# Computational-Linguistic Approaches to Biological Text Mining

Andrew B. Clegg

March 7, 2008

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy

School of Crystallography
Birkbeck, University of London
Malet Street
London WC1E 7HX
United Kingdom

# Declaration

Parts of Chapter 2 are adapted from Clegg and Shepherd (2005), parts of Chapter 3 from Clegg and Shepherd (2007a), and parts of Chapter 1 and Appendix A from Clegg and Shepherd (2007b). Although these were all joint publications, Dr. Shepherd's input into the text presented herein was purely supervisory.

I hereby declare that this thesis is all my own original work, with all quotations, extracts and derivations from the work of others clearly marked.

# Abstract

As the body of published literature grows at an accelerating rate, increasingly sophisticated computational methods for natural language processing are required to manage and mine the written knowledge available to life sciences researchers. One important topic within this field is the problem of relationship extraction. Given a text about molecular biology, the challenge is to automatically retrieve the biophysical, biochemical or genetic interactions described therein.

Much progress has been made on this problem and others like it by using statistical information retrieval techniques, regular expressions, finite state automata, sequence alignment and other relatively superficial approaches. However, there are a variety of more linguistically-informed methods available which treat each sentence as a tree or graph rather than simply a collection or sequence of words.

Various natural-language parsers are available which facilitate this kind of solution, and the experimental work in this thesis begins with a comparison of several of these on a standard molecular biology corpus using established benchmarking techniques. This is followed by some experiments using evaluation measures tailored to specific biologically-important tasks. A processing pipeline is then described which uses the best of these parsers, along with several other open-source tools, to produce high-quality dependency graph representations of input sentences.

Finally, three novel deterministic algorithms for relationship extraction are presented. Two of these take dependency graphs as input and return interactions between pre-tagged gene and protein entities, outperforming most existing methods on a standard publically-available test corpus; the other is a strong baseline method using no linguistic information. An appendix discusses the related problems of entity recognition and identification, which—while outside the main scope of this thesis—are prerequisites for the development of relationship extraction applications.

# Acknowledgements

The work presented herein was sponsored by a CASE studentship from the Biotechnology and Biological Sciences Research Council and AstraZeneca.

I am greatly indebted to the authors of all the software covered in this thesis for kindly making their code available, and in many cases for helping me with the usual teething problems—particularly Marie-Catherine de Marneffe of Stanford University for her crucial debugging efforts. Additionally I am very grateful to the maintainers of the GENIA and LLL corpora for their painstaking annotation work. Without the tools and the data to test them on, none of this would have been possible.

I would like to thank Adrian Shepherd for his invaluable supervision and for being a constant voice of reason; my examiners, Andrew Martin and Robert Gaizauskas, for all their time and advice; Kenneth Evans for suggesting I investigate genetic algorithms for the LLL experiment; and Ian Dix of AstraZeneca, the readers of the BioNLP mailing list, and the anonymous reviewers of our papers, all of whom provided critical advice and feedback. This thesis is dedicated with love to Amelia Nel for her sympathy, encouragement, and tolerance of my odd working hours, and to my parents, for all their support.

# Contents

# List of Figures

# List of Tables

# Acronyms and Abbreviations

| | |
|---|---|
| ASCII | American Standard Code for Information Interchange |
| CCG | Combinatory Categorial Grammar |
| CFG | Context-Free Grammar |
| CRF | Conditional Random Field |
| $F$ | F-measure (effectiveness) |
| GB | Government and Binding |
| GO | Gene Ontology |
| GPSG | Generalized Phrase Structure Grammar |
| GTB | Genia Treebank |
| HMM | Hidden Markov Model |
| HPSG | Head-driven Phrase Structure Grammar |
| IE | Information Extraction |
| IR | Information Retrieval |
| LFG | Lexical Function Grammar |
| LLL | Learning Language in Logic |
| LTAG | Lexicalized Tree-Adjoining Grammar |
| MEMM | Maximum-Entropy Markov Model |
| MPL | Metapattern Language |
| NLP | Natural Language Processing |
| OBO | Open Biological Ontologies |

| | |
|---|---|
| $P$ | Precision |
| PDF | Portable Document Format |
| POS | Part of Speech |
| PTB | Penn Treebank |
| $R$ | Recall |
| SVM | Support Vector Machine |
| UMLS | Unified Medical Language System |

# Chapter 1

# Introduction

The biological sciences are very much knowledge-driven, with facts recorded in the text of scientific journals being the gold standard of accepted truth. However, the exponential growth in the amount of available biological literature (Cohen and Hunter, 2004) and the ever-increasing fragmentation of the life sciences into more and more specialized disciplines do not bode well for the ability of researchers to find and keep track of all the information relevant to their fields. The MEDLINE database [1] has added between 2,000 and 4,000 bibliographic records *per working day* since 2005, [2] over three quarters of which have abstracts, and now comprises over 15 million entries. However, this is only one source of textual biological knowledge, along with full-text electronic journals and textbooks, patent filings, internal lab and project reports, comments fields in biomolecular databases, and topic-specific curated repositories such as The Interactive Fly[3] (Brody, 1999). The development of natural language processing (NLP) techniques tailored to the biological domain has been driven by the resulting problems of literature overload that face scientists today.

It is easy to conceive of situations where a biologist's workload might be lessened by suitably-targeted literature analysis tools. One scenario involves the construction of a descriptive picture of the causal relationships between a set of genes that are differentially expressed in a disease state, as revealed by microarray experiments for example. While it is often possible to make some headway by clustering these genes according to their functional annotations, a literature trawl will be necessary in all but the simplest cases in order to identify the specific regulatory pathways involved (Shatkay *et al.*, 2000). Another task where the application of text mining techniques has been seen as a great potential boon is the curation of model organism databases. Curators at these projects spend many hundreds of biologist-hours each week scanning the newly-

[1] `http://www.pubmed.org/`

[2] `http://www.nlm.nih.gov/pubs/factsheets/medline.html`

[3] `http://flybase.bio.indiana.edu/allied-data/lk/interactive-fly/aimain/1aahome.htm`

published literature for discoveries about genes or proteins in the organism of interest, in order to enter the results into publically-searchable databases. Much research has been focused on developing NLP approaches to finding the appropriate parts of the appropriate articles and extracting candidate annotations to present to the curators (Hersh and Bhupatiraju, 2003; Hirschman *et al.*, 2005). More generally, the problem of integrating structured databases with unstructured textual resources, through terminological indexing or document classification for example, is an ever-present challenge in industrial-scale biology.

By analogy with data warehousing and data mining in conventional database technology, the terms 'document warehousing' and 'text mining' have come to refer to (respectively) the management and analysis of large collections of unstructured text, usually using NLP techniques, with help from statistics, and from other branches of information science (Sullivan, 2001). Some authors restrict the definition of text mining specifically to the deduction or inference of *novel* facts from multiple documents, or the process of uncovering latent trends or 'nuggets' of previously-unknown information in the data (Hearst, 1999), but it is frequently used in a broader sense than this to refer to any sort of exploratory, explanatory or evidence-gathering analyses of large corpora or document collections. A good rule of thumb is that text mining is *what* someone does—or eventually, what a computer could do unaided—and NLP is *how* their software accomplishes it. The terms 'language engineering' and '[human] language technology' are also sometimes used in this context; the first refers to the process of building an NLP-based system, and the second describes the system itself, more or less.

The term *computational linguistics*, meanwhile, refers to the long-established interdisciplinary field at the intersection of linguistics, phonetics, computer science, cognitive science, artificial intelligence and formal logic, which again is frequently assisted by statistical techniques (Jurafsky and Martin, 2000). While it might be tempting to believe that NLP is just applied computational linguistics, or computational linguistics is theoretical NLP, this is an over-simplification. There is NLP, and indeed text mining, that does not particularly rely on (or contribute to) linguistic theory or practice, beyond the use of superficial features like word frequencies (e.g. Jenssen *et al.*, 2001) or categories (e.g. Hakenberg *et al.*, 2005). The distinction is somewhat similar to the difference between computational biology and bioinformatics. A bioinformatics specialist might be writing a data visualization plug-in for R, adapting a sequence alignment algorithm to run in parallel on a grid, or indeed tagging a few thousand MEDLINE abstracts with an NLP tool, without being said to be "doing computational biology." Conversely, pretty much everything a computational biologist does will involve some bioinformatics at some point, but the results of their work may well be of more interest to the biological research community.

However, the overall aim of this work is to further the state of the art of biological

text mining by applying linguistically-motivated computational techniques in a novel manner. As a result, most of the NLP tools, methods and concepts under discussion will have some degree of linguistic theory behind them, which will be explained to the level of detail appropriate to the discussion. Unlike a computational linguistics thesis, however, the ultimate aim is not to understand language better or to bring computers closer to understanding language. Rather, the intent is to help unlock a valuable source of scientific knowledge—the written word—so that it can be put to use in the context of biological research.

## 1.1 Outline of this thesis

The particular problem area chosen for this project is that of biomolecular interaction extraction (see Section 1.3 and Section 1.10), a popular task with readily-available test data. The solutions developed encompass both constituent-based (see Section 1.9.1) and dependency-based (see Section 1.9.2) models of language, and illustrate several key concepts in computational linguistics. Chapter 2 and Chapter 3 describe preliminary experiments performed in order to select the right NLP tools for the job from the many which are freely available. Chapter 4 presents a complete linguistic processing pipeline including three new algorithms to infer biomolecular relationships from text, and describes a successful test of their performance against previously-published results. Chapter 5 summarizes the experiments in relation to current and future research, highlights some important lessons learnt from the project and discusses some theoretical and practical issues involved in developing text mining applications. Appendix A examines the related problems of named-entity recognition and identification, and describes a prototype solution. For the benefit of non-linguists, Appendix B provides a glossary of the most important linguistic terms used in this thesis, and Appendix C gives definitions of the word, phrase and dependency category labels used by the formalisms discussed. Finally, Appendix D provides the specifications for MPL, a pattern description language designed for the information extraction task in Chapter 4.

All of the source code written for this project is available from my website at `http://biotext.org.uk/` along with any errata, updates or additional data that might appear.

### 1.1.1 Outline of this chapter

The next section of this chapter contains an overview of the fields of computational linguistics and NLP, and Section 1.3 discusses some of the goals of biological text mining pursued by current and previous researchers. Section 1.4 describes some of the broad strategies or frameworks for NLP algorithm development with reference to their particular advantages and disadvantages in the biological domain. The six sec-

tions which follow it each discuss a particular NLP task (or task family) with relevance to bioinformatics, including the goals of the task and its potential applications, and the computational methods that have been employed in tackling it. Examples are taken from both biomedical research and the general English domain where possible, comparing and contrasting the principles followed, the problems encountered and the degree of success achieved. As each of these sections deals with progressively more abstract and less well-explored subject matter, particularly in relation to the biomedical domain, there is a gradual shift in focus from the more practical to the more theoretical aspects of NLP over the course of the discussion. Finally, Section 1.11 discusses the motivation for and objectives of the experiments which follow.

## 1.2 Topics in linguistic computing

NLP and computational linguistics are broad and diverse fields, and the current strands of research include (but are not limited to):[4]

- syntactic analysis (Kaplan *et al.*, 2004);
- semantic interpretation (Pradhan *et al.*, 2004);
- language generation (Chambers and Allen, 2004);
- speech processing (Levow, 2004);
- automatic translation (Och *et al.*, 2004);
- prediction of language difficulty (Collins-Thompson and Callan, 2004);
- quantification of emotional content (Forbes-Riley and Litman, 2004);
- discourse analysis (Polanyi *et al.*, 2004);
- and dialogue tracking (Jain *et al.*, 2004).

It should be immediately obvious that there is much within the field of NLP that is not and will likely never be relevant to bioinformatics, or at least not within any realistic timeframe. The subfields of speech and gesture processing can be ignored wholesale, as can the large bodies of work dealing with multi-agent dialogue systems, emotional tone, colloquial and ungrammatical utterances, and so on. Machine translation and other multi-lingual operations may be of interest to non-Anglophone research groups, but as long as English is the *lingua franca* of biology this will remain a minority concern. As time progresses, more sophisticated processes may become relevant to biomedical NLP research, perhaps including auto-summarization and paraphrasing,

[4] Citations in this chapter's topic lists are illustrative examples rather than broad surveys.

question answering, and educational aids like conversational tutoring and essay evaluation, all of which are current topics of research in less technical textual domains. However, all of these depend on the fundamental linguistic concepts that will be discussed here, and which are largely embodied by the first two areas listed above: syntax, referring to the grammatical structure of text, and semantics, referring to its meaning.

One large and important theme within NLP research is information retrieval (IR), the science that underlies search engines such as Google[5] and PubMed[6] as well as more specialized applications (van Rijsbergen, 1979). IR is classically concerned with the indexing of text and the selection of relevant documents or parts of documents in response to user queries, which may be in the form of keywords, seed documents or classification categories. Traditionally IR is largely a mathematical/statistical endeavour, supported by the computer science required to manage what are often very large datasets, and I suspect many practitioners would see it as something different but related to NLP. Nonetheless, the two fields are often discussed side by side, there is a healthy cross-fertilization of ideas, and many of the fundamental concepts of NLP had their origin in IR such as the precision[7] and recall[8] measures.

Hersh (2005) paints an appealing picture of an idealized knowledge acquisition process, which distills *all literature* into *possibly relevant literature*, then into *definitely relevant literature* and finally into *structured knowledge*. The first two stages are taken care of by a process labeled "information retrieval" and the last two are covered by "information extraction, text mining" with some overlap in the middle. IR is a vast field with decades of published research, and will not be treated in detail in this thesis; information extraction, on the other hand, is one of the central themes of this thesis and will be discussed at length below.

In order to understand NLP, some background in linguistic concepts is required, which will be introduced piecemeal as necessary to ground discussion of each new theme. Linguistics is not without its controversies and opposing schools of thought, but to a large extent such matters can be ignored when one's concern is to provide pragmatic and computationally-tractable models within which empirical progress can be made and ultimately solutions can be developed. Volumes have been written in traditional linguistics about the differences between Government and Binding theory (GB), Lexical Function Grammar (LFG) and Generalized Phrase-Structure Grammar (GPSG) (Horrocks, 1997), for example, and even in the somewhat more pragmatic world of computational linguistics there is a degree of competition between these and other theoretical frameworks such as Head-driven Phrase Structure Grammar (HPSG), Lexicalized Tree-Adjoining Grammar (LTAG) and Combinatory Categorial Grammar (CCG) (Clark and Curran, 2007; Munroe, 2006). However, much ground has been

[5] `http://www.google.com/`
[6] `http://www.pubmed.org/`
[7] The fraction of predictions which are correct/relevant.
[8] The fraction of correct/relevant answers which are predicted.

gained with computational approaches that avoid too close commitment to one or another of these competing theories. These tend to focus on those things that the theories have in common, such as the hierarchical phrase-structure model for sentences, rather than on the differences between them (see Section 1.9.1). This degree of theory-neutrality enables us to avoid getting bogged down too deeply in linguistic formalisms. A similar shortcut is provided by the fact that some NLP methodologies, such as Link Grammar (Sleator and Temperley, 1993), explicitly or implicitly employ instrumental models of how we *can* assign meaningful metadata to sentences or words computationally, rather than theories about how humans actually *do* generate and understand language (see Section 1.9.2).

## 1.3 Goals of biological text mining

It is useful to break the umbrella term 'text mining' into a few subdisciplines with distinct but aligned goals. It must be remembered however that the actual NLP tasks and processes which will be presented later are to a large extent shared between these research strands. The main desired endpoints of biological text mining include:

- Document search and retrieval applications tailored to the needs of life scientists, which utilize the available linguistic information and biological data resources in a 'smart' way in order to reduce ambiguity and increase relevance. This is the domain of information retrieval as mentioned above. (Hersh and Bhupatiraju, 2003)
- Information extraction (IE) systems which can analyse unstructured text and pick out relevant facts to be stored in a database and queried. The nature of these facts depends on the particular problem or application, but in bioinformatics they will typically be related to biomolecular events involving genes and proteins. This thesis is concerned primarily with information extraction. (Nédellec, 2005)
- Better integration of textual data sources and non-textual databases, so that everything that is known about (for example) a gene of interest can be accessed and cross-referenced in a uniform manner, despite ambiguity and variation in nomenclature. This is of particular importance in commercial organizations where proprietary data must be integrated with data imported from the public domain. (Smith and Cleary, 2003)

In addition to these primary goals, there are several other areas of text mining research which may lead to useful applications in bioinformatics in the medium term, and have been under development for some years in a medical informatics context, but which are not yet undergoing intensive study by the biological text mining community. These include:

- Knowledge discovery methodologies by which deeply-buried trends and patterns, or new hypotheses, conclusions or inferences can be arrived at by the analysis of multiple textual sources. Such systems must of course rely on IE or IR techniques (or both) in order to perform the data-gathering prior to analysis. (Weeber *et al.*, 2003)

- Automatic summarization methods for 'digesting' a document—or a set of documents on the same topic—into a shorter synopsis. This involves both the analysis and synthesis of language. (Elhadad, 2006)

- Question answering systems, where documents, parts of documents or individual facts are retrieved in answer to queries which are themselves entered by the user as written questions. These are the text mining equivalent of natural language query interfaces for conventional databases. (Niu *et al.*, 2003)

- Ontology induction processes, where a classification schema for entities of interest, including relationships between the classes in the schema (*is-a*, *part-of* etc.), is constructed and populated automatically. This can be thought of as a specific kind of IE. (Kawasaki *et al.*, 2003)

Biological NLP research began in earnest in the early 1990s (Futrelle *et al.*, 1991; Baclawski *et al.*, 1993) although some of its foundations were laid as far back as 1982 (Futrelle and Smith, 1982), and it draws on related work from the field of medical informatics which stretches back at least four decades (Baruch, 1965). The linguistic analysis of biological text, independent from (but influential on) its computational processing, likewise has a long history (Harris, 2002; Gopnik, 1972). IR (Hersh and Greenes, 1990) and knowledge discovery (Swanson, 1986) in biomedicine were originally somewhat distinct from NLP as few linguistic techniques were used in these endeavours, but in recent years the fields have come together such that it is common to see ideas from all of these originally distinct disciplines discussed at the same events and even in the same papers.

## 1.4 NLP strategies for bioinformatics

Biology and medicine—the two domains are often conflated (Pustejovsky *et al.*, 2002) although they have been shown to have qualitatively (Friedman *et al.*, 2002) and quantitatively (Bodenreider and Pakhomov, 2003) different linguistic properties—pose interesting problems in NLP. There is a clear demand for NLP research and development in the life sciences; companies such as AstraZeneca (Hayes, 2004) and Novartis (Vachon, 2004) are investing increasingly in text mining projects and products, and the number of MEDLINE entries concerning genomic NLP seems to have grown exponentially from 1999 to the end of 2005 (although the rate appears to be slowing

slightly—see Figure 1.1). However, the current state of the art in mainstream English NLP has been facilitated by the availability of large, hand-annotated textual resources, such as the million-word Penn Treebank (PTB)[9] collection of *Wall Street Journal* articles and their associated phrase-structure trees (Marcus *et al.*, 1994), against which machine-learning algorithms can be trained. These are currently small and scarce in the biomedical genre(s), due to a skills bottleneck: a postgraduate linguist working alone can accurately annotate newspaper English with word categories, phrase structures, named entities and semantic relations, but to perform the same tasks on a biological corpus requires the input of a trained biologist too (Tateisi *et al.*, 2004). This is due to the high proportion of unfamiliar technical terms, non-standard usages and domain-specific phrases with complex internal structure.

Faced with this stumbling block, there have been four broad classes of response from the bioinformatics NLP community, which are discussed in brief below.

### 1.4.1 Domain-specific annotated corpora development

This is the approach taken by, for example, the GENIA project[10] at the University of Tokyo (Tateisi *et al.*, 2005), the Mining the Bibliome project[11] (sometimes PennBioIE or just BioIE) at the University of Pennsylvania (Bies *et al.*, 2005), and more recently the ProSpecTome project[12] at Birkbeck College (Kabiljo *et al.*, 2007). It is labour-intensive but rigorous, and produces gold-standard benchmarks against which new and existing NLP algorithms can be trained and tested. In addition, several algorithm-research projects have produced their own development corpora, some of which have been made available to the public (Smith *et al.*, 2004; Temkin and Gilder, 2003); these are usually smaller and more closely-tailored than those which result from dedicated corpus annotation projects. However, annotation project teams tend to codify their annotation protocols and rules more explicitly (Tateisi and Tsujii, 2004; Kulick *et al.*, 2004), which is useful for third-party investigators.

### 1.4.2 Untrained (non-corpus-based) methods

Not all NLP algorithms are based on machine-learning principles. Many rely on hand-crafted heuristics or grammars (Hull, 1996; Sleator and Temperley, 1993) which reflect assumptions about language that may or may not remain true when applied to a new domain such as biology. Given that such tools are only tied to any particular genre in a somewhat subjective way, it is not unreasonable to apply them to biomedical corpora without major alteration (Yakushiji *et al.*, 2001). Alternatively, they may be designed

[9] `http://www.cis.upenn.edu/~treebank/home.html`
[10] `http://www-tsujii.is.s.u-tokyo.ac.jp/~genia/`
[11] `http://bioie.ldc.upenn.edu/`
[12] `http://textmining.cryst.bbk.ac.uk/ProSpecTome/`

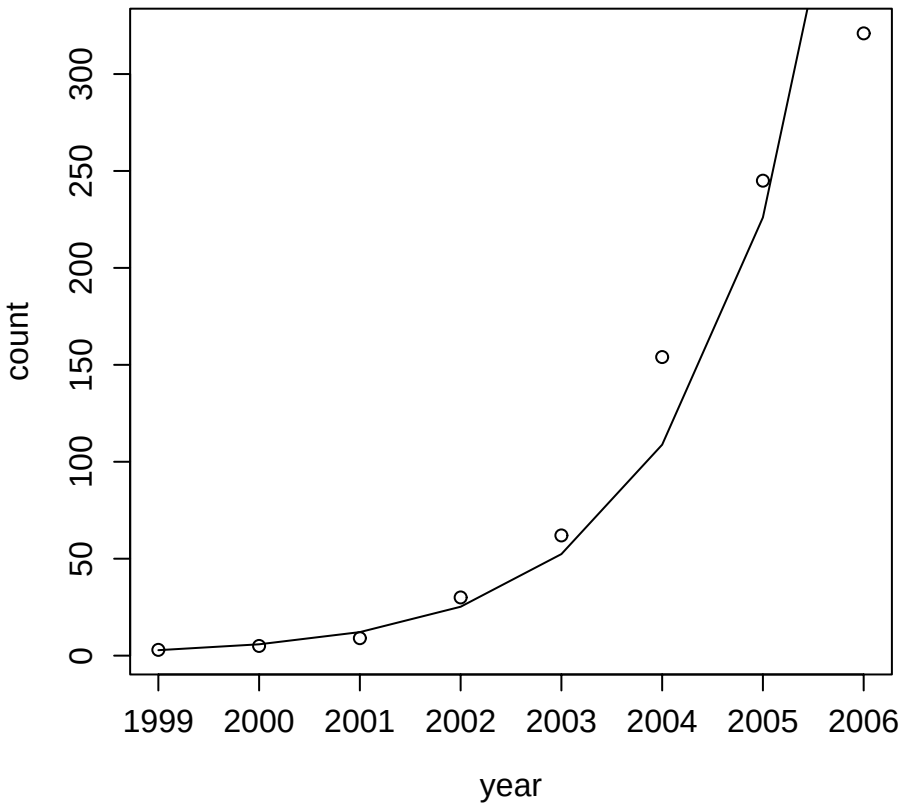


Figure 1.1: Number of MEDLINE entries matching strings *natural language processing* or *text mining*, as well as *gene* or *protein*, compared to an exponential curve (after Verspoor *et al.*, 2006). Although the growth rate in MEDLINE looks to be slowing, there are many other refereed publications that are not indexed by MEDLINE (such as NLP and informatics journals and conference proceedings) which frequently publish material on topics relating to biomedical text mining.

or built with biomedical applications in mind, according to more (Friedman *et al.*, 2001) or less (Blaschke *et al.*, 1999) linguistically-oriented principles. However, formal quantitative evaluation can prove difficult in such cases. While machine learning systems can automatically obtain test data by holding out a portion of the training data, deterministic methods which lack any manually-annotated development data at all are sometimes published with coverage figures but no actual quantification of accuracy (Svolovits, 2003), or with a small amount of manually-marked output functioning as worked examples (Rzhetsky *et al.*, 2004).

### 1.4.3 Utilization of tools trained on conventional English

Some of the most successful algorithms in general-English NLP are statistical models which have been trained on the large corpora of annotated text mentioned above (Collins, 1999; Charniak, 1999). Although the development of domain-specific corpora has not yet progressed far enough to allow such models to be retrained from scratch on biological English, it is nevertheless important to establish how well or badly these methods perform on such texts. We may be able to correct for the specific mistakes they make on unfamiliar kinds of writing, by using additional sources of background knowledge about biology (Rindflesch *et al.*, 2000), or by treating biomedical texts as if they are written in a formal sublanguage which is analytically distinct from colloquial English (Harris, 2002). Indeed the performance dropoff of certain tools or algorithms may be small enough that it can be safely disregarded for practical purposes. Without rigorous evaluation on a case-by-case basis, there is no way to tell.

### 1.4.4 Unsupervised machine learning methods

A specialized area of computational linguistics that has yet to be widely explored in a biomedical context relates to the induction of stochastic models from unannotated corpora. This approach is more feasible in some areas (particularly automatic grammar acquisition, and probabilistic modeling for speech processing) than others (Jurafsky and Martin, 2000). So far its success has proved to be limited, as it is highly sensitive to both statistical and linguistic subtleties, and can lead to rules which appear statistically sound yet have no linguistic basis (de Marcken, 1995). For example, prepositions follow nouns very frequently in English not because noun-preposition is some meaningful linguistic unit, but because prepositional phrases (which start with prepositions) often follow the noun phrases they modify (which end with nouns). We can think of any number of phrases like *the man on the bridge*, *the man in the car* and so on, all of which contain the noun *man* followed by a preposition, but constructions like *man in* are not constituents (see Section 1.9.1) in their own right. Rather, the prepositional phrase starting with the preposition is the grammatical unit which can be substituted and moved around atomically, as in *the man in the car on the bridge*.

Given the large amount of biomedical text available and the labour-intensive nature of the annotation process, however, developments in unsupervised techniques that overcome these problems may prove important in a bioinformatics context.

## 1.5 Tokenization

Perhaps the most basic task in the processing of written language is the splitting of a string of text into a list of words and other symbols. This may sound straightforward at first, at least for Western languages with visible word separators, but can lead to

the introduction of serious errors that propagate throughout the (conceptual or actual) downstream processing pipeline if performed naïvely. The potential complexities in the tokenization process fall into two major categories: those that apply across all domains, and those that are more likely to be found in biomedical corpora, as well as in certain other scientific domains with a large technical vocabulary.

### 1.5.1 Domain-independent tokenization issues

Most of these problems arise from the use of punctuation to conjoin words, and therefore are language-specific, but sometimes dependent on an individual author's or editor's style. Common examples in English include the hyphenation of adverb-participle (*tightly-wound*) or noun-adjective (*language-specific*) pairs, and the coordination of words in the same category using slashes or hyphens (*his/her*, *adverb-participle*). Perhaps most human readers would intuitively see these constructions as consisting of two word tokens joined by a punctuation token, but dictionaries contain many superficially-similar examples that are nonetheless listed as a single lexical entry. This seems to reflect frequent usage, and can occur whether or not the string has idiomatic meaning in addition to, or instead of, its literal interpretation—as in *newly-wed* vs. *side-lined* (American Heritage Dictionaries, 2000). On the other hand, one might suppose that humans would typically see verb contractions and genitives (*I'm*, *would've* or *Andrew's*) as single words; word count routines such as the GNU `wc` command tend to treat them as such, at least. The influential PTB tokenization algorithm[13] however splits such cases into two tokens—and yet keeps *all* hyphenated constructions together. We must then treat such decisions as (at least to a certain degree) matters of convention; as a result, what matters most in these cases is consistency of actual and expected behaviour between NLP components that are working together, rather than 'correctness' *per se*. Another class of punctuation issues can be identified as genuine errors rather than differences of convention: those that occur when a hyphen is used to break a word at the end of a line in a formatted file (e.g. a PDF) and is not removed correctly on conversion to plain text for processing (see Section 5.1.3).

### 1.5.2 Biomedical tokenization issues

The use of chemical and biological names with embedded punctuation is a particular source of ambiguity, although in these cases it is clearer that certain tokenizations are correct or incorrect rather than simply matters of convention. Naïve tokenization schemes designed to cope well with newspaper English will not necessarily perform well on such data. For example, the assumption that an alphanumeric string ending with a single quote is two tokens, the first a word and the second a punctuation symbol

[13] `http://www.cis.upenn.edu/~treebank/tokenizer.sed`

indicating the end of a quotation, will cause references to the ends of DNA (*5'* and *3'*) to be misinterpreted. Similarly, a decision to split on slashes and hyphens and consider each resulting substring as a separate token will break the names or symbols of many genes, proteins and chemical compounds (*Porcupine-A*, *TbNT2/927*, *N(2),N(2)-dimethylguanosine*). Conversely however, treating hyphens as non-splitting characters may obscure the specific biological meaning of certain words in constructions like *secretase-mediated* and *sigma(F)-sigma(E)-sigma(G)-sigma(K)*. Clearly there is a complex relationship between these issues and those involved in named entity recognition (see Section 1.8).

### 1.5.3 Tokenization methods

Despite these complexities, this task is often approached in a fairly ad-hoc way, using simple pattern-matching with hand-crafted rules for common exceptions, and rarely evaluated rigorously as a task in its own right. Also, many higher-level NLP tools perform their own tokenization without making explicit the principles and assumptions that have been employed. It is not uncommon in bioinformatics to simply apply the Penn rules without modification (Smith *et al.*, 2004) or with post-processing to adjust for certain biological and chemical constructions (Tateisi and Tsujii, 2004). An interesting alternative perspective is presented by Futrelle *et al.* (2003), whose principle of 'Extreme Tokenization' states that any runs of alphanumeric characters and any other single symbols should be broken off into new tokens, and that the tokenizer should not be responsible for making complex decisions that might run contrary to the requirements of a module further down the pipeline. However the implicit corollary of this is that later modules must be designed with this in mind, since many English words and biological terms (*it's*, *NF-kappa-B*) will be split into multiple tokens by this method. Experimentation will tell whether this is a sensible way to deal with tokenization ambiguity.

## 1.6 Sentence boundary detection

Like tokenization, sentence splitting appears simple at first glance but reveals hidden complexities on further examination. Again, mistakes made here can cause error cascades that may not be easily detectable at a later stage of processing. The demarcation of text into sentences is a required step for further syntactic processing and for almost all forms of IE, although the reasons for doing so may differ. It has been shown, for example, that when simply looking for co-occurrences of entity names in order to predict putative connections from the text, operating at the level of individual sentences gives the best trade-off between precision and recall scores (Ding *et al.*, 2002). At the other end of the scale, analysis of text using a sophisticated grammatical parser will

be difficult if not totally impossible if the boundaries between the sentences are not accurately determined, as the parser will be hampered in its attempts to fit syntactic tree or graph structures to the sequences of words. Indeed, most parsers bypass the thorny problem of boundary detection by requiring text that has already been split into sentences (Collins, 1999; Charniak, 1999, for example). As with tokenization, general English sentence boundary detection issues are distinguishable from those specific to biomedical English.

### 1.6.1 Domain-independent sentence demarcation issues

The intuitive English formula—look for a full stop (or exclamation/question mark) followed by whitespace and then a capital letter—must be adjusted to cope with abbreviations such as *Dr.* or *etc.*. Initials are another source of problems—the string *Mr. A. B. Clegg* for example contains three points where a false boundary might be inserted. Further confusion arises when a string like *etc.* is found at the end of the sentence, and the full stop that indicates abbreviation is conflated with the end-of-sentence marker. This means that a more sophisticated approach must be taken than simply flagging *etc.* as a non-boundary string (Walker *et al.*, 2001).

### 1.6.2 Biomedical sentence demarcation issues

The problems described above boil down to the occurrence of false-positive or false-negative ends of sentences. However, biological text provides a whole different source of error—false-negative beginnings of sentences. These can occur when for example a gene name or symbol, spelt with a lowercase initial (e.g. *sonic hedgehog* or *shh*), is found at the beginning of a sentence.

