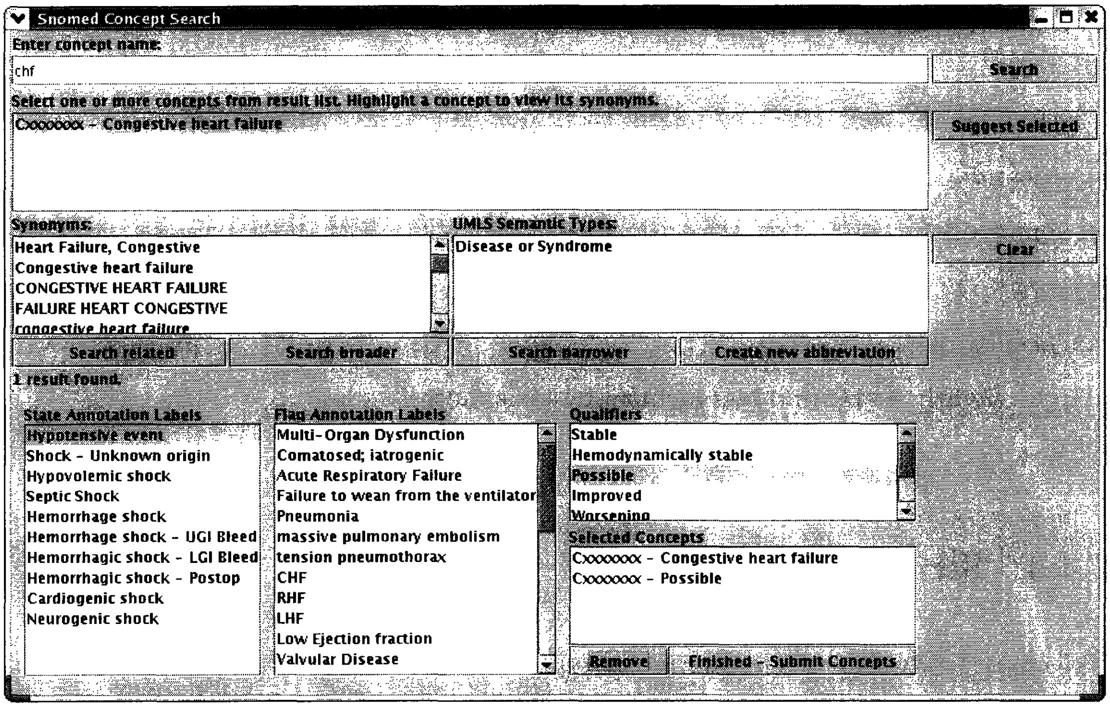
The amount of detail to retrieve about each concept can be configured as either *regular* (the default) or *light*. The *regular* mode retrieves the concept's unique identifier (*cui*), all synonyms (*strs*), and all semantic types (*stys*). The *light* mode only retrieves the concept's *cui* and preferred form of the *str*. If the semantic types and synonyms are not needed, it is recommended that *light* mode be used, because database retrievals may be slightly faster and less memory is consumed.

2.4.3 Strictness

The concept searches may either be *strict* or *relaxed*. When this value is set to *strict*, then only the concepts that match every word in the input phrase are returned. This mode is useful when it needs to be known exactly which words were coded into which concepts. For example, in this mode, no codes would be found for the phrase *thick white sputum*, because no UMLS concept contains all three words. In *relaxed* mode, partial matches of the input phrase may be returned, so a search for *thick white sputum* would find concepts containing the phrases *thick sputum* and *white sputum*, even though none of them completely covers the original input phrase.

2.4.4 Cache

To improve the efficiency of the program, a cache of searched terms and results may be kept, so that if the same phrase is searched for multiple times while the program is running, only the first search will access the UMLS database (which is usually the bottleneck). When the cache is full, a random entry is chosen to be kicked out of the cache so that a new entry can be inserted. The current implementation sets the maximum number of cache entries to be a fixed value. The user has an option of not using the cache (e.g., if memory resources are limited).



Enter concept name:
chf

Select one or more concepts from result list. Highlight a concept to view its synonyms.
Cxxxxxxx - Congestive heart failure

Synonyms:
Heart Failure, Congestive
Congestive heart failure
CONGESTIVE HEART FAILURE
FAILURE HEART CONGESTIVE
congestive heart failure

UMLS Semantic Types:
Disease or Syndrome

1 result found.

State Annotation Labels
Hypotensive event
Shock - Unknown origin
Hypovolemic shock
Septic Shock
Hemorrhage shock
Hemorrhage shock - UGI Bleed
Hemorrhagic shock - LGI Bleed
Hemorrhagic shock - Postop
Cardiogenic shock
Neurogenic shock

Flag Annotation Labels
Multi-Organ Dysfunction
Comatosed; iatrogenic
Acute Respiratory Failure
Failure to wean from the ventilator
Pneumonia
massive pulmonary embolism
tension pneumothorax
CHF
RHF
LHF
Low Ejection fraction
Valvular Disease

Qualifiers
Stable
Hemodynamically stable
Possible
Improved
Worsening

Selected Concepts
Cxxxxxxx - Congestive heart failure
Cxxxxxxx - Possible

Figure 2-2: A screenshot of the UMLS coding application that has been integrated into the Annotation Station.

2.5 User Interface

A graphical user interface for the coding program was developed and integrated into the Annotation Station for expert annotators to use. The process of labelling an annotation typically consists of the following steps:

1. The expert identifies a significant clinical event or finding (e.g., a blood pressure drop in the patient).

2. The expert supplies a free text descriptor for the event (e.g., *hemorrhagic shock*).

3. The expert invokes the free-text coding application, which performs a search and returns a list of possible UMLS codes.

4. From the list of results, the expert chooses one or more concepts that aptly describe the phrase (e.g., *C1 - Shock, Hemorrhagic*).

Figure 2-2 shows a screenshot of the interface. The input phrase is entered in the field at the top, labelled *Enter concept name.* If the *interactive* spelling mode is used, a dialog will prompt the user to correct any unrecognized words. After the search procedure is done, the list of concept candidates appears in the results list below the input field. The *Synonyms* field is populated with all of the distinct *strs* (from the UMLS MRCONSO table) for the currently highlighted concept. Similarly, the *Semantic Types* field is populated with all of the concept's different *stys* from the UMLS Semantic Type (MRSTY) table.

The *Search related*, *Search broader*, and *Search narrower* buttons search for concepts with the related, broader, or narrower relationships, as described in Section 2.3. The *Create new abbreviation* button opens up a dialog box allowing the user to add a custom abbreviation that is linked to one or more selected concepts from the candidate list.

Up to this point, the standalone and Annotation Station versions of the interface are essentially the same. The remaining panels below are specifically designed for Annotation Station use. Expert clinicians found that in labelling state and flag annotations [9], there was a small subset of UMLS concepts that were often reused. Rather than recoding them each time, a useful feature would be to have pre-populated lists of annotation labels, each mapped to one or more UMLS concepts, to choose from. Therefore, the *State Annotation Labels*, *Flag Annotation Labels*, and *Qualifiers* panels were added. The state and flag annotation lists each contain a collection of commonly used free-text annotation labels, which are each linked to one or more concepts. The qualifiers are a list of commonly used qualifiers, such as *stable*, *improved*, and *possible*, to augment the annotation labels. Upon selecting any of the annotation labels or qualifiers, the concepts to which they are mapped are added to the *Selected Concepts* box. In addition, the annotator can use the coding function to search for additional free-text phrases that are not included in the pre-populated lists. An annotation label can be coded with multiple concepts because often there is no single UMLS concept that completely conveys the meaning of the label. To request a new concept to be added to the static lists, the user can highlight concepts from the search results and press the *Suggest* button. After all of the desired concepts are added to the *Selected Concepts* list, the *Finished* button is pressed and the concept codes are added to the annotation.

2.6 Algorithm Testing and Results

To evaluate the speed and accuracy of the coding algorithm, an unsupervised, non-interactive batch test of the program was run, using as input almost 1000 distinct medical phrases that were manually extracted by research clinicians from a random selection of almost 300 different BIDMC nursing notes. Specifically, the focus was narrowed to three types of clincal information (*medications*, *diseases*, and *symptoms*) to realistically simulate a subset of phrases that would be coded in an annotation situation.

2.6.1 Testing Method

The batch test was run in *light* (retrieving only concept identifiers and names) and *relaxed* (allowing concept candidates that partially cover the input phrase) mode, using *automatic* spelling correction. No cache was used, since all of the phrases searched were distinct. The custom abbreviation list was also empty, to avoid unfairly biased results. The 2004AA UMLS database was stored in MySQL (MyISAM) tables on an 800MHz Pentium III with 512MB RAM, and the coding application was run locally from that machine. Comparisons were made between searching on the entire UMLS and using only the SNOMED-CT subset of the database.

The test coded each of the phrases and recorded the concept candidates, along with the time that it took to perform each of the steps in the search (shown in Figure 2-1). If there were multiple concept candidates, all would be saved for later analysis. To judge the accuracy of the coding, several research clinicians manually reviewed the results of the batch run, and for each phrase, indicated whether or not the desired concept code(s) appeared in the candidate list.

In addition, as a baseline comparison, the performance of the coding algorithm was compared to that of a default installation of NLM's MMTx [3] tool, which uses the entire UMLS. A program was written that invoked the MMTx *processTerm* method

Table 2.6: A summary of the results of a non-interactive batch run of the coding algorithm. For each of the three tests (SNOMED-CT, UMLS, and MMTx), the percentage of the phrases that were coded correctly and the average time it took to code each phrase are shown, with a breakdown by semantic type.

		Diseases	Medications	Symptoms
SNOMED-CT	% Correct	80.1%	50.7%	77.5%
	Time	149.3ms	151.6ms	203.9ms
Entire UMLS	% Correct	85.6%	83.3%	86.4%
	Time	169.7ms	107.1ms	227.0ms
MMTx with UMLS	% Correct	71.9%	66.9%	80.2%
	Time	1192.0ms	614.8ms	893.9ms

on each of the medical phrases and recorded all of the concept candidates returned, as well as the total time it took to perform each search.

2.6.2 Results

Out of the 988 distinct phrases extraced from the nursing notes, 285 were diseases, 278 were medications, and 504 were symptoms. There were 77 phrases that were categorized into more than one of the semantic groups by different people, possibly depending on the context in which the phrase appeared in the nursing notes. For example, *bleeding* and *anxiety* were both considered diseases as well as symptoms. The phrases that fell into multiple semantic categories were coded multiple times, once for each category. The phrases were generally short; disease names were on average 1.8 words (11.3 characters) in length, medications were 1.3 words (9 characters), and symptoms were 2.2 words (13.8 characters).

The results for the three types of searches (using SNOMED-CT, using the entire UMLS, and using MMTx with the entire UMLS) are summarized in Table 2.6. Each of the percentages represents the fraction of phrases for which the concept candidate list contained concepts that captured the full meaning of the input phrase to the best of the reviewer's knowledge. If only a part of the phrase was covered (e.g., if a search on *heart disease* only returned *heart* or only returned *disease*), then the result was usually marked incorrect.

Using the SNOMED-CT subset of the UMLS, only about half of the medications were found, and around 80% of the diseases and symptoms were found. Of the diseases, medications, and symptoms, 4.2%, 33.7%, and 4.2% of the searches returned no concept candidates, respectively. Expanding the search space to the entire UMLS increased the coding success rate to around 85% for each of the three semantic categories. Only 2.8%, 4%, and 1.6% of the disease, medication, and symptom searches returned no results using the entire UMLS. For both versions of the algorithm, the average time that it took to code each phrase was approximately 150 milliseconds for diseases and a little over 200 milliseconds for symptoms. Using the entire UMLS generally took slightly longer than using only SNOMED-CT, except in the case of medications, where the UMLS search took about 100 milliseconds and SNOMED-CT search took approximately 150 milliseconds per phrase. In comparison, MMTx took over one second on average to code each disease, over 600 milliseconds for each medication, and almost 900 milliseconds for each symptom. The percentage accuracy for medications and symptoms was slightly better than that of SNOMED-CT, but in all cases the UMLS version of the coding algorithm performed better than MMTx. For the disease, medication, and symptom semantic categories, the MMTx search found no concept candidates 12.6%, 27%, and 9.2% of the time, respectively.

