

**Using Symbolic Network Logical Analysis  
as a Knowledge Extraction Method on MEDLINE Abstracts**

by

**Aaron Michael Cohen, M.D.**

A THESIS

Presented to the Department of Medical Informatics and Clinical Epidemiology  
and the Oregon Health & Science University  
School of Medicine  
in partial fulfillment of the requirements for the degree of  
Master of Science  
August 2004

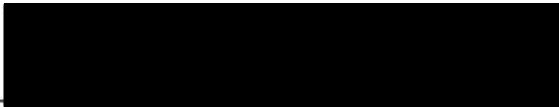
School of Medicine  
Oregon Health & Science University


---

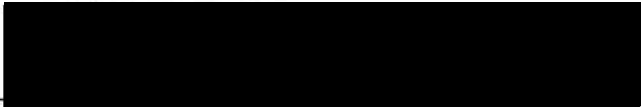
CERTIFICATE OF APPROVAL


---

This is to certify that the Master's thesis of  
Aaron Michael Cohen  
has been approved.

  
\_\_\_\_\_  
Professor in charge of thesis

  
\_\_\_\_\_  
Thesis committee member

  
\_\_\_\_\_  
Thesis committee member



## TABLE OF CONTENTS

|   |     |
|---|-----|
| TABLE OF CONTENTS.....  | i   |
| TABLE OF FIGURES.....   | iv  |
| TABLE OF EQUATIONS.....   | v   |
| ACKNOWLEDGEMENTS.....   | vi  |
| ABSTRACT.....   | vii |
| INTRODUCTION .....  | 1   |
| I. BACKGROUND AND SIGNIFICANCE.....   | 2   |
| II. PROBLEM DEFINITION.....   | 8   |
| Focus on Bibliographic Data .....   | 8   |
| The Process of Knowledge Extraction.....  | 9   |
| Bounded Problem Domain.....   | 10  |
| Data Set.....   | 10  |
| Initial Knowledge.....  | 10  |
| Instance Knowledge.....   | 10  |
| Recognition Patterns .....  | 10  |
| Application of Patterns to the Data Set.....                                      | 11  |
| Extraction of Instance Knowledge.....   | 11  |
| Rating of Instance Confidence.....  | 11  |
| Recognition of New Patterns .....   | 12  |
| Rating of Pattern Utility.....  | 12  |
| Evaluation .....  | 12  |
| Gold Standard .....   | 12  |
| Results.....  | 12  |
| Process Framework for Knowledge Extraction .....                                  | 13  |
| Purpose of Study.....   | 13  |
| Gene and Protein Name Synonymy as the Initial Problem Domain.....                 | 15  |
| Research Question .....   | 19  |
| III. LITERATURE REVIEW .....  | 20  |
| High level knowledge extraction tasks .....                                       | 20  |
| Early work.....   | 20  |
| Current Work .....  | 23  |
| Related Research.....   | 25  |
| Other work .....  | 53  |
| Components of the knowledge extraction process.....                               | 55  |
| Prior work on the use of initial instance data as seed data.....                  | 55  |
| Prior work on the manual and automatic generation of extraction patterns .....    | 56  |
| Prior work on the use of graph and network analysis for knowledge extraction..... | 58  |
| Prior work on evaluation of results of medical knowledge extraction systems.....  | 59  |
| IV. THEORETICAL FRAMEWORK.....  | 65  |
| Basic Theoretical Foundation .....  | 65  |

|   |     |
|---|-----|
| Representing and Inferring Knowledge with Symbolic Networks .....       | 67  |
| Edge and Relationship Types.....  | 68  |
| Integration with the Knowledge Extraction Framework .....               | 73  |
| SNLA Provides a Unified Approach .....                                  | 76  |
| V. RESEARCH QUESTION AND OPERATIONAL DEFINITIONS .....                  | 78  |
| Applying SNLA to the Research Question.....                             | 78  |
| Operational Definitions.....  | 78  |
| Gene and protein names.....   | 78  |
| Gene and protein name synonyms .....                                    | 79  |
| Applying the Knowledge Extraction Framework to Synonym Extraction.....  | 80  |
| Bounded Problem Domain.....   | 81  |
| Data Set.....   | 81  |
| Initial Knowledge.....  | 82  |
| Instance Knowledge.....   | 83  |
| Recognition Patterns .....  | 83  |
| Application of Patterns to the Data Set.....                            | 84  |
| Extraction of Instance Knowledge.....                                   | 85  |
| Rating of Instance Confidence.....                                      | 85  |
| Recognition of New Patterns .....                                       | 87  |
| Rating of Pattern Utility.....  | 87  |
| Evaluation .....  | 90  |
| Gold Standards.....   | 90  |
| Results.....  | 91  |
| Algorithm for Gene and Protein Synonym Extraction .....                 | 91  |
| VI. SOFTWARE IMPLEMENTATION .....                                       | 95  |
| System Components.....  | 96  |
| Implementation Parameters, System Constants, and Design Decisions ..... | 97  |
| Symbol Delimiters .....   | 98  |
| Case Insensitivity.....   | 98  |
| Pattern Length.....   | 98  |
| Determining Possible Gene Names .....                                   | 99  |
| Filtering Out Non-Contributing Sentences .....                          | 100 |
| High Confidence Pair Selection Parameters .....                         | 101 |
| Pattern Selection System Parameters.....                                | 101 |
| Synonym Inference Required Co-occurrences .....                         | 103 |
| Genetic Optimization Parameters .....                                   | 103 |
| VII. EXPERIMENTAL DESIGN .....  | 106 |
| Data Sets .....   | 107 |
| Gold Standards.....   | 107 |
| Precision Gold Standard .....   | 108 |
| Recall Gold Standard .....  | 109 |
| Application of Algorithm to the Data Sets .....                         | 110 |
| Computation of Performance Measures.....                                | 110 |
| Computation of Precision .....  | 110 |
| Computation of Recall .....   | 111 |
| Computation of F-score and Maximum Performance .....                    | 111 |



|  |     |
|--|-----|
| VIII. RESULTS .....  | 113 |
| Performance Measures .....                                   | 113 |
| Error Analysis .....   | 120 |
| Incorporation of Error Analysis in Performance Results ..... | 122 |
| IX. DISCUSSION AND LIMITATIONS .....                         | 124 |
| X. CONCLUSIONS AND FUTURE WORK .....                         | 130 |
| REFERENCES .....   | 132 |
| APPENDIX A - SOURCE CODE .....                               | 141 |
| Source file: SNLASynonyms.py .....                           | 141 |
| Source file: SNLAPatterns.py .....                           | 152 |
| Source file: SNLACoOccurrences.py .....                      | 157 |
| Source file: SNLAGraph.py .....                              | 158 |
| Source file: Table.py .....                                  | 164 |
| Source file: ExtractAbstractSentences.py .....               | 166 |
| Source file: ScreenSentences.py .....                        | 170 |
| APPENDIX B - ERROR ANALYSIS RAW DATA .....                   | 171 |
| Precision Error Analysis Raw Data .....                      | 171 |
| Recall Error Analysis Raw Data .....                         | 174 |

## TABLE OF FIGURES

|   |     |
|---|-----|
| Figure 1: Knowledge Extraction Process Framework .....                          | 13  |
| Figure 2: Swanson's ABC Model (from Weeber [27]) .....                          | 21  |
| Figure 3: Simple Bidirectional Network.....                                     | 70  |
| Figure 4: Simple Unidirectional Network.....                                    | 71  |
| Figure 5: Relationships between Thalidomide, IL-12, and acute pancreatitis..... | 72  |
| Figure 6: Synonym Extraction Algorithm .....                                    | 94  |
| Figure 7: Entity-Relationship Diagram of Software Architecture.....             | 95  |
| Figure 8: Experimental Design .....   | 106 |
| Figure 9: Precision verses Recall by Iteration .....                            | 114 |
| Figure 10: SNLA F-score by Iteration.....                                       | 115 |
| Figure 11: SNLA Number Verified Pairs by Iteration.....                         | 116 |
| Figure 12: SNLA Precision and Recall Comparison with Prior Work.....            | 117 |
| Figure 13: SNLA Maximum F-score Comparison with Prior Work .....                | 118 |
| Figure 14: Comparison of Number Seeds, Verified Pairs, and Ratio.....           | 119 |
| Figure 15: Maximum F-score Comparison including Precision Errors.....           | 123 |

## TABLE OF EQUATIONS

|                     |     |
|---------------------|-----|
| (Equation 1) .....  | 86  |
| (Equation 2) .....  | 86  |
| (Equation 3) .....  | 86  |
| (Equation 4) .....  | 89  |
| (Equation 5) .....  | 89  |
| (Equation 6) .....  | 89  |
| (Equation 7) .....  | 89  |
| (Equation 8) .....  | 89  |
| (Equation 9) .....  | 105 |
| (Equation 10) ..... | 111 |
| (Equation 11) ..... | 111 |
| (Equation 12) ..... | 112 |

## ACKNOWLEDGEMENTS

I'd like to thank my thesis committee for all their help along the way:

- Kent Spackman, M.D., Ph.D.
- William Hersh, M.D.
- Christopher Dubay, Ph.D.
- P. Zoë Stavri, Ph.D.

And my wife, Linda, for her continued support in this and all my other endeavors.

## ABSTRACT

OBJECTIVE: The *bibliome* of biomedical literature is already too large and growing much too rapidly for researchers to stay current on all information relevant to their work. Text-mining and knowledge extraction (KE) can assist researchers by analyzing bibliographic databases as a whole, and extracting knowledge by connecting information between multiple records. Symbolic Network Logical Analysis (SNLA), a novel text-mining method based on analyzing the network structure created by symbol co-occurrences, was developed as a way to extend the capabilities of KE. METHODS: Computer software was created applying SNLA to the task of automatic gene and protein name synonym extraction. Datasets containing ~50,000 abstracts were created from a year's worth of MEDLINE records containing the word "gene." Abstracts from 2001, 2002, and 2003 served as training, validation, and test sets, respectively. Synonyms extracted from controlled gene name terminologies were used as a gold standard. Performance was measured by recall and precision combined via the F-score. RESULTS: The system obtained a maximum F-score of 22.21% (23.18% precision and 21.36% recall). Error analysis provided insight into the strengths and weaknesses of SNLA. CONCLUSIONS: SNLA compared favorably with other gene name synonym extraction methods, and has a wide range of potential applications in KE.

## INTRODUCTION

This work is divided into ten sections. Section I covers the background and significance of knowledge extraction and text mining as it applies to the biomedical literature. Section II first defines the problem-space, and then presents an operational definition of the knowledge extraction process and its components. The Section II introduces an approach to knowledge extraction called *Symbolic Network Logical Analysis* (SNLA), discusses the purpose of the research, and defines the research question. Section III surveys the current biomedical literature most relevant to the development of SNLA methods and the problem domains to which it will be applied. Section IV presents the theoretical framework of SNLA, including the graph and network theory, and mathematical analysis on which it is based. Section V restates the research question and presents the operational definitions and constructs necessary for the application of SNLA to gene and protein name synonym extraction. The specific algorithm to be used will also be described here. Section VI presents the design of the software system that implements the algorithm described in the previous section. Section VII details the experimental design, which includes what will be measured and how the performance of the system will be evaluated. Section VIII presents the results of the gene and protein name synonym extraction and computation of basic statistics on these data. Section IX discusses the results, what they mean, error analysis, and limitations. Section X summarizes what has been learned and discusses potential directions for future work.

## I. BACKGROUND AND SIGNIFICANCE

The volume of published biomedical research, and therefore the underlying biomedical knowledge base, is expanding at a fantastic pace. While scientific information in general has been growing exponentially for several centuries [1], the absolute numbers specific to modern medicine are very impressive. The MEDLINE 2004 baseline distribution contains over 12.5 million records [2], and the database is currently growing at the rate of 500,000 new citations each year [3]. With such explosive growth, it is extremely challenging to keep up-to-date with all of the new discoveries and theories even within one's own field of biomedical research.

In discussing the biomedical literature, it is useful to differentiate between three components of the biomedical knowledge base: data, information, and knowledge. Data are simply recorded observations, for example, "the patient's cholesterol was 280mg/dL." Information is a judgment based on that data, for example, "a cholesterol of 280mg/dL is significantly higher than normal." Knowledge is the ability to make a decision based on general principles derived from that information, perhaps to produce a desired result. For example, "the patient has an elevated risk of heart disease, we should treat with a statin-type drug to lower the cholesterol." Knowledge can also be build on top of other knowledge, for example, "the 20 year old patient with normal cholesterol has a gene associated with high cholesterol, and will need monitoring and likely need treatment by the time he is 30."

Therefore biomedical "information overload" is really a data, information, and knowledge overload, since all are being produced constantly in large volume. The goal is to continue to make progress in medical science by identifying the right data,

information, and knowledge, and connecting them together to produce new knowledge. The problem is that as the knowledge base grows, it is more and more difficult for anyone to be aware of the complete set of information and it is therefore less likely that the right data, information, and knowledge will be recognized and combined into new knowledge. We are at the point where it is essentially impossible for anyone to keep up to data on all the knowledge in all fields of biomedicine [4].

Balas and Boren found that it takes an average of 15.6 years for research evidence to reach the level of 50% clinical practice. This is divided into a period of about 6.3 years for the research to make it into review articles and text books, and 9.3 years for the published recommendations to be put into practice [5]. These periods are at the tail end of the life cycle of medical discoveries. While it is unclear how long it takes new medical research discoveries to be disseminated among biomedical researchers, Altman and Goodman found that it takes between four and six years for new statistical techniques to be used and documented in the biomedical literature [6]. While these studies quantify the duration of just two steps in the process, the overall period between the making of a discovery and that discovery being put to use for the good of patients is much too long.

This has negative effects both at the individual and societal level. For individual doctors and patients, the lack of the ability to keep abreast of all new development relevant to their patients means that important information may not be available to them when decisions of care are made. This is in spite of the fact that the information may be “out there”, say spread over several articles referenced in MEDLINE. The information exists, it may even be known to a few specialists or researchers. It is just not accessible to



some of those that need it, perhaps because the physician has no way of knowing that the information applies to their patient [7].

At a higher societal level, without a comprehensive, “big picture” view of the current state of medical knowledge, it is likely that important connections will not be made in a timely manner, resulting in a delay in the understanding of disease mechanisms and the discovery of new treatments. Even in a field with thousands of researchers, the connection between available information that leads to new knowledge must be made within an individual. In this case, there is information available that a researcher or specialist could synthesize into new knowledge, but no one has yet recognized that this opportunity exists.

Medicine and biomedical research is divided into highly specialized fields and sub-fields, with poor communication between disciplines [8]. While this may be a necessary pre-condition for the complex and detailed research that biomedical science requires, it also tends to narrow the perspective, impeding the establishment of connections between discoveries arising within different research specialties. With the recent sequencing of the human genome, the addition of detailed genetic information to medical research makes the situation even more complicated, since genetics may play a part in almost all areas of health and disease and it is likely that many connections between different branches of medical may be based on related genomic mechanisms.

Clearly with the current rate of growth in published biomedical research, it becomes increasingly likely that important connections between individual elements of biomedical knowledge are not being recognized because there is no individual in a position to make the necessary connections. Methods must be established to aid physicians and researchers

in making better use of the existing published research and helping them to discover potential connections and turn these into new discoveries [9].

Text mining and knowledge extraction are ways to aid physicians and researchers in identifying the connections between data, information, and knowledge available in the biomedical knowledge base. A subset of full-blown natural language processing (NLP), which attempts to understand the meaning of text as a whole, text mining and knowledge extraction concentrate on solving a specific problem in a specific domain identified a priori. For example, literature searching may be improved by identifying all of the abbreviations used by researchers in journal articles [10], or potential new treatments for migraine may be determined by looking for pharmacological substances that are associated with biological processes associated with migraine [11, 12].

For the purpose of discussion, knowledge extraction can be used as a general term for the process of distilling knowledge from a collection of data. Text mining specifically applies knowledge extraction techniques to data in the form of narrative text. Srinivasan quotes Hearst when she states that “the key goal of mining whether from well structured databases or numeric data or from text collections is the discovery of new knowledge” and states that the goal for information extraction systems is to extract “nuggets of information from collections of texts” [9]. As applied to biomedical text, the goal is to draw out connections between terms or concepts that are present but unrecognized in medical documents and the medical literature. These terms and concepts may be coded from a controlled vocabulary, or may be present in free text fields. Several types of extracted knowledge have been identified: referential (e.g., lists of names of drugs),

attributive (e.g., attributes of a gene or set of genes), or relational (e.g., interactions between proteins) [13].

Hersh distinguishes between several kinds of medical data, textual (or narrative) versus coded (or structured), and patient-specific versus knowledge-based. The patient-specific class consists of information pertaining to a specific patient and includes structured information such as lab results and vital signs, and narrative information such as a physician's progress notes. Knowledge-based information pertains to the science of health and medicine and includes original research, review articles, books, practice guidelines, and structured summaries of this information such as bibliographic databases [1].

Both of these classes include textual data than can serve as a source for text mining as well as structured data that can be used as a source for information extraction. However, for the purposes of knowledge extraction, there are some significant differences between patient-specific and knowledge-based information. Knowledge-based information is usually intended for a wide audience and is highly reviewed and edited. Patient-specific information is usually not reviewed or edited, is read only by people who need to take care of the patient, may be brief and use non-standard abbreviations, and often contains errors. Structured, knowledge-based information is typically coded using a taxonomy intended for the purpose of codifying medical knowledge concepts. The MEDLINE database, for example, uses the National Library of Medicine's Medical Subject Heading (MESH) vocabulary [14]. Structured patient-specific information is often coded for purposes other than representing scientific knowledge, such as the ICD-9 and DRG codes used for billing. Interestingly, one important research area of text-mining is to provide an

automatic means of determining appropriate billing codes from discharge summaries and other patient-specific free text [1].

Therefore, knowledge-based information tends to be less noisy, have a lower rate of error, and a lower level of semantic ambiguity than patient-specific information. This often makes it simpler to apply knowledge extraction techniques to knowledge-based information, especially structured knowledge-based information such as the MeSH term fields of the MEDLINE database. In fact, a significant amount of knowledge extraction research in medical informatics does exactly that [9, 15]. However, some of the most important applications of knowledge extraction and text-mining, such as epidemic and bio-terrorism surveillance, may require patient-specific (and perhaps narrative as well) information to produce timely results [16].

## II. PROBLEM DEFINITION

### **Focus on Bibliographic Data**

This work will focus on the extraction of knowledge from the bibliographic database MEDLINE, a knowledge-based resource. Both narrative text, in the form of titles and abstracts and structured data in the form of MESH headings are available in the MEDLINE database. MEDLINE contains data from a wide array of medical, biological, and genomic specialty areas, and documents much of the accumulated knowledge in these areas back to the 1960s. While text mining and knowledge extraction from patient-centric data is an important area of research, the depth and breadth of biomedical knowledge represented in MEDLINE, and the quality of the data, make it a useful source on which to begin. Furthermore, focusing on MEDLINE has the very desirable quality of maximizing the potential of research already conducted (and paid for).

Use of the MEDLINE database is central to the work of biomedical researchers, who review the database for articles relevant to their work. Researchers use MEDLINE essentially as a searchable collection of records, in other words, the sum of its parts. Text mining and knowledge extraction can approach MEDLINE from a different direction - as a whole, where the value is in the implicit relationships between the concepts in the database. Since the knowledge is implicit, rather than explicit, it is important to develop robust, reliable, flexible methods to extract this knowledge into useful forms.

This additional knowledge can benefit the biomedical research community in several ways and have a potentially strong synergistic effect on the research community. Search and retrieval may be improved by uncovering related terms and concepts useful for searching, such as synonyms, or abbreviations [10, 17-20]. Indexing may be improved by

uncovering concepts not explicitly coded that should be [21]. Knowledge extraction may be used to group concepts in novel ways, or bring previously unrecognized concepts into a well-defined group [22-24]. Lastly, text mining and knowledge extraction may be used to uncover unrecognized functional relationships, stimulating new research in treatment or disease pathophysiology [12, 21, 25-27].

### **The Process of Knowledge Extraction**

The process of knowledge extraction can be represented as a system of interacting components. Each of these components represents a functional unit or set of data in a knowledge extraction system. It is important to note that all knowledge extraction systems may not include all types of components.

The basic components of a knowledge extraction system are (see Figure 1):

- Bounded Problem Domain
- Data Set
- Initial Knowledge
- Instance Knowledge
- Recognition Patterns
- Application of Patterns to the Data Set
- Extraction of Instance Knowledge
- Rating of Instance Confidence
- Recognition of New Patterns
- Rating of Pattern Utility
- Evaluation
- Gold Standard
- Results

In the figure, actions or processes are represented by white rectangles. Data or representations are shown using gray ovals. Each of these components will be described separately.

### *Bounded Problem Domain*

Any text mining or knowledge extraction task must be done in a well-defined manner. The definition of the subject area, the specific terms or concepts of interest, the significant relationships between the concepts, and the form of the data representing those concepts comprise the bounded problem domain [28, 29].

### *Data Set*

This is the set of data to be used as the source material for knowledge extraction. If a system evaluation is to be done, the data set may be divided into learning, testing, and validation subsets [30-32].

### *Initial Knowledge*

This is the expertise that the investigator brings to the knowledge extraction task. The initial knowledge may be used to produce an initial set of knowledge instances and/or recognition patterns to “seed” the knowledge extraction process [4, 33-35].

### *Instance Knowledge*

These are instances of the relationships of interest. The relationship may be between concepts, terms, or phrases and indicate specific relationships between two or more specific concepts or terms. In many cases the goal of knowledge extraction is to increase the number and quality of the set of instance knowledge [10, 25, 36, 37]. Note that instance knowledge may be obtained directly by application of recognition patterns to the data set, or indirectly based on logical analysis of pre-existing instance knowledge.

### *Recognition Patterns*

These are patterns or templates for recognizing instance knowledge in the data set. These patterns may be lexical, semantic, statistical, or take some other form [11, 20, 26,

27]. The patterns process the data set in pieces through a window that corresponds to the level of granularity required by the template. For example, MEDLINE records may be processed by record or field within a record, and text fields such as the abstract field may be processed by word, sentence, clause, paragraph, or as a whole [38].

#### *Application of Patterns to the Data Set*

This is the process of reviewing the data set for matches with the recognition patterns. A match between a unit of the data set and a recognition pattern will result in some evidence that may lead to instance knowledge. This is a deductive process: the application of the general to the specific [4, 39, 40].

#### *Extraction of Instance Knowledge*

This is the process of examining the evidence collected so far in the process to find instances of new knowledge. This component may essentially be a pass through of the results of the *Application of Patterns to the Data Set* component, or there may be significant additional processing done here to extract instance knowledge from collected evidence. For example, the process of following a chain of logical reasoning may be done by this component [26, 27, 37].

#### *Rating of Instance Confidence*

This is the process of evaluating the confidence or certainty of an instance of knowledge. This is often expressed as a percentage, with certainty being 100%. Initial knowledge instances may be treated as certain, while extracted instances will have a confidence between 0% and 100% [35, 39].



### *Recognition of New Patterns*

This is the process of recognizing new extraction patterns, based on the initial knowledge and gathered instance knowledge from the data set. These new extraction patterns can be used to extract new instances of knowledge and can be rated in terms of their utility. This is an inductive process: the creation of a general rule from specific instances [37, 40, 41].

### *Rating of Pattern Utility*

This is the process of evaluating the accuracy (specificity and sensitivity) of an extraction pattern or method. Given a set of patterns or methods, those with higher accuracy will produce better results than those with lower sensitivity and accuracy [10, 19, 40].

### *Evaluation*

This is the process of examining the results for correctness and errors, strengths and weaknesses, failures, and oddities of the system [42, 43].

### *Gold Standard*

This is a set of known correct and complete results to use as a comparison with the extracted knowledge during evaluation [44-46].

### *Results*

Results constitute the final useful output of the system intended to answer the initial knowledge extraction question. In other words, the purposeful knowledge extracted from the data set. These may include the accumulated instance knowledge, the discovered recognition patterns, and results of the evaluation, and the making of the system available for public use [10, 36].

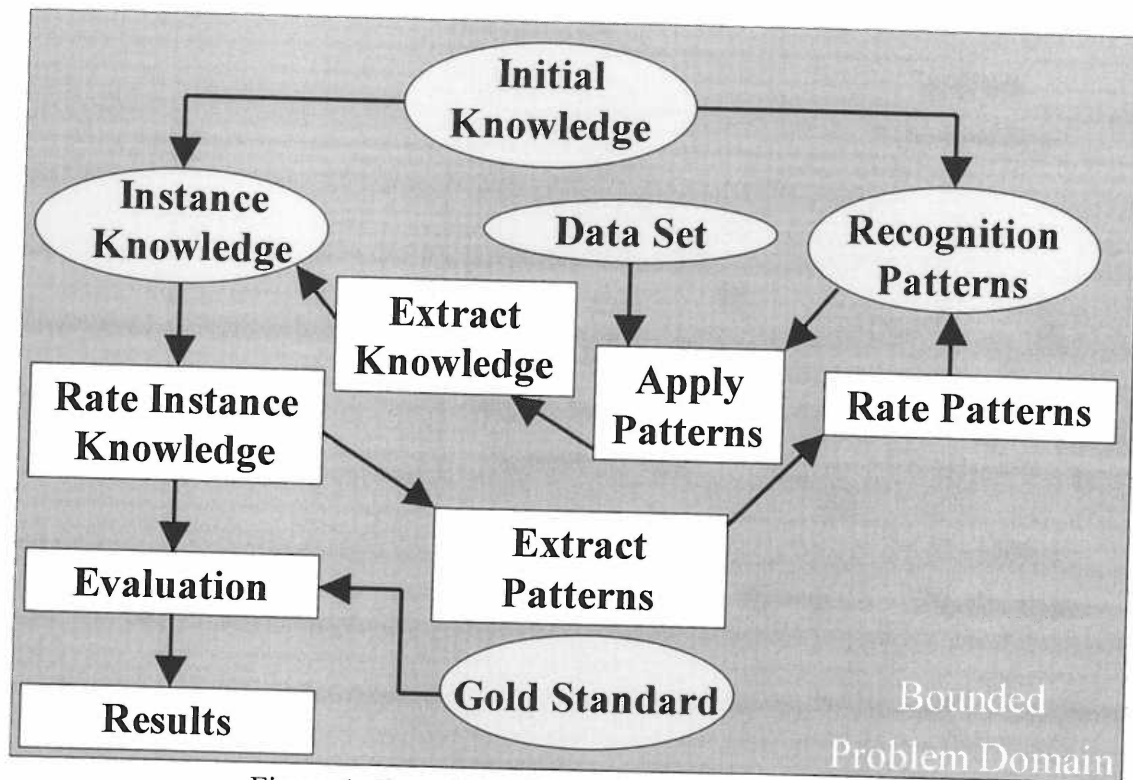


Figure 1: Knowledge Extraction Process Framework

### Process Framework for Knowledge Extraction

For the purpose of this thesis, these component descriptions will serve as a process framework and the operational definitions of the components of a knowledge extraction system. The literature review, theoretical framework, and the experimental design sections will be organized around these operational definitions and framework. Frequent reference to this diagram and the corresponding component definitions will be made.

### Purpose of Study

The purpose of this study is to determine whether mathematical graph and network analysis can be applied to the text mining of bibliographic data. *Instance knowledge* will be extracted from the fields of MEDLINE records using *recognition patterns*, and recognition patterns will be induced from the extracted instance knowledge. Graph and network analysis will be used to extract instance knowledge, and evaluate the quality of the instance knowledge and the quality of the recognition patterns. This process is called

Symbolic Network Logical Analysis (SNLA) because the accumulated instance knowledge is used to create a symbolic network of logical relationships, and then graph and network analysis is used to extract new knowledge and evaluate the confidence in that knowledge.

The mathematical analysis of graphs and networks for the purpose of text mining and knowledge extraction of the medical data has not been well studied, although viewing semantic relationships as a graph has been discussed in the medical informatics literature [11]. A few investigators have used the network structure of biomedical literature co-occurrences to visualize and uncover generic relationships between genes [47, 48]. Sociograms, the network representation of relationships between people, is a well known tool in social science qualitative research [49, 50]. Researchers in other domains have analyzed the graphical properties of relationships of interest to them, for example, the work of Newman, Strogatz, and Watts on the “small world phenomenon” of social networks [51-53] (for instance their analysis of MEDLINE co-authorship[54]), and Sigman and Cecchi’s work on the organizational function of antonymy (opposites), hypernymy/hyponymy (more generic/more specific), and meronymy/holonymy (is part of/contains) in the English language [55].

With graphical and network analysis having demonstrated usefulness in knowledge extraction for other fields, it seems likely that a similar approach can be used for the purpose of knowledge extraction and text mining on the biomedical literature as represented by the text and coded fields of the MEDLINE database. A well-defined and focused problem of practical value must be selected to test this assumption.

## Gene and Protein Name Synonymy as the Initial Problem Domain

The problem of gene and protein name synonymy has been selected as the initial area in which to develop the techniques of SNLA. Many genes and proteins have multiple names with several orthographic and lexical variants. Gene names are often not used consistently, and new names continue to be created [56, 57]. The many potential attributes of a gene may lead to it being given several names over time. This includes attributes such as its phenotypes and Mendelian inheritance, linkage to a marker, *pseudo-genes* (non-functional copies of genes), anti-sense encoding, and transcribed but untranslated segments. Genes may receive names that are retracted when new information becomes available [58]. Since the gene names and symbols used in a journal article in the biomedical literature are fixed once published, later correction of improper names does not affect the prior published literature. Therefore, the name space representing a gene can become quite large between the time a gene is first suspected and when it is well studied and has a universally agreed upon name. Additionally, since gene and protein names overlap and are often used in place of one another within the literature (with the intended gene or protein being dependent upon context), when conducting a literature review it is useful to search for both gene and protein names simultaneously [56].

An automatically generated list of synonyms would be a useful aid in searching the biomedical literature. The development of SNLA techniques to automatically extract synonyms from MEDLINE records will both demonstrate the practicality of SNLA and provide functional methods towards development of an automated service that provides a nearly complete list of gene name synonyms by scanning new MEDLINE records for

additional synonym information in real-time. This service could then be used to improve the research capabilities of genomics investigators trying to find all known information on a gene or protein, regardless of the name or names used in a specific article.

An automatically generated list of name synonyms would also be useful in further work on extracting genomics knowledge from textual sources and to other investigators investigating the properties of the biomedical literature [59]. To make efficient use of the available data when mining the biomedical literature for relationships, it is important to recognize differing names for identical concepts and treat these as a single concept. Within the genomic literature, this can be difficult since so many lexical strings are often used to represent a single gene. For example, the LocusLink database [46] lists at least 20 alias names for the gene PTEN (phosphatase and tensin homolog)<sup>1</sup>, and this does not include many simple orthographic punctuation variations currently in use, for example, MMAC1 [60] and MMAC-1 [61]. Two independent research groups studying the visualization of gene relationships (using a display of network structure based on literature co-occurrences) found that the main sources of error were “insufficient synonym lists, synonym case variation, and complex gene families with immature or complex or naming conventions” [48, 62]. All three of these sources of error can be reduced with a high quality, automatically generated list of name synonyms.

While databases of gene names and symbols exist, they have several limitations. Gene name databases such as FlyBase [63-65] and HUGO are restricted to a single species (fruit flies and humans, respectively). While the LocusLink database includes genes and names for several species, it does not attempt to include all names, symbols, and lexical

---

<sup>1</sup> PTEN, daf-18, T07A9.6, DPTEN, PTEN3, dPTEN, CG5671, CT17882, CT40892, CT40894, TEPI, MMAC1, BZS, MHAM, TEPI, PTEN1, Mmac, PTH2, PTEN2, and PTEN-rs as of September 11, 2003.

variations that refer to a gene. In this respect, the FlyBase and HUGO databases are similar to LocusLink. The HUGO database in particular was created by the Human Genome Organisation for the purpose of establishing an approved set of unique gene names and symbols for every gene in the human genome. Currently the HUGO database is incomplete; it contains approved symbols for 13,000 genes out of an expected total of about 30,000 [66, 67]. HUGO is focused on creating the set of gene names recommended for use in biomedical writing. It is not intended to be a complete collection of the gene names and symbols actually used in the biomedical literature. Actually the reverse is true: for genes that are already well studied, HUGO functions as a source of approved names for journal authors [68]. For new gene discoveries, HUGO serves as an approval body for the creation of new names [66].

While other investigators have examined the problem of gene name synonymy [19, 35], as of this writing, these research systems have not been put into general use, leaving open the opportunity for other approaches, such as SNLA, which may be more appropriate than these other approaches for online, real-time, knowledge extraction. For example, several prior approaches to extracting gene name synonyms from the medical literature rely on part-of-speech (POS) taggers that have been modified to recognize gene names as parts of speech [19, 43]. While tagger-based systems require that names be recognized independently of their participation in text that defines synonyms, this is by no means a required first step to detecting synonyms [35]. In fact, reliably recognizing biological entity names (such as gene and protein names), called *named entity extraction*, is in general thought to be a harder problem than synonym detection [56], implying that systems extracting synonyms using POS taggers may be doing more work than

necessary, and not taking full advantage of the available information because the additional context of the synonymy relationship may simplify the problem. Approaches based on concept relationships such as SNLA can efficiently take advantage of this additional context.

Another limitation to these systems is that the taggers must be trained on a manually annotated document corpus. Few annotated document sets are publicly available [44, 46, 69], and it is not clear that those that are available adequately represent the whole of biomedical literature. It is also unclear whether a POS tagger once trained will continue to perform to the level necessary over time. Clearly an approach that trains as it examines new data would be preferable. Since symbolic network graphs can be built and analyzed incrementally over time, SNLA can be used in this manner.

For the purpose of developing SNLA, gene name synonymy has the advantage of being a symmetric and transitive relationship, the simplest kind that can be represented in a network. Symmetric means that relationships go in both directions: if A is a synonym of B, then B is a synonym of A. Transitive means that logical inferences can be made about synonyms: if A is a synonym of B, and B is a synonym of C, then A is a synonym of C. This makes gene name synonymy a tractable problem to start with since the straightforward semantic representation of concepts and relationships allows the research to focus on the development of methods. Generalizations of these methods will then have further applications to text mining and knowledge extraction. For example, the extraction of new potential therapies based on the discovery of a chain of reasoning spread over several bibliographic entries. Swanson has termed this “complementary structures in disjoint literatures” (CSD) [26]. He proposed a simple “A influences B, and

B influences C, therefore A may influence C” model for detecting instances of CSD which is commonly referred to as *Swanson’s ABC model* [8, 27]. This is one exciting potential application of SNLA, since the network approach facilitates logical analysis over many semantic concepts simultaneously, and can also has the ability to represent and uncover relationships across many interconnections, providing for much greater potential than Swanson’s ABC model which has only two interconnections. While Swanson’s ABC model could be extended, to say ABCD model, this model still begins at a pre-defined concept and each new interconnection exponentially increases the work of the investigator [25, 27]. SNLA has the potential to analyze a large set of relationships simultaneously without pre-defining a starting place and with little or no additional burden on the investigator (although the computer has more to do).

### **Research Question**

In order to investigate the potential of symbolic network logical analysis as an approach to knowledge extraction from bio-medical textual sources, SNLA will be applied to the problem of automatically generating a list of gene and protein name synonyms. The research question to be studied is whether symbolic network logical analysis can be used as a tool in knowledge extraction, and specifically, can it be used to extract a high quality set of gene and protein name synonyms from MEDLINE database records?



### III. LITERATURE REVIEW

There is a vast amount of literature that pertains to text mining and knowledge extraction, with contributions coming from many fields including computer science, library science, the social sciences, and informatics. The prior work is so vast and so distributed across the scientific specialties that a complete review would be both impractical and impossible. Nevertheless, the literature that applies directly to the research question, both the study of SNLA methods, and the study design itself, is much more manageable. This literature review focuses on the prior work most directly related to proposed research question. Much of this literature originates in the field of biomedicine and medical informatics, but relevant articles from computer science and the social sciences are included as well.

The prior literature is organized according to two main themes: related high-level text mining and information extraction tasks, and lower level components of the knowledge extraction process pertinent to using and evaluating SNLA. The section on high-level tasks concentrates on knowledge extraction goals from biomedical text sources. The section on lower-level components reviews prior work relevant to components of the SNLA process and the evaluation of SNLA-based research.

#### **High level knowledge extraction tasks**

##### *Early work*

The idea of extracting knowledge from biomedical bibliographic and other text sources goes back at least 35 years, if not further. During the late 1960s, Zellig Harris did important theoretical work in describing the structural characteristics of language in a limited domain such as scientific or medical literature [28]. He termed these

*sublanguages*. Harris described how these structural characteristics could be determined and then used to help with knowledge extraction. Much of the current work in text mining, knowledge extraction from bibliographic databases, and natural language processing (NLP) in limited domains (such as medical specialty reports) has roots in Harris's ideas [29, 70].