### 1.6.3 Boundary detection methods

Despite the importance of the sentence as a unit of meaningful information, sentence boundary detection in bioinformatics is still in a fairly primitive state. While mainstream computational linguists have developed sophisticated classifiers based on single (Reynar and Ratnaparkhi, 1997) or multiple (Hillard *et al.*, 2004) machine-learning principles, particularly in the speech-recognition field where punctuation and orthography are not available as clues, biomedical NLP research has largely relied on deterministic, heuristic preprocessing filters using techniques such as regular expressions (Smith *et al.*, 2004). Simple heuristic models have achieved accuracy as high as 99% in informal tests (Futrelle *et al.*, 2003), so it is not clear whether developing highly sophisticated solutions would be a good direction of research effort, although a study of a clinical IR system found that sentence boundary mistakes were responsible for 6% of errors in a 1,377-document corpus (Averbuch *et al.*, 2004). A simple test of current

systems' accuracy against the full 2,000-abstract GENIA corpus would help determine whether much additional work is necessary.

## 1.7 Part-of-speech tagging

Part-of-speech (POS) or word category tagging involves assigning each word in the tokenised input stream to one of a set of pre-defined categories such as 'adjective', 'common noun, plural' or 'preposition or subordinating conjunction'. The set of tags used (and thus the set of categories available) is referred to as a tagset. There are various tagsets in common use, with the *de facto* standard perhaps being that defined by the PTB (Santorini, 1990). The fact that the various tagsets available do not exactly map onto one another indicates subtle differences in their underlying conceptions of English; for example, the MedPost tagset (Smith *et al.*, 2004) distinguishes between normal lexical verbs and the special verbs *be*, *do* and *have*, whereas the PTB tagset considers all of these to be just verbs.

POS tagging is rarely an end in itself, unless one is performing linguistic corpus analysis for its own sake, but it is a pre-requisite for accurate syntactic processing. Although some parsers (see Section 1.9) perform POS tagging (and indeed tokenization) themselves (Charniak, 1999), others require pre-tagged text (Collins, 1999), and a third kind only consider word categories to be emergent clusters of similarly-behaved words (Sleator and Temperley, 1993); in any case the two tasks are conceptually distinct so will be treated individually here. Other tasks that can make use of POS information include named entity recognition (see Section 1.8) and word-sense disambiguation for information retrieval (Cutting *et al.*, 1992).

### 1.7.1 Problems in part-of-speech assignment

Several sources of information must be drawn on in order to assign a POS tag with any degree of certainty, and in many cases there will still be ambiguous results that must be guessed at from the available context. Some words can be assigned to a syntactic category by matching against a dictionary, but many English words can belong to multiple categories (*flies*, *like*—see Section 1.9) meaning contextual information (e.g. the previous and next words) must be brought into play. However, there will always be a fraction of cases that even human annotators will disagree on or be unable to make a judgement on; the sentence *the ambassador was entertaining last night* for example could mean that said dignitary was personally amusing (*entertaining* = adjective) or had guests (*entertaining* = verb). Humans can only judge such cases intuitively based on extra-sentential semantic information, and not at all given just the sentence in isolation; this puts an upper bound on the performance of machines at the same task. Inter-annotator agreement can be measured using the kappa statistic (Carletta, 1996)

which quantifies this uncertainty. Unknown words present another source of error, particularly in bioinformatics and other technical domains, although many words contain morphological clues that can help identify their roles—the suffix *-tion* being found almost exclusively on nouns for example.

### 1.7.2 Part-of-speech tagging methods

Traditionally, mainstream English POS taggers fall into two camps, those that use deterministic processes based on morphological analysis and lexical lookup along with hand-crafted disambiguation rules (Voutilainen, 1995), and those that use stochastic models trained on annotated text (Ratnaparkhi, 1996). A hybrid error-driven approach learns explicit classification rules from text (Brill, 1995). Recently, highly-optimized taggers trained on hand-annotated MEDLINE abstracts have begun to appear, using hidden Markov models (HMMs) (Smith *et al.*, 2004) or maximum entropy methods. [14] These can expect accuracies of around 97-98% on biomedical text.

## 1.8 Named entity recognition

Named entity recognition involves the tagging of biomedically important names in text, such as those for genes, proteins, RNAs, small molecules, tissue or cell types, diseases, organisms and so on. This problem has been particularly well-studied in the past few years, since it is hard to see how one could extract any sort of useful meaning from this kind of text without accurately determining what objects or substances are being described. The equivalent entities in news articles are things like companies, places, people and products—anything that would be represented by a proper noun, more or less—but there is an added dimension to the problem in bioinformatics that relates to interoperability and system integration. It would be acceptable in a newspaper NLP system to identify *IBM* and *Lotus* as named entities, for example, and thus flag the event *IBM bought Lotus* as a newsworthy factoid, allowing this nugget of information to be reported directly as-is or used in summary generation or other further processing. However it is less useful to recognize *Amias* and *ACE* as named entities in the relationship *Amias inhibits ACE*, without also categorizing *Amias* as a drug and *ACE* as a protein, and relating both of those names—along with the various other names by which those substances can be identified (e.g. *candesartan cilexetil*, *angiotensin-converting enzyme*)—to entries in a database, an ontology or other structured data repository. In short, named entity recognition in bioinformatics involves detection, disambiguation and classification, although not all research projects tackle all three steps.

A naïve approach to this task would be to perform simple string matching between the text and a canonical database of entities, but there are various reasons why such a

[14] `http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/postagger/`

technique will perform very poorly, as discussed below. Nonetheless, many methods require a dictionary of some kind from which rules can be generalized. Terminological resources such as the Gene Ontology (GO) (Ashburner *et al.*, 2000) and the Unified Medical Language System (UMLS) (McCray *et al.*, 2001), as well as other structured databases such as LocusLink (Pruitt *et al.*, 2000), have proved to be useful sources of raw lexical data for this purpose. These resources encode a great deal of non-lexical background facts, such as meronymy (part-whole) and hyponymy (set-superset) relations, which gives them an enormous amount of potentially useful semantic richness (see Section 1.10); however, for the purposes of named entity processing they are much more immediately useful as broad-coverage dictionaries or thesauri of names and synonyms.

### 1.8.1 Named entity recognition issues

The sheer amount of variation in the naming of biological entities poses immediate problems, over and above the simple fact that public databases of such entities are populated with data drawn from published accounts, and thus will always lag behind the literature. It is not uncommon for one entity such as a gene or its product to be referred to by many different names and symbols in the literature and in public databases (*FLICE*, *MACH* and *Mch5* are all the same protein), and although there are nomenclature guidelines in place, the extent to which these are followed varies with organism and date of publication (White *et al.*, 2001). Even if near-exhaustive synonym lists can be compiled from structured data sources, further variation within each name can occur due to differences in orthographic conventions between different texts or databases. These include the representation of special characters ($\alpha$*-Spec*, *alpha-Spec*, *&agr;-Spec*[15]), hyphenation and abbreviation (*NF-kappaB*, *NF-kappa-B*, *NFkB*), capitalization standards and international spelling differences.

Polysemy and homonymy, where the same term has several different related and unrelated meanings respectively, are also widespread issues in biology. Polysemy occurs when a name is used to refer to both a gene and its product (*Cdk2*), or to the members of a homologous family across several species (*Integrase*), and also when an enzyme is named after its biochemical activity (*alcohol dehydrogenase*). Homonymy is less systematic; amongst other things it arises from acronym clashes with other biological entities and with general scientific or mathematical terms, and imaginative naming schemes leading to genes being given English words as names (*hand*, *shark*, *for*). Just as orthographic variation can amplify synonymy, typographic normalization—taking place when a formatted journal article is converted to plain ASCII text—can amplify polysemy and homonymy. This occurs when the removal of formatting information destroys the distinction between italic type for genes and standard type for their pro-

[15] *&agr;* is the encoding for $\alpha$ used in MEDLINE.

teins, as followed by some journals, and renders superscripts, subscripts and normal text indistinguishable.

### 1.8.2 Methods in named entity recognition

Many early attempts to improve performance over and above the levels of naïve string matching concentrated on loosening the match criteria in order to improve recall, while applying a variety of *ad hoc* methods in order to mitigate the associated loss of precision. Jenssen *et al.* (2001) for example relaxed case sensitivity in some cases, and allowed for likely family variants, where a name like *cyclin E2* would be tagged since it consisted of a known family identifier followed by a novel variant specifier. Short gene symbols proved to be a particular source of false positives, so disambiguation procedures based on context words extracted from the corresponding full-length gene names were introduced in these cases. One ingenious method of match relaxation was proposed by Krauthammer *et al.* (2000), who adapted the algorithm from BLAST (Altschul *et al.*, 1997) to tag strings sufficiently similar to entries in the dictionary. Some investigators however dispensed with the dictionary entirely, such as Fukuda *et al.* (1998), who manually constructed a set of rules, rather like a grammar, that encapsulated the common surface patterns of protein names.

Nowadays there is sufficient training data available in the form of biomedical corpora annotated with named entity information (see Section 1.4.1) for machine learning solutions to become feasible and in fact *de rigeur*. The broad idea is that given enough examples and a well-chosen feature set, an inductive algorithm can be trained to classify incoming text tokens as belonging, or not belonging, to an entity name. The features taken into account often include orthography (e.g. embedded numbers or Greek letters), common prefixes and suffixes (like *-ase* for enzymes), POS tags, context words (adjacent or nearby tokens in the text stream), presence in (or similarity to entries in) a dictionary and so on. Furthermore, the same algorithm can also learn to distinguish between different classes of entity (e.g. DNA vs. protein), or a secondary classifier can be applied to the hits identified by the first. The current ubiquity of the machine learning paradigm is demonstrated by the the BioNLP/NLPBA 2004 workshop,[16] where it was adopted by all seven entries in the competitive shared task. Of these, three groups were based on support vector machines (SVMs) (Park *et al.*, 2004; Lee *et al.*, 2004; Rössler, 2004), a further two used SVMs alongside hidden Markov models (Zhou and Su, 2004) or conditional random fields (CRFs) (Song *et al.*, 2004), one entry used just an HMM (Zhao, 2004), and one used a maximum-entropy Markov model (MEMM) (Finkel *et al.*, 2004).

[16] `http://www.genisis.ch/~natlang/JNLPBA04/`

### 1.8.3 Acronyms and abbreviations

A distinct sub-problem of named entity processing is the handling of acronyms and other abbreviations, which are much more common in biomedical text (and other technical genres) than they are in mainstream English. Various approaches have been tried in order to tackle this problem, some of them as part of general named entity recognizers (Zhou, 2004), and some (such as those that follow) as projects in their own right; there is no widely-accepted 'right way' to perform this task. Yu *et al.* (2003) employ an SVM-based classifier to MEDLINE abstracts, operating under the 'One Sense Per Discourse' hypothesis which states that one abbreviation will not be used to stand for more than one different thing in the same abstract, or at least not often enough to become a significant problem. They report 84% accuracy for their system. Schwartz and Hearst (2003) on the other hand use a deterministic system based on fitting parenthetical expressions to their most likely expansions from the same body of text; they report their results in terms of precision and recall, reporting 96% and 82% respectively. Similar scores were obtained by Adar (2004), who has also built a dictionary out of the results of his cluster-based approach and made it available for manual and automatic querying over the web.

### 1.8.4 Difficult named entity problems

A serious unaddressed issue in named entity processing is the handling of nested named entities. Take for example the protein named *leukocyte elastase inhibitor*—within its name, the cell type *leukocyte* and the enzyme *elastase* are also mentioned. A similar situation occurs with *Grave's disease* in *Grave's disease carrier protein*. Names can be nested several levels deep; this can be taken to ridiculous extremes as in the *mitogen-activated protein kinase kinase kinase (MAPKKK)* family, and although it is tempting to assume that authors would avoid such a cumbersome name in favour of a simpler synonym, PubMed returns 1510 hits for the exact string *kinase kinase kinase* at the time of writing. Early corpora did not even attempt to address the engineering issues raised by nested entity names, and even now most tools and annotation formats are unable to represent such phenomena (e.g. Song *et al.*, 2005), although these are being gradually superseded by more sophisticated markup schemes that can (Kim *et al.*, 2003; Pyysalo *et al.*, 2007a). Related issues concern coordination (*CD2 and CD25 receptors*) and list (*Arp2/3*) structures which refer to multiple entities, some of which are not actually substrings of the text sequence referring to them.

There are also issues of semantics to be considered; does the name *penicillin-binding protein 3* contain the name *penicillin* even if the protein to which it refers is being discussed in a context unrelated to its affinity for said drug? Or is it analogous to the central London location *Oxford Circus* in a mainstream IE setting, which would not ordinarily be considered to contain a mention of Oxford? Another problem is

that of actually identifying entities, once candidate names have been tagged—in other words, grounding names in actual reference database objects. Very few of the systems currently available even attempt this, even though for many applications a string of text with no grounding in reality is next to useless as an entity identifier. A further discussion of this issue, along with an outline solution, can be found in Appendix A.

## 1.9 Syntax structure analysis

A piece of software designed to transform a sentence or other utterance into a structured grammatical representation is known as a parser; the process is referred to as parsing. A parser must encapsulate or be associated with a grammar, that is, a set of rules specifying which syntactic phenomena can occur in which situations and how these can combine to produce the text at hand. (It is common in linguistics to see syntactic rules as defining how well-formed strings in a language can be produced, hence 'generative grammar'.) To be of much practical use, a parser must also employ some mechanism which enables it to select one parse out of several valid alternatives, or to rank a set of possible parses in order of likelihood. This is because even very simple natural language phrases and sentences can exhibit a high degree of ambiguity. There are two broad kinds of ambiguity: lexical category ambiguity, where multiple POS tags are possible for at least one word, and structural ambiguity, where multiple grammatical readings are possible even with the same sequence of tags (Jurafsky and Martin, 2000). The distinction is neatly illustrated by a pair of Groucho Marx puns:

> *Time flies like an arrow; fruit flies like a banana.*[17]

The humour here arises from the change in category of *flies* (verb to noun) and *like* (adverb to verb) between the two phrases, which in turn forces a whole new phrase structure to emerge. Ideally, good POS tagging should weed out this kind of confusion before the parser is brought into play.

> *One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.* (Heerman, 1930)

In this case there is only one grammatical way to assign the words to categories. The humour instead comes from the multiple valid syntactic structures for the first sentence, specifically, the fact that the prepositional phrase *in my pajamas* can modify the subject (*I*), the verb (*shot*) or the object (*an elephant*).[18] Some mechanisms employed by parsers to resolve this kind of ambiguity are discussed below.

[17] This quote has been widely attributed to Groucho Marx but its exact provenance seems to have been lost.

[18] Note however that standard phrase-structure formalisms cannot distinguish between the first two cases, for English sentences where the preposition follows the object.

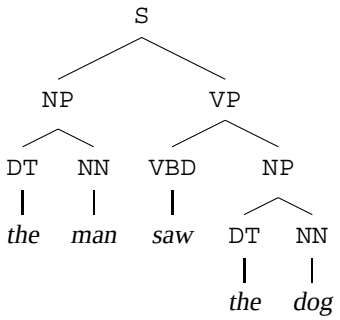


Figure 1.2: A simple constituency structure, for the sentence *the man saw the dog*. The immediate parent of each word is its POS tag, and the higher nodes are the constituent labels. The recursive nature of language is made very clear in the nested arrangement of clauses and phrases.

### 1.9.1 Constituent parsers

The modern orthodox study of syntax is an uneven battleground between various competing grammatical schools, such as GB, GPSG, LFG and so on, all of which depend to a greater or lesser degree on the work Noam Chomsky began in the 1950s (Chomsky, 1956; Horrocks, 1997). The theoretical paradigm shared by all of these competitors conceives of sentences as hierarchical structures composed of clauses, phrases and subphrases known as *constituents*. Each constituent has a label, rather like a POS tag, indicating its grammatical role: sentence, noun phrase, verb phrase, etc., although the labels used (and the underlying distinctions they represent) can vary between grammars and between parsers in the same way as the POS tagset can. An example of a simple constituent parse of the sentence *the man saw the dog* is given in Figure 1.2; such diagrams are known as *phrase structure trees*, *phrase markers*, *constituent trees* or *syntax trees*. (A dictionary of the POS tags and constituent labels in the PTB standard, as used in this figure and all the examples trees in this thesis, is supplied in Appendix C.) The same structure can also be written linearly as `(S (NP (DT` *the*`) (NN` *man*`)) (VP (VBD` *saw*`) (NP (DT` *the*`) (NN` *dog*`))))`. The word that determines the category of each constituent (e.g. the verb *saw* in the verb phrase *saw the dog*) is known as the *head* of that constituent, although this is not usually marked in the tree diagram.

The goal of a constituent parser is clear: to recover this structural information, to a greater or lesser degree of detail, for each raw or tagged sentence in the input corpus. Some parsers seek to perform this task in accordance with an explicit grammatical theory (Kaplan and Maxwell, 1996; Tsuruoka *et al.*, 2004), but the development of constituent parsers has been very much influenced by the development of syntactically-annotated corpora, particularly the PTB. This has given rise to the advent of high-

accuracy, broad-coverage parsers that are concerned less with encapsulating generative processes and ideas of grammaticality, and more with learning and applying statistical models that represent the frequency of different patterns of usage in different contexts within the training corpus. This evidential approach allows these so-called treebank parsers to disambiguate by comparing likelihoods for each plausible parse of a sentence based on similarity to training data. This kind of parser is very popular in computational linguistics research and has begun to be employed in biological NLP tasks too (see Chapter 2).

The most fundamental requirement of a constituent parser is the grammar itself, which comprises a set of production rules of the basic form `S` $\rightarrow$ `NP VP` ('sentence consists of noun-phrase followed by verb-phrase'), `NP` $\rightarrow$ `DT NN` ('noun phrase consists of determiner followed by noun') and so on. These specify all the allowable ways each type of constituent can break down into parts. Empirical treebank grammars will consist of all the production rules observed in the training data, although smoothing functions can allow unseen productions to be given non-zero probability in order to avoid breaking the parsing process. Each non-terminal, non-POS node in a parse tree represents the application of one of these rules. A modern treebank grammar will also contain statistics obtained during the training phase which encode the frequency of occurrence of each rule, conditioned on what actual words instantiate the word categories and heads of phrases on each side of the rule, along with various other features such as ancestral node labels, nearby POS tags etc. which vary from parser to parser (Jurafsky and Martin, 2000). Typically, parsing occurs in a bottom-up manner, starting with the maximum probabilities for single words and proceeding recursively; a dynamic programming matrix is thus built up, with each element holding the maximum probability of a single constituent, while a separate array of back-pointers holds the links between constituents needed to reconstruct the tree structure. One variant on this procedure holds multiple derivations in memory and so can return the $k$-best (most probable) parses for a given input (Bikel, 2004).

### 1.9.2 Dependency-style parsers

Set against the mainstream constituency view of grammar are several linguistic theories and analytical formalisms referred to collectively as dependency grammars. These set out to capture the connections between words directly, rather than by grouping them into phrases; this results in a graph representation where each node is a word and each arc a typed relationship between two words. In *the man saw the dog* for instance, *man* and *saw* would be related by a dependency reflecting the subject-verb relationship holding between them, as in Figure 1.3 and Figure 1.4—contrast this with Figure 1.2 where the path between *man* and *saw* is the joint second-longest in the tree. The principles of dependency grammar were formulated in the 1940s and 50s

by French linguist Lucien Tesnière, with their roots in European linguistic traditions going back to the Middle Ages (Schneider, 1998). There is no dominant paradigm for graph-based annotation of English text, partly because there are no projects on a comparable scale to the PTB which provides a *de facto* standard for constituency annotations. The closest dependency-like corpus in size terms is SUSANNE [19] which is an order of magnitude smaller (130,000 words compared to over a million). As a result, there are a broad range of dependency grammars which differ not only on the types and granularities of the dependencies they use, but also on the rules they use for modeling certain linguistic phenomena. This means that two 'correct' dependency parses of the same sentence from different parsers can be significantly different in topology (Pyysalo *et al.*, 2006a). Also, various other graph-based representations of syntax such as Link Grammar (Sleator and Temperley, 1993) and Word Grammar (Hudson, 1997) are often discussed alongside dependency grammars even though they have slightly different features.

As a result of this heterogeneity and lack of gold standards, dependency parsers can be rather hard to evaluate and compare (Pyysalo *et al.*, 2006a; de Marneffe *et al.*, 2006). Nonetheless, they have achieved a certain amount of popularity among practical NLP researchers, mainly due to the simplicity of dependency graphs compared to constituent trees, and the fact that the semantic structure of a sentence is often more intuitively apparent from a dependency graph than a syntax tree. Various algorithms do exist for mapping from constituent trees to dependency graphs (Schneider, 1998; Briscoe *et al.*, 2002; Pyysalo *et al.*, 2006a), albeit in potentially lossy ways; the grouping of words into phrases is abandoned, although this can sometimes be retrieved to a certain level of detail from the structure of the graph. One way around this problem is to retain the original tree structure and add dependencies to it as connections between leaf nodes (de Marneffe *et al.*, 2006). Despite their difference in output, however, some dependency parsers actually use probabilistic algorithms to learn and predict sentence structure that are very similar to those employed by classic treebank parsers (Schneider, 2003). Others such as the Link Grammar Parser are entirely rules-based. It must also be noted that many constituent parsers actually produce dependencies internally as they parse, as these relationships are an important part of their statistical models of the training data (Collins, 2003). This information can in some cases be retrieved and used (Bunescu and Mooney, 2005), but it is not as rich as a proper dependency grammar—the dependencies have no syntactic (or semantic) typology, and do not necessarily function as meaningful grammatical relations in the same way.

[19] `http://www.grsampson.net/Resources.html`

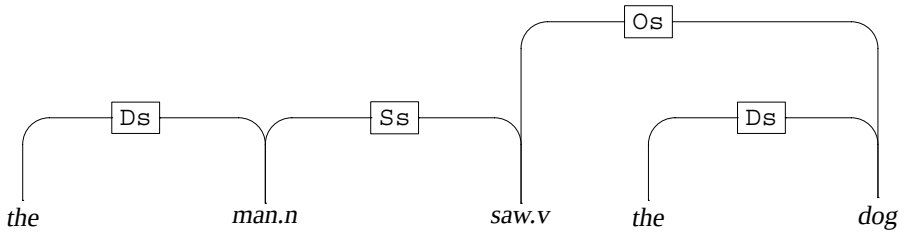


Figure 1.3: A Link Grammar linkage diagram of the sentence *the man saw the dog*. The suffixes after the nouns and verbs indicate their categories. The labels on the links indicate the link types. Links are nominally bidirectional, with no sense of subordination or government, but are constrained as to what words can go at either end, for example `S` (and subtypes like `Ss` shown here) which connects a verb to its subject. Importantly, a legal linkage must connect all the words in a sentence, in the original order, without any links crossing.

### 1.9.3 Shallow, partial, chunking parsers

Shallow or partial parsing generally refers to the process of grouping words linearly into 'chunks' based on likely phrase boundary positions in the stream of text, rather than deriving a fully-specified tree or graph reflecting all aspects of phrasal attachment. For example, a shallow parse of the sentence in Figure 1.2, ignoring POS tags, might read `(NP` *the man*`)` `(VP` *saw*`)` `(NP` *the dog*`)`; the information that *the dog* is an argument of the verb *saw* is not represented. The difference is more acute in the sentence *the man saw the dog with the binoculars* (see Figure 1.5). A full parser would make a guess about whether to attach *with the binoculars* to either *saw* or *the dog*—and ideally would correctly attach it to *saw* based on lexical co-occurrence data—while a typical shallow parser would remain uncommitted, simply bracketing it off into one or two additional chunks at the end of the sentence. In other words, *the man saw the dog with the collar* would produce exactly the same chunk pattern.

Furthermore, a common principle of shallow parsing is that the parser should produce annotations for those parts of a sentence that it can handle (hence 'partial'), ignoring any unparseable fragments, rather than rejecting the whole sentence as ungrammatical. There is a continuum between the shallowest and the deepest parsers in terms of richness of output information; some parsers can produce nested phrase annotations for a limited subset of syntactic phenomena using only surface pattern matching algorithms (Kinyon, 2001). The Link Grammar Parser can be asked to return fully-specified but unconnected graph structures for only those parts of a sentence which it considers grammatical, rather than rejecting the whole sentence based on a localized error, giving it some of the properties of a partial parser.

Shallow parsers are generally based on fast, non-recursive algorithms like finite-

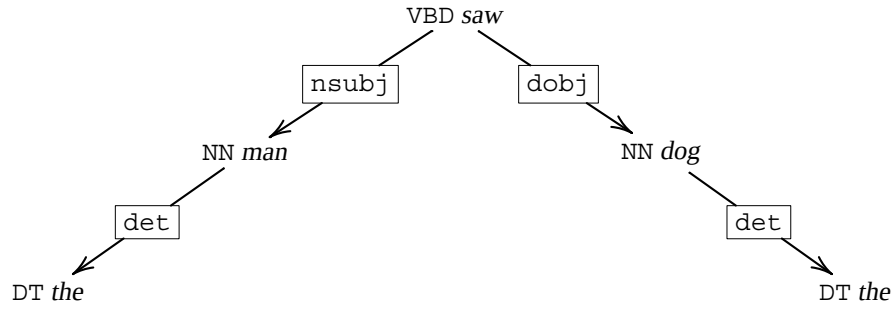


Figure 1.4: A dependency graph for the sentence *the man saw the dog* using the Stanford dependency grammar (de Marneffe *et al.*, 2006). The labels on the dependencies indicate the dependency types, and the arrows indicate directionality, from the governor (parent) to the dependent (child). Each node is a word annotated with a POS tag, although the tags are typically omitted from diagrams. The visual layout of a dependency graph (arc direction and ordering) is arbitrary, and does not reflect word order, which must be indexed separately in software. A glossary of the dependency types in the Stanford standard is supplied in Appendix C.

state machines, as used in regular expressions, sometimes with extensions allowing them to simulate a certain degree of recursion in limited cases. (A less common deep-but-partial approach (Temkin and Gilder, 2003) uses a context-free grammar designed to cover domain-specific content-bearing words and disregard the rest, for a similar overall effect.) These automata process the text linearly in reading order, applying language-specific rules that indicate where to insert chunk boundaries—for example, a determiner signals the start of a noun group in English (as in ***the*** *relatively regular word order*). The rules can be manually coded or induced from a training corpus, depending on whether a deterministic or stochastic model is in use, and can operate at the grammatical (POS tag) or lexical (word) levels (Li and Roth, 2001). In practice the most useful clues to sentence structure often come from closed-class words such as determiners (e.g. *a*, *some*), prepositions (e.g. *of*, *in*) and pronouns (e.g. *they*, *its*). The relatively regular word order in English is an advantage when processing text in this manner.

Shallow parsers are popular in natural language engineering projects such as task-oriented IE, partly due to their computational efficiency, and also because they can be easily tailored to particular tasks, and combined with each other and with different NLP techniques in a reasonably modular way. Indeed, Yakushiji *et al.* (2001) have used one to increase the coverage of a full parser and reduce the ambiguity in its output. Shallow parsers will never be able to process as wide a range of syntactic constructions as full parsers, due to the context-free and context-sensitive features of natural language (Jurafsky and Martin, 2000), and cannot represent as richly the phrasal relationships

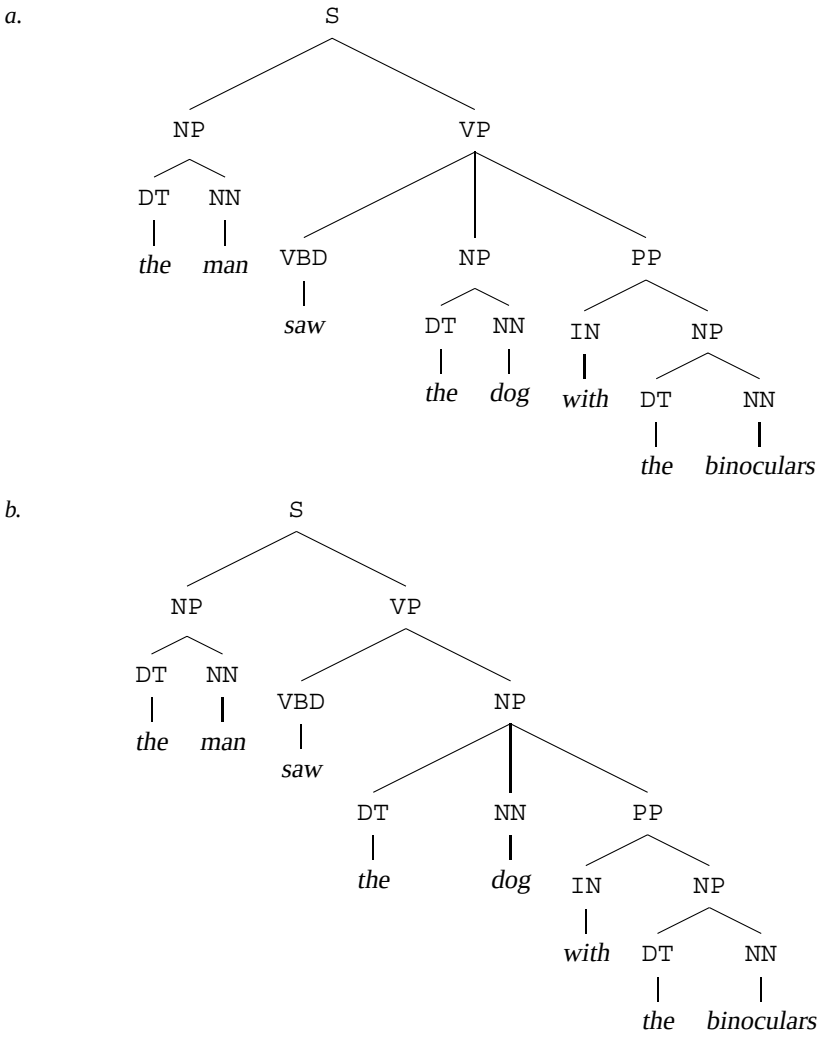


Figure 1.5: An example of ambiguous prepositional attachment—does *with the binoculars* relate to the man's act of seeing, or the dog?

holding between a sentence's constituents, as discussed above. However, advocates of shallow parsing often take the opinion that there is *sufficient* information in the output of a good shallow parser for semantic interpretation to proceed with a reasonable degree of accuracy. Leroy *et al.* (2003) even argue that within a scientifically-rigorous domain such as biology, underspecification of phenomena like prepositional phrase attachment will be less problematic than in general English, because authors construct their sentences with a view to minimizing the potential ambiguity in the message. Whether this ideal really does influence the syntactic structure of biological language though is not evident *a priori*.

### 1.9.4 Syntactic function tagging

There is one potentially useful aspect of syntax that leads naturally into semantics and yet has been relatively poorly-explored by parser researchers. The Penn and GENIA treebanks are annotated with function tags relating to certain constituents' grammatical (case) roles, phrasal subcategories and other information above and beyond simple constituent labels, some of which could assist in the correct extraction of meaning.

For example, in *the man saw the dog*, *the man* and *the dog* are both noun phrases (`NP`), but *the man* will be marked `NP-SBJ` to indicate that it is the subject of the verb of the verb phrase following it. If the sentence is rephrased in the passive voice, as *the dog was seen by the man*, *the dog* is given the `NP-SBJ` label as it is now the subject of the verb, in a grammatical sense. However, the *logical* subject of the predicate communicated by the verb is *the man*, as it is the man who is doing the seeing; therefore *the man* is labeled `NP-LGS`. However, treebank parsers ignore these function tags when training and thus cannot use them when labeling unseen text. Fortunately there are algorithms that can post-process parser output and recreate them with a good degree of accuracy (Blaheta and Charniak, 2000).

Dependency-style parsers tend to treat syntactic functions in a slightly different way, meaning that although some of these aspects of grammar are taken into account, this does not happen consistently between different formalisms. The Link parser's link labels for example pick out subject-verb (and indeed object-verb) relationships as a property of the dependency, but do not identify logical subjects in passive statements; the Stanford dependency grammar on the other hand does make this distinction. The Enju parser (Tsuruoka *et al.*, 2004) embodies a rather different view, treating the relationship between a passive verb and its logical subject as a *semantic* dependency indistinguishable from that which holds between an active verb and its subject (see Section 1.10.1).

## 1.10 Semantic interpretation

The assignment of syntactic structure to free text is not in itself a particularly rewarding task unless some meaning is then ascribed to this structure. Intuitively, part of the process of understanding language is the linking of references in text with their referents in the real world, which in NLP corresponds to the processes of named entity recognition and identification, and disambiguation (but see also Section 1.10.3). This is only the tip of the conceptual iceberg, however, although there are plenty of IE systems where the sum total of the information extracted about any given referent is that it happens to be mentioned in the vicinity of a particular set of other referents (Jenssen *et al.*, 2001; Ding *et al.*, 2002). While this can provide a useful, high-recall overview of a lot of different entities at once, it is a necessarily coarse and haphazard way to analyse text which suffers from low precision—due in part to identical treatment of positive, negative and non-committal statements—and which furthermore is not especially instructive or enlightening, from a computational, linguistic or philosophical perspective.