A distribution of the search times between the various stages of the automatic coding algorithm, for both SNOMED-CT and UMLS, is shown in Figure 2-3. Timing results were recorded for the spell checking, exact name search, medical abbreviation search, and normalized string search stages of the coding process. Because the custom abbreviation list was not used, this stage was not timed. For each stage, the number of phrases that reached that stage is shown in parentheses, and the average times were taken over that number of phrases. For example, in the medications category, 205 of the exact phrase names were not found in SNOMED-CT, and the algorithm proceeded to the medical abbreviation lookup. In contrast, using the entire UMLS, only 110 of the medication names had not been found after the exact name lookup.

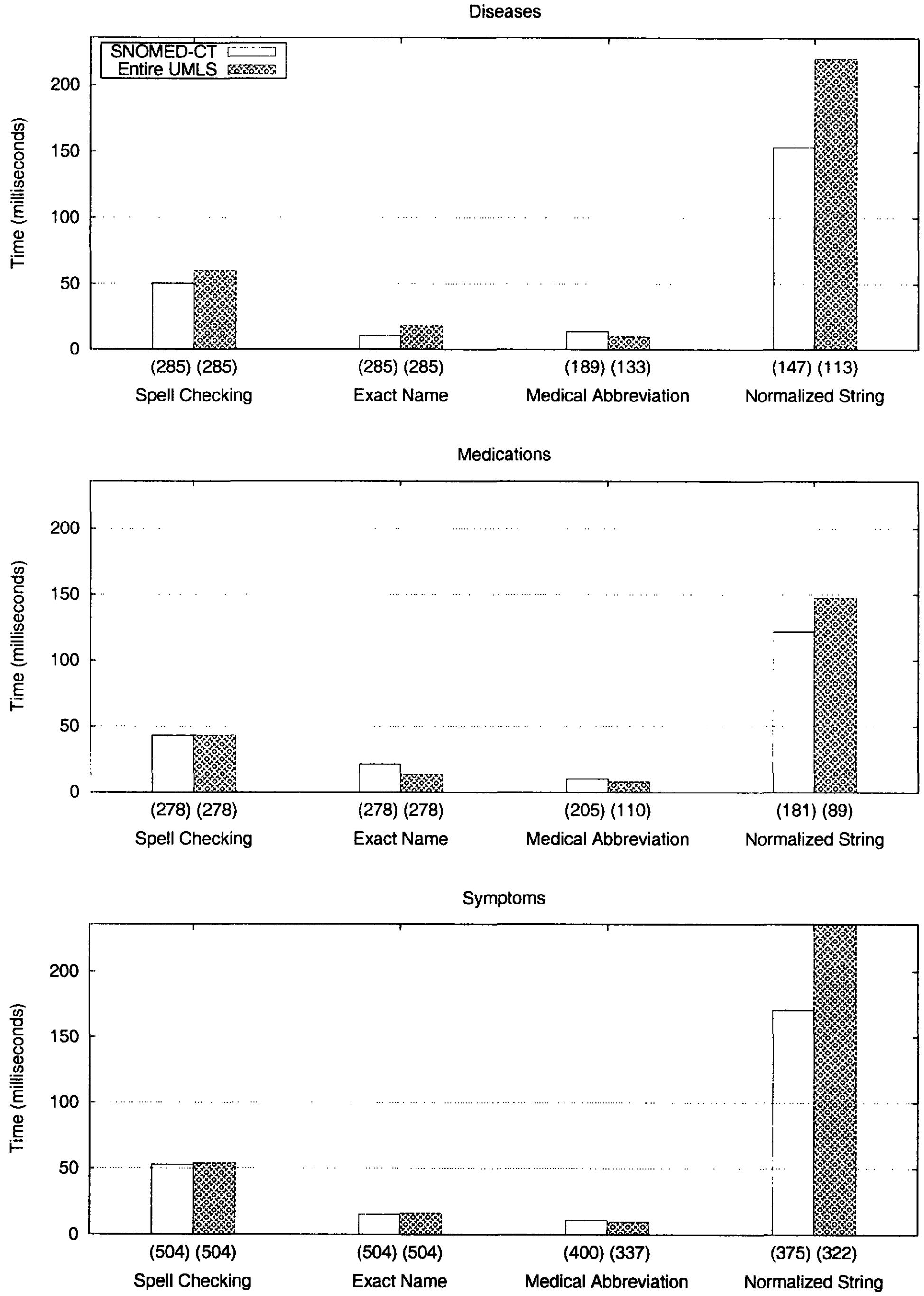


Figure 2-3: The average time, in milliseconds, that the coding algorithm spent in each of the main stages of the coding process. The custom abbreviation search is omitted because it was not used in the tests. Comparisons are made between searching on the entire UMLS and on only the SNOMED-CT subset. In parentheses for each stage are the number of phrases in the test set, out of 670 total, that made it to that stage of the process.

In all cases, the largest bottleneck was the normalized string search, which took approximately 150-250 milliseconds to perform. Because only about 50-65% of the phrases reached the normalized search stage, however, the average total search times shown in Table 2.6 were below the average normalized search times. Of the time spent in the normalized search stage, 50-70 milliseconds were spent invoking the Norm tool to normalize the phrase. The second most time-consuming stage was the spell checking stage. For the diseases, 66 spelling errors were found and 45 of those were automatically corrected; for medications, 69 of 112 mistakes were corrected; for symptoms, 92 of 124 mistakes were corrected.

2.6.3 Discussion

The timing and accuracy tests show that on average the coding algorithm is very fast, and is a vast improvement over MMTx when using the same search space. The concept coverage of SNOMED-CT was noticeably narrower than that of the entire UMLS, especially for medications. Currently, annotators have been labelling medications with their generic drug names if the brand names cannot be found in SNOMED-CT, but it might be useful to add a vocabulary of drug brand names, such as RxNorm [4], to make coding medications in SNOMED-CT faster. If annotation labels are to be limited to SNOMED-CT concepts, another possibility is for the coding algorithm to search the entire UMLS, and from the results, use the UMLS relationship links to search for related concepts, until the most closely related SNOMED-CT concept is found.

Although not all phrases in the batch test were successfully coded, the test was intended to evaluate how many of the phrases could be coded non-interactively and on the first try. In the interactive version of the coding algorithm, the user would be able to perform subsequent searches or view related concepts to further increase the chance of finding the desired codes. Furthermore, the test only used distinct phrases, whereas in a practical setting (e.g., during annotation or extraction of phrases from free-text notes) it is likely that the same phrase will be coded multiple times. The addition of both the custom abbreviation list and the cache would make all searches on repeated phrases much faster, and also increase the overall rate of successful coding.

One noticeable problem in the non-interactive algorithm was that the spell checker would sometimes incorrectly change the spelling of words that it did not recognize, such as *dobuta* (shorthand for the medication *dobutamine*), which it changed to *doubt* and subsequently coded into irrelevant concepts. This problem would be resolved in the interactive version, because the user has the option of keeping the original spelling of the word and adding it to the spelling dictionary or adding it as an abbreviation. A solution to the problem in the non-interactive version might be to only change the spelling if there is exactly one spelling suggestion (increasing the likelihood that the spelling suggestion is correct), but without human intervention there is still no way of knowing for certain if the spelling is correct. Furthermore, if the original word was not found in the dictionary lists, it is unlikely that it would be coded successfully anyway, because the dictionary list includes all known abbreviations and normalized strings. There are other open source spell checkers that might have been used instead, such as the NLM's GSpell [1], which is intended to be useful in medical applications. However, Jazzy was chosen because it is much faster than GSpell and does not require a large amount of disk space for installation.

Another problem that occurred was in the normalization phase of the program. Norm often turns words into forms that have completely different meanings than the original word. For example, it turns *bs's coarse* (meaning *breath sounds coarse*) into both *b coarse* and *bs coarse*; in this case, the second normalization is correct, but because the coding algorithm only uses the first form, it does not find the correct one. A possible fix would be for the algorithm to consider all possible normalized forms; although the performance would decrease, the coverage of the algorithm might improve.

Many of the diseases and symptoms that were incorrectly coded were actually observations or measurements that implied a problem or symptom. For example, number ranges such as (*58-56, 60-62*) were taken to mean *low blood pressure*, *101.5* meant *high temperature*, *bl sugars>200* meant *hyperglycemia*, and *creat 2.4 rise from baseline 1.4* meant *renal insufficiency.* The coding algorithm currently does not have the capacity to infer meaning from such observations, but it appears that annotators and other clinicians find such interpretations useful.

Another problem that the algorithm had was that, despite using a medical abbreviation list, it still did not recognize certain abbreviations or symbols used by the nurses, such as *ˆchol*, meaning *high cholesterol*. The algorithm also had trouble at times finding the correct meaning for an ambiguous abbrevation. The abbreviation *arf* expands into *acute renal failure*, *acute respiratory failure*, and *acute rheumatic fever*. In the SNOMED-CT subset of the UMLS, the MRCONSO table does not have a string matching *acute renal failure*, but it does have strings matching the other two phrases. Therefore, the other two phrases were coded first, and the program terminated before *acute renal failure* (in this case, the desired concept) could be found. The mistakes also included some anomalies, such as *k* being coded into the keyboard letter "k" instead of *potassium*, *dm2* being coded into a qualifier value *dm2* instead of *diabetes type II*, and the medication abbreviation *levo* being coded into the qualifier value, *left*. In these cases, a method to retain only the more relevant results might have been to filter the results by semantic category, keeping only the concepts that belong to the *disease*, *medication*, or *symptom* categories. For example, after searching for an exact concept name for *levo*, if the only result had been the qualifier value *left*, the search would continue on to the medical abbreviation list lookup. Assuming that *levo* was on the abbreviation list, then the concept code for the medication *levo* would then be found. Filtering might help in cases where the desired semantic category is known in advance, as in the case of the batch testing, where clinicians had manually extracted phrases from these three specific categories. In a completely automated system, however, it is not known which parts of the text might belong to which semantic categories, so it might be better to explore all possibilities rather than filtering.

One important issue that also must be considered is that human annotators often have very different ways of interpreting the encoding of phrases. Among the experts that judged the results of the batch test, some were more lenient than others in deciding the correctness of codes. Sometimes the UMLS standardized terminology was different from what the clinicians were used to seeing, and there was disagreement or confusion as to whether the UMLS concept actually described the phrase in question. Some standardization of the way the human judging is done may make the test results more relevant and help in improving the algorithm in the future.

Despite some of the difficulties and issues that exist, the coding algorithm has been shown to be efficient and accurate enough to be used in a real-time setting; a graphical version of the program is currently being used by clinicians in the Annotation Station. Furthermore, although the algorithm currently performs relatively well without human intervention, there are several possible ways to help improve the relevance of the concept candidates returned. A better spell checking method might be explored, so that words are not mistakenly changed into incorrect words. The addition of UMLS vocabularies, particularly for medications, may help in returning more relevant results more quickly, given a larger search space. Finally, a way to infer meaning from numerical measurements may prove to be a useful future extension of the algorithm.

Chapter 3

Development of a Training Corpus

In order to develop an algorithm that efficiently and reliably extracts clinical concepts from nursing admission and progress notes, a "gold standard" corpus is needed for training and testing the algorithm. There currently are no known clinical corpora available that are similar in structure to the BIDMC nursing notes and that have the significant clinical phrases extracted. This chapter describes the development of a corpus of nursing notes with all of the diseases, medications, and symptoms tagged. Creating the corpus involved an initial, automatic "brute force" tagging, followed by manual review and correction by experts.

3.1 Description of Nursing Notes

To comply with federal patient privacy regulations [35, 34], the nursing notes used in this project consist of a subset of *re-identified* notes selected from the MIMIC II database. As detailed in [20], a corpus of over 2,500 notes was manually de-identified by several clinicians and then dates were shifted and protected health information manually replaced with surrogate information. A small subset of the re-identified notes was used to form a training corpus for automatic clinical information extraction.

The nursing notes are a very valuable resource in tracking the course of a patient, because they provide a record of how the patient's health was assessed, and in turn how the given treatments affected the patient. However, because there exist many notes and they are largely unstructured, it is difficult for annotators and automated programs to be able to quickly extract relevant information from them. The nurses generally use short phrases that are densely filled with information, rather than complete and grammatical sentences. The nurses are prone to making spelling mistakes, and use many abbreviations, both for clinical terms and common words. Sometimes the abbreviations are hospital-specific (e.g., an abbreviation referring to a specific building name). Often, the meaning of an abbreviation depends on the context of the note and is ambiguous if viewed alone. Appendix A shows a number of sample nursing notes from the BIDMC ICUs.