Swanson was perhaps the first person to propose using text mining on the biomedical literature to uncover previously unrecognized relationships worthy of further investigation, an approach sometimes called *hypothesis generation*. Swanson developed a theory he called “complementary structures in disjoint literatures” (CSD) [26]. Swanson realized that large databases of large scientific literature would allow discoveries to be made by connecting concepts using logical inference. He proposed a simple “A influences B, and B influences C, therefore A may influence C” model for detecting instances of CSD which is commonly referred to as *Swanson's ABC model* [8, 27]. In this model, A is a pharmacologically active substance such as the drug thalidomide, B is a biological factor, and C is a disease or syndrome (see Figure 2).

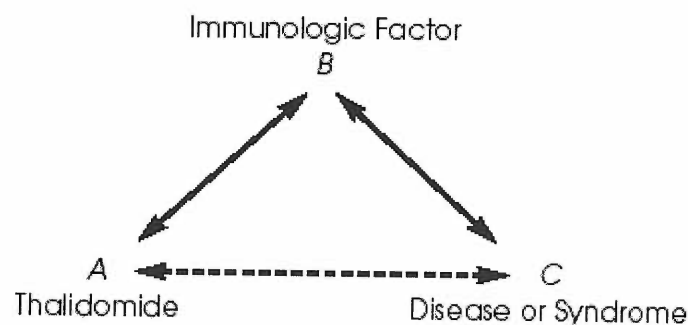


Figure 2: Swanson's ABC Model (from Weeber [27])

In several published papers in the 1980s and early 1990s Swanson gave examples of discovering new hypotheses by manually connecting concepts between journal articles. In 1986, he found a connection implying patient benefit between fish oil and Reynaud's syndrome, two years before clinical trials established that the benefit was real [12, 25]. In another article he traced 11 indirect connections between migraine and magnesium using summarizations of published articles. He used MEDLINE title word co-occurrence frequencies as a means of assisting the investigator to logically connect concepts related to both migraine and its possible therapies. He also suggested that the process could be improved by using MESH subheadings to narrow the document space [26].

Swanson predicted the future development of more highly automated systems to help researchers accomplish the task of uncovering CDB connections on a much larger scale. The ARROWSMITH system, built by Smalheiser and Swanson, is the most recent version of such a tool [71]. This and similar systems are still very much in the research phase [8, 15, 38, 72], and currently, there is little use of these systems by medical (as opposed to informatics) researchers [71]. While, in his writing, Swanson establishes a long history in science for discovery based on uncovering previously unrecognized connections, his ideas are still very much leading-edge, and are currently being actively explored [27].

Other early investigators investigated the use of the bibliographic indexes by themselves, rather than using the title, abstract, or journal article text, as raw material for knowledge extraction. In 1993, Cimino and Barnett first constructed a medical knowledge base from MEDLINE using statistical citation co-occurrence analysis [73]. Mendonca and Cimino extended this work to focus on improving information retrieval by

automatically identifying semantic relationships between co-occurring pairs of MESH terms. One study found that 7% to 8% of MESH term pairs to occur together significantly more frequently than others. The authors suggest that the extracted relationships could be useful in an automated system that helped physicians to perform more effective literature searches, for example by suggesting inclusion of “propranolol” in a search for information on therapies for heart failure [15]. In another study by these investigators, a group of physicians found 60% of the automatically extracted semantic pairs to be clinically relevant [74].

### *Current Work*

This section covers current tasks to which researchers are applying text mining to biomedical literature. This work is important not only because it covers the current state-of-the-art problems and techniques, but also because many of the tasks may be aided by an SNLA-based approach. Furthermore, the nature of biomedical text mining is such that more sophisticated tasks often must be built on top of solutions for other, more basic, text mining problems [57]. Therefore the opportunities here are two-fold: to use prior results as a basis for solving more sophisticated knowledge extraction problems and to identify areas where there are basic problems still needing to be solved.

de Bruijn and Martin have summarized the current state-of-the-art in medical text mining from a bioinformatics perspective. They give several examples of specific tasks (which they call goals) for text-mining in bio-informatics and medicine. They include tasks such as finding protein-protein and protein-gene interactions, finding sub-cellular localization of proteins, functional annotation of protein, vocabulary construction, and discovering gene functions and relations [72].

de Bruijn and Martin separate NLP research in clinical medicine from that of molecular biology by stating that the former focuses on the patient clinical record while the later focuses on scientific articles. However, this separation is artificial, and there are many counter examples, such as Swanson's well-known example of finding that fish oil may be a possible treatment for Raynaud's syndrome by examining the MEDLINE database. A more accurate classification is Hersh's separation between research focused on patient-specific information, and that focused on knowledge-based information. Text-mining in clinical medicine and molecular biology, as well as other medical and biological specialties, may use either of these information sources. Therefore, it is not the scientific specialty that characterizes the text-mining work as much as the source, quality, and reliability of the source data.

de Bruijn and Martin have also outlined the process of text mining into four stages: (1) document categorization; (2) named entity tagging; (3) fact and information extraction; and (4) collection-wide analysis. However, this four-stage process lacks a very important aspect of knowledge extraction: the feedback between discovered facts and rules or templates for extracting these facts. Indeed, de Bruijn and Martin allude to this in the context of the interaction between investigator creation of NLP models and the investigators' use of those models, but they neglect to point out that this feedback can exist within the text-mining system itself. This feedback loop forms an important part of applying SNLA.

de Bruijn and Martin also assume that named entity tagging is an essential independent step, when in fact biomedical relationship mining has been found to be somewhat independent of the quality of biological named entity tagging. If one assumes

that relationship extraction requires identification of three biomedical terms, two entities and one relationship, the performance of relationship extraction should be approximately equal to the cube of the performance of biological named entity tagging. While this assumption appears to be true for news article extraction, for biomedical applications the F-score performance rates of named entity tagging and relationship mining are about equal, 75-85% [56, 75]. It appears that the independence assumption does not hold for biological relations and it may be easier to extract concepts and the relationships between them all at once without a prior tagging step due to the increased local context that relationships provide. This work therefore uses the knowledge extraction framework previously described instead of the de Bruijn and Martin stages.

#### *Related Research*

Currently, there is much interest and ongoing research in the application of knowledge extraction and text mining to knowledge-based biomedical text. This section will focus on research that either has goals similar to that of the research question, or whose methods are related to or serve as inspiration for the present work. Many of these knowledge extraction tasks may be amenable to the techniques of SNLP.

The related research includes work on:

- Named entity recognition
- Word-sense disambiguation
- Curation
- Functional gene grouping
- Abbreviation and acronym extraction
- Name synonym extraction
- Relationship extraction

## Named entity recognition

At first glance, the task of named entity recognition (NER) appears straightforward. The goal is to identify, within a collection of text, all of the instances of a name for a specific type of thing. For example, identify all of the drug names within a collection of journal articles, or identify all of the gene names and symbols within a collection of MEDLINE abstracts. Hansich and others [59, 72] believed that solving this problem would allow more complex text mining tasks to be tackled.

It turns out that this task is anything but simple and straightforward. There does not exist a complete dictionary for most types of biological named entities, so a simple text-matching algorithm will not suffice. The same word or phrase can refer to a different thing depending upon context. Many biological entities have several names.

The named entity task is an easier task when applied to general news stories. Systems performing named entity extraction on news stories typically perform at an F-score over 90%, the scores for biological names are much lower, about 75-80%. As mentioned above, the independence assumption that seems to hold on news stories does not hold for biological relations. The biological relation extraction scores are about the same as that for news relations, about 75%, even though the named entity recognition scores are about 10% lower [56].

One possible explanation is that context provided by the biological relation and other terms provide stronger evidence for the location of biological names than they do for locations in news text. This would go along with Harris' sublanguage theories, and suggest that the sublanguage used in biology is more distinct from general English than is the language found in news stories. Context seems to be very important in biomedical

text mining, and because of that, NER requires contextual knowledge to achieve high accuracy. Since the contextual knowledge needed for recognizing some types of entities may be broad, NER can be a more complex task than relationship extraction because the definition of the relationship provides some very helpful context [56].

Perhaps because of the difficulty of the NER problem, several researchers have focused their efforts on it. While current work has been fairly successful, the complex sophisticated methods used provide further evidence that biological NER is not a simple problem. NER remains an open area of research and the approaches of several researchers will be reviewed here.

#### Proux

As their first step toward automatically detecting interactions between genes, Proux et al. constructed a software system to identify *Drosophila melanogaster* (the fruit fly) gene names in scientific text. *Drosophila* was chosen as a good test subject because the FlyBase database contains a list of 550 distinct gene names, as well as short abstracts of research papers stored in the “Phenotypic Information” field for each gene. As a test set, sentences were extracted from this field in the database that contained at least two known gene symbols, with the rationale being that interactions would more likely be found in these sentences.

Identification of these gene names was complicated largely by the observations that gene names do not follow construction rules (unlike Fukuda’s assumption about protein names [21]), and that authors do not use approved names consistently. Also, new genes are frequently discovered and added to the literature. Therefore, name dictionaries are



frequently out-of-date and their use is a limited solution to the identification of gene names in text.

Proux et al. found that fruit fly gene names come in three categories. First are names including special categories, such as mixed case, hyphens, dashes, slashes, or numbers (e.g., Hrp54, Lam-B1, Laer\mt). Second are names using lower case letters only and also being an English word (e.g., vamp, ogre, zip, zen). And third are names using only lower case letters and not an English word (e.g., ynd, zhr, wp, unr). Proux found that while about 50% of *Drosophila* gene names belong to the last two categories, most of those belong to the third category and only about 5.5% belong to the second category. Furthermore, within this second category, about one-fifth are English words “out of scope” in scientific writing and unlikely to be found in an abstract (e.g., ogre). This leaves about 4% of gene names being ambiguous with English words.

Proux et al.’s gene name identifier was built on a series of finite state machines organized as a part of speech (POS) tagger that included gene names as a type. General POS tagging, while not essential to the problem at hand, was included in preparation for the next step of recognizing gene interactions. The process has two levels, lexical and contextual. The lexical analysis has four steps: re-writing common domain specific expressions into clearer forms to aid the tagger, the POS tagger itself, error recovery using domain specific dictionaries that aid in recovering out of scope English words that are gene names, and suffix and prefix recognition. The second level, contextual analysis, looks at nearby words in order to distinguish ambiguous gene names from common English words. Other forms that tended to confuse the tagger, such as bibliographic references, were also removed at this step.

The system was trained and tested on 1200 sentences extracted from Flybase, and divided into a 450 sentence tuning set and a 750 sentence experimental set. Recall on the experimental set was 94.4%, and precision was 91.4%. It should be noted that these results are outliers and similar researchers consistently achieve only about 80% precision and recall [56]. This may be due to the smaller domain of the Flybase gene namespace. Most of the false positives were found to be words specific to biology but not found in the dictionaries used (e.g. ommatidial), and chromosome locations, which may be combinations of letters and numbers (e.g., 102Efc). False negatives included numerical expressions, ambiguous in-domain English words, biological vocabularies, proper nouns, and others. Furthermore, the authors defined gene names as a single lexeme, and so gene names of two or more words were not recognized by the system and not included in the test set [57].

### Hanisch

Hanisch et al. have addressed the problem of identifying known proteins in scientific text. Note that this problem is not simply solved by creation of a dictionary of protein names, since authors often do not use the exact same lexical forms when discussing the same proteins. Hanisch et al. uses a large dictionary of protein names and semantically classified words that tend to appear in context with protein names. The classifier words are used as “delimiter” and “specifier” tokens to increase the specificity of matching biomedical text to the dictionary by helping detect ambiguous and irrelevant synonyms. Hanisch et al. created a set of pattern and heuristic filter rules to make use of this information as well as lexical information about the protein name, such as only considering case-sensitivity when the name was a single word. The matches were scored

using a weighting of the various types of matching, with the optimal weight determined by using a supervised machine learning algorithm called robust linear programming (RLP) [59].

#### Aronson

Aronson, et al. created MetaMap, a program that maps biomedical text to terms in the UMLS Metathesaurus. MetaMap works by first parsing the text into noun phrases, and then generating a set of spelling, abbreviation, acronym, synonym, inflectional, and derivational variants for sub-phrases that are either single words, or those that occur in the SPECIALIST lexicon. These variants are candidate phrases, and they are mapped to the closest match in the UMLS Metathesaurus using an evaluation function that takes into account centrality, variation, coverage, and cohesiveness. The scores for the candidates are then combined for disjoint parts of the noun phrase to determine which UMLS Metathesaurus best matches the noun phrase. The score of the best match is reported as a final confidence value that can be used as threshold value for valid mappings [76].

#### Tanabe and Wilbur

Tanabe and Wilbur investigated and evaluated a method of automatically identifying gene names in MEDLINE abstracts called AbGene. They extended the Brill POS tagger [41, 77, 78] to include gene and protein names as a tag type and then trained their system on 7000 hand-tagged sentences from biomedical text. The system then applied manually generated post-processing rules based on lexical-statistical characteristics that helped further identify the context in which gene names are used and eliminate false positives and negatives.

Evaluation on a test corpus showed that they were able to identify 77% of the correct and 93% of the incorrect gene name training examples. Interestingly, the post-processing rules usually helped to improve recall, but often decreased precision. For example, at the highest lexical-statistic scoring level, recall improved from 0.417 just using the modified Brill tagger, to 0.667 using their full system, but precision dropped from 1.000 down to 0.857 [43].

#### Change, Schütze, and Altman

Change, Schütze, and Altman took a very different approach to biological NRE. Instead of basing their algorithm on tagging names as parts of speech (such as in AbGene), the GAPSCORE system [79] assigns a numeric score to each word within a sentence. Words with higher scores are more likely to be gene and protein names or symbols, and words with low scores are unlikely to represent genes or proteins. The GAPSCORE is computed by examining the appearance, morphology, and context of the word to be scored and applying a classifier trained on these features. The scores are then divided into bins representing excellent, good, or poor candidate predictions for gene names.

Interestingly, the GAPSCORE system uses the Brill tagger [41, 77, 78] as a pre-filter to remove any words that are not nouns, adjectives, participles, proper nouns or foreign words, so POS tagging is still an important part of the approach. POS tags are also used to extend single word partial names into multi-word names by including preceding nouns, adjectives, participles, and subsequent numerals.

After training on the Yapex gold standard [80], precision, recall, and F-score were computed at every GAPSCORE cut off for both the exact matches and ‘sloppy’ matches (defined as a true positive if any part of gene name is predicted correctly). As would be

expected, the system performed much better with sloppy matches (precision 73.5%, recall 81.4%, F-score 77.3%), than with exact matches (precision 58.5%, recall 50.1%, F-score 54.3%).

The utility of sloppy matches is unclear. For example, running the web-based GAPSCORE implementation [69] on this sentence from a MEDLINE abstract:

Based on its structure, chromosomal location, sequence homology and cytokine-like properties, mda-7 has now been renamed IL-24 and classified as a member of the expanding IL-10 cytokine gene family. [81]

produced a score of “good” or “excellent” for “mda-7”, “been renamed IL-24”, “expanding IL-10”, and “Based”. Clearly sloppy matching includes words that are not specifically related to the gene. In this example, the number of non-gene words retrieved is greater than the number of gene words retrieved.

The performance of GAPSCORE for exact match was significantly lower than those found by many investigators during the BioCreative conference [75], where the average F-score for gene and protein NER was approximately 85%. However, both the test set and evaluation procedure were different. As with many results in text mining and NLP, it is difficult to draw firm conclusions by comparing studies using separate data sets.

## Word-sense disambiguation

Another task thought to be fundamental in biomedical text mining is word-sense disambiguation. This is the problem of *polysemy*, a single word having more than one meaning. Unfortunately for text mining, this is a common occurrence in biology, medicine, and the English language. The Unified Medical Language System (UMLS) lexicon [82] has many terms that are associated with multiple concepts. A *capsule* can be an anatomical site or refer to a pill. *Potassium* can be a lab value or a medication. *Cold* can be a viral infection or a sensation of temperature [83]. Resolving these ambiguities

are important in many information retrieval and knowledge extraction applications. This is termed word-sense disambiguation (WSD). To researchers working on this problem, Weeber et al. have developed a biomedical word-sense disambiguation test collection containing 5000 unambiguous instances (in context) of 50 ambiguous UMLS text strings [84].

There have been three basic approaches applied to the WSD problem. The first uses a set of disambiguation rules handcrafted by domain experts. The second uses a supervised machine-learning algorithm that is trained on an annotated corpus. A basic problem with both of these approaches, common to many text-mining approaches, is the large amount of expert labor they require, and the fact that they cannot be applied to a new or broader domain without repeating much of the work. A third approach, based on the *one sense per discourse assumption*, has neither of these disadvantages.

#### Liu and Freidman

Liu and Freidman use the third approach, which requires no specific domain expertise. It is based on two important assumptions. First, the one sense per discourse assumption states that the sense of a term used in a unit of text (e.g., an abstract or journal article) is constant across that text unit. This assumption is largely correct for domain-specific terms used in medical literature, but is less likely to be valid for terms that have both general and medical meanings, such as cold and discharge. The second assumption is that conceptual relatives of a term will tend to be used in the same sense as the term itself. A conceptual relative of a term is another term that has an ontological relationship with the first term, such as is-A or synonymy. The approach then proceeds as follows. An automatically sense-tagged corpus is created by assigning a sense to ambiguous terms

based on the sense of unambiguous conceptually related terms that co-occur in the text unit (e.g. the same abstract). Then a system for sense tagging the ambiguous terms is created by training a supervised machine learning algorithm on the automatically sense tagged corpus. Liu and Friedman used the Naive Bayes' algorithm for this purpose [85].

Liu and Friedman applied their approach to ambiguous abbreviations in MEDLINE abstracts. They selected 35 ambiguous abbreviations as meeting the conditions of the experiment and used the UMLS MRREL relationships table to determine conceptually related terms. Overall precision was about 93% with recall about 47% [83].

## **Curation**

The goal of the *curation* problem is to determine whether a document should be included in a collection based on some particular criteria, usually based on whether the document discusses a given topic or range of topics. This is a specific type of document classification. The curation task is different from many other text mining and knowledge extraction tasks in that instead of a list of extracted facts, the end result is a group of decisions on whether each document in the set belongs to the collection or not. In many ways this is like a “diagnosis” of the document, does it have a certain characteristic (e.g. discuss a disease, or a medical finding) or not.

Yeh, Hirschman, and Morgan ran a text mining competition as part of the *Knowledge Discovery in Databases* (KDD) challenge cup. The task was a realistic curation problem to evaluate papers from the FlyBase data set and determine whether the paper should be curated based on whether it contains experimental evidence of gene products. The competing programs were to extract three items of information: a ranked list of papers in the order of probability for the need of curation, a yes/no decision on whether to curate

each paper, and, for each gene listed in each paper, a yes/no decision about whether a paper contained experimental evidence for a gene's RNA and polypeptide products.

A training set of 862 cleaned full text papers was provided, along with XML mark-up containing the gold standard results for each paper. Of these, 283 were judged by experts to require curation. The test set contained 213 papers. Papers were cleaned by converting non-plain text (superscripts, subscripts, italics, Greek letters) into a plain text convention.

The best performing entry used a set of manually constructed rules for recognizing patterns of interest that provided the required information. They focused on figure captions, which were found to be useful [86]. The F-score of the ranked list was 84%, with 78% overall yes/no per paper and 67% for gene products. Another well performing team looked for manually chosen "keywords" and computed the distance between keywords and gene names [87]. Two other well-performing teams used regular expressions to find patterns of words and then used a support vector machine (SVM) to classify the papers [30].

Problems and complications noted were many. Systems had difficulty in distinguishing between gene names and the names of gene products. FlyBase is intended to contain only products based on the wild type of fruit fly, but systems found genes and products in laboratory-induced mutations. Gene name synonyms were a significant problem, as names in the papers often did not match that in FlyBase, and many gene products have several names. Naming conventions are also often not followed, resulting in many variants. Furthermore a protein could have several forms as well as a more general name and it can be difficult to determine which is meant.



Other lessons learned include that full electronic versions of papers are often not easily available. Abstracts are readily available, but much of the information needed by the curators to make a decision was only in the full text and not in the abstract.

Processing HTML has challenges (it is difficult to separate presentation from structure; XML is much better in this respect), and getting to the full text can involve multiple download steps for each document. Most text processing systems handle plain text, but important information may be contained in typesetting conventions.

## **Functional gene grouping**

The problem of *functional gene grouping* is to collect genes into sets of biologically meaningful groups, based on their function. Some researchers have attempted to do this by mining the biomedical literature. Raychaudhuri and Altman have done much work in this area. They have used entropy measures to associate genes with gene ontology (GO) codes representing biological processes mentioned with the genes in literature abstracts [23]. The same group has done work in automatically determining whether a group of genes are functionally related by applying a measure termed *neighbor divergence per gene* to the gene names in the articles most similar to a given article [22, 24].

## **Abbreviation and acronym extraction**

A knowledge extraction task that has attracted much attention recently is abbreviation and acronym extraction for biomedical terms. The goal of this task is to extract a list of abbreviations and their definitions from a set of text documents. As medicine and genomics are full of specialized terms and corresponding abbreviations and acronyms, a list of these in a given domain could be very useful both for readers needing to know the

full form of an abbreviation, and authors wishing to use already existing common abbreviations and not invent unnecessary new ones.

The problem of acronym extraction is similar to synonym extraction problem of the current work in several ways. It is based on a specific type of relationship between terms, it is narrowly defined, and evaluation is relatively straightforward and does not require much domain expertise to identify the false positives and negatives to evaluate results. Both may also have problems of polysemy, i.e., one term mapping to several concepts.

Acronym extraction differs from the synonymy problem in that that relationship between the long form and the abbreviations is unidirectional and not symmetric as is the synonym relationship. Another difference is that at least one published test set, Medstract, exists for evaluating abbreviation and acronym extraction, currently none exists for synonym extraction. Medstract is a collection of MEDLINE abstracts manually annotated in the XML format described and available at [www.medstract.org](http://www.medstract.org) [58].

Many of these attributes make abbreviation and acronym extraction an attractive research problem. It is not surprising that several groups have published work in this area over the last three years. All of their approaches make use of the observation that terms and their definitions frequently occur next to each other in biomedical text, with one or the other being enclosed in parenthesis. Some of the most successful work in this area includes that done by Pustejovsky, Lui, Aronson and Friedman, Chang and Altman, Schwartz and Hearst, and Larkey.

#### Pustejovsky et al.

Pustejovsky et al.'s research centers on extracting relations in biomedical literature. Their system, Acromed, uses a set of semantic automata to model the relations of interest.

These semantic automata are designed to recognize sequences of tagged parts of speech that correspond to the relation of interest. The relationship type, such as “inhibit”, is part of the pattern recognized by the automata, and several automata are generated to recognize the nominal and verbal forms of the relationship (e.g, inhibits, is inhibited by). The Brill tagger is used [41, 77, 78], followed by the Porter stemmer [88] to preprocess the text and make the set of automata simpler. The automata are separated into five phases: noun chunking, noun phrases, coordinated noun and verbs, chunks with prepositional phrases, and finally, recognition of subordinate clauses by the semantic automata. The authors note that the use of general *anaphoric* terms (e.g., “it”) is relatively infrequent in MEDLINE abstracts, but the use of *sortal* anaphors (group types, e.g., “statins”) is prevalent. Their system includes ordered sentence frames to help resolve anaphora [89].

The researchers used their system to find the long forms of abbreviations in biomedical text. They achieved performance of 72% recall and 97% precision on the manually annotated MEDLINE records they created as part of the Medstract test collection. The authors noted that their system performed better on biomedical texts than did other systems for general text, however, no comparison was made with other biomedical abbreviation extraction systems [18].

#### Lui, Aronson, and Friedman

Lui and Friedman studied the use of *parenthetical expressions* in text mining [17, 90]. They note that terms associated with parenthetical terms are often related terms, such as synonyms or close semantic relations such as abbreviations and definitions. These occur more frequently together in a corpus than by chance. Several types of relationships are

identified: A(B), and B(A), where A is the short, abbreviated form and B is the long form; A (i.e., B), where A and B are synonyms in the given context, and hypernymy A (B) or A(e.g., B) where B is an instance of the more general A. Lui and Friedman's work is unique in looking at the co-occurrences in an automated way to acquire terminological knowledge. They note that this method is not suitable for recognizing expansions that only occur once in a text corpus, since frequency is an important filter. However, no hand crafted rules or patterns are required, and no manually annotated training set is needed.

They used the NLM's SPECIALIST lexicon to normalize words, and then collected statistics associating three letter abbreviations within parenthesis with the one-to-many word phrases preceding the parenthesis in the sentences contained in a set of MEDLINE abstracts. Frequency threshold parameters were set to determine which co-locations were most significant. An internal analysis showed that their system detected correct pairings 96.3% of the time on a gold standard test set. A total of 381,126 unique pairs were collected, where 308,339 were used in the abbreviation knowledge base and 72,787 were considered as other types of pairing. The extracted abbreviation knowledge base covered 38.3% of the pathology abbreviation list manually compiled by J. Berman [91].

#### Chang and Altman

Chang and Altman created an online dictionary of abbreviations used in MEDLINE based on aligning abbreviations with their full form in the text using a machine learning approach. They observed that the number of abbreviations grows at least as quickly as the number of abstracts, and that half of all abstracts contain abbreviations. Furthermore, less than half of abbreviations are formed from initial word letters, so in order to support the

many ways that investigators create abbreviations, they broadly defined abbreviations to include more than just acronyms.

They broke the task into four components: scanning the text for possible abbreviations, aligning the candidates to the preceding text, converting the abbreviation and text into a feature vector, and scoring the feature vector. To find a possible abbreviation, the lexical pattern of short space delimited spans of text inside parenthesis was used. The preceding  $3*N$  words in the same sentence were used for the prefix text window, where  $N$  was the number of letters in the abbreviation. They approached the alignment task as a form of the longest common substring (LCS) problem, and used a dynamic programming algorithm to maximize the matched number of letters between two strings, where the first string was the candidate abbreviation, and the second was the document text that came before the abbreviation. Vectors were computed based on a heuristic set of features. Logistic regression was on a training set to determine the best coefficients with which to score the feature vectors. The score of an abbreviation is then the maximum score of the possible prefix alignments. Evaluation of the system against the Medstract gold standard abbreviation annotated database [58] showed a maximum of 83% recall and 80% precision at the best score points for each. Error analysis showed that the most frequent error was due to the gold standard using synonymous words and phrases in the definitions [10]. Note that this was better recall than that achieved by Pustejovsky, at the expense of lower precision.

#### Schwartz and Hearst

Schwartz and Hearst created a simple algorithm for identifying long forms of abbreviations in biomedical text that performs as well as much more complex

approaches. They use a simple two-stage process [92]. The first stage identifies abbreviation long form/short form pairs by looking for short text sequences in or next to parentheses. These candidate pairs are then filtered by heuristic lexical criteria based on the length and number of words in the short and long forms. The second stage identifies the long forms by performing a greedy match of letters in the short form with letters in the long form, starting at the end of each, with the additional constraint that the first letter in the abbreviation must match to the first letter of a word in the long form. This is based on the observation that abbreviations rarely begin with an internal character of the first word. Evaluation on both a MEDLINE subset and the Medstrat gold standard evaluation corpus [93] showed precision (96%) and recall (82%) approximately the same or better than that reported by other investigators with much more complex systems, such as Chang and Yu [10] and the Brandeis ACROMED system [18].

Larkey et al.

Larkey et al. have built Acrophile, a web-based acronym and abbreviation server and database populated by an automatic extraction algorithm [36]. Their system scans web pages and is not specifically for biomedical text, but their approach is related to the other work already described. They developed and compared four relatively simple algorithms to automatically extract abbreviations and acronyms from web pages: contextual, canonical, canonical/contextual, and simple canonical. Simple canonical was the most basic, finding only abbreviation pairs fitting well-defined forms, such as “expansion (ACRONYM)”. The contextual algorithm looked for an expansion of an abbreviation in nearby text. The other two algorithms combined features of these two. A simple regular-

expression definition of abbreviations was used, such as all uppercase letters, or uppercase separated by periods.

Evaluation used precision and recall on a given set of abbreviations in a randomly chosen set of 170 web pages. For abbreviations of length greater than 3, the simple algorithm was found to have the highest precision (99%), but the lowest recall (57%). The hybrid canonical/contextual had a precision of 92% with the highest recall of 84%. The investigators graphed precision and recall was graphed for all four algorithms and selected the canonical/contextual algorithm as the best visually.

## **Name synonym extraction**

The idea of name synonym extraction is simply to extract sets of synonymous names for a given concept. Genomics is an interesting area in which to apply synonym extraction systems because genes and proteins commonly have many names, in both short and long forms. There is significant prior work in this area, most of it done over the last five years by Yu and Agichtein [19, 35].

This is the research task to which SNLA has been initially applied. While the work of Yu and Agichtein is significantly different from that of the present work, the variety of methods they use shows that the gene name synonym extraction task is open to many approaches and therefore it is a good task on which to develop the methods of SNLA.

### Yu and Agichtein

Yu first worked on gene name synonym extraction with a system that extracted gene name synonyms based on manually identifying patterns in which gene name synonyms commonly occur. These are common patterns that authors use to list synonyms and include identifying phrases such as “dr4, also known as trail-r1”, parenthetical phrases

such as “dr3 (also called ws-1, tramp, apo-3, or lard)”, and slash separated synonym lists such as “p21/Cip1/Sdi1”. Domain experts were used to identify common patterns. Yu et al. estimated the precision of their system to be approximately 71%. Recall measurements were not published [35].

Yu and Agichtein worked together to combine several previously investigated text-mining approaches to the problem of gene and protein name synonym extraction. Their goal was to combine approaches and improve both precision and recall [19]. They implemented and tested four systems using complementary approaches. Three of the four approaches required pre-processing the corpus to tag the gene and protein names with type information. For this they chose Tanabe and Wilber’s Abgene tagger, previously described [43].

The first system, called *Similarity*, detected synonyms using a contextual similarity metric based on words appearing nearby within a fixed sized window. Confidence of name pairs was simply the similarity measure between the contexts of the two names.

The second system, entitled *Snowball*, was based on Brin’s Dual Iterative Pattern Expansion (DIPRE) system for the Web [39], which Agichtein adapted for extracting relationships from large text collections [37]. A small set of initially known facts is used to find the patterns in which these facts occurred within a large corpus. Then these patterns were used to extract more facts, which in turn were used to find more patterns. In this way the set of extracted facts grows like a “snowball rolling down a hill”. Four crucial steps in this approach were: 1) matching of patterns to the text, 2) determination of confidence in the extracted facts, 3) determination of confidence in the extracted



patterns, and 4) a means of generating patterns from the context in which the high confidence facts occur.

Snowball stores patterns as tuples of 5 values,  $\langle l_p, t_1, m_p, t_2, r_p \rangle$ , where  $l_p$  is the vector of term frequencies in the string before the first tag,  $t_1$  is the tag value (type) of the first term in the potential fact,  $m_p$  is the vector of term frequencies in the string between the tags,  $t_2$  is the tag value (type) of the second term in the potential fact, and  $r_p$  is the vector of term frequencies in the string after the second tag. A similarity metric is used to measure the degree that strings match with a pattern. It divides the context into left, middle, and right vector and weights each term in each vector based on the frequency of appearance. Then the degree of match between a pattern and a string of text are computed as the dot product of the vectors if the tags match, and zero if they do not.

Snowball uses a simple confidence measure that gives high confidence to facts extracted in the current iteration that match facts extracted in previous generations. Then the confidence in a pattern is computed as the log of the fraction of the positive facts divided by a weighted sum of the known positive facts, the known negative facts, and the unknown facts. The weights are set empirically during system tuning. For rating the confidence in the extracted facts, Snowball uses a measure based on the number of patterns that generated the extracted fact. Essentially the confidence is 1 minus the probability that all of the patterns that generated the fact were incorrect. Snowball discards facts with low confidence on each generation, since low confidence facts could add noise to the patterns generator.

While the iterative fact extraction/pattern finding approach is a clever and powerful approach, the Snowball method has several limitations. First, confidence is based

primarily on having seen a fact in an earlier iteration. However, if a single iteration has several new patterns discovering the same fact, or a single pattern finds the same fact in several places, this should logically increase the confidence in the pattern. In Snowball it does not. Also, facts discovered earlier in the process have a propagating and therefore stronger influence than those discovered later. Except for the initially provided facts, which are assumed to actually be true, it is not clear that this should be the case. Furthermore, Snowball can only find synonyms (facts) that are explicitly stated in a single sentence of the text. There is no means of inferring facts, even if there is strong evidence of synonymy across several articles in the corpus. This may explain why the Snowball method has low recall at high confidence scores ( $<10\%$  at confidence  $> 0.40$ ) [19], and why the researchers chose to combine it with other methods.

*SVM*, the third method that Yu and Agichtein investigated, used a text classification tool, *SVMLight*, trained on an initial set of user-provided positive and negative examples of gene name pairs and the surrounding text context. After training the classifier was run over the text corpus, this generated a confidence score for every text context of gene name pairs. When a pair appeared in more than one context, *SVM* assigned the confidence of the highest context for the pair.

The forth system that Yu and Agichtein included was “GPE”, a template-based system with hand-coded rules specifically for extracting gene and protein name synonym pairs they had previously investigated [35]. A domain expert examined a sample of journal articles and manually created patterns that authors commonly used to list synonyms. GPE did not use gene or protein name taggers and initially generated pairs of strings that were not gene or protein names. A set of heuristics was used to filter these

out. The system had no way to compare confidence in different synonym pairs and so extracted pairs were assigned a confidence of 1. This system does not make use of the increased confidence gained by extracting the same synonym pair from different abstracts or with different templates.

Yu and Agichtein combined the four approaches in a system called *Combined*. It combines the confidence of the pairs of the four systems in a way similar to how Snowball combines the confidence in individual patterns generating a synonym pair, that is, it computes the confidence as one minus the probability that all of the other systems are incorrect (which is one minus the confidence). This approach has the advantage of incorporating approaches that did not detect the synonym. These would have a confidence of zero. In their evaluation, Yu and Agichtein found that the Combined approach worked best, having higher recall at higher confidence levels. However, precision of Combined was less than that of both Snowball and SVM at confidence levels greater than about 0.60. Also, the time required of the combined approach, 11.5 hours to run over 32,000 journal articles, was the most of any of the systems, being the sum of the time taken by the system plus the time taken by tagging, although the computation of the four approaches, being independent, could be done in parallel on separate machines.

While the Combined system produces the highest recall of 0.80 with a precision of about 0.09, these results leave much room for improvement. The high recall comes with very low precision. Even though the Combined system allows the subcomponents to compensate for each other somewhat and improve precision over any single component, the overall precision is still very low.

Furthermore, the Combined system is a “kitchen sink” approach that combines the confidence levels from four different systems into one. This both requires a great deal of processing and makes the manner in which the system makes decisions somewhat opaque. It is difficult to see how changes to the system would lead to improved results since there is no intuitively understandable knowledge model in the Combined system. The effect of changing any one of the subsystems may make no change on the final result. It is not clear what features of the text are important in identifying a synonym pair, and it is not easy to see what text features constitute evidence of a synonym pair in the final Combined system. Nevertheless, the Combined approach does integrate the detection of several kinds of text features (pattern, statistical, co-occurrence) that appear to be important in identifying synonym pairs.

## **Relationship extraction**

The basic idea of *relationship extraction* is to identify evidence for relationships between concepts in a text corpus. While acronym and synonym extraction can also be thought of as a simple type of relationship extraction, research in biomedical relationship extraction focuses on identifying a range of relationships from a single text corpus, and the relationship types may or may not be defined in advance. Current research into relationship extraction usually falls into one of three types. Some relationship extraction systems simply attempt to identify pairs of concepts are related in some unspecified by interesting way, and leaves the determination of that relationship up to the investigator [74]. Other work specifies a relationship between classes of objects, such as proteins and cellular locations, and attempts to identify evidence of specific instances of that relationship [94]. Still other work specifies both the object classes and the types of

relationships, and looks for any combination of two object classes and a relationship type. For example, the object classes may be *drugs*, *cellular mechanisms*, and *diseases*, and the relationship type may be *inhibits*. The relationship extraction system would look for any combination of the object classes and the inhibit relation, such as drugs inhibiting cellular mechanisms, or diseases inhibiting cellular mechanisms, or drugs inhibiting other drugs, etc [89].

### Craven and Kumlien

Craven and Kumlien have explored the possibility of extracting structured information from MEDLINE abstracts for inclusion in a database to allow arbitrarily complex relational queries and support automated structured summarization [94]. They define the task as information extraction (IE) rather than natural language understanding, defining IE as “a limited form of natural language processing in which the system tries to only extract predefined classes of facts from the text.”