The essential purpose of any semantic interpretation strategy is to transform text into a formal representation, generally expressed as some form of data structure or logical notation, which can then be computationally queried, merged, summarized or otherwise manipulated according to requirements in ways that free text cannot. Even given a single *syntactic* reading of text, there are potentially several competing semantic interpretations that can arise due to such factors as multiple word senses; to take a trivial example, our likely interpretation of the statement

> *One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.*

will be different depending on whether it is delivered by Groucho holding a shotgun or a camera. Jurafsky and Martin (2000) argue that the selection of the most likely interpretation is not the job of the semantic analysis module *per se*, although this is a departure from the prevailing trend in computational linguistics for each processing module to treat ambiguity as an internally-solvable problem.

The scope of the problem of ambiguity is demonstrated by Leech (1974). In addition to the conceptual meaning of an utterance—the set of symbols, properties and relationships that it logically encodes—he identifies several different kinds of associative meaning, that is, the subtle connotations and overtones carried by the utterance that reflect such things as the beliefs or emotions of the speaker or writer, the social context within which the utterance has been made, the complex network of associations communicated by the choice of one word over another synonym or near-synonym, and so on. Consider by way of illustration the two statements *fruit flies like bananas* and *D. melanogaster are drawn to members of the genus Musa*, which have roughly the same conceptual meaning. However, the style and vocabulary of the latter sentence

lend it an air of scholarly authority, whilst avoiding the anthropomorphic connotations of aesthetic preference that the verb *like* carries in the former. In addition, the thematic meaning of an utterance refers to the information implied by word and phrase ordering and other mechanisms for changing emphasis. Fortunately, in a domain like biology, one can justifiably consider the conceptual meaning of sentences to be of exclusive interest. While subtler shades of meaning are undoubtedly present in scientific texts, particularly in the employment of rhetorical machinery and the establishment of academic credentials in the first place, these are little more than semantic sugar.

### 1.10.1 Predicate-argument structure

Without getting bogged down in the formalisms of predicate logic, the general goal of predicate-argument analysis is to transform declarative statements like *aspirin inhibits IL-4* into a logical representation like `inhibit(`*aspirin*`,` *IL-4*`)`, where `inhibit` is a known predicate with an ordered list of arguments—ordered in the sense that `inhibit(`*IL-4*`,` *aspirin*`)` would be taken to mean something different. There are various other ways to represent such relationships, such as frames (Yakushiji *et al.*, 2001), templates (Gaizauskas *et al.*, 2003) or labeled graphs (Fiszman *et al.*, 2004), but these differences are largely superficial. From a linguistic point of view, such a representation is a simplification of the sentence as it does not record, for example, the tense difference between *inhibits* and *inhibited*, although this is probably not important for most applications in bioinformatics. At first glance this process might seem like nothing more than the identification of the subject and direct object of the verb *inhibit*, but one would hope that the passive-voiced statement *IL-4 is inhibited by aspirin* would result in the same predicate-argument representation. One can imagine yet more complex ways of phrasing the statement that would (or should) nevertheless lead to the same results, such as nominalization (see below).

An additional level of complexity becomes apparent if one allows that the arguments of some predicates can in fact be other predicate-argument tuples (Friedman *et al.*, 2001; Pyysalo *et al.*, 2007a). For example, the statement *adrenaline inhibits macrophage nitric oxide production* might be best represented by the nested predicate `inhibit(`*adrenaline*`, produce(`*macrophage*`,` *nitric oxide*`))`. This example also illustrates the problem of nominalization. *Macrophage nitric oxide production* is not a verb phrase with a subject and an object, but a compound noun phrase, syntactically very different from the statement *macrophages produce nitric oxide*. Pustejovsky *et al.* (2002) point out that even some actual entity names (*Ron receptor*) implicitly encode relationships, meaning that protocols where names of entities are treated as atomic chunks (Yakushiji *et al.*, 2001) will lose a certain amount of information. Another interesting point brought up by the same authors is that partial relationships—e.g. `inhibit(`*X*`, IL-4)` where *X* is an entity whose identity has not been successfully

determined—may still be of value. The fact that an inhibitor of IL-4 exists might be of interest to the investigator and might trigger a manual inspection to discover what the missing argument should represent.

In order to facilitate the mapping from syntactic structure to semantic representation, a standard, semi-formal definition for each predicate must be available. The details of the scope and notation of these definitions (often known as frames) vary from project to project, but they would typically record such things as the arity of the predicate (the number of arguments), the role of each argument and whether each argument is optional or required—although another way to express this variation is by defining alternative usages for each predicate with different arities. Arguments can include generic roles such as `agent`, `location` or `temporal-modifier` as well as predicate-specific roles such as `entity-undergoing-mutation` (Wattarujeekrit *et al.*, 2004). These definitions are constructed in a manual or semi-automated way (Pustejovsky *et al.*, 2002), although automatic induction from statistical co-occurrence data (Hatzivassiloglou and Weng, 2002) or parser output (Yakushiji *et al.*, 2004) is a current area of research. Another interesting application of inductive learning is in the development of the actual translation rules that map between the syntactic and semantic levels. Although deterministic algorithms are the rule in bioinformatics, due to the lack of semantically-annotated training data, the availability of such corpora in mainstream English makes such methods more feasible (Gildea and Hockienmaier, 2003). It must be noted, however, that biological IE research projects often concentrate on a small number of different predicates or even just one (Pustejovsky *et al.*, 2002), or collapse many related predicates into a single class (see Chapter 4).

### 1.10.2 Alternative representations

Predicate-argument models are sometimes referred to as 'shallow' semantics, particularly in cases where each predicate's arguments are labeled with generic rather than predicate-specific roles, and when the entities that they are filled with are represented simply by phrases or names rather than more deeply-specified clusters of semantic properties. While this general approach provides a useful and perhaps necessary base level of semantic structure, it is important to be aware that there are other, deeper frameworks within which semantic processing can occur.

The simple predicate-based analysis discussed above is only capable of capturing the meaning of declarative statements in their entirety—whole sentences or clauses—along with statements that have been bound into noun phrases by nominalization, assuming a sophisticated enough analyser is in use. *Lexical semantics*, by contrast, is concerned with meaning down to the level of words, and sometimes to individual stems, suffixes and prefixes (Jurafsky and Martin, 2000). The idea of word-sense is a lexical semantic notion, and although much of the effort of word-sense disambigua-

tion in biomedicine is (or should be) applied in named entity classification, there are other technical terms that suffer from polysemy too—such as *clone* (DNA vs. organism) or *species* (taxonomy vs. chemistry). Conversely, failing to recognize when two words are synonymous, or when one subsumes the other, can lead to information loss. The inclusion of such relations makes a lexicon begin to resemble an ontology.

Ontologies are a familiar concept to most bioinformatics and medical informatics specialists, and indeed many biologists, due to the success of GO and UMLS, the Open Biological Ontologies (OBO) initiative[20] and various other projects that have followed in their wake. Such resources, designed with practical considerations in mind, embody fairly pragmatic approaches to the organization of knowledge compared to the languages developed by computer scientists (Bechhofer *et al.*, 2001) and the formalisms employed by philosophers (Smith, 2003). Indeed, it is interesting to note that although GO calls itself an ontology, it was originally designed as a 'controlled vocabulary', and that the ontological component of the UMLS is small and sparse compared to its lexical components. While these resources are all manually curated, an active field of research in NLP is the automatic extraction of ontological knowledge from text by detecting syntactic patterns characteristic of hyponymy, meronymy and so on. This approach has been applied with some success in biology already (Kawasaki *et al.*, 2003).

### 1.10.3 Coreference resolution

Named entity identification is a process of reference resolution, that is, a mapping from explicit linguistic entities to the extralinguistic entities to which they refer. Another crucial aspect of semantic interpretation, however, is *coreference* resolution, that is, the process of determining which linguistic entities (noun and pronoun phrases) refer to the same things—a process which can cross sentence as well as phrase barriers, and which must incorporate a certain degree of background knowledge codified in one of the lexical or ontological manners described above. There are several scenarios that must be considered here. One is the situation where a pronoun (mostly *it* and *they* and their variants, in biology) is used to stand for an entity already named, or in some cases, about to be named. Before such references can be resolved, it must first be determined whether the pronoun really is being used in a referential sense or simply as a rhetorical device (Litrán *et al.*, 2004), as in constructions like *it must first be determined...*

Beyond simple pronouns, however, it is very common for a new noun phrase to be used in place of a previously-introduced entity, or sometimes more than one entity. This phenomenon is illustrated by such phrases as *the substrate*, *this gene*, *both enzymes* or *each protein*, all of which would need to be interpreted with reference to other noun phrases in the same sentence or elsewhere. One way to increase the reliability of this process has been to incorporate background knowledge from an ontological or lexi-

[20] `http://obo.sourceforge.net/`

cal data source such as UMLS in order to filter out unlikely correspondences (Castaño *et al.*, 2002; Lin and Liang, 2004). An annotated test corpus consisting of 32 MEDLINE abstracts has been released by the Medstract[21] project, although given the short, restricted nature of abstracts, it seems intuitively likely that coreference resolution will be a harder task over full-text documents.

## 1.11 Concluding remarks

Given the breadth of current research in NLP, one must be judicious in choosing an avenue of investigation that is tractable with current resources and that pays dividends in terms of practical applications, and yet which can be easily extended and integrated with other work in related areas. From the point of view of a single investigator, it is also important to avoid embarking on projects which require unfeasibly large data preparation commitments; corpus development is an area which is best left to teams of researchers, and—at least at the more abstract levels of syntactic and semantic annotation—preferably those with both linguistic and biological skills.

Taking these criteria into account, information extraction is an eminently attractive topic. It is already well established as a research theme with a set of broad yet fairly clear and widely-applicable goals: the identification of the relationships, interactions or events that take place between a set of biological entities of interest described in a corpus of text. At the same time, despite the growing volume of work in this area, there is plenty of relevant linguistic research relating to the processing of syntax and semantics which has not yet been brought to bear on the problem. Similarly, while one can easily isolate the core problems such as relationship or event extraction to focus on, advances in most of the other topics mentioned above (such as entity identification, term-sense disambiguation, coreference resolution or ontology management) will be in a position to improve the practical performance of real-life IE applications. IE is a somewhat open-ended problem, since although the level of 'understanding' required to extract simple factoids is somewhat superficial, one can easily conceive of systems capable of capturing more of the nuances of the statements in the text, such as temporal relations, degrees of belief, quantitative comparisons and so on. Furthermore, data extracted using IE techniques can provide an important starting point for higher-level knowledge discovery methodologies. Some thoughts on these ideas are presented in Chapter 5. Test sets facilitating the benchmarking of IE algorithms in specific biological subdomains already exist (e.g. Nédellec, 2005), and most importantly, there is a real demand for IE and related technologies among biomedical 'customers' (I. Dix, AstraZeneca, pers. comm.).

For these reasons, the following experiments represent some efforts to improve the

[21] `http://medstract.org/`

current state of the art in biological IE with the application of novel methods grounded in computational-linguistic principles.

# Chapter 2

# Parsing biomedical texts

A pre-requisite for a deeper linguistic understanding of biological documents is an accurate and informative syntactic analysis of the text. This chapter describes a preliminary experiment to investigate the behaviour of several natural language parsers on an annotated corpus, with the aim of choosing one for use in an information extraction pipeline. The work presented herein is a refinement of an investigation first presented in Clegg and Shepherd (2005).

## 2.1 Motivation

Until fairly recently, most of the information extraction (IE) methods used in bioinformatics tasks could be placed into two broad categories: those based on term co-occurrence, and those based on sequential textual patterns. The earliest methods were almost entirely co-occurrence based, taking the simple proximity of two or more entity names in a corpus of text as an indicator of some kind of relationship between them. The entities under consideration might include genes, proteins, diseases, chromosomal locations etc., depending on the application. The precision/recall tradeoff in such algorithms can be tuned by narrowing or broadening the window within which proximity is detected (Ding *et al.*, 2002), or by adjusting a frequency threshold below which candidate entity pairs will be discarded (Jenssen *et al.*, 2001).

The slightly more sophisticated textual pattern methods use the presence of keywords such as interaction verbs (*regulate*, *inhibit* etc.) as indicators that a genuine relationship is being described, and descriptors of the *type* of relationship, where several distinct types are of interest (Pustejovsky *et al.*, 2002). These methods are often augmented with regular expressions describing allowable part-of-speech (POS) sequences (Domedel-Puig and Wernisch, 2005) or non-recursive phrase boundaries (Saric *et al.*, 2004) within the text, requiring the input of a POS tagger or shallow (chunking) parser respectively.

Recently, however, interest has begin to flourish in techniques that make use of more complex grammars and syntactic structure. These non-sequential models of text should in principle be better able to account for long-distance dependencies, complex phrasing and the great diversity of expression of natural language. An example of a sentence which might benefit from full syntactic parsing is shown in Figure 2.1 (see Appendix C for a glossary of the linguistic labels used in this chapter). This is an example sentence from the GENIA corpus with a graphical representation of its syntactic structure as marked by the annotators. An IE algorithm based on co-occurrence at the sentence or phrase level would incorrectly infer a relationship between *c-fos* and *c-jun*, even though no such relationship is asserted. On the other hand, a method based on regular expressions would be unlikely to contain a template broad enough to capture the relationships between *PAF* and *c-fos* or *c-jun*. However, a path can be traced across the syntax tree between *PAF* and either *c-fos* or *c-jun* via the verb phrase headed by the verb *inducing*, but the path between *c-fos* and *c-jun* does not cross a verb phrase. Therefore a suitable syntactic approach might be able to identify the relationships between *PAF* and *c-fos*/*c-jun*, despite their wide separation in the text, without allowing a false positive to occur from the close proximity of *c-fos* and *c-jun*.

Progress has been made in the analysis of biomedical text using off-the-shelf commercial parsers (Wattarujeekrit *et al.*, 2004), hand-coded grammars dedicated to a specific aspect of the molecular biology domain (Temkin and Gilder, 2003), or open-source academic parsers based on dependency- (Ahmed *et al.*, 2005) or constituent-based (Clegg and Shepherd, 2005) models of language. The research presented in this last paper and the current chapter was undertaken with the goal of selecting one or more suitable parsers for biological sentences as a pre-processing stage for semantic analysis. Constituent-based parsers were chosen on account of the freely available software and evaluation data (see below), the richness of their output, the sound theoretical underpinnings of the probabilistic context-free grammars involved (Jurafsky and Martin, 2000), and the recent flurry of research applying them to biomedical texts (Rosario and Hearst, 2004; Shimbo *et al.*, 2004; Xiao *et al.*, 2005; Bies *et al.*, 2005; Lease and Charniak, 2005).

Context-free grammars (CFGs; Chomsky, 1956) are grammars consisting of *production rules* of the form $A \rightarrow \alpha$. $A$ is a non-terminal symbol (e.g. phrase or POS label) and $\alpha$ is a string consisting of terminal symbols (words) and non-terminals, to which further production rules can be applied until all of the non-terminals have been replaced. They are described as 'generative' grammars because each one defines a language which is the set of all possible sentences (strings of terminals) that can be generated by application of production rules in the grammar (Jurafsky and Martin, 2000). Such grammars model the hierarchical arrangement of constituents (clauses and phrases) underpinning the selection and ordering of words in a sentence. If recursively defined, a generative grammar can produce infinitely many distinct sentences.

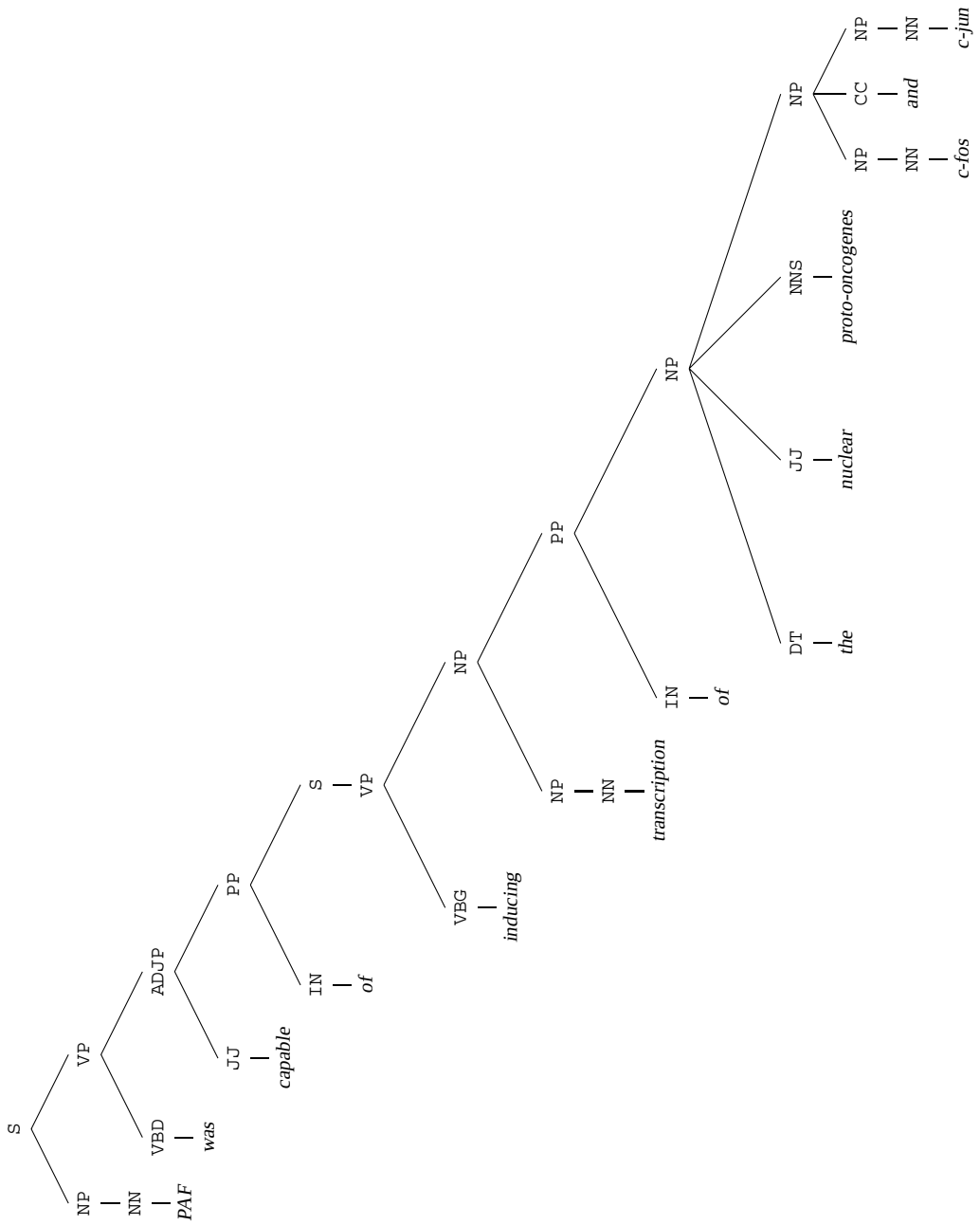


Figure 2.1: Syntactic structure of an example sentence from the GENIA corpus (paraphrased slightly for brevity).

The derivation of a sentence produced by a generative grammar is the series of rule applications which are required to produce it, starting from a root node (usually an S node for 'sentence', as in Figure 2.1). Examples of production rules include S $\rightarrow$ NP VP (read as 'sentence produces noun phrase and verb phrase') or NP $\rightarrow$ DT NP (read as 'noun phrase produces determiner and noun phrase'). Each application of a production rule is known as a production; in the resulting syntax tree, each non-terminal node represents a single production, with the node's label on the left-hand side of the production rule and the daughter label(s) on the right-hand side. The same principles underlie the syntax of most computer languages (Aho *et al.*, 1986) and have been applied to fields as far removed from linguistics as gene prediction in eukaryotic genomes (Dong and Searls, 1994) and the modeling of RNA secondary structure (Kato *et al.*, 2003). It must be noted that CFGs are only an approximation to the complexity of natural language, since they are unable to represent certain phenomena (Jurafsky and Martin, 2000), but they are used in preference to their richer cousins (context-sensitive and unrestricted grammars) because of their computational tractability.

For all but the most trivial of sentences, there will be multiple grammatically-valid parses, even where the correct part of speech (POS) has been assigned to each word—see Section 1.9, and Figure 2.2 for a biological example. Many parses, like those in these examples, have clear but contradictory meanings, but perhaps more still will have no obvious interpretation which makes sense to a human (Charniak, 1997). Modern broad-coverage, high-accuracy parsers will choose the single most likely parse (or $n$-best parses) according to a set of probabilities induced from a hand-annotated training corpus by machine learning methods. These probabilities attached to production rules distinguish probabilistic CFGs from deterministic ones; a sentence in a given language is not 'grammatical' or 'ungrammatical' but rather has a probability of occurring based on the frequencies (in the training corpus) of the production rules it uses. Typical training sets are in the region of several hundred thousand words, an order of magnitude larger than the amount of data available in the molecular biology domain. The most widely-used corpus, the PTB, consists largely of newspaper English, which is easy for linguists to syntactically annotate without any subject-specific background knowledge; the specialist vocabulary of a domain like molecular biology is a rate-limiting factor in the annotation process, which makes the adaptation of existing tools without retraining a more attractive option.

### 2.1.1 Parser selection

As discussed in Section 1.9, there are various kinds of parsers currently available, in the broad categories of partial (shallow/chunking) parsers, constituent parsers and dependency parsers. Although partial parsers have been used for biomedical NLP tasks (e.g. Leroy *et al.*, 2003), they are not well suited to capturing the long-distance dependencies

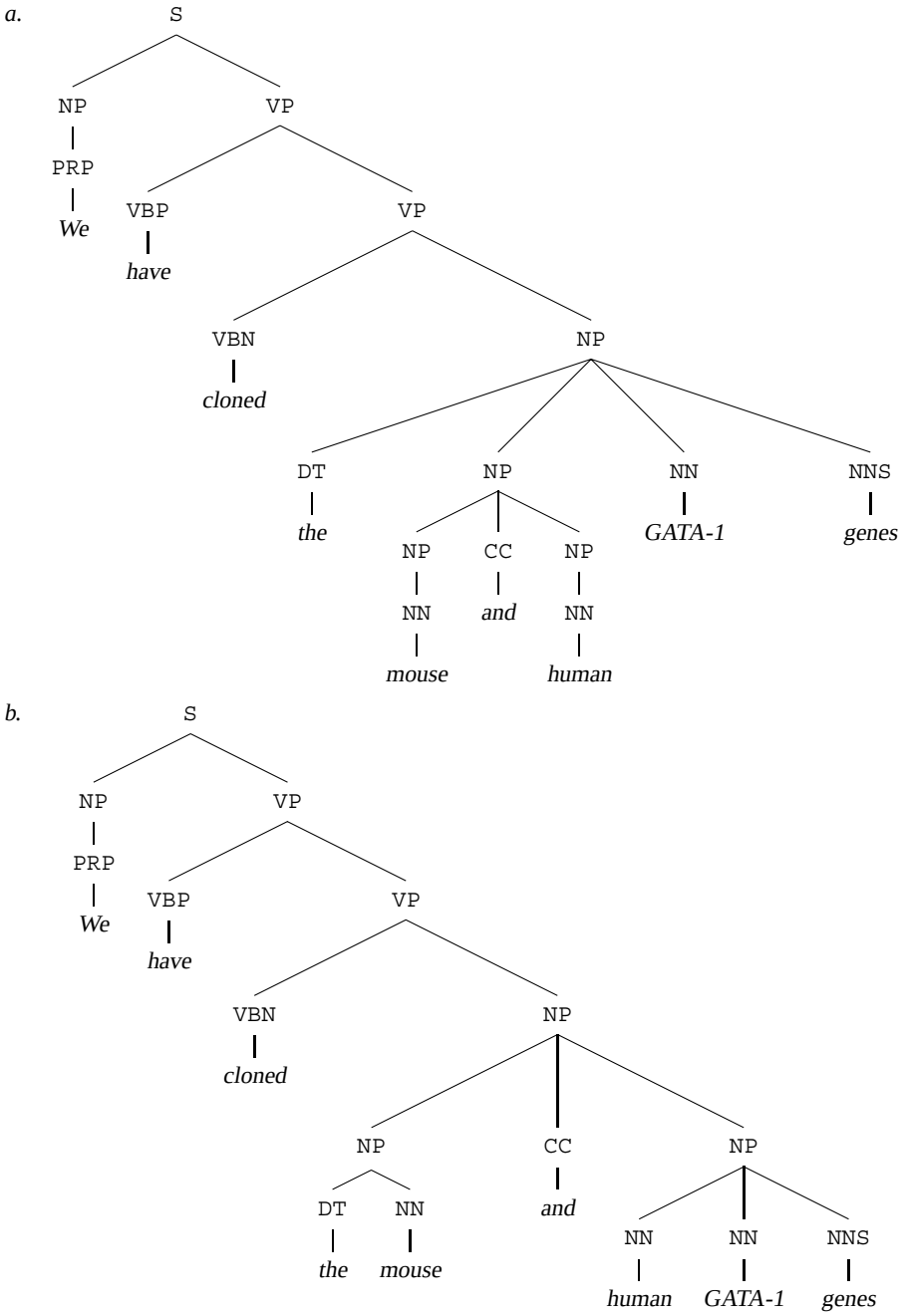


Figure 2.2: Coordinating conjunction ambiguity. Parse *a*, which is correct, implies that the GATA-1 genes of mice and humans were cloned. Parse *b* suggests instead that human GATA-1 genes and a mouse were cloned.

| Parser name | Original citation | Release used | Source language |
|---|---|---|---|
| Bikel 0.9.8[1] | Bikel (2002) | Apr 2004 | Java |
| Bikel 0.9.9c[1] | *Ibid.* | Sep 2005 | Java |
| Charniak[2] | Charniak (2000) | Aug 2005 | C++ |
| Charniak-Lease[3] | Lease and Charniak (2005) | Jul 2005 | C++ |
| Collins 1[4] | Collins (1999) | Dec 2002 | C |
| Collins 2[4] | *Ibid.* | Dec 2002 | C |
| Collins 3[4] | *Ibid.* | Dec 2002 | C |
| Stanford-lex[5] | Klein and Manning (2002) | Jun 2006 | Java |
| Stanford-unlex[5] | Klein and Manning (2003) | Jun 2006 | Java |

Table 2.1: The treebank parsers chosen for this investigation.

[1] `http://www.cis.upenn.edu/~dbikel/software.html`
[2] `ftp://ftp.cs.brown.edu/pub/nlparser/`
[3] `http://www.cog.brown.edu/Research/nlp/resources.html`
[4] `http://people.csail.mit.edu/mcollins/code.html`
[5] `http://nlp.stanford.edu/software/lex-parser.shtml`

and subtleties of phrasal attachment which could facilitate the successful analysis of a sentence like that in Figure 2.1. Constituent parsers and dependency parsers all have the appropriate level of sophistication, but a wide variety of different grammars and conceptual frameworks that makes comparing them difficult. However, there is one class of parsers that is both numerous and up-to-date, and covers a variety of different algorithms which all use the same output format (bar a few small details). These are sometimes referred to as treebank parsers as they are usually trained and optimized on the PTB and produce output conformant with its standards. For the purposes of this investigation, another advantage is that there is sufficient annotated test data available from the biological domain to test their adaptability to bioinformatics applications. Finally, the main advantage of using dependency parsers—a closer relationship between syntactic and semantic representations useful for applications like IE—can also be obtained using treebank parsers (see the following chapters).

The treebank parsers chosen for this project are listed in Table 2.1. All of them are trained on the Penn Treebank, and take roughly the same input—POS-tagged text—although the Charniak parsers have their own built-in POS taggers so take unannotated text. They all produce directly comparable output—constituent trees in formats similar to the PTB's—but differ in the details of the features of the training sentences they condition probabilities on, the learning methods they use for estimating and smoothing these probabilities, and the algorithms by which they arrive at the most probable parse for a given unseen sentence. A full discussion of the theory and implementation of the probabilistic context-free grammars embodied by these parsers is beyond the scope of this chapter, but the interested reader is referred to Jurafsky and Martin (2000, chapters

10 and 12) for background and then the papers listed in Table 2.1 for details.

All of the parsers are freely-available and come with source code. I chose to test more than one version of each to determine whether refinements made on the PTB do in fact cause corresponding improvements on biological text, as this is far from certain *a priori*. The three versions of the Collins parser are not actually separate releases, but different language models of increasing complexity. Model 2 treats complements and adjuncts of constituents as distinct categories, where a complement functions together with its parent to express one fact (*the student **of physics***), and an adjunct expresses an attribute incidental to the meaning of its parent (*the student **with red hair***).[6] It incorporates statistical parameters representing the likely patterns of allowable complements (known as subcategorization frames) for words which take complements. Model 3 incorporates model 2 and adds improved handling of relative clauses introduced with 'wh-pronouns' such as *which*, *where*, and *that*.

The Charniak-Lease version of the Charniak parser is the only one with any biological adaptation, although I attempted to ensure that the other parsers were provided with appropriate assistance from external tools to put them on a level playing field (see Section 2.1.2). The unlexicalized version of the Stanford parser (Stanford-unlex) is unlike the others in that it considers text to be a sequence of POS tags, and discards the actual word information. This makes it much more efficient than its lexicalized cousin, but it has also been suggested (Finkel *et al.*, 2004) that this strategy might prove less error-prone on biological text, which has a different vocabulary and word distribution to the PTB text on which the lexicalized parsers were trained.

### 2.1.2 Preparing the evaluation

The evaluation data for these experiments was drawn from the GENIA Treebank (GTB), a beta-stage corpus of abstracts drawn randomly from MEDLINE with the search terms "human", "blood cell" and "transcription factor". The 200 abstracts from the first release of the corpus were used. These have been manually annotated with POS tags, named entity classes and boundaries, and syntax trees which broadly follow the conventions of the PTB. I discarded the named entity information and gold-standard POS tags, and re-tagged the corpus with the MedPost POS tagger (Smith *et al.*, 2004) in PTB mode to better simulate the kind of scenario where a parser would be employed on completely unseen text. MedPost is a specialized biomedical POS tagger trained on MEDLINE abstracts which can use the PTB tagset as well as its own tagging format; its output was not hand-corrected or otherwise manually modified.

The Collins parser requires pre-tagged text from a PTB-compatible POS tagger, and the Bikel parser documentation recommends the same—it can also be run on untagged text if necessary, although I did not explore this option as it would almost certainly

[6] Examples from `http://www-rohan.sdsu.edu/~gawron/syntax/lectures/lec4.htm`

reduce performance considerably. The 'vanilla' Charniak parser has a built-in POS tagger which is trained as part of the normal syntactic training process. The Charniak-Lease parser's tagging model, on the other hand, is de-coupled from the parsing model so that it can be trained separately on any body of POS-tagged text; in fact, it is supplied pre-trained on the sentences from the broader GENIA POS corpus [7] which are not also in the GTB, plus all the sentences from sections 2–21 of the PTB.

Some manual editing was required to correct annotation errors and remove sentences with uncorrectable errors, leaving 1,757 sentences, 42,013 tokens (discounting punctuation) and 35,061 constituents in the gold standard—on average, 23.9 tokens and 20.2 constituents per sentence. All errors were reported to the GENIA group. For comparison purposes, the parsers were also tested against Section 23 of the PTB, which is customarily used as a set-aside benchmark in the computational linguistics community. In this case, instead of MedPost, the MXPOST tagger (Ratnaparkhi, 1996)—which is optimized for the PTB and similar corpora—provided POS tag information. This corpus had 2,416 sentences, 50,101 tokens (discounting punctuation) and 44,177 constituents—on average, 20.7 tokens and 18.3 constituents per sentence.

Tweaking of parser options was kept to a minimum, aside from trivial changes to allow for unexpectedly long words, long or complex sentences (e.g. default memory/time limits), and differing standards of tokenization and punctuation, although a considerable degree of pre- and post-processing by Perl scripts was also necessary to bring these into line (see Section 2.3.9 and Section 5.1.3). More detailed tuning would have massively increased the number of variables under consideration, given the number of compile-time constants and run-time parameters available to the programs; furthermore, it was assumed that each author distributes his software with an optimal or near-optimal configuration, at least for in-domain data.