3.2 Defining a Semantic Tagset

Because the nursing notes are so densely filled with information, almost everything in the notes is important when analyzing a patient's course. However, it is useful to categorize some of the important clinical concepts and highlight or extract them from the notes automatically. For example, when reviewing the nursing notes, annotators typically look for problem lists (diseases), symptoms, procedures or surgeries, and medications. It would be useful if some of this information were automatically highlighted for them. Moreover, developing such an extraction algorithm would further the goals of an intelligent patient monitoring system that could extract certain types of information and automatically make inferences from collected patient data. This research focuses on extracting three types of information in the notes - diseases, medications, and symptoms. It is imagined that the algorithms developed can be easily expanded to include other semantic types as well.

The 2004AA version of the UMLS contains 135 different semantic types (e.g., *Disease or Syndrome*, *Pharmacologic Substance*, *Therapeutic or Preventive Procedure*, etc.); each UMLS concept is categorized into one or more of these semantic groups. These semantic types are too fine-grained for the purposes of an automated extraction algorithm; researchers or clinicians may not need to differentiate between so many different categories. Efforts have been made within the NLM to aggregate the UMLS

Table 3.1: The mappings between semantic types and UMLS *stys* for diseases, medications, and symptoms.

Semantic Type	UMLS Semantic Types (*stys*)
DISEASE	Disease or Syndrome, Fungus, Injury or Poisoning, Anatomical Abnormality, Congenital Abnormality, Mental or Behavioral Dysfunction, Hazardous or Poisonous Substance, Neoplastic Process, Pathologic Function, Virus
MEDICATION	Antibiotic, Clinical Drug, Organic Chemical, Pharmacologic Substance, Steroid, Neuroreactive Substance or Biogenic Amine
SYMPTOM	Sign or Symptom, Behavior, Acquired Abnormality

semantic groups into less fine-grained categories [6]. These NLM-defined groupings, however, are not ideal for differentiating between the types of information that must be extracted from the nursing notes. For example, they do not differentiate between diseases and symptoms, and the medications are all included in a *Chemicals & Drugs* category that may be too broad. Therefore, a different classification was used instead, as shown in Table 3.1.

3.3 Initial Tagging of Corpus

Creating a gold standard corpus of tagged phrases involves going through all of the notes and marking where the phrases of interest (diseases, medications, and symptoms) occur. It is very time-consuming for humans to manually perform this task. Therefore, an automated algorithm was first run through the corpus of notes, tagging everything that appeared to be a disease, medication, or symptom. The hope was that the automated method would do most of the work, and then the human experts, when reviewing the tagged output, would only need to mark each highlighted phrase as correct or incorrect. For each note, the automated tagging algorithm first tokenizes the note, and then determines the best coding of each sentence. From the concepts that constitute the best coding, the diseases, medications, and symptoms are saved for later analysis by the human experts.

3.3.1 Tokenization

The first step of the automated tagging process was to tokenize each note into separate words and symbols, so that each different token could be understood. The algorithm uses a list of acronyms and abbreviations containing punctuation or numbers that should not be broken up (e.g., `p.m.`, `Dr.`, `r/o`, and `a&ox3`) and a large list of stop words. The stop words include all of the strings from the UMLS SPECIALIST Lexicon's agreement and inflection (LRAGR) table that belong to the following syntactic categories: auxiliaries, complementizers, conjunctions, determiners, modals, prepositions, and pronouns.

Below are the rules that were used for tokenization. For each step, spaces are not inserted if they would split up an acronym or stop word.

1. Add a space between a number and a letter if the number comes first (e.g., `5L`, `7mcg`, `3pm`).

2. Do not add a space between a letter and number if the letter comes first (e.g., `x3`, `o2`, `mgso4`).

3. Do not separate contractions (e.g., `can't`, `I'm`, `aren't`).

4. Add a space between letters and punctuation, unless the punctuation is an apostrophe (e.g., `eval/monitoring.` is changed to `eval / monitoring .`, but `iv's` stays the same).

5. Add a space between punctuation and numbers, unless the punctuation is a period between two numbers (e.g., `1.2`), or a period preceded by whitespace and followed by a number (e.g., `.5`)

6. Add a space between two punctuation marks or symbols (e.g., `...` becomes `. . .`).

For example, the phrase `echo 8/87 showing EF 20-25%` would be tokenized into `echo 8 / 87 showing EF 20 - 25 %`.

Within a word, letters that are followed by numbers are not separated because such words are usually either abbreviations or intended to be a single word, as in the examples above. On the other hand, numbers followed by letters often refer to units and times and can be separated. Words with apostrophes are not tokenized because they would split up known contractions. For words in which apostophes are used to indicate the possessive form or (incorrectly) used to indicate plurality, the lack of separation is acceptable because when coding such words, normalization will remove the `'s` endings. Other punctuation marks and symbols are separated from words and numbers (unless the punctuation is a decimal point within a number) so that they can be treated as tokens separate from the words. After tokenizing a note, most sentences or phrases can be found by looking for punctuation tokens, such as periods (`.`), semicolons (`;`), and commas (`,`), that are set off from other tokens by spaces. Periods that do not have a space both before and after them are either part of acronyms or part of numbers with decimal points.

3.3.2 Best Coverage

For the initial tagging of the corpus, an automated coding and search algorithm was used to find as many of the diseases, medications, and symptoms in the notes as possible. The algorithm converts each sentence in a nursing note into a graph-like structure, where the phrases within a sentence make up the nodes, and each node has a cost associated with it, depending on the semantic type of the phrase. The best coding of the sentence is the sequence of nodes with the lowest total cost that covers the sentence completely.

The clinicians generally regarded the task of manually removing incorrectly tagged phrases as less tedious and time-consuming than manually looking for phrases that were missed by the automatic tagger. Thus, the goal of this automatic algorithm, which in effect was a "brute force" lookup method, was to extract any phrase that had a chance of being a medication, disease, or symptom, with the risk of producing

```
createNodes(sentence):
  for length=1 to min(numWords,maxWords)
    for each subset phrase of sentence consisting of length words
      if phrase is a stop word or
      phrase contains only numbers and symbols
        create new node(cost=4*length+5)
      else if length > 1 and
      phrase begins or ends with stop word or punctuation
        do not create node
      else
        try to code phrase
        if results empty
          create new node(cost=10*length+5)
        else if results contains disease, medication, or symptom
          create new node(cost=2*length+5)
        else
          create new node(cost=6*length+5)
```

Figure 3-1: Pseudo-code showing the creation of weighted nodes from a sentence, where *numWords* is the number of words in the sentence after tokenization, and *maxWords* is a pre-specified maximum phrase length, currently set to 6 words. After all of the nodes in the sentence are created, the best path is found using the graph search algorithm in Figure 3-2.

many false positives.

The phrases that potentially belonged to one of the desired semantic categories were given the lowest cost, thus making it more likely that they would be part of the best path through the sentence. In order to determine the cost of each phrase, the meaning of the phrase had to first be determined. For each note, the algorithm first tokenizes the note using the tokenization algorithm from Section 3.3.1, and then divides the note into sentences (where "sentences" also include phrases) by looking for periods, commas, and semi-colons. Then, for each sentence, each sub-phrase (minus some exceptions) was coded using the coding algorithm from Chapter 2 and the results were used to determine the meaning, and associated cost, of the phrase. Figure 3-1 shows the algorithm used to create these nodes.

Considering the terse language and abundance of abbreviated terms in the notes, nurses seemed unlikely to describe a phrase using more than a few words; accordingly, the maximum length of phrase searched was set to a constant number of words (6) to limit the number of searches performed. For a sentence of *numWords* words, and maximum phrase length *maxWords*, at most n*(n+1)/2 nodes will be created, where n is the lesser of *numWords* and *maxWords*.

For each sentence, each subset of the sentence consisting of between 1 and n consecutive words is considered for node creation. If the phrase contains more than one word, and the first or last word is a stop word, then no node is created for the phrase. This check is done to prevent phrases such as *and coughing* from being coded, because if that phrase were to be coded into the concept for *coughing*, then the phrase *and coughing* would incorrectly be highlighted in the corpus. The gold standard corpus must contain the exact indices of the medical terms that have been coded, so that a word like *and*, which really should not be part of the phrase, is not mistakenly tagged as a symptom in the future, for example. A phrase such as *coughing and wheezing* is still coded because the *and* is in the middle of the phrase, rather than being extraneous.

If the phrase itself is a stop word, then it is not coded. Otherwise, the phrase is run through the free-text coding algorithm, and a node is created based on the results of the search. The coding algorithm uses the entire UMLS, rather than only the SNOMED-CT subset, in order to increase the chances of finding a code for each phrase. It also uses a list of custom abbreviations that was created and used by experts on the Annotation Station. The configuration options for the coding algorithm include automatic spell checking (because the whole process is automated), and strict searches, which require all words in the phrase to be found in a single concept. If there are multiple concept candidates, the list is traversed in order until the first medication, disease, or symptom is found. The phrase is assigned the code and semantic type that was found.

The costs are designed to favor phrases that can be coded into medications, diseases, or symptoms. The exact costs given to each node were a bit arbitrary, but overall they conveyed the relative priorities that the different phrase types had in the coding algorithm. Phrases that could be coded into diseases, medications, and symptoms were given the lowest cost because they were the most important. Floating stop

```
 1 getBestPath(startNode):
 2   push startNode onto stack
 3   while stack not empty
 4     node n = node at top of stack
 5     if n has no adjacent nodes
 6       n = settled
 7       pop n from stack
 8     else
 9       for each adjacent node adj
10         if adj not settled
11           push adj onto stack
12           goto 3
13         else if no lowest cost edge from n exists
14           add edge (n,adj)
15         else if cost(n->end) > cost(n)+cost(adj->end)
16           replace lowest cost edge from n with (n,adj)
```

Figure 3-2: Pseudo-code showing the graph search algorithm, where (*a*,*b*) is an edge from node *a* to node *b* and *a*->*b* is a path from *a* to *b* with 0 or more nodes in between. The *lowest cost* edge from a node is the first edge in the lowest-cost path from the node to an end node.

words, symbols, and numbers were also given a low cost so that they would not be attached to the beginning or end of a "meaningful" phrase. Phrases that were coded into UMLS concepts, but were not diseases, medications, or symptoms, were given a slightly higher cost, and phrases that could not be coded were given the highest cost. Furthermore, by adding a constant to the cost of each node created, the algorithm favors paths containing fewer nodes, and thus larger phrases.

As each new node is created, an edge from node *a* to node *b* is created if the last word of *a*'s phrase and the first word of *b*'s phrase are adjacent words in the sentence. Thus, from each possible start node (i.e., those nodes with no incoming edges), each complete path through the sentence is guaranteed to cover all of the words in the sentence. The method used for finding the best path through a sentence is essentially a depth first graph search, where the cost of a path is the sum of the costs of all the nodes on the path, and the path with the lowest total cost is the best path.

The graph search algorithm is shown in Figure 3-2. It takes as input a start node, and produces the ordered list of nodes from the start node to the end of the

Table 3.2: The nodes created for the example sentence *treated for mi and swollen legs*. For each subset of words in the sentence, if the subset is a stop word, it is not coded. Otherwise, the automatic coding algorithm is run. The results determine the cost of the node.

Phrase	Result	# Words	Score
treated for mi and swollen legs	(uncoded)	6	65
treated for mi and swollen	(uncoded)	5	55
mi and swollen legs	(uncoded)	4	45
treated for mi	(uncoded)	3	35
mi and swollen	(uncoded)	3	35
swollen legs	SYMPTOM	2	9
treated	OTHER	1	11
for	(stop word)	1	9
mi	DISEASE	1	7
and	(stop word)	1	9
swollen	OTHER	1	11
legs	OTHER	1	11

sentence, that has the lowest total cost. The edges are unweighted. An end node is reached when a node that does not have any adjacent nodes is encountered. The *getBestPath* method is run on each potential start node, and out of these best paths, the one path with the lowest cost is kept. Out of the coded phrases on that path, those that are marked as medications, diseases, or symptoms are extracted. The final output of the initial tagging stage for each nursing note is the collection of diseases, medications, and symptoms that were found, along with their UMLS codes and their locations (beginning and ending character indices) within the text, so that they can be displayed to the experts later during the manual correction stage.