Their system takes a set of classes and relations among these classes, and is therefore an example of the second type of relationship extraction. The system uses a supervised learning approach, taking each sentence containing a pair of classes in a given relation and running a Naive B yesian classifier with the bag-of-words for that sentence on a manually annotated set of training data. The bag-of-words approach treats the words in a text unit, for example a sentence, as a set, ignoring the position of the words in the text.

Some word preprocessing is performed using the Porter stemmer [88], and the conditional probabilities were computed using Laplace estimates [95], which makes the estimates robust with infrequently encountered words. Essentially the system learns the conditional probabilities for a sentence belonging in a class on the training data and then

classifies the test data based on the posterior probabilities computed by using the conditional probabilities.

For their initial implementation, the classes were: protein, subcellular-structure, cell type, tissue, disease, and pharmacologic-agent. The relations were: subcellular-localization(protein, subcellular-structure), tissue-localization(protein, tissue), cell-localization(protein, cell-type), associated-diseases(protein, disease), and drug-interactions( protein, pharmacologic-agent). Confidence in a relation detected multiple times was computed using the *noisy or function* [96] which computes confidence as one minus the probability that all of the detected instances are incorrect.

The system was tested by looking in MEDLINE for subcellular-localization relationships documented in the Yeast Protein Database (YPD). They found a set of 20 words that contributed most highly to a sentence being classified to the subcellular-localization relation, the highest being “local” with a log-odds ratio of 0.00571 (note that this is an odds ratio of just 1.00573). Precision on the YPD test set was 77%, with a recall of 30%.

#### Pustejovsky et al.

Pustejovsky et al. used their Acromed system to find inhibit relations in MEDLINE records [58]. This is also an example of the second type of relationship extraction. They evaluated their system on 56 abstracts that were distinct from the training and evaluation set extracted from the Medstract test collection that included mark-up for inhibit relations, and were used as a gold standard for evaluation. Results showed a precision of 90.4% and recall of 58.9% in finding inhibit relations in the test set [89].

### Swanson and related work

Swanson's ABC model, with the relation *influences*, falls into the third, and most general type of relationship extraction. In several published papers in the 1980s, Swanson gave examples of manually discovering new inferences by connecting concepts between journal articles. In 1986, he found a connection implying patient benefit between fish oil and Raynaud's syndrome, two years before clinical trials established that the benefit was real [12, 25]. In another article, he traced 11 indirect connections between migraine and magnesium using summarizations of published articles. He used MEDLINE title word co-occurrence frequencies as a means of assisting the investigator to logically connect concepts related to both migraine and its possible therapies. He also suggested that the process could be improved by using MeSH subheadings to narrow the document space [26].

Swanson predicted the future development of more highly automated systems to help researchers accomplish the task of uncovering CDB connections on a much larger scale. The ARROWSMITH system, built by Smalheiser and Swanson, is the most recent version of such a tool [71]. This and similar systems are still very much in the research phase [8, 15, 38, 72], and currently there is little use of these systems by medical (as opposed to informatics) researchers [71]. While Swanson establishes a long history in science for discovery based on uncovering previously unrecognized connections, this is still not done routinely on an automated basis.

Even though Swanson began his work in the 1980s, his ideas are still being actively explored. In fact, other investigators working on relationship extraction systems have

tested their approaches by attempting to simulate Swanson results for Raynaud's and migraine headache [27].

Following on the work of Swanson, Lindsay and Gordon attempted to discover hidden connections in the medical literature based on lexical statistics commonly used in information retrieval [11]. Lindsay and Gordon state that it is an "open and heavily debated question whether a computational system will ever be able to fully master the use and comprehension of a natural language", and that current systems must examine text at the level of phrases and words. They investigate whether the "most tractable methods, that is systems based solely on lexical statistics, can be useful in uncovering implicit connections in the literature."

Lindsay and Gordon were able to replicate Swanson's migraine/magnesium findings by lexical analyzing text broken in one, two, and three word phrases, with token frequency, document frequency, relative frequency (in the migraine literature as compared to MEDLINE as a whole), and the TF\*IDF [1] product being computed for each token. The idea here is that two concepts appearing frequently together in documents are likely to influence each other in some way. In contrast to Swanson who used only titles and MESH terms, their approach uses the complete MEDLINE record.

Lindsay and Gordon first identify the relevant primary literature, for example, literature containing the word "migraine." Then they analyze this literature for tokens that occur frequently across the primary literature. These tokens are used to collect a set of records termed the intermediate literature. This intermediate literature is then examined in a similar manner, looking for tokens of high frequency that were not identified in the



primary literature. These tokens are then areas for exploration of concepts implicitly related to the primary topic.

They found that most (10 of 12) of the important intermediate topics identified by Swanson were determined by lexical statistics. Lindsay and Gordon also identified 16 biological substances with possible implicit associations to migraine for further investigation. Their work shows that useful information can be extracted from the medical literature based solely on lexical statistics. They note that more efficient use of the lexical information could be obtained by combining synonyms and lexical variants, and they suggested the Unified Medical Language System (UMLS) Metathesaurus for this purpose.

Weeber et al. have investigated the use of automated systems using NLP techniques, applying Swanson's ABC model to scientific bibliography in the DAD-system [8], using UMLS Metathesaurus concepts instead of simple words as the analysis units. Their system extracts biomedical concepts using NLP techniques and assigns them to categories with the aid of the Metathesaurus. Then, Swanson's ABC model is applied in reverse, first looking for concepts associated in the MEDLINE literature with C (e.g., a disease), and then sites of action B that are associated with disease C. Substances A associated with site B are then found to generate potential hypotheses for substance A acting on disease C. The DAD-system then assists with a manual review of the literature connecting A and C by presenting the relevant complementary sentences side by side. In one investigation, concepts having the categories immunologic factor and disease/syndrome were examined by the system for relationships to each other and to the drug thalidomide. By applying their system to PubMed titles and abstracts they were able

to recreate Swanson's finding of the relationship between fish oil and Raynaud's disease [8] and discover new potential uses for thalidomide in the treatment of chronic hepatitis C, myasthenia gravis, H. pylori-induced gastritis, and acute pancreatitis [27].

#### *Other work*

There is additional ongoing research using biomedical text that is not directly related to the present work but is interesting and warrants mention. The goal of much of this work centers on summarization or paraphrasing of knowledge-based text such as journal articles or patient-centered narrative such as exam reports. This work is different from the work already discussed in that the attempt is to understand the meaning of the text as a whole instead of extracting evidence of the truth of a set of relationships. This work leans heavily on traditional natural language processing (NLP) techniques.

Kittredge has reviewed the current NLP work on paraphrasing, for purposes such as automatic journal abstraction. Current research structures the problem into three stages: content planning, where the important content to be summarized is selected and given an outline structure; sentence planning where the outline is converted to a plan of paragraphs, sentences, and what they should contain; and realization, generation of the output text. Ongoing empirical study of the process that writers use to condense text into abstracts has led to developments in rhetorical structure theory (RST) which postulates about two dozen relations between clauses in text, and assists in constructing a tree structure representing the text.

Kittredge developed a set of condensation process types that can be used in a specific domain to create transformation rules for the analyzed text. These types include: local substitutions, sublanguage dependent structural operations, anaphora removal, and

semantic aggregation. Along with domain knowledge of what information is important, a summary can be created [97].

Perhaps the most successful of the biomedical text summarization systems is MedLEE, created by Friedman et al. at Columbia University. MedLEE is a general system for medical NLP that has been adapted for work in several areas of medical radiology, including chest radiograph and mammography reports, and discharge summaries [33, 34]. They have also incorporated MedLEE into a system called GENIES that automatically extracts molecular pathways from journal articles. They approach the problem by using MedLEE to assign semantic classes with actions, processes and other relationships that are identified in the articles by the words used and the phrase structure. Identified phrases that are well formed with respect to pre-defined patterns are output as bio-molecular pathways. Evaluation of the system showed that it identified 57 binary relations in the document corpus, 55 of which were judged correct by a domain expert [4].

## Components of the knowledge extraction process

Just as important as reviewing the high level tasks and results of prior biomedical knowledge extraction is the examination of some of the components of that work in more detail. The knowledge extraction process previously described defines several components that are important to SNLA, and it is interesting and useful to understand the prior work in terms of a focus on these components. This section will review prior work relevant to the knowledge extraction components in the present research on SNLA in the areas of:

- Prior work on the use of initial instance data as seed data
- Prior work on the manual and automatic generation of extraction patterns
- Prior work on the use of graph and network analysis for knowledge extraction
- Prior work on evaluation of results of medical knowledge extraction systems

### *Prior work on the use of initial instance data as seed data*

Sergey Brin was one of the first to have the idea to use initial instance data to discover patterns for knowledge extraction. Brin's Dual Iterative Pattern Expansion (DIPRE) system was initially targeted at Web pages, and was used to extract relationships between authors and titles [39]. The basic idea is that a small set of initially known facts (the initial instance data) is used to find the patterns in which these facts occur from a large corpus. The initially known facts may also include negative facts, in other words, such as two gene names are not synonyms. Then these patterns are used to extract more facts, which in turn are used to find more patterns. Only extracted facts with high confidence are used to find patterns. Obviously, the stability of the system is highly dependent upon the quality of the initial instance data, as well as a reliable means of determining which discovered instances are most reliable.

*Prior work on the manual and automatic generation of extraction patterns*

Patterns, or templates that are matched against the text corpus are basic to text mining and knowledge extraction. There are basically two kinds of templates: manually authored, and automatically generated. The manually authored templates often can be very reliable and lead to good results [35], unfortunately, they require a large amount of both time and domain expertise to author [90]. Manually authored templates are also incapable of adaptation; they must be re-authored if there is a significant change to the syntax or semantics of the source text.

The SGPE system by Yu et al. is an example of the use of manually authored templates. The investigators used MEDLINE abstracts to detect gene and protein name synonyms and extracted gene names from MEDLINE abstracts and full articles by looking for common patterns that authors use to list synonyms. These include identifying phrases such as “dr4, also known as trail-r1”, parenthetical phrases such as “dr3 (also called ws-1, tramp, apo-3, or lard)”, and slash separated synonym lists such as “p21/Cip1/Sdi1”. Domain experts were used to determine common patterns. The investigators estimated the precision of their system to be approximately 71% [35]. Recall was not measured.

Automatically generated templates do not require nearly as much time or domain expertise, but the system can more easily be misled by unexpected lexical characteristics and it may be difficult to distinguish between good and bad templates. Therefore, systems using automatically generated templates must have good measures of confidence for both the discovered instance knowledge and patterns.

The Snowball system developed by Agichtein is an example of an automatic pattern generation approach. Snowball uses a simple confidence measure that gives high confidence to facts extracted in the current iteration that match facts extracted in previous generations. Then the confidence in a pattern is computed as the log of the fraction of the positive facts divided by a weighted sum of the known positive facts, the known negative facts, and the unknown facts. The weights are set empirically during system tuning. For rating the confidence in the extracted facts, Snowball uses a measure based on the number of patterns that generated the extracted fact. Essentially the confidence is one minus the probability that all of the patterns that generated the fact were incorrect. Snowball discards facts with low confidence on each generation, since low confidence facts could add noise to the patterns generator [37, 40].

There are some potential problems with this approach. First, confidence is based primarily on having seen a fact in an earlier iteration. However, if a single iteration has several new patterns discovering the same fact, or a single pattern finds the same fact in several places, this should logically increase the confidence in the pattern. In Snowball it does not. Also, facts discovered earlier in the process have a propagating and therefore stronger influence than those discovered later. Except for the initially provided facts, which are known to actually be true, it is not clear that this should be the case. Furthermore, Snowball can only find synonyms (facts) that are explicitly stated in a single sentence of the text. There is no means of inferring facts, even if there is strong evidence of synonymy across several articles in the corpus. This may explain why the Snowball method has low recall at high confidence scores (<10% at confidence > 0.40)

[19], and why the researcher extracting gene names synonyms chose to combine it with other methods [19].

*Prior work on the use of graph and network analysis for knowledge extraction*

Many researchers have mentioned the idea that relationships among elements of the biomedical literature can be interpreted as a graph or network. However, most of the time the authors simply mention that relationships can be visualized this way and then do not pursue the idea. Lindsay and Gordon mention that textual information can be thought of as a graph where nodes are individual documents and the links are implicit connections between documents, for example by citation, but they do not use this idea in their work [11].

Almost no literature exists that analyzes the biomedical literature by constructing a graph or network based on co-occurrences across articles and then uses that network as an analysis tool for text mining and knowledge extraction. One notable exception is the work of Jenssen et al. This group has created a gene relationship visualization tool called PubGene, which uses an approach based on MEDLINE title and abstract co-occurrences. Their tool allows displaying co-occurrences relationships graphically and displaying relative expression values using color-coding. They demonstrated that the tool shows biologically meaningful relationships by comparing the relationships found in the co-occurrence network to that of published gene micro-array data. They also discuss the use of the tool to visualize a related set of genes weighted by functional activity from prior knowledge [48]. Jenssen et al.'s work is similar to Stapley and Benoit, but uses the entire MEDLINE database rather than focusing exclusively on *Saccharomyces cerevisiae* genes [62].

Jenssen et al., and Stapley, and Benoit's work are essentially graph-based visualization tools. They do not attempt to analyze the literature based on the mathematical properties of the extracted network structure. Wren and Garner have taken network analysis one step further, using a measure of node "cohesiveness" to uncover related groups of genes within those found to share specific GO codes [47]. While their work does use network metrics as a tool in knowledge extraction, the extracted relationship is very generic and non-specific. In this way the present work on SNLA demonstrating the extraction of a specific conceptual relationship using network metrics appears unique.

*Prior work on evaluation of results of medical knowledge extraction systems*

Evaluation is an important part of any approach or system of knowledge extraction. Without evaluation it is impossible to get an idea of the quality of results of an individual system and multiple approaches cannot be compared. Unfortunately, evaluation and especially comparison of systems is largely dependent upon a standardized test collection or an agreed set of expert evaluators that will apply a set of criteria uniformly across the results of various systems [42].

The use of precision and recall is often recommended and adapted for use in text mining tasks that result in a collection of proposed facts or relationships. These are dependent on having available an appropriate gold standard or a set of experts to judge relevance. While several researchers have questioned the real-world significance of measures such as precision and recall, these measures are the ones most often used, are the current state-of-the-art, and usually nothing better is available [1, 98].



Unfortunately for many biomedical text-mining tasks, a gold standard test set or set of expert evaluators is often not available. Several authors have called for development of standardized biomedical test sets, and at least two groups are working on annotation mark up to create genomics test collections: the GENIA corpus [44] and the Medstract project [58]. These test collections are very helpful, and have been used in the evaluation of several systems [10, 18, 89, 92], however they can only be used to evaluate results for problems for which they have been annotated. For example, the Medstract project supports evaluation of systems extracting abbreviations and acronyms, but, at least not currently, does not support evaluation of systems extracting gene name synonyms. Annotation is an expertise and labor-intensive process and it is unlikely that annotated text collections will be able to keep up with the new research tasks that knowledge extraction researchers are investigating.

Because of this problem, many evaluations are dependent upon the knowledge of locally available expertise. Additionally, for work that produces a large number of individual postulated facts as results, random sub-sampling is often done to reduce the workload of the experts. One example of this type of evaluation is that performed by Yu and Agichtein to evaluate their five synonym extraction systems. Yu and Agichtein measured precision and recall, however they had to estimate these quantities since they could not manually verify the truth of every extracted synonym pair, and they did not have a complete list of all synonym pairs described in the corpus. Therefore, to estimate precision they randomly selected 20 extracted pairs at each 0.1 increment of confidence for each system, and had experts manually determine the correctness of the synonym pair. Then they combined these results into average precision at each confidence score.

To estimate recall, they used SWISSPROT as a base and designated the gold standard list of synonyms as every synonym pair in SWISSPROT that co-occurred at least one time in a sentence in the test document corpus. However, the investigators disagreed with the SWISSPROT definition of gene synonymy and had six experts review the pairs in the gold standard list. This dropped the size of the gold standard list from about 900 to about 600 pairs. The investigators did not define a specific purpose for their synonym list, and this definition would surely effect the definition of a gene or protein name synonym. The investigators defined a synonym as the same gene or protein, but the SWISSPROT database uses a broader definition that appears to be more based on genes or proteins related closely enough to group them together for the purposes of information retrieval.

It is unclear what definitions various authors use in designating synonyms in their articles, but clearly these should be the gold standard for rating the recall of synonym extraction. The authors attempted to choose a very restrictive definition of synonymy by stating that the name must refer to the same gene. Unfortunately, this is an incomplete definition. What does the “same gene” mean? In the human genome a gene may have several names, each allele of the gene may have several names, and the gene may be present in more than one species. Does this include alleles, which are located in the same place on the chromosome but specify different sequences?

Research in synonym extraction requires defining what constitutes a synonym; ideally this should be done before the experiment is conducted. Yu and Agichtein made heavy use of the SWISSPROT database but did not agree with many of the synonym pairs in this database. Because of this difficulty Yu and Agichtein relied on a group of six biology experts to evaluate gene and protein name pairs as synonyms. They used the 588

synonym pairs that the experts agreed on for evaluating their systems. The 318 pairs that the experts disagreed on and the 83 pairs that they were unsure of were not used. This introduces a possible source of bias into their study, since the agreed upon pairs are likely the more common names, and these more common names may be more often found in the journal articles examined by their system. The effect of this may have been to positively bias their recall measures of the four approaches. Interestingly, even though their definition was intended to be very restricted, it was solely based on expert opinion, and the level of agreement was only 0.61 among experts, 0.83 between experts, and 0.77 overall.

Studies that evaluate text curation or classifier systems most often treat the process like a diagnostic procedure and evaluate the curate/no-curate decision as a diagnosis. Sensitivity and specificity are can be measured as well as area under the *receiver operating characteristic* curve. These studies are also dependent upon some type of gold standard, which usually is a group of experts. The experts are often chosen to be the same people that make the curation decision under test manually themselves. For example, in the system designed by Chapman et al. to detect radiology reports containing mediastinal finds associated with inhalation anthrax, the experts were five internists who read through the radiology reports just as they did in their medical practice [99].

This study is also interesting because while the test collection included almost 80,000 documents, evidence of anthrax was estimated to occur in only about 1% of the reports. Clearly it would have been burdensome for the five internists to read all 80,000 reports. However, random sampling would have resulted in a very small number of positive documents, making it difficult to achieve statistical significance.

The researchers resolved this dilemma by using an *enriched* sampling method. All documents classified as positive by their system was included in the test set, along with an equal sized random sample of documents not classified positive by the system, for a total of 1,258 radiology reports. After internist review of these reports, the true-positive, false-positive, true-negative, and false-negative rates determined in the randomly sampled subset was extrapolated to the rest of the documents previously not classified as positive by the system, and then sensitivity and specificity were computed [99].

Some studies do not attempt to do any formal evaluation. In particular, knowledge extraction tasks that look for new hypotheses, called *hypothesis generation systems*, such as those based on Swanson's ABC model, are most often evaluated based on whether anything novel and interesting is found [12, 25]. It would be difficult to apply the measures of precision and recall to these studies because there is no gold standard to evaluate the quality of a generated hypothesis. One method that some researchers working in this area have used is to try to duplicate the results of Swanson using automated systems. The experiments are deemed a success if the extracted hypotheses include those found by Swanson, specifically the relationship between fish oil and Reynaud's syndrome, and that between magnesium and migraine headache [11, 27]. Of course, this is a fairly low measure of success, and it is further diminished by the fact that the investigators know exactly what results they are looking for in advance.

One way that hypothesis generation systems could be evaluated is by expert evaluation of the usefulness of the results [42]. It remains to be demonstrated whether there would be any consistency of results across experts for an evaluation of this type. It is conceivable that these systems could be tested by putting them into use by biomedical

researchers and then comparing the research output of those using the systems verses those who do not. However, this would be a large, cumbersome, and long-term evaluation project.

In summary, evaluation of knowledge extraction systems remains an essential, but difficult, subjective, error-prone, and labor-intensive process. When available, test collections can ease the burden considerably and encourage the production of results that can be meaningfully compared across systems. On the other hand, some of the most exciting and potentially useful tasks for knowledge extraction, such as hypothesis generation, cannot be easily tested by current methods and may have to be put into actual practice to determine whether they generate useful results.

## IV. THEORETICAL FRAMEWORK

### Basic Theoretical Foundation

The central premise of SNLA is that the relationships between symbols (the S in SNLA) in the problem domain can be represented by constructing a weighted network (the N in SNLA) based on symbol co-occurrence using a set of recognition patterns, and that important logical (the L in SNLA) relationships between symbols can be identified by analyzing (the A in SNLA) the graph as a mathematical object. Here, symbols represent units of knowledge. These can be units of language, such as terms, words, synonyms, or units of meaning, such as concepts (each of which may have several names or associated terms).

The recognition patterns used must be chosen specifically for each knowledge extraction task. The mathematical analysis techniques should be reusable and applicable to many different knowledge extraction tasks; it should be possible to create a “toolbox” of techniques ready for application to many appropriate tasks.

It is useful to define several terms in the central premise more completely. A graph is a mathematical object having *vertices* and *edges*. Vertices, also called nodes, represent symbols. An edge is a connection between two symbols. Edges represent relationships between symbols and may be unidirectional or bi-directional. A graph with directional edges is called a *digraph*. A network is a graph with weighted edges, that is, each edge has an assigned numeric value. Note that there does not need to be an edge between every pair of vertices, and a vertex may have no edges. A path is a traversal from one vertex to another along edges, going through zero or more intermediary vertices. More formally a path between two vertices  $u, v$  is an ordered list of vertices starting with vertex  $u$  and

ending with vertex  $v$  where there is an edge between each consecutive pair of vertices in the list, and no vertex is repeated [100].

SNLA represents symbols (or semantic constructs) in the problem domain as vertices or nodes and the relationships between semantic constructs as edges. Weighting of the network edges is accomplished by using the frequency of observing the two symbols together in recognition patterns as defined in Section II. Observing two symbols and their relationship together in a recognition pattern is termed a *co-occurrence*. The edges of the network are weighted by a measure derived from the count of co-occurrences for each pair of symbol nodes connected by a relationship edge. The recognition patterns are specific to the knowledge extraction task at hand and define what types symbols and relationships can be recognized. Recognition patterns can be either limited and specific, recognizing only a pre-defined set of symbols and relationships, or flexible and extensible, recognizing an unlimited set of symbols and relationships based on some criteria specific to the problem domain. For example, the set of MESH terms used in MEDLINE make up a predefined set of symbols. A limited and specific recognition pattern for these could be simply each pair of MESH terms that occur together in a MEDLINE record. An example of a flexible and extensible recognition pattern would use a noun phrase and verb phrase POS tagger to identify actions and entities in journal articles. Note the different window sizes for these templates. The first example uses the set of MESH terms used on an individual MEDLINE record as the window size. The second example uses sentences within a journal article as a window.

Given a symbolic network created by this process, knowledge can be extracted by analyzing the graph mathematically. The possibilities are essentially endless, but a few

examples will illustrate the potential. Tightly related concept pairs can be extracted by searching the graph for the vertices connected by the highest weighted edges. Symbols that are related, but whose relationship is unrecognized in the literature can be extracted by searching the graph for pairs of vertices that have a path between them which are not directly connected. Sets of tightly related symbols can be found by examining the graph for clusters of high interconnectivity. Note that mathematical measurements can be defined for what constitutes a strong connection or a tight cluster, and these measurements can be reused across different knowledge extraction tasks requiring these definitions. The selection of what mathematical definitions are necessary is specific to a given knowledge extraction task

### **Representing and Inferring Knowledge with Symbolic Networks**

Given this basic approach to representing and extracting knowledge, various kinds of knowledge extraction tasks can be handled by SNLA by appropriate selection of the represented concept vertices and relationship edges and their types. Both concept vertices and relationship edges may be associated with a specific type and class. A concept node class is the general category of what it represents. For example, in a network of gene names, the concept node class is a gene name, and each node is a specific, instance of an individual gene name. A relationship edge may have both a type and a class. The edge type is the mathematical type of the edge, either bidirectional or unidirectional. The edge class is the specific relationship represented between two vertices of given type. Note that a network may have many vertices of the same class, but only one node for each specific instance. A network may have many edges of the same type and class. These symbolic networks are different from semantic networks in that semantic networks are created by



domain experts. Semantic networks are created in part to allow document authors to use codes and mark up to precisely identify important types and relationships within their documents. Symbolic networks are created automatically from pre-existing text given a set of symbol and relationship types, as well as a means of identifying these symbols and relationships with some measurable confidence. Note that the symbols and relationships in a symbolic network are not expected to be 100% accurate. Uncertainty is taken into account when analyzing the network.

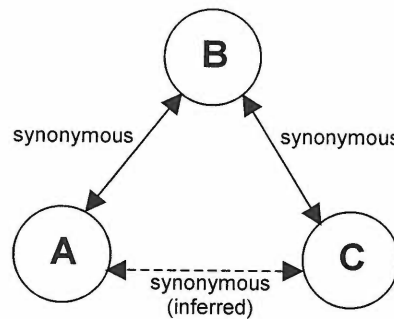
In the simplest network, all the vertices may be of a single class and all of the edges may also be of a single class and type. For example in a network representing synonymy between gene names, each node represents a name for a gene, and each edge represents a synonymous relationship between gene names. However, much more complex networks can be created. There may be several classes of vertices representing several groups of symbols. For example, there may be subject vertices and action vertices. In this network an edge may represent “is capable of” where an edge between a subject node and an action node means that the subject is capable of an action. In this network there would not be any edges between subjects or between actions, since the edge represents a relationship between a subject and an action. The network can be made more complex by adding an additional class of edge, say an edge between subjects that represents the subjects being co-located. In this case there would never be an edge of class co-located between a subject and an action or two actions.

### **Edge and Relationship Types**

While the classes of symbols and relationships are specific to the knowledge extraction task, relationship types have common characteristics based on their mathematical

properties of invertibility and transitivity. Each relation falls into one of a few basic types: invertible (symmetric, bi-directional) verses non-invertible (asymmetric or uni-directional), and transitive verses non-transitive. An invertible relationship is one that is symmetric, if A has relationship R with B; then B has relationship R with A. These are represented in SNLA as a bi-directional edge. A transitive relationship is one that applies across connections. If A has transitive relationship R with B, and B has relationship R with C; then A has relationship R with C.

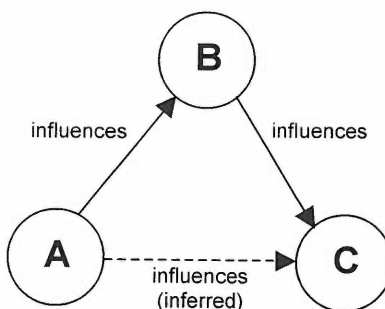
SNLA allows these various types of semantic relationships to be represented by controlling the directionality of the edges and the types of nodes between which edge types may appear. This will be illustrated with examples of several different types of edges and the kinds of symbolic networks that they help represent.



**Figure 3: Simple Bidirectional Network**

*Simple bidirectional networks* include only relationships that are invertible and transitive. For example in the network shown in Figure 3, if the labeled nodes represent terms, and the bidirectional arrows represent “is synonymous with”, then the solid arrows state that A is synonymous with B, and B is synonymous with C. The dashed arrow, which represents A’s synonymy with C can be inferred, and this relationship is invertible: if A is a synonym of C, then C is a synonym of A. These relationships and the method of inference continue to be true in much larger networks, even across multiple nodes.

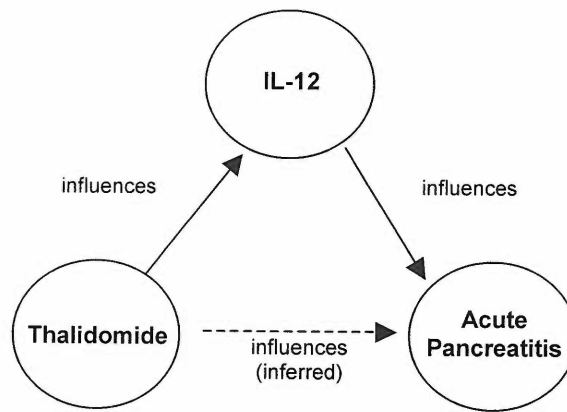
In contrast, a *simple unidirectional network* includes only relationships that are transitive, but not invertible. For example in the network shown in Figure 4, if the labeled nodes represent biologically active substances found in an organism, and the bidirectional arrows represent “influences”, then the solid arrows represent the knowledge that substance A influences substance B, and substance B influences substance C. The dashed arrow, which represents substance A’s net influence on substance C, can be inferred. Note that this relationship is not invertible. Substance C may have no effect on substance A at all. Again, the method of inferring the relationships between substances is valid in much larger networks, and can be followed across multiple nodes and arrows.



**Figure 4: Simple Unidirectional Network**

A useful property of simple, unidirectional relationships is that the conceptual relation, such as “influences”, can be general enough to apply to many different kinds of nodes. The nodes do not all have to be of the same semantic type, such as the biologically active substances in the example above. The same type of unidirectional relationship can represent the semantic association between concept nodes of very different types, such as between pharmacologic substances and biologic processes or between biologic processes and diseases. In fact, these different kinds of relationships can be expressed in the same network, and that network can be used to infer knowledge across the concept node types.

For example, Figure 5, shows a simple relationship, adapted from Weeber [27], between thalidomide (a pharmacologic substance), IL-12 (an immunologic factor), and acute pancreatitis (a disease). Thalidomide is known to influence the production of IL-12, and IL-12 is thought to play a role in susceptibility to acute pancreatitis. We can therefore infer that thalidomide may have an influence the disease acute pancreatitis. Note that the unidirectional relationships preclude inferring nonsense such as acute pancreatitis influences IL-2 or that IL-12 influences thalidomide.



**Figure 5: Relationships between Thalidomide, IL-12, and acute pancreatitis**

The directional nature of this type of network is capable of representing restricted relationships that increase the validity of inferred relationships. When constructed, the network can be restricted to include only relationships between types that are predetermined to be acceptable. For example, in a network including concepts of pharmacologic substances (P), immunologic factors (I), and diseases (D), construction can be limited to only include the associations  $P \rightarrow I$ ,  $I \rightarrow D$ , and  $P \rightarrow D$ . Note that since disease nodes have no outgoing edges, the network will never lead to the inference that a disease effects pharmacologic substances or immunologic factors. And since pharmacologic substance nodes have no incoming edges, the network will never lead to the inference that a disease or immunologic factor affects a pharmacologic substance. This property of a network built on a restricted set of relationship types is termed *symbolic network logical consistency*.

For greater semantic richness, it is possible to extend a network such as this with edges representing the influence of one immunologic factor on another, that is, the  $I \rightarrow I$  edges. This would allow the discovery of a drug's effect on a disease through an intermediate immunologic mechanism. Adding this type of relationship greatly enhances the expressiveness and usefulness of the network while maintaining the desirable property of network logical consistency.

## **Integration with the Knowledge Extraction Framework**

SNLA can play a key role in several important parts of the previously defined knowledge extraction framework. This includes the components: rating of instance confidence, rating of template confidence, and accumulate of instance knowledge. Mathematical analysis of the graph properties provides the connection between the symbolic network and the components of the knowledge extraction process.

Instance confidence can be derived from the symbolic network by using the co-occurrence information contained in the network edge weights, and comparing this to some threshold or standard value. This can be either a binary or continuous measure. A binary measure would set a threshold, relationships with co-occurrences above this value would be treated as true, and those below the value would be treated as undecided. A continuous measure could be used as a relative measure of certainty. For example, the structure of weighted connections between a set of symbols could be compared to the probability that the structure could occur randomly in a network with similar average properties. The less random the structure, the higher the confidence that the knowledge represented by that structure is actually true.

The mathematical analysis of the symbolic network can also be used in pattern selection, based on the idea that a good pattern should create networks that have the properties appropriate for the kind of knowledge being extracted. The investigator must first determine what kind of network structure will best represent the extracted knowledge of interest. Patterns that lead to appropriately constructed networks are kept, and those that lead to incorrect constructed networks are discarded. For example, say that the task is to determine sets of synonyms. In this case the graph should have the

appearance of a number of tightly connected clusters that are disconnected from each other. Obviously, a graph where every node is connected to every other would not represent the synonym relationship, and would imply that every concept in the network was a synonym with every other, an unlikely situation. Conversely, a graph where no nodes are tightly connected with any others would imply that there are no synonyms in the data set, also an unlikely situation if the data set contains the knowledge that we think it does. Graph analysis techniques include measures of graph clique-ness and clustering that can be used as measurements for these knowledge extraction tasks [51-54].

Third, mathematical analysis of the graph structure can reveal new instance knowledge. Instance knowledge consistent with the problem space may not appear directly in the data set, but instead it may be possible to infer this knowledge based on the semantics of the concept nodes and the edges that represent transitive relationships. Graph traversal algorithms such as Dijkstra's shortest path algorithm [101] can be used to determine which symbols are connected and which are not, and how the strength of those connections.

A graph traversal algorithm called *weakest link, strongest chain*, has been created and may be especially useful for knowledge inference in symbolic networks. Weakest link, strongest chain can most easily be explained by using a chain made of links as a metaphor for relationships between concepts or terms. The strength of the chain is the strength of the weakest link, and is analogous to the confidence of the relationship. With a set of chains connecting two objects, the strongest connection is the strongest individual chain.

The co-occurrence counts are used directly as the edge weights in the graph. The edges may be directed or non-directional. Applying the chain metaphor to logical inference, the strength of a synonym “link” between two gene names is the number of co-occurrences of the two gene names. A chain of inference can be constructed between a series of concepts. The strength of that chain is the strength of the weakest link of that chain, that is, the smallest number of co-occurrences in the set of symbol pairs connecting the two given symbols at the end of the chain. Multiple chains may connect a pair of symbols. The strength of the strongest connecting chain is a measure of the confidence that two symbols actually are related to each other according to the semantics of the relationship edge. The result of applying the weakest link, strongest chain algorithm to the co-occurrence data is a list of the strongest connections between each pair of connected symbols for a given relationship class. This “candidate” instance knowledge can then be rated according to the methods used to evaluate instance confidence discussed above.

The implementation of this algorithm is much like the implementation of the classic shortest path algorithm [101]. However, instead of adding the edge weight as a path is walked, a minimum is taken, and instead of choosing the path with the shortest total, the path with the largest value is taken. The time complexity is also similar to the shortest path algorithm, being proportional to the square of the number of symbols (graph nodes) [101].

A variation of the weakest link, strongest chain algorithm would treat the individual links as confidence measures and evaluate the strength of a chain as the product of the individual confidences. This variation requires a method of converting the co-occurrences at each connection into a probability. This can be done using a Poisson random-graph



model as discussed above, or by more sophisticated methods such as examining the statistical distribution of individual node metrics.

A particular strength of the weakest link, strongest chain algorithm is that it automatically discounts indirect connections. Since the strength of a chain is the strength of the weakest link, a long chain must have a larger number of strong links to be as strong as a shorter chain. Therefore, longer connections must include more co-occurrences (or at a minimum, more certain information) than shorter connections to have the same confidence. These applications of SNLA are used in the software system investigating the research question on extracting gene and protein name synonyms from MEDLINE abstracts.

### **SNLA Provides a Unified Approach**

Srinivasan quotes Hearst when she states that “the key goal of mining whether from well structured databases or numeric data or from text collections is the discovery of new knowledge” and states that the goal for information extraction systems is to “extract nuggets of information from collections of texts” [9]. She lists several types of extracted knowledge: referential (lists of names of drugs), attributive (attributes of a gene or set of genes), or relational (interactions between proteins) [13]. SNLA is useful in many of these knowledge extraction and text mining tasks and provides a unified approach to several kinds of knowledge extraction tasks that have been previously treated separately. For example, referential and attribute lists can be approached as tight network clusters. Relational interactions can be analyzed following relationships along network edges.

Logical inference can take relational interaction one step farther by connecting concepts (or more generally, symbols) through a chain of reasoning. In this way, new

potential discoveries can be identified based solely on evidence in the medical literature. Researchers can then pursue the most promising of these discoveries. Currently most work on applying logical inference to medical bibliographic data uses Swanson's ABC model [11, 25, 27]. This model is limited in that it only can connect three concepts by two relationships in a linear "A implies B implies C" progression. SNLA provides much more general logical processing, allowing discovery of relationships separated by any number of intermediate concepts.

## **V. RESEARCH QUESTION AND OPERATIONAL DEFINITIONS**

In this section the research question will be further defined, focusing on stating how SNLA will be applied to the research question and supplying operational definitions necessary to fully specify the research question.

### **Applying SNLA to the Research Question**

Again, the research question to be studied is whether SNLA can be used as a tool in knowledge extraction, and specifically, can it be used to extract a high quality set of gene and protein name synonyms from MEDLINE database records?