### 2.1.3 Post-processing procedure

After parsing, but before calculating any of the accuracy measures described in the following section, my post-processing scripts stripped the following stand-alone punctuation symbols from the parser output and the gold standard: period, comma, semicolon, colon, and double-quotes (whether they were expressed as a single double-quotes character, or pairs of opening or closing single-quotes). This is because the attachment and labeling of these symbols is (to a large extent) a matter of convention rather than grammatical necessity.

In addition, several operations had to be performed to remove or correct technical features of the parse trees that lay outside the common set of features supported by all the parsers and the reference treebanks. The GTB and PTB both contain function tags

[7] `http://www-tsujii.is.s.u-tokyo.ac.jp/~genia/topics/Corpus/pos3.02p.html`

(see Section 1.9.4), and although software exists which can add these to parser outputs (Blaheta and Charniak, 2000), this is not universally compatible, so these tags were stripped. Any redundant constituents were removed—that is, constituents whose only daughter is another constituent of the same label, like the outer NP in the construction (NP (NP (NNS *cells*))). Similarly, the outermost constituent in any sentence (the root node of the parse tree) is customarily held within a container constituent labeled TOP, S1 or ROOT. These non-meaningful constituents were removed in order to avoid artificially inflating accuracy scores, since they do not vary between sentences.

The PTB uses the pseudo-constituents NAC and NX to mark out the internal structure of noun phrases, but the GENIA corpus simply uses NP in these cases (NP also denotes the overall noun phrase). Therefore, it was necessary to replace these tags with NP in the parses of GENIA. I also replaced PRT labels with ADVP labels throughout the parses and gold standard corpora, as these two kinds of phrases are commonly held to be equivalent for scoring purposes—see for example Collins (2003).

An high-level overview of the preparation, parsing and post-processing procedure is shown in Figure 2.3.

## 2.2 Performance evaluation methodologies

I initially chose to rate the parsers in the assessment by several different means which can be grouped into two broad classes: constituent- and lineage-based. While Sampson and Babarczy (2003) showed that there is a limited degree of correlation between the per-sentence scores assigned by the two methods, they are independent enough that a fuller picture of parser competence can be built up by combining them and thus sidestepping the drawbacks of either approach. However, overall performance scores designed for competitively evaluating parsers do not provide much insight into the aetiology of errors and anomalies, so I developed a third approach based on production rules that enabled the megabytes of syntactic data to be mined for enlightening results more effectively. These evaluation strategies are described below.

### 2.2.1 Constituent-based assessment

Most evaluations of parser performance are based upon three primary measures: labeled constituent precision and recall, and number of crossing brackets per sentence. Calculation of these scores for each sentence is straightforward. Each constituent in a candidate parse is treated as a tuple $\langle lbound, LABEL, rbound \rangle$, where $lbound$ and $rbound$ are the indices of the first and last words covered by the constituent. Precision (known in other fields as Positive Predictive Value) is the proportion of candidate constituents that are correct and is calculated as follows:

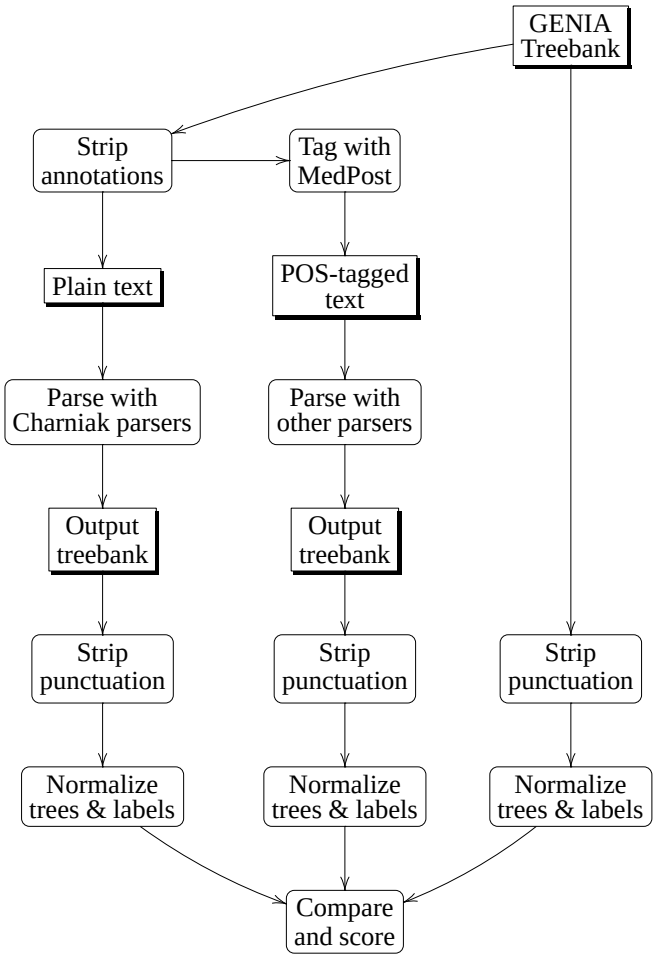


Figure 2.3: Overview of experimental protocol for parser comparison against GENIA Treebank. The protocol for comparing the parsers against Section 23 of the PTB was identical except that the MXPOST tagger was used instead of MedPost. Shaded rectangles represent data and rounded boxes are actions.

$$P = \frac{\text{\# true positives}}{\text{\# true positives + \# false positives}} \tag{2.1}$$

Recall (equivalent to the Sensitivity metric in other fields) is the proportion of constituents from the gold standard that are in the candidate parse:

$$R = \frac{\text{\# true positives}}{\text{\# true positives + \# false negatives}} \tag{2.2}$$

In other words, precision errors are false positives, and recall errors are false negatives. The crossing brackets score is reached by counting the number of constituents in the candidate parse that overlap with at least one constituent in the gold standard, in such a way that one does not subsume the other.

Although this scoring system is in wide use, it is not without its drawbacks (Sampson, 2000). Most obviously, it gives no credit for partial matches, for example when a constituent in one parse covers most of the same words as the other but is truncated or extended at one or both ends. Indeed, one can imagine situations where a long constituent is truncated at one end and extended at the other compared to the gold standard; this would incur a penalty under each of the above metrics even though some or even most of the words in the constituent were correctly categorized. One can of course suggest modifications for these measures designed to account for particular situations like these, although not without losing some of their elegance. The same is true for label mismatches, where a constituent's boundaries are correct but its category is wrong. This scoring scheme originally analysed the tree structure only, giving unlabeled precision and recall (Black *et al.*, 1991), but the improved performance of modern parsers has made labeled comparison the norm.

More fundamentally, it could be argued that by taking 'horizontal' slices through the syntax tree, these measures lose important information about the ability of a parser to recreate the gross grammatical structure of a sentence. The height of a given constituent in the tree, and the details of its ancestors and descendants, are not directly taken into account, and it is surely the case that these broader phenomena are at least as important as the extents of individual constituents in affecting meaning. However, constituent-based measures are not without specific advantages too. These include the ease with which they can be broken down into scores per label to give an impression of a parser's performance on particular kinds of constituent, and the straightforward message they deliver about whether a badly-performing parser is tending to over-generate, under-generate or mis-generate (low precision, low recall, or high crossing brackets respectively).

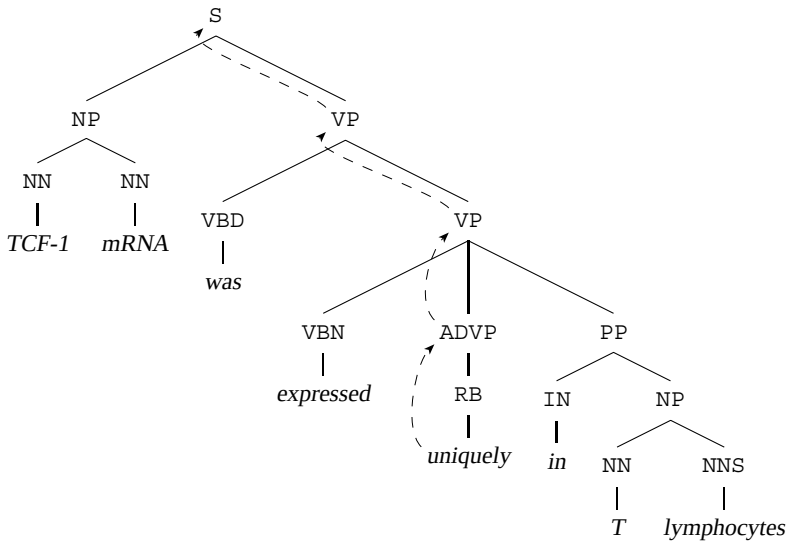


Figure 2.4: Skipping the POS tag, the lineage string for *uniquely* is: `[ ADVP ] VP VP S` . The left and right boundary markers record the fact that the `ADVP` constituent both starts and ends with this word.

### 2.2.2 Lineage-based assessment

In contrast to this horizontal-slice philosophy, Sampson and Babarczy (2003) advocate a vertical view of the syntax tree. By walking up the tree structure from the immediate parent of a given word until the top node is reached, and adding each label encountered to the end of a list, a 'lineage' representing the word's ancestry can be retrieved. Boundary symbols are inserted into this lineage before the highest constituent that begins on the word, and after the highest constituent that ends on the word, if such conditions apply; this allows potential ambiguities to be avoided, so that the tree as a whole has one and only one corresponding set of 'lineage strings' (see Figure 2.4).

Using dynamic programming, an edit distance (Levenshtein, 1966) can be calculated between each word's lineage strings in the candidate parse and the gold standard, by determining the smallest number of symbol insertions, deletions and substitutions required to transform one of the strings into the other. The leaf-ancestor ($LA$) metric, a similarity score ranging between 0 (total parse failure) and 1 (exact match), is then calculated by taking into account the lengths of the two lineages ($dist$ is the edit distance):

$$LA = 1 - \frac{dist(\text{lineage}_1, \text{lineage}_2)}{length(\text{lineage}_1) + length(\text{lineage}_2)} \tag{2.3}$$

The per-word score can then be averaged over a sentence or a whole corpus in order

to arrive at an overall performance indicator. Besides avoiding some of the limitations of constituent-based evaluation discussed above, one major advantage of this approach is that it can provide a word-by-word measure of parser performance, and thus draw attention easily to those regions of a sentence which have proved problematic (see Section 2.3.5 for an example). The algorithm can be made more sensitive to near-matches between phrasal categories by tuning the cost incurred for a substitution between similar labels, e.g. those for 'singular noun' and 'proper noun', rather than adhering to the uniform edit cost dictated by the standard Levenshtein algorithm. In order to avoid over-complicating this study, however, I chose to keep the standard penalty of 1 for each insertion, deletion or substitution.

One drawback to leaf-ancestor evaluation is that although it scores each word (sentence, corpus) between 0 and 1, and these scores are presented here as percentages for readability, it is misleading to think of them as percentages of correctness in the same way that one would regard constituent precision and recall. Indeed, the very fact that it results in a single score means that it reveals less at first glance about the broad classes of errors that a parser is making than precision, recall and crossing brackets do. Another possible objection is that since an error high in the tree will affect many words, the system implicitly gives most weight to the correct determination of those features of a sentence which are furthest from being directly observable. One might argue, however, that since a high-level attachment error can grossly perturb the structure of the tree and thus the interpretation of the sentence, this is a perfectly valid approach; it is certainly complementary to the uniform scoring scheme described in the previous section, where every mistake is weighted identically.

### 2.2.3 Production-based assessment

In order to properly characterize the kinds of errors that occurred in each parse, and to help elucidate the differences between multiple corpora and between each parser's behaviour on each corpus, I developed an additional scoring process based on comparing the production rules used to generate the structure of the sentence, between the gold standard and a candidate parse. A production, being the application of a specific production rule at a particular point in the tree, can be expressed as:

$$LABEL_p(lbound, rbound) \rightarrow LABEL_1 \ldots LABEL_n$$

Production precision and recall (and thus F-measure) can be calculated as in a normal labeled constituent-based assessment, except that a proposed production is a true positive if and only if there exists a production in the gold standard with the same parent label and boundaries, and the same daughter labels in the same order. (The respective widths of the daughter constituents, where applicable, are not taken into ac-

count, only their labels and order; any errors of width in the daughters are detected when they are tested as parents themselves.) Although the scoring scripts do not count POS tags as constituents when calculating constituent- and lineage-based scores for each parser, since the POS tags are not provided by the parsers, they do allow POS tags as daughter labels when computing the production precision and recall scores, since they occur on the right-hand sides of some of the production rules that make up each parser's grammar.

Furthermore, as an aid to the detection and analysis of systematic errors, I developed a heuristic for finding the closest-matching candidate productions in a parse, in each case where a production $PROD_g$ in the gold standard is not exactly matched in the parse.

1. First, the heuristic looks for productions with correct boundaries and parent labels, but incorrect daughters. The corresponding production rules are returned.

2. Failing that, it looks for productions with correct boundaries and daughters, preserving the order of the daughters, but with incorrect parent labels. The corresponding production rules are returned.

3. Failing that, it looks for productions with correct boundaries but incorrect parent labels and daughters. The corresponding production rules are returned.

4. Failing that, it looks for all extensions and truncations of the production (boundary modifications such that there is at least one word from $PROD_g$ still covered) with correct parent and daughter labels and daughter order, keeping only those that are closest in width to $PROD_g$ (minimum number of extensions and truncations). The meta-rules `EXT_ALLMATCH` and/or `TRUNC_ALLMATCH` as appropriate are returned.

5. Failing that, it looks for all extensions and truncations of the production where the parent label is correct but the daughters are incorrect, keeping only those that are closest in width to $PROD_g$. The meta-rules `EXT_PARENTMATCH` and/or `TRUNC_PARENTMATCH` are returned.

6. If no matches are found in any of these classes, a null result is returned.

Note that in some cases, $m$ production rules of the same class may be returned, for example when the closest matches in the parse are two productions with the correct parent label, one of which is one word longer than $PROD_g$, and one of which is one word shorter. It is also conceivable that multiple productions with the same parent or same daughters could occupy the same location in the sentence without branching, although it seems unlikely that this would occur apart from in pathologically bad parses, and the redundant constituent removal process (Section 2.1.3) would have collapsed

such cases anyway. In any ambiguous cases, no attempt is made to decided which is the 'real' closest match; all $m$ matches are returned, but they are downweighted so that each counts as $\frac{1}{m}$ of an error when error frequencies are calculated. In no circumstances are matches from different classes returned.

The design of this procedure reflects my requirements for a tool to facilitate the diagnosis and summarization of parse errors. I wanted to be able to answer questions like "given that parser A has a low recall for `NP` $\rightarrow$ `NN NN` productions, what syntactic structures is it generating in their place? Why might this be so? And what effect might these errors have on the interpretation of the sentence?" Accordingly, as the heuristic casts the net further and further to find the closest match for a production $PROD_{\ g}$, the classes to which it assigns errors become broader and broader. Any match at stages 1–3 is not simply recorded as a substitution error, but a substitution for a *particular* incorrect production rule. However, matches at stages 4 and 5 do not make a distinction between different magnitudes of truncation and extension, and at stage 5 the information about the daughters of incorrect productions is discarded. This allowed me to identify broad trends in the data even where the correspondences between the gold standard and the parses were weak, yet nonetheless recover detailed substitution information akin to confusion matrices where possible.

Similar principles guided the decision not to consider extensions and truncations with different parent labels as potential loose matches, in order to avoid uninformative matches to productions at different levels in the syntax tree. In practice, the matches returned by the heuristic accounted for almost all of the significant systematic errors suffered by the parsers (see Section 2.3.7)—null matches were infrequent enough in general that their presence in larger numbers on certain production rules was itself useful from an explanatory point of view.

### 2.2.4 Alternative approaches

Several other proposed solutions to the evaluation problem exist, and it is an ongoing and continually challenging field of research. Suggested protocols based on grammatical or dependency relations (Crouch *et al.*, 2002), head projection (Ringger *et al.*, 2004), alternative edit distance metrics (Roark, 2002) and various other schemes have been suggested. Many of these alternative methodologies, however, suffer from one or more disadvantages, such as close coupling to a different syntactic formalism (e.g. head-driven phrase structure grammar) or class of parser (e.g. partial parsers), or a requirement for a specific manually-prepared evaluation corpus in a non-treebank format. In addition, none of them deliver the richness of information that production-based assessment does, particularly in combination with the other methods outlined above. A different approach to parser evaluation is explored in the next chapter.

## 2.3 Results and discussion

Here I present the comparative parser accuracy results for all parsers on each corpus, followed by a more detailed analysis of the errors suffered by the parsers on the GENIA corpus, along with some computational efficiency measurements.

### 2.3.1 Overall performance comparison

The results of this experiment are summarized in Table 2.2, showing the scores on both GENIA and the PTB. The $LA$ score given is the mean of the leaf-ancestor scores for all the words in the corpus, and the precision and recall scores are taken over the entire set of constituents in the corpus. Initially, these measures were calculated per sentence, and then averaged across each corpus, but the presence of pathologically short sentences such as *Energy.* gives an unrepresentative boost to per-sentence averages. (Interestingly, many published papers do not make clear whether the results they present are per-sentence averages or corpus-wide scores.)

*F-measure* (effectiveness; van Rijsbergen, 1979) is the harmonic mean of precision and recall; it is a balanced score that penalizes algorithms which favour one to the detriment of the other, and is calculated as follows:

$$F = \frac{2 \times P \times R}{P + R} \tag{2.4}$$

The first important point to note about these scores is that while the Charniak and Stanford parsers processed both corpora without any parse failures (completely unparseable sentences), the other parsers encountered total failures on one or both corpora, so their $LA$, precision, recall and $F$ scores are distorted (see below). Some parse failures did occur on the PTB but they were more widespread and numerous on GENIA. Parse failures aside, the overall difference in difficulty between the two corpora is immediately apparent, with each parser version achieving more than twice as many perfect parses on the PTB as on GENIA. However, all of them proved to be at least reasonably capable on this unfamiliar genre of writing, indicating that an NLP strategy based on applying PTB-trained parsers to biological texts is not unreasonable. In terms of both $LA$ and $F$ scores, the Charniak parser comes out on top when processing the PTB, and the Charniak-Lease parser when parsing GENIA.

### 2.3.2 Parse failures

Since most of the parsers suffered from a considerable number of parse failures in GENIA, Table 2.3 shows recalculated scores based on evaluation of successfully-parsed sentences only. Conflating the performance drops caused by poorly parsed sentences

| | Raw scores on GENIA (1,757 sentences) | | | | | |
|---|---|---|---|---|---|---|
| Parser | $LA$ | $P$ | $R$ | $F$ | Perfect | Failure |
| Bikel 0.9.8 | 91.1 | 81.4 | 77.5 | 79.4 | 14.3 | 0.06 |
| Bikel 0.9.9c | 90.8 | **81.6** | 77.2 | 79.4 | 14.5 | 0.11 |
| Charniak | 90.8 | 79.3 | 77.0 | 78.1 | 13.3 | **0.00** |
| Charniak-Lease | **91.5** | 81.5 | **78.8** | **80.2** | **14.9** | **0.00** |
| Collins 1 | 88.7 | 79.1 | 73.9 | 76.4 | 13.2 | 0.68 |
| Collins 2 | 87.9 | 81.3 | 74.5 | 77.8 | 14.0 | 1.42 |
| Collins 3 | 86.3 | **81.6** | 73.3 | 77.2 | 14.0 | 2.28 |
| Stanford-lex | 88.3 | 72.8 | 69.6 | 71.1 | 10.0 | **0.00** |
| Stanford-unlex | 88.6 | 73.9 | 70.8 | 72.3 | 11.1 | **0.00** |
| | Raw scores on PTB (2,416 sentences) | | | | | |
| Parser | $LA$ | $P$ | $R$ | $F$ | Perfect | Failure |
| Bikel 0.9.8 | 94.5 | 88.1 | 88.2 | 88.2 | 33.9 | 0.04 |
| Bikel 0.9.9c | 94.5 | 88.1 | 88.2 | 88.2 | 34.1 | **0.00** |
| Charniak | **95.1** | **89.9** | **89.7** | **89.8** | **38.3** | **0.00** |
| Charniak-Lease | 94.8 | 88.9 | 89.0 | 89.0 | 36.6 | **0.00** |
| Collins 1 | 94.2 | 86.9 | 86.8 | 86.8 | 31.5 | **0.00** |
| Collins 2 | 94.4 | 87.4 | 87.3 | 87.3 | 33.9 | 0.04 |
| Collins 3 | 94.3 | 87.4 | 87.2 | 87.3 | 33.5 | 0.08 |
| Stanford-lex | 93.1 | 84.4 | 85.2 | 84.8 | 27.0 | **0.00** |
| Stanford-unlex | 92.5 | 83.7 | 83.4 | 83.6 | 26.4 | **0.00** |

Table 2.2: Initial performance comparison. All scores are percentages. Leaf-ancestor ($LA$) score is an average over all the words in each corpus. Precision ($P$) and recall ($R$), and therefore F-measure ($F$), are averages over all the constituents in each corpus. Perfect and Failure indicate the proportion of sentences which were parsed perfectly, and completely unparseable, respectively.

with those caused by totally unparseable sentences, where no constituents were generated at all, gives an inaccurate picture of parser behaviour.

Sentence failures are clearly more of a problem for the Collins parser than for the others. There is a known problem with models 2 and 3 failing on two sentences in the PTB section 23 due to complexity, but this problem is exacerbated in GENIA, with even the simpler model 1 failing on a number of sentences, one of which was only 24 words long plus punctuation:[8]

> *Moreover, kappa 1-kappa 3 can each be deleted from the TNF-alpha promoter with little effect on the gene's inducibility by PMA.*

[8]The possessive suffix *'s* is treated as a distinct word by treebank-style parsers.

| | Scores on GENIA, successfully-parsed sentences only | | | | | |
|---|---|---|---|---|---|---|
| Parser | $LA$ | $P$ | $R$ | $F$ | Mean $X$ | # parsed |
| Bikel 0.9.8 | 91.2 | 81.4 | 77.5 | 79.4 | 2.05 | 1,756 |
| Bikel 0.9.9c | 91.3 | **81.6** | 77.6 | 79.6 | 2.00 | 1,755 |
| Charniak | 90.8 | 79.3 | 77.0 | 78.1 | 2.14 | **1,757** |
| Charniak-Lease | **91.5** | 81.5 | **78.8** | **80.2** | **1.90** | **1,757** |
| Collins 1 | 90.5 | 79.1 | 75.4 | 77.2 | 2.30 | 1,745 |
| Collins 2 | 91.2 | 81.3 | 77.2 | 79.2 | 2.01 | 1,732 |
| Collins 3 | 91.3 | **81.6** | 77.4 | 79.4 | 1.95 | 1,717 |
| Stanford-lex | 88.3 | 72.8 | 69.6 | 71.1 | 3.18 | **1,757** |
| Stanford-unlex | 88.6 | 73.9 | 70.8 | 72.3 | 2.98 | **1,757** |
| | Scores on PTB, successfully-parsed sentences only | | | | | |
| Parser | $LA$ | $P$ | $R$ | $F$ | Mean $X$ | # parsed |
| Bikel 0.9.8 | 94.5 | 88.2 | 88.3 | 88.2 | 1.07 | 2,415 |
| Bikel 0.9.9c | 94.5 | 88.1 | 88.2 | 88.2 | 1.09 | **2,416** |
| Charniak | **95.1** | **89.9** | **89.7** | **89.8** | **0.88** | **2,416** |
| Charniak-Lease | 94.8 | 88.9 | 89.0 | 89.0 | 0.99 | **2,416** |
| Collins 1 | 94.2 | 86.9 | 86.8 | 86.8 | 1.23 | **2,416** |
| Collins 2 | 94.5 | 87.4 | 87.4 | 87.4 | 1.19 | 2,415 |
| Collins 3 | 94.5 | 87.4 | 87.4 | 87.4 | 1.18 | 2,414 |
| Stanford-lex | 93.2 | 84.4 | 85.2 | 84.8 | 1.40 | **2,416** |
| Stanford-unlex | 92.5 | 83.7 | 83.5 | 83.6 | 1.55 | **2,416** |

Table 2.3: Performance scores, discounting all parse failures. Scores for the Charniak and Stanford parsers, and Collins model 1 on the PTB, are shown again for comparison, although they did not fail on any sentences. Once again, all scores are percentages, except for Mean $X$ (average crossing brackets per successfully-parsed sentence) and number parsed.

Apart from this example, the parse failures for all of the parsers tended to occur in longer, more complex sentences.

Discounting unparseable sentences, the three Collins models do show a consistent monotonic increase in precision, recall and $LA$ from the simplest to the most complex, accompanied by a decrease in the number of crossing brackets per sentence. Interestingly, these intervals are much more pronounced on GENIA than on the PTB, where the performance seems to level off between models 2 and 3. Difficult sentences aside, then, it appears that the advanced features of models 2 and 3 are actually more valuable on this unfamiliar corpus than on the original development domain—but only when they do not trip the parser up completely. In the documentation supplied with his parser, Collins suggests a strategy whereby model 3 is used to obtain a parse where possible,

falling back to model 2 and then model 1 if necessary on failure, thus taking advantage of the improved features of models 2 and 3 without sacrificing robustness. Other methods for combining the output of several parsers are discussed in Clegg and Shepherd (2005).

### 2.3.3 Lexicalized vs. unlexicalized parsing

It is interesting to examine the behaviour of the unlexicalized Stanford parser (Stanford-unlex, which only considers POS tags, rather than actual words) compared to its lexicalized sister (Stanford-lex), and to all the other parsers which are also lexicalized. Although Stanford-lex scores marginally better on every measure on the PTB, the situation on GENIA is reversed, with Stanford-unlex coming out ahead. This indicates that Stanford-lex is making mistakes due to the unfamiliar nature of the biological domain vocabulary in GENIA, which Stanford-unlex is oblivious to as it is making decisions based only on the POS tags of the words (supplied by MedPost).

Finkel *et al.* (2004) suggest that this phenomenon makes Stanford-unlex particularly suitable for use in the biological domain, and if one were to compare it only to its lexicalized cousin, this would be a fair conclusion. However, in the broader comparison of multiple parsers presented here, one can see that this conclusion is only part of the bigger picture. The Bikel, Charniak and Collins parsers all do consistently better than Stanford-unlex on those sentences that do not cause complete parse failure, despite the fact that they are all lexicalized. The Charniak parsers, like the Stanford parsers, do not fail on any sentences, and both outperform Stanford-unlex across the board on both corpora. This is impressive given that the vanilla Charniak parser has the 'wrong' POS-tagging model for GENIA.

Therefore, a more accurate conclusion might be that the unlexicalized approach is a *potentially* powerful tool when tackling lexically unfamiliar domains like biology, but Stanford's implementation is not good enough to compete with the state of the art in lexicalized parsers.

### 2.3.4 Precision-recall balance

The results in Table 2.3 show that all the parsers on GENIA consistently score lower for recall than they do for precision. In contrast, for the PTB these scores tend to come out roughly equal. This indicates that the parsers are 'under-generating' on GENIA, that is, producing sparser parse trees than those recorded by the annotators. However, since both corpora have roughly the same distribution of constituents per sentence (Clegg and Shepherd, 2005), this suggests that there are unfamiliar constructions in GENIA that the parsers are not fleshing out sufficiently.

| | Charniak | Charniak-Lease |
|---|---|---|
| Correctly-tagged words | 34,715 | 38,543 |
| ...Mean $LA$ | 92.1 | 92.7 |
| Incorrectly-tagged words | 7,298 | 3,470 |
| ...Mean $LA$ | 84.6 | 77.5 |
| Mean LA score, all words | 90.8 | 91.5 |

Table 2.4: Mean $LA$ scores on GENIA for the Charniak parsers, broken down by words that are correctly or incorrectly POS-tagged.

### 2.3.5 Part-of-speech tagging

While the specialized MedPost tagger which was used to prepare the corpus for the other parsers achieved a POS-tagging accuracy of 93% on GENIA, and the Charniak-Lease tagging model achieved 96%, the original Charniak parser's built in tagger only achieved 83%, since it was trained on the PTB only and therefore not exposed to most of the specialised vocabulary found in GENIA.

In order to test the impact of this design decision on the Charniak parser, I calculated the mean $LA$ scores for correctly-tagged and incorrectly-tagged words only, for each version; see Table 2.4. Each version achieves a similar score for the lineages of correctly-tagged words, with the overall score for the vanilla version being dragged down further by larger numbers of incorrectly-tagged words than is the case for the Lease version. The results suggest that incorrectly-tagged words account for much of the difference in overall parser performance between the vanilla version and the extended-vocabulary Lease version.

### 2.3.6 Parser effectiveness by constituent type

To help elucidate further the strengths and weaknesses of the parsers on the GENIA corpus, I then measured precision and recall for each gold standard constituent, and calculated the F-measure per phrase type (see Section 2.3.1). The results are presented in Figure 2.5. It is immediately apparent that all the parsers achieve roughly similar performance on each of the common constituent types, apart from `PRN` (parenthetical phrases) which seem to have posed a particular problem for the Stanford parsers. This aside, the parsers only differ strikingly from each other on the comparatively rare constituents toward the right of the figure. This suggests that a significant proportion of the errors made on the more common phrase types are caused by genuine differences between GENIA and the training corpus, whether they are differences of English usage or merely of convention, rather than syntactic quirks that bring to light specific and idiosyncratic deficiencies in any one parser's approach.

It is not obvious why all the parsers performed so badly on adjective phrases

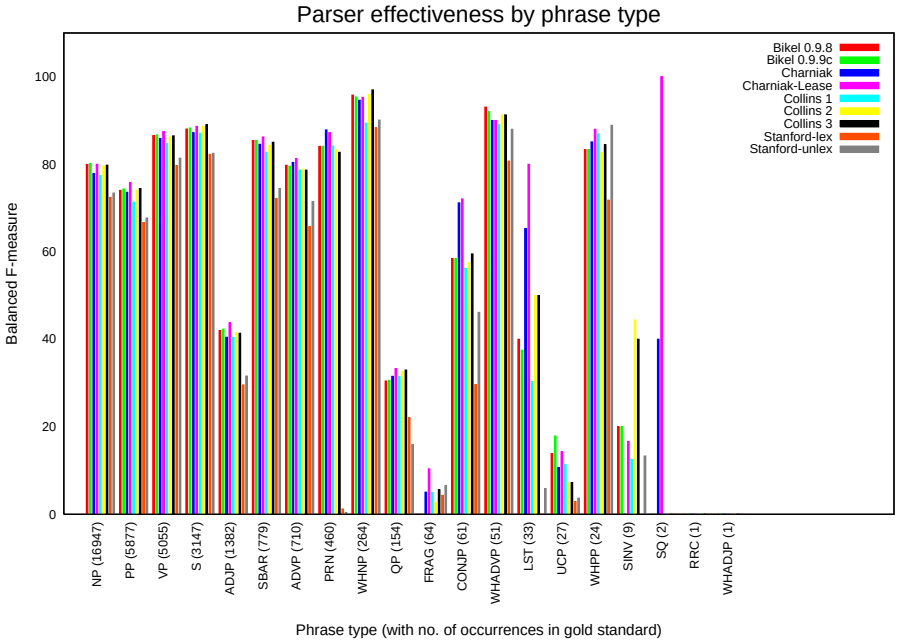


Figure 2.5: Parser effectiveness by phrase type on the GENIA corpus. The number of times each phrase type appears is given in brackets after its name.

(ADJP), especially since adjectives themselves (JJ) were tagged very effectively by MedPost (87% correct), the Charniak-Lease parser (92%) and even the original Charniak parser which did not have the benefit of any exposure in training to biomedical adjectives (90%). However, an analysis at the level of production rules (see below) can shed some light on this.