Table 3.2 shows an example of the nodes created for the sentence *treated for mi and swollen legs*. Each subset of the sentence that does not begin or end with a stop word is made into a node. The words *for* and *and* are stop words, so they are not coded. The remaining phrases are run through the automatic coding algorithm.

After the nodes are created, the graph search algorithm considers every possible combination of nodes that creates a full path through the sentence:

```
[treated for mi and swollen legs],
```

[treated for mi and swollen] [legs],

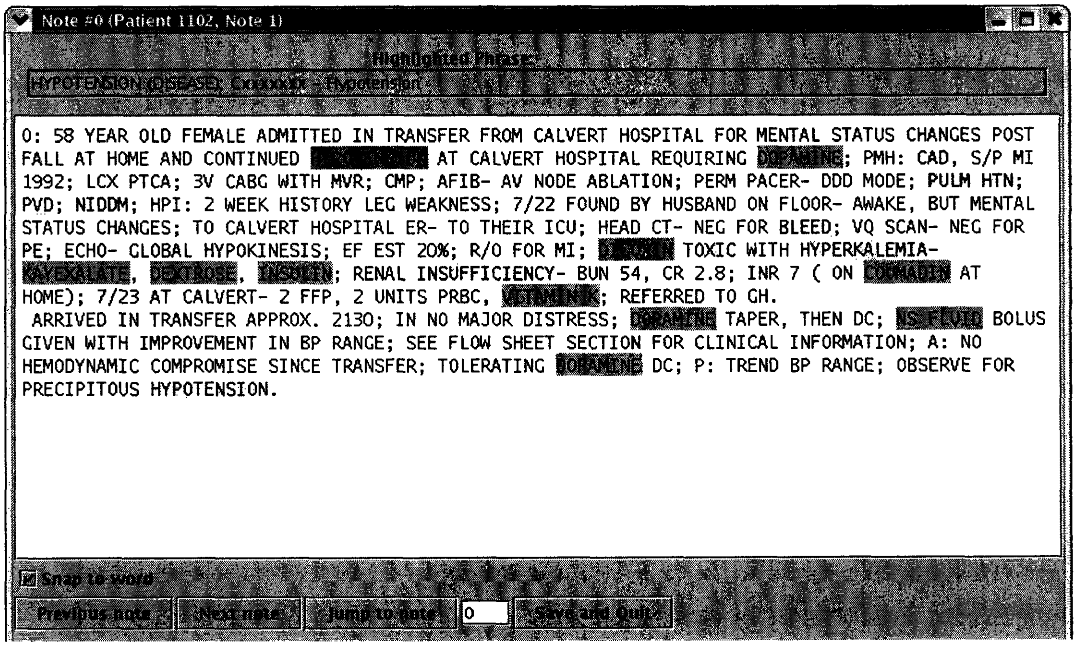
[treated] [for] [mi and swollen legs], etc.

Using the depth-first graph search, the best path through the sentence is found to be:

[treated] [for] [mi] [and] [swollen legs]

with a total score of 45 (11+9+7+9+9). The result is as desired, because the maximal phrases of type *medication*, *disease*, or *symptom* - in this case, the disease *mi* and the symptom *swollen legs* - were extracted.

3.4 Manual Correction of Initial Tagging



HYPOTENSION (DISEASE): Cxxxxxx - Hypotension

0: 58 YEAR OLD FEMALE ADMITTED IN TRANSFER FROM CALVERT HOSPITAL FOR MENTAL STATUS CHANGES POST FALL AT HOME AND CONTINUED HYPOTENSION AT CALVERT HOSPITAL REQUIRING DOPAMINE; PMH: CAD, S/P MI 1992; LCX PTCA; 3V CABG WITH MVR; CMP; AFIB- AV NODE ABLATION; PERM PACER- DDD MODE; PULM HTN; PVD; NIDDM; HPI: 2 WEEK HISTORY LEG WEAKNESS; 7/22 FOUND BY HUSBAND ON FLOOR- AWAKE, BUT MENTAL STATUS CHANGES; TO CALVERT HOSPITAL ER- TO THEIR ICU; HEAD CT- NEG FOR BLEED; VQ SCAN- NEG FOR PE; ECHO- GLOBAL HYPOKINESIS; EF EST 20%; R/O FOR MI; DIGOXIN TOXIC WITH HYPERKALEMIA- KAYEXALATE, DEXTROSE, INSULIN; RENAL INSUFFICIENCY- BUN 54, CR 2.8; INR 7 (ON COUMADIN AT HOME); 7/23 AT CALVERT- 2 FFP, 2 UNITS PRBC, VITAMIN K; REFERRED TO GH.
ARRIVED IN TRANSFER APPROX. 2130; IN NO MAJOR DISTRESS; DOPAMINE TAPER, THEN DC; NS FLUID BOLUS GIVEN WITH IMPROVEMENT IN BP RANGE; SEE FLOW SHEET SECTION FOR CLINICAL INFORMATION; A: NO HEMODYNAMIC COMPROMISE SINCE TRANSFER; TOLERATING DOPAMINE DC; P: TREND BP RANGE; OBSERVE FOR PRECIPITOUS HYPOTENSION.

☑ Snap to word

Figure 3-3: A screenshot of the interface used to manually correct the initial tagging of the nursing notes.

The final step in creating the gold standard corpus was to have humans review the output of the intial, automatic tagging algorithm, and make any necessary corrections. An application was created that loaded each note and highlighted the diseases, medications, and symptoms that were extracted from the automatic tagging algorithm. Figure 3-3 shows a screenshot of the interface of the program. The phrases are color-coded by semantic type, and when selected, the semantic type, concept identifier, and concept name are displayed at the top of the screen.

The expert may change the semantic type if it is incorrect, as well as re-code the phrase if the concept is incorrect. For each phrase, once the semantic type and concept identifier are judged to be correct, the phrase must be verified by pressing a specific key. The explicit verification makes it less likely that the users will skip over a phrase by mistake. If the phrase is not a disease, medication, or symptom, it may be deleted by pressing another key. All instances of a phrase can be deleted from all notes at once by "flushing" the phrase. If new phrases are found that were not extracted by the automatic tagger, the user may also add the new phrase, and choose its semantic type (disease, medication, or symptom) and UMLS code.

At the end of this stage, each of the nursing notes that has been manually verified will have a list of the corrected phrases, semantic types, and concept identifiers that were extracted from it. This corpus of notes forms the gold standard corpus that will be used to train more advanced automatic tagging algorithms.

3.5 Results

The automatic tagging algorithm was run through several hundred different nursing notes, taking an average of approximately one minute to code and extract the phrases from each note. Four clinical experts helped to manually verify the output of the automatic initial tagging. In total, 252 different nursing notes from 10 different patients were manually verified. The corpus consisted of over 50,000 words (after tokenization) and 227,500 characters. Of the gold standard verified phrases, there were 1359 medications, 559 diseases, and 978 symptoms. Overlapping phrases were allowed. For example, the phrase *digoxin toxic* was tagged as a disease, and the first word of the phrase, *digoxin*, was also tagged as a medication.

The precision and recall for the initial automatic tagger, broken down into the

Table 3.3: The total number of phrases and words in the gold standard corpus, and the precision and recall of the initial tagging, for both phrases and words.

	Diseases	Medications	Symptoms
Total Phrases	559	1359	978
Phrase Precision	13.6%	34.5%	47.6%
Phrase Recall	58.0%	77.0%	31.7%
Total Words	706	1475	1705
Word Precision	15.7%	29.7%	55.5%
Word Recall	64.9%	81.7%	29.0%

three semantic types, are presented in Table 3.3. Here, *precision* is defined as:

$$precision = \frac{\#\ phrases\ correctly\ retrieved}{\#\ total\ phrases\ retrieved} \tag{3.1}$$

and *recall* is defined as

$$recall = \frac{\#\ phrases\ correctly\ retrieved}{\#\ phrases\ in\ gold\ standard} \tag{3.2}$$

The *phrase* precision and recall were defined in terms of the number of exact phrases from the gold standard that matched phrases extracted by the automatic algorithm. If the phrases were only partial matches, then it did not count as a match. Because many of the phrases extracted by the automatic tagger overlapped with phrases in the gold standard, but were not exact matches, the *word* precision and recall are also given. These values are defined in terms of the actual number of tokenized words that matched, between the automatic tagger output and the gold standard.

3.6 Discussion and Improvement of Corpus

As expected, the initial tagging produced many false positives. However, it still missed many phrases that the human annotators extracted, for several possible reasons. The automated algorithm was designed to pick out only phrases that could be coded with a single UMLS concept. However, when the experts manually highlighted the concepts and picked out ones that were missed, they often highlighted phrases that encompassed more than one concept, such as a symptom and one or more qualifiers (e.g., *sm amount of thick rusty sputum*). This tendency might suggest that modifiers and other qualifier values should have been extracted along with the main clinical concepts. There might have been abbreviation ambiguities and spelling mistakes that were not resolved by the automatic tagger. It is also very likely that there were semantic type ambiguities. Some concepts have several possible semantic types, according to the UMLS, and the mapping between UMLS semantic types and the more coarse-grained categories of *disease*, *medication*, and *symptom* might not have been an ideal mapping.

Among human annotators, there were many inconsistencies in determining what kind of phrases belonged in which of the semantic categories, or whether they belonged to any category at all. Medications were generally classified as drugs that were administered to the patient, but there was some question about whether or not to include fluids and nutrition. Some annotators did include these, and some were not sure whether or not to. The definition of a symptom was by far the most difficult to agree on. Generally, the experts described them as something that the patient felt. Three experts included signs in the same category as symptoms (the UMLS has one category, *Sign or Symptom*, that includes both) and one did not. One of the three who included signs only did so for test findings that were positive and categorical, rather than numeric. One uncertainty that most people had was whether a chronic symptom counted as a disease or symptom, and whether "normal" symptoms (such as absence of pain) were considered symptoms. When faced with these ambiguities, some people would take the initial, automatic tagging as the default correct answer. Sometimes people would tag a phrase differently each time they saw it. Diseases were relatively easy to define, except for the uncertainty about whether certain phrases counted as a disease or a symptom. For most annotators, if the UMLS identified a phrase as a disease, then it would be verified as a disease. Diseases and symptoms that occurred in negative phrases (e.g., the *mi* in *no mi*) were still generally extracted, because the fact that the patient did not have *mi* is still important. A smart extraction program

Table 3.4: The newly calculated precision and recall of the initial tagging, after making a second pass through the gold standard corpus to remove many of the inconsistencies.

	Diseases	Medications	Symptoms
Total Phrases	499	1322	868
Phrase Precision	12.7%	34.2%	46.5%
Phrase Recall	60.5%	78.4%	34.9%
Total Words	634	1426	1557
Word Precision	14.9%	29.4%	55.2%
Word Recall	68.5%	83.7%	31.6%

would be able to extract *mi* and recognize the *no* as a modifier.

Some specific inconsistencies were seen in a small set of nursing notes that was verified separately by multiple annotators. In one note, one person tagged *alert* and *oriented* as symptoms, whereas the other person considered them as false positives and removed them. Some people highlighted smaller subsets of a phrase than others; for example, one person marked *left leg hurts* as a symptom, while the other only marked *hurts*. One person marked *tylenol #3* as a medication, whereas the other one only marked *tylenol*. The word *confusion* was tagged in different notes alternately as a disease and as a symptom. Some annotators relied more than others on the automatic tagger output, by verifying a phrase as correct if they thought that it was roughly close to being correct.

Because there was so much variation among the tagging techniques exhibited by different people, another manual pass was made through all of the notes in the gold standard corpus, in an attempt to somewhat standardize the tagging. Although overlapping phrases had been allowed during the manual review of the notes, they were deleted in this second pass. The automatic tagging algorithms only assign one tag to each word, and it would have been a better comparison if the gold standard had similar restrictions imposed on it. For most overlapping phrases, the shorter phrase was deleted in favor of the longer one (e.g., *digoxin* was deleted and *digoxin toxic* was kept). Many phrases that sounded like findings rather than symptoms were deleted. It was unclear whether phrases such as *sedated* or *comfortable* should be tagged as symptoms or not, so they were usually kept if already tagged. However, many phrases, such as *follows commands* and *opens eyes to verbal stimuli* were considered too broad or "normal" to be tagged as a symptom. Sometimes for medications, units and modifiers (e.g., *drip*, *mcg*, or *iv*) were tagged as part of the medication, and these phrases were usually kept as they were. For symptoms such as *rash on back of legs*, it was unclear whether the gold standard should contain the entire phrase, or only *rash*. For the most part, the larger phrase was kept in these situations. A newer version of the gold standard corpus was created after going through each of the notes again. Table 3.4 shows the new precision and recall measurements of the initial tagging algorithm, compared to the new gold standard corpus. Because many phrases were deleted, the precision is slightly worse than before, and the recall is slightly higher. The many inconsistencies and uncertainties suggest that there need to be more well-defined standards for what constitutes a disease, medication, or symptom, that the human annotators can adhere to. A more systematic and standardized method of tagging needs to be developed and agreed upon, if a larger, more accurate gold standard corpus is to be made in the future.