To apply SNLA to the task of gene and protein name synonym extraction, this task will be outlined in terms of the knowledge extraction framework detailed above. Each component of the knowledge extraction framework will be defined for the purpose of studying the research question, and the components that will use SNLA will be described. The task-specific portions of SNLA as applied to the research question will also be specified.

### **Operational Definitions**

In order to detail the specific knowledge extraction method to be used in studying the research question, it will be useful to have an operational definition for key constructs to be used in defining the research question.

#### *Gene and protein names*

The research will focus on gene and protein short names and symbols. Full multi-word names of genes will not be considered for two reasons. First abstracts appear to make much heavier use of short names and symbols than full names, perhaps to save

word count. Second, recognizing multi-word full names is a difficult problem in itself [48, 56], and would complicate the investigation of the research question.

Gene and protein nomenclature is inconsistent and complex, and often the difference between a gene name and the name of the corresponding protein is expressed in typesetting features (e.g. italics) that are not present in MEDLINE abstracts. For the purpose of this research, no distinction will be made between names of genes and the proteins that they code for.

The upper and lower case of names as well as individual characters within names will not be considered as significant. Examination of the MEDLINE database showed no consistent or significant differentiation between upper and lower case gene names, although gene names for some species, such as yeast, tended to appear in lowercase, while for other organisms, such as human, usually appeared in uppercase.

These names will all be collectively termed *gene and protein names*, *gene/protein names*, or when the context makes the usage clear, simply *names*.

#### *Gene and protein name synonyms*

What exactly is meant by a gene or protein name synonym? One possible definition for a name synonym is that the names specify exactly the same sequence of base pairs in a specific organism. However, this definition is very limiting. For example, this definition would not treat names of different alleles of the same gene as synonymous, nor would it treat names for genes of similar sequence and function in different organisms as synonymous.

It will help to define the intended use for a list of gene and protein name synonyms. For the purposes of studying the research question, the intended use of the synonym list is to

aid researchers in literature search and retrieval, and to provide an essential first step in gene knowledge extraction research by collapsing many names into a single concept.

With much biomedical research being done on non-human organisms, it is reasonable to assume that many researchers would want to look at literature that discusses these related genes. For similar reasons, it is helpful to not separate gene and protein names related to the same biological product.

The definition to be used here will be based on the stated intended use of the synonym list. The operationalized definition of gene name synonym is that two gene or protein names are synonyms if they specify two genes that are identically located in a single species, or if they specify for functionally equivalent genes or proteins between two species that are treated as homologs in the biomedical literature. Functional equivalence is defined in terms base pair sequence similarity and *synteny*, the preservation across species of gene order and location inter-relationships [102]. The two synonymous gene names together constitute a *synonym pair*.

### **Applying the Knowledge Extraction Framework to Synonym Extraction**

The application of each component of the knowledge extraction framework to the task of extracting gene and protein name synonyms from MEDLINE abstracts framework will be described here. Task-specific details of the application of SNLA will be described in the sections covering the relevant knowledge extraction components. The components of the knowledge extraction system for extracting gene and protein name synonyms are as follows.

### *Bounded Problem Domain*

The domain is the set of gene and protein names used in the biomedical literature. The knowledge in this domain includes how genes and proteins are discovered and named, and why genes and proteins often have several names with many variants [56, 57]. These names can be grouped into groups of synonyms, using the operationalized definition of synonymy given above.

### *Data Set*

This is the set of data to be used as the source material for knowledge extraction. For the research question, the abstract field of a subset of MEDLINE database records was used. It should be noted that this is a database very heavily biased toward research published in English [1], therefore it is expected that the gene and protein synonyms will most often be those used in research done in English speaking countries. The data sets consist of abstracts from a year's worth of MEDLINE records containing the word "gene," and retrieved on-line using the PubMed query system. Abstracts meeting the query criteria from the year 2001 served as training data used in the development of the system. Abstracts from the year 2002 served as validation data, allowing the algorithm to be checked for robustness and generality on a dataset separate from the training data. Finally, abstracts from the year 2003 were used as the test collection. The final version of the program developed using the training and validation sets was run on the test collection and results computed. This will be presented in more detail in the sections on experimental design and results.

### *Initial Knowledge*

Two forms of initial knowledge are used in this research. First, a set of orthographic rules are used that will specify combinations of letters, numbers, and symbols likely to constitute a gene name. These orthographic rules form a portion of the recognition patterns used to extract synonym relationships. This is a very simple form of NER [41, 43, 72, 77, 78]. This simple approach to NER has been used successfully by other researchers [10, 36], and helps to reduce the size of the constructed network, as well as some of the noise. The orthographic rules are based on the analysis of the different types of gene names done by Proux et al. [57] and are similar to that used by Fukuda et al. for protein names [21], with some added flexibility to support the greater irregularity of gene names and increase recall at the expense of some precision. The named entity recognition does not have to be perfect since the SNLA network is the main mechanism of screening out noise (see rating of instance confidence below), hence the sacrifice of some precision for recall. Descriptively, the orthographic rules recognize gene and protein names as space or punctuation delimited strings, which must begin with a letter and contain between three and fourteen characters consisting of any number of letters, numbers and hyphens. The regular expression syntax supported by the Python programming language [103], was used to represent gene names in the recognition patterns combined with orthographic-based procedural rules for determining what symbols are likely to be gene names. A list of stop patterns was also used to help remove symbols that look like gene names, but are not (e.g., “mRNA”).

The second form of initial knowledge is a small number of known correct gene and protein name synonym pairs that will be used to “seed” (or initiate) the knowledge

extraction process. The initial knowledge will be used to find patterns that then can be used to find more instance knowledge. The initial knowledge was obtained by choosing the most common gene name pairs found in the training dataset that were listed in SWISSPROT. During training, a set of eight initial seed pairs was found to be adequate. Raising the number of seed pairs to 12, 16, 24, and 32 did not significantly change the performance.

#### *Instance Knowledge*

These are instances of the synonym relationship between gene and protein names. In terms of SNLA, gene and protein names are the vertices, and the edges represent the synonym relationship between names. The network edge weights will be equal to the number of times that the synonym relationship has been detected by the patterns in the data set. This is the co-occurrence count as discussed above. The result of the extraction process will be a list of synonym name pairs, which can then be evaluated for accuracy.

#### *Recognition Patterns*

These are patterns or templates for recognizing instance knowledge in the data set. For the research question, recognition patterns will be applied to the text in MEDLINE abstract fields sentence by sentence. Therefore the window size or granularity of the recognition patterns will be a single sentence [38]. A recognition pattern will consist of two or more repetitions of the orthographic pattern for a gene or protein name separated by the text found between known pairs. One or two leading and trailing words will be included in the pattern to increase the contextual information.



The initial set of recognition patterns was found using the initial knowledge. Thereafter, more recognition patterns were detected using the accumulated instance data. Only the discovered instance data with the highest confidence will be used to help find patterns.

A specific example will help make this clear. If the known synonym pair is (CIP1, WAF1), and this name pair is found in the text:

Two percent or greater nuclear staining with WAF1/CIP1 monoclonal antibody was determined by hazard ratio analysis to constitute positive p21 expression.[104]

Then the following four patterns will be extracted by the system, where \$GENE\$ represents the orthographic pattern that matches possible gene and protein names:

- \$GENE\$/\$GENE\$
- with \$GENE\$/\$GENE\$
- \$GENE\$/\$GENE\$ monoclonal
- with \$GENE\$/\$GENE\$ monoclonal

#### *Application of Patterns to the Data Set*

This is the process of reviewing the data set for matches with the recognition patterns. In this case, the abstract field of the MEDLINE records are searched for sequences that match the recognition patterns. For matching sequences, the gene and protein name portions identify the evidence for the synonym pair. This results in one or more co-occurrence counts being added to the network. This provides evidence to the next step, extraction of the instance knowledge.

Again, a specific example will make this clear. The pattern “with \$GENE\$/\$GENE\$” matches the following sentences:

Of the children with NOD2/CARD15 variants, 44% were < or =5th percentile for weight at diagnosis, whereas only 15% of children without mutations were < or =5th percentile (chi2=8.7; p=0.003; OR=4.5; 95% CI=1.4-14.4).[105]

Human glioma xenografts treated with PTEN/MMAC gene transfer exhibited significantly decreased vascularity both in an orthotopic and in an ectopic model. [106]

The pair co-occurrence (**CARD15**, **NOD2**) is extracted from the first sentence, and the pair (**MMAC**, **PTEN**) is extracted from the second.

#### *Extraction of Instance Knowledge*

Instance knowledge is extracted by applying the *weakest link, strongest chain* algorithm to the co-occurrence network. This algorithm extracts name pairs using both direct (a found co-occurrence) and indirect (inferring synonymy) evidence of synonymy. It was found during algorithm training that synonym pairs having only indirect evidence of synonymy and a *weakest link, strongest chain* count of one had a very high error rate. The current implementation only follows inference chains links having a count of at least two.

#### *Rating of Instance Confidence*

Rating of instance confidence is important in order to select the highest-confidence pairs for the next iteration of pattern and instance extraction. SLNA is used in this component. Instance confidence will be rated by comparing the co-occurrence count with the predicted count in a random network with the same number of vertices and edge counts [52, 53]. This will give the probability that a synonym relationship with the given co-occurrence count would be seen by chance in a random network. Lower probabilities of the co-occurrence count being seen randomly imply a higher confidence in the instance knowledge. This idea is similar to the *noisy or function*, where the probability of a conclusion based on multiple co-occurrences being correct is defined as one minus the probability that all of the observed co-occurrences are incorrect [57].

Since many synonym relationships are examined simultaneously, it is important to take this into account, much like applying the Bonferroni correction for multiple comparisons

in statistical hypothesis checking [107]. Therefore the measure is based on the probability of seeing at least one synonym relationship of the given co-occurrence count or greater in the entire network.

Mathematically, this can be described as follows. Given a random graph created by a Poisson process with  $C$  nodes,  $M = C*(C-1)$  possible edges, and  $N$  total co-occurrence counts in the entire network, the probability  $P$  that the co-occurrence count between any two nodes will have a count of  $n$  is:

$$P(X = n) = \frac{\mu^n e^{-\mu}}{n!} \mid \mu = N / M$$

(Equation 1)

Then the probability that there does not exist at least one edge with  $n$  or greater co-occurrence counts represents our confidence in the synonym pair and is:

$$confidence = P(X < n) = \left[ \sum_{k=0}^{k=n-1} P(X = k) \right]^M$$

(Equation 2)

Which reduces to:

$$confidence = P(X < n) = \left[ \sum_{k=0}^{k=n-1} \frac{\mu^k e^{-\mu}}{k!} \right]^M \mid \mu = N / M$$

(Equation 3)

This is used as an estimate of the probability of the synonym relationship being detected in the literature by chance alone. During training, it was found that a confidence threshold of 0.999 was necessary in order to select out only the best pairs for use in the next iteration of extraction.

### *Recognition of New Patterns*

This is the process of recognizing new extraction patterns, based on the initial knowledge and gathered instance knowledge from the data set. For the research question, recognition patterns were created by first examining the text in the area of known pairs of name synonyms within the sentences in the data set as described above. From this text, recognition patterns were built by using the orthographic pattern (described in the section on initial knowledge) to replace the gene and protein names, along with text fragments found near the known gene and protein name pairs in the data set. The initial recognition patterns will be found by searching the text for the initial knowledge seed pairs. Subsequent recognition patterns are found by search for new high confidence synonym pairs found in the prior iteration.

### *Rating of Pattern Utility*

This is the process of evaluating the quality of the extraction patterns. Given a set of patterns, we wish to use the subset that leads to the best results. SNLA is used to access the quality of subsets of the extraction patterns by examining the contribution of the extraction patterns to the network structure. For gene and protein name synonyms, the expected network structure is a large number of tightly linked small clusters. This follows directly from the definition of gene/protein name synonym and our domain knowledge of gene and protein names: there should be more gene names than there are genes, and groups of gene and protein name synonyms should be distinct from each other. The cluster size (the number of names in a group of name synonyms), should be relatively small, say less than 20, compared to the number of names in the network nodes, which may be several thousand (this is based on examining the number of synonyms for a gene

in Locus Link [46, 108] and allowing for orthographic variants). The clusters should be tightly linked, this means that the nodes in each cluster should be much more strongly connected to other members of the cluster, and non-connected, or only weakly connected to nodes (names) that are not part of the cluster.

Mathematically, this can be measured as a property of the network. Authors have previously described a “clustering coefficient” which is the degree to which a node’s neighbors are neighbors of each other [51]. The average taken over all the nodes of the network is the *mean clustering coefficient* or *MCC*. A related concept is the non-clustering coefficient, which would represent the degree to which a node’s neighbors are not neighbors of each other. The average taken over all the nodes of the network is the *mean non-clustering coefficient* or *MNCC*. A high quality pattern would create a network structure having a MCC and a low MNCC. This can be reduced to a single measure by taking the ratio of MCC to MNCC. This ratio is then the measure of quality for the pattern. The best patterns will be selected and used to extract knowledge. Inferior patterns will be thrown away.

To formally define MCC, we will compute the clustering coefficient as the average co-occurrence count between neighbors of a given node, and then compute the mean clustering coefficient as the mean of the average co-occurrence count between neighbors of a given node for all nodes. Only nodes with at least two neighbors are included, since these are the only nodes that could possibly have co-occurrences between neighbors.

Given a network  $C$  with  $|C|$  vertices, define *neighbors*( $c$ ) as an operation that returns the list of the vertices which share an edge with vertex  $c$ , that is, those that have at co-occurrence count of at least one with the gene/protein name associated with node  $c$ . Also

define an operation  $weight(a,b)$  as returning the weight of the edge between nodes  $a$  and  $b$ , that is, the co-occurrence count of the pair of gene/protein names associated with nodes  $a$  and  $b$ . The standard combination function of  $n$  items taken  $x$  at a time is notated as  $combinations(n,x)$ . Then the clustering coefficient for each node  $c$  is:

$$CC(c) = \frac{1}{combinations(|neighbors(c)|, 2)} \times \sum_{\forall a,b \in neighbors(c) | a \neq b} weight(a,b) \quad (\text{Equation 4})$$

And the average over all the nodes in the network with at least two neighbors is:

$$MCC = \frac{1}{|C|} \sum_{\forall c | |neighbors(c)| > 1} \frac{1}{combinations(|neighbors(c)|, 2)} \times \sum_{\forall a,b \in neighbors(c) | a \neq b} weight(a,b) \quad (\text{Equation 5})$$

Furthermore, we can compute the mean non-clustering coefficient (MNCC) as the mean of the average link count between neighbors of a given node and non-neighbors of the given node for all nodes. The non-clustering coefficient for each node  $c$  is:

$$NCC(c) = \frac{1}{|neighbors(c)| \times (|C| - |neighbors(c)|)} \times \sum_{\forall a \in neighbors(c), \forall b \notin neighbors(c)} weight(a,b) \quad (\text{Equation 6})$$

And the mean non-clustering coefficient for the entire network is:

$$MNCC = \frac{1}{|C|} \sum_{\forall c \in C} \frac{1}{|neighbors(c)| \times (|C| - |neighbors(c)|)} \times \sum_{\forall a \in neighbors(c), \forall b \notin neighbors(c)} weight(a,b) \quad (\text{Equation 7})$$

Then the quality of a pattern based on the network structure created by adding the co-occurrences detected by the pattern to the current network structure is:

$$quality = MCC / MNCC$$

(Equation 8)

Similarly, the CC/NCC ratio can also be computed for an individual node. This value will be useful in sorting high confidence extracted pairs before using these pairs to find

new extraction patterns. To order the extracted high confidence pairs, the CC/NCC ratio for each of the two nodes in a synonym pair will be computed, and then the individual node ratios will be multiplied. This will give a measure of relative confidence for the extracted high confidence pairs that can be used to sort the pairs.

### *Evaluation*

The evaluation of the system is based on comparing the extracted synonym list to a gold standard and computing precision and recall for the synonym pairs. The absolute number of correct pairs extracted will also be evaluated, as well as a measure of the efficient use of the initial seed pair knowledge. This will be fully described in the section on experimental design.

### *Gold Standards*

No complete list of gene and protein name synonyms currently exists. There are also no standard test sets for name synonyms, although some do exist for abbreviation extraction and other biomedical text mining tasks [44, 46]. However, there are data sets that can function as reasonable approximations of a gold standard for the purpose of studying the research question.

A gold standard synonym pair set for evaluating precision was constructed out of symbol, synonym and alias fields from several protein and genome databases available on-line. The gold standard synonym pair set for evaluating recall was constructed by extracting the synonym pairs listed in the SWISSPROT database [109] that are present in sentences in the abstract test collection. Details on how these test sets were constructed and used as gold standards will be more fully described in the section on experimental design.

## *Results*

Results are the final useful output of the system intended to answer the initial knowledge extraction question. For this research question, the useful output will consist of the gene and protein name synonym list, as well as the evaluation results. The gene and protein name synonym list will be a useful contribution in improving the searching of the gene and protein biomedical literature. The evaluation results will help establish the credibility of SNLA as a tool in knowledge extraction.

### **Algorithm for Gene and Protein Synonym Extraction**

At this point the full algorithm for extracting gene and protein name synonym pairs from MEDLINE abstracts can be presented in terms of the components discussed above. The steps of the algorithm are as follows (see Figure 6):

1. Select the data set of MEDLINE abstracts.
2. Parse the abstracts into sentences.
3. Filter out any sentences that do not contain a least two potential names, based on the orthographic pattern and procedural rules for gene and protein names.
4. Create an empty high confidence pairs list.
5. Establish a threshold confidence value for instance knowledge to be used for creating new patterns. A confidence threshold of 0.999 was found to be effective during training and was used here.
6. Based on examining the training data, provide a small number of gene and protein name synonym pairs to be used as the initial “seed” data. This results in the initial set of instance knowledge. Place these pairs into the high confidence pairs list. Eight initial seed pairs were found to work well.



7. Create an empty candidate pattern list.
8. Create an empty knowledge network.
9. Sort the high confidence pairs list based on the product of the MCC/MNCC ratio for each of the two symbol nodes in the pair. The pair with the highest product was used to find new patterns first, followed by the other high confidence pairs in descending order. For the initial seed pairs, which are all known to be correct, the ordering is not significant, and therefore a default MCC/MNCC node product value of 1.0 was used to for the initial seed pairs.
10. Construct candidate recognition patterns based on the text fragments that occur within the text between the current set of high confidence pairs. Add the new patterns found to candidate recognition pattern list. Each new high confidence pair is used to scan through the test collection, looking for sentences containing the pair, and constructing four patterns based on the matching sentences. As previously described, the four patterns consist of the pattern created by the gene name placeholders for the symbols of the pair along with the text in-between, along with three additional patterns formed by including the non-gene name token preceding the pair, the non-gene name token following the pair, and the pattern formed by including both the non-gene name token preceding and following the location of the gene name pair in the matching sentence. Stop adding patterns to the candidate pattern list when all high confidence pairs have been used or the number of patterns reaches some predetermined limit. During training, a pattern count limit of 150 was found to be adequate, balancing the

need to consider a large variety of patterns with the processing time needed to evaluate them.

11. For each new candidate pattern, search the text collection and record the set of symbol co-occurrences that the pattern detects.
12. From the list of candidate patterns, determine the subset whose co-occurrences produce the knowledge network with the highest quality score. During this step, each subset of patterns is treated as creating its own, temporary, knowledge network. This is essentially a discrete combinatorial optimization problem, which is best solved using a genetic algorithm [110, 111]. Details of the genetic algorithm are included in the following section on software implementation.
13. Remove the patterns selected by the previous optimization step from the candidate pattern list and add their co-occurrences to the knowledge network.
14. Extract the synonym pairs from the knowledge network.
15. Using the confidence threshold selected above, replace the contents of the previous high confidence pairs list with the set of extracted synonyms that have not yet been used to generate patterns and whose co-occurrence count is greater than that required for the confidence threshold.
16. If the high confidence pairs list is not empty, then go to step 9.
17. Scan the network to extract out the final list of gene and protein name synonym pairs and the computed confidence of each pair.

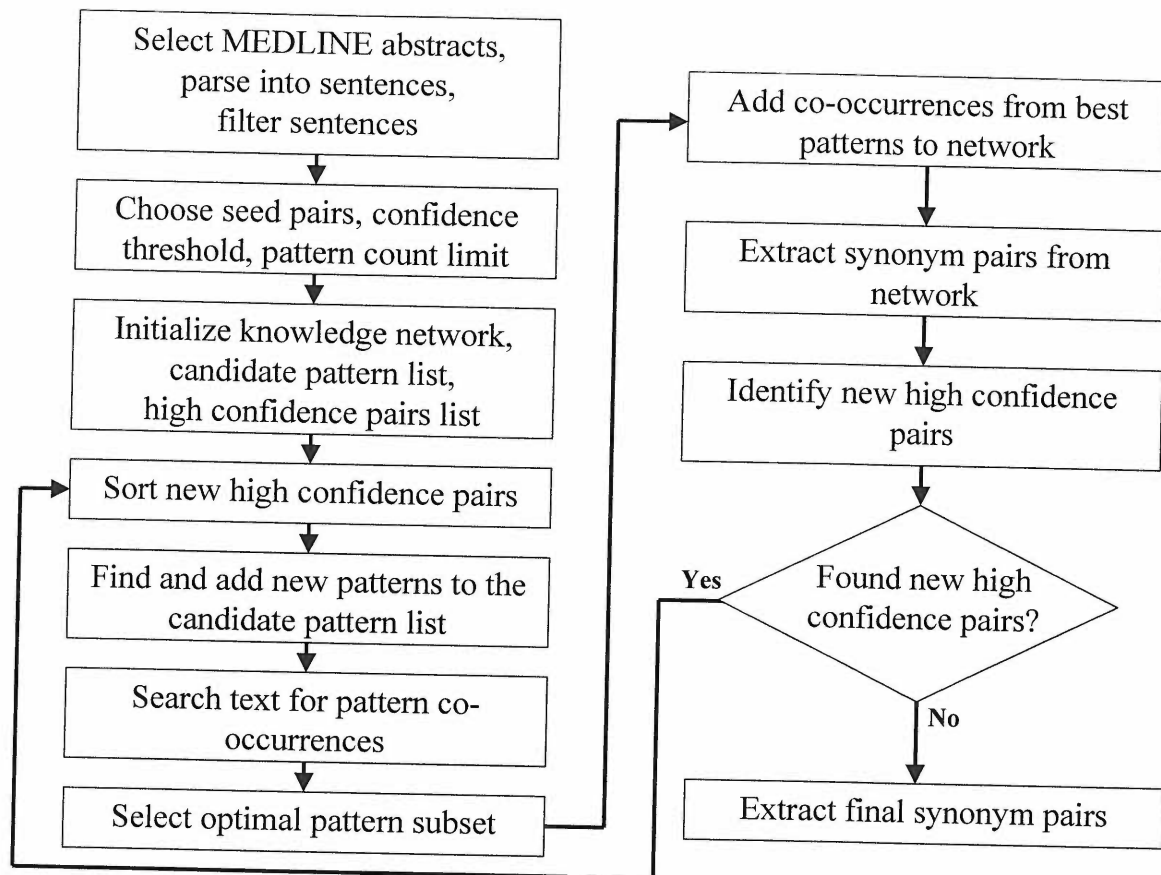


Figure 6: Synonym Extraction Algorithm

## VI. SOFTWARE IMPLEMENTATION

The software system is implemented in object-oriented Python [103]. Python was chosen because it allows rapid, object-oriented development, and provides high-level built-in data types such as dictionaries and lists that make the implementation straightforward. Object-oriented design allowed the system to be developed in an incremental, testable manner, and facilitates future extension as well as code reuse for other research applying SNLA-based techniques.

From a high level perspective, the software system takes a large text file containing MEDLINE records as input, performs analysis using the SNLA-based algorithm described above, and produces as output a list of synonym name pairs along with the co-occurrence count and confidence measure for each pair. A separate log file contains the list of patterns found and chosen, as well as the list high confidence pairs used at each iteration to generate new patterns.

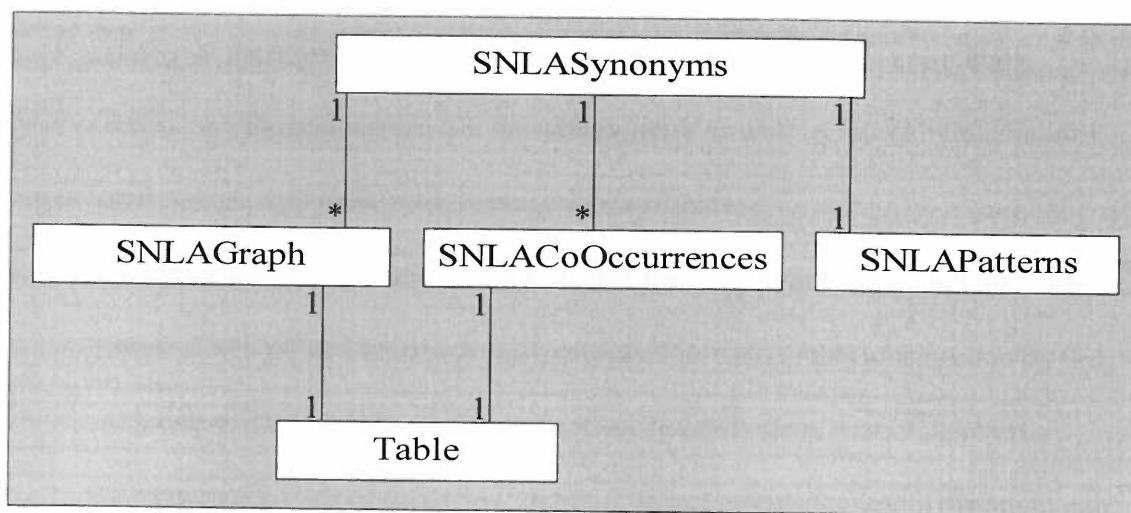


Figure 7: Entity-Relationship Diagram of Software Architecture

list of uppercase gene name pairs.

- **SNLACoOccurrences.** This module implements the class used to track the sentences a symbol co-occurrences has been found in. Since many patterns may match a given co-occurrence within a particular sentence, it is important to make sure that a symbol co-occurrence within a sentence is used only once. The **SNLACoOccurrences** object tracks the source identifiers (sentence numbers) where a symbol co-occurrence has been found. This allows the main function in the **SNLASynonyms** module to check whether a co-occurrence is new or has already been used.
- **Table.** This module implements a sparse 2-dimensional table, where the row and column keys, and the value stored at the combination of keys, can be any Python value. The implementation uses a two-level Python dictionary structure to maximize time and space efficiency. An efficient **Table** class is central to the implementation because the **SNLAGraph** and **SNLACoOccurrences** modules use it heavily.

### **Implementation Parameters, System Constants, and Design Decisions**

Several parameters, system constants and design decisions had to be determined while implementing and training the system. These decisions will be listed and described here. For each parameter, constraint or design decision a descriptive title will be followed by an explanatory paragraph that details the function of any related parameters. Parameters and constants are listed by their associate module name, followed by a period, followed by the parameter name.

### *Symbol Delimiters*

In order to parse the sentences into individual tokens, white space and other delimiters were used to separate words and other sequences of characters. Since parsing was completely lexical, a decision had to be made on which punctuation characters would be delimiters and which could appear within a symbol. The system constant `SNLAPatterns.TOKEN_DELIMITER_LIST` contains the characters that are treated as symbol delimiters. These were determined during system training and include: period, comma, parenthesis, braces, brackets, equals, semicolon, colon, explanation point, question mark, caret, dollar sign, pip, forward slash, percent sign, greater-than, less-than, plus, ampersand, and single and double quotes.

### *Case Insensitivity*

Early on in the development of the system it was found that there was little consistency in the use of capitalization in the training data. Making distinctions about gene and protein name symbols based purely on the capitalization was not found to be useful. In fact, it was counterproductive to distinguish between upper and lower case gene symbols. Therefore the system searches for symbols case-insensitively and converts recognized gene name pairs to uppercase for entry into the knowledge network.

### *Pattern Length*

In order to limit the overall symbol distance between candidate synonym pairs, a system parameter `SNLAPatterns.MAXIMUM_PATTERN_LENGTH` is included to limit the maximum length of recognition patterns. This includes slots for gene names as well as words and punctuation providing context. During training a value of 10 was found to be an adequate limit.

### *Determining Possible Gene Names*

The software uses a function in the `SNLAPatterns` module called `isGeneName()` to determine whether a delimited string might be a gene name. This function is not intended to provide highly accurate named-entity recognition. Instead, this function simply looks for orthographic evidence that a symbol may be or is likely not to be a gene name. While this method would not be accurate on its own, the context provided by the synonym patterns provides some additional gene name entity filtering later in the algorithm.

There are two reasons that this simple form of gene and protein name entity recognition was chosen. First, one of the goals of the research was to determine what level of performance could be obtained using the contextual information provided by the synonymy relationship without high quality named entity recognition, as well as to quantify the improvement that accurate named-entity recognition could provide. Second, getting approval, obtaining the source code, and integrating a named entity recognizer from another research group into the software system could have become a complex project in and of itself.

Developing and implementing a set of orthographic rules for making approximate “gene name, not gene name” decisions was straightforward. First, the symbol must match the regular expression for a gene name. The regular expression is given in the system constant `SNLAPatterns.GENE_NAME_REGEX`. Basically it allows gene names to contain mixes of letters and numbers three to fourteen characters in length. Colons as well as some other special characters such as asterisk are allowed.

Second, it must not be of a form listed in the stop list. The stop list is a list of regular expressions found in the training data to often occur in gene name contexts but that do

not represent gene names. Included are terms such as “mRNA”, and “VIRUS”, as well as patterns such as “.\*-ASSOCIATED”, and “.\*-INDUCED”. Note that the recognized symbol patterns may contain gene names, but not be gene names in and of themselves (e.g., “WAF1-ASSOCIATED”). The current version of the software makes no attempt to extract out the gene name portion. The stop list is implemented as system constant **SNLAPatterns.GENE\_NAME\_STOP\_LIST**.

Lastly a series of orthographic rules are applied to the symbol. Symbols consisting of all lower case letters are most likely to be English words and not considered gene names. A simple future improvement would be to incorporate a dictionary to screen out lowercase English words, but accept non-words. Symbols containing mixes of letters and numbers are considered gene names. Finally, gene names are not allowed to begin with the characters dash, colon, period, asterisk, or tilde, and they are not allowed to end with the characters dash, period, or colon.

#### *Filtering Out Non-Contributing Sentences*

The algorithm works with a text window size of one sentence, that is, evidence of name synonymy must occur within a sentence for the algorithm to recognize it. This implies that a sentence must contain at least two potential gene names in order to contribute to the knowledge network. Furthermore, the system contains a parameter **MAXIMUM\_EXTRACTED\_DISTANCE** that sets the maximum number of non-gene or protein name symbols allowed to occur between a pair within a pattern. A value of 4 was chosen during system training. Therefore, any sentences that do not have at least two gene names separated by less than or equal to the **MAXIMUM\_EXTRACTED\_DISTANCE** will not contribute any information to the algorithm and can therefore be filtered out before processing. This



reduces the size of the data set by about two-thirds, which helps with efficiency since the algorithm makes multiple passes over the data set looking for pattern matches.

#### *High Confidence Pair Selection Parameters*

The algorithm has two system parameters that help select the only most likely discovered synonym pairs for later use in generating patterns.

- **SNLASynonyms.PAIR\_HIGH\_CONFIDENCE\_THRESHOLD.** This is the minimum confidence value that a synonym pair must have in order to be considered a high quality synonym pair and used for pattern extraction. During system tuning on the training data set a value of 0.999 was found to give the best results.
- **SNLASynonyms.MINIMUM\_PAIR\_QUALITY.** As discussed earlier, high confidence extracted synonym pairs are sorted by the product of the node quality prior to using the pairs to search for patterns. This system parameter sets the minimum required quality product. The pair quality product is a measure of the collaborative evidence for pair synonymy that exists in the graph. Pairs which are synonyms of each other and do not share other common synonyms will have a pair quality product of zero. The purpose of this parameter is to screen out pairs that have little or no corroborating evidence of synonymy. During system tuning, a value of 2.0 was found to be effective.

#### *Pattern Selection System Parameters*

The algorithm includes two system parameters that screen out unlikely patterns from being further evaluated. These include patterns that match too many or not enough sentences.

- **SNLASynonyms.MINIMUM\_PATTERN\_MATCHING\_CASES.** This system pattern is used to require a pattern to match a minimum number of sentences. The purpose is to prevent overwhelming the graph and the genetic optimization

algorithm with patterns that contribute little evidence of synonym pairs.

During training a value of 4 was found to give good results.

- **SNLASynonyms.MAXIMUM\_PATTERN\_MATCHING\_CASES.** This system parameter is used to screen out patterns that are excessively general and match too many sentences. For example “\$GENE\$ and \$GENE\$” will match any two tokens that could be gene names, separated by the word “and”. This is necessary both to reduce false positives and improve the efficiency of evaluation during the genetic optimization step. A value of 1000 was chosen after examining the frequency of matches for patterns found during training. In the current version of the system, this parameter is an absolute number, which is an adequate approach because all three data sets had about the same number of sentences. However in a future version of the system intended for use on various size data sets, this parameter must be converted into a percentage of sentences. The current filtered data sets contain approximately 150,000 sentences, so this works out to a threshold of approximately 0.7% of the data set size.

Another pattern selection parameter, **PATTERNS\_PER\_ITERATION**, controls how many patterns are considered at one time:

- **SNLASynonyms.PATTERNS\_PER\_ITERATION.** The maximum number of patterns to evaluate with the genetic optimizer at each iteration of the main SNLA loop. For each iteration, the algorithm uses new high confidence pairs to search the text for new patterns. When the number of total patterns found but not yet selected by the optimizer reaches the value of this parameter, the algorithm stops looking for new patterns and proceeds to the optimization step. Since the speed of the optimization step is very sensitive to the number of patterns being evaluated at once. A value of 150 was chosen as a balance between speed and broad coverage of a variety of patterns.

### *Synonym Inference Required Co-occurrences*

As was previously stated, synonyms that are inferred across the synonymous relationship of two or more other synonyms have a higher error rate than synonyms detected explicitly in the text. To compensate for this the algorithm includes a parameter `SNLASynonyms.MINIMUM_LINK_TO_FOLLOW`. This is the minimum number of co-occurrences required for a synonym pair to be used in logically inferring other synonym pairs. While a value of 1 (uses all synonym pairs during logical inference) results in finding the most synonyms, the error rate during training was very high. A value of 2 gave a good balance between precision and number of inferred pairs discovered.

### *Genetic Optimization Parameters*

A genetic (or evolutionary) algorithm is used to find the optimal (highest quality) set of patterns during each iteration. For the genetic optimization procedure, the samples (genomes in typical genetic algorithm terminology) are lists of flags (genes, in typical genetic algorithm terminology) of whether or not to include a candidate pattern in the optimal pattern set. For each generation, there is a best sample, which gives the combination of patterns to include for the best quality score. The optimizer runs for multiple generations, until no better solutions can be found.

The genetic algorithm used here is a variation of the canonical genetic algorithm that uses rank-order based selection pressure. It is used simply as a combinatorial optimizer. This version was chosen because it works well and is simple to implement. Other genetic algorithm variants and other approaches to combinatorial optimization likely would work just as well.

The genetic optimizer has several parameters that determine how quickly the algorithm converges to a local optimum and the criteria for exiting the optimization loop. While the parameters will be described and defined here, an in-depth presentation of the effects of each parameter is beyond the scope of this work. Most parameters were chosen by rules-of-thumb given in published work and were not specifically tuned for the SNLA system. See papers by Whitley for more detailed technical information on genetic algorithms in general and the canonical and rank-order based version used here [110, 112].

- **SNLASynonyms.RANDOM\_SEED.** A random seed for initializing the population sample and randomly assigning candidate patterns to the pattern flags.
- **SNLASynonyms.NUM\_GENOMES\_TO\_FREE\_PASS.** The number of top scoring samples to pass unmutated from one generation to the next. Given the rule-of-thumb value of 1.
- **SNLASynonyms.MUTATION\_RATE.** The probability that the inclusion status of a particular pattern in a particular sample will change during reproduction from one generation to the next. Given the rule-of-thumb value of 0.01.
- **SNLASynonyms.CROSSOVER\_RATE.** The probability that a pair of samples will recombine via the crossover operation during reproduction from one generation to the next. Given the rule-of-thumb value of 0.90
- **SNLASynonyms.NUM\_GENERATIONS.** The maximum number of generations to run the optimization during each iteration of the SNLA algorithm. During system training a value of 100 was found to be adequate.
- **SNLASynonyms.MAX\_NO\_IMPROVEMENT\_ITERATIONS.** The number of generations to run the optimization without seeing any improvement in the best quality score. During system training a value of 5 was found to be adequate.
- **SNLASynonyms.GENERATION\_SIZE.** The number of samples to include in each generation. During system training a value of 200 was found to be adequate.