### 2.3.7 Analysis of production errors

Much more detailed information can be garnered from an inspection of the F-measures for individual production rules—calculated as described in Section 2.2.3, with a true positive requiring that the parent and daughter nodes all match. Figure 2.6 shows these results for the most common production rules found in GENIA; while they constitute less than 1% of all the distinct rules used in the corpus, they account for over 20% of the instances of actual productions. It is interesting to note that while Figure 2.5 averages the effectiveness over all parent nodes with the same phrase type, Figure 2.6 highlights particular difficulties in retrieving specific syntactic structures. For example, all parsers do very well on NP $\rightarrow$ DT NN constructions (single nouns with determiners), since a determiner is a closed-class word that characteristically appears at the start of noun phrases, but much more poorly on NP $\rightarrow$ NP CC NP constructions (conjoined noun phrases, such as *monocytes and macrophages*). An analysis based purely on phrase

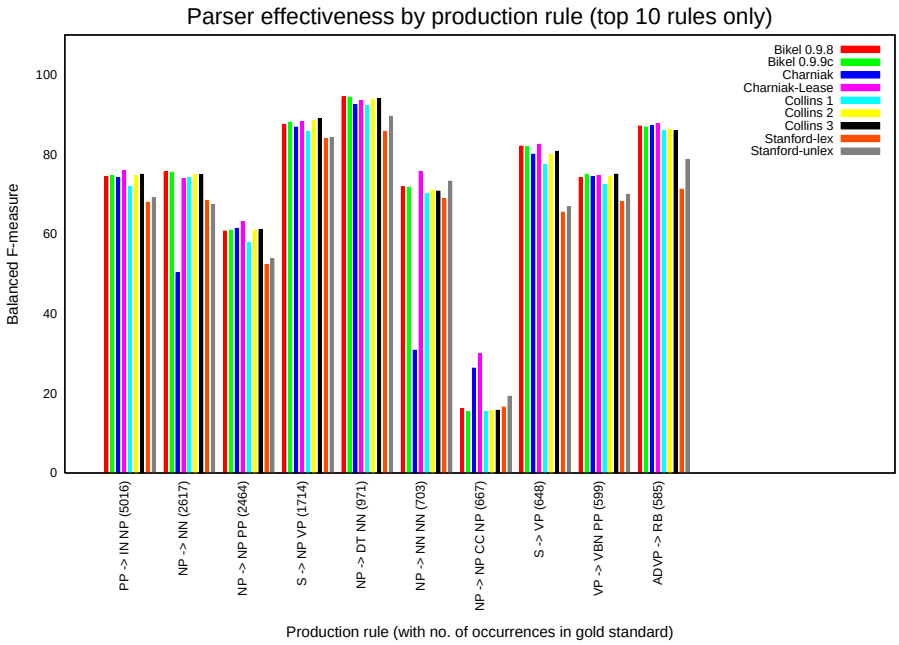


Figure 2.6: Parser effectiveness by production rule on the GENIA corpus. The number of times each production rule is used is given in brackets after its name.

types cannot make this distinction.

Another important point raised by the results in Figure 2.6 is the issues all parsers encounter with prepositional phrase attachment (see Section 1.9). It is demonstrated by the comparatively low scores for noun and verb phrases with prepositional daughters (NP $\rightarrow$ NP PP and VP $\rightarrow$ VBN PP) in Figure 2.6 compared to the scores for noun and verb phrases in general in Figure 2.5. As well as these issues that affect all the parsers, certain phenomena specific to individual parsers are apparent in Figure 2.6. The vanilla Charniak parser scores particularly badly on the productions NP $\rightarrow$ NN and NP $\rightarrow$ NN NN compared to the others, and the reasons for this are investigated below. On the other hand, both Charniak parsers are clearly better at reconstructing coordinating conjunctions than the others (NP $\rightarrow$ NP CC NP) although this is still a difficult problem.

In order to determine the effects of the errors in Figure 2.6 on the resulting parse trees, I applied the heuristic (see Section 2.2.3) to match each missing production with the closest production in the parse. This produced a large amount of data, aggregated first by missing production and then by replacement production, for each parser. While many of the classes of error are difficult to make general statements about, there are a few kinds where this level of detail allowed me to diagnose the causes of the problem or to differentiate between significant mistakes and insignificant differences of representation.

There are several cases where the knock-on effects of the original Charniak parser's vocabulary problem are evident in these results. Instead of NP → NN, most parsers make the mistake of proposing a wider noun phrase around 800 times (over 1,000 for the Stanford parsers), a phenomenon that may be indicative of parsers collapsing the internal structure of multi-word noun phrases. However, the Charniak parser additionally makes over 500 mistakes where the daughter is labeled as a proper rather than common noun. Presumably, this is partly due to non-proper nouns in GENIA that begin with capital letters, such as *Staphylococcus*, *Tissue Factor* or *Northern*. (Similarly, the Charniak-Lease parser very frequently makes the inverse of this mistake on the PTB.) Such errors are of minor significance, especially when one considers that in an IE scenario a dedicated named-entity recognizer would be responsible for tagging the noun phrases of interest. A similar effect is visible for NP → NN NN, where the Charniak parser is most likely to confuse the first noun for an adjective, a rare mistake amongst the others. This substitution could conceivably cause more interpretational problems since a noun generally represents an entity or process, whereas an adjective generally corresponds to an attribute of an entity or process.

All parsers had problems with NP → NP CC NP productions, which can be attributed at least in part to small differences in the annotation scheme for co-ordinated constituents between GENIA and the PTB.[9] In brief, this leads to coordinations being annotated with more internal structure in GENIA than in the original corpus on which the parsers were all trained, although many of the errors that result from these differences can be considered somewhat illusory. For example, a common error for most of the parsers is to omit the daughter noun phrases when these consist only of single words, and thus 'flatten out' the coordination structure (see Figure 2.7), as this is the convention used in the PTB; however, it is after all just a convention, and it is hard to imagine situations where this would adversely affect the interpretation of a sentence.

A different kind of harmless error is demonstrated by inspecting the substitutions for the production ADVP → RB. This is always a single-word production, since the label on the right-hand side is the part of speech tag for an adverb rather than another phrase type. The most common mistake for all the parsers is to simply omit this production, suggesting no alternative production with the same parent, child or location. This leaves the adverb attached directly to the parent phrase (typically a verb phrase), but semantically speaking, it is not clear that this would impede the adverb's ability to modify the verb to which it is attached (see Figure 2.8). Indeed, both corpora use both versions in roughly equal numbers with no clear separation of duties, and as a result the parsers show little consistency in their handling of this phenomenon.

A similar phenomenon is behind the poor performance of all the parsers on ADJP constituents, as noted above. By far the most common production rule expanding ADJP

[9] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/topics/Corpus/manual-for-bracketing.html

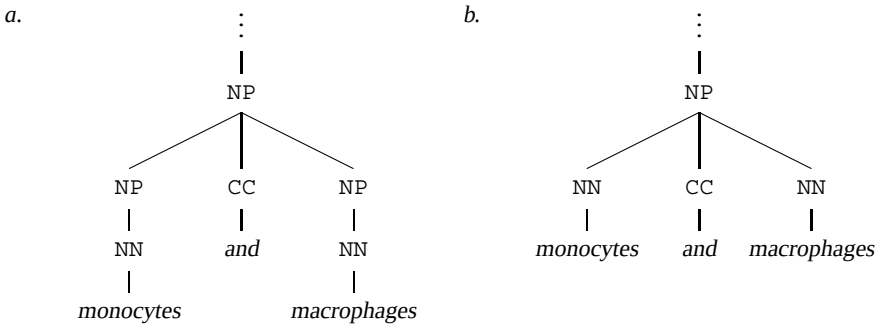


Figure 2.7: Conventions for the representation of coordinating conjunctions in GENIA (*a*) and the PTB (*b*); the parsers are all trained on the PTB and thus follow its standard. The additional layer of NP nodes is a matter of policy rather than semantic necessity.

in the gold standard is ADJP → JJ, another single-word production. For each of the parsers, almost every missing instance of this production is either replaced by a wider adjective phrase, subsuming some of the material adjacent to the adjective, or left out completely, leaving the adjective attached directly to the parent phrase with no change of meaning. These two cases are split roughly 50-50. In the former case, any semantic effects depend on the specific details of the tree for each sentence in which it occurs, but in the latter case, semantics are unaffected.

Such errors of structural convention account for at least some of the precision-recall disparity discussed in Section 2.3.4.

### 2.3.8 Computational efficiency

Parsing is not a fast process, and is arguably less suited to high-throughput, broad-coverage literature sweeps than to focused analyses of document collections relating to specific subjects. There is no reason however why parsing cannot be distributed over large clusters of individual computers, since it is a canonically easy task to parallelize, and in fact this strategy has been used to parse all of MEDLINE using a head-driven phrase structure parser (a slightly different formalism to those discussed here) on a network of several hundred PCs (J. Tsujii, U. of Tokyo, pers. comm.).

The easily distributable nature of parsing mitigates concerns over its computational cost somewhat, but it is nonetheless useful to determine some measure of efficiency for each of the parsers. To that end, I measured the execution times of each parser on the GENIA treebank using the GNU `time` command. The total processor time for each parser, calculated as the sum of the user and system times for the process as reported by `time`, is given in Table 2.5. The times do not include pre- or post-processing scripts although these are negligible compared to the actual parsing process in each case. All processes were running on one processor of a 3 GHz SMP Linux PC, without exclusive

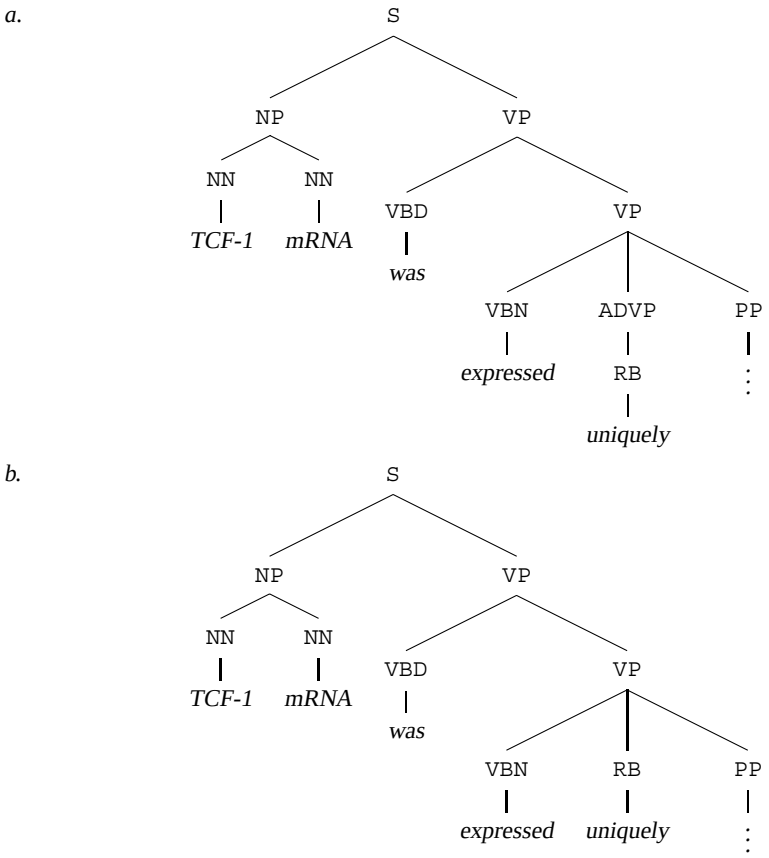


Figure 2.8: Single-word adverbs can be plausibly located within minimal adverb phrases (*a*), or attached directly to the parent verb phrase (*b*), without changing the interpretation of a sentence. Both versions occur in both corpora.

access. Had the parsers been allowed exclusive utilization of the machine, the actual wall times[10] would have been close to the total processor times.

Further analysis of the `time` output indicated that the older Bikel parser, and the lexicalized Stanford parser, suffered much more from I/O page faults (swapping memory to/from disk) than the others, with Bikel causing 133,000 page faults and Stanford-lex 1.3 *million*. This suggests that much of the performance difference between these parsers and the others is down to excessive memory usage, particularly for Stanford-lex. Indeed, I observed that the Java virtual machine running Stanford-lex was taking up over 2.5 gigabytes of system memory at one point. The number of I/O page faults suffered by the other parsers ranges from 10,000 for Stanford-unlex to none at all for the much more efficient Charniak parsers.

[10]The amount of real time passed during the execution of the process.

| Parser | Time (h:mm:ss) |
|---|---|
| Bikel 0.9.8 | 7:21:08 |
| Bikel 0.9.9c | 3:20:38 |
| Charniak | 1:49:34 |
| Charniak-Lease | 1:18:36 |
| Collins 1 | **0:52:55** |
| Collins 2 | 1:16:22 |
| Collins 3 | 1:21:33 |
| Stanford-lex | 7:12:31 |
| Stanford-unlex | 2:30:02 |

Table 2.5: Total processor time for each parser on the GENIA treebank, on a 3GHz PC.

### 2.3.9 Notes on tokenization and punctuation

Although the tagset and linguistic conventions used by the parsers and corpora in this evaluation conform in theory to the *de facto* PTB standards, many weeks of programming and testing time were spent developing 'plumbing' scripts to account for the different representations of tokenization and punctuation across the various parsers, POS taggers and corpora. The impact of such considerations on a multi-stage NLP project must not be underestimated. A fuller discussion of these issues is presented in Section 5.1.3.

## 2.4 Concluding remarks

In terms of sheer parse accuracy, the Charniak-Lease parser emerges as the overall winner, its dedicated biomedical POS-tagging model making it competitive with those parsers which rely on MedPost, and giving it a spectacular lead over the original Charniak parser which has a general-English POS tagger built in. It achieves this without sacrificing the robustness and efficiency of the original version; both of these considerations are important in parser selection for real-world NLP projects as slowness and parse failures are highly undesirable. However, it must be noted that while MedPost is trained on a broad selection of MEDLINE abstracts across various topics, the Charniak-Lease POS-tagging model is trained on GENIA (albeit a different section of GENIA to that which was used in this evaluation), meaning it has a home-topic advantage.

Since these experiments were performed, the Mining the Bibliome project [11] has released two syntactically-annotated corpora covering different biomedical topics than GENIA, and a similar experiment on these would help tease out the magnitude of this advantage, as would a retrained POS-tagging model for this parser (trained for

[11] `http://bioie.ldc.upenn.edu/`

example on the MedPost training set). However, the differences between all these data sources (both of superficial formatting standards and subtle but important linguistic conventions) meant that such investigations were not judged to be an optimum use of research time.

One general lesson learnt from the analysis of errors at the production level is that the evaluation of performance with overall parse accuracy scores is seriously limited. While high $LA$ and $F$ scores indicate that a particular parser is able to reproduce the gold standard syntactic annotation accurately, including any conventions or quirks that the annotators may have adopted, lower scores do not reveal whether the errors responsible are semantically significant. Nor do they assist in the identification of specific problem areas, such as prepositional phrase attachment or the original Charniak parser's POS-tagging problems. It seems vital, therefore, that any IE system (or more broadly, any practical NLP project) that employs a syntactic parsing stage is designed and tested with these considerations in mind.

Various methods have been devised to boost the accuracy of parses of biomedical text, despite the absence of sufficient quantities of syntactic data to retrain on, using both knowledge-poor (Clegg and Shepherd, 2005) and knowledge-rich (Lease and Charniak, 2005) approaches. While statistically-significant increases in performance scores have been achieved, it is not clear whether these gains equate to more meaningful and correct parses or simply more accurate reconstructions of the conventions of the test corpora. There are a number of methods for inferring semantic information from syntactic data (Blaheta and Charniak, 2000; Miller *et al.*, 2000, e.g.), and it is ultimately the effect of any parsing improvements upon such methods that will determine their worth. The following chapters explore these ideas further.

# Chapter 3

# Generating and evaluating dependency graphs

Although constituent trees (see Section 1.9.1 and the previous chapter) are the syntactic standard of choice for many of the best parsers available, they are not the easiest structures to extract semantic information from, nor are they necessarily simple to unambiguously evaluate. This chapter explores these themes; it is based on the work first published in Clegg and Shepherd (2007a).

## 3.1 Motivation

As described in Section 1.9.2, dependency grammars provide an alternative way of characterizing syntax to the constituent-based model used in the previous chapter, and one that is convenient and tractable for the extraction of semantic information. If one's ultimate goal is to recreate a network of biological entities that are represented by a network of semantic concepts, it makes sense intuitively that a network-like linguistic representation would make a good starting point. While purpose-built dependency parsers of various kinds exist, the problems of comparing and selecting dependency parsers for practical use have been discussed already. In brief, the lack of consistency between different dependency grammars and the lack of dependency-annotated biological text for evaluation are particularly problematic. While Pyysalo *et al.* (2007a) have begun to address the latter problem, their dataset was unfortunately not available when this experiment was carried out (see also Section 5.1.2 and Section 3.4.2).

Fortunately, a recently published extension to Stanford University's NLP toolkit (de Marneffe *et al.*, 2006) provides a procedure for the straightforward mapping of constituent trees to corresponding dependency graphs using a set of deterministic rules. [1]

[1] Hereafter referred to as the Stanford algorithm.

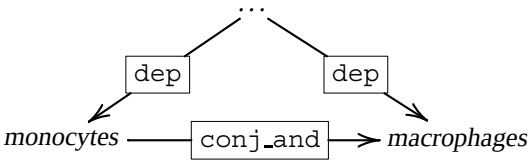


Figure 3.1: Using the Stanford algorithm, both semantically-identical tree fragments in Figure 2.7 map to this graph fragment. Both elements of the conjunction are attached directly to the parent phrase by the same dependency type, determined by their grammatical role in the sentence (shown just as `dep` here). Another dependency between them reflects their order in the sentence.

This means that any parser that outputs standard syntax trees can be used in place of a dedicated dependency parser, and therefore the advantages of accuracy, compatibility and transparent comparability of treebank parsers can be combined with the intuitive semantic convenience of dependency grammars.

While it is true that dependency graphs typically contain less information than constituent trees, one advantage of this approach over native dependency parsing is that the constituent trees are still available if they are required as input to NLP algorithms which rely on them. Constituent trees have been used to tackle such problems as pronoun resolution (Ge *et al.*, 1998), labeling phrases with semantic roles such as `CAUSE`, `EXPERIENCER`, `RESULT` or `INSTRUMENT` (Gildea and Jurafsky, 2002), automatic document summarization (Knight and Marcu, 2000), unsupervised lexicon acquisition (Merlo and Stevenson, 2001), and the assignment of functional category tags like `TEMPORAL`, `MANNER`, `LOCATION` or `PURPOSE` to phrases (Blaheta and Charniak, 2000). All of these features may be of use in a fully-featured NLP system, so it is desirable to retain the original phrase-structure representation of each sentence as well as the final dependency graph.

There is another useful side-effect of taking such an approach, which circumvents some of the drawbacks of constituent- and lineage-based evaluation identified in the Chapter 2. I described in that chapter certain linguistic constructions that vary according to the conventions of different parsers or corpora, and which might look like errors if one were to inspect only the overall accuracy scores, even though the semantics are identical between the different representations. Examples of these are shown in Figure 2.7 and Figure 2.8. When converted to dependency graphs using the Stanford algorithm, many of these insignificant differences are removed; both tree fragments in Figure 2.7 result in the graph fragment in Figure 3.1, and both in Figure 2.8 result in Figure 3.2.

This process therefore provides a convenient way to evaluate constituent parsers on those aspects of their output that most affect meaning, as well as forming a useful inter-

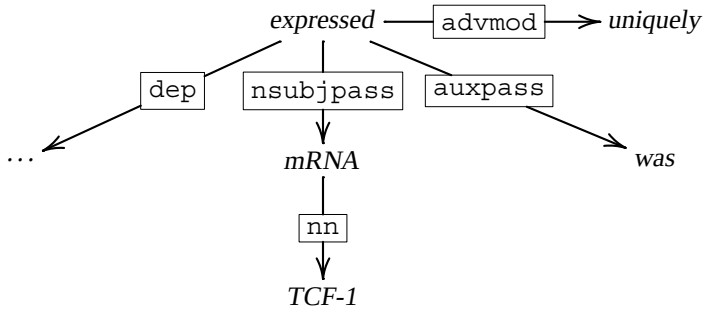


Figure 3.2: Using the Stanford algorithm, both semantically-identical tree fragments in Figure 2.8 map to this graph fragment. The adverb *uniquely* is attached directly to the verb it modifies by an `advmod` dependency, regardless of whether it was contained within an unnecessary adverbial phrase constituent.

mediate representation between phrase structure and logical predicates. Furthermore, given such a framework, it becomes easy to define application-specific evaluation criteria reflecting the requirements that will be placed upon a parser in a biological NLP scenario. This chapter describes a benchmarking experiment similar to the previous chapter, but using dependency graphs and biologically-motivated evaluation criteria defined over dependencies important to the correct interpretation of the biomolecular interactions in the GENIA corpus. The parsers are scored on their ability to correctly generate the grammatical dependencies in each sentence, by comparing the corresponding dependency graphs derived from their output and from the constituent structure of the original treebank. It is a prelude to the following chapter, which presents a biological information extraction framework relying on dependency graphs.

## 3.2 Performance evaluation methodology

Before generating and testing any dependency graphs, I parsed the GENIA treebank with the parsers in the test set as described in the previous chapter. All pre- and post-processing stages were performed as described therein; essentially, the overall experimental protocol is very much like that shown in Figure 2.3. The only major difference, apart from the scoring methods used, was that immediately before comparison and scoring, all treebanks (GENIA and parser output) were passed to the Stanford algorithm for mapping from phrase structure trees to dependency graphs.

### 3.2.1 Building the dependency graphs

I will not discuss this process in detail as it is described thoroughly by de Marneffe *et al.* (2006) and in the documentation for the Stanford NLP toolkit.[2] Briefly, it defines a taxonomy of directed, labeled grammatical relations, from the most general default type, dependent (`dep`), to highly specific types such as nominal passive subject (`nsubjpass`) or phrasal verb particle (`prt`). A list of these types with the full names and labels is provided in Appendix C. Each type has a list of allowable source constituents, target constituents and local tree structures that may hold between source and target; these definitions can include both structural constraints and lexical constraints (e.g. lists of valid words within the constituents). The algorithm attempts to match the patterns against the supplied tree structure of a sentence, from most specific to most general, and when a match is found, a dependency arc is added to the output graph from the head word of the source constituent to the head word of the target constituent. (A head word of a constituent is the word that is central to that constituent's meaning, upon which all the other words within it ultimately depend; e.g. the head of a verb phrase is the verb itself, and the head of a noun phrase is the rightmost noun.) The result of the algorithm is a directed graph for each sentence, with each node representing a word and each arc representing a grammatical dependency of the appropriate type.

The algorithm also provides the facility to 'collapse' graphs into a slightly simplified form, replacing certain words such as prepositions or possessives with dependencies, and optionally adding extra dependencies that make the semantics of each sentence slightly more explicit (at the expense of making the sentence's graph potentially cyclic rather than guaranteed acyclic). When scoring the parsers' overall performance, I used the collapsed versions of the dependency graphs with all additional dependencies added in, as this is the kind of graph one would find most useful in an information extraction project. I also used these versions of the graphs for testing the parsers' abilities to attach verb arguments correctly, because in certain situations a verb's arguments may not be directly attached to the verb in unprocessed graphs. The other specific subtasks however used the unmodified graphs as these allowed a more fine-grained analysis of behaviour.

### 3.2.2 Scoring measures

For this analysis I used the dependency-based scoring measure $F_{dep}$ , which is analogous to the F-measure used in the previous chapter:

$$F_{dep} = \frac{2 \times P_{dep} \times R_{dep}}{P_{dep} + R_{dep}} \tag{3.1}$$

[2] `http://nlp.stanford.edu/software/lex-parser.shtml`

$P_{dep}$ (dependency precision) is the proportion of predicted dependencies which are correct, and $R_{dep}$ (dependency recall) is the proportion of true dependencies which are predicted.

This chapter uses two variants of the $F_{dep}$ measure, using two criteria for identifying matches between the predicted graphs and the gold standard. Under the loose matching criterion, a dependency is considered to match if it connects the correct parent node and child node, regardless of its grammatical type. Under the strict criterion, which is used throughout unless indicated otherwise, the type of the dependency must match as well. For brevity, individual precision and recall scores are not reported in this study, since all parsers scored almost exactly the same for precision and recall on successfully parsed sentences. This suggests that omitted dependencies were usually replaced with a single erroneous arc.

## 3.3 Results and discussion

For each parser, I calculated overall $F_{dep}$ scores using several different criteria, for comparison with the overall scores in Chapter 2. After that, the two highest-scoring parsers were subjected to a battery of tests designed to analyse their performance on a variety of biologically- and linguistically-important tasks.

### 3.3.1 Overall performance comparison

The raw $F_{dep}$ scores for each parser on GENIA are given in Table 3.1. Column 1 uses the strict matching criterion, and as in Chapter 2, these scores are summed over the entire corpus rather than averaged per sentence. There is a much clearer separation between the high-scoring parsers here—the Charniak-Lease parser and the two versions of the Bikel parser—than when using lineage-based or constituent-based scoring measures (see Table 2.2). This suggests that these parsers suffered more 'errors of convention' (rather than semantically-important errors) than the other parsers.

Note that the $F_{dep}$ scores given in Table 3.1 Column 1 use the strict criterion for matching dependencies, where a match is only recorded if an arc with the same parent node, child node and label (dependency type) exists. This is important as the type of a dependency can be crucial for correct interpretation, discriminating for example between the subject and direct object of a verb. However, many assessments of dependency parsers use a weaker matching criterion which disregards the dependency type, and thus only takes into account the topology of the graph and not the arc labels.

For comparison purposes, the mean scores using this weaker untyped criterion are given in Column 3 (see also the Related Work section). Note that the grouping of the parsers is similar, with the Bikel and Charniak-Lease parsers ahead with very close scores. However the vanilla Charniak parser sits between the front-runners and the rest

| | $F_{dep}$ scores on GENIA (1,757 sentences) | | |
|---|---|---|---|
| Parser | strict criterion | strict, ignoring failures | loose criterion |
| Bikel 0.9.8 | 76.7 | 76.7 | **81.2** |
| Bikel 0.9.9c | 76.5 | 76.9 | 81.0 |
| Charniak | 68.7 | 68.6 | 78.1 |
| Charniak-Lease | **77.1** | **77.1** | 81.1 |
| Collins 1 | 68.0 | 69.4 | 72.6 |
| Collins 2 | 68.3 | 70.9 | 72.7 |
| Collins 3 | 67.1 | 71.1 | 71.5 |
| Stanford-lex | 66.9 | 66.9 | 72.8 |
| Stanford-unlex | 68.5 | 68.5 | 74.0 |

Table 3.1: Overall performance scores on the GENIA treebank using dependency graph comparison.

of the pack, doing much better if dependency types are ignored. All remaining discussion in this paper refers to scores using the strict matching criterion unless otherwise specified.

As in the previous chapter, the overall effectiveness scores for some of the parsers are distorted by the fact that they encountered sentences which could not be parsed at all (see Table 2.2). It is useful to separate out the effects on the mean scores of complete parse failures as opposed to individual errors in successfully-parsed sentences. The $F_{dep}$ scores in Table 3.1 Column 2 show the mean effectiveness for each parser averaged *only* over those sentences which resulted in a successful parse. This reinforces the lesson from the previous chapter that the Collins parser's performance is somewhat hampered by parse failures, but even discounting these failures it is under-performing compared to the Bikel and Charniak-Lease parsers.

The highest-scoring parsers overall, the Charniak-Lease parser and the Bikel parser, achieved very similar scores. Therefore, I subjected these two parsers to a series of tests designed to determine where the strengths and weaknesses of each lay when assessed on tasks important to biological language processing applications. I used the older version of the Bikel parser (0.9.8) as it failed on only one sentence, as opposed to two for version 0.9.9c.

### 3.3.2 Prepositional phrase attachment

One problem that is frequently cited as hard for parsers is the correct attachment of prepositional phrases—modifiers attached to nouns or verbs that convey additional information regarding time, duration, location, manner, cause and so on. It is important to correctly attach such modifiers as errors can alter the meaning of a sentence con-

| $F_{dep}$ scores for prepositions | | |
|---|---|---|
| Parser | Modified phrase | Modifying phrase |
| Bikel 0.9.8 | 78.7 | **91.6** |
| Charniak-Lease | **80.2** | 90.1 |

Table 3.2: Parser effectiveness for the task of prepositional attachment.

siderably. For example, consider the phrase *Induction of NF-KB during monocyte differentiation by HIV type 1 infection*. Is it the induction (correct) or the differentiation (incorrect) which is caused by the infection? Furthermore, the targets of many biological interactions are expressed in prepositional phrases, e.g. X *binds* ***to*** **Y**—the bold section is a prepositional phrase. However this problem is non-trivial because correct attachment relies on the use of background knowledge (for humans), or an approximation of background knowledge based on frequencies of particular words in particular positions in the training corpus (for parsers). These frequencies are often sparse, and for previously unseen words (e.g. many of the technical terms in biology) they will be missing altogether.

To assess the potential impact of this phenomenon, I tested the two best parsers on their ability to correctly generate dependencies between prepositions and both the head words of the phrases they modify and the head words of the modifying phrases, by calculating $F_{dep}$ scores over just these arcs. (The Bikel parser was not penalized for missing dependencies in the one sentence it failed to parse at all, in any of these tasks.) For example, in the phrase *inducing NF-KB expression in the nuclei*, the modifying phrase of the preposition *in* is *the nuclei*—*nuclei* being the head of this phrase—and the modified word is *inducing*. The results are given in Table 3.2. Surprisingly, both parsers scored slightly higher on the harder portion of this task (attaching prepositions to the appropriate modified words) than they did across all dependency types, where both achieved an $F_{dep}$ of around 77 as shown in Table 3.1. On the easier portion of this task (attaching prepositions to the appropriate modifying words), both scored considerably higher. Their success on attaching prepositions to the words they modify ran contrary to my expectations, and indicates that the conventional 'folk wisdom' that prepositional phrase attachment is a particularly hard task is not necessarily true within the constrained environment of biological texts.

### 3.3.3 Reconstructing coordinating conjunctions

Another syntactic phenomenon that is problematic for similar reasons is coordinating conjunction—the joining on an equal footing of two equivalent grammatical units (e.g. two noun phrases) by a conjunction such as *and* or *or*. Since the scope of the conjunction relies on extra-linguistic knowledge or assumptions, there are often several

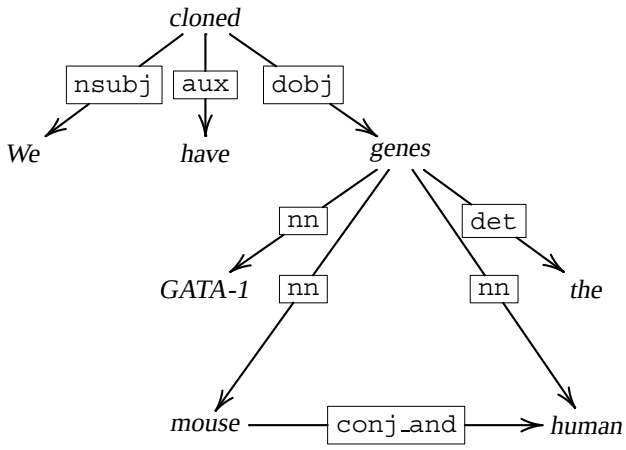


Figure 3.3: This graph is the correct interpretation of the sentence *we have cloned the mouse and human GATA-1 genes*. The *mouse* and *human* conjuncts are related symmetrically to the *genes*—the directionality of the dependency between them is just an artifact of word order. Compare Figure 3.4.

equally grammatical but semantically quite different readings available. An example of this is given in Figure 3.3 and Figure 3.4. The correct reading (Figure 3.3) refers to the cloning of GATA-1 genes from mice and from humans—*mouse* and *human* are both attached directly to *genes*. An alternative, grammatical, yet incorrect reading is shown in Figure 3.4, where *human* is attached to *genes*, but *mouse* is attached directly to *cloned*, implying that some human genes and a whole mouse were cloned.