Chapter 4

Automatic Extraction of Phrases from Nursing Notes

As shown in Chapter 3, exhaustive search methods are computationally expensive and are not necessarily the most accurate way to extract terms from free text. They also depend on having a comprehensive knowledge base from which to extract information about each word. A better way would be to consider the context of each phrase and to utilize known information about how the meaning of a word is affected by surrounding text. For example, in one of the sentences from a nursing note, there might be an indication of a dosage (e.g., *40mg iv*), followed by an unknown word X. Knowledge about the context in which medications appear in the text would hint that X is the name of a medication. Because such a method would not rely entirely on matching words to a large vocabulary, it would not require as extensive a knowledge base as more exhaustive methods would. This chapter compares various algorithms that were developed to automatically extract a subset of clinical concepts, including medications, diagnoses, and symptoms, from nursing notes, using statistical guessing methods. To evaluate their accuracy, the results of the different methods are compared to the gold standard corpus described in Chapter 3.

4.1 Approaches

Some natural language processing algorithms analyze sentences using a *top-down* approach, by starting at a high level and looking at the overall structure of the sentence, and then breaking it down into its constituent parts. Because the nursing notes are made up of dense, often short phrases that do not have much of a high-level structure, they might be better analyzed using a *bottom-up*, word-by-word approach. In this research, semantic and syntactic tagging methods were used to guess the semantic type of each word in a nursing note, and in turn extract phrases of interest from the note.

Two different methods commonly used to tag free text are rule-based methods and probabilistic methods. The Brill tagger [13] is a well-known transformation-based part of speech tagger that uses a lexicon of words and their possible parts of speech, along with a set of learned lexical and contextual rules that constrain the context in which specific words and tags can appear. The tagger applies these rules to the text until the tagging cannot be further improved. There are also many different probabilistic (or statistical) algorithms that can be used to tag free text. A probabilistic tagger uses observed frequencies of word-tag pairs and common tag sequences from a training corpus to assign the most likely tag to each word.

To extract medical phrases from nursing notes, different implementations of a statistical tagging algorithm were created, including a forward-based algorithm and best-path algorithm. The gold standard training corpus described in Chapter 3 was used to generate probabilities to be used in the tagging algorithm, and from the results of the tagging, a list of the relevant phrases was extracted. Additionally, the rule-based Brill tagger was used to generate parts of speech for each word, to evaluate whether the addition of syntactic data improves the results of the statistical algorithm.

4.2 System Setup

Before the statistical tagging algorithms can be run, a lot of training data must be gathered and stored in a format allowing for easy retrieval. Chapter 3 detailed the development of a gold standard corpus of 252 nursing notes that had the diseases, medications, and symptoms extracted from them. The corpus was randomly divided repeatedly into training sets of 232 notes and testing sets of 20 notes, and the training and testing described below was done separately for each different set. Each time, the information contained in the training set was used to create the statistical data for the tagging algorithms.

While the training corpus contains information about the semantic tags that were most frequently assigned to each word, other potentially useful data about each word, such as its syntactic tag (part of speech), is not captured in the gold standard corpus. The sections below detail the collection of syntactic information for each note in the training corpus, and the combination of semantic and syntactic data to form a database of training tables.

4.2.1 Syntactic Data

Semantic tags alone reveal the meaning of a phrase or word, but part-of-speech tags can reveal the function of a specific word within a phrase, and thus help to find the full extent of each phrase that should be extracted from the text. Thus, it would be useful to have gold standard syntactic tags to go with each word in the training corpus, in addition to the semantic tags that have already been extracted. Because it would have been very time consuming for humans to manually verify the syntactic tag for every single word in the gold standard corpus, a more automated method of syntactic tagging was performed using the Brill tagger.

The *kappa* statistic [17] is often used to evaluate how well classification algorithms agree with a gold standard. It is calculated as follows:

$$kappa = \frac{P_a - P_e}{1 - P_e} \tag{4.1}$$

where P_a is the percent agreement,

$$P_a = \frac{\#\ \mathit{correct\ trials}}{\#\ \mathit{total\ trials}} \tag{4.2}$$

The number of total trials is the total number of words tagged. Each word can either be tagged correctly or incorrectly. The sum of the correct and incorrect trials is equal to the total number of trials. P_e is the percent expected agreement by chance,

$$P_e = \sum_{i=1}^{n} \frac{\#\ \mathit{times\ gold\ chose\ tag_i}}{\#\ \mathit{total\ trials}} * \frac{\#\ \mathit{times\ tagger\ chose\ tag_i}}{\#\ \mathit{total\ trials}} \tag{4.3}$$

where n is the total number of different distinct tags that were either found in the gold standard or found by the automatic tagger, and $\#$ *times gold chose* tag_i and $\#$ *times tagger chose* tag_i are the number of times the gold standard and the automatic tagger chose tag_i, respectively.

By default, the Brill tagger comes with a lexicon trained on the Brown Corpus and Wall Street Journal, which do not contain a large amount of medical terminology, such as abbreviations and medication names. However, in a previous project [21], it was shown that simply adding a medical lexicon to the Brill tagger improved the kappa from 0.7363 (with the default lexicon) to 0.8839, for a small corpus of nursing notes where the parts of speech had been manually verified. Because the gold standard corpus created for phrase extraction is about 10 times larger than the one tested with the Brill tagger, it was too tedious and unrealistic to have all of the syntactic tags in the larger corpus manually verified. However, because the Brill tagger was shown to perform relatively well on the smaller corpus, the tagger was used for the larger corpus and its output was assumed to be correct.

The input that the Brill tagger expects includes a lexicon, a lexical rule file, a contexual rule file, and the untagged text. Each line of the lexicon must contain a word followed by its possible parts of speech, separated by spaces. The most likely part of speech for each word must occur first in the list, but the rest of the parts of speech are not ordered. The format of the lexicon is as follows:

```
infarction NN
```

```
diseases NNS
suspended JJ VBN VBD
```

where in this example, NN, NNS, JJ, VBN, and VBD are Penn Treebank [28] tags representing the parts of speech *noun*, *plural noun*, *adjective*, *past participle*, and *past tense verb*, respectively. The contextual rule file contains rules for changing tags when certain words or tags are encountered. For example, if an adjective is followed by a determiner, the adjective should be changed to a past-tense verb. The lexical rule file uses clues about unknown words to guess their tags. The default rule files that come with the Brill tagger were both trained on Wall Street Journal text and contain several hundreds of rules in total. The untagged input text to the Brill tagger must be tokenized, with one sentence per line.

To create the part-of-speech medical lexicon to be used by the Brill tagger, syntactic information was extracted from UMLS SPECIALIST Lexicon [32] tables. The LRAGR (agreement and inflection) table contains a collection of medical concept strings, each mapped to a syntactic category (e.g., *noun* or *verb*) and tense or agreement information (e.g. *third person singular* or *past*). Many, but not all, of the UMLS concepts are represented in this table. Because the Brill tagger was trained using Penn Treebank [36] part of speech tags, and the contextual and lexical rules are specific to these tags, the UMLS syntactic information had to be translated into Penn Treebank tags.

The parts of speech from the LRAGR table do not correspond directly to Penn Treebank tags; for example, LRAGR contains a syntactic category for complementizers, and also contains a larger number of modal, auxiliary, determiner agreement types than the Penn Treebank has. On the other hand, the Penn tagset contains some parts of speech that are not in the UMLS, such as pre-determiners and WH-words. Furthermore, the Penn Treebank tagset makes a distinction between proper nouns and regular nouns, whereas the LRAGR table does not. In these cases, the UMLS table of properties (LRPRP) was used to find out whether a noun was proper. See Appendix B for the complete mappings (including some exceptions) used to convert UMLS syntactic categories into Penn Treebank parts of speech.

Because there was no gold standard text that had the parts of speech all tagged, there was no frequency information for ordering the parts of speech for each word. Therefore, for words in the lexicon with multiple parts of speech, the tags were ordered according to the overall frequencies of tags in the lexicon. For example, nouns are the most frequently occurring part of speech in English text, so for all words that have a noun as a possible part of speech, the noun would be listed first.

To tag each nursing note, the note was first tokenized using the tokenization method described in Chapter 2, and then lines were split where periods, commas, or semicolons occurred. Additionally, because the capitalization varied greatly among different nursing notes (i.e., some were in all uppercase, some were all lowercase, and some were mixed), each note was converted to lowercase so that the Brill tagger would not try to tag uppercase words as proper nouns. The notes were then sent through the Brill tagger one by one, using the medical lexicon and the default lexical and contextual rule files as input. The output produced for each note consisted of the tokenized text with a slash (/) and part of speech after each word, as in the following example:

```
pt/NN admitted/JJ for/IN mental/JJ status/NN changes/NNS
```

4.2.2 Statistical Data

After the notes from the training corpus were all tagged with parts of speech, the (word, syntactic tag, semantic tag) frequencies could be calculated. For each note, the following steps were performed:

1. Tokenize the note into separate words and convert everything to lowercase.

2. For each token, find the corresponding syntactic tag from the Brill tagger output.

3. For each token, if it was part of a phrase tagged in the gold standard corpus, tag the token as MEDICATION, DISEASE, or SYMPTOM accordingly. Otherwise, if the word contains only punctuation, tag it as PUNCT. If it contains only

Table 4.1: The TAGS table contains all of the combinations of word, syntactic tag, and semantic tag that occurred in the training corpus, along with their frequencies.

word	syntactic	semantic	freq
confirmed	VBN	OTHER	1
discomfort	NN	SYMPTOM	8
withdrawn	VBN	OTHER	1
withdrawn	VBN	SYMPTOM	1
...			

Table 4.2: The BIGRAMS table contains all of the different consecutive sequences of semantic tags of length 2 that occur in the training corpus, and their frequencies.

first	second	freq
PUNCT	NUM	1412
DISEASE	NUM	15
SYMPTOM	OTHER	148
...		

numbers and punctuation, tag it as NUM. If it is a stop word, tag it as STOP. If it is none of the above, tag it as OTHER.

The semantic categories PUNCT, NUM, and STOP were added to give more information about words that would otherwise be tagged as OTHER. After each token in the training corpus was tagged semantically and syntactically, the results were recorded in a database table, as shown in Table 4.1. The TAGS table contains an entry for every combination of word, semantic tag, and syntactic tag that was found, along with the total number of times it occurred in the training corpus.

Next, three different n-gram tables were created. Each of them contains all of the distinct sequences of semantic tags of length n that occurred in the training corpus. For each note, all groups of 2, 3, or 4 consecutive words were found, and the corresponding semantic tag for each word (MEDICATION, DISEASE, SYMPTOM, NUM, PUNCT, STOP, or OTHER) was used to create the semantic tag sequence, which was added to the BIGRAMS, TRIGRAMS, or TETRAGRAMS tables (Tables 4.2-4.4) accordingly. The total number of times each sequence occurred in the

Table 4.3: The TRIGRAMS table contains all of the different consecutive sequences of semantic tags of length 3 that occur in the training corpus, and their frequencies.

first	second	third	freq
PUNCT	OTHER	MEDICATION	127
MEDICATION	OTHER	STOP	161
SYMPTOM	SYMPTOM	PUNCT	251
...			

Table 4.4: The TETRAGRAMS table contains all of the different consecutive sequences of semantic tags of length 4 that occur in the training corpus, and their frequencies.

first	second	third	fourth	freq
OTHER	STOP	MEDICATION	PUNCT	102
SYMPTOM	SYMPTOM	PUNCT	OTHER	137
SYMPTOM	SYMPTOM	SYMPTOM	SYMPTOM	162
...				

corpus was recorded as well. In the n-gram tables, the first $n - 1$ entries are allowed to be blank. If the table contains many n-grams of the form "<blank> TAG," for example, then it indicates that the semantic type TAG is frequently the first word of a text note. The TAGS and n-gram tables are used in the automatic extraction algorithms to find the most likely semantic tag for each word, given its part of speech and the semantic types of preceding words.