- **SNLASynonyms.PATTERN\_INCLUSION\_PROBABILITY.** The probability that any given pattern is included in a sample. This is used while initializing the population for the genetic algorithm. Since each pattern flag is essentially a true/false about whether a given pattern is included in the sample, one might think that this parameter should be set to 0.50. However, after optimization most patterns are rejected, so using 0.50 results in the optimization spending a lot of time waiting for the mutation operator to set a flag to false. A parameter value of 0.25 was found to speed up the optimization without diminishing the quality score of the final result.
- **SNLASynonyms.RANK\_SELECTION\_PRESSURE.** This parameter defines how rank order is transformed into selection pressure. After computing quality, each sample is sorted into rank, ordered by quality. The probability,  $p$  that a sample will be selected to participate in the next iteration is computed as:

$$p = \frac{1.0}{RANK\_SELECTION\_PRESSURE * rank + 1.0}$$

(Equation 9)

During system tuning a value of 0.50 was found to give good results.

## VII. EXPERIMENTAL DESIGN

To provide insight into the research question, the experiment was done in three steps. The first step was to develop, debug and tune the algorithm detailed in the previous section on the training and validation data sets. In the second step, the synonym extraction algorithm was run on the MEDLINE records in the test set. In the third step, the quality of the extracted synonym list was evaluated by validating the synonymy of each extracted pair, and then computing metrics such as the rates of precision and recall for each iteration of the algorithm. See Figure 8.

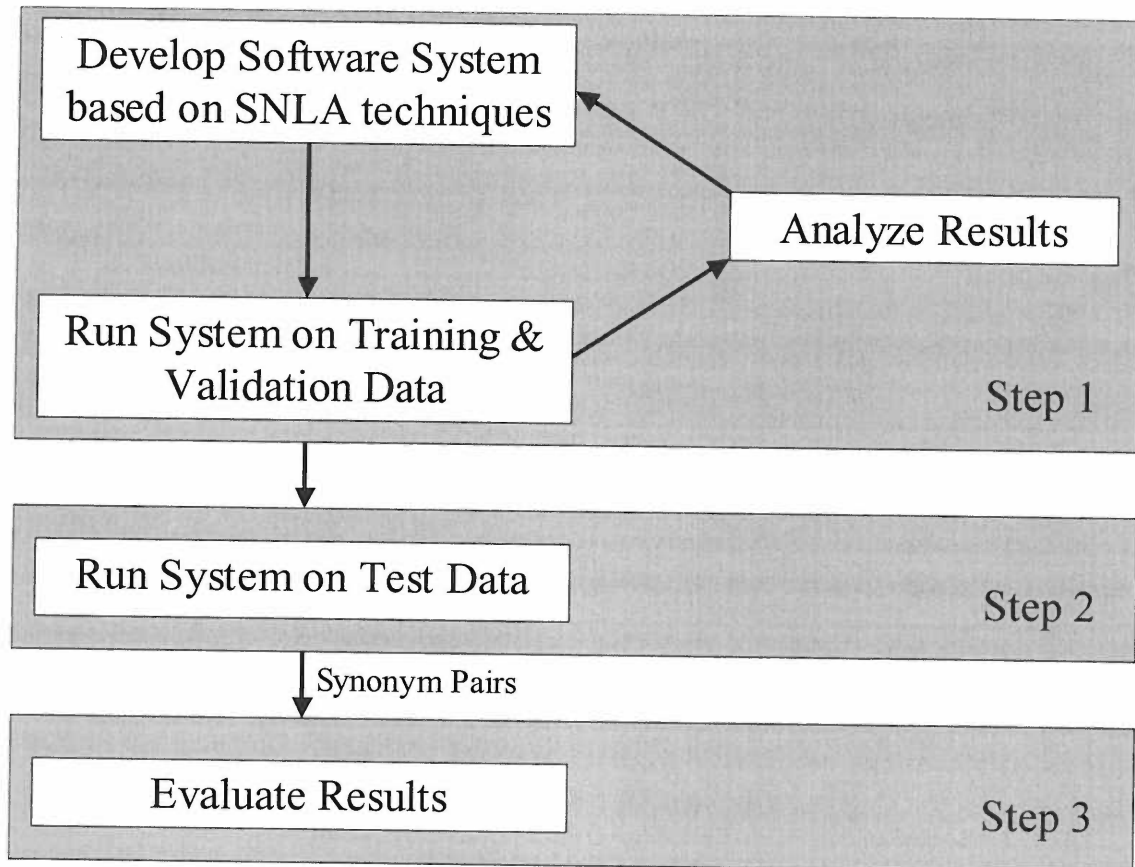


Figure 8: Experimental Design

## Data Sets

The training, validation, and test data sets used in this experiment consist of sentences extracted from approximately 50,000 abstracts from a year's worth of MEDLINE records containing the word "gene." Abstracts from 2001, 2002, and 2003 served as training, validation, and test sets, respectively. Originally it was planned to use the test set from the 2003 Text Retrieval Conference (TREC) genomics track [113], however the test collection did not contain enough records to separate into separate training, validation, and test sets.

After downloading the MEDLINE records from PubMed, the records were parsed to extract the abstract field. The abstracts were then separated into sentences using a simple, lexically based sentence boundary detection algorithm. Finally, the sentences were screened to remove non-contributing sentences as described previously in the section on *Filtering Out Non-Contributing Sentences*. This resulted in the three data sets used in this experiment. Size properties of these data sets are listed in Table 1.

| Data Set                                 | Size in Kilobytes | Number of Sentences |
|--|-------------------|---------------------|
| medline-gene-2001-sentences-screened.txt | 25,251            | 140,591             |
| medline-gene-2002-sentences-screened.txt | 26,665            | 148,525             |
| medline-gene-2003-sentences-screened.txt | 27,410            | 152,432             |

Table 1: Size Properties of Data Sets

## Gold Standards

This experiment requires gold standards for both precision and recall. Since gene name synonyms are part of the curated data available in the on-line genomics databases, these served as the basis for both gold standards. Slightly different methods were used to create the precision and recall gold standards.

### *Precision Gold Standard*

The approach used here was to download a snapshot of several genomics databases available on-line, extract out the name, alias, and synonym fields, and combine them into a single gold standard for use in the computation of recall and precision. Since the databases do not contain all synonyms in common use, and orthographic variations (e.g. “WAF1” and “WAF-1”) are usually missing, during training and validation extracted candidate synonym pairs that were not found in the precision gold standard were manually reviewed for pairs that were likely to be correct (e.g. “CONNEXIN32” and “CX-32”), and these pairs were checked by reviewing MEDLINE for supporting information in the titles and abstracts. Manually verified pairs were added to the precision gold standard for use in scoring the results from the test data. Information on the databases and fields used are listed in Table 2.

| <b>Database</b> | <b>Organism</b> | <b>Fields</b>  | <b>Download Date</b> |
|-----------------|-----------------|--|----------------------|
| Flybase         | Drosophila      | ALL (synonym only file)  | 2004/01/12           |
| Genew           | Human           | SYMBOL,<br>PREVIOUS_SYMBOL,<br>ALIAS   | 2004/01/12           |
| LocusLink       | Multiple        | OFFICIAL_SYMBOL,<br>PREFERRED_SYMBOL,<br>ALIAS_SYMBOL,<br>OFFICIAL_NAME,<br>PREFERRED_NAME | 2004/01/12           |
| MGI             | Mouse           | ACCESSION_ID,<br>MARKER_SYMBOL,<br>SYNONYMS  | 2004/01/22           |
| SGD             | Yeast           | LOCUS_NAME,<br>OTHER_NAMES,<br>GENE_PRODUCT,<br>ORF_NAME                                   | 2004/01/22           |
| SwissProt       | Multiple        | GN (both AND and OR)   | 2003/12/10           |

**Table 2: Precision Gold Standard Databases**



### *Recall Gold Standard*

Creation of a recall gold standard for a knowledge extraction task is a difficult problem. Typically an accurate gold standard requires multiple experts to agree on definitions and then manually review the literature for the information in question, comparing multiple expert opinions and computing inter- and intra-rater agreement. This research did not have the necessary expert resources available to use this method; a simpler method based on that used by Yu and Agichtein in their gene and protein synonymy research was used instead [19].

To approximately sample the synonym pairs that could be extracted from the MEDLINE sentence test sets, the synonym pairs extracted from the SWISSPROT database were compared to the sentences in the test collection. If both symbols of a synonym pair extracted from SWISSPROT were present in a sentence in the test collection, that synonym pair was included in the recall gold standard. This resulted in a recall standard set of 483 synonym pairs. While this biases the recall gold standard towards the genes and protein names and formats present in SWISSPROT, the bias is independent of any feature of the algorithm. Also, using a recall gold standard construction method like that of Yu and Agichtein facilitates comparison of results. These comparisons will be shown later.

Note that even though a pair of gene synonymous names from SWISSPROT may be present in a single sentence of the test set, it may be impossible for this or any other pattern-based algorithm to extract the pair. The synonyms could be separated by too many words, or the synonym pair may not occur in a repeated pattern. Nevertheless, a

recall gold standard constructed by this method provides a useful benchmark. The error analysis presented later sheds light on some of these issues complicating recall.

### **Application of Algorithm to the Data Sets**

After development of the software was complete and the system had been debugged and tuned on the training data set, the software system was run on the test data set without changes. This produced a list of gene and protein name synonym pairs, along with co-occurrence counts for each pair, and an indication of whether a synonym pair was detected in the text or inferred logically from the network structure.

### **Computation of Performance Measures**

Computation of precision and recall was performed for the synonyms extracted after each iteration of the algorithm. Synonym pair extraction was cumulative. The set of extracted synonym pairs at each set included the pairs extracted at all prior iterations.

#### *Computation of Precision*

To compute precision at each iteration, each extracted synonym pair was assigned a value of correct or incorrect, based on whether the pair is present in the precision gold standard database. Pairs marked incorrect were reviewed manually, and the reviewed pairs deemed likely to be correct were subjected to an additional database and literature review. If the pair was found to be correct based on the existence of an orthographically similar synonym (e.g., a missing or added hyphen) in the on-line databases, or there was clear evidence of synonymy in the MEDLINE database title and abstract fields (e.g., “The PEA3/E1AF/ETV4 gene encodes...”), these synonym pairs were added to the gold standard database. Since many synonyms commonly in use are not included in the on-line database (e.g., “IL-5” and “Interleukin-5”), this secondary review process was necessary

to correct the many false negatives that would occur if only curated genes symbols were used in the gold standard.

Given the counts of extracted and correct synonym pairs at each iteration, the value for precision can be easily computed by the following formula:

$$precision_{iteration} = \frac{correct\ extracted\ synonym\ pairs}{number\ of\ synonym\ pairs\ extracted}$$

(Equation 10)

#### *Computation of Recall*

To compute recall at each iteration, each synonym pair in the recall gold standard was assigned a value of found verses unfound by performing an exact string match comparison with each extracted synonym pair. A recall gold standard pair was given the value of found if the exact pair was present in the set of extracted synonym pairs, and unfound otherwise. Given the counts of found and unfound gold standard synonym pairs at each iteration, the value for recall can be easily computed by the following formula:

$$recall_{iteration} = \frac{recall\ pairs\ found}{number\ of\ known\ correct\ pairs}$$

(Equation 11)

#### *Computation of F-score and Maximum Performance*

The F-score (the harmonic mean of precision and recall) was computed at each iteration. Maximum performance was deemed at the point of maximum F-score of confidence. While the limitation of any single number summary score must be kept in mind, the F-score is used here as a single measure for comparison across iterations and algorithms [56]. The F-score for a given iteration is computed as:

$$Fscore_{iteration} = \frac{2 * precision_{iteration} * recall_{iteration}}{precision_{iteration} + recall_{iteration}}$$

(Equation 12)

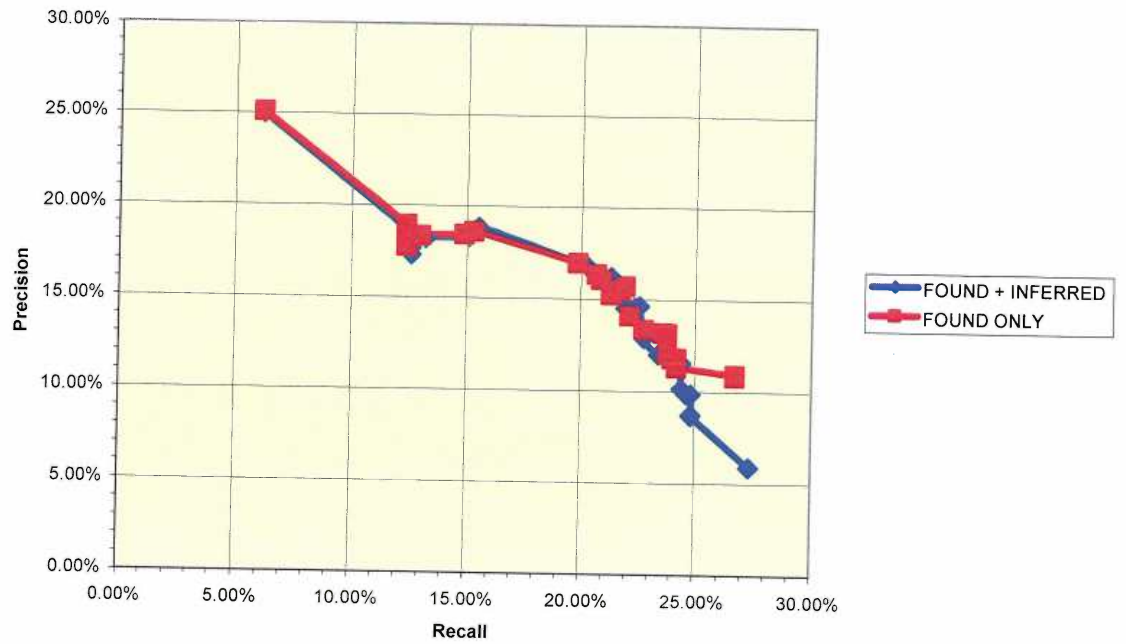
## VIII. RESULTS

The experiment produced two kinds of results: performance measures and error analysis. The performance measures summarized the quality of the extracted information, while error analysis provided insight into the strengths and weaknesses of the approach. Performance measures are presented first, followed by a comparison with the prior work on synonym extraction done by Yu and Agichtein, and then an error analysis.

### **Performance Measures**

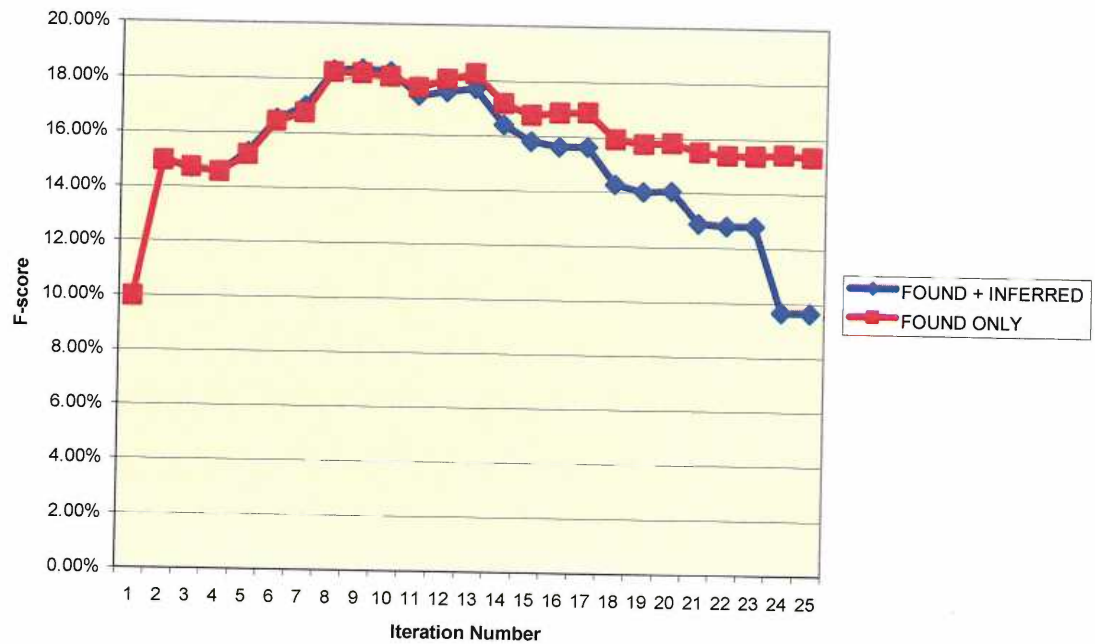
The performance measures include the precision, recall, and F-score of the extracted synonym pairs, as well the absolute and relative number of correct pairs extracted at each iteration. Figure 9 shows the precision verses recall of the extracted synonym pairs, starting with the first iteration at the left-most point and continuing to the 25<sup>th</sup> iteration at the right-most point. The graph includes plots of both found (synonym pairs explicitly found in the text by the patterns) plus inferred synonyms (pairs inferred by the graph traversal algorithm) as well as plotting only the synonyms that were explicitly found in the text.

The first iteration achieved a precision of about 25% (24.95% for the FOUND+INFERRED pairs, 25.05% for FOUND ONLY), at a recall of 6.21% (both FOUND+INFERRED and FOUND ONLY). Precision decreased and recall increased practically monotonically over the 24 following iterations to a high recall of 27.33% (for FOUND+INFERRED, 26.71% for FOUND ONLY), and a precision low of 5.87% (for FOUND+INFERRED, 10.81% for FOUND ONLY).



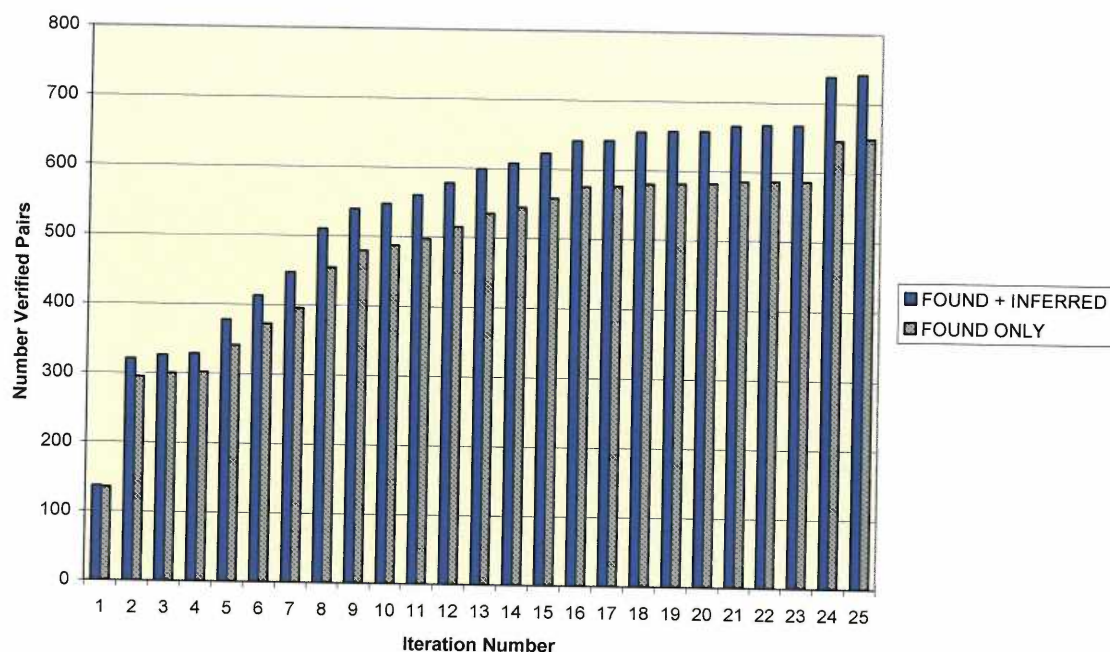
**Figure 9: Precision versus Recall by Iteration**

Figure 10 presents the computed F-score at each iteration, and again the graph includes plots of both found plus inferred synonyms as well as found only. The F-score at iteration one is 9.91% (for FOUND+INFERRED, 9.92% for FOUND ONLY), and rises to a maximum of 18.35% (for FOUND+INFERRED, 18.19% for FOUND ONLY) with a precision of 16.18% and recall of 21.33% (FOUND+INFERRED) at iteration 9, gradually falling off during subsequent iterations. It can be seen that the use of inference in finding synonyms does not significantly hurt the algorithm's overall accuracy (as measured by the F-score) until approximately iteration 15.



**Figure 10: SNLA F-score by Iteration**

The absolute number of correct pairs extracted is presented in Figure 11. As is enforced by the algorithm, the number of verified pairs grows monotonically with the iteration number. Including pairs found using the inference capability of the network consistently found more pairs than not using the inference capability. At the maximum F-score (iteration 9) the system using FOUND+INFERRED synonyms extracted 539 correct synonym pairs, including only the FOUND pairs yielded 479 synonym pairs. The approximately 10% (12.5% at iteration 9) difference in extracted pairs is fairly consistent across all iterations after the initial iteration.

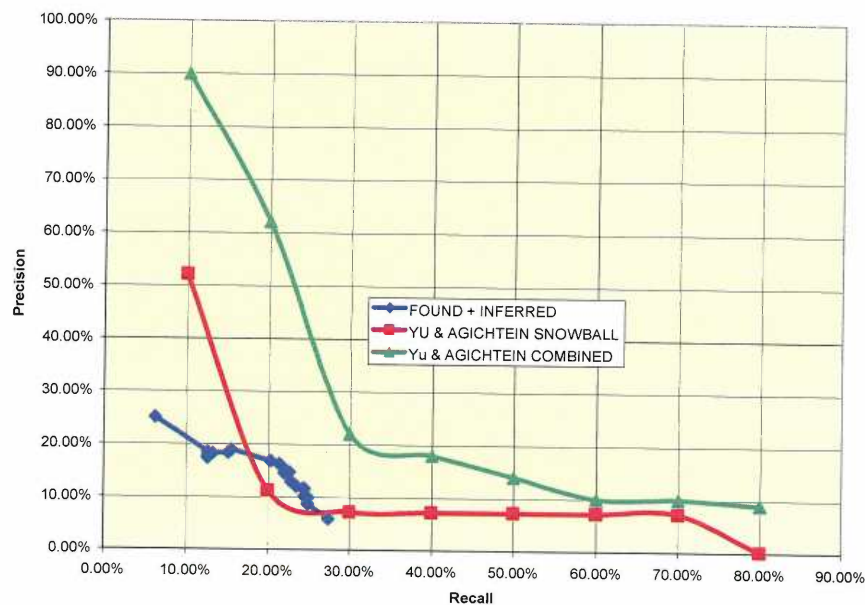


**Figure 11: SNLA Number Verified Pairs by Iteration**

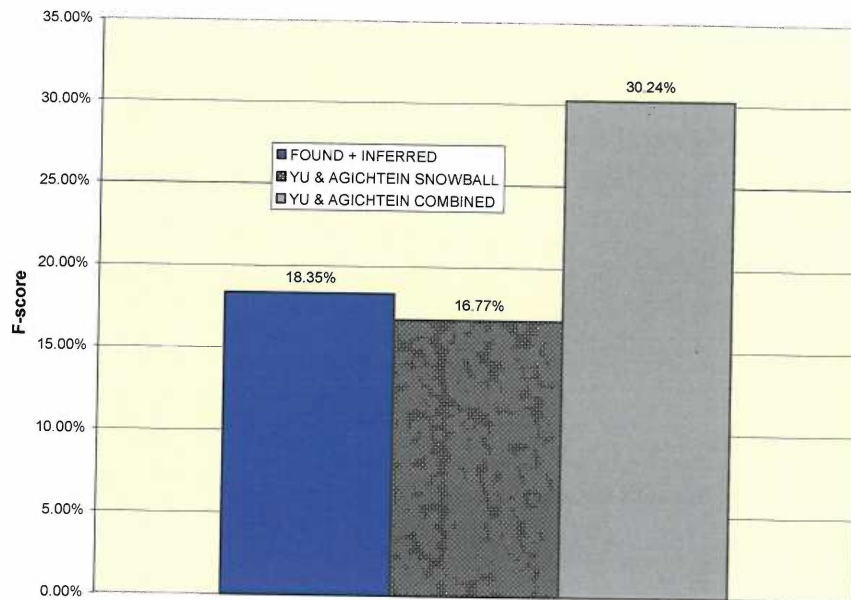
Figures 12 and 13 compare the results of the SNLA system with those of Yu and Agichtein. The results of their Snowball (their best single algorithm) and Combined (their best overall approach) systems were interpolated from published graphs [19], and re-plotted here for comparison. After the first few iterations of SNLA, the recall and precision of SNLA compares favorably with that of the Snowball system, with SNLA having better recall and precision than Snowball during iterations 8 (precision 16.73%, recall 20.29%) through iteration 23 (precision 8.64%, recall 24.84%). The maximum F-score obtained for SNLA (18.35%) is comparable with that of Snowball (16.77%), but significantly less than that of the Combined system (30.24%). The combined system of Yu and Agichtein had superior accuracy to any single system including SNLA, Snowball, and the three other extraction systems studied by Yu and Agichtein (Similarity, SVM, and GPE).



It should be noted that the system of Yu and Agichtein provided a confidence score that could be used as a precision/recall tradeoff threshold. Yu and Agichtein found that using only two iterations of Snowball gave the best results, and those are the results that are plotted here. The current version of SNLA does not provide a tradeoff threshold, and score of extraction over cumulative iterations are shown here for comparison. Even though the details by which the system precision/recall tradeoff are made differ, the graph still serves as a valid comparison of the results of the three systems. It should also be noted that by using only two iterations, the majority of the results of the Snowball algorithm was based on pairs detected using the initial seed pairs, and not on the iterative use of discovered high confidence pairs in detecting new patterns.



**Figure 12: SNLA Precision and Recall Comparison with Prior Work**



**Figure 13: SNLA Maximum F-score Comparison with Prior Work**

Another measure of system performance is the amount of knowledge extracted per unit of instance knowledge input to the system. This can be interpreted as a measure of how efficient the algorithm uses the seed data. Here, the input instance knowledge is the number of seed pairs. During system tuning on the training data, SNLA was found to work well with only eight seed pairs. These same eight seed pairs established for the training data were used on the test data. The seed pairs used are presented in Table 3.

**Table 3: Seed Pairs used by SNLA**

| Seed Number | First Synonym | Second Synonym |
|-------------|---------------|----------------|
| 1           | CIP1          | WAF1           |
| 2           | LPS           | TLR4           |
| 3           | CD82          | KAI1           |
| 4           | IGF2R         | M6P            |
| 5           | MMAC1         | PTEN           |
| 6           | DR5           | KILLER         |
| 7           | CCN3          | NOV            |
| 8           | ASF           | SF2            |

Figure 14 compares the number of correct extracted pairs to the number of seeds used for SNLA and for Yu and Agichtein's Snowball and Combined systems. Results are shown at the point of maximum F-score in order to provide a consistent comparison. The SNLA system (FOUND+INFERRED) used 8 seed pairs, while the Snowball and Combined systems used 650. SNLA extracted 539 correct synonym pairs at the maximum F-score, while Snowball and Combined extracted 700 and 950 respectively. Computing the ratio of correct pairs divided by number of seeds used gives a ratio of 67.38 for SNLA, with the Snowball and Combined systems having much smaller ratios of 1.08 and 1.46 respectively.

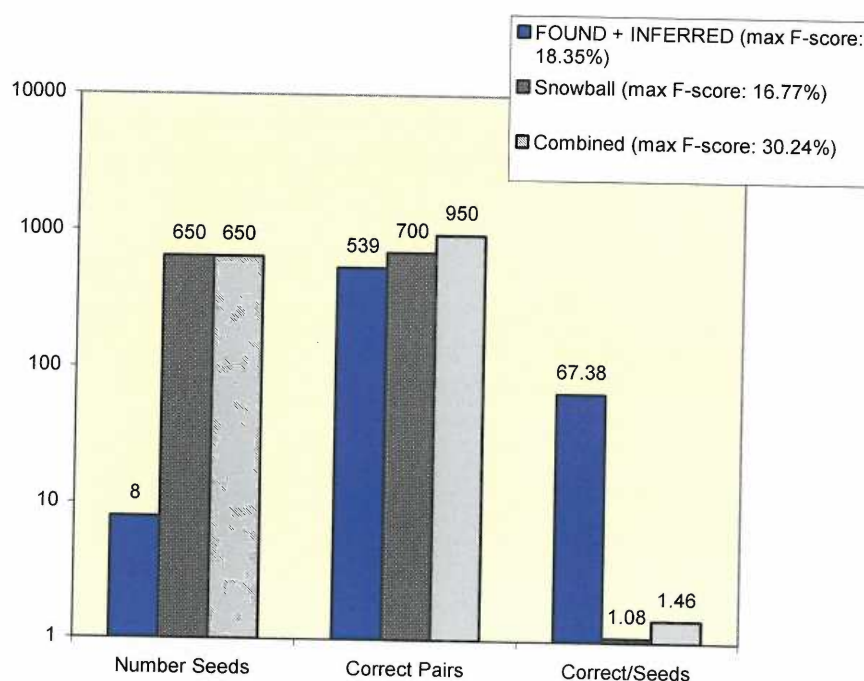


Figure 14: Comparison of Number Seeds, Verified Pairs, and Ratio

## Error Analysis

Error analysis provides insight into the strengths and weaknesses of the SNLA approach, as well as similar pattern-based approaches. Two kinds of errors were studied, precision errors and recall errors. Precision errors occurred when the algorithm extracted symbol pairs that were later not verified as synonyms by the precision gold standard data set. These are false positives. Recall errors occurred when the algorithm failed to extract symbol pairs present in the recall gold standard data set. These are false negatives. For consistency and to facilitate later comparison, precision and recall errors were studied at the point of maximum F-score, iteration 9.

Recall errors for the SNLA algorithm were categorized into two pre-defined and mutually exclusive categories, *No Matching Pattern*, and *Pattern Not Accepted*. The *No Matching Pattern* error category included all recall errors for which the SNLA pattern generation routines failed to identify a pattern that matched the given pair in the abstract text. *Pattern Not Accepted* errors included those recall errors for which a matching pattern was found, but the matching pattern or patterns were not accepted during the pattern selection optimization step. All recall errors fell into one of these two categories.

Table 4 presents a summary of the recall error analysis. 100 recall errors were manually reviewed in order to determine the relative frequencies of the two errors with a confidence interval of  $\pm 10\%$  at an alpha of 95%. The majority, 65% of recall errors, can be attributed to the system failing to generate a pattern that matched the recall synonym pair. The remaining 35% of recall errors are due to matching patterns not being accepted by the pattern optimization step.

Table 4: SNLA Recall Error Analysis

| Recall Error Type           | Frequency | Percentage | Low 95% C.I. | High 95% C.I. |
|-----------------------------|-----------|------------|--------------|---------------|
| <i>No Matching Pattern</i>  | 65        | 65%        | 55%          | 75%           |
| <i>Pattern Not Accepted</i> | 35        | 35%        | 25%          | 45%           |

Precision errors were categorized by first reviewing a small random sample of 20 errors. From this pilot set of errors, a set of mutually exclusive precision error categories was determined by inspection. The resulting set of six error categories was then applied to an additional random sample of 100 precision errors. The six error categories are:

- **Not a Gene Name.** One or both symbols are not the name of a gene, allele, mutation, or gene family.
- **Partial Gene Name.** One or both symbols are part of an incompletely extracted gene name pair.
- **Biochemically Related.** Genes have been studied together as interacting within the context of a biochemical mechanism in some organism or two distinct genes from the same functionally related family.
- **Unrelated Genes.** Two complete gene names but unable to establish family or biochemical relationship by reviewing the test data set or MEDLINE.
- **Mutation Variants.** Two mutation names for the same gene but nonspecific for that gene. This category distinguishes between allele or mutation names that are recognizable to a particular gene, verses those that are more generic and used only within a specific abstract. The category is somewhat of a judgment call, in general short mutation names were considered non-specific. The errors assigned to this category appeared non-specific when compared to the genes studied and included: P-0.48/P-0.52, CYS106ALA/CYS7ALA, K-

RAS/TMDELTA4A, DELTACDTABC/DELTALTXA, and D138E/FECR. It is conservation to assign these errors to this category. The most reasonable other category would be to call these synonyms correct.

- **Correct.** A correct gene synonym pair not found by gold standard dataset, found by later abstract review.

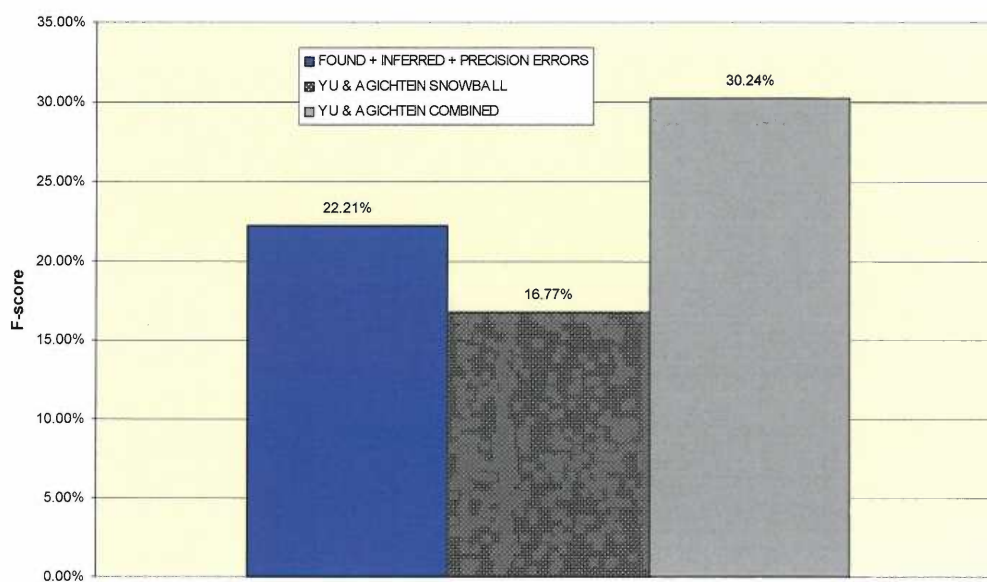
The results of the precision error analysis are presented in Table 5. By far, the most commonly occurring error was a pair of gene symbols being chemically or biologically related but distinct, non-synonymous entities. These errors accounted for 48% of the total. The next most common error, occurring 28% of the time, resulted when one or both of the extracted pair of symbols were not a gene or protein name or symbol. The remaining errors were much less common, and included when one or both extracting symbols consisted of a partial or incomplete gene name (9%), two extracted symbols representing mutation names non-specific to an individual gene (5%), and two complete gene names found but no mention was found in the data set of in MEDLINE of a biochemical (3%).

### **Incorporation of Error Analysis in Performance Results**

Interestingly, 7% of precision errors were later determined to be false negatives, that is, the synonym pair was determined by manual inspection to be correct but was not part of the gold standard data set. These correct pairs have not been added back into the performance results already shown. Incorporating the additional estimated 7% (95% C.I.: 0%-17%) correct extracted synonym pairs gives an estimated precision of 23.18% and an estimated peak F-score of 22.21%. A comparison of this performance estimate with the prior work of Yu and Agichtein is shown in Figure 15.

**Table 5: SNLA Precision Error Analysis**

| Precision Error Type         | Frequency | Percentage | Low 95% C.I. | High 95% C.I. |
|------------------------------|-----------|------------|--------------|---------------|
| <i>Not a Gene Name</i>       | 28        | 28%        | 18%          | 38%           |
| <i>Partial Gene Name</i>     | 9         | 9%         | 0%           | 19%           |
| <i>Biochemically Related</i> | 48        | 48%        | 38%          | 58%           |
| <i>Unrelated Genes</i>       | 3         | 3%         | 0%           | 13%           |
| <i>Mutation Variants</i>     | 5         | 5%         | 0%           | 15%           |
| <i>Correct</i>               | 7         | 7%         | 0%           | 17%           |



**Figure 15: Maximum F-score Comparison including Precision Errors**



## IX. DISCUSSION AND LIMITATIONS

The results of this thesis demonstrate that the SNLA method compares well to other single methods of synonym extraction and is a useful general approach to knowledge extraction. The method is highly efficient in its use of seed pairs compared to other methods. This may be an advantage in situations where large numbers of seed pairs are difficult or expensive to collect.

It was determined that using eight initial seed pairs was adequate during system tuning using the training dataset. It was observed that the performance was largely stable for initial numbers of seed pairs between 8 and 32. This suggests that an initial “critical mass” of seed pairs was necessary to get the process started. Beyond the critical number, the algorithm automatically found additional common seeds. Including additional common synonym pairs as seeds simply gave as input high confidence pairs that the algorithm could find on its own.