To measure the ability of the parsers to make the right choices in these situations, I recalculated the $F_{dep}$ score over only those subgraphs (in the parse or the gold standard) whose root words are at either end of a conjunction dependency. For example, if one were comparing the incorrect parse in Figure 3.4 to the sentence in Figure 3.3, the gold standard would consist of all the dependencies from Figure 3.3 that go to or from the words *mouse* and *human*, as these are connected by the `conj_and` conjunction. The test set would consist of all the dependencies in Figure 3.4 that connect to any of the words *the*, *mouse*, *human*, *GATA-1* and *genes*, as the conjunction joins the words *mouse* and *genes* upon which the words *the*, *human* and *GATA-1* depend. True and false positive counts, and thus precision, recall and $F_{dep}$ can then be calculated over just these dependencies. It would not be sufficient to compare the conjunction dependency alone between the two graphs as this would not measure the extent of this initial error's consequences. In some circumstances, such as nested coordinations involving complex multiword phrases—e.g. *the octamer site and the Y, X1 and X2 boxes*—these consequences can be particularly far-reaching. Both parsers' scores on this task (Table 3.3) were slightly lower than their averages of around 77 across all

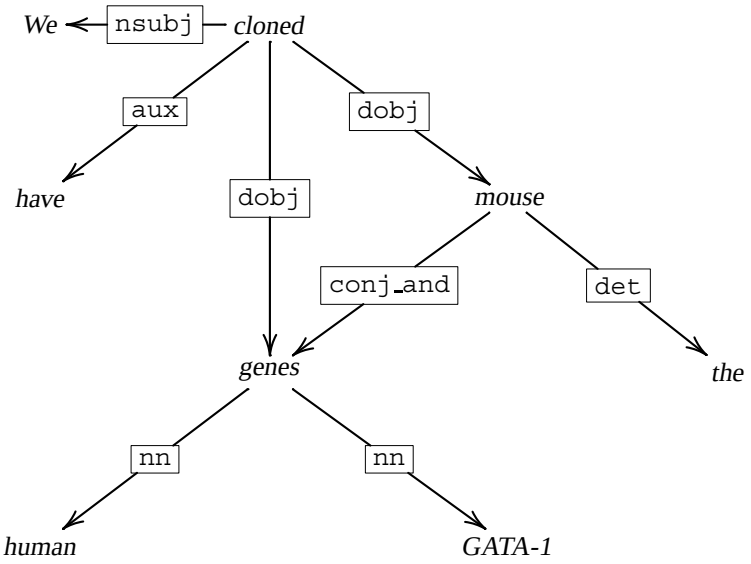


Figure 3.4: This graph demonstrates a mis-parse of the sentence in Figure 3.3. It implies that some human genes, and a mouse, have been cloned.

| $F_{dep}$ scores for coordinating conjunctions | |
|---|---|
| Parser | $F_{dep}$ |
| Bikel 0.9.8 | **76.0** |
| Charniak-Lease | 75.4 |

Table 3.3: Parser effectiveness for the task of reconstructing subgraphs covered by coordination structures.

dependency types, but not spectacularly lower.

### 3.3.4 Detecting negation

Reliably distinguishing between positive and negative assertions and determining the scope of negation markers are perennial difficulties in NLP, and have been well studied in the medical informatics context (Goldin and Chapman, 2003; Mutalik *et al.*, 2001). It is not uncommon in information extraction projects to skip sentences containing negation words (Domedel-Puig and Wernisch, 2005), but *not* appears in 10% of the sentences in my test corpus, and this figure does not count all the other ways of negating a statement in English. Thus a case should be made for attempting to tackle the problem in a more methodical way. In order to gain some initial insight into whether dependency parses might be of use here, I calculated the $F_{dep}$ score for all dependency arcs beginning or ending at any of these words: *not*, *n't*, *no*, *none*, *negative*, *without*,

| $F_{dep}$ scores for negations | |
|---|---|
| Parser | $F_{dep}$ |
| Bikel 0.9.8 | 70.6 |
| Charniak-Lease | **81.0** |

Table 3.4: Parser effectiveness for the task of attaching common negation words.

*absence*, *cannot*, *fail*, *failure*, *never*, *without*, *unlikely*, *exclude*, *disprove*, *insignificant*. The results (Table 3.4) are encouraging and the use of dependency graphs in resolving negations warrants further investigation. The difference between these two parsers is much clearer in this task than in any of the others, and demonstrates that the Charniak-Lease parser may be particularly suited to tackling this problem, as it scores higher than its all-dependencies average while the Bikel parser scores considerably lower.

### 3.3.5 Verb argument assignment

Although there are uncountably many ways to express most logical predicates in natural language, molecular biology texts and abstracts in particular are generally rather constrained and essentially designed for the efficient reporting of sequences of facts, observations and inferences. As a result, much of the important semantic content in this genre is encoded in the form of declarative statements, where a main verb expresses a single predicate more or less exactly, and its syntactic arguments (the subject, direct object and any indirect or prepositional objects) correspond to the entities over which the predicate holds. This being the case, it is important that the arguments of content-bearing verbs are assigned correctly. Failing to recover the subject or object of a verb will render it less useful—not completely useless, however, since we may like to know e.g. that ***X*** *inhibits B cell Ig secretion* even if we do not yet know what ***X*** is. Furthermore, most important predicates are very much directional, meaning that a confusion between subject and object at the level of syntax will lead to a disastrous reversal of the roles of agent and target at the level of semantics. Put more simply, ***X*** *phosphorylates* ***Y*** and ***Y*** *phosphorylates* ***X*** are very different statements. Of course, this is not unique to biological texts but is true of most semantic interpretation tasks.

In order to detect any latent parsing problems that might hinder this process, I chose one of the most common biological predicate verb in the corpus (*induce* in any of its forms) and divided the dependency types that can hold between it and its (non-prepositional) arguments into two sets: those which one would expect to find linking it to its agent, and those which one would expect to find linking it to its target. For example, in the statement *Cortivazol significantly induced GR mRNA*, *Cortivazol* is the agent and *GR mRNA* is the target. I then calculated an $F_{dep}$ score for each parser

| | Parser performance for attachment of arguments to *induce* | | | | | |
|---|---|---|---|---|---|---|
| Parser | $F_{dep}$ | FP | FN | Mismatches | Nonmatches | Missing |
| Bikel 0.9.8 | 97.5 | 4 | 3 | 1 | 0 | 2 |
| Charniak-Lease | **98.3** | 4 | 1 | 1 | 0 | 0 |

Table 3.5: Parser effectiveness at assigning the arguments of the verb *induce* into the correct category (agent or target). FP = false positives, FN = false negatives, Mismatches = false negatives where dependencies from the other set were suggested instead, non-matches = false negatives where dependencies from neither set were offered instead, missing = false negatives where no substitute dependency was found.

over these dependencies only, counting as a match those which connect the correct two nodes and which are from the correct set, even if the exact dependency type is different. For example, if the gold standard contained a nominal subject dependency (`nsubj`) between two nodes, and the parse contained a clausal subject dependency (`csubj`) in the same place, this would count as a match since both are in the agent dependencies set. (A nominal subject is a noun phrase which is the subject of a verb, whereas a clausal subject is an entire clause in the same role, as in ***whether this is true** remains to be seen*.)

The resulting $F_{dep}$ scores are given in Table 3.5, together with a breakdown of false negatives (recall errors): the numbers of mismatches (substitutions for dependencies from the other set), non-matches (substitutions for dependencies from neither set), and completely missing dependencies. The scores for both parsers are very high, with the Charniak-Lease parser only mis-categorizing one out of 145 instances of arguments for *induce* (putting it in the wrong category) and proposing only three other erroneous arguments for this verb in the whole corpus. These results bode well for the semantic accuracy of information extraction systems based on these principles.

### 3.3.6 Error analysis

The importance of correct POS tagging for accurate parsing was mentioned in Chapter 2; the effect of POS errors can be inferred, at quite a gross level, from the difference in performance between the Charniak-Lease parser, and the other—newer—version of the Charniak parser which does not have the benefit of biomedical-domain POS tagging. To measure the consequences of POS errors in the context of dependency graphs, I counted the number of false negatives (recall errors) in the outputs of the two leading parsers where either one or both of the words which should have been joined by the missing dependency were incorrectly tagged. (Remember that, since the strict matching criterion is being applied here, a recall error means that a dependency *of a specific type* is missing; it will usually be the case that another dependency of a different type

| | Reasons for recall errors | | | |
|---|---|---|---|---|
| Parser | 1 bad tag | 2 bad tags | 1 missing node | 2 missing nodes |
| Bikel 0.9.8 | 28.7% | 3.42% | 0.42% | 0.00% |
| Charniak-Lease | 20.6% | 2.00% | 0.39% | 0.00% |

Table 3.6: Breakdown of missing dependencies caused by POS tagging errors and missing nodes.

has been substituted.)

Also, in a very small minority of cases, it is possible for nodes to be present in a dependency graph from the gold standard, but actually missing from the same graph in a parser's output, or *vice versa*. This can occur because punctuation symbols are not always retained as nodes in the graph in the same way that words are. If a word is mistakenly treated as a discardable punctuation symbol, it will be omitted from the dependency graph. This can result from a POS tagging error, an error in the Stanford algorithm or a mismatch between the conventions used by a parser or the gold standard and those used by the Stanford algorithm's developers. Conversely, if a punctuation symbol is treated as a word for the same reasons, it may be present as a node in its own right in the resulting graph even if it would otherwise have been suppressed. Therefore, I also counted the number of missing dependencies in each parser's output where one or both of the nodes that the dependency should have connected were also missing. The results of both of these tests are given in Table 3.6. The results—one in five missing dependencies being associated with at least one POS error for the Charniak-Lease parser, and almost one in three for the Bikel parser—should provide all the more motivation for the development and refinement of biological POS tagging software.

In addition, I counted the missing dependencies for each parser by type, in order to get an idea of which types were the most problematic. The results (Table 3.7) are rather interesting. The same five types (out of roughly 50) account for the majority of errors in both cases, although there is some difference in the relative proportions. One in five missing dependencies are of the generic dependent (`dep`) type, which the Stanford algorithm produces when it cannot match a syntactic construction in a phrase structure tree to a more specific type of dependency. The presence of large numbers of `dep` arcs in the graphs of the gold standard corpus indicates that the GENIA annotators are using syntactic constructions that are unfamiliar to the Stanford algorithm. On closer inspection, it became apparent that around one fifth of the `dep` arcs missed by each parser had been substituted for more specific dependencies joining the same words; it is impossible to judge by comparison to GENIA whether the types of these dependencies are truly correct or not.

| Recall errors by dependency type | | | |
|---|---|---|---|
| Bikel 0.9.8 | | Charniak-Lease | |
| Dependent | 20.4% | Dependent | 20.8% |
| Noun compound modifier | 11.7% | Prepositional modifier | 12.2% |
| Prepositional modifier | 11.5% | Punctuation | 11.6% |
| Punctuation | 10.6% | Noun compound modifier | 8.1% |
| Conjunction | 7.2% | Conjunction | 7.9% |

Table 3.7: Breakdown of missing dependencies by dependency type (top five types only).

## 3.4 Concluding remarks

I have presented a method for evaluating treebank parsers based on dependency graphs that is particularly suitable for analysing their capabilities with respect to semantically-important tasks crucial to biological information extraction systems. Applying this method to various versions of four popular, open-source parsers that have been deployed in the bioinformatics domain has produced some interesting and occasionally surprising results relevant to previous and future NLP projects in this domain.

In terms of overall parse accuracy, the Charniak-Lease parser—a version of the venerable Charniak parser enhanced with access to a biomedical vocabulary for POS-tagging purposes—and version 0.9.8 of the Bikel parser achieved joint highest results. Both parsers relied on good POS tagging to achieve their scores, with large proportions of the dependency recall errors being attributable to POS errors. An interesting comparison can be drawn here between the Charniak-Lease parser, for which just over 20% of the missing dependencies connect to at least one incorrectly-tagged word, and the original Charniak parser, which uses a POS-tagging component trained on newspaper English, and for which almost 60% of the recall errors relate to at least one incorrectly-tagged word.

Both parsers performed well on tasks simulating the semantic requirements of a real-world NLP project based on dependency graph analysis, and achieved mostly similar scores. The reconstruction of coordinating conjunctions (e.g. *and*/*or* constructs) was slightly more difficult than average for each parser, and the correct attachment of negation words (e.g. *not* or *without*) proved problematic for the Bikel parser, although the Charniak-Lease parser was more successful on this task. Both parsers identified the arguments of the verb *induce* almost perfectly when I relaxed the matching criterion to allow substitutions between agent-argument dependencies (e.g. `nsubj` and `csubj`) and between target-argument dependencies (e.g. `dobj` and `iobj`).

While the overall results of this experiment (Charniak-Lease and Bikel coming out on top) are not surprising given the results presented in Chapter 2, what is striking is that the separation between the top parsers and the rest is much clearer when analysing

their behaviour by dependency rather than by constituent- or lineage-based methods. The best explanation for this, given that the semantics of a sentence are much 'closer to the surface' when using dependency graphs, is that many of the apparent parse errors which dragged these parsers' scores down in the original experiment are in fact illusory differences of convention that disappear when the transition from tree to graph is made. It is also interesting that although the Stanford algorithm was developed in parallel with the Stanford parser, the broad compatibility achieved by using standard PTB-style constituent trees means that it is not only happy to process the output of any of these other parsers, but in fact does a better job of producing dependency graphs when presented with the output of a parser better suited to the biological domain. This highlights the importance of using common standards for data exchange rather than in-house formats.

### 3.4.1 Benefits of dependency graphs

Given that none of the parsers in this evaluation use dependency grammars natively, one might ask two questions. Firstly, what are the practical advantages of translating the output of treebank-style constituent parsers into dependency graphs? And secondly, how do the graphs thus generated compare to the raw output of dependency parsers on biological texts? I will address the latter question below in the Related Work section. In answer to the former question, the benefits are manifold and apply to both the evaluation process and the engineering of NLP applications.

I hope that the semantic evaluation tasks presented in this chapter demonstrate the ease by which application-specific benchmarks can be designed and applied with reference to dependency graphs. Granted, one could conceive of similar phrase-structure tree-based algorithms to test the positioning of, say, negation words with respect to the words they modify, but these would require the comparison of two subtrees and would therefore require much more coding and processing than their dependency equivalents. Indeed, since several subtrees can result in the same grammatical relation, one would have to manually account for a degree of allowable variation. Furthermore, some application-specific tests—such as the analysis of arguments for the verb 'induce' in this experiment—would be impossible using raw constituent trees. This kind of information is not explicitly represented in constituent trees, but rather is implicit (albeit buried rather deeply) in the phrase structures and the rules of English, and to test such relations from trees alone requires the design and implementation of mapping rules that would essentially result in local dependency structures anyway.

### 3.4.2 Related work

The inspiration for this experiment, and its predecessor in Chapter 2, came from the observation that constituent parsers are beginning to appear in bioinformatics papers

on a wide variety of topics, but without any analysis of how well they perform as isolated components in broader projects. For example, the Bikel parser has been used to produce rough treebanks for human correction in a biological treebanking initiative (Bies *et al.*, 2005). Subtrees from the Collins parser have been used as features in a protein interaction extractor (Xiao *et al.*, 2005) and in a classifier for semantic relations between biomedical phrases (Rosario and Hearst, 2004). The Charniak parser has been employed to assist in the re-ranking of search results in a search engine for genomics documents (Shi *et al.*, 2005) and in the acquisition of causal chains from texts about protein interactions (Sanchez and Poesio, 2005). The Stanford parser has been used to provide syntactic clues for identifying key clinical terms in the medical domain (Huang *et al.*, 2005) and gene and protein names in the biological domain (Finkel *et al.*, 2004).

A thorough analysis of the effectiveness of these parsers in this domain is vital to identifying the source of errors, to developing workarounds for these errors, and indeed to selecting the right parser to begin with. Since the first version of Chapter 2 was published (Clegg and Shepherd, 2005), a few other papers on the benchmarking of parsers on biological texts have appeared. Lease and Charniak (2005), in introducing the modified version of the Charniak parser that performed so well here, present some comparative scores for various versions of the parser on both the GENIA treebank and the Penn Treebank, but they use constituent-based precision, recall and F-measure ($F_{const}$) and therefore implicitly suffer from the inability of such measures to distinguish between differences of meaning and convention discussed in this paper. Their paper is the most similar to Chapter 2 currently in circulation.

Grover *et al.* (2005) present several experiments on parsing MEDLINE abstracts with three hand-crafted grammars. First they show that although the low-coverage but high-accuracy ANLT parser (Grover *et al.*, 1993) can return a successful parse on only 39.5% of the sentences in their 79-sentence test set, 77.2% of those sentences (30.5% overall) were parsed perfectly. This strategy seems somewhat dubious for real-world applications, however, since a parse with a handful of minor errors is surely more desirable in practice than no parse at all. The ANLT parser also returns a set of logical predicates representing the sentence; whether this is more or less useful for application development than a dependency graph remains to be seen. They then present some experiments on using the Cass (Abney, 1996) and TSG (Briscoe and Carroll, 2002) parsers to correctly interpret compound nouns which encode predicate relationships, differentiating for example between 'treatment response' = response TO treatment, and 'aerosol administration' = administration BY aerosol. Their results for this unique investigation are interesting and encouraging, but it is unfortunate that they do not apply the ANLT parser to the compound noun task, and conversely, they do not provide general measures of coverage and accuracy for the Cass and TSG parsers.

Other papers have been published on the behaviour of native dependency parsers on biomedical text. The paper by Pyysalo *et al.* (2006a) is perhaps the closest to the

investigation in this chapter. They compare the free Link Grammar parser (Sleator and Temperley, 1993) to a commercial parser, the Connexor Machinese Syntax parser, [3] both of which have been used in bioinformatics (Ahmed *et al.*, 2005; Franzén *et al.*, 2002). The parsers use different dependency grammars, so the authors prepared a 300-sentence protein-protein interaction corpus with a dual annotation scheme that accommodated the major differences between the two parsers' dependency types. They also disregarded dependency types, as well as directions, as the Link parser's 'links' are not explicitly directional, resulting in an even looser matching criterion than the loose criterion used briefly in this paper.

The Link parser can return multiple parses in ranked order of likelihood, and taking only the first parse for each sentence, it achieved a recall of 72.9%, and parsed 7.0% of sentences perfectly, although the same group shows elsewhere (Tsivtsivadze *et al.*, 2005) that this figure may be raised slightly by using an independently-trained re-ranker. The Connexor parser returns a single parse for each sentence; it scored 80.0% for recall and also achieved 7.0% perfect parses. For comparison, our best parser (Charniak-Lease) achieved an overall recall of 81.0% and parsed an impressive 23.1% of sentences perfectly, even given a slightly stricter dependency matching criterion. The authors also scored the parsers on their ability to return perfect interaction subgraphs—minimal subgraphs joining two protein names and the word or phrase stating their interaction—although we disagree that a *perfect* interaction subgraph is necessarily a pre-requisite for successful retrieval of an actual interaction. (Neither is it sufficient, since a negation word might be outside the interaction subgraph yet still able to completely reverse its meaning.)

Schneider *et al.* (2004b) present results comparable to my results for the Pro3Gres parser (Schneider *et al.*, 2004a) on performing several specific syntactic tasks over a small subset of GENIA. Their general approach is very similar to mine, but they do not provide performance indicators over all dependency types, and they chunk multi-word terms into single elements before parsing. They report $F_{dep}$ scores of 88.5 and 92.0 for identifying the subjects and objects of verbs respectively, although it is not clear whether or not these relation types are defined as broadly as the categories used above in the study of the verb 'induce', where the Charniak-Lease parser scored 98.0 and the Bikel parser scored 97.0, averaged across both agent and target relations. They also report $F_{dep}$ scores of 83.5 and 83.0 for prepositional modification of nouns and verbs respectively, which are slightly better than my best parsers' scores on this task; their system contains a module specifically written to correct ambiguous prepositional phrase attachments. (Note that the $F_{dep}$ scores reported here are calculated from the individual precision and recall scores given in the original Schneider *et al.* (2004b) paper.)

One factor common to Pyysalo *et al.* (2006a) and Schneider *et al.* (2004b) is the

[3] `http://www.connexor.com/`

small size of the evaluation datasets (300 and 100 sentences respectively) since both required the manual preparation of a dependency corpus tailored to the parsers under inspection. Another advantage of producing dependency parses from constituent parses is that we can make use of the larger and rapidly-growing body of treebank-annotated biological text. Since this project was begun, the GENIA treebank has grown from 200 to 500 MEDLINE abstracts, and the BioIE project has released 642 abstracts annotated in a similar format. The Stanford algorithm provides a *de facto* standard for comparing a variety of constituent parsers and treebanks at the dependency level; if the dependency parser community were to adopt the same set of grammatical relations as standard, then native dependency parsers could be compared to constituent parsers and to biological treebanks fairly and transparently. In response to this suggestion, Pyysalo *et al.* (2007b) have adapted their BioInfer corpus (Pyysalo *et al.*, 2007a) to the Stanford dependency scheme, thus creating the first hand-prepared Stanford evaluation corpus. The corpus was converted automatically from the original Link Grammar version of BioInfer using some highly accurate translation rules, but then manually corrected according to the consensus of two annotators working in parallel. The authors have made both the conversion program and the corrected corpus available on their website [4] in order to encourage the adoption of the Stanford format in the biomedical NLP community. Section 5.1.2 contains some more discussion of BioInfer.

The use of dependency graph analysis as an evaluation tool is not a new idea, having been discussed by the NLP community for several years, but to the best of my knowledge the application of such methods to specific problem domains like bioinformatics is a recent development. An early proposal along these lines (Lin, 1995) also acknowledged that inconsequential differences exist between different dependency representations of the same text, and included some suggested ways to exclude these phenomena, although without a comprehensive treatment. While such differences do exist, I believe that dependency graphs are much less prone to this problem than constituent trees, given a consistent and logical set of dependency types. The same paper also discussed the mapping of constituent trees to dependency graphs via phrasal heads; the Stanford toolkit relies on a more sophisticated version of this process. Its author later used this approach to evaluate his own MINIPAR dependency parser (Lin, 2003).

Later, the EAGLE and SPARKLE projects used hierarchically-classified grammatical relations, which are comparable to the Stanford toolkit's dependency types, to evaluate parsers in several languages (Carroll *et al.*, 1998, 1999; Briscoe *et al.*, 2002). Similar scoring measures have been proposed for partial parsers (Srinivas *et al.*, 1996; Kübler and Telljohann, 2002)—those parsers which only return complete syntactic analyses of parts of each sentence. However, despite the well-known issues with constituent-based methods and the wealth of research on alternatives such as these, constituent precision and recall (along with supplementary information like number

[4] `http://www.it.utu.fi/BioInfer/`

of crossing brackets per sentence) remain the *de facto* standard for reporting parser accuracy.

Of course, while an essentially syntactic evaluation of parser performance may be invaluable in choosing a parser to focus one's research efforts on, and may suggest ways in which errors can be identified or worked around, the true test of a parser's usefulness is its ability to aid in the accurate extraction of biologically-relevant information. The following chapter provides such a test by putting the Charniak-Lease parser to work in an information extraction framework.

# Chapter 4

# Information extraction from dependency graphs

One of the primary goals of biological NLP, and a pre-requisite for practical text mining, is automatic information extraction. As discussed in Chapter 1, this refers to the process of translating a human-readable corpus into structured data for visualization, querying and mining, or indeed any other computational process. Although a rich syntactic representation of a sentence is not necessarily required for IE, this chapter presents an experiment to test the hypothesis that using dependency graphs as an intermediate stage can facilitate the extraction of biological interactions from parse trees and provide a powerful and flexible pipeline from raw text to semantic relations.

The experiment was initially performed in three parts, with three different solutions to the same problem. Each one was developed after the previous had been evaluated and all new results had been analysed, so I will present them chronologically, after describing the LLL Challenge (Learning Language in Logic) which provided the opportunity.

## 4.1 Background

The GENIA corpus has been widely used in biological NLP evaluations because it contains various different kinds of linguistic annotation—parts of speech, entity names and types, and constituent structure. However, it is not immediately useful in testing IE algorithms as there is no annotation of the relationships between the entities in each sentence, and this is what is usually required of IE applications in this domain. In order to provide a competitive benchmark for such systems, the LLL Challenge at the 22nd International Conference on Machine Learning (Nédellec, 2005) provided participating groups with a set of training sentences drawn from MEDLINE abstracts

| Subtask difficulty matrix for LLL Challenge | | | |
|---|---|---|---|
| | Easy sentences | Hard sentences | Both |
| Linguistic annotation | 4, 5, 6, (6) | 5 | 3, 6 |
| No linguistic annotation | 1, 2, 5, (6) | 5 | (6) |

Table 4.1: This table shows which groups (by number, see Table 4.2) submitted official runs to which versions of the challenge. The top-left cell represents the easiest, the bottom-right the hardest. (6) indicates Group 6's unofficial results that were reported nonetheless.

on transcription in *Bacillus subtilis*. Each sentence was annotated by the organizers with a set of ordered tuples representing the pairs of interacting genes and proteins described therein; the causal agent and target of each interaction was marked as such. A test set was also provided, with sentences on the same subject, but with the annotations withheld.

The goal of the challenge was to retrieve the interactions described in the test set, using an algorithm trained on the training set. For example, given the sentence *Both SigK and GerE were essential for ykvP expression*, the correct answer would consist of two tuples, where the first entity in each tuple is the agent: $SigK \rightarrow ykvP$ and $GerE \rightarrow ykvP$. The interactions in the training set fell into three categories: genetic regulatory relationships where no physical mechanism is specified (68%), direct physical interactions such as promoter binding (25%), and relationships implied by membership of a regulon (7%). According to the task guidelines, these relative proportions were similar in the test set, but 50% of the test set sentences had no interactions in (unlike the training set where every sentence had at least one).

### 4.1.1 Organization of the LLL Challenge

The training data was split into two sets. Each sentence in the harder set of sentences contained more difficult linguistic phenomena like coreference, where for example a gene might be referred to indirectly elsewhere in the sentence by a pronoun like *it*. The easier set did not contain such constructions. The test data contained both types of sentence mixed together, but each group could declare whether they had trained on the easier set, the harder set or both together, and be assessed accordingly on only the corresponding sentences from the test set. The hardest option was both together, since no particular optimization for either sort was possible. This most closely reflects the situation of a real application, as there is no oracular way to determine the composition of an incoming sentence without actually analysing it.

As well as the basic annotation for the training set containing interaction tuples, the organizers provided a more sophisticated annotation file for each dataset (training *and* test). These contained syntactic structure information for each sentence, provided by

| Groups in the LLL Challenge | | |
|---|---|---|
| Number | Affiliation | Reference |
| 1 | Humboldt U./EBI | Hakenberg *et al.* (2005) |
| 2 | U. Sheffield | Greenwood *et al.* (2005) |
| 3 | U. Amsterdam | Katrenko *et al.* (2005) |
| 4 | U. Brno | Popelínský and Blaťák (2005) |
| 5 | U. Madison | Goadrich *et al.* (2005) |
| 6 | U. Edinburgh | Riedel and Klein (2005) |

Table 4.2: The groups competing in the LLL Challenge; for brevity I will refer to them by their numbers throughout.

the Link Grammar parser (Sleator and Temperley, 1993), but checked, simplified and corrected by hand. They also provided lemmas for each word—a lemma is the canonical form of a word, e.g. *regulate* for *regulates* or *regulated*, or *mouse* for *mice*. These were also manually corrected. It was left up to each group to use or ignore this information; obviously ignoring it leads to a more realistic test since in a live system the only linguistic data comes from one's own noisy analyses. With three choices of sentence set, and the option to use or ignore the linguistic annotations, the competition actually decomposed into six distinct subtasks with results that were not directly comparable (Table 4.1). Each group was allowed to enter multiple runs if they desired, although only the highest in each subtask would be counted as official. However Group 6 also recorded several unofficial runs in their report, fortuitously, since (as we shall see) the unofficial scores provide an extra angle of analysis. This is very helpful, since the profusion of subtasks, and the reluctance of many of the teams to tackle the harder ones, would hamper attempts at in-depth analysis otherwise. Also, if it wasn't for Group 6's late entry, there would have been no result at all on the hardest task and thus nothing to compare my own algorithms to.

### 4.1.2 Results of the LLL Challenge

The contestants in the challenge were scored on precision ($P$, the proportion of predicted interactions that were correct), recall ($R$, the portion of genuine interactions that were predicted) and F-measure ($F$, the harmonic mean of the previous two scores). These measures are analogous to those used in the previous two chapters, but are now measuring semantic rather than syntactic accuracy. I have summarized the scores for the LLL contestants in Table 4.3, putting them in descending order of F-measure. The results are rather striking. The top five runs belong to Group 6, and although the winning run used the easy sentences with linguistic annotation, second and third places go to runs that processed both kinds. The only other group that attempted this, Group 3,

| Results of the LLL Challenge | | | | | |
|---|---|---|---|---|---|
| Sentences | Annotations | Group/Run | $P$ | $R$ | $F$ |
| easy | yes | 6a | 65.0 | 72.2 | **68.4** |
| both | yes | 6b | 63.2 | 66.2 | 64.7 |
| both | yes | 6c | 55.6 | 53.0 | 54.3 |
| easy | no | 6d | **68.5** | 44.4 | 53.9 |
| easy | yes | 6e | 60.9 | 46.2 | 52.6 |
| easy | no | 1 | 50.0 | 53.8 | 51.8 |
| easy | yes | 4 | 37.9 | 55.5 | 45.1 |
| both | no | 6f | 50.0 | 33.7 | 40.2 |
| easy | no | 5a | 25.0 | 81.4 | 38.2 |
| easy | yes | 5b | 20.5 | 90.7 | 33.4 |
| both | yes | 3 | 51.8 | 16.8 | 25.4 |
| hard | yes | 5c | 14.0 | 93.1 | 24.4 |
| hard | no | 5d | 14.0 | 82.7 | 24.0 |
| easy | no | 2 | 10.6 | **98.1** | 19.1 |

Table 4.3: The results of the LLL Challenge, in descending order of F-measure.

are ranked 11th out of 14, with comparable precision but half the recall of the lowest-ranking Group 6 run—despite the fact that Group 3 used the additional annotations for the mixed sentence set, but the Group 6 bottom run didn't. The approaches and results of each of the teams are discussed below.

**Group 1**

Group 1 began by POS-tagging the data (training and test), and then tagging gene/protein names and interaction keywords (*activation*, *inhibits*) by dictionary matching. This resulted in a sequence of tags representing each sentence. They tried two methods for learning patterns characteristic of genuine relationships from the training data and matching these patterns to unseen sentences. The first used multiple sequence alignment techniques with weighted substitution matrices for the tags, and performed poorly. The second used a genetic algorithm to evolve finite state automata encoding sequences of tags, using their ability to match to the training data as the fitness function. The best of the second method's runs was submitted (see Table 4.3).

Neither approach used any syntactic information at all, treating sentences and phrases as simply linear sequences of tags. They did not attempt to tackle the hard sentences with coreference and other difficult constructions. The authors suggested that more training patterns would have helped with coverage, and syntactic information could have improved their system's ability to get the agent and target the correct way round—

this was a particular problem.

### Group 2

Group 2 parsed the training set with MINIPAR (Lin, 2003), a dependency parser which produces grammatically-labeled graphs similar to Stanford's (see Chapter 3), and extracted patterns (graph fragments) containing pairs of gene/protein entities. (The manual syntactic annotations supplied by the organizers were not used.) After parsing, they used an iterative learning algorithm, starting with a manually-selected set of eight seed patterns, to select an optimal subset of patterns which discriminate between (ordered) pairs of entities that do interact and pairs that don't. This worked by iteratively adding new patterns from the candidate set by selecting those most similar to the patterns in the working set which perform well on the training data. After parsing the test data, they used the patterns derived in the training stage to classify each pair of entities as interacting or not. Unfortunately, this was not successful, despite attempting only the easier sentences. The best attempt scored $P$ = 21.6 and $R$ = 14.8 for an overall $F$ = 17.5.

The authors concluded that the limited number of training sentences adversely affected their algorithm, noticing that none of the original eight seed patterns (chosen for their apparent discriminatory power) actually matched any interactions in the test set. Also, the patterns were not generalized from the training examples, but lifted node-for-node and arc-for-arc, although words were replaced with their lemmas (canonical forms), e.g. *repress* for *represses*, *repressed* etc. They suggested that an approach allowing substitutions between words with similar meaning would do better. They also speculated that MINIPAR errors may have been a contributory factor, but made no attempts to quantify this effect.

Group 2 also generated a baseline set of predictions consisting of two interactions for every pair of genes/proteins that occur in the same sentence (one for each agent-target direction). This scored a higher $F$ than their other method, on account of its near-perfect recall, and became their official submitted result (see Table 4.3). It did not score 100% recall because a small number of sentences in LLL describe more than one interaction between the same agent and target; the LLL scoring criteria expect these to be reported individually.

### Group 3

Group 3's approach was based on generating sets of logical predicates in Prolog which described the words in the training set, and the relations between them, at the lexical, syntactic and semantic levels. The syntactic information was drawn from the manual Link annotations supplied by the organizers, and the semantic information by reference to a small domain-specific ontology prepared by Group 3 themselves. They then used

a rule induction algorithm to learn a set of rules for distinguishing between interacting and non-interacting pairs of entities, based on unifying parts of the logical structures inferred from the training set.

Their results were somewhat mediocre, although after fixing a bug present in the submitted run, their F-measure rose from the 25.4 shown in Table 4.3 to 31.6. They trained and tested on both the easy and hard sentence sets together, which may have impaired their algorithm's effectiveness, but their paper contains no error analysis or even speculation about causes of error.

**Group 4**

Group 4 took a similar approach to Group 3, experimenting with various rule induction methods to learn discriminatory patterns of rules from logical predicates describing the training data. They only attempted the easy sentences, and used the manual syntactic annotations in the training set, along with automatically-generated POS tags and word-synonym relations. Their submitted run scored $F$ = 45.1, although the best run reported in their paper (using a rule induction algorithm called Aleph) scored $F$ = 48.2.