4.2.3 Semantic Lexicon

During the automatic tagging of text, if a word is encountered that was not in the training corpus, there should be another way of finding out what the most probable semantic categories for the word are. For example, if the training corpus is very small, many unknown words will likely be encountered when tagging new next. A lexicon would be able to help distinguish between words that are simply unknown and words that are known but just did not appear in the training corpus. Therefore, a semantic lexicon containing known words and their possible semantic tags was created.

The semantic types of the words listed in the lexicon were limited to DISEASE, MEDICATION, SYMPTOM, and OTHER. If a word had additional semantic types other than DISEASE, MEDICATION, or SYMPTOM (such as a *procedure* or *finding*), the additional semantic types were listed as OTHER because there was no training data for those semantic types. The words in the lexicon were extracted from the strings in the UMLS normalized string table (MRXNS_ENG). For each string, all of its possible concept identifiers were matched to entries in the UMLS Semantic Type (MRSTY) table [33], to find the possible semantic types. Because the statistical algorithms analyze the text on a word-by-word basis, the normalized strings that consisted of multiple words were broken up into their constituent words. Each word was added to the lexicon separately, and given the same semantic type as the full phrase. For example, a word like *chest* refers to a body part by itself, but when it appears in the phrase *chest pain*, it is part of a symptom name. Thus, for the word *chest*, both OTHER (i.e., a body part) and SYMPTOM are listed as possible semantic types in the lexicon. For phrases with multiple semantic types, the types were not ordered.

4.3 Statistical Extraction Methods

Two different statistical algorithms were developed to automatically extract medications, diseases, and symptoms from a collection of nursing notes. The first one uses a forward-based algorithm to evaluate the probability of a tag being assigned to a word, given the tags of the previous one, two, or three words. The second algorithm uses a Viterbi-based [24] best-path algorithm to find the combination of semantic tags that would produce the highest-probability tag sequence for a given nursing note. Both algorithms have the option of only using semantic training data from the gold standard corpus, or additionally using the syntactic data produced by the Brill tagger.

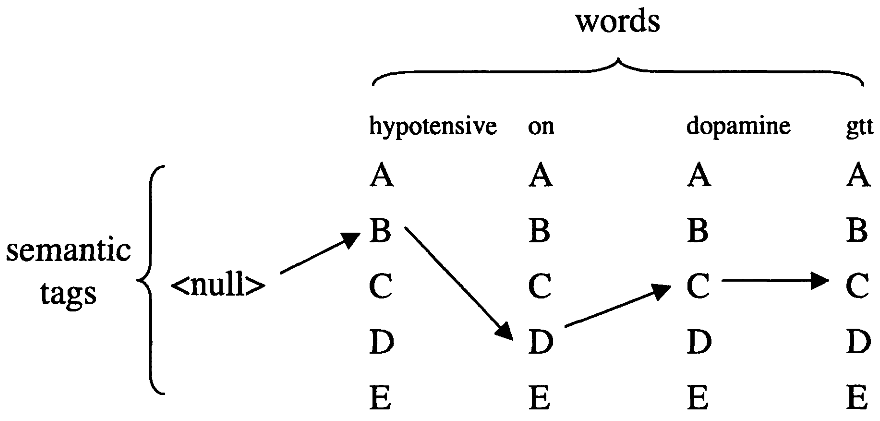


Figure 4-1: The forward-based algorithm calculates the most likely tag for each word by finding the probability of the word given the tag, and the probability that the tag occurs given the previous one, two, or three tags. Each semantic tag is considered for each word. There is an implicit "null" first word of each nursing note. This diagram shows an example phrase along with the possible semantic tags that each word from the phrase can have.

4.3.1 Forward-Based Algorithm

This method makes one pass through each nursing note, tagging each word based on the tags of previous words (see Figure 4-1). The training corpus and lexicon most likely are not complete, so relying solely on them for finding the possible semantic tags for a word may cause the algorithm to miss the correct tag. Thus, the algorithm considers every one of the semantic tags from the tagset for each word, in order to avoid mistakenly ruling out a correct tag for a word. For an n-gram implementation, the algorithm looks at the previous $n - 1$ tags to guess the tag for the nth word. Choosing a very small n (e.g., 2) provides less context with which to make a decision, whereas choosing a larger n gives more context but at the same time there might be fewer instances of each n-gram in the training set. The algorithm that was developed can be run using either bigram (2-stage), trigram (3-stage) or tetragram (4-stage) frequencies, and can be run with or without using the syntactic data from the Brill tagger. Comparisons were made between the output of each of the different modes.

Given an untagged nursing note, the program first tokenizes the note using the method described in Chapter 2. For each word, four different semantic tag types are considered: DISEASE, MEDICATION, SYMPTOM, and one of (OTHER, NUM, PUNCT, or STOP). For each possible tag, a score is computed for the word-tag pair, given the semantic tags of the previous one, two, or three words. Out of the four possible tags, the one yielding the highest score is assigned to the word. For the bigram implementation, without the use of syntactic data, the tagging equation used for finding the score for a tag T for the ith word, $word_i$, is:

$$score_T = P(word_i|T) * P(T|tag_{i-1}) \tag{4.4}$$

where tag_{i-1} is the semantic tag of the preceding word, $word_{i-1}$. $P(word_i|T)$, the probability of word $word_i$ occurring, given that its semantic tag is T, is calculated using:

$$P(word_i|T) = \frac{\#\ occurrences\ of\ word_i/T}{\#\ occurrences\ of\ T} \tag{4.5}$$

where the number of occurrences of $word_i$ being tagged with tag T ($word_i/T$) and the total number of times any word was tagged with T are both taken from the training corpus. The value of $P(T|tag_{i-1})$ is calculated as follows:

$$P(T|tag_{i-1}) = \frac{\#\ occurrences\ of\ tag_{i-1}, T}{\#\ occurrences\ of\ tag_{i-1}} \tag{4.6}$$

The total number of times in the training corpus that a word tagged with tag_{i-1} was followed immediately by a word tagged with T (tag_{i-1},T) is divided by the total number of occurrences of the tag tag_{i-1}, to find the probability of the semantic bigram, given the tag of the previous word. To tag the first word in the text, the semantic type of the previous word is assumed to be the empty string.

If the Brill output is used, the text to be tested must first be tokenized and tagged using the Brill tagger and medical lexicon. The Brill output is read in by the extraction program so that the score calculations can be performed with this extra data. With syntactic data included, the bigram equation for the score for a tag T is:

$$score_T = P(word_i, pos_i|T) * P(T|tag_{i-1}) \tag{4.7}$$

where $P(word_i, pos_i|T)$ is the probability of T occurring at $word_i$ given that the part of speech of $word_i$ is pos_i. The only difference from the non-syntactic equation is that the probability of the word having the given part of speech is now figured into the equation. Thus,

$$P(word_i, pos_i|T) = \frac{\#\ occurrences\ of\ word_i/pos_i/T}{\#\ occurrences\ of\ T} \tag{4.8}$$

An issue that often arises in a statistical training algorithm is that a new word, word-tag pair, or n-gram sequence that was not seen in the training corpus might be encountered in the text. Some method of smoothing needs to be involved, so that these unrecognized words and semantic tag sequences are not automatically ruled out. Therefore, unseen word-tag pairs were given a default small, non-zero value. However, by adding these extra frequencies, there is then the risk of reducing the relative frequency of words and tags that actually do appear in the training corpus. Thus, the result of $P(word_i|T)$ or $P(word_i, pos_i|T)$ is multiplied by a certain weight if the value is found in the training corpus. If it is not found in the training corpus but the lexicon contains T as a possible semantic type for $word_i$, then $P(word_i|T)$ or $P(word_i, pos_i|T)$ is multiplied by a smaller weight. These different weights are intended to ensure that entries in the training corpus receive the most weight, lexicon entries receive the next largest weight, and the word-tag pairs that do not occur in either the corpus or the lexicon are given the smallest frequency value. Similarly, for the n-gram frequency calculation, if the n-gram does not occur in the training corpus, then a default value is given. If it does occur in the training corpus, then the frequency is multiplied by a constant to give more weight to the n-gram that has been observed before. So, for example, the bigram equation becomes:

$$score_T = \frac{k1 * \#\ occurrences\ of\ word_i/T}{\#\ occurrences\ of\ T} * \frac{k2 * \#\ occurrences\ of\ tag_{i-1}, T}{\#\ occurrences\ of\ tag_{i-1}} \tag{4.9}$$

where $k1$ and $k2$ are the smoothing constants.

The trigram and tetragram equations are similar to the bigram equations, except that they count the frequencies of semantic trigrams and tetragrams, respectively.

For trigrams, the equation to calculate a tag's score, without the use of syntactic data, was:

$$score_T = P(word_i|T) * P(T|tag_{i-2}, tag_{i-1}) \tag{4.10}$$

where $P(word_i|T)$ is calculated as before. $P(T|tag_{i-2}, tagi - 1)$ is the probability of tag T occurring at word $word_i$, given that the tag for word $word_{i-2}$ is tag_{i-2} and the tag for word $word_{i-1}$ is tag_{i-1}. This value is calculated as follows:

$$P(T|tag_{i-2}, tagi - 1) = \frac{\#\ occurrences\ of\ tag_{i-2}, tag_{i-1}, tag_i}{\#\ occurrences\ of\ tag_{i-2}, tag_{i-1}} \tag{4.11}$$

If the syntactic option is used, then $P(word_i|T)$ becomes $P(word_i, pos_i|T)$, as with the bigram equation. The equations for tetragrams follow similarly, but involve the previous three consecutive tags.

After each word from the beginning to the end of the nursing note is tagged in this way, all of the consecutive words having the same semantic tag (either DISEASE, MEDICATION, or SYMPTOM) are extracted and returned as output. For example, if the note contains the text *admitted for mental status changes, treated for hypotension* and the program tagged the words as:

```
admitted/OTHER
for/STOP
mental/SYMPTOM
status/SYMPTOM
changes/SYMPTOM
,/PUNCT
treated/OTHER
for/STOP
hypotension/DISEASE
```

then the phrases that would be extracted are *mental status changes* (as a symptom) and *hypotension* (as a disease).

4.3.2 Best Path Algorithm

The other statistical algorithm that was explored was a Viterbi-based best path algorithm. It was also implemented for bigrams, trigrams, and tetragrams. Instead

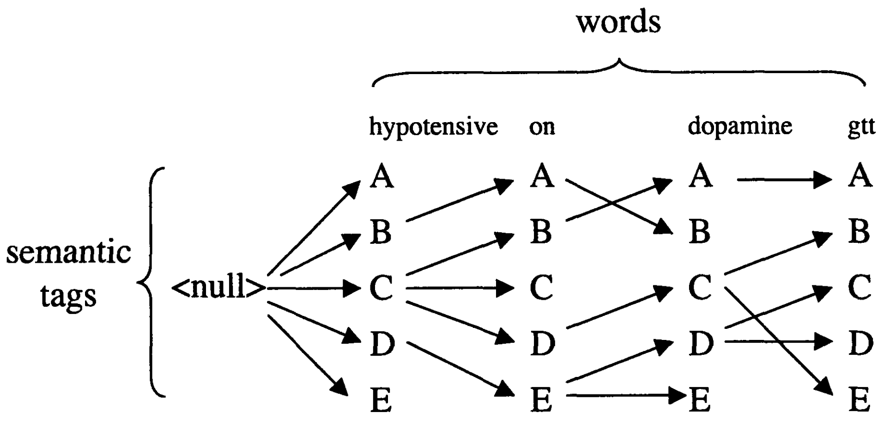


Figure 4-2: The best-path algorithm calculates the most likely previous tag for every possible tag for each word, using bigrams, trigrams, and tetragrams. Back pointers are added from each word-tag entry to the most likely tag of the preceding word. The most likely previous tag is calculated by maximizing the value of the scores calculated in Section 4.3.1, multiplied by the score of the previous tag. After all back pointers are created, the path with the highest score through the text is found by starting from the last word of the text with the highest score, and following its back pointers to the beginning.

of making a single forward pass through a nursing note, the path through the entire note with the highest score is chosen. A two-dimensional array, *backPtrs*, was used to store back pointers from each possible semantic type of each word to the tag of the previous word that gives the best score, using similar score calculations as those in the forward-based method. See Figure 4-2 for a depiction of this structure as used in the bigram implementation, where the back pointer at each node underneath a word refers back to a node that is underneath the previous word. The cost of each node is the score calculation for the given word $word_i$ and tag T (i.e., $P(word_i|T) * P(T|tag_{i-1})$ for bigrams), multiplied by the score of the node located at word $word_{i-1}$ and tag tag_{i-1}.