Optimizing the network structure based on the quality metric of the overall network MCC/MNCC ratio was an effective way to pick the best text patterns for gene synonym pair extraction. Using the symbolic network to support inference of synonym pairs improved both the recall at any given iteration as well as the absolute number of synonym pairs discovered. The FOUND+INFERRED system consistently found approximately 10% more verified pairs at any iteration than the FOUND pairs alone. Based on Figure 10 it can be seen that, while there is some cost in precision for these additional verified pairs, the cost is modest until well past the iteration giving the peak F-score. The inference ability of SNLA adds to its capability as a tool in knowledge



discovery, and helps extract pairs additional synonym pairs earlier in the iterative process and beyond those found strictly in the text.

The confidence measure was found to be an effective way to separate pairs more likely to be true synonyms, and therefore useful for pattern extraction and generation, from pairs less likely to be true synonyms and therefore not useful for pattern creation. However, the confidence measure did not have fine enough granularity and was not found to be a useful measure to sort and therefore prioritize the pairs. Instead a measure based on the node pair MCC/MNCC product was used as a sorting value. This was an adequate approach for the current work, but further investigation is needed to find more accurate, finer-grained, and more comparable measures of knowledge confidence based on the structure of the graph. One possibility is to collapse the synonym pair into a single node and compute the MCC/MNCC for that combined node.

The relative frequencies of the two types of recall errors present interesting evidence suggesting the possibility of making general observations about pattern-based text relationship mining systems. Two-thirds of the recall errors were due to the system not having discovered a pattern that matched the non-recalled pair, and only one-third of errors were due to the system having found a matching pattern, rejecting it based on the network metric criteria. The current system used a large number of very specific patterns based on the text surrounding high confidence gene symbol pairs. The Snowball system by Yu and Agichtein used more flexible patterns, allowing “fuzzy” matching based on the relative importance of word in a pattern. The two different systems performed similarly, leading one to think that there may be some inherent limitation of the pattern-based approach to uncovering gene relationships. The textual context of interesting

biological relationships may not be specific enough to significantly improve performance. Certainly, more work is needed in this area before drawing strong conclusions.

While the method does not perform as well as the combined method of Yu and Agichtein, it does perform at a level at least equivalent to the best of their component methods. It is thought that the Combined system achieves improved performance by using a variety of algorithms to double-check one another. Including SNLA into the Combined system of Yu and Agichtein may result in further improvement in gene name synonym extraction.

The full text test collection used by Yu and Agichtein is not in the public domain and was not made available to the investigator. One of the major limitations of this study as well as future work on synonym extraction is the lack of availability of a full text test collection of adequate size, and the inability to use the same test collection as previous investigators in order to facilitate comparison. The MEDLINE abstracts were used because they are plentiful and readily available. While prior investigators have stated that full text articles are better sources data for the extraction of gene name synonyms [35], it is encouraging to find that applying the SNLA method to abstracts produced comparable results.

Since there is no standard test collection for gene symbol synonym extraction research and no absolute gold standard for recall, the recall standard used is an approximation. The method of constructing a recall standard used in this work facilitated comparison with prior work in the field. However, it is by nature a biased sampling

method, and does not completely characterize the recall capabilities of current knowledge extraction systems as compared to manual expert review.

Another limitation of the evaluation method is the lack of multiple experts available to review the results. The principal investigator, an M.D., made all of the precision and recall judgments based on the criteria described in the section on experimental design. The investigator used a consistent approach to determine gene name synonymy, and erred on side of designating pairs as non-synonyms in ambiguous cases. When MEDLINE text needed to be reviewed directly to verify synonymy, multiple records were examined looking for consistent use of synonyms. Ambiguous cases were designated as non-synonyms. With only a single investigator, there was no ability to measure inter-rater agreement. To compensate for this, results were evaluated with a slight intentional bias towards marking synonym pairs as incorrect. This is supported by the precision error analysis that shows that 7% of the sampled errors were actually correct.

The performance of the current system is limited somewhat by the simple orthographic approach used for named-entity recognition. Gene names and symbols were required to be a single string delimited by spaces and other punctuation characters. Not all gene names fit this description, although the gene name pairs extracted for the recall gold standard from the SWISSPROT did not use any unusual characters and met this requirement.

Precision error analysis showed that approximately 28% of precision errors are due to a non-gene or protein symbol being treated as a gene or protein. Another 9% of precision errors are due to an incomplete portion of a gene symbol being identified as a gene symbol. These two categories together represent failure of NER and account for 37% of

precision errors. Current state-of-the-art performance of biological named-entity recognition is approximately 80%, however this figure assumes that token boundaries are given, and glosses over the complex problem of determining these boundaries [43, 75]. Nevertheless, using 80% as the measure of performance, it can be estimated that incorporating a state-of-the-art gene and protein named-entity recognizer into the system would decrease these errors to about 20%, a 17% reduction in errors. This would result in a precision at the peak F-score of about 27.12%, as compared to the current precision of 23.18%, an improvement of 17%.

The article titles were not used by the system. Some MEDLINE titles do contain gene name synonym pairs. Determining whether including titles would improve the system performance is an open question.

The results also support an interesting observation about the relative ease of a relationship text-mining task being inversely proportional to the specificity of the relationship. The largest source of precision errors was due to the algorithm extracting a related pair of gene symbols that were not synonyms. This type of error accounts for almost half (48%) of the precision errors. Several investigators have reported on knowledge extraction systems that extract pairs of related genes [47], sometimes followed by constructing a gene relationship network [24, 48]. While biochemical induction/suppression relationships are at least as specific as the synonym relationship, it is certainly a less specific task to extract pairs of gene names that are simply related in some general way. If the current system and results were to be left unchanged and the results re-evaluated with the goal of the simply extracting genes related in any manner, then half of the precision errors would actually be correct, and the precision would be

approximately 60%! While may be at first surprising that a system designed for one purpose should performed well on another task, it is clear that the synonym relationship is a specialization of the “generally related” relationship. The main point here is that extracting non-specific relationships between genes is not a particularly difficult task, and other more functionally based relationships should be used to meaningfully evaluate biological relationship extraction systems.

One way to improve system performance by reducing these biochemically related gene synonym pair errors would be to filter the results by removing known associated gene pairs. There are several on-line databases of gene relationship networks [114-117], and the information in these databases could be used as evidence of the genes being distinct and non-synonymous. While it is unlikely that this filtering could remove all of the false positives due to biochemically related genes, since this is the single largest source of error the improvement is likely to be significant.

## X. CONCLUSIONS AND FUTURE WORK

The current work has shown that SNLA is a useful method in extracting relationships from the biomedical literature. The current system could be improved by incorporating state-of-the-art named entity recognition, and by using richer data sources such as full text articles, and gene network databases. While the current system does not perform as well as the combined system of Yu and Agichtein, it performs as well as any individual method and with more accurate named entity recognition and known biochemically related pair post-filtering the system would perform even better. It is reasonable to expect that making these improvements and incorporating the SNLA system into the Combined system of Yu and Agichtein would result in an overall performance improvement in the state of the art of gene and protein synonym extraction.

There are many other applications of SNLA to mining the biomedical literature. Many inter-entity relationships, such as enhance/inhibit between drugs, biological substances, and diseases, and the promoter/suppressor relationships between genes could be modeled as graph structures and appropriate metrics created to measure the relevant network properties. Multiple separate networks can be created simultaneously and then used together during the logical inference step to extend the use of SNLA to multiple types and entities and multiple types of relationships between those entities.

Perhaps the most exciting application for SNLA is in mining the biomedical literature for hypothesis generation, such as that done by Swanson. While Swanson was limited to ABC-style relations between three entities, the SNLA network can support practically limitless intermediate inferences, limited largely by the confidence in the individual relationships. Future refinements will have to go beyond the simple method used in the

current work to determine which relationships were strong enough to support inference and model the chain of inference as a confidence path with each link reducing the confidence in the entire path by a fraction based on the uncertainty of the relationship.

Having the ability to infer useful hypotheses across several individual relationships has the exciting potential to accelerate the rate of medical progress and focus efforts on the most promising prospects. With the biomedical knowledge and the corresponding bibliome growing at an exponential rate, the raw material exists for computer assisted hypothesis generation. Further work on text mining and knowledge extraction will be necessary in order to harness this great resource to its full potential.

## REFERENCES

1. Hersh WR. Information retrieval : a health and biomedical perspective. 2nd ed. New York: Springer; 2003.
2. Medicine® USNLo, File Names, Record Counts, and File Size for 2004 MEDLINE® Baseline Database Distribution, 2004;  
[http://www.nlm.nih.gov/bsd/licensee/2004\\_baseline\\_med\\_filecount.html](http://www.nlm.nih.gov/bsd/licensee/2004_baseline_med_filecount.html), accessed June 23, 2004.
3. Mitchell JA, Aronson AR, Mork JG, Folk LC, Humphrey SM, Ward JM. Gene Indexing: Characterization and Analysis of NLM's GeneRIFs. Proc AMIA Symp 2003:460-4.
4. Friedman C, Kra P, Yu H, Krauthammer M, Rzhetsky A. GENIES: a natural-language processing system for the extraction of molecular pathways from journal articles. Bioinformatics 2001;17 Suppl 1:S74-82.
5. Balas EA, Boren SA. Managing Clinical Knowledge for Health Care Improvement. In: Yearbook of Medical Informatics: Schattauer; 2000. p. 65-70.
6. Altman DG, Goodman SN. Transfer of technology from statistical journals to the biomedical literature. Past trends and future predictions. Jama 1994;272(2):129-32.
7. Gorman PN, Helfand M. Information seeking in primary care: how physicians choose which clinical questions to pursue and which to leave unanswered. Med Decis Making 1995;15(2):113-9.
8. Weeber M, Klein H, Aronson AR, Mork JG, de Jong-van den Berg LT, Vos R. Text-based discovery in biomedicine: the architecture of the DAD-system. Proc AMIA Symp 2000:903-7.
9. Srinivasan P. MeSHmap: a text mining tool for MEDLINE. Proc AMIA Symp 2001:642-6.
10. Chang JT, Schutze H, Altman RB. Creating an online dictionary of abbreviations from MEDLINE. J Am Med Inform Assoc 2002;9(6):612-20.
11. Lindsay RK, Gordon MD. Literature-based discovery by lexical statistics. Journal of the American Society for Information Science 1999;50(7):574-587.
12. Swanson DR. Medical literature as a potential source of new knowledge. Bull Med Libr Assoc 1990;78(1):29-37.
13. Srinivasan P, Rindflesch T. Exploring text mining from MEDLINE. Proc AMIA Symp 2002:722-6.



14. U.S. National Library of Medicine. Fact Sheet Medical Subject Headings (MeSH®), 2002; <http://www.nlm.nih.gov/pubs/factsheets/mesh.html>, accessed June 16, 2003.
15. Mendonca EA, Cimino JJ. Automated knowledge extraction from MEDLINE citations. *Proc AMIA Symp* 2000:575-9.
16. Lober WB, Karras BT, Wagner MM, Overhage JM, Davidson AJ, Fraser H, et al. Roundtable on bioterrorism detection: information system-based surveillance. *J Am Med Inform Assoc* 2002;9(2):105-15.
17. Liu H, Aronson AR, Friedman C. A study of abbreviations in MEDLINE abstracts. *Proc AMIA Symp* 2002:464-8.
18. Pustejovsky J, Castano J, Cochran B, Kotecki M, Morrell M. Automatic extraction of acronym-meaning pairs from MEDLINE databases. *Medinfo* 2001;10(Pt 1):371-5.
19. Yu H, Agichtein E. Extracting synonymous gene and protein terms from biological literature. *Bioinformatics* 2003;19(Suppl. 1):i340-i349.
20. Yu H, Hripcsak G, Friedman C. Mapping abbreviations to full forms in biomedical articles. *J Am Med Inform Assoc* 2002;9(3):262-72.
21. Fukuda K, Tamura A, Tsunoda T, Takagi T. Toward information extraction: identifying protein names from biological papers. *Pac Symp Biocomput* 1998:707-18.
22. Raychaudhuri S, Altman RB. A literature-based method for assessing the functional coherence of a gene group. *Bioinformatics* 2003;19(3):396-401.
23. Raychaudhuri S, Chang JT, Sutphin PD, Altman RB. Associating genes with gene ontology codes using a maximum entropy analysis of biomedical literature. *Genome Res* 2002;12(1):203-14.
24. Raychaudhuri S, Schutze H, Altman RB. Using text analysis to identify functionally coherent gene groups. *Genome Res* 2002;12(10):1582-90.
25. Swanson DR. Fish oil, Raynaud's syndrome, and undiscovered public knowledge. *Perspect Biol Med* 1986;30(1):7-18.
26. Swanson DR. Complementary structures in disjoint science literatures. In: *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*; 1991; Chicago, Illinois, United States: ACM Press; 1991. p. 280--289.
27. Weeber M, Vos R, Klein H, De Jong-Van Den Berg LT, Aronson AR, Molema G. Generating hypotheses by discovering implicit associations in the literature: a case report

- of a search for new potential therapeutic uses for thalidomide. *J Am Med Inform Assoc* 2003;10(3):252-9.
28. Harris ZS. Mathematical structures of language. New York,: Interscience Publishers; 1968.
29. Harris ZS. The structure of science information. *J Biomed Inform* 2002;35(4):215-21.
30. Ghanem MM, Guo Y, Lodhi H, Zhang Y. Automatic scientific text classification using local patterns: KDD Cup 2002 (task 1). *ACM SIGKDD Explorations Newsletter* 2003;4(2):95-96.
31. Regev Y, Finkelstein-Landau M, Feldman R. Rule-based extraction of experimental evidence in the biomedical domain: the KDD Cup 2002 (task 1). *ACM SIGKDD Explorations Newsletter* 2003;4(2):90-92.
32. Shi M, Edwin DS, Menon R, Shen L, Lim JYK, Loh HT. A machine learning approach for the curation of biomedical literature-KDD Cup 2002 (task 1). *ACM SIGKDD Explorations Newsletter* 2003;4(2):93-94.
33. Jain NL, Friedman C. Identification of findings suspicious for breast cancer based on natural language processing of mammogram reports. *Proc AMIA Annu Fall Symp* 1997:829-33.
34. Jain NL, Knirsch CA, Friedman C, Hripcsak G. Identification of suspected tuberculosis patients based on natural language processing of chest radiograph reports. *Proc AMIA Annu Fall Symp* 1996:542-6.
35. Yu H, Hatzivassiloglou V, Friedman C, Rzhetsky A, Wilbur WJ. Automatic extraction of gene and protein synonyms from MEDLINE and journal articles. *Proc AMIA Symp* 2002:919-23.
36. Larkey LS, Ogilvie P, Price MA, Tamilio B. Acrophile: an automated acronym extractor and server. In: *Proceedings of the fifth ACM conference on Digital libraries*: ACM Press; 2000. p. 205-214.
37. Agichtein E, Gravano L. Snowball: extracting relations from large plain-text collections. In: *Proceedings of the ACM International Conference on Digital Libraries*; 2000; San Antonio, TX; 2000. p. 85-94.
38. Ding J, Berleant D, Nettleton D, Wurtele E. Mining MEDLINE: abstracts, sentences, or phrases? *Pac Symp Biocomput* 2002:326-37.
39. Brin S. Extracting patterns and relations from the World-Wide Web. In: *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB '98)*; 1998 March 1998; 1998.

40. Agichtein E, Gravano L, Pavel J, Sokolova V, Voskoboynik A. Snowball: A prototype system for extracting relations from large text collections. *Sigmod Record* 2001;30(2):612-612.
41. Brill E. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics* 1995;21(4):543-565.
42. Hripcsak G, Wilcox A. Reference standards, judges, and comparison subjects: roles for experts in evaluating system performance. *J Am Med Inform Assoc* 2002;9(1):1-15.
43. Tanabe L, Wilbur WJ. Tagging gene and protein names in biomedical text. *Bioinformatics* 2002;18(8):1124-32.
44. Kim JD, Ohta T, Tateisi Y, Tsujii J. GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics* 2003;19(Suppl. 1):i180-i182.
45. Baxevanis AD. The Molecular Biology Database Collection: 2002 update. *Nucleic Acids Res* 2002;30(1):1-12.
46. NCBI. Locus Link Introduction, 2003;  
<http://www.ncbi.nlm.nih.gov/LocusLink/index.html>, accessed September 3, 2003.
47. Wren JD, Garner HR. Shared relationship analysis: ranking set cohesion and commonalities within a literature-derived relationship network. *Bioinformatics* 2004;20(2):191-8.
48. Jenssen TK, Laegreid A, Komorowski J, Hovig E. A literature network of human genes for high-throughput analysis of gene expression. *Nat Genet* 2001;28(1):21-8.
49. Roberts CW. Text analysis for the social sciences : methods for drawing statistical inferences from texts and transcripts. Mahwah NJ.: Erlbaum; 1997.
50. Berg BL. Qualitative research methods for the social sciences. In. 4th ed. Boston: Allyn and Bacon; 2001. p. 6-11.
51. Newman MEJ. Small worlds: The structure of social networks. 1999.
52. Newman MEJ, Strogatz SH, Watts DJ. Random graphs with arbitrary degree distributions and their applications. *Physical Review E* 2001;6402(2):art. no.-026118.
53. Newman MEJ, Watts DJ, Strogatz SH. Random graph models of social networks. *Proceedings of the National Academy of Sciences of the United States of America* 2002;99:2566-2572.

54. Newman MEJ. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences of the United States of America* 2001;98(2):404-409.
55. Sigman M, Cecchi GA. Global organization of the Wordnet lexicon. *Proc Natl Acad Sci U S A* 2002;99(3):1742-7.
56. Hirschman L, Morgan AA, Yeh AS. Rutabaga by any other name: extracting biological names. *J Biomed Inform* 2002;35(4):247-59.
57. Proux D, Rechenmann F, Julliard L, Pillet VV, Jacq B. Detecting Gene Symbols and Names in Biological Texts: A First Step toward Pertinent Information Extraction. *Genome Inform Ser Workshop Genome Inform* 1998;9:72-80.
58. Wain HM, Bruford EA, Lovering RC, Lush MJ, Wright MW, Povey S, Guidelines for Human Gene Nomenclature (2002), 2002; <http://www.gene.ucl.ac.uk/nomenclature/guidelines.html>, accessed September 29, 2003.
59. Hanisch D, Fluck J, Mevissen HT, Zimmer R. Playing biology's name game: identifying protein names in scientific text. *Pac Symp Biocomput* 2003:403-14.
60. Ito K, Kubokawa M, Harada N, Mibu R, Nawata H. p53 and PTEN/MMAC1 mutational analysis of the small-intestinal cancer. *Dig Liver Dis* 2003;35(5):347-50.
61. Yamada KM, Araki M. Tumor suppressor PTEN: modulator of cell signaling, growth, migration and apoptosis. *J Cell Sci* 2001;114(Pt 13):2375-82.
62. Stapley BJ, Benoit G. Biobibliometrics: information retrieval and visualization from co-occurrences of gene names in Medline abstracts. *Pac Symp Biocomput* 2000:529-40.
63. Indiana University. FlyBase - A Database of the Drosophila Genome, 2003; <http://flybase.bio.indiana.edu/>, accessed September 29, 2003.
64. The FlyBase database of the Drosophila genome projects and community literature. *Nucleic Acids Res* 2003;31(1):172-5.
65. Gelbart WM, Crosby M, Matthews B, Rindone WP, Chillemi J, Russo Twombly S, et al. FlyBase: a Drosophila database. The FlyBase consortium. *Nucleic Acids Res* 1997;25(1):63-6.
66. Povey S, Lovering R, Bruford E, Wright M, Lush M, Wain H. The HUGO Gene Nomenclature Committee (HGNC). *Hum Genet* 2001;109(6):678-80.
67. Wain HM, Lush M, Ducluzeau F, Povey S. Genew: the human gene nomenclature database. *Nucleic Acids Res* 2002;30(1):169-71.
68. HUGO--a UN for the human genome. *Nat Genet* 2003;34(2):115-6.

69. Chang JT, Schütze H, and Altman RB. Gene and Protein Name Server, 2003; <http://bionlp.stanford.edu/gapscore>, accessed June 17, 2004.
70. Friedman C, Kra P, Rzhetsky A. Two biomedical sublanguages: a description based on the theories of Zellig Harris. *J Biomed Inform* 2002;35(4):222-35.
71. Smalheiser NR, Swanson DR. Using ARROWSMITH: a computer-assisted approach to formulating and assessing scientific hypotheses. *Comput Methods Programs Biomed* 1998;57(3):149-53.
72. de Bruijn B, Martin J. Getting to the (c)ore of knowledge: mining biomedical literature. *Int J Med Inf* 2002;67(1-3):7-18.
73. Cimino JJ, Barnett GO. Automatic knowledge acquisition from MEDLINE. *Methods Inf Med* 1993;32(2):120-30.
74. Mendonca EA, Cimino JJ. Building a knowledge base to support a digital library. *Medinfo* 2001;10(Pt 1):221-5.
75. Results of BioCreative. In: *BioCreative: Critical Assessment for Information Extraction in Biology*; 2004; Granda, Spain; 2004.
76. Aronson AR. Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program. *Proc AMIA Symp* 2001:17-21.
77. Brill E, Mooney RJ. An overview of empirical natural language processing. *Ai Magazine* 1997;18(4):13-24.
78. Brill E. Processing natural language without natural language processing. *Computational Linguistics and Intelligent Text Processing, Proceedings 2003*;2588:360-369.
79. Chang JT, Schutze H, Altman RB. GAPSCORE: finding gene and protein names one word at a time. *Bioinformatics* 2004;20(2):216-25.
80. Franzen K, Eriksson G, Olsson F, Asker L, Liden P, Coster J. Protein names and how to find them. *Int J Med Inf* 2002;67(1-3):49-61.
81. Fisher PB, Gopalkrishnan RV, Chada S, Ramesh R, Grimm EA, Rosengeld MR, et al. mda-7/IL-24, a novel cancer selective apoptosis inducing cytokine gene: from the laboratory into the clinic. *Cancer Biol Ther* 2003;2(4 Suppl 1):S23-37.
82. Humphreys BL, Lindberg DA, Schoolman HM, Barnett GO. The Unified Medical Language System: an informatics research collaboration. *J Am Med Inform Assoc* 1998;5(1):1-11.

83. Liu H, Johnson SB, Friedman C. Automatic resolution of ambiguous terms based on machine learning and conceptual relations in the UMLS. *J Am Med Inform Assoc* 2002;9(6):621-36.
84. Weeber M, Mork JG, Aronson AR. Developing a test collection for biomedical word sense disambiguation. *Proc AMIA Symp* 2001:746-50.
85. Dunham MH. Data mining introductory and advanced topics. Upper Saddle River, N.J.: Prentice Hall/Pearson Education; 2003.
86. Regev Y, Finkelstein-Landau M, Feldman R. Rule-based extraction of experimental evidence in the biomedical domain: the KDD Cup 2002 (task 1)}. *ACM SIGKDD Explorations Newsletter* 2002;4(2):90-92.
87. Shi M, Edwin DS, Menon R, Shen L, Lim JYK, Loh HT. A machine learning approach for the curation of biomedical literature-KDD Cup 2002 (task 1). *ACM SIGKDD Explorations Newsletter* 2002;4(2):93-94.
88. Porter MF. An algorithm for suffix stripping. *Program* 1980;14(3):127-130.
89. Pustejovsky J, Castano J, Zhang J, Kotecki M, Cochran B. Robust relational parsing over biomedical literature: extracting inhibit relations. *Pac Symp Biocomput* 2002:362-73.
90. Liu H, Friedman C. Mining terminological knowledge in large biomedical corpora. *Pac Symp Biocomput* 2003:415-26.
91. Berman JJ. Pathology abbreviated: a long review of short terms. *Arch Pathol Lab Med* 2004;128(3):347-52.
92. Schwartz AS, Hearst MA. A simple algorithm for identifying abbreviation definitions in biomedical text. *Pac Symp Biocomput* 2003:451-62.
93. Brandeis University. Medstrat Project -- Initial Annotation Corpora, <http://scylla.cs.brandeis.edu/gold-standards.html>, accessed June 24, 2003.
94. Craven M, Kumlien J. Constructing biological knowledge bases by extracting information from text sources. *Proc Int Conf Intell Syst Mol Biol* 1999:77-86.
95. Craven M, Slattery S, Nigam K. First-Order Learning for Web Mining. In: 10th European Machine Learning Conference (ECML-98); 1998: Springer; 1998.
96. Pearl J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Mateo, CA: Morgan Kaufmann; 1988.

97. Kittredge R. Paraphrasing for condensation in journal abstracting. *J Biomed Inform* 2002;35(4):265-77.
98. Hobbs JR. Information extraction from biomedical text. *J Biomed Inform* 2002;35(4):260-4.
99. Chapman WW, Cooper GF, Hanbury P, Chapman BE, Harrison LH, Wagner MM. Creating a Text Classifier to Detect Radiology Reports Describing Mediastinal Findings Associated with Inhalational Anthrax and Other Disorders. *J Am Med Inform Assoc* 2003.
100. Chartrand G. Introductory graph theory. Unabridged and corr. ed. New York: Dover; 1985.
101. Sedgewick R. Algorithms. 2nd ed. Reading, Mass.: Addison-Wesley; 1989.
102. Wei L, Liu Y, Dubchak I, Shon J, Park J. Comparative genomics approaches to study organism similarities and differences. *J Biomed Inform* 2002;35(2):142-50.
103. Van Rossum G, Drake FL, Jr., Python Labs. Python reference manual October 14, 2002, release 2.2.2, 2002; <http://www.python.org/doc/current/ref/ref.html>, accessed May 2, 2003.
104. Rose SL, Goodheart MJ, DeYoung BR, Smith BJ, Buller RE. p21 expression predicts outcome in p53-null ovarian carcinoma. *Clin Cancer Res* 2003;9(3):1028-32.
105. Tomer G, Ceballos C, Concepcion E, Benkov KJ. NOD2/CARD15 variants are associated with lower weight at diagnosis in children with Crohn's disease. *Am J Gastroenterol* 2003;98(11):2479-84.
106. Abe T, Terada K, Wakimoto H, Inoue R, Tyminski E, Bookstein R, et al. PTEN decreases in vivo vascularization of experimental gliomas in spite of proangiogenic stimuli. *Cancer Res* 2003;63(9):2300-5.
107. Pagano M, Gauvreau K. Principles of biostatistics. 2nd ed. Australia ; Pacific Grove, CA: Duxbury; 2000.
108. Wheeler DL, Church DM, Federhen S, Lash AE, Madden TL, Pontius JU, et al. Database resources of the National Center for Biotechnology. *Nucleic Acids Res* 2003;31(1):28-33.
109. EBI. UniProt/Swiss-Prot, 2003; <http://www.ebi.ac.uk/swissprot/index.html>, accessed March 12, 2003.
110. Whitley D. A Genetic Algorithm Tutorial. Colorado State University, Dept. of CS, TR CS-93-103. 1993.

111. Michalewicz Z. Genetic algorithms + data structures = evolution programs. 3rd rev. and extended ed. Berlin ; New York: Springer-Verlag; 1999.
112. Whitley D. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In: Proc 3rd International Conference on Genetic Algorithms; 1989: Morgan-Kaufmann; 1989. p. 116-121.
113. Oregon Health & Science Univeristy. TREC Genomics Track, 2003; <http://medir.ohsu.edu/~genomics/>, accessed September 3, 2003.
114. Kanehisa M, Goto S. KEGG: kyoto encyclopedia of genes and genomes. Nucleic Acids Res 2000;28(1):27-30.
115. Kanehisa M. A database for post-genome analysis. Trends Genet 1997;13(9):375-6.
116. Kolchanov NA, Anan'ko EA, Kolpakov FA, Podkolodnaia OA, Ignat'eva EV, Goriachkovskaia TN, et al. [Gene networks]. Mol Biol (Mosk) 2000;34(4):533-44.
117. Kolpakov FA, Ananko EA, Kolesov GB, Kolchanov NA. GeneNet: a gene network database and its automated visualization. Bioinformatics 1998;14(6):529-37.



## APPENDIX A - SOURCE CODE

### Source file: SNLASynonyms.py

```
#
# SNLASynonyms.py
#
# Top level program runs the SNLA based gene and protein name
# knowledge extraction algorithm.
#
# Expects a file of sentences from MEDLINE abstracts or the
# biomedical literature as STDIN, writes results as synonym pair and
# confidence tuples to STDOUT.
#
# Lots of progress and debugging information written to STDERR.
#

# general Python library imports...
import sys
import re
import time
import random

# import graph and pattern classes...
import SNLAGraph
import SNLAPatterns
import SNLACoOccurrences

# algorithm constants...
PAIR_HIGH_CONFIDENCE_THRESHOLD = 0.999 # confidence required for template searching
MINIMUM_PAIR_QUALITY = 2.00 # minimum pair quality to use for pattern
searching
MINIMUM_PATTERN_MATCHING_CASES = 4 # minimum number of matches to include pattern
MAXIMUM_PATTERN_MATCHING_CASES = 1000 # prevent runaway pattern matches
MINIMUM_LINK_TO_FOLLOW = 2 # minimum inference link in GetStrongestChains()
RANDOM_SEED = 2001 # random seed for genetic algorithm
UNKNOWN_SCORE = -1.0 # sentinel value
NUM_GENOMES_TO_FREE_PASS = 1 # free pass to next generation
MUTATION_RATE = 0.01 # gene mutation rate
CROSSOVER_RATE = 0.90 # genome crossover rate
NUM_GENERATIONS = 100 # maximum generations per iteration
GENERATION_SIZE = 200 # number genomes in generation
MAX_NO_IMPROVEMENT_ITERATIONS = 5 # early bailout with no change
PATTERNS_PER_ITERATION = 150 # patterns to evaluate each iteration
PATTERN_INCLUSION_PROBABILITY = 0.25
RANK_SELECTION_PRESSURE = 0.50

# collection of initial seed pairs...
# EACH PAIR MUST BE IN ALPHABETICAL ORDER BECAUSE OF THE
# WAY THAT THE TUPLE MATCHING WORKS...
INITIAL_SEED_PAIRS = [
    ('CIP1', 'WAF1'),
    ('LPS', 'TLR4'),
    ('CD82', 'KAI1'),
    ('IGF2R', 'M6P'),
    ('MMAC1', 'PTEN'),
    ('DR5', 'KILLER'),
    ('CCN3', 'NOV'),
    ('ASF', 'SF2'),
]

# start time...
startTime = time.time()

# seed the random number generator for consistent results...
random.seed(RANDOM_SEED)

# algorithm global data structures...
```

```

# list of found and used high confidence synonym pairs...
allUsedHighConfidencePairs = []

# dictionary cache of pattern matches found and used...
# the keys are the regex pattern and the values are
# lists of tuples with the matched pair and sentence number...
# the printable form of the regex will be used as the key
# for efficiency...
# a value of None indicates that the pattern has been used
# either the pattern has been accepted and the co-occurrences
# have been incorporated into the graph, or the pattern has
# been reject because it is outside boundary constraints...
patternCache = {}

# count each iteration...
iterationCount = 0

# global network graph object...
networkGraph = SNLAGraph.SNLAGraph()

# create SNLAPatterns object to manage pattern creation and matching...
patternManager = SNLAPatterns.SNLAPatterns()

# object to track co-occurrence sources...
networkCoOccurrences = SNLACoOccurrences.SNLACoOccurrences()

# utility function for sorting genomes by score...
def genomeSortFxn(a,b):
    return cmp(b[1], a[1])

# utility function for sorting high confidence pairs by confidence...
def highConfidenceSortFxn(a,b):
    return cmp(b[2], a[2])

# utility function for randomizing the order of a list...
def randomizeListSortFxn(a,b):
    return cmp(random.random(), random.random())

# begin main program...

# set defaults and process command line arguments...
iterationLimit = -1
numSeeds = 8
acceptNewSeedIterations = -1
maxNumActivePatterns = -1
progName = sys.argv[0]
sys.argv = sys.argv[1:]
while sys.argv:
    arg = sys.argv[0]
    sys.argv = sys.argv[1:]
    if arg == "-h":
        sys.stderr.write("Usage: %s [-i iterations] [-n #seeds] [-s seedfile] [-m max-
num-patterns] < STDIN\n" % progName)
        sys.exit(0)
    elif arg == "-i":
        iterationLimit = int(sys.argv[0])
        sys.argv = sys.argv[1:]
    elif arg == "-n":
        numSeeds = int(sys.argv[0])
        sys.argv = sys.argv[1:]
    elif arg == "-a":
        acceptNewSeedIterations = int(sys.argv[0])
        sys.argv = sys.argv[1:]
    elif arg == "-m":
        maxNumActivePatterns = int(sys.argv[0])
        sys.argv = sys.argv[1:]
    elif arg == "-s":
        # seed file format must be one tuple per line,
        # with each line a tuple of (synA,synB,count)...
        seedFile = sys.argv[0]

```

```

sys.argv = sys.argv[1:]
sys.stderr.write("Reading seed file: %s\n" % seedFile)
seedPairs = []
file = open(seedFile, "r")
for line in file.xreadlines():
    seedPairs.append(eval(line))
file.close()
seedPairs.sort(lambda a, b: cmp(b[2], a[2]))
INITIAL_SEED_PAIRS = tuple(seedPairs)

# check the number of seeds...
if numSeeds > len(INITIAL_SEED_PAIRS):
    numSeeds = len(INITIAL_SEED_PAIRS)
sys.stderr.write("Using %d initial seeds.\n" % numSeeds)

# read input file into sentences list, strip any leading
# or trailing whitespace...
sentences = []
for s in sys.stdin.xreadlines():
    sentences.append(s.strip())
numSentences = len(sentences)

# loop until we complete the given number of iterations
# or until the program is interrupted...
while 1:
    # output progress to stderr...
    sys.stderr.write("Beginning iteration %d.\n" % (iterationCount + 1))

    # determine the high confidence pairs to use for this iteration...
    newHighConfidencePairs = []
    if iterationCount == 0:
        sys.stderr.write("Using seed initial high confidence pairs.\n")
        for index in xrange(0, numSeeds):
            pair = INITIAL_SEED_PAIRS[index]
            newHighConfidencePairs.append((pair[0], pair[1], 1.0))
    else:
        # retrieving the current synonym pairs...
        sys.stderr.write("Retrieving current synonym pairs:\n")
        synTriples = networkGraph.getStrongestChains(MINIMUM_LINK_TO_FOLLOW)

        # display the current synonym pairs...
        for (synA, synB, count) in synTriples:
            # get stats for this pair...
            confidence = networkGraph.computeConfidence(count)
            cooccur = networkCoOccurrences.getCoOccurrenceCount(synA, synB)

            # format according to the type of pair, these case changes
            # will not affect the evaluation programs, which convert
            # everything to upper case anyway...
            # this provides some additional information for accessing the
            # affect of network inference...
            if cooccur == 0:
                # if this is a DERIVED pair, print both in lower case...
                quartet = (synA.lower(), synB.lower(), count, confidence)
            elif cooccur > count:
                # if this is a STRONGER pair, print only the first in lower case...
                quartet = (synA.lower(), synB, count, confidence)
            else:
                # otherwise, leave both in upper case...
                quartet = (synA, synB, count, confidence)

            sys.stderr.write("%s\n" % str(quartet))

        # print the count...
        sys.stderr.write("\nExtracted %d synonym pairs.\n" % len(synTriples))
        sys.stderr.flush()

        # determine the number of co-occurrences required for high confidence...
        requiredCooccurrences =
networkGraph.computeRequiredCooccurrences(PAIR_HIGH_CONFIDENCE_THRESHOLD)