**Group 5**

Group 5 also used the Aleph algorithm, along with another learning algorithm called Gleaner, to learn combinations of features from the training set which were more likely to be present in genuine interactions than false positives. The features were based on the words themselves, POS tags derived using a standard English tagger, non-nested phrase boundaries from a shallow parser, orthographic and word-frequency features, and the presence of words in an external biomedical ontology. They also incorporated the hand-corrected parses from the organizers in certain runs, although it is not clear from their description that they were aware that this information was manually edited.

They attempted both subsets of sentences, albeit separately, rather than together. Interestingly, their best result overall (5a, on the easy sentences) did not use the manually-corrected syntactic data. The authors did not speculate about the causes of the large drop in precision which caused the run using the Link structures performed worse. They did, however, identify difficulties distinguishing agents from targets as a particular problem, as well as a lack of non-interacting examples in the training data.

**Group 6**

Group 6's approach was based on Markov logic (Domingos *et al.*, 2006), an inductive model based on weighted logical clauses describing features of the training data. These features were extracted from the syntactic and semantic pathways connecting each interacting pair of entities in the training set. The syntactic pathway between a

pair of entities was simply the shortest route between them on the Link structure supplied by the organizers. The semantic pathway was obtained by parsing the sentences with a Combinatory Categorial Grammar (CCG) parser (Clark and Curran, 2004) and then feeding the parser's output into a program called ccg2sem (Bos, 2005) which produced a logical representation of the semantic structure of the sentence. Their hope was that certain normalizing effects of the conversion into logical form (e.g. coreference resolution) would make the Markov logic network's job easier.

The runs using the CCG parser, therefore, did not use any of the Link graph structures (or any of the other hand annotations). I have included Group 6's official late submissions (6a, b, d, f) as well as their official runs (6c, e) because their unofficial run 6f was the only attempt made at the hardest version of the task (all sentences together, no hand annotations) and was therefore my target to beat.

The dominance of the score chart by Group 6 certainly reflects well on the ability of Markov logics to generalize successfully from relatively small amounts of data, compared to the various learning methods used by the other teams. Their top-scoring entry (6a) contains an interesting lesson too. They entered it after the close of the competition in order to test the hypothesis that adding negative examples to the model—syntactic and semantic chunks from the paths between genes that *didn't* interact—would help its discriminatory power. The hypothesis was validated; an earlier run (not shown in table), which used the same clause set as 6a but without the negative clauses, scored 4 points less on precision and an impressive 20 less on recall. It may seem counterintuitive that training on negative examples can boost recall, but apparently they can be "highly confident that a gene pair is 'not a non-interaction' while positive clauses are uncertain." (Riedel and Klein, 2005)

One point must be made, however, in response to these results. While Group 6 did thoroughly outperform the rest of the teams, there is a considerable gulf between 6a, which used the hand-corrected Link parses, and their worst performer (6f) which only used the semantic relations they had parsed and extracted themselves. Much of the middle ground consists of runs that used both (6b, c, e). Expending effort to optimize an algorithm for specific situations makes sense only if those situations are ever likely to occur 'in the wild', but the data that allowed 6a to achieve $F = 68.4$ is not just hand-corrected but simplified compared to an actual Link parse. The Link Grammar uses over a hundred link types[1] whereas the LLL equivalent uses 27.[2] A discussion of the effects of negative clauses on the model driven only by real data would have been a more interesting and realistic evaluation.

[1] `http://www.link.cs.cmu.edu/link/dict/summarize-links.html`

[2] `http://data.jouy.inra.fr/unites/mig/text/LLLChalenge05/doc/Relations_Definitions.pdf`

## 4.2 Algorithm I

The LLL organizers did not release their test set answers after the challenge, so that they could be kept aside for completely blind testing; all evaluation during and after the challenge took place via their online scoring system.[3] Using this opportunity, I decided to build a prototype extraction algorithm based on the Charniak-Lease parser and the Stanford dependency graphs described in the previous chapter, eschewing the use of the linguistic annotations supplied with the corpus and testing on the combined sentence set only. I had no desire to spend any time performance-tuning for a grammar I don't use, and there is no point giving a new technology an easy ride at the proof-of-concept stage. The LLL data is already slightly artificial as there are no relationships between more than two entities (although there are plenty of cases where an entity is involved in more than one) or reciprocal interactions without a clear agent/target distinction. Also, the entity list supplied is exhaustive and unambiguous, meaning the gene and protein names are effectively already marked. However, this is not as big an issue with the quite distinctive nomenclature of *B. subtilis* as it would be with, for example, *D. melanogaster* whose gene names and symbols are notorious for overlapping with English words: *an*, *by*, *for*, *limited*, *reduced*, *similar* (Hirschman *et al.*, 2002). However, practicality invariably demands some constraints, especially for small-scale, short-term initiatives.

The very variable performance of most of the groups, even on the easier subtasks, indicated that the task would not be trivial. One possible reason for the mediocre overall performance during the challenge might be a shortage of training data. Only 77 sentences were supplied for this purpose, and those teams that wanted to tackle the sentence types individually had even less to use for each type (55 easy and 22 hard). Despite this paucity of data, all of the teams based their approaches on the automatic induction of extraction patterns of one kind or another. In order to differentiate my approach from those used in the challenge and reduce the impact of the training data limitations, I decided to start with a simple heuristic based on intuitive assumptions about the task, and use the training data only to fine-tune its behaviour on example sentences. At the core of my approach is a fairly straightforward graph traversal algorithm, but its behaviour is flexible and widely parameterized.

### 4.2.1 Methods

The algorithm was designed to make an initial pass by graph traversal, and then three recovery passes to deal with cases where the initial pass found a likely interaction but failed to assign entities. The aim was to achieve high-precision results with the first pass, then increase recall with each successive pass until the desired balance between

[3] `http://genome.jouy.inra.fr/texte/LLLchallenge/scoringService.php`

recall and precision was reached. As many design decisions as possible were parameterized rather than hard-coded, and the LLL training data was used instead as a tuneset to pick an optimal (or near-optimal) set of parameters to use on the test set. The trade-off between precision and recall is a key concept in information extraction, and many of the algorithm's parameters were designed with this balance in mind. For example, a parameter like **removeNegatives** (see below) would be expected to restrict the number of candidate entity pairs that are marked as interacting and thus tend to increase precision at the risk of reducing recall.

Other parameters were introduced without any firm expectations as to their effect upon precision and recall. All of the passes after the first make use of multi-valued parameters called *selectors* and *restrictors*. A selector enables the algorithm to choose a candidate entity for a role (agent or target) based on its position in the sentence, and a restrictor governs whether this candidate will be admitted or not in the role for which it has been chosen. These are brought into play at various stages when it has been impossible to determine an agent and/or target for an interaction using graph-based methods alone. As the frequencies and locations of such 'difficult' entities in the sentences of the dataset was not known in advance, these decision-making processes were parameterized so that any useful patterns in these phenomena could be determined empirically.

Finally, some constraints were introduced in response to common causes of error that were identified during the early development of the algorithm, for example **banSymmetricalSubgraphs** and **removeNegatives**. These were parameterized rather than hard-coded as it is difficult to predict how they will interact with the other parameters and whether they will work best if applied repeatedly or simply checked at one or two key points in the process. As we will see, the amount of linguistic input into the design of this algorithm was fairly minimal, with the structure of the graphs much more important than the labels on the arcs, and the POS tags ignored completely. Rather, the idea was to get a prototype system working to test the basic principles of the dependency graph framework and to use as a frame of reference for other techniques.

Algorithm I's approach makes extensive use of concept of *interaction subgraphs* (see Figure 4.1 for examples):

> "The interaction subgraph for an interaction between two proteins $A$ and $B$ in a dependency parse $P$ is the minimal connected subgraph of $P$ that contains $A$, $B$, and the word or phrase that states their interaction." (Pyysalo *et al.*, 2006a)

In this task, most of the interacting entities are genes rather than proteins, but the principle is the same, although I did further constrain the definition to include only those where the interaction word is at the root. I constructed a lexicon of interaction words that are likely to indicate that an interaction is being described, based on manual

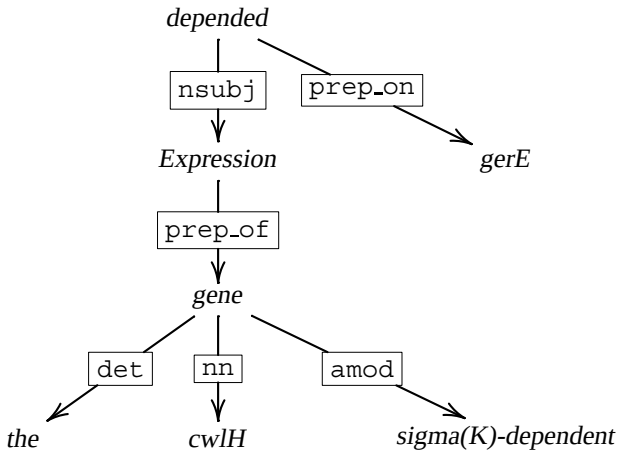


Figure 4.1: This graph of the sentence *Expression of the sigma(K)-dependent cwlH gene depended on gerE* contains several candidate interaction subgraphs, which overlap. An interaction subgraph connects two entities at its leaves via an interaction keyword at its root, and *Expression* and *depended* are both interaction keywords. Therefore, *cwlH:Expression:sigma(K)*, *cwlH:depended:sigma(K)*, *cwlH:depended:gerE* and *sigma(K):depended:gerE* all make up interaction subgraphs, as do their inverses (e.g. *gerE:depended:sigma(K)*) since relationships in this task are directional. Furthermore, unless these cases are removed by filtering rules in post-processing (see below), reflexive subgraphs like *gerE:depended:gerE* are also allowed, giving a total of 13 potential candidate entity pairs.

inspection of the training set and extrapolation based on past experience. This included verbs (in various forms) such as *bind*, *phosphorylates* and *stimulated*, nouns such as *expression*, *target* and *repressor*, and various other keywords such as *dependent*. In addition to the notion of interaction subgraphs, Algorithm I also makes use of the following concepts:

*Interaction triplet*: The interaction triplet for an interaction between two genes/proteins $A$ (agent) and $T$ (target) via an interaction word $I$ is the ordered tuple $\langle A, I, T \rangle$. I use this concept because later passes of the algorithm allow nodes to take part in an interaction even though they do not form a subgraph.

*Dependency path*: A route from node $A$ to node $B$ in a dependency graph of a sentence, consisting of all the arcs and nodes between $A$ and $B$ (inclusive) in order. Note that there may be more than one dependency path from $A$ to $B$. Note also that the arcs in a dependency graph are directional, so a path from $A$ to $B$ is not also a path from $B$ to $A$. The direction from parent node to child node is referred to hereafter as 'downstream'.

*Contradiction*: In the context of this algorithm, 'contradiction' refers specifically

to cases where the same pair of entities has been marked as interacting twice, with one entity as the agent in one case, and the other as the agent in the other case. This can occur between two passes of the algorithm, or within one pass. Because in reality this is very rare, the algorithm includes several optional rules to resolve such cases, which are controlled by various parameters described below.

Before listing the parameters which affect the behaviour of Algorithm I, I will describe the data preparation stages. At the end of the Methods subsection, before the results are presented, there is a graphical overview of the experimental protocol (Figure 4.4), including pre- and post-processing of the data and configuration of the algorithm itself, which the reader may find useful.

**Data preparation**

Given that the test data was split equally between sentences with interactions and sentences without interactions, but the training sentences all contained interactions, the first step was to acquire negative examples to add to the training data so that the impact of sentences without interactions on precision could be judged more accurately. The data was obtained by querying MEDLINE with the terms *Bacillus subtilis* and *transcription*, restricting the results to articles which were published after the LLL Challenge workshop took place in August 2005. It was split into sentences and the sentences were shuffled randomly, after which the first 77 which did not describe interactions were manually selected and added to the training set.

A thesaurus of gene/protein names used in the challenge was supplied by the organizers, mapping all of the aliases for each entity to a canonical name. None of these were ambiguous with each other or with general English words, meaning that all entity names could be tagged trivially in both the training and test sets with no ambiguity. I replaced each gene/protein name in each set with a symbol of the form `Entityxx`, where the last two characters are a two-letter code unique to that name, using a simple search-and-replace on the training data. This had several advantages. Firstly, some of the original names were in several word tokens, which would make extracting them from dependency graphs much more complicated. Ensuring that they were all single tokens meant that no entity would be split over more than one node in the graphs. Secondly, the numeric characters in some of the original entity names meant that the Charniak-Lease parser tagged them as numbers rather than nouns, thus altering the syntactic structure of the sentence context and obscuring interpretation. Finally, decoupling the interaction extraction from the entity recognition meant that it would be more straightforward to adapt the extraction algorithm to other datasets where a more sophisticated named entity recognition strategy would be necessary. Ongoing experiments with MSc students using unannotated data have borne this out.

The expanded training set and the test set were parsed with the Charniak-Lease

parser (Lease and Charniak, 2005) on account of its proven accuracy on biomedical text (see Chapter 2 and Chapter 3). Then the resulting parse trees were tagged with a grammatical function tagger (see Blaheta and Charniak, 2000, and Section 1.9.4) before being converted into dependency graphs. A function tagger is a program that can assign additional descriptive suffixes to certain phrase labels in parse trees, for example `-TMP` and `-LOC` for temporal and location modifiers respectively—phrases that describe when or where something is taking place—or `-LGS` for the logical subject of a passive-voice sentence. The Stanford tools can use these as hints when performing the tree→graph conversion, but they were not used in the experiment in Chapter 3 because the Blaheta and Charniak function tagger (apparently the only one available) is highly sensitive to small details of the parser's output format and as a result only works on parsers from the Charniak family.

The dependency graphs were generated with the 'collapsing', 'CC processing' and 'extra dependencies' options (see de Marneffe *et al.* (2006) and the documentation for Stanford's `GrammaticalStructure` class included with their software distribution[4]). This primarily meant that: preposition nodes were replaced with typed preposition dependencies from the head of the modified phrase to the head of the modifying phrase; the elements of coordinating conjunctions were attached in parallel to the word governing the coordination; and pronouns in relative clauses were replaced, where possible, with the referents from the parent clause in the argument positions of the relative clause's verb. For example, in the sentence *I saw the man who loves you*, an extra dependency will be added to indicate that *man* is logically the subject of *loves*. All these transformations are designed to bring the semantics of the sentence slightly closer to the surface in order to facilitate interpretation.

A summary of the steps involved in preparing and processing the data is presented in Figure 4.4.

### Global parameters

The details of the individual passes of the algorithm, and the parameters governing their behaviour, are described below; however, the algorithm also provides several global parameters that can affect more than one pass.

**banReflexives**: If set, interactions where the agent and target refer to the same entity are filtered out, unless there is a clue word in the sentence that indicates a reflexive relationship may be under discussion (*auto-*, *self-* or *itself*). Note that this parameter covers cases where the agent and target are different instances ('mentions') of the same entity, as well as those where the exact same node in the sentence graph has been used as agent and target of an interaction. Thus, it is a more general constraint than **banIdentityTriplets** (see below).

[4] `http://nlp.stanford.edu/software/lex-parser.shtml`

**banSymmetricalSubgraphs**: If set, the algorithm filters out interaction subgraphs where the sequence of arc labels from the interaction word to the agent (if applicable) is the same as the sequence from the interaction word to the target.

**banIdentityTriplets**: If set, interactions where the agent and the target are the same node are filtered out.

**removeNegatives**: If set, interactions where a negation word is found in the dependency path from the interaction word to either of the agent or target nodes are filtered out; likewise if the negator is directly attached (upstream or downstream) to any node in the path. Negation words include *not*, *no*, *none*, *negative*, *never*, *without*, *absence*, *cannot* etc.

**resolveContradictionsBetweenPasses**: If set, the **agentSelectorContradictionBetweenResolver** parameter (below) is used to choose between two interactions from *different* passes that contradict each other (i.e. one pass reports $X \rightarrow Y$, another reports $Y \rightarrow X$). If not set, both are admitted.

**agentSelectorContradictionBetweenResolver**: When two contradictory interactions are reported by different passes, and the previous parameter is set, the one whose agent meets the criteria determined by this parameter will be chosen. `WORD_FURTHEST`, `WORD_NEAREST`, `LEFTMOST`, `RIGHTMOST`, `PASS_EARLIER` and `PASS_LATER` are the allowable values. `WORD_FURTHEST` and `WORD_NEAREST` refer to the distance in words from the interaction word; e.g. if `WORD_NEAREST` is set, the interaction whose agent is closest to the interaction keyword will be chosen. Ties are broken by pass precedence, with interactions from earlier passes winning. `LEFTMOST` and `RIGHTMOST` simply indicate absolute position in the sentence. `PASS_EARLIER` means interactions from earlier passes win, `PASS_LATER` is the opposite.

**resolveContradictionsWithinPasses**: If set, the **agentSelectorContradictionWithinResolver** parameter (below) is used to choose between two interactions from the *same* pass that contradict each other. If not set, both are admitted.

**agentSelectorContradictionWithinResolver**: In the case of two contradictory interactions from the same pass, the one whose agent meets the criteria determined by this parameter will be chosen. `WORD_FURTHEST`, `WORD_NEAREST`, `LEFTMOST` and `RIGHTMOST` are the allowable values; the definitions of these are as above. Ties are broken randomly.

### First pass

The first pass is the core of the algorithm, and is required for a minimum base level of operation. It begins by locating interaction words in the sentence. Then for each of these, the algorithm traverses the dependency graph in the downstream direction in order to find any entities that can form candidate interaction subgraphs with the keyword (as described in Figure 4.1). If suitable entities are found, this is taken as

evidence that a genuine interaction is being described in the sentence, and the entities become potential agents and targets for the interaction. The algorithm does not restrict itself to looking for just a single agent and target for each interaction; every pair of accessible entities is considered, unless invalidated by one of the rules below. A set of dependency types characteristic of causal agents are used to assign each entity to the agent or target role of the interaction. For each valid agent/target pair for the interaction word, an interaction subgraph is formed with the interaction word at its root, and from this a new interaction triplet is created. The details of this process are governed by the following parameters.

**deepSearchAgentsFirstPass**: If set, an entity whose path from the interaction word contains a dependency type characteristic of the agent role *at any point* is classified as an agent.

**lateSearchAgentsFirstPass**: If **deepSearchAgentsFirstPass** is not set, and this parameter is, the algorithm requires that a dependency type characteristic of the agent role is present on the last arc in the path from the interaction keyword to a given entity, in order for that entity to qualify as an agent.

If neither of the above parameters are set, the algorithm only considers the dependency type of the first arc in the path from the interaction keyword to the entity, when choosing agents for the interaction keyword. This is illustrated in Figure 4.2.

**deepSearchTargetsFirstPass**: If set, an entity whose path from the interaction word contains no dependency types characteristic of the agent role *at any point* is classified as a target.

**lateSearchTargetsFirstPass**: If **deepSearchTargetsFirstPass** is not set, and this parameter is, the algorithm requires that no dependency type characteristic of the agent role is present on the last arc in the path from the interaction keyword to a given entity, in order for that entity to qualify as a target.

As with **deepSearchAgentsFirstPass** and **lateSearchAgentsFirstPass**, if neither of these parameters is set, the critical arc on the path to a candidate target is the nearest one to the interaction keyword. However:

**banAgentsAsTargets**: If this is not set, the state of the previous two parameters is ignored, and any entity downstream of the interaction keyword is a valid target, regardless of the presence of agent-like dependencies in the paths from the keyword.

It should be clear then that picking targets for an interaction is easier than picking agents, as the dependency types that characterize the agent role are in a minority, and the target selection process has the option of performing no filtering at all on dependency types. The design of the second pass (see below) reflects this fact.

**haltSearchAgentsFirstPass**: If set, only the first agent entity found along a given dependency path will be admissible. If not set, multiple agents along the same path will be allowed.

**haltSearchTargetsFirstPass**: The counterpart of the previous parameter; if set,

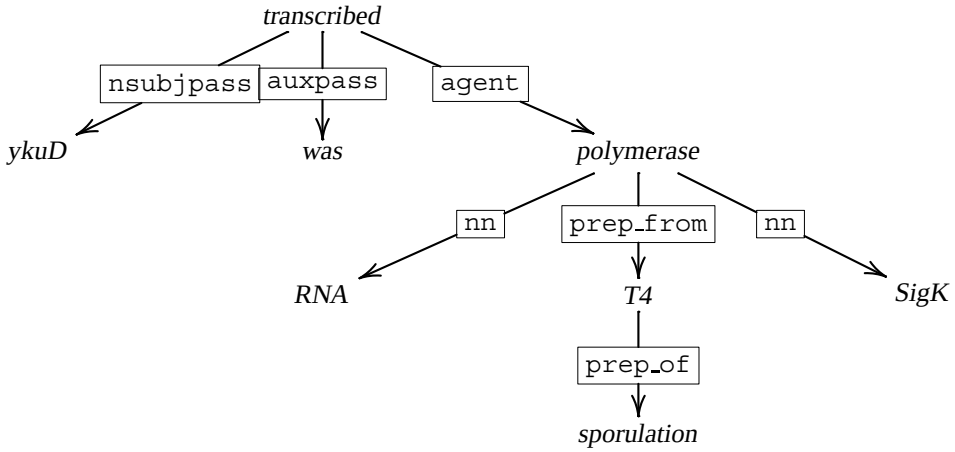


Figure 4.2: This graph of the sentence *ykuD was transcribed by SigK RNA polymerase from T4 of sporulation* illustrates the agent selection process in the first pass. It contains one interaction keyword, *transcribed*, and two entities which can form an interaction subgraph. The *syntactic* dependency type `agent`, used for the by-complement of passive verbs, is (unsurprisingly) one of the dependency types characteristic of the *semantic* agent role, so its presence can be used to identify *SigK* as the agent of the interaction rather than *ykuD*. However, if **lateSearchAgentsFirstPass** is set (and **deepSearchAgentsFirstPass** isn't) the algorithm will only look at the dependency *immediately governing* each candidate entity when trying to assign the agent role. In this case, that is `nn` (noun compound modifier), which is not characteristic of the agent role; the first pass will thus be unable to determine which of the entities is the agent. Note that there is an error in this graph, because the parser has chosen to attach *from T4 of sporulation* to *polymerase*, and not to *transcribed*, which it really modifies; fortunately this does not affect the algorithm's interpretation.

only the first target entity found along a given dependency path will be admissible. If not set, multiple targets along the same path will be allowed. See Figure 4.3 for an illustrative example.

**filterTripletsAfterFirstPass**: If set, the global filtering criteria defined by **banIdentityTriplets**, **banSymmetricalSubgraphs** and **removeNegatives** will be applied after the first pass completes.

#### Second pass

The second pass operates on those interaction keywords which have multiple candidate targets from the first pass, but for which the first pass has been unable to identify an agent. It was introduced because the criteria in the first pass for selecting agents are much narrower than those for identifying targets, and parse errors and complex

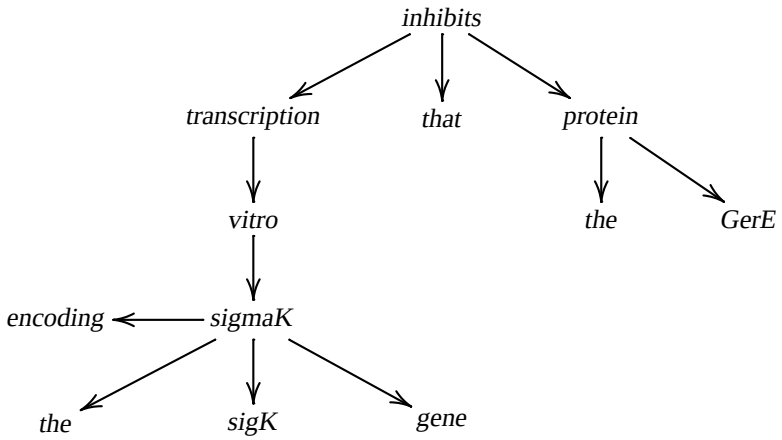


Figure 4.3: This graph of the sentence *the GerE protein inhibits transcription in vitro of the sigK gene encoding sigmaK* demonstrates the principles of the **haltSearchTargetsFirstPass** parameter. The dependency types have been left off this diagram for simplicity because they are not relevant here; assume that the algorithm has already identified *GerE* as the agent of *inhibits*, and *sigK* and *sigmaK* as targets. However, if **haltSearchTargetsFirstPass** is set, then *sigK* will be disallowed from acting in this role, as to get to it from *inhibits* by graph traversal is impossible without crossing *sigmaK* which would be a breach of the halting condition. In this case, this strategy makes little practical difference; since the LLL dataset does not distinguish between genes and their products, *sigK* and *sigmaK* are interchangeable, but such things are not considered at the graph traversal stage. There are two significant but not catastrophic parse errors evident in the structure of this graph, but as they do not make this demonstration of search halting any less valid, identifying them is left as an exercise to the reader.

phrase structures can exacerbate this. Based on the following parameters, the algorithm attempts to reassign one of the candidate targets as an agent, thus creating a new interaction triplet between the new agent and each of the remaining targets.

**doSecondPass**: This controls whether the second pass is run at all.

**agentSelectorSecondPass**: Allowable values are the same as for **agentSelectorContradictionResolver**, together with values `GRAPH_FURTHEST` and `GRAPH_NEAREST` which refer to the distance in arc 'hops' from the interaction word. This parameter is used to select an agent from the candidate targets (ties are broken randomly), provided the criterion specified by the next parameter is met.

**agentRestrictorSecondPass**: Allowable values are `BEFORE`, `AFTER` and `EITHER`. If this is set to `BEFORE` or `AFTER`, the candidate agent will be admissible only if it is before or after the interaction word in sentence word order respectively. If the entity selected according to the previous parameter is inaccessible because of this one, no agent is returned; the selected entity is returned to the pool of targets.

**filterTripletsAfterSecondPass**: If set, the global filtering criteria defined by **banIdentityTriplets**, **banSymmetricalSubgraphs** and **removeNegatives** will be applied after the second pass completes.

#### Third pass

The third pass operates on those interaction words which have at least one candidate target from the first pass, but for which the first and second passes have been unable to identify an agent. The algorithm attempts to identify a plausible agent from *any* of the entities in the sentence, apart from those which are already marked as targets for this interaction word, regardless of whether or not they are accessible by downstream graph traversal from the interaction word. The process is governed by the following parameters.

**doThirdPass**: This controls whether the third pass is run at all.

**agentSelectorThirdPass**: Allowable values are `WORD_FURTHEST`, `WORD_NEAREST`, `LEFTMOST` and `RIGHTMOST`. `GRAPH_FURTHEST` and `GRAPH_NEAREST` are not allowed because graph traversal is not used in this pass.

**agentRestrictorThirdPass**: Allowable values are `BEFORE`, `AFTER` and `EITHER`, as in **agentRestrictorSecondPass**. As before, if the entity selected according to the previous parameter is overruled by this parameter, no agent is returned.

**filterTripletsAfterThirdPass**: If set, the global filtering criteria defined by **banIdentityTriplets**, **banSymmetricalSubgraphs** and **removeNegatives** will be applied after the third pass completes.

#### Fourth pass

The fourth pass is a final fallback stage for interaction words for which neither an agent nor a target has been identified by any of the previous passes. It attempts to assign an agent and a target from anywhere else in the sentence, regardless of graph connectivity, based on the following parameters.

**doFourthPass**: This controls whether the fourth pass is run at all.

**agentSelectorFourthPass**: Allowable values are `WORD_FURTHEST`, `WORD_NEAREST`, `LEFTMOST` and `RIGHTMOST`, as in **agentSelectorThirdPass**.

**agentRestrictorFourthPass**: Allowable values are `BEFORE`, `AFTER` and `EITHER`, as in **agentRestrictorThirdPass**.

**targetSelectorFourthPass**: This parameter takes the same values as **agentSelectorFourthPass**, but governs the search for a target of the interaction rather than an agent.

**targetRestrictorFourthPass**: This parameter takes the same values as **agentRestrictorFourthPass**, but again governs the search for a target.

As in the previous two passes, if a decision made in accordance with a selector parameter is overruled by its corresponding restrictor parameter, no entity is chosen for the role in question.

**filterTripletsAfterFourthPass**: If set, the global filtering criteria defined by **banIdentityTriplets**, **banSymmetricalSubgraphs** and **removeNegatives** will be applied after the fourth pass completes.

#### Parameter selection

The algorithm has a total of 28 parameters, many of which take between two and six values, although obviously some combinations of parameters result in the same behaviour—for example, if **doSecondPass** is off, then the other second pass parameters are rendered irrelevant. In addition, it was not known in advance which of the entity selection or restriction parameters would yield the best results, either individually or in combination with each other. Therefore, some method for empirically determining a good parameter set was required. An exhaustive search of the parameter space was impractical since there are over four billion possible combinations of parameters (before removing redundant variations), so I used the JGAP genetic algorithms library [5] to perform stochastic parameter selection based on the LLL training set.

The process began by creating a population of randomly-generated parameter sets—I chose 200. A first generation was then produced by randomly 'breeding' the members of the random seed population with each other, producing offspring that displayed mixtures of their parents' characteristics, and introducing random 'point mutations' into their parameters. This produced a new population of around 550 parameter sets. The LLL training set was then processed once with each of these parameter sets, after which JGAP evaluated each result with a fitness function that measured each parameter set's ability to solve the problem at hand (that is, to correctly predict interactions). Applying a stochastic selection process, where fitter individuals are more likely to survive than less fit ones, JGAP could then select a subpopulation which would be used to breed the next generation via hybridization and mutation.

The breed-test cycle was repeated over 20 generations, with the fittest individual at the end of the last generation being chosen as the overall winner. The entire process was performed three times, once each with precision, recall and F-measure acting as the fitness function, thus producing a different winner for each of the three scoring measures. Note that there is no guarantee that these parameter sets are globally optimal, as an exhaustive search of the parameter space is not performed, but the random elements introduced in the breeding and mutation steps are designed to avoid the search getting stuck in local maxima, and the reasonably large population size (fluctuating around 550) is also an advantage. Note also that the fitness functions used in this process are

[5] `http://jgap.sourceforge.net/`

not the raw precision, recall and F-measure scores on the training set, for the following reason. Imagine two parameter sets, *X* (precision = 50.0, recall = 20.0) and *Y* (precision = 50.0, recall = 45.0). If raw precision was used as the fitness function, the genetic algorithm would not have any grounds to choose *Y* over *X* for breeding as they would be ranked equally. Instead, I set the fitness function for precision to be

$$f_P = P + \frac{R}{100} \tag{4.1}$$

where $P$ is the raw precision on the training set and $R$ is the raw recall. Therefore, a difference in recall acts as a tie breaker between parameter sets that result in equal precision. Similarly, I used

$$f_R = R + \frac{P}{100} \tag{4.2}$$

as the fitness function for recall, and

$$f_F = F - \frac{|P - R|}{100} \tag{4.3}$$

for F-measure, so that a result with unbalanced precision and recall would rank lower than one with the same F-measure but more closely balanced precision and recall.

Most JGAP options were left at their default values. I did however specify that the population size was allowed to vary from generation to generation, and that the best-performing individual from each generation should be kept automatically rather than being subjected to the normal stochastic selection procedure; both of these are disabled by default. Evaluating the effects of population size and number of generations on the selection process is not trivial, but some investigations into this issue are presented after the results on the LLL datasets.

#### Overview of experimental protocol

A high-level overview of the structure of this experiment is presented in Figure 4.4 for the parameter optimization process using labeled (training) data, and Figure 4.5 for the testing process on unseen (test) data. The latter process is much simpler, being essentially a linear series of steps that incorporates the results (optimal parameter sets) from the former process.