For trigrams and tetragrams, not only does the previous tag need to be stored as a node, but all combinations of the previous $n-1$ tags need to be stored, and the score must be calculated for each possibility. Thus, the back pointer array is of size $numWords$ by $numTags^{n-1}$, where $numWords$ is the total number of words in the text, n is the size of the n-grams being used (2, 3, or 4), and $numTags$ is the number

```
bestPath(note):
  tokenize note into words
  for each word $w_i$ in note
    for each possible tag $t_i$ for $w_i$
      find tag $t_{i-1}$ (for word $w_{i-1}$) that gives best score
      create edge from ($w_{i-1}$,$t_{i-1}$) to ($w_i$,$t_i$)
  find best path through note
```

Figure 4-3: Pseudo-code showing the best path tagging algorithm, where each (word, semantic tag) pair is a node, and scores are calculated for paths between nodes based on n-gram probabilities. The best path through the nursing note is found using a graph search algorithm similar to the one described in Chapter 3, except that here, the best path is the path with the highest score.

of different possible semantic tags for each word, which in this implementation is four (DISEASE, MEDICATION, SYMPTOM, and one of OTHER, PUNCT, NUM, or STOP). Although the size of the array grows exponentially with n, only a subset of the nodes at $word_{i-1}$ have to be considered when creating back pointers at $word_i$. For example, in the tetragram implementation, at each possible node, every possible combination of the previous two semantic tags and the current semantic tag is stored. However, every node containing the n-gram sequence `ABC` must be preceded by a node ending in the sequence `AB`, so only those preceding nodes are considered. Figure 4-3 shows a summary of how the best path algorithm works.

4.4 Testing and Results

A series of automated tests was run to evaluate the relative performance of each of the extraction algorithms, using their various configurations. For both the foward-based and best-path algorithms, the bigram, trigram, and tetragram methods were tested, both with and without syntactic data. For each test, a random set of 20 notes was extracted out of the 252 total gold standard nursing notes to be used as the testing set, and the other 232 notes were used as the training set. Each test was repeated five times, using different randomly chosen training and testing sets. The smoothing constants were chosen at run time with some trial and error (developing a rigorous

Table 4.5: The phrase-based and word-based precision (p) and recall (r) for the forward-based extraction algorithm. Results are shown for bigrams (n=2), trigrams (n=3), and tetragrams (n=4), with and without the use of syntactic data, for diseases, medications, and symptoms.

		Using Syntactic				Not Using Syntactic			
n	Semantic	p_{phrase}	r_{phrase}	p_{word}	r_{word}	p_{phrase}	r_{phrase}	p_{word}	r_{word}
2	Diseases	58.2%	56.4%	66.4%	56.0%	60.1%	57.7%	68.8%	57.7%
2	Medications	64.1%	83.2%	65.5%	83.3%	67.3%	84.2%	68.8%	83.6%
2	Symptoms	46.0%	31.3%	71.2%	30.7%	48.9%	33.0%	75.2%	32.2%
2	Overall	62.8%	64.4%	72.7%	58.2%	65.2%	65.2%	75.4%	58.7%
3	Diseases	51.9%	57.6%	58.9%	59.5%	56.0%	58.7%	62.6%	59.8%
3	Medications	67.9%	83.6%	69.7%	83.5%	70.4%	85.3%	72.2%	84.5%
3	Symptoms	46.6%	34.9%	67.1%	33.7%	49.2%	36.6%	69.5%	34.6%
3	Overall	62.6%	65.5%	71.2%	59.5%	65.4%	66.8%	73.5%	59.9%
4	Diseases	44.3%	58.0%	51.5%	59.9%	47.4%	59.2%	54.2%	60.2%
4	Medications	68.8%	82.2%	71.0%	83.3%	71.5%	84.1%	73.6%	84.2%
4	Symptoms	42.3%	35.6%	63.4%	35.9%	43.5%	36.9%	64.2%	36.8%
4	Overall	59.3%	65.5%	68.4%	60.6%	61.5%	66.8%	70.4%	61.3%

method to find the best smoothing constants was out of the scope of this project). The average values of the precision and recall for the different configurations are shown in Tables 4.5 and 4.6. After reviewing the output of the algorithm, it seemed like many of the phrases were successfully extracted, but extracted into the wrong category. Thus, the *overall* precision and recall are also given, based on the total phrases and words extracted, disregarding which of the specific semantic categories they were tagged as.

Medications generally were tagged with both the highest precision and recall, and symptoms were usually the most difficult to extract. Running each of the n-gram methods using a fixed set of smoothing constants produced better results for smaller n. The addition of the Brill syntactic tags caused some phrases to be joined together, such as *rales* and *bases* forming the phrase *rales in bases.* However, it generally did not improve the overall precision and recall of the algorithm.

Table 4.6: The phrase-based and word-based precision (p) and recall (r) for the best-path extraction algorithm. Results are shown for bigrams (n=2), trigrams (n=3), and tetragrams (n=4), with and without the use of syntactic data, for diseases, medications, and symptoms.

		Using Syntactic				Not Using Syntactic			
n	Semantic	p_{phrase}	r_{phrase}	p_{word}	r_{word}	p_{phrase}	r_{phrase}	p_{word}	r_{word}
2	Diseases	35.8%	55.9%	33.5%	67.8%	38.6%	57.0%	35.7%	67.6%
2	Medications	70.1%	79.1%	72.2%	81.4%	72.7%	80.5%	74.7%	81.7%
2	Symptoms	42.0%	38.0%	49.4%	53.1%	44.1%	39.0%	52.6%	53.6%
2	Overall	56.4%	65.0%	55.0%	71.7%	59.3%	66.2%	58.3%	71.6%
3	Diseases	24.8%	56.7%	27.1%	70.1%	27.0%	57.5%	29.2%	69.4%
3	Medications	69.3%	80.3%	71.8%	83.7%	71.0%	81.8%	73.7%	84.3%
3	Symptoms	40.2%	40.0%	49.2%	53.4%	40.1%	39.6%	50.4%	53.8%
3	Overall	49.2%	66.8%	51.0%	73.4%	51.0%	67.1%	53.2%	72.9%
4	Diseases	21.8%	54.4%	21.2%	68.8%	23.7%	54.2%	22.7%	66.8%
4	Medications	69.5%	81.4%	71.6%	83.2%	69.4%	82.0%	71.8%	83.5%
4	Symptoms	38.2%	42.6%	47.4%	53.6%	40.0%	44.4%	48.7%	54.6%
4	Overall	46.7%	67.6%	46.7%	73.5%	48.6%	68.2%	48.8%	73.3%

4.5 Discussion

One of the main weaknesses of the statistical methods presented is that they do not use proper smoothing constants and the extracted output can change very significantly if a small change is made to one of the weights. To find the values that best fit the data, it is often necessary to perform some type of rigorous cross validation. The simple trial and error method of guessing the smoothing constants was shown to be inadequate, and is something that should be improved upon in future algorithms. There are many free or open source tools available that perform more complicated statistical training, and although they were not explored in this research, they could potentially be useful for phrase extraction, provided there is a large enough training corpus.

Adding the part of speech of each word as an extra observation point lowered the precision and recall in almost all cases. There may be several possible reasons for the negative effect of incorporating the syntactic observations. The Brill tagger was trained on more formal and well-formed text than the nursing notes, and thus the syntactic tags assigned to the gold standard corpus of nursing notes might not have been accurate. There was no human validation done on the syntactic tags, but future work might involve re-training the Brill tagger using a corpus of nursing notes tagged with human-verified parts of speech.

The use of a relatively small training corpus may have been a disadvantage, because there was not a large amount of semantic data that could be used to train the statistical algorithms. The corpus of nursing notes contained only 50,000 tokenized words, whereas the Wall Street Journal corpus that the Brill tagger was trained on contains millions of words. However, the extraction algorithms were still able to find the majority of phrases using this small training corpus, which suggests that there is not as much variation in the language used in the nursing notes as there is in more formally structured text.

One possible improvement to the algorithm might be the use of a larger semantic tagset. Experiments were done using a lexicon containing a larger number of different semantic types, such as procedures, modifiers, and body parts. For each of the statistical methods, every possible semantic type was considered for each word. The results of the extraction algorithm improved slightly by using these extra semantic types, but the use of these additional tags was not further explored.

From the test results, the recall is very low (30-50%) in many cases, for both symptoms and diseases, possibly because they are generally more complex in structure than medications. The problem might also have been caused by the fact that many words in the nursing notes did not match exactly with words in the lexicon or training set. Another option might be to use a tool such as the Porter Stemmer [5] to stem all words in the text and training corpus, and then search on the stemmed words. However, judging by a small test, the use of the Porter Stemmer only slightly increases the recall (i.e., by about 1%), while lowering the precision drastically (by 10-15%).

Overall, the results show that the statistical algorithms are promising, in that the majority of phrases were extracted successfully (even with a relatively small gold standard corpus), and it only took at most a few seconds (on a Pentium III machine) to process each note. With a larger and more accurately tagged gold standard corpus, along with better ways to estimate smoothing constants, such algorithms can be very useful in real-time analysis of textual patient data.

Chapter 5

Conclusions and Future Work

In this project, methods to code free-text medical phrases and extract phrases from unstructured notes have been developed and evaluated. The free-text coding algorithm translates unstructured phrases, such as those used in the clinical nursing notes, into standardized concept identifiers. This algorithm can also be used in conjunction with additional training data and statistical algorithms, to automatically extract phrases of interest from the nursing notes.

It has been shown that the automatic coding algorithm performs rapidly enough to be used in a real-time setting, and when the vocabulary is expanded to include the entire UMLS, it outperforms MMTx, an existing open source tool. The interactive version of the coding algorithm is likewise efficient and able to find most concepts on the first try. It is currently being used by clinical experts in an annotation setting. There are possible additions that can be made to the user interface, such as filtering by semantic category and searching by more fine-grained relationship types (e.g., *associated morphology of* or *has procedure site*). There is also room for improvement in other areas of the algorithm, such as having more intelligent spell checking and abbreviation expansion. Test results show that limiting the medical vocabulary to SNOMED-CT may not be ideal for annotation, because many commonly used medication names are not included in the SNOMED-CT vocabulary. However, it has been shown that incorporating a medication vocabulary or searching on the entire UMLS for related concepts are simple solutions to this limitation. Although it was out of the scope of this project, it would also be useful in the future if the algorithm could look at numerical values and measurements and be able to infer from them that a patient exhibits certain symptoms or problems.

The algorithms created for automatic clinical phrase extraction are efficient and can potentially be useful to clinicians; however, the algorithms require a lot of supplemental data, including lexicons and a training corpus of manually tagged data. Only a subset of phrase types (diseases, medications, and symptoms) was extracted in this project, but extracting other types of phrases (such as procedures and body parts) might also be useful. Another useful extension is to code the extracted phrases into UMLS concepts, based on the context in which they appear. Some helpful additions to the algorithm might include the use of existing open source tools that have more sophisticated methods to guess smoothing constants, or possibly the use of author recognition tools along with the training data to better decipher the meaning of the notes.

Even though the gold standard corpus used in the tests was relatively small, the majority of medication, disease, and symptom names were successfully extracted from the tested nursing notes, and the algorithms performed quickly enough to be used in a real time setting. In the future, a larger and more consistently tagged corpus might improve results further. The coding and extraction techniques that have been developed will hopefully be useful tools in future medical applications.