```

```

        sys.stderr.write("Requiring %d co-occurrences for high confidence, computing
pairs.\n" % requiredCooccurrences)
        sys.stderr.flush()

        # extract the high confidence synonym pairs...
        for synTriple in synTriples:
            if synTriple[2] >= requiredCooccurrences:
                # require found, not inferred coOccurrences...
                cooccur =
networkCoOccurrences.getCoOccurrenceCount(synTriple[0], synTriple[1])
                if cooccur >= requiredCooccurrences:
                    qualA = networkGraph.computeNodeQuality(synTriple[0])
                    qualB = networkGraph.computeNodeQuality(synTriple[1])
                    qualPair = qualA * qualB
                    if qualPair >= MINIMUM_PAIR_QUALITY:
                        sys.stderr.write("Identified high confidence pair: %s (%0.2f)\n"
% (str((synTriple[0], synTriple[1])), qualPair))
                        newHighConfidencePairs.append((synTriple[0], synTriple[1],
qualPair))
                        sys.stderr.flush()

                # release the memory for the synTriples list...
                synTriples = None

        # process candidate high confidence pairs list...
        if acceptNewSeedIterations > 0 and iterationCount >= acceptNewSeedIterations:
            # clear list if we are only using the original seeds...
            newHighConfidencePairs = []
            sys.stderr.write("\nNot accepting new high confidence synonym pairs...\n")
            sys.stderr.flush()
        else:
            # sort list highest confidence first...
            sys.stderr.write("\nSorting new high confidence synonym pairs...\n")
            newHighConfidencePairs.sort(highConfidenceSortFxn)

        # remove any high confidence pairs that we have already
        # used to search for patterns...
        # note that order is always alphabetical (see getStrongestChains)...
        sys.stderr.write("\nLooking for new high confidence synonym pairs...\n")
        tempHighConfidencePairs = newHighConfidencePairs
        newHighConfidencePairs = []
        for (synA, synB, conf) in tempHighConfidencePairs:
            if (synA, synB) not in allUsedHighConfidencePairs:
                newHighConfidencePairs.append((synA, synB))
                sys.stderr.write("Found new high confidence pair: %s (%0.5f)\n" %
(str((synA, synB)), conf))
                sys.stderr.write("Found %d new high confidence pairs.\n\n" %
len(newHighConfidencePairs))
                sys.stderr.flush()

        # release the memory for the tempHighConfidencePairs list...
        tempHighConfidencePairs = None

        # allow for bailout if specified on command line...
        if iterationCount == iterationLimit:
            break

        # keep track of the pairs actually used...
        usedHighConfidencePairs = []

        # count the current number of active patterns...
        currentNumActivePatterns = 0
        for p in patternCache.keys():
            if patternCache[p]:
                currentNumActivePatterns += 1

        # flag for early exit of pattern searching loop...
        stopFindingPatterns = 0

        # search sentences for the newly found synonym pairs
        # and create search patterns from the sentences

```

```

# that we find the pairs in...
# note that a sentence can be used more than once as a source of
# patterns but it can only be used once as a source of a given
# pair of co-occurrences...
for hpair in newHighConfidencePairs:
    # check for early exit of loop...
    if stopFindingPatterns:
        break

    # mark this pair as used...
    usedHighConfidencePairs.append(hpair)

    # compile the pair names into case-insensitive
    # matching patterns...
    # could have some bugs in here if either name
    # contains special regex characters, will have
    # to remove this possibility in the isGeneName()
    # function in SNLAPatterns...
    hmatchSynA = re.compile(hpair[0], re.IGNORECASE)
    hmatchSynB = re.compile(hpair[1], re.IGNORECASE)
    for s in sentences:
        # check for early exit of loop...
        if stopFindingPatterns:
            break

        # test if each member of the pair is found in the
        # current sentence, if not found, the pair is definitely
        # not in the sentence, but if it is found, the names may
        # only be part of a larger name since we are not checking
        # delimiter boundaries. this is okay because the
        # convertSynonymsAndSentencesToRegEx() method will check
        # delimiter boundaries and return None if the pair is not
        # found in the sentence...
        if hmatchSynA.search(s) and hmatchSynB.search(s):
            # try a set of leading and trailing padding of
            # non-gene name tokens around the known high confidence
            # name pair for our patterns...
            for (leading, trailing) in ((0,0), (1,0), (0,1), (1,1)):
                # check for early exit of loop...
                if stopFindingPatterns:
                    break

            # otherwise, convert to pattern...
            regPair =
patternManager.convertSynonymsAndSentencesToRegExPair(hpair, s, leading, trailing)
            if regPair:
                # we don't want to accept the same pattern twice...
                (pattern, printable) = regPair
                if printable in patternCache:
                    continue

                # found a new pattern...
                sys.stderr.write("Found new pattern: %s %s\n" % (printable,
str(hpair)))

                sys.stderr.write("In sentence: %s\n" % s)

                # now accumulate the matches for this pattern...
                sys.stderr.write("Matching pattern: %s\n" % printable)

                # initialize list of pairs found by this pattern...
                pairs = []

                # pattern NOT found in the cache...
                # compile the pattern...
                matcher = re.compile(pattern)

                # apply it to all of the sentences, which are indexed
                # by number in order to help track the sources of
                # the co-occurrences...
                numPairs = 0
                for sourceID in xrange(numSentences):

```

```

        # apply the regular expression to the sentence...
        sentence = sentences[sourceID]
        matchObj = matcher.search(sentence)
        if matchObj:
            synList = matchObj.groups()
        else:
            # move on to the next sentence...
            continue

        # convert all pairs to uppercase at this stage and
        # do not allow duplicates in the name list...
        # the lexical criteria for a gene name are applied here...
        tempPairs =
patternManager.convertTupleToUppercaseGeneNamePairList(synList)
        for pair in tempPairs:
            # add pair tuple with sourceID to new pairs list...
            pairs.append((pair, sourceID))
            numPairs += 1
            sys.stderr.write("Found pair: %s in sourceID %s\n" %
(str(pair), str(sourceID)))

        # release the tempPairs memory...
        tempPairs = None

        # early bailout for efficiency...
        if numPairs > MAXIMUM_PATTERN_MATCHING_CASES:
            break

        # check that number of pairs are within constraints...
        numPairs = len(pairs)
        if numPairs < MINIMUM_PATTERN_MATCHING_CASES:
            # too few pairs, discard pattern...
            sys.stderr.write("Discarding pattern, below boundary
constraints.\n")

            patternCache[printable] = None
        elif numPairs > MAXIMUM_PATTERN_MATCHING_CASES:
            # too many pairs, discard pattern...
            sys.stderr.write("Discarding pattern, above boundary
constraints.\n")

            patternCache[printable] = None
        else:
            # add found pairs for this pattern to the patternCache...
            # priority is the number of matches that the pattern finds...
            sys.stderr.write("Retaining pattern, has %d pairs.\n" %
numPairs)

            patternCache[printable] = pairs
            currentNumActivePatterns += 1

        # release the pairs memory...
        pairs = None

        # report progress,,,
        sys.stderr.write("\n")
        sys.stderr.flush()

        # limit number of active patterns if specified...
        if maxNumActivePatterns > 0 and currentNumActivePatterns >=
maxNumActivePatterns:
            sys.stderr.write("Reached maximum active pattern count of
%d.\n" % currentNumActivePatterns)
            stopFindingPatterns = 1
            break

        # we now have all of the patterns that are findable by the
        # chosen set of high confidence pairs,
        # now use a genetic algorithm to find which set of patterns
        # from a randomly chosen subset gives the highest quality network...

        # report progress...
        sys.stderr.write("High confidence pairs used for this iteration: %s\n" %
str(usedHighConfidencePairs))

```

```

sys.stderr.write("Randomizing current patterns...\n")
sys.stderr.flush()

# get list of all the current patterns...
patterns = []
for p in patternCache.keys():
    if patternCache[p]:
        patterns.append(p)
totalPatterns = len(patterns)

# determine number of patterns to use this iteration...
if maxNumActivePatterns > 0:
    numPatterns = min(maxNumActivePatterns, totalPatterns)
else:
    numPatterns = min(PATTERNS_PER_ITERATION, totalPatterns)

# randomly sort the list, we will use only the
# first numPatterns in the list for this iteration...
patterns.sort(randomizeListSortFxn)

# report progress...
sys.stderr.write("There are %d active patterns of %d total patterns under
consideration.\n" % (numPatterns, totalPatterns))
sys.stderr.write("Computing best set of patterns...\n")
sys.stderr.flush()

# create random list of pattern genomes...
genomeList = []
for i in range(0, GENERATION_SIZE):
    # each genome is initially a random string
    # of '+' and '-' characters that represents
    # whether a specific pattern is present in the
    # genome or not...
    genome = []
    for j in range(0, numPatterns):
        # each new pattern has avchance of
        # being included in this genome...
        if random.random() <= PATTERN_INCLUSION_PROBABILITY:
            genome.append('+')
        else:
            genome.append('-')

    # convert list to string...
    genome = ''.join(genome)

    # add to genome list with a dummy score
    genomeList.append((genome, UNKNOWN_SCORE))

# cycle through the generations...
generationNumber = 0
previousGenerationQuality = 0.0
generationsWithNoImprovement = 0
while 1:
    # save the start time...
    generationStartTime = time.time()

    # loop through and score the current genome list...
    averageScore = 0.0
    for i in range(0, GENERATION_SIZE):
        (genome, score) = genomeList[i]
        if score == UNKNOWN_SCORE:
            # create a graph and coOccurrence table to use
            # while scoring this genome...
            tempGraph = SNLAGraph.SNLAGraph()
            tempCoOccurrences = SNLACoOccurrences.SNLACoOccurrences()

            for j in range(0, numPatterns):
                # check whether to include this pattern...
                if genome[j] == '+':
                    # include the pairs from this pattern...
                    for (pair, sourceID) in patternCache[patterns[j]]:

```

```

once...
# we don't want to use the same co-occurrence more than
pair[1], sourceID):
    if not tempCoOccurrences.checkCoOccurrenceSourceList(pair[0],
    # add pair tuple with sourceID to graph...
    tempGraph.addSynonymPair(pair)
    tempCoOccurrences.addCoOccurrenceSource(pair[0], pair[1],
sourceID)

    # score this genome...
    score = tempGraph.computeQuality()
    genomeList[i] = (genome, score)

    # release the memory...
    tempGraph = None
    tempCoOccurrences = None
    # total up the scores....
    averageScore += score

# compute the current genome average fitness...
averageScore = averageScore / GENERATION_SIZE

# sort the current genomeList by score, highest first...
genomeList.sort(genomeSortFxn)
currentGenerationQuality = genomeList[0][1]

# output progress...
generationTime = time.time() - generationStartTime
sys.stderr.write("Generation %d complete, computed in %0.2f seconds.\n" %
(generationNumber, generationTime))
sys.stderr.write("Average score is %0.4f.\n" % averageScore)
sys.stderr.write("Best score is %0.4f.\n" % currentGenerationQuality)
sys.stderr.write("Best genome: %s\n\n" % genomeList[0][0])
sys.stderr.flush()

# exit loop if generations complete...
if generationNumber >= NUM_GENERATIONS:
    break

# also exit loop if we haven't made any improvement in a while...
if currentGenerationQuality <= previousGenerationQuality:
    generationsWithNoImprovement += 1
    if generationsWithNoImprovement == MAX_NO_IMPROVEMENT_ITERATIONS:
        break
else:
    generationsWithNoImprovement = 0
    previousGenerationQuality = currentGenerationQuality

# breed the genomes into the next generation...
intermediateGenomeList = []
numIntermediateGenomeList = 0

# the best ones get a free pass...
for i in range(0, NUM_GENOMES_TO_FREE_PASS):
    intermediateGenomeList.append(genomeList[i])
numIntermediateGenomeList = NUM_GENOMES_TO_FREE_PASS

# rank sampling to complete the rest
# of the selection process...
rank = 0
while numIntermediateGenomeList < GENERATION_SIZE:
    # compute probability of being selected...
    p = 1.0 / (RANK_SELECTION_PRESSURE*rank + 1.0)

    # randomly include...
    if random.random() < p:
        intermediateGenomeList.append(genomeList[rank])
        numIntermediateGenomeList += 1

    # advance to the next ranked genome and loop if necessary...
    rank += 1

```



```

        if rank == GENERATION_SIZE:
            rank = 0
    sys.stderr.write("Bred %d genomes.\n" % GENERATION_SIZE)

    # move genomes from intermediate generation to next
    # generation using crossover...
    genomeList = []

    # give the top scorers a free pass...
    for i in range(0, NUM_GENOMES_TO_FREE_PASS):
        genomeList.append(intermediateGenomeList[i])
    numGenomeList = NUM_GENOMES_TO_FREE_PASS

    # compute the crossovers...
    crossoverCount = 0
    while numGenomeList < GENERATION_SIZE:
        # update the count...
        numGenomeList += 2

        # choose a pair of genomes to crossover...
        j = random.randint(0, GENERATION_SIZE - 1)
        k = random.randint(0, GENERATION_SIZE - 1)

        # crossover with given probability...
        if random.random() < CROSSOVER_RATE:
            # randomly choose a 1-point crossover location...
            l = random.randint(1, numPatterns - 1)

            # convert to lists...
            (genome1, score) = intermediateGenomeList[j]
            (genome2, score) = intermediateGenomeList[k]
            genome1 = list(genome1)
            genome2 = list(genome2)

            # perform the crossover...
            newGenome1 = genome1[0:l] + genome2[l:]
            newGenome2 = genome2[0:l] + genome1[l:]

            # convert back to strings...
            genome1 = ''.join(newGenome1)
            genome2 = ''.join(newGenome2)

            # put into the gene pool...
            genomeList.append((genome1, UNKNOWN_SCORE))
            genomeList.append((genome2, UNKNOWN_SCORE))

            # count the crossovers...
            crossoverCount += 1
        else:
            # just move these genomes without crossover...
            genomeList.append(intermediateGenomeList[j])
            genomeList.append(intermediateGenomeList[k])
    # report progress...
    sys.stderr.write("Performed %d crossovers.\n" % crossoverCount)

    # release memory...
    intermediateGenomeList = None

    # randomly mutate, but not the top scorers...
    aveMutations = MUTATION_RATE * GENERATION_SIZE * numPatterns
    numMutations = max(0, int(random.gauss(aveMutations, aveMutations/4.0) + 0.50))
    for i in range(0, numMutations):
        # choose a genome and a gene to mutate...
        j = random.randint(NUM_GENOMES_TO_FREE_PASS, GENERATION_SIZE - 1)
        k = random.randint(0, numPatterns - 1)

        # mutate the chosen gene of the selected genome...
        (genome, score) = genomeList[j]
        genome = list(genome)
        if genome[k] == '+':
            genome[k] = '-'

```

```

        else:
            genome[k] = '+'
            genome = ''.join(genome)

            # place back in the genome list...
            genomeList[j] = (genome, UNKNOWN_SCORE)
            sys.stderr.write("Performed %d mutations.\n" % numMutations)

            # count the generation...
            generationNumber += 1

# done with the genetic optimization, now take the best
# genome and update the network graph with it...
(genome, score) = genomeList[0]

# loop through the patterns...
numKept = 0
for j in range(0, numPatterns):
    # check whether to include this pattern...
    if genome[j] == '+':
        # report progress...
        sys.stderr.write("Keeping GOOD pattern: %s\n" % patterns[j])
        # include the pairs from this pattern...
        for (pair, sourceID) in patternCache[patterns[j]]:
            # we don't want to use the same co-occurrence more than once...
            if not networkCoOccurrences.checkCoOccurrenceSourceList(pair[0], pair[1],
sourceID):
                # add pair tuple with sourceID to graph...
                networkGraph.addSynonymPair(pair)
                networkCoOccurrences.addCoOccurrenceSource(pair[0], pair[1], sourceID)
            # mark pattern as used, only if we have accepted it,
            # otherwise the pattern will continue into the next
            # iteration...
            patternCache[patterns[j]] = None
            # count it...
            numKept += 1

# flush progress log...
sys.stderr.write("Kept %d GOOD patterns.\n" % numKept)
sys.stderr.flush()

# release any memory used...
patterns = None
genomeList = None

# score this graph...
currentQuality = networkGraph.computeQuality()

# all accepted patterns for this iteration used, now
# move the new high confidence pairs into the found list...
allUsedHighConfidencePairs.extend(usedHighConfidencePairs)
sys.stderr.write("Used high confidence pairs list: %s\n" %
str(allUsedHighConfidencePairs))

# current time...
currentTime = time.time()

# count the iterations and give update...
iterationCount += 1
sys.stderr.write("At end of iteration, nextwork quality is %0.4f\n" % currentQuality)
sys.stderr.write("Completed %d iterations in %0.2f seconds.\n\n" % (iterationCount,
currentTime - startTime))

# flush stderr for each iteration...
sys.stderr.flush()

# retrieve the final set of synonym pairs...
sys.stderr.write("Retrieving final synonym pairs:\n")
synTriples = networkGraph.getStrongestChains(MINIMUM_LINK_TO_FOLLOW)

# display the current synonym pairs...

```

```

for (synA, synB, count) in synTriples:
    # get stats for this pair...
    confidence = networkGraph.computeConfidence(count)
    cooccur = networkCoOccurrences.getCoOccurrenceCount(synA, synB)

    # format according to the type of pair, these case changes
    # will not affect the evaluation programs, which convert
    # everything to upper case anyway...
    # this provides some additional information for accessing the
    # affect of network inference...
    if cooccur == 0:
        # if this is a DERIVED pair, print both in lower case...
        quartet = (synA.lower(), synB.lower(), count, confidence)
    elif cooccur > count:
        # if this is a STRONGER pair, print only the first in lower case...
        quartet = (synA.lower(), synB, count, confidence)
    else:
        # otherwise, leave both in upper case...
        quartet = (synA, synB, count, confidence)

    sys.stderr.write("%s\n" % str(quartet))

# print the count and exit...
sys.stderr.write("\nExtracted %d synonym pairs.\n" % len(synTriples))
sys.stderr.write("Done.\n")
sys.stderr.flush()

```

## Source file: SNLAPatterns.py

```
#
# SNLAPatterns.py
# Handles pattern creation and processing.
#
import sys
import re

# defined constants...

# RE patterns to match possible gene names...
# a GENE name pattern is a sequence of characters between 3 and 14
# in length that does not contain spaces or the other listed
# special characters...

GENE_NAME_REGEX = r"([^\s,%<>+&()=\[\]\?\\$'\"]{3,14})"
REQUIRED_WHITE_SPACE_REGEX = r"\s+"
OPTION_WHITE_SPACE_REGEX = r"\s*"

# list of word delimiters, including whitespace...
TOKEN_DELIMITER_LIST = r" .,(){}[]=;:!?^$|/%<>+&'\\"
REGEX_SPECIAL_CHARACTER_LIST = r"^\$*+|\?() []-{}+\""

# special-case not gene name stop words...
GENE_NAME_STOP_LIST = (
    'RNA', 'mRNA', 'tRNA', 'DNA', 'cDNA', 'THE', 'METHODS', 'RESULTS', 'OBJECTIVE',
    'PURPOSE', 'A-CHAIN', 'CO-DELETION', 'DOWN-REGULAT.*', 'UP-REGULAT.*',
    'CELL-SURFACE', 'ADD', '.*\'.EDU', '.*\'.COM', '.*\'.GOV', '.*\'.ORG', '.*\'.UK',
    '.*\'.HTML?', 'AIDS', '.*-RICH', 'CMV', '.*-BINDING', '.*-LIKE', '.*-POSITIVE',
    '.*-NEGATIVE', '.*-MEDIATED', '.*-TERMINAL', '.*-ACTIVATED', '.*-REGULATED',
    '.*-SINGLE', '.*-DEPENDENT', '.*-RELATED', '.*-STRAND', 'AD-.*', 'AD\d*',
    'AD.?CMV-.*', 'VIRUS', 'GENBANK', 'EMBL', 'DDBJ', 'ACTIVATED', '.*-ASSOCIATED',
    'DROSOPHILA', '.*-INDUCED', '.*-SENSITIVE', 'ASPERGILLUS',
)

# maximum pattern length in tokens...
MAXIMUM_PATTERN_LENGTH = 10

# maximum pair distance in tokens...
MAXIMUM_EXTRACTED_DISTANCE = 4

# matcher for gene name...
geneNameMatcher = re.compile(GENE_NAME_REGEX)

# matcher for stop words...
stopTerms = []
for t in GENE_NAME_STOP_LIST:
    # match full terms to end of string...
    stopTerms.append(t + "\Z")
stopPattern = "|".join(stopTerms)
stopWordMatcher = re.compile(stopPattern, re.IGNORECASE)

class SNLAPatterns:
    # static class functions...

    # optional position argument for detecting capitalized
    # words at beginning of sentences...
    def isGeneName(s, pos = -1):
        # save length...
        totalCount = len(s)

        # must match GENE_NAME_REGEX...
        m = geneNameMatcher.match(s)
        if not m or m.end() != totalCount:
            return 0

        # must not match stop list...
        m = stopWordMatcher.match(s)
        if m:
```

```

        return 0

    # do not allow gene name to begin with a digit...
    if s[0].isdigit():
        return 0

    # do not allow gene name to begin with a
    # dash or a colon or a period or an asterisk or a tilde...
    if s[0] in "-.:*~":
        return 0

    # do not allow gene name to end with a dash or a period or a colon...
    if s[totalCount - 1] in "-.:":
        return 0

    # count upper, lower, numbers, and other characters...
    upperCount = 0
    lowerCount = 0
    digitCount = 0
    otherCount = 0
    for t in s:
        if t.isupper():
            upperCount += 1
        elif t.islower():
            lowerCount += 1
        elif t.isdigit():
            digitCount += 1
        else:
            otherCount += 1

    # simple additional rules...
    if lowerCount == totalCount:
        return 0
    elif upperCount == 0 and lowerCount == 0:
        return 0
    elif pos == 0 and upperCount == 1 and lowerCount == totalCount - 1:
        return 0
    else:
        return 1

# make it a static method...
isGeneName = staticmethod(isGeneName)

def convertSentenceToTokenList(sentence):
    # punctuation is retained in the token list
    # but whitespace is not...
    tokens = []

    # initialize...
    length = len(sentence)
    start = 0
    cur = 0

    # loop over characters in sentence...
    while cur < length:
        if sentence[cur] in TOKEN_DELIMITER_LIST:
            # a period is a delimiter only at the end of the sentence...
            if sentence[cur] == "." and cur != length - 1:
                cur += 1
                continue

            # excise token...
            if cur > start:
                tokens.append(sentence[start:cur])

            # add delimiter token if not space...
            if not sentence[cur].isspace():
                tokens.append(sentence[cur])

            # save new start offset...
            start = cur + 1

```

```

        # advance current pointer...
        cur += 1

    # possible last token...
    if cur != start:
        tokens.append(sentence[start:cur])

    # all done...
    return tokens

# make it a static method...
convertSentenceToTokenList = staticmethod(convertSentenceToTokenList)

def convertSynonymsAndSentencesToRegexPair( synPair, sentence, leading=0,
trailing=0):
    # break the sentence into a list of tokens...
    tokens = SNLAPatterns.convertSentenceToTokenList(sentence)

    # find the indexes where the pairs are
    # doing a case-insensitive match...
    synA = synPair[0].upper()
    synB = synPair[1].upper()
    synAList = []
    synBList = []
    for n in range(0, len(tokens)):
        tok = tokens[n].upper()
        if tok == synA:
            synAList.append(n)
        if tok == synB:
            synBList.append(n)
    if len(synAList) == 0 or len(synBList) == 0:
        # if either of the synonyms is not found in the
        # sentence then the method returns None...
        # sys.stderr.write("SYNONYMS NOT FOUND\n")
        return None

    # pick the synonym pair closest to each other...
    first = synAList[0]
    last = synBList[0]
    distance = abs(first - last)
    for n in synAList:
        for m in synBList:
            d = abs(n - m)
            if d < distance:
                first = n
                last = m
                distance = d

    # ensure that last > first...
    if first > last:
        (first, last) = (last, first)

    # check the distance between the pairs...
    if (last - first - 1) > MAXIMUM_EXTRACTED_DISTANCE:
        return None

    # initialize regex string...
    regexp = ""
    printableRegex = ""
    tokenCount = 0

    # avoid degenerate patterns by including non-gene name
    # matching words or symbols before and after the pattern
    # when possible...
    # try to allow leading and trailing non-gene symbols on the
    # front of the pattern...
    start = first
    pad = leading
    while start > 0 and pad > 0:
        start -= 1
        if SNLAPatterns.isGeneName(tokens[start], start):

```

```

        pad = leading
    else:
        pad -= 1
# and now the back of the pattern...
end = last
pad = trailing
while end < len(tokens)-1 and pad > 0:
    end += 1
    if SNLAPatterns.isGeneName(tokens[end], end):
        pad = trailing
    else:
        pad -= 1

# flag to detect degenerate patterns, which are
# patterns consisting solely of gene name matches...
degenerate = 1

# flag for previous token type...
prevWord = 0

# add tokens to the regex...
for i in range(start, end + 1):
    if tokens[i][0] in TOKEN_DELIMITER_LIST:
        # delimiter, add to string with optional surrounding
        # whitespace...
        # need to check for need to escape special characters...
        printableRegexp += tokens[i]
        if tokens[i][0] in REGEX_SPECIAL_CHARACTER_LIST:
            tokens[i] = "\\" + tokens[i][0]
        regexp += OPTION_WHITE_SPACE_REGEX + tokens[i] + OPTION_WHITE_SPACE_REGEX
        prevWord = 0
        degenerate = 0
        tokenCount += 1
        # print "DELIMITER: " + tokens[i]
    elif i == first or i == last or SNLAPatterns.isGeneName(tokens[i], i):
        # potential gene name, add to string with
        # required previous whitespace if needed...
        if prevWord:
            regexp += REQUIRED_WHITE_SPACE_REGEX
            printableRegexp += " "
        regexp += GENE_NAME_REGEX
        printableRegexp += "$GENE$"
        prevWord = 1
        tokenCount += 1
        # print "GENE NAME: " + tokens[i]
    else:
        # add the word literally to the pattern...
        if prevWord:
            regexp += REQUIRED_WHITE_SPACE_REGEX
            printableRegexp += " "
        # need to be careful that literals do not have
        # any unescaped special characters in them...
        word = ""
        for c in tokens[i]:
            if c in REGEX_SPECIAL_CHARACTER_LIST:
                word += "\\" + c
            else:
                word += c
        regexp += word
        printableRegexp += word
        # print "LITERAL: " + word
        prevWord = 1
        degenerate = 0
        tokenCount += 1

# add last whitespace if necessary...
if prevWord:
    regexp += REQUIRED_WHITE_SPACE_REGEX
    printableRegexp += " "

# all done, return the pair of the regex and the printable regex

```

```

    # if not degenerate...
    if tokenCount < MAXIMUM_PATTERN_LENGTH and not degenerate:
        return (regexp, printableRegexp)
    else:
        # sys.stderr.write("DEGENERATE\n")
        return None
# make it a static method...
convertSynonymsAndSentencesToRegExPair =
staticmethod(convertSynonymsAndSentencesToRegExPair)

# utility function to create a list of tuple pairs from a list...
def convertTupleToPairList(tup):
    pairs = []
    count = len(tup)
    for n in range(0, count - 1):
        for m in range(n + 1, count):
            pairs.append((tup[n], tup[m]))
    return pairs
convertTupleToPairList = staticmethod(convertTupleToPairList)

# utility function to create a list of uppercase
# tuple pairs from a list...
def convertTupleToUppercaseGeneNamePairList(tup):
    # check if acceptable gene name and convert
    # to upper case...
    # do not allow duplicates in the list...
    names = []
    for t in tup:
        if SNLAPatterns.isGeneName(t):
            name = t.upper()
            if name not in names:
                names.append(name)

    pairs = []
    count = len(names)
    for n in range(0, count - 1):
        for m in range(n + 1, count):
            pairs.append((names[n], names[m]))
    return pairs
convertTupleToUppercaseGeneNamePairList =
staticmethod(convertTupleToUppercaseGeneNamePairList)

# constructor...
def __init__(self):
    # initialize member variables here...
    return

```



## Source file: SNLACoOccurrences.py

```
#
# SNLACoOccurrences.py
#
# Tracks the sample identifiers in which a co-occurrence pair is
# found by using an internal Table object.
#
import Table

class SNLACoOccurrences:
    # constructor.
    def __init__(self):
        # create the table with None as the default value...
        self.__table = Table.Table(None)

    # member function to add co-occurrence source identifier...
    def addCoOccurrenceSource(self, symA, symB, sourceID):
        # get the current list...
        entries = self.__table.get(symA, symB)

        # add the identifier...
        # if the list is None, create a new list...
        if entries is None:
            entries = [sourceID]
        else:
            entries.append(sourceID)

        # make sure that table is diagonal symmetric...
        self.__table.set(symA, symB, entries)
        self.__table.set(symB, symA, entries)

    # member function to get co-occurrence source identifier list...
    def getCoOccurrenceSourceList(self, symA, symB):
        entries = self.__table.get(symA, symB)
        if entries is None:
            entries = []
        return entries

    # member function to check whether a sourceID is present
    # in a symbol pair source list...
    def checkCoOccurrenceSourceList(self, symA, symB, sourceID):
        entries = self.__table.get(symA, symB)
        if entries is None:
            return 0
        else:
            return sourceID in entries

    def getCoOccurrenceCount(self, symA, symB):
        entries = self.__table.get(symA, symB)
        if entries is None:
            return 0
        else:
            return len(entries)
```

## Source file: SNLAGraph.py

```
#
# SNLAGraph.py
# Associates gene synonyms in a graph structure and computes linkages.
#
import Table
import sys
import math

# small factor for MCC/MNCC computation avoid divide by zero...
QUALITY_COMPUTATION_OFFSET = 0.10

class SNLAGraph:
    # constructor.
    def __init__(self):
        # create the table with zero as the default value...
        self.__table = Table.Table(0)

    # static class utility methods...

    # iterative factorial method...
    def fact(n):
        f = 1.0
        while n > 1:
            f *= n
            n -= 1
        return f
    fact = staticmethod(fact)

    # combinations of n objects taken x at a time
    def combinations(n, x):
        return SNLAGraph.fact(n) / (SNLAGraph.fact(x) * SNLAGraph.fact(n - x))
    combinations = staticmethod(combinations)

    # combinations of n objects taken 2 at a time
    def combinationsOfTwo(n):
        return (n * (n - 1)) / 2
    combinationsOfTwo = staticmethod(combinationsOfTwo)

    def poisson(x, u):
        return (pow(u, x) * math.exp(-u)) / SNLAGraph.fact(x)
    poisson = staticmethod(poisson)

    def probabilityOfCoOccurrences(C, N, M, cooccurrences):
        # C is the total number of nodes in the network
        # N is the total number of link counts in the network
        # M is the total number of edges in the network

        # compute mean bucket hits...
        u = float(N) / float(M)

        # compute sum of poisson probability less than n0
        sum = 0.0
        for n in range(0, cooccurrences):
            sum += SNLAGraph.poisson(n, u)

        # compute probability of all buckets having < cooccurrences hits...
        p = pow(sum, M)

        # this can be used directly as the confidence that the given
        # number of co-occurrences did not occur randomly...
        return p
    probabilityOfCoOccurrences = staticmethod(probabilityOfCoOccurrences)

    #def computeHypotheticalRequiredCooccurrences(nodeCount, linkCount, edgeCount,
    confidence):
    #    return SNLAGraph.probabilityOfCoOccurrences(nodeCount, linkCount, edgeCount,
    cooccurrences)
```

```

    # computeHypotheticalRequiredCooccurrences =
    staticmethod(computeHypotheticalRequiredCooccurrences)

    # interface function for computing confidence at a
    # given number of co-occurrences...
    def computeConfidence(self, cooccurrences):
        # compute the network statistics...
        nodeCount = len(self.__table.getAllRows())
        linkCount = 0
        for (r,c,v) in self.__table.getAll():
            linkCount += v

        # getAll() returns both (x,y) and (y,x) so must divide by two...
        linkCount = 0.50*linkCount
        # use the total number of possible edges in the random network...
        edgeCount = nodeCount * (nodeCount - 1)

        # compute the probability of the given number of
        # cooccurrences in a random graph with the same
        # overall network statistics...
        return SNLAGraph.probabilityOfCoOccurences(nodeCount, linkCount, edgeCount,
cooccurrences)

    # interface function for computing number of co-occurrences
    # required for a given level of confidence...
    def computeRequiredCooccurrences(self, confidence):
        # using NEW definition of confidence...
        n = 1
        c = 0.0
        while (c < confidence):
            n += 1
            c = self.computeConfidence(n)
        return n

    # addSynonymns() adds a list of gene synonyms to the internal table.
    # an optional weight argument is included but is mostly useful in
    # testing...
    def addSynonyms(self, synonyms, weight = 1):
        # cycle over all pairs exactly once...
        count = len(synonyms)
        for n in range(0, count - 1):
            for m in range(n + 1, count):
                # create a pair to enter...
                synA = synonyms[n]
                synB = synonyms[m]

                # update both the row, column and column, row entries
                # in the table since each connection is bi-directional...
                result = self.__table.get(synA, synB) + weight
                self.__table.set(synA, synB, result)
                self.__table.set(synB, synA, result)

    # addSynonymPair() adds a pair of gene synonyms to the internal table.
    # an optional weight argument is included but is mostly useful in
    # testing...
    def addSynonymPair(self, pair, weight = 1):
        # create a pair to enter...
        synA = pair[0]
        synB = pair[1]

        # update both the row, column and column, row entries
        # in the table since each connection is bi-directional...
        result = self.__table.get(synA, synB) + weight
        self.__table.set(synA, synB, result)
        self.__table.set(synB, synA, result)

    # getSynonymPairFrequency() retrieves the number of times that
    # a pair of genes have been added as synonyms.
    def getSynonymPairFrequency(self, synA, synB):
        # return the frequency...
        return self.__table.get(synA, synB)

```

```

# getSynonymsAndFrequency() returns a list of tuples consisting
# of two synonyms and the number of times that they have been
# added together in the synonym table using addSynonym().
def getSynonymsAndFrequency(self):
    # get all the entries from the table...
    entries = self.__table.getAll()

    # because the table is diagonal symmetric there will be
    # duplicates that need to be removed...
    clean = []
    for (synA, synB, val) in entries:
        if (synB, synA, val) not in clean:
            clean.append((synA, synB, val))

    # cleaning done...
    return clean

# clone creates a copy of the SNLAGraph object...
def clone(self):
    g = SNLAGraph()
    g.__table = self.__table.clone()
    return g

# getAll() returns a list of tuples actually stored in the graph,
# each consisting of the row key, the column key, and the
# value at that location in the table.
def getAll(self):
    return self.__table.getAll()

# getStrongestChainsFrom() will return the maximum value for the paths
# between synA and all other reachable nodes, where the value for a
# specific path is the "weakest link" in the path, that is, the
# minimum non-zero frequency in the path.
def getStrongestChainsFrom(self, synA, minFollow):
    # initialize visited dictionary...
    visited = {}

    # initialize best chain value dictionary...
    best = {}

    # initialize current node...
    # the connection between synA and itself is set to zero
    # which represents no connection...
    current = synA
    best[current] = 0

    # initialize the ondeck nodes...
    ondeck = [current]

    while ondeck:
        # remove a node from the ondeck list...
        current = ondeck[0]
        del ondeck[0]

        # mark the current node as visited...
        visited[current] = 1

        # initialize the adjacent list for the current node...
        adjacent = self.__table.getRowKeys(current)

        # remove links that are not to be followed...
        tempAdjacent = adjacent
        adjacent = []
        for next in tempAdjacent:
            if self.__table.get(current, next) >= minFollow:
                adjacent.append(next)

        # debugging aid...
        # print current, adjacent

```

```

while adjacent:
    # remove a node from the adjacent list...
    next = adjacent[0]
    del adjacent[0]

    # get the best value for getting to the current node...
    bestCurrent = best[current]

    # debugging aid...
    # print "bestCurrent is", current, bestCurrent

    # get the best value so far for getting to the next node
    # or 0 if there is none...
    bestNext = best.get(next, 0)

    # debugging aid...
    # print "bestNext so far is", next, bestNext

    strength = self.__table.get(current, next)
    if strength > 0:
        # the value for getting to the next node
        # is the MAX of the current value for bestNext
        # and the MIN of the current value for bestCurrent
        # and the non-zero link strength between the current
        # and the next node...
        if bestCurrent == 0:
            best[next] = max(bestNext, strength)
        else:
            best[next] = max(bestNext, min(bestCurrent, strength))

        # debugging aid...
        # print "bestNext is now", next, best[next]

        # since there is a connection between the current and
        # next nodes, put the next node in the ondeck list
        # if it has not been visited...
        if visited.get(next, 0) == 0:
            ondeck.append(next)

    # add links that were not followed...
    for next in self.__table.getRowKeys(synA):
        direct = self.__table.get(synA, next)
        if direct > best.get(next, 0):
            best[next] = direct

    # return the list of nodes reachable from synA and the
    # strength of the chain connecting them, removing the connection
    # between synA and itself...
    del best[synA]
    return best.items()

# getStrongestChains() returns a list of the strongest chains between
# each pair of synonyms for which there is a chain.
# the optional minFollow parameter specifies the minimum link strength
# that will be included in a longer chain...
def getStrongestChains(self, minFollow=1):
    # first get all the synonyms stored in the table...
    all = self.__table.getAllRows()

    chains = []
    for synA in all:
        # get the strongest chains for each synonym...
        for (synB, strength) in self.getStrongestChainsFrom(synA, minFollow):
            # order the synonym pair alphabetically and construct
            # the chain tuple...
            if synA < synB:
                chain = (synA, synB, strength)
            else:
                chain = (synB, synA, strength)

            # add each chain to the chains list...

```

```

        # note that the chains may not be unique
        # and we will have to clean this up with
        # a post-processing phase, which is better
        # than checking for uniqueness here...
        chains.append(chain)

    # sort and extract each unique chain in the chains list...
    tempChains = chains
    chains = []
    tempChains.sort()
    prevChain = (None, None, None)
    for chain in tempChains:
        if chain[0] != prevChain[0] or chain[1] != prevChain[1]:
            chains.append(chain)
            prevChain = chain

    # return all strongest synonym chains...
    return chains

# computes the Clustering Coefficient (CC) for an individual
# node in the graph...
def computeCC(self, node, neighbors):
    # make sure node is not a neighbor of itself...
    # this is commented out, since it should not be needed,
    # as this is taken care of in the patternManager
    # convertTupleToUppercaseGeneNamePairList function...
    # if node in neighbors:
    #     neighbors.remove(node)

    # compute the CC between pairs of neighbors...
    num = len(neighbors)
    cc = 0.0 # forces floating point addition and division...

    # must have at least 2 neighbors for CC to be non-zero...
    if num > 1:
        for a in range(0, num-1):
            for b in range(a+1, num):
                cc += self.__table.get(neighbors[a], neighbors[b])
        cc = cc / SNLAGraph.combinationsOfTwo(num)
    return cc

# computes the Mean Clustering Coefficient (MCC) for the graph...
def computeMCC(self):
    nodes = self.__table.getAllRows()
    C = 0
    sum = 0.0
    for node in nodes:
        neighbors = self.__table.getRowKeys(node)

        # only include nodes with > 1 neighbor since
        # these are the only ones that could have a
        # non-zero CC...
        if len(neighbors) > 1:
            C += len(neighbors)
            sum += self.computeCC(node, neighbors)
    if C == 0:
        return 0.0
    else:
        return sum / C

# computes the Non-Clustering Coefficient (NCC) for an
# individual node in the graph...
def computeNCC(self, node, allNodes):
    # get all the neighbors of node...
    neighbors = self.__table.getRowKeys(node)

    # the non-neighbors list is list of all nodes, minus the neighbors...
    notneighbors = allNodes[:] # makes top level copy of the allNodes list
    for node in neighbors:
        notneighbors.remove(node)

```

```

# size the lists...
numN = len(neighbors)
numNN = len(notneighbors)

# old method...
# loop through all combinations of neighbors and not neighbors...
# note that these are disjoint lists...
# oldncc = 0.0
# for a in neighbors:
#     for b in notneighbors:
#         oldncc += self.__table.get(a,b)

# loop through all combinations of neighbors and not neighbors...
# note that these are disjoint lists...
# optimization verified by comparison of values
# to old method above...
ncc = 0.0
for a in neighbors:
    # optimization...
    # get the neighbors of the neighbor a...
    neighborsOfA = self.__table.getRowKeys(a)
    # loop through the neighbors of a, adding
    # in the counts for those that are not
    # neighbors of the current node...
    for b in neighborsOfA:
        if b not in neighbors:
            ncc += self.__table.get(a,b)

# divide by the number of pairs and return...
return ncc / (numN * numNN)

# computes the Mean Non-Clustering Coefficient (MNCC) for the graph...
def computeMNCC(self):
    allNodes = self.__table.getAllRows()
    C = len(allNodes)
    if C == 0:
        return 0.0
    sum = 0.0
    for node in allNodes:
        sum += self.computeNCC(node, allNodes)
    return sum / C

# computes graph quality, which is really the quality of the
# pattern that created this graph, as the ratio of MCC to MNCC...
def computeQuality(self):
    mcc = self.computeMCC()
    mncc = self.computeMNCC()
    return mcc / (mncc + QUALITY_COMPUTATION_OFFSET)

# computes node quality, which is really the ratio of
# MCC to MNCC for a given node...
def computeNodeQuality(self, node):
    neighbors = self.__table.getRowKeys(node)
    allNodes = self.__table.getAllRows()
    cc = self.computeCC(node, neighbors)
    ncc = self.computeNCC(node, allNodes)
    return cc / (ncc + QUALITY_COMPUTATION_OFFSET)

```

## Source file: Table.py

```
#
# Table.py
# Implements an automatically extending table using two levels of
# Python dictionaries.
#
class Table:
    # construct the table giving the default
    # value for table entries as an optional
    # argument.
    def __init__(self, default=0):
        self.__defaultValue = default
        self.__rowDict = {}

    def set(self, rowkey, columnkey, value):
        # get the column dictionary for the row
        # key value, or an empty dictionary if
        # there is none...
        colDict = self.__rowDict.get(rowkey, {})

        # save the value to the column key...
        colDict[columnkey] = value

        # put the column dictionary back in the row dictionary...
        self.__rowDict[rowkey] = colDict

    def get(self, rowkey, columnkey):
        # get the column dictionary for the row
        # key value, or None if there is none...
        colDict = self.__rowDict.get(rowkey, None)

        # if no column dictionary, return the default value
        # otherwise look up the column key returning the
        # default if the key is not present there...
        if not colDict:
            return self.__defaultValue
        else:
            return colDict.get(columnkey, self.__defaultValue)

    # getRowKeys() returns the list of all valid columns in this row.
    # if the rowkey is invalid return an empty list.
    def getRowKeys(self, rowkey):
        colDict = self.__rowDict.get(rowkey, None)
        if colDict:
            return colDict.keys()
        else:
            return []

    # getAll() returns a list of tuples actually stored in the graph,
    # each consisting of the row key, the column key, and the
    # value at that location in the table.
    def getAll(self):
        entries = []
        for r in self.__rowDict.keys():
            for c in self.__rowDict[r]:
                entries.append((r, c, self.get(r,c)))
        return entries

    # getAllRows() returns the list of all valid row keys.
    def getAllRows(self):
        return self.__rowDict.keys()

    # hasRow() tests if a given entry is a valid row key.
    def hasRow(self, key):
        return self.__rowDict.hasKey(key)

    # clone creates a new table object that is equivalent
    # to the original...
    # does NOT clone the individual key, but these have to be
```



```
# immutable anyway.  
# does NOT clone the individual stored values, so these should be  
# numbers or strings or some other immutable object  
def clone(self):  
    t = Table()  
    for r in self.getAllRows():  
        t.__rowDict[r] = self.__rowDict[r].copy()  
    return t
```

## Source file: ExtractAbstractSentences.py

```
#
# Module: ExtractAbstractSentences.py
#
# Python program to extract sentences from the abstract field of
# MEDLINE records and break them into sentences.
#
# Uses only lexical information.
#
# Takes file of MEDLINE records as STDIN and outputs one line for
# each sentence in each abstract to STDOUT.
#
# Implements a set of heuristic rules to find sentence boundaries.
#
# Rules:
# 1. Periods, question marks, and exclamation points denote sentence
# boundaries except in the cases that follow.
# 2. Periods, question marks, and exclamation points inside parenthesis
# or brackets do not denote sentence boundaries.
# 3. Periods that are preceded and followed by numbers do not denote
# sentence boundaries. A trailing percent sign is optional.
# 4. Periods that are followed by any number of spaces and a word
# beginning with a lower case letter do not denote sentence boundaries.
# 5. Periods that follow an abbreviation and are followed by an
# abbreviation do not denote sentence boundaries. (see Mikheev)
# 6. Words containing all capitals or a mix of capitals and lowercase,
# besides initial capitalization, or a mix of letters and numbers
# will be considered abbreviations.
# 7. Words consisting of a single letter or two non-vowel letters will
# be considered abbreviations.

import sys
import Medline

# initialize constants..
ABSTRACT_FIELD_CODE = "AB"

# sentence requires trailing space, gets around use of period in numbers...
SENTENCE_DELIMITER = "."
QUESTION_DELIMITER = "?"
EXCLAMATION_DELIMITER = "!"

# list of upper and lower case vowels...
VOWEL_LIST = "aeiouyAEIOUY"

# list of token delimiters...
TOKEN_DELIMITER_LIST = " .,/\\-+(){}[]=;"

# initialize counters...
abstractCount = 0
sentenceCount = 0

# helper function to test for sentence period inside parenthesis or brackets..
def insideMatchedPair( pos, openPos, closePos):
    if openPos >= 0 and closePos >= 0 and pos > openPos and pos < closePos:
        return 1
    else:
        return 0

def isNextCharLowerOrComma(text, pos):
    end = len(text)
    pos = pos + 1
    while pos < end and text[pos].isspace():
        pos = pos + 1
    if pos == end:
        return 0
    elif text[pos] == ",":
        return 1
    else:
```

```

        return text[pos].islower()

def isDelimiterChar(s, pos):
    if TOKEN_DELIMITER_LIST.find(s[pos]) == -1:
        return 0
    else:
        return 1

def splitAroundDelimiter(text, pos):
    # skip leading and trailing spaces...
    textLen = len(text)
    lpos = pos - 1
    while lpos > 0 and text[lpos].isspace():
        lpos -= 1
    rpos = pos + 1
    while rpos < textLen and text[rpos].isspace():
        rpos += 1

    # isolate tokens on either side...
    lpos2 = lpos
    while lpos2 > 0 and not isDelimiterChar(text, lpos2):
        lpos2 -= 1
    rpos2 = rpos
    while rpos2 < textLen and not isDelimiterChar(text, rpos2):
        rpos2 += 1

    # extract pre-and post tokens...
    return (text[lpos2 + 1:pos], text[rpos:rpos2])

def isAbbreviation(s):
    textLen = len(s)

    # eliminate initial capitalization case...
    if textLen > 1 and s[0].isupper() and s[1:].islower():
        return 0

    # abbreviation if multiple all upper case...
    if textLen > 1 and s.isupper():
        return 1

    # count upper, lower, and numbers...
    upCount = 0
    lowCount = 0
    numCount = 0
    for t in s:
        upCount += t.isupper()
        lowCount += t.islower()
        numCount += t.isdigit()

    # mixes are abbreviations...
    if upCount > 0 and lowCount > 0:
        return 1
    if upCount > 0 and numCount > 0:
        return 1
    if lowCount > 0 and numCount > 0:
        return 1

    # check for short words...
    if textLen == 1 and VOWEL_LIST.find(s[0]) == -1:
        return 1
    if textLen == 2 and VOWEL_LIST.find(s[0]) == -1 and VOWEL_LIST.find(s[1]) == -1:
        return 1

    # otherwise return no abbreviation...
    return 0

def isBetweenAbbreviationsOrNumbers(text, pos):
    (preceedor, successor) = splitAroundDelimiter(text, pos)
    if preceedor.isdigit() and successor.isdigit():
        return 1
    if preceedor.isdigit() and successor[:-1].isdigit() and successor[-1] == '%':

```

```

        return 1
    if isAbbreviation(precedor) and isAbbreviation(succeedor):
        return 1
    if isAbbreviation(precedor) and succeedor=="": # between abbrev and delimiter
        return 1
    return 0

# loop over all fields in all records in STDIN...
medline = Medline.Medline(sys.stdin)
field = medline.readField()
while field:
    # extract the header and field text...
    (header, text, flag) = field

    if header == ABSTRACT_FIELD_CODE:
        # increment count...
        abstractCount += 1

    # break field text into sentences...
    while text:
        # trim whitespace from front...
        text = text.lstrip()

        # make sure sentence does not end prematurely for
        # several possible reasons...
        pos = -1
        while 1:
            # find position of next sentence delimiter...
            spos = text.find(SENTENCE_DELIMITER, pos + 1)
            qpos = text.find(QUESTION_DELIMITER, pos + 1)
            epos = text.find(EXCLAMATION_DELIMITER, pos + 1)
            if spos < 0:
                spos = max(spos, qpos, epos)
            if qpos < 0:
                qpos = max(spos, qpos, epos)
            if epos < 0:
                epos = max(spos, qpos, epos)
            pos = min(spos, qpos, epos)

            # make sure that we found one...
            if pos < 0:
                break

            # find position of next open parenthesis...
            openParen = text.rfind("(", 0, pos)
            # find position of next close parenthesis...
            if openParen >= 0:
                closeParen = text.find(")", openParen)
            else:
                closeParen = -1

            # find position of next open bracket...
            openBracket = text.rfind("[", 0, pos)
            # find position of next close bracket...
            if openBracket >= 0:
                closeBracket = text.find("]", openBracket)
            else:
                closeBracket = -1

            # make sure sentence does not end inside matched
            # pairs of brackets or parenthesis...
            if insideMatchedPair(pos, openParen, closeParen) or
insideMatchedPair(pos, openBracket, closeBracket):
                continue
            # check for period in-between lower case words...
            elif text[pos] == "." and isNextCharLowerOrComma(text, pos):
                continue
            # check for period in between abbreviations...
            elif text[pos] == "." and isBetweenAbbreviationsOrNumbers(text, pos):
                continue
            else:

```

```

        break

    if pos > 0:
        # print until delimiter (including period but not space)...
        print text[0:pos+1]
        # increment count...
        sentenceCount += 1

        # trim off the sentence just output...
        text = text[pos+1:]
    else:
        # no more delimiters, treat rest of text as one sentence...
        if len(text) > 0:
            print text
            # increment count...
            sentenceCount += 1
        text = None

# show progress...
sys.stderr.write("\r%d abstracts/%d sentences..." % (abstractCount, sentenceCount))

# read next field...
field = medline.readField()

# final progress counts...
sys.stderr.write("\r%d abstracts/%d sentences...Done.\n" % (abstractCount,
sentenceCount))

```

## Source file: ScreenSentences.py

```
#
# ScreenSentences.py
#
# Read a list of sentences from STDIN, and write to STDOUT
# only the sentences that have at least two possible GENE
# or PROTEIN name symbols. These are the only sentences that
# could provide evidence to SNLASynonyms.py and will reduce
# the size of the data set, speeding up the knowledge
# extraction process.

# general Python library imports...
import sys

# import SNLA-specific classes...
import SNLAPatterns

# create SNLAPatterns object to manage pattern creation and matching...
patternManager = SNLAPatterns.SNLAPatterns()

# line counters...
linesRead = 0
linesWritten = 0

# read through each line of stdin...
for line in sys.stdin.xreadlines():
    # count line...
    linesRead += 1

    # remove leading and trailing whitespace...
    line = line.strip()

    # break into a token list...
    tokens = patternManager.convertSentenceToTokenList(line)

    # loop through tokens and count potential GENE/PROTEIN
    # names, as well as the space between them...
    prevName = -(SNLAPatterns.MAXIMUM_EXTRACTED_DISTANCE + 2)
    minDist = SNLAPatterns.MAXIMUM_EXTRACTED_DISTANCE + 1
    count = 0
    for n in xrange(len(tokens)):
        if patternManager.isGeneName(tokens[n], n):
            count += 1
            d = n - prevName - 1 # distance is num tokens in-between
            if d < minDist:
                minDist = d
            prevName = n

    # output line if count is at least 2...
    if count >= 2 and minDist <= SNLAPatterns.MAXIMUM_EXTRACTED_DISTANCE:
        print line
        linesWritten += 1

    # update status...
    sys.stderr.write("\rRead %d lines from text file, wrote %d lines..." %
        (linesRead, linesWritten))
```

## APPENDIX B - ERROR ANALYSIS RAW DATA

### Precision Error Analysis Raw Data

| NGN   | <i>Not a Gene Name</i>                              |    |   |   |   |   |                             |
|-------|---|----|---|---|---|---|-----------------------------|
| PGN   | <i>Partial Gene Name</i>                            |    |   |   |   |   |                             |
| BR    | <i>Biochemically Related</i>                        |    |   |   |   |   |                             |
| UG    | <i>Unrelated Genes</i>                              |    |   |   |   |   |                             |
| MV    | <i>Mutation Variants</i>                            |    |   |   |   |   |                             |
| CT    | <i>Correct</i>                                      |    |   |   |   |   |                             |
|       |   | NP | G | B | U | M |                             |
|       |   | N  | R | G | V | T |                             |
| INDEX | ERROR   |    |   |   |   |   | Reason                      |
| 1     | INCORRECT PAIR: HSWI PBAF, 1 co-occurrences         | X  |   |   |   |   |                             |
| 2     | INCORRECT PAIR: MAPK P38, 33 co-occurrences         | X  |   |   |   |   |                             |
| 3     | INCORRECT PAIR: HIF PVHL, 1 co-occurrences          |    | X |   |   |   |                             |
| 4     | INCORRECT PAIR: IL-23 P40, 1 co-occurrences         |    | X |   |   |   |                             |
| 5     | INCORRECT PAIR: CRY1AC CRY1D, 1 co-occurrences      | X  |   |   |   |   | primer names                |
| 6     | INCORRECT PAIR: FK228 FR901228, 2 co-occurrences    | X  |   |   |   |   | drug name                   |
| 7     | INCORRECT PAIR: GLN HBD-2, 2 co-occurrences         | X  |   |   |   |   | amino acid name             |
| 8     | INCORRECT PAIR: ETV6 NERF, 1 co-occurrences         |    | X |   |   |   |                             |
| 9     | INCORRECT PAIR: GAG LLO, 3 co-occurrences           |    | X |   |   |   |                             |
| 10    | INCORRECT PAIR: Y170F Y181F, 1 co-occurrences       |    | X |   |   |   |                             |
| 11    | INCORRECT PAIR: GLEEVEC GLIVEC, 1 co-occurrences    | X  |   |   |   |   |                             |
| 12    | INCORRECT PAIR: FUZ7 UBC3, 1 co-occurrences         |    | X |   |   |   |                             |
| 13    | INCORRECT PAIR: IGH TCR, 1 co-occurrences           |    | X |   |   |   |                             |
| 14    | INCORRECT PAIR: HDAC SMAD3, 1 co-occurrences        |    | X |   |   |   |                             |
| 15    | INCORRECT PAIR: ADD1 SREBP1C, 1 co-occurrences      | X  |   |   |   |   |                             |
| 16    | INCORRECT PAIR: P130 PFAK, 1 co-occurrences         | X  |   |   |   |   | P130 CAS is the full symbol |
| 17    | INCORRECT PAIR: A20 IKAPPABALPHA, 1 co-occurrences  |    | X |   |   |   |                             |
| 18    | INCORRECT PAIR: LPPB NLPD, 2 co-occurrences         |    |   |   | X |   | PMID: 10948113              |
| 19    | INCORRECT PAIR: FR2A IGH, 1 co-occurrences          |    | X |   |   |   |                             |
| 20    | INCORRECT PAIR: IGF-I IGF-II, 1 co-occurrences      |    | X |   |   |   |                             |
| 21    | INCORRECT PAIR: DIAPHANOUS MDIA, 1 co-occurrences   |    |   |   | X |   | PMID: 12847276              |
| 22    | INCORRECT PAIR: IRS-2 JAK2, 1 co-occurrences        |    | X |   |   |   |                             |
| 23    | INCORRECT PAIR: GALPHA GPR, 1 co-occurrences        |    | X |   |   |   |                             |
| 24    | INCORRECT PAIR: JNK THR183, 1 co-occurrences        | X  |   |   |   |   | amino acid site             |
| 25    | INCORRECT PAIR: NR2B NR2D, 1 co-occurrences         |    | X |   |   |   |                             |
| 26    | INCORRECT PAIR: KIR NKG2A, 1 co-occurrences         |    | X |   |   |   |                             |
| 27    | INCORRECT PAIR: LEU PHE, 1 co-occurrences           | X  |   |   |   |   | amino acid names            |
| 28    | INCORRECT PAIR: P-0.48 P-0.52, 1 co-occurrences     |    |   |   | X |   |                             |
| 29    | INCORRECT PAIR: TH2 TH3, 1 co-occurrences           |    | X |   |   |   |                             |
| 30    | INCORRECT PAIR: GAS PHR, 1 co-occurrences           |    | X |   |   |   |                             |
| 31    | INCORRECT PAIR: HNF-1ALPHA MODY3, 1 co-occurrences  | X  |   |   |   |   |                             |
| 32    | INCORRECT PAIR: CYS106ALA CYS7ALA, 1 co-occurrences |    |   |   | X |   |                             |
| 33    | INCORRECT PAIR: CACS PS-CAC, 1 co-occurrences       | X  |   |   |   |   |                             |

|    |   |   |   |   |   |                        |  |
|----|---|---|---|---|---|------------------------|--|
| 34 | INCORRECT PAIR: STRABISMUS VAN, 1 co-occurrences              | X |   |   |   |                        |  |
| 35 | INCORRECT PAIR: DHFR PHE22, 2 co-occurrences                  | X |   |   |   |                        |  |
| 36 | INCORRECT PAIR: ETF ETFDH, 1 co-occurrences                   |   | X |   |   |                        |  |
| 37 | INCORRECT PAIR: DQB1*05 HLA-DQA1*01, 2 co-occurrences         |   |   | X |   |                        |  |
| 38 | INCORRECT PAIR: MIN TCF, 6 co-occurrences                     | X |   |   |   |                        |  |
| 39 | INCORRECT PAIR: CIP1 E2F-1, 1 co-occurrences                  |   | X |   |   |                        |  |
| 40 | INCORRECT PAIR: K-RAS TMDELTA4A, 2 co-occurrences             |   |   | X |   |                        |  |
| 41 | INCORRECT PAIR: AC5 GALPHAOLF, 1 co-occurrences               |   | X |   |   |                        |  |
| 42 | INCORRECT PAIR: AKT RHOA, 1 co-occurrences                    |   | X |   |   |                        |  |
| 43 | INCORRECT PAIR: EGFR FKBP12, 1 co-occurrences                 |   | X |   |   |                        |  |
| 44 | INCORRECT PAIR: ASAYAMA K-81, 1 co-occurrences                | X |   |   |   |                        |  |
| 45 | INCORRECT PAIR: PL5 PL8, 1 co-occurrences                     |   | X |   |   |                        |  |
| 46 | INCORRECT PAIR: CD31 CD34, 1 co-occurrences                   |   | X |   |   |                        |  |
| 47 | INCORRECT PAIR: CHRYSOCHUS COLEOPTERA, 1 co-occurrences       | X |   |   |   |                        |  |
| 48 | INCORRECT PAIR: CDK4 PRB, 2 co-occurrences                    |   | X |   |   |                        |  |
| 49 | INCORRECT PAIR: D-AN IIT, 1 co-occurrences                    | X |   |   |   |                        |  |
| 50 | INCORRECT PAIR: CIS-33 SSI-3, 1 co-occurrences                |   |   |   | X | PMID: 12591901         |  |
| 51 | INCORRECT PAIR: ABL1 BCR, 1 co-occurrences                    |   | X |   |   |                        |  |
| 52 | INCORRECT PAIR: DPJ9 GROEL, 4 co-occurrences                  | X |   |   |   |                        |  |
| 53 | INCORRECT PAIR: FRENCH-ACADIAN KOREAN, 1 co-occurrences       | X |   |   |   |                        |  |
| 54 | INCORRECT PAIR: EGLNS PHDS, 1 co-occurrences                  |   |   |   | X | PMID: 14695194         |  |
| 55 | INCORRECT PAIR: IOC3P ISWLP, 1 co-occurrences                 | X |   |   |   |                        |  |
| 56 | INCORRECT PAIR: MMP-2 TIMP-2, 2 co-occurrences                |   | X |   |   |                        |  |
| 57 | INCORRECT PAIR: RASSF1A ROBO1, 1 co-occurrences               |   | X |   |   |                        |  |
| 58 | INCORRECT PAIR: BPI LBP, 1 co-occurrences                     |   | X |   |   |                        |  |
| 59 | INCORRECT PAIR: HTRA2 SMAC, 1 co-occurrences                  |   | X |   |   |                        |  |
| 60 | INCORRECT PAIR: P55 P68, 1 co-occurrences                     |   | X |   |   |                        |  |
| 61 | INCORRECT PAIR: GLUT SLC2A, 1 co-occurrences                  |   | X |   |   |                        |  |
| 62 | INCORRECT PAIR: ERK5 MEK5, 2 co-occurrences                   |   | X |   |   |                        |  |
| 63 | INCORRECT PAIR: CAVEOLIN-1 ERK, 1 co-occurrences              |   | X |   |   |                        |  |
| 64 | INCORRECT PAIR: BACTEROIDES CYTOPHAGA, 1 co-occurrences       | X |   |   |   |                        |  |
| 65 | INCORRECT PAIR: ACS PRPE, 1 co-occurrences                    |   | X |   |   |                        |  |
| 66 | INCORRECT PAIR: MT1-MMP TIMP-2, 3 co-occurrences              |   | X |   |   |                        |  |
| 67 | INCORRECT PAIR: NOD RIPE3, 1 co-occurrences                   | X |   |   |   |                        |  |
| 68 | INCORRECT PAIR: ERK1 MEK-1, 1 co-occurrences                  |   | X |   |   |                        |  |
| 69 | INCORRECT PAIR: INTERLEUKIN-4 INTERLEUKIN-5, 1 co-occurrences |   | X |   |   |                        |  |
| 70 | INCORRECT PAIR: MCK MST, 1 co-occurrences                     |   | X |   |   |                        |  |
| 71 | INCORRECT PAIR: MIXED-LINEAGE MLL, 1 co-occurrences           | X |   |   |   |                        |  |
| 72 | INCORRECT PAIR: MAPK SRC, 2 co-occurrences                    |   | X |   |   |                        |  |
| 73 | INCORRECT PAIR: ARC C3H, 1 co-occurrences                     |   |   | X |   |                        |  |
| 74 | INCORRECT PAIR: FYA FYB, 1 co-occurrences                     |   |   |   | X | alleles, PMID: 9746760 |  |
| 75 | INCORRECT PAIR: BCFII EBP, 1 co-occurrences                   |   | X |   |   |                        |  |
| 76 | INCORRECT PAIR: DC-SIGNR L-SIGN, 1 co-occurrences             |   | X |   |   |                        |  |
| 77 | INCORRECT PAIR: BERLIN GERMANY, 2 co-occurrences              | X |   |   |   |                        |  |
| 78 | INCORRECT PAIR: EGF HGF, 1 co-occurrences                     |   | X |   |   |                        |  |
| 79 | INCORRECT PAIR: ADTG5327 IL2, 1 co-occurrences                | X |   |   |   |                        |  |
| 80 | INCORRECT PAIR: C-JUN P44-MAP, 2 co-occurrences               | X |   |   |   |                        |  |



|     |   |   |   |   |   |   |                |
|-----|---|---|---|---|---|---|----------------|
| 81  | INCORRECT PAIR: APOB-38.9-ONLY APOB38.9, 1 co-occurrences | X |   |   |   |   |                |
| 82  | INCORRECT PAIR: CREB ERK2, 2 co-occurrences               |   | X |   |   |   |                |
| 83  | INCORRECT PAIR: AURORA STK15, 1 co-occurrences            | X |   |   |   |   |                |
| 84  | INCORRECT PAIR: GUSN INTN, 1 co-occurrences               |   | X |   |   |   |                |
| 85  | INCORRECT PAIR: FAS INF, 1 co-occurrences                 |   |   | X |   |   |                |
| 86  | INCORRECT PAIR: R1448 R1448C, 1 co-occurrences            | X |   |   |   |   |                |
| 87  | INCORRECT PAIR: DELTACDTABC DELTALTXA, 1 co-occurrences   |   |   |   | X |   |                |
| 88  | INCORRECT PAIR: A11 CD40L, 1 co-occurrences               | X |   |   |   |   |                |
| 89  | INCORRECT PAIR: BXPC-3 EGF, 1 co-occurrences              | X |   |   |   |   |                |
| 90  | INCORRECT PAIR: D138E FECR, 1 co-occurrences              |   |   |   | X |   |                |
| 91  | INCORRECT PAIR: MALK MALR, 1 co-occurrences               | X |   |   |   |   |                |
| 92  | INCORRECT PAIR: ERK P44-MAP, 2 co-occurrences             | X |   |   |   |   |                |
| 93  | INCORRECT PAIR: CAMP CGMP, 1 co-occurrences               | X |   |   |   |   |                |
| 94  | INCORRECT PAIR: PRP19 PS04, 1 co-occurrences              |   |   |   |   | X | PMID: 12960389 |
| 95  | INCORRECT PAIR: SP-B THREE-FOLD, 1 co-occurrences         | X |   |   |   |   |                |
| 96  | INCORRECT PAIR: EMP1 EMP3, 1 co-occurrences               |   | X |   |   |   |                |
| 97  | INCORRECT PAIR: CGI-BIN SCAN.CGB.KI.SE, 1 co-occurrences  | X |   |   |   |   |                |
| 98  | INCORRECT PAIR: ALR HPO, 1 co-occurrences                 |   |   |   |   | X | PMID: 12681483 |
| 99  | INCORRECT PAIR: MK6ALPHA MK6BETA, 1 co-occurrences        |   | X |   |   |   |                |
| 100 | INCORRECT PAIR: CREB GATA-1, 1 co-occurrences             |   | X |   |   |   |                |

# Recall Error Analysis Raw Data

| ERROR                          | INDEX | No Matching Pattern | Pattern not Accepted |
|--------------------------------|-------|---------------------|----------------------|
| RECALL FAILURE: LIMK LIMK1     | 1     | X                   |                      |
| RECALL FAILURE: CD40L TNFSF5   | 2     | X                   |                      |
| RECALL FAILURE: ZP2 ZPA        | 3     | X                   |                      |
| RECALL FAILURE: NXF1 TAP       | 4     |                     | X                    |
| RECALL FAILURE: BLC CXCL13     | 5     |                     | X                    |
| RECALL FAILURE: CD94 KLRD1     | 6     |                     | X                    |
| RECALL FAILURE: BCL-6 BCL6     | 7     | X                   |                      |
| RECALL FAILURE: PON PON1       | 8     | X                   |                      |
| RECALL FAILURE: TDH TDH1       | 9     | X                   |                      |
| RECALL FAILURE: TRPC TRPC1     | 10    | X                   |                      |
| RECALL FAILURE: AHC NR0B1      | 11    | X                   |                      |
| RECALL FAILURE: ABCC7 CFTR     | 12    | X                   |                      |
| RECALL FAILURE: KLF2 LKLF      | 13    | X                   |                      |
| RECALL FAILURE: RFXANK RFXB    | 14    | X                   |                      |
| RECALL FAILURE: DAT1 SLC6A3    | 15    |                     | X                    |
| RECALL FAILURE: HE4 WFDC2      | 16    | X                   |                      |
| RECALL FAILURE: NR2C1 TR2      | 17    | X                   |                      |
| RECALL FAILURE: ADH1C ADH3     | 18    | X                   |                      |
| RECALL FAILURE: LIS1 PAFAH1B1  | 19    | X                   |                      |
| RECALL FAILURE: MEF2 MEF2A     | 20    | X                   |                      |
| RECALL FAILURE: GAD2 GAD65     | 21    | X                   |                      |
| RECALL FAILURE: THI3 YDL080C   | 22    |                     | X                    |
| RECALL FAILURE: BSAC MKL1      | 23    | X                   |                      |
| RECALL FAILURE: ASNS TS11      | 24    |                     | X                    |
| RECALL FAILURE: AML1 CBFA2     | 25    | X                   |                      |
| RECALL FAILURE: MCAM MUC18     | 26    |                     | X                    |
| RECALL FAILURE: A33R A36R      | 27    | X                   |                      |
| RECALL FAILURE: CBP CREBBP     | 28    |                     | X                    |
| RECALL FAILURE: IB1 MAPK8IP1   | 29    | X                   |                      |
| RECALL FAILURE: JIP3 JSAP1     | 30    | X                   |                      |
| RECALL FAILURE: FBNP3 HYPA     | 31    |                     | X                    |
| RECALL FAILURE: STE24 ZMPSTE24 | 32    | X                   |                      |
| RECALL FAILURE: MTH1 NUDT1     | 33    | X                   |                      |
| RECALL FAILURE: ALOX5AP FLAP   | 34    |                     | X                    |
| RECALL FAILURE: ABCB11 BSEP    | 35    | X                   |                      |
| RECALL FAILURE: AROM CYP19     | 36    | X                   |                      |
| RECALL FAILURE: CD52 HE5       | 37    |                     | X                    |
| RECALL FAILURE: ABCD3 PMP70    | 38    |                     | X                    |
| RECALL FAILURE: ZIC ZIC1       | 39    | X                   |                      |
| RECALL FAILURE: IL-2 IL2       | 40    | X                   |                      |
| RECALL FAILURE: AAT SERPINA1   | 41    | X                   |                      |
| RECALL FAILURE: ACVRL1 ALK1    | 42    |                     | X                    |
| RECALL FAILURE: KLK11 TLSP     | 43    | X                   |                      |
| RECALL FAILURE: TPH TPH1       | 44    | X                   |                      |

|                                 |    |   |   |
|---------------------------------|----|---|---|
| RECALL FAILURE: FTZF1 SF1       | 45 |   | X |
| RECALL FAILURE: TBP TFIID       | 46 | X |   |
| RECALL FAILURE: MST1R RON       | 47 | X |   |
| RECALL FAILURE: FPRL1 LXA4R     | 48 |   | X |
| RECALL FAILURE: RANKL TRANCE    | 49 | X |   |
| RECALL FAILURE: NOS NOS2        | 50 | X |   |
| RECALL FAILURE: SDCT1 SLC13A2   | 51 |   | X |
| RECALL FAILURE: PDS SLC26A4     | 52 |   | X |
| RECALL FAILURE: TIMP TIMP-1     | 53 | X |   |
| RECALL FAILURE: ABCG2 ABCP      | 54 | X |   |
| RECALL FAILURE: DEFCAP NALP1    | 55 |   | X |
| RECALL FAILURE: HSP70-1 HSP70-2 | 56 |   | X |
| RECALL FAILURE: PDP PDP1        | 57 | X |   |
| RECALL FAILURE: F2R PAR1        | 58 |   | X |
| RECALL FAILURE: SDH SDHB        | 59 | X |   |
| RECALL FAILURE: HTT SLC6A4      | 60 |   | X |
| RECALL FAILURE: HGK NIK         | 61 | X |   |
| RECALL FAILURE: CARD11 CARMA1   | 62 | X |   |
| RECALL FAILURE: A10 OS-D        | 63 |   | X |
| RECALL FAILURE: FKHR FOXO1      | 64 | X |   |
| RECALL FAILURE: IK1 SK4         | 65 |   | X |
| RECALL FAILURE: ASC TMS1        | 66 |   | X |
| RECALL FAILURE: AITR GTR        | 67 | X |   |
| RECALL FAILURE: CALM1 CALM2     | 68 |   | X |
| RECALL FAILURE: ABCC4 MRP4      | 69 |   | X |
| RECALL FAILURE: ABCD1 ALD       | 70 |   | X |
| RECALL FAILURE: AXR3 IAA17      | 71 |   | X |
| RECALL FAILURE: NTRK1 TRK       | 72 | X |   |
| RECALL FAILURE: IBD1 NOD2       | 73 | X |   |
| RECALL FAILURE: FBP FBP1        | 74 | X |   |
| RECALL FAILURE: DAP12 KARAP     | 75 |   | X |
| RECALL FAILURE: RPS6KA3 RSK2    | 76 | X |   |
| RECALL FAILURE: PDC PDC1        | 77 | X |   |
| RECALL FAILURE: C10 CCL6        | 78 | X |   |
| RECALL FAILURE: LDH LDHA        | 79 | X |   |
| RECALL FAILURE: SCYA17 TARC     | 80 | X |   |
| RECALL FAILURE: OPGL TNFSF11    | 81 | X |   |
| RECALL FAILURE: ALAS-E ALAS2    | 82 | X |   |
| RECALL FAILURE: ETV6 TEL        | 83 | X |   |
| RECALL FAILURE: STAT5 STAT5A    | 84 |   | X |
| RECALL FAILURE: HDAC1 RPD3      | 85 | X |   |
| RECALL FAILURE: MRP14 S100A9    | 86 | X |   |
| RECALL FAILURE: CDC40 PRP17     | 87 | X |   |
| RECALL FAILURE: NR1H1 VDR       | 88 | X |   |
| RECALL FAILURE: PMEL17 SILV     | 89 |   | X |
| RECALL FAILURE: SAT1 SLC26A1    | 90 |   | X |
| RECALL FAILURE: BLH1 GAL6       | 91 |   | X |

|                               |     |   |   |
|-------------------------------|-----|---|---|
| RECALL FAILURE: NOS NOS3      | 92  | X |   |
| RECALL FAILURE: F5F8D LMAN1   | 93  | X |   |
| RECALL FAILURE: PEX2 PXMP3    | 94  | X |   |
| RECALL FAILURE: SERT SLC6A4   | 95  | X |   |
| RECALL FAILURE: OPG TNFRSF11B | 96  | X |   |
| RECALL FAILURE: PER1 PER2     | 97  |   | X |
| RECALL FAILURE: HERG KCNH2    | 98  | X |   |
| RECALL FAILURE: PLP PLP1      | 99  | X |   |
| RECALL FAILURE: ECM33 YBR078W | 100 |   | X |