Appendix A

Sample Re-identified Nursing Notes

O: 58 YEAR OLD FEMALE ADMITTED IN TRANSFER FROM CALVERT HOSPITAL FOR MENTAL STATUS CHANGES POST FALL AT HOME AND CONTINUED HYPOTENSION AT CALVERT HOSPITAL REQUIRING DOPAMINE; PMH: CAD, S/P MI 1992; LCX PTCA; 3V CABG WITH MVR; CMP; AFIB- AV NODE ABLATION; PERM PACER- DDD MODE; PULM HTN; PVD; NIDDM; HPI: 2 WEEK HISTORY LEG WEAKNESS; 7/22 FOUND BY HUSBAND ON FLOOR-AWAKE, BUT MENTAL STATUS CHANGES; TO CALVERT HOSPITAL ER- TO THEIR ICU; HEAD CT- NEG FOR BLEED; VQ SCAN- NEG FOR PE; ECHO- GLOBAL HYPOKINESIS; EF EST 20%; R/O FOR MI; DIGOXIN TOXIC WITH HYPERKALEMIA- KAYEXALATE, DEXTROSE, INSULIN; RENAL INSUFFICIENCY- BUN 54, CR 2.8; INR 7 (ON COUMADIN AT HOME); 7/23 AT CALVERT- 2 FFP, 2 UNITS PRBC, VITAMIN K; REFERRED TO GH.
ARRIVED IN TRANSFER APPROX. 2130; IN NO MAJOR DISTRESS; DOPAMINE TAPER, THEN DC; NS FLUID BOLUS GIVEN WITH IMPROVEMENT IN BP RANGE; SEE FLOW SHEET SECTION FOR CLINICAL INFORMATION; A: NO HEMODYNAMIC COMPROMISE SINCE TRANSFER; TOLERATING DOPAMINE DC; P: TREND BP RANGE; OBSERVE FOR PRECIPITOUS HYPOTENSION.

ros:
neuro: a&ox3, mae. at times anxious, ativan x1 per pej w/ fair effect. prn pain med for c/o ha.
cv: nsr -> st no ectopy. htn when anxious. denies cp. to fluro for (r) basilic 4fr single lumen picc.
resp: sx for thick tan secretions. strong cough. pox 95-98. gi: tf cont at goal. pos flatus no stool. fsbs tx w/ riss. denies n/v.
gu: u/o >60cc/h.
plan: to rehab when vent bed available. cont w/ current plan of care.

```
NPN
CCU
7 PM - 7 AM
VF ARREST C/B ANOXIC INJURY
CV HEMODYN STABLE LIDO D/C'D AT 0400..WITH UNIFOCAL PVCS ...AM K AND MG
PNDG...SBP 110-130'S/60'S...
RESP AC MODE ..RATE OF 12 ..OVERBREAHING 2-4 ..TV 650..40% 5
PEEP...SUCTIONED Q2-3 FOR MOD AMOUNTS OF TAN SXNS ..LUNGS COARSE ..
GI OGT CLAMPED ..MINIMAL OUTPUT ..NO STOOL
GU URINE OUTPUT QS ...
NEURO NO PURPOSEFUL MOVEMENT NOTED ....PLS SEE FLOWSHEET FOR NEURO
ASSESSMENT ...
A HEMODYN STABLE OF ANTI ARRTHYMICS ..VENT IN PLACE
P AWAIT NEURO CONSULT ..FAMILY AT BEDSIDE
```

Appendix B

UMLS to Penn Treebank Tag Translation

Table B.1: UMLS parts of speech translated to Penn Treebank parts of speech.

sca	agr	Penn	Description
adj	comparative	JJR	adjective, comparative
	positive	JJ	adjective or numeral, ordinal
	positive;periph	JJ	adjective or numeral, ordinal
	superlative	JJS	adjective, superlative
adv	comparative	RBR	adverb, comparative
	positive	RB	adverb
	positive;periph	RB	adverb
	superlative	RBS	adverb, superlative
	Exceptions: 1. *whereupon*, *whereof*, *whyever*, *why*, *whither*, *wherever*, *wherein*, *whereby*, *where*, *whenever*, *whence*, *when*, *how*, *however* should be tagged as WRB (WH-adverb)		
conj	-	CC	conjunction, coordinating
prep	-	IN	preposition or conjunction, subordinating
det	-	DT	determiner
	Exceptions: 1. *what*, *whatever*, *which*, *whichever* should be tagged as WDT (WH-determiner) 2. *that* should be tagged as both DT and WDT		
modal	-	MD	modal auxiliary
noun	count(thr_plur)	NNS	noun, common, plural
	count(thr_sing)	NN	noun, common, singular or mass
	uncount(thr_plur)	NNS	noun, common, plural

continued on the next page

blah blah (*continued*)

sca	agr	Penn	Description
	uncount(thr_sing)	NN	noun, common, singular or mass
	Exceptions: 1. NNS should be changed to NNPS (noun, proper, plural) if LRPRP.fea='proper' 2. NN should be changed to NNP (noun, proper, singular) if LRPRP.fea='proper'		
pron	-	PRP	pronoun, personal
	Exceptions: 1. *her*, *hers*, *his*, *its*, *mine*, *my*, *our*, *ours*, *their*, *theirs*, *your*, *yours* should be tagged as PRP$ (pronoun, possessive) 2. *that*, *these*, *this*, *those*, *what*, *whatever*, *when*, *which*, *whichever*, *who*, *whoever*, *whom*, *whomever*, *whatsoever* should be tagged as WP (WH-pronoun) 3. *whose* should be tagged as WP$ (WH-pronoun, possessive) 4. *her* should be tagged as both PRP and PRP$		
verb	infinitive	VB	verb, base form
	past	VBD	verb, past tense
	past_part	VBN	verb, past participle
	pres(fst_sing, fst_plur, thr_plur, second)	VBP	verb, present tense, not 3rd person singular
	pres(thr_sing)	VBZ	verb, present tense, 3rd person singular
	pres_part	VBG	verb, present participle or gerund
aux	infinitive	VB	verb, base form
	past	VBD	verb, past tense
	past(fst_plur, second, thr_plur)	VBD	verb, past tense
	past(fst_plur, second, thr_plur):negative	VBD	verb, past tense
	past(thr_sing, fst_sing)	VBD	verb, past tense
	past(thr_sing, fst_sing):negative	VBD	verb, past tense
	past:negative	VBD	verb, past tense
	past_part	VBN	verb, past participle
	pres(fst_plur, second, thr_plur)	VBP	verb, present tense, not 3rd person singular
	pres(fst_plur, second, thr_plur):negative	VBP	verb, present tense, not 3rd person singular
	pres(fst_sing)	VBP	verb, present tense, not 3rd person singular
	pres(fst_sing, fst_plur, second, thr_plur)	VBP	verb, present tense, not 3rd person singular
	pres(fst_sing, fst_plur, second, thr_plur):negative	VBP	verb, present tense, not 3rd person singular
	pres(thr_sing)	VBZ	verb, present tense, 3rd person singular
	pres(thr_sing):negative	VBZ	verb, present tense, 3rd person singular
	pres_part	VBG	verb, present participle or gerund

Bibliography

[1] GSpell. http://specialist.nlm.nih.gov/SpellingResources.html.

[2] IndexFinder demo. http://fargo.cs.ucla.edu/umls/demo.aspx.

[3] MetaMap Transfer (MMTx) Home. http://mmtx.nlm.nih.gov/.

[4] RxNorm. http://www.nlm.nih.gov/research/umls/rxnorm main.html.

[5] The Porter Stemming Algorithm. http://www.tartarus.org/martin/PorterStemmer/.

[6] Aggregating UMLS Semantic Types for Reducing Conceptual Complexity. In *Proceedings of MEDINFO 2001*, volume 10(Pt 1), pages 216–220, 2001.

[7] A knowledge-based approach for retrieving scenario-specific medical text documents. *Control Engineering Practice*, Volume 13, Issue 9:1105–1121, September 2005.

[8] O. T. Abdala, G. D. Clifford, M. Saeed, A. Reisner, G. B. Moody, I. Henry, and R. G. Mark. The annotation station: An open-source technology for annotating large biomedical databases. *Computers in Cardiology*, 31:681–685, 2004.

[9] Omar T. Abdala. The Annotation Station: An open source technology for data visualization and annotation of large biomedical databases. Master's thesis, Massachusetts Institute of Technology, 2005.

[10] A. Aronson. Effective Mapping of Biomedical Text to the UMLS Metathesaurus: The MetaMap Program. In *Proceedings of the AMIA 2001 Annual Symposium*, pages 17–21, 2001.

[11] J. J. Berman. Pathology abbreviations and acronyms. *http://www.pathinfo.com/abbtwo.htm*, May 2001.

[12] Neha Bhooshan. Classification of Semantic Relations in Different Syntactic Structures in Medical Text using the MeSH Hierarchy. Master's thesis, Massachusetts Institute of Technology, 2005.

[13] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, IT, 1992.

[14] Friedman C. Towards a comprehensive medical language processing system: methods and issues. In *Proceedings of the AMIA 1997 Annual Fall Symposium*, pages 595–599, 1997.

[15] Friedman C. A Broad-Coverage Natural Language Processing System. In *Proceedings of the AMIA 2000 Annual Symposium*, pages 270–274, 2000.

[16] Friedman C, Alderson PO, Austin JH, Cimino JJ, and Johnson SB. A general natural-language text processor for clinical radiology. *Journal of the American Medical Informatics Association*, 1(2):161–174, 1994.

[17] Jean Carletta. Assessing Agreement on Classification Tasks: The Kappa Statistic. *Computational Linguistics*, 22(2):249–254, 1996.

[18] College of American Pathologists. SNOMED International. http://www.snomed.org/index.html.

[19] College of American Pathologists. SNOMED Clinical Terms Technical Specification: Revision 23, 2000.

[20] Margaret Douglass. Computer-Assisted De-Identification of Free-text Nursing Notes. Master's thesis, Massachusetts Institute of Technology, 2005.

[21] Margaret Douglass and Jennifer Shu. Information Extraction from Medical Patient Notes. MIT 6.863 class project, May 2004.

[22] Lexical Systems Group. http://specialist.nlm.nih.gov/LexSysGroup/index.html.

[23] Lexical Systems Group. Lexical Tools, 2005 Release. http://umlslex.nlm.nih.gov/lvg/current/index.html.

[24] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*, chapter 5.9. Prentice-Hall, Inc., 2000.

[25] NLM Lexical Systems Group homepage. http://umlslex.nlm.nih.gov/.

[26] Link Grammar Parser. http://bobo.link.cs.cmu.edu/link/.

[27] William Long. Extracting Diagnoses from Discharge Summaries.

[28] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[29] R. G. Mark. Integrating Data, Models and Reasoning in Critical Care, 2003. National Institute of Biomedical Imaging and Bioengineering Proposal R01 EB001659.

[30] MedLEE homepage. http://lucid.cpmc.columbia.edu/medlee/.

[31] National Library of Medicine. About the UMLS Resources. http://www.nlm.nih.gov/research/umls/about_umls.html.

[32] National Library of Medicine. The SPECIALIST LEXICON: UMLS Documentation, 2004. http://nls4.nlm.nih.gov:8000/SPECIALIST/UMLSLexDocs/.

[33] National Library of Medicine. UMLS Knowledge Sources, 15th Edition - April Release: 2004AA Documentation, 2004. http://www.nlm.nih.gov/research/umls/archive/2004AA/UMLSDOC.html.

[34] National Institutes of Health. http://privacyruleandresearch.nih.gov/.

[35] National Institutes of Health. Policy and Procedures for De-Identification of Protected Health Information and Subsequent Re-Identification. *NIH*, 2004. Document 45 CFR 164.514(a)-(c).

[36] Penn Treebank Tag set. http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html.

[37] Zou Q, Chu WW, Morioka C, Leazer GH, and Kangarloo H. IndexFinder: A Method of Extracting Key Concepts from Clinical Texts for Indexing. In *Proceedings of the AMIA 2003 Annual Symposium*, pages 763–767, 2003.

[38] M. Saeed, C. Lieu, G. Raber, and R.G. Mark. MIMIC II: A Massive Temporal ICU Patient Database to Support Research in Intelligent Patient Monitoring. *Computers in Cardiology*, 29:641–644, 2002.

[39] N. Sager, M. Lyman, C. Bucknall, N. Nhan, and L. Tick. Natural Language Processing and the Representation of Clinical Data. *Journal of the American Medical Informatics Association*, 1:142–160, 1994.

[40] SourceForge.net. Jazzy - Java Spell Check API. http://sourceforge.net/projects/jazzy.

[41] P Szolovits. Adding a Medical Lexicon to an English Parser. In *Proceedings of the AMIA 2003 Annual Symposium*, pages 639–643, 2003.