### 4.2.2 Results and discussion

Each of the three top parameter sets chosen by JGAP—hereafter referred to as *high-P*, *high-R* and *high-F*—were then used to process the LLL test set, and the results were submitted to the LLL scoring service for evaluation. Table 4.4 shows the scores achieved on the LLL training set (during parameter selection) and test set, on the hard-

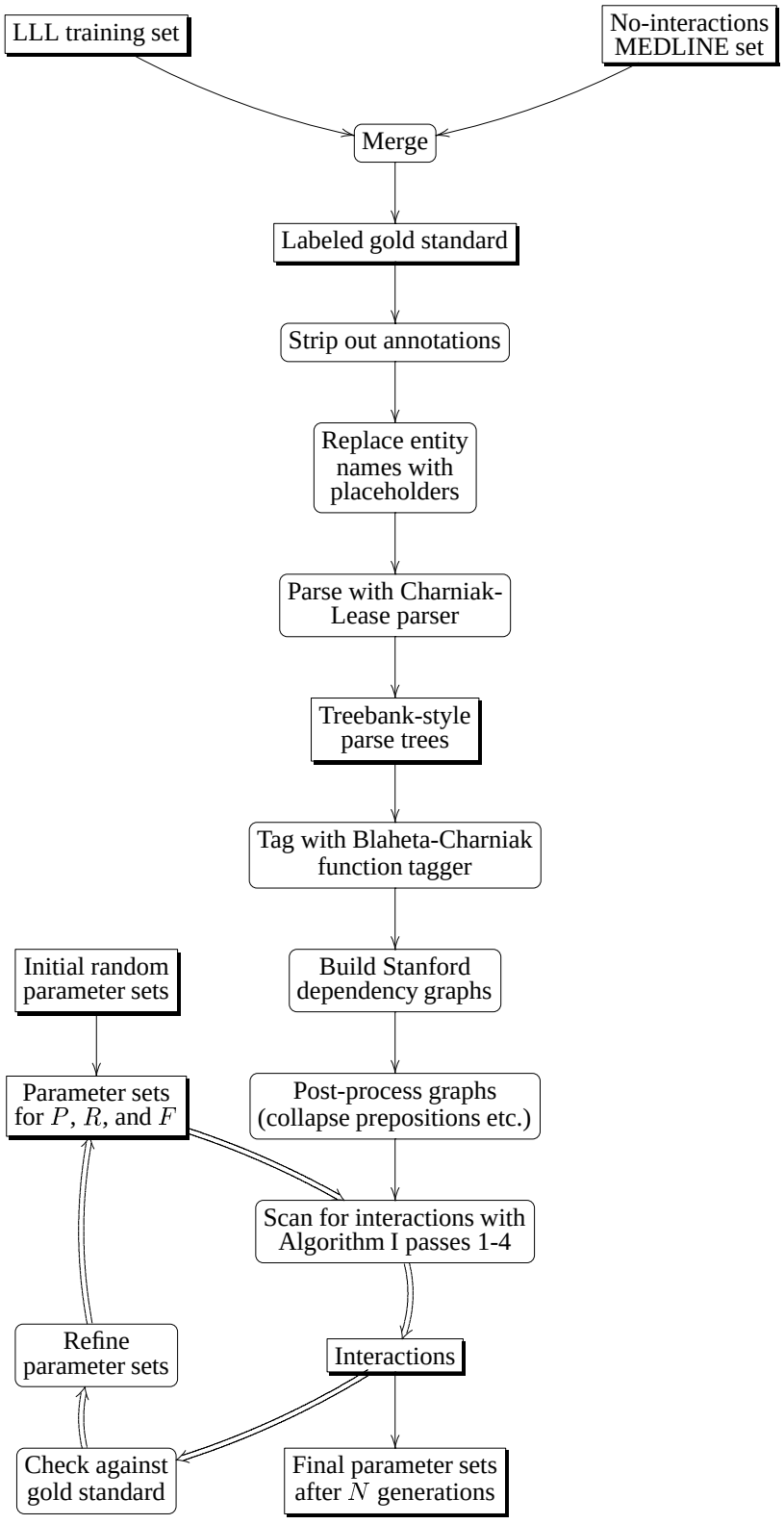


Figure 4.4: Overview of data preparation and parameter optimization protocol for Algorithm I. Shaded rectangles represent data and rounded boxes are actions. The lines in the training cycle are doubled to indicate iteration; the cycle repeats once every generation under the control of the genetic algorithm.

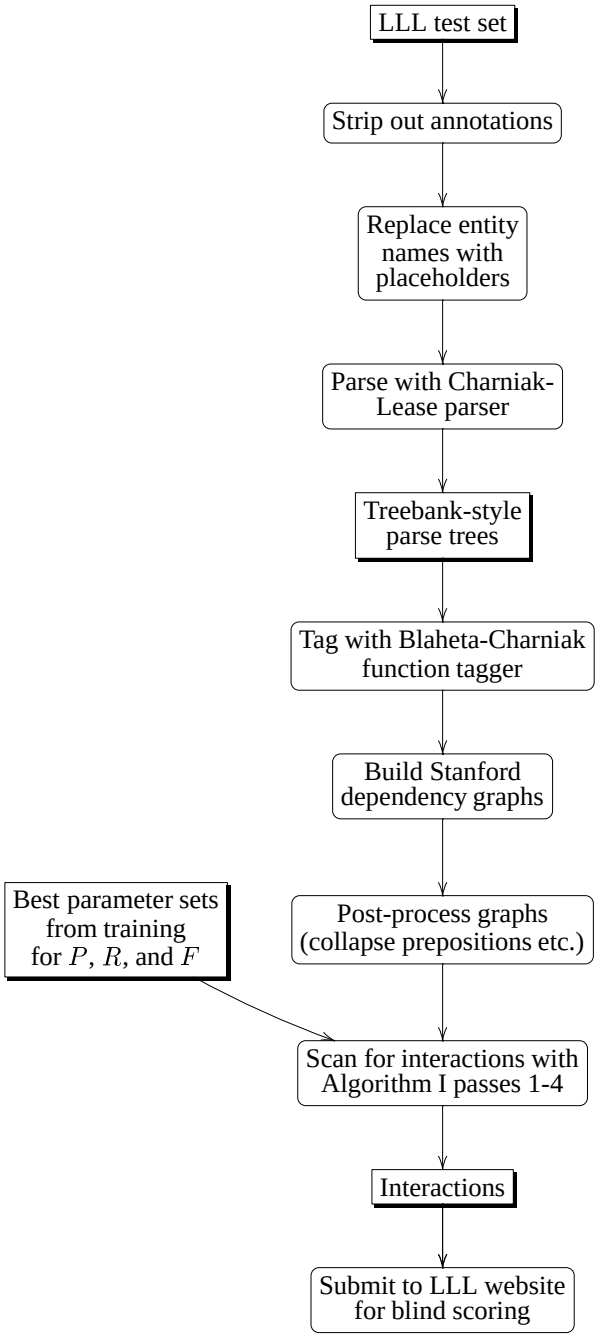


Figure 4.5: Overview of data preparation and evaluation protocol for Algorithm I. Shaded rectangles represent data and rounded boxes are actions.

| Scores on LLL corpus for Algorithm I | | | | | | |
|---|---|---|---|---|---|---|
| | Training set | | | Test set | | |
| Parameter set | $P$ | $R$ | $F$ | $P$ | $R$ | $F$ |
| *high-P* | **71.9** | 25.8 | 38.0 | **85.7** | 21.6 | 34.6 |
| *high-R* | 32.3 | **77.4** | 45.6 | 31.4 | **61.4** | 41.6 |
| *high-F* | 58.9 | 60.4 | **59.6** | 57.3 | 46.9 | **51.6** |

Table 4.4: Algorithm I's relationship extraction scores (precision, recall and F-measure) on LLL corpus, training and test, hardest subtask.

est subtask. Each parameter set has a considerably lower recall on the test set than on the training set; the fall-off was a little larger than I expected, given that the training and test sentences were drawn from the same topic and time period, and even selected so that they have the same distribution of interaction types. The teams in the challenge were not required to report their scores on the training set, which is a shame as that would have been informative. Group 6 mention that their official run which scored $F$ = 52.6 was the original highest-scorer on the training set, with $F$ = 65; although they don't give the source of the errors, looking at the individual test set scores ($P$ = 60.9, $R$ = 46.2) suggests that it was also primarily a recall problem. It is likely that there are syntactic constructions in the test set that are not found in the training set—such is the danger of using small datasets. Furthermore, given the big difference between Group 6's results with the perfect Link annotations and the noisy CCG parse, I suspect that some of these are of a kind that is problematic for parsers (see also Section 4.4.2).

Precision, on the other hand, was much easier to maintain on the test set, and even went up considerably under the *high-P* parameter set to 85.7. There are uses for high-precision methods; although a recall of 21.6 means that only one in five interactions are successfully recovered, this is less of a problem in large datasets where the better-established claims will be repeated several times anyway (see Section 5.2.2).

Table 4.5 shows the results for each of these parameter sets—hereafter referred to as *high-P*, *high-R* and *high-F*—in context with the original LLL contestants, ranked once again by F-measure. *high-P* is near the bottom, but *high-R* and *high-F* both come in above the only other run to handle both sentence types without using the linguistic annotations. Although they are still quite far from Group 6's high scorers, those rely heavily on the perfect Link data. Comparing like for like, then, *high-R* and *high-F* were very successful.

Two small points must be made about the rules of the task which may have had a minor negative impact on these scores. One is that in cases where a sentence reports two distinct interactions between the same two entities, each one is annotated separately, so there are potentially two true positives available. For example:

| Results of the LLL Challenge, plus Algorithm I | | | | | |
|---|---|---|---|---|---|
| Sentences | Annotations | Group | $P$ | $R$ | $F$ |
| easy | yes | 6a | 65.0 | 72.2 | **68.4** |
| both | yes | 6b | 63.2 | 66.2 | 64.7 |
| both | yes | 6c | 55.6 | 53.0 | 54.3 |
| easy | no | 6d | 68.5 | 44.4 | 53.9 |
| easy | yes | 6e | 60.9 | 46.2 | 52.6 |
| easy | no | 1 | 50.0 | 53.8 | 51.8 |
| both | no | $\langle F \rangle$ | 57.3 | 46.9 | 51.6 |
| easy | yes | 4 | 37.9 | 55.5 | 45.1 |
| both | no | $\langle R \rangle$ | 31.4 | 61.4 | 41.6 |
| both | no | 6f | 50.0 | 33.7 | 40.2 |
| easy | no | 5a | 25.0 | 81.4 | 38.2 |
| both | no | $\langle P \rangle$ | **85.7** | 21.6 | 34.6 |
| easy | yes | 5b | 20.5 | 90.7 | 33.4 |
| both | yes | 3 | 51.8 | 16.8 | 25.4 |
| hard | yes | 5c | 14.0 | 93.1 | 24.4 |
| hard | no | 5d | 14.0 | 82.7 | 24.0 |
| easy | no | 2 | 10.6 | **98.1** | 19.1 |

Table 4.5: The results of the LLL Challenge, in descending order of F-measure. $\langle 6 \rangle$ indicates Group 6's unofficial runs. $\langle P \rangle$, $\langle R \rangle$ and $\langle F \rangle$ are my *high-P*, *high-R* and *high-F* runs.

*These results suggest that a rising level of GerE in sporulating cells may first activate cotD transcription from the upstream site then repress transcription as the downstream site becomes occupied.*

If an algorithm only reports one *GerE*→*cotD* interaction, its recall on this sentence is only 50.0. I chose not to report both in cases like this, for the pragmatic reason that in many sentences, several of the words in my interaction keyword list appear even if there is only one interaction being described. Reporting multiple interactions between the same agent and target led to seriously impaired precision, as in those cases each keyword could report the same interaction.

Also, there is a rather ambiguous statement in the task instructions[6] about sentences where "the absence of interaction between two genes is explicitly stated"—this sounds like it might refer to negations but the example given has nothing to do with negation, so I was not sure how to proceed. A full discussion of this issue appears in Section 5.3.4. There are very few examples in the training set where either of these details might be a problem, and I assumed they would be just as rare in the test set.

At the time this experiment was performed, there was little that could be done in the way of error analysis without essentially re-creating the test corpus annotation from scratch and removing its status as a blind gold standard. However the scoring program does break each run's predictions down by interaction category, and provides a count of the number of spurious predictions that were made for sentences that did not have any interactions in, which I will call $X$ here for brevity. These scores are given in Table 4.6. Statements about regulons were a particular problem for all the runs, presumably because they tend to be phrased using idioms related to family or group membership which are not modeled well by the interaction subgraph concept as used here. Thankfully they came up rarely enough that this was not a great handicap. The physical interactions category, on the other hand, is large enough to have a significant impact on scores; the considerable difference in performance between the genetic and physical categories suggests to me that the small number of training examples in the latter category did not provide a broad enough range of sentence structures to reliably predict likely performance on the unseen data when performing parameter optimization. This of course is essentially the same problem faced by those who use wholly stochastic methods. On the other hand, *high-P*'s perfect precision on the largest category is impressive, even given a recall of just under a third, and it showed the same lack of false positives on the non-interaction sentences too—a reminder of the benefits of expanding one's training data.

[6] `http://genome.jouy.inra.fr/texte/LLLchallenge/`

| Scores by category on LLL test set for Algorithm I | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic | | Physical | | Regulon | | |
| Parameter set | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | $X$ |
| *high-P* | 100.0 | 29.1 | 60.0 | 8.7 | - | 0.0 | 0 |
| *high-R* | 50.7 | 67.3 | 30.2 | 56.5 | 100.0 | 20.0 | 45 |
| *high-F* | 76.7 | 60.0 | 54.5 | 26.1 | - | 0.0 | 14 |

Table 4.6: Algorithm I's precision and recall on each interaction category in the LLL Challenge test set, and the number of false positives produced for sentences without interactions ($X$). Categories in which no predictions were made cannot have a precision score since this would mean dividing by zero. Bear in mind that the relative sizes of the interaction categories are roughly in the ratio 68:25:7 in the order given here; *high-R*'s recall of 20.0 for regulon memberships comes from a single prediction.

### Analysing the parameters selected

One useful by-product of rules-based methods is that their behaviour is relatively transparent, unlike some stochastic classifiers. Table 4.7, Table 4.8 and Table 4.9 show which parameters were selected by the genetic algorithm for each winning parameter set. (Those parameters which are rendered redundant by other parameters have not been shown—e.g. if **doFourthPass** is false, none of the fourth pass parameters are shown.) Several interesting observations can be made regarding these lists. First of all, it is clear that the genetic algorithm chose parameters which reflect the overall architecture of this approach. When precision is chosen as the fitness function (Table 4.7), only the first pass is used. Only those interactions that can be found by downstream graph traversal from interaction keywords according to the appropriate rules are allowed. This is to be expected, as the later passes successively relax the constraints of the first parse in order to increase recall, at the risk of allowing more false positives. Sure enough, the genetic algorithm saw fit to use all four passes when the goal was maximum recall (Table 4.8) and passes 1–3 only when selecting for F-measure (Table 4.9). This corroboration between the design goals of Algorithm I and the parameters chosen by the selection process demonstrates the power of genetic algorithms in this kind of problem.

Furthermore, the parameters chosen to fine-tune the behaviour of the first pass in the *high-P* set also reflect the high-precision goal of that set. **banReflexives**, **banSymmetricalSubgraphs** and **banAgentsAsTargets** both filter out common causes of error identified during development, simultaneously increasing the risk of false negatives, but these errors are of minimal consequence when optimizing for precision. **banIdentityTriplets** also falls into this category, but it is redundant during pass 1 when either **banSymmetricalSubgraphs** or **banAgentsAsTargets** are set. (Since pass 1 can only find agents and targets by graph traversal from the interaction keyword, an identity

| Parameter set *high-P* for Algorithm I | | |
|---|---|---|
| Pass | Parameter | Value |
| Global | **banReflexives** | true |
| | **banSymmetricalSubgraphs** | true |
| | **banIdentityTriplets** | false |
| | **removeNegatives** | false |
| | **resolveContradictionsWithinPasses** | true |
| | **agentSelectorContradictionWithinResolver** | `WORD_FURTHEST` |
| Pass 1 | **lateSearchAgentsFirstPass** | true |
| | **deepSearchTargetsFirstPass** | true |
| | **haltSearchAgentsFirstPass** | false |
| | **haltSearchTargetsFirstPass** | true |
| | **banAgentsAsTargets** | true |
| | **filterTripletsAfterFirstPass** | true |
| Pass 2 | **doSecondPass** | false |
| Pass 3 | **doThirdPass** | false |
| Pass 4 | **doFourthPass** | false |

Table 4.7: Parameters selected for Algorithm I by the genetic optimization procedure, using precision as the fitness function.

triplet can only result from a symmetrical subgraph; also this entails the same node acting in both the target role and the agent role.) Less predictably, **removeNegatives** is not set either. This is surprising because as a filtering criterion it can only reduce recall, not precision. If the optimization process had performed an exhaustive search of the parameter space, this result would indicate that there were no candidate interactions proposed by pass 1 with these parameters where **removeNegatives** would have made any difference at all. However, since the parameter space was not exhaustively explored, there is a possibility that a parameter set without **removeNegatives** proved to have the highest precision at the last round of selection.

**haltSearchAgentsFirstPass** and **haltSearchTargetsFirstPass** both cause the algorithm to be conservative about making predictions, which is obviously a benefit when aiming for high precision. However the choices of **lateSearchAgentsFirstPass** and **deepSearchTargetsFirstPass** require a little more explanation. **lateSearchAgentsFirstPass** dictates that a candidate agent for a given interaction keyword is only valid if the last arc in the dependency path from the keyword to the entity is one of a small number that are typically characteristic of the agent role in a declarative statement (see Section 4.2.1). This is a fairly severe constraint, as there are many situations where this would not be the case. Take for example the sentence *sequestration of SpoIIE protein into the prespore plays an important role in the control of sigmaF activation*. Although

*SpoIIE* is embedded in a phrase which is the subject of the verb *plays*, it is not the head of that phrase—*sequestration* is. This means that *sequestration* will be attached to *plays* via that verb's nominal subject arc, and *SpoIIE* will be attached to *sequestration* via the prepositional *of* dependency which is not considered by the algorithm to be evidence of agent-hood. Therefore, the algorithm is not able to identify the agent for this interaction when using the *high-P* parameter set. Of course, it is trivial to suggest examples where there are no arcs in the subgraph at all which are identified as agent arcs—for example *sigma(H)-dependent expression of spo0A*—but these remain invisible to the first pass regardless of the parameters in use.

Looking at the *high-R* parameter set (Table 4.8), it may seem counter-intuitive that some of the global filtering criteria have been left on, since they can contribute nothing to recall. Note however that while **filterTripletsAfterFirstPass** and **filterTripletsAfterThirdPass** are true, their second-pass and fourth-pass equivalents are false, so each (presumably) higher-precision filtered stage is followed by a higher-recall unfiltered stage. Also remember that precision is used as a tie-breaker for runs that have equal recall, so it is not surprising to see a small amount of filtering taking place even here.

In pass 1, the halt search parameters are both off, and **deepSearchAgentsFirstPass** is on. This means that every downstream entity will be accepted as an argument for a given keyword, and evidence from every position between the keyword and the entity will be examined to decide whether it is the agent. Then, it is quite fascinating to see the algorithm 'cast the net wider' progressively from pass 2 to pass 4. Pass 2 looks for missing agents before the interaction word, pass 3 looks for them in either direction, and finally pass 4 looks for both agents and targets in either direction. This is perhaps what I would have done had I decided to specify the behaviour of each pass myself, but that would have been based on assumptions rather than evidence. By leaving these decisions open and learning the rules from the training data, they have some empirical justification.

The agent selector parameters also make an interesting point. I would have not known to pick `WORD_FURTHEST` in the second pass and `WORD_NEAREST` in the third, and I can't think of a convincing justification that doesn't sound slightly contrived. But it would appear that this is a genuine characteristic of the dataset, rather than an artefact of this run, as the same selectors and restrictors have been picked for these passes in *high-F* as well (Table 4.9).

The strategy adopted for *high-F* is interesting. The search parameters for the first pass are quite conservative, but not the same as those selected for *high-P*. It only uses the type of the first dependency arc on each downstream path as evidence that there might be an agent on that path, and does not go any further when it finds the first one. It will accept multiple targets on the same path, but only if there are no agent-like dependencies anywhere on that path. It does a lot of filtering during and after the

| Parameter set *high-R* for Algorithm I | | |
|---|---|---|
| Pass | Parameter | Value |
| Global | **banReflexives** | false |
| | **banSymmetricalSubgraphs** | true |
| | **banIdentityTriplets** | true |
| | **removeNegatives** | false |
| | **resolveContradictionsBetweenPasses** | false |
| | **resolveContradictionsWithinPasses** | true |
| | **agentSelectorContradictionWithinResolver** | LEFTMOST |
| Pass 1 | **deepSearchAgentsFirstPass** | true |
| | **haltSearchAgentsFirstPass** | false |
| | **haltSearchTargetsFirstPass** | false |
| | **banAgentsAsTargets** | false |
| | **filterTripletsAfterFirstPass** | true |
| Pass 2 | **doSecondPass** | true |
| | **agentSelectorSecondPass** | WORD_FURTHEST |
| | **agentRestrictorSecondPass** | BEFORE |
| | **filterTripletsAfterSecondPass** | false |
| Pass 3 | **doThirdPass** | true |
| | **agentSelectorThirdPass** | WORD_NEAREST |
| | **agentRestrictorThirdPass** | EITHER |
| | **filterTripletsAfterThirdPass** | true |
| Pass 4 | **doFourthPass** | true |
| | **agentSelectorFourthPass** | RIGHTMOST |
| | **agentRestrictorFourthPass** | EITHER |
| | **targetSelectorFourthPass** | LEFTMOST |
| | **targetRestrictorFourthPass** | EITHER |
| | **filterTripletsAfterFourthPass** | false |

Table 4.8: Parameters selected for Algorithm I by the genetic optimization procedure, using recall as the fitness function.

| Parameter set *high-F* for Algorithm I | | |
|---|---|---|
| Pass | Parameter | Value |
| Global | **banReflexives** | true |
| | **banSymmetricalSubgraphs** | true |
| | **banIdentityTriplets** | false |
| | **removeNegatives** | true |
| | **resolveContradictionsBetweenPasses** | true |
| | **agentSelectorContradictionBetweenResolver** | WORD_FURTHEST |
| | **resolveContradictionsWithinPasses** | true |
| | **agentSelectorContradictionWithinResolver** | RIGHTMOST |
| Pass 1 | **deepSearchAgentsFirstPass** | false |
| | **lateSearchAgentsFirstPass** | false |
| | **deepSearchTargetsFirstPass** | true |
| | **haltSearchAgentsFirstPass** | true |
| | **haltSearchTargetsFirstPass** | false |
| | **banAgentsAsTargets** | true |
| | **filterTripletsAfterFirstPass** | true |
| Pass 2 | **doSecondPass** | true |
| | **agentSelectorSecondPass** | WORD_FURTHEST |
| | **agentRestrictorSecondPass** | BEFORE |
| | **filterTripletsAfterSecondPass** | false |
| Pass 3 | **doThirdPass** | true |
| | **agentSelectorThirdPass** | WORD_NEAREST |
| | **agentRestrictorThirdPass** | EITHER |
| | **filterTripletsAfterThirdPass** | false |
| Pass 4 | **doFourthPass** | false |

Table 4.9: Parameters selected for Algorithm I by the genetic optimization procedure, using F-measure as the fitness function.

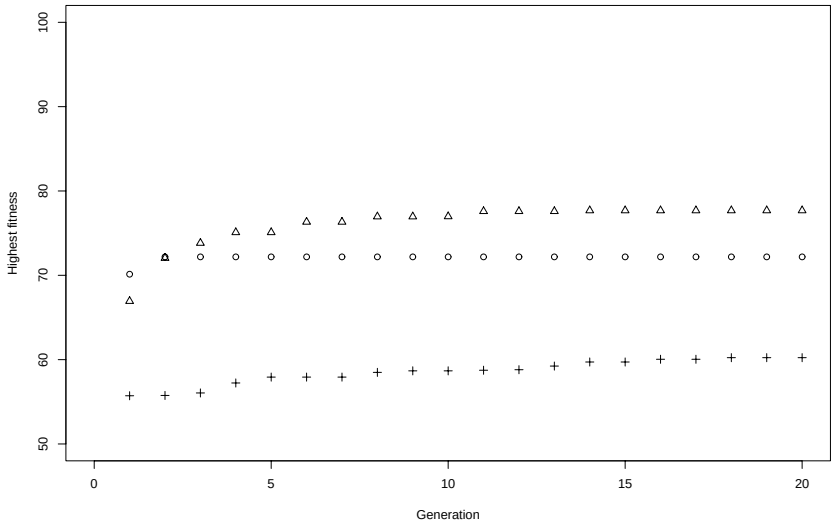


Figure 4.6: Graph showing increase in maximum fitness over time during the parameter selection process, over 20 generations. The three data series represent selection runs with each of the three fitness functions: precision (∘), recall (△), and F-measure (+).

first pass, then none after that. Manually designing a balanced system is much harder than one that favours precision or recall, as there are some very obvious choices you can make to widen or narrow the search space, but too many *potentially* well-balanced strategies to test them all one by one, especially with this many parameters. Delegating that part of the design process to an automated, stochastic procedure is a great benefit.

One drawback to that procedure is that although the run-time behaviour of Algorithm I is transparent and deterministic, the stochastic elements in parameter selection mean that the winning parameter sets may not represent globally optimal solutions. Also, JGAP provides the freedom to choose the size of the populations and the number of breeding generations, as well as various additional options. This means that one cannot assume that another selection run with a larger population or longer timescale would not produce radically different results. However, certain analytical methods can help us interpret the results of parameter selection.

One useful technique is to look at the change in highest achievable fitness score from generation to generation as the genetic algorithm runs. Figure 4.6 plots this curve for each of the three runs optimized for different fitness functions. The results are rather interesting. Precision hits a plateau almost immediately, changing very little from generation 2 to the end, with the same score achieved in the original optimization. Enabling passes 2-4 of the algorithm is likely to cause reduced precision, as a tradeoff for increased recall, therefore parameter sets which choose these options are

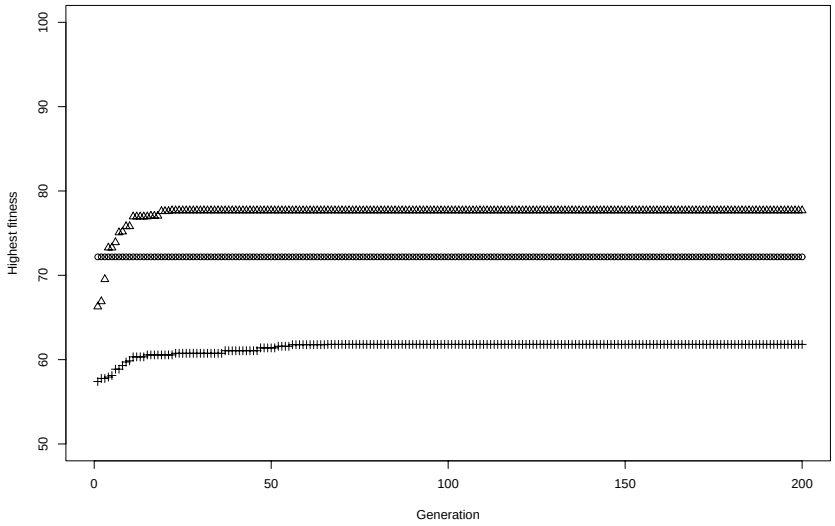


Figure 4.7: Graph showing increase in maximum fitness over time during the parameter selection process, over 200 generations. The three data series represent selection runs with each of the three fitness functions: precision (∘), recall (△), and F-measure (+).

more likely to get 'killed off' quickly—the net result meaning that large regions of parameter space are effectively closed off in this run. Recall starts low but grows rapidly, effectively leveling off at generation 11, whereas F-measure undergoes a slow, gradual increase throughout the selection process.

These plots suggest that of the three parameters, F-measure is the most likely to benefit from additional generations. In order to test this hypothesis, I ran the parameter selection process again, once for each fitness function, but with 200 generations instead of 20. The results are shown in Figure 4.7. As predicted, precision remained level; indeed, it achieved a result of P = 72.2 on the first generation, and remained constant. Once again recall went through a rapid growth spurt in the first few generations before leveling off, although its final plateau of R = 77.7 was not reached until generation 24, suggesting a small benefit for longer training times that was not apparent over 20 generations. The comparatively poor scores in the earlier generations reflect the fact that high-recall optimization requires more of the parameter space to be searched, particularly those regions relating to the fourth pass of the algorithm. Contrast this with the stability of precision optimization, during which solutions exploring passes 2-4 will be strongly selected against. Finally, F-measure enjoyed the only tangible benefit of the extra training time, reaching an ultimate F = 61.8 at generation 66, compared with 59.6 in the original 20-generation run. Interestingly, in this run it passed 59.6 at generation 9, demonstrating the unpredictable nature of stochastic optimization.

Another question worth addressing is whether the maxima being converged upon by the three runs are local or global. This is impossible to determine from just one run. Indeed, it is not even guaranteed that there *are* distinct global maxima for the three functions, as opposed to multiple local maxima without significant differences in fitness. Certainly it is true that there are contiguous regions of parameter space with identical fitness, for example where fourth-pass parameters vary but **doFourthPass** remains unset. It is conceivable that these can act as 'traps', since within these regions, changing any one random parameter is more likely than usual to have no effect whatsoever on the fitness score.

In order to look for evidence of such phenomena, I ran the parameter selection process 20 additional times for each fitness function, keeping the same seed population size (200 parameter sets) and duration (20 generations) as in the original experiments. To determine whether the winner of each run had arrived at solution similar to the original winner for the same function, I designed a similarity measure $S$ between two solutions. This is defined as the proportion of parameter values the two sets have in common, discounting any parameters from passes that are disabled in both solutions, and counting as non-matches any parameters from passes that are only enabled in one. Thus, when comparing two parameter sets that both have **doFourthPass** set to false, $S$ = the proportion of parameters from passes 1-3, plus global parameters, that are set to the same values. However, if one has **doFourthPass** set to true, $S$ = the proportion of all parameters that have the same values in both sets, but counting all fourth-pass parameters as non-matching regardless of their actual values.

$S$ is only an estimate of functional similarity; there are a few other specific parameter changes that do not affect Algorithm I's behaviour, such as toggling **banIdentityTriplets** when **banSymmetricalSubgraphs** is set and only the first pass is performed. However, it does enable us to plot solution similarity vs. fitness score difference against the original winner, for each new winner found for a given fitness function. Low score differences at low levels of similarity would indicate that there are distinctly different solutions which are equally effective.

Figure 4.8 shows the results of this experiment. Precision shows the most variation in parameter similarity, but each run achieved exactly the same score as the original. This indicates that there are many diverse parameter sets that produce equally effective behaviour, and is consistent with the results in Figure 4.7 that show a failure to improve precision even with ten times as much training time. For recall, there are several runs that do as well or nearly as well as the original, some down to comparatively low similarities, but not as low as for precision. This suggests that there are more key parameters that must be set correctly for high recall, presumably relating to the later passes of Algorithm I. On inspection, all 20 of the recall winners performed all four passes, with the exception of one that skipped pass 2 and scored the second lowest out of any of the recall winners. In the case of F-measure, the fact that most runs

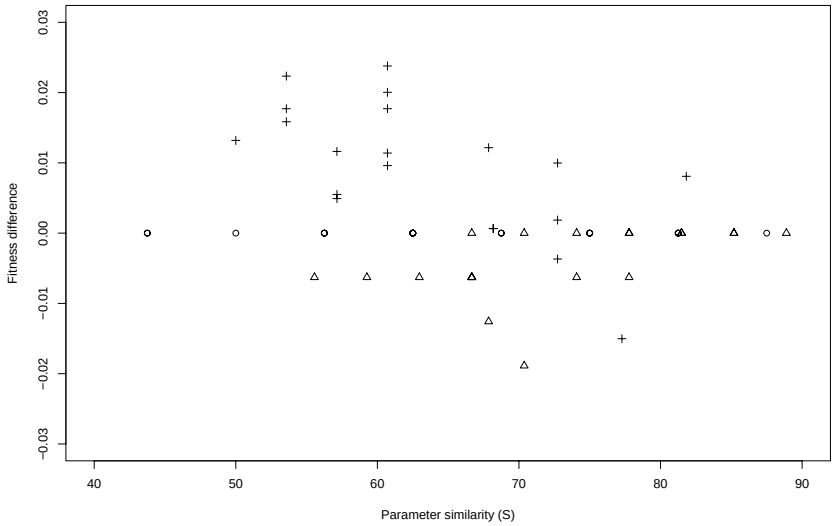


Figure 4.8: Graph showing similarity of parameters selected against differences in winning score against original winners, for 20 runs each of precision, recall and F-measure: $\circ$, $\triangle$, and $+$. Points above the midline represent runs that scored better than the original run for that fitness function, and points below scored worse.

scored higher than the original backs up the suggestion of Figure 4.7 that the original F-measure run could have benefited from further training. The absence of low-similarity, low-score-difference points for F-measure makes it less likely that the fitness landscape for F-measure contains multiple local maxima, although this cannot be ruled out.

## 4.3 Later developments

While I was preparing Algorithm I for publication, I became aware of two new papers that also dealt with the topic of the LLL corpus, but which unfortunately scored rather better. The first (Giuliano *et al.*, 2006) used kernel functions based on shallow linguistic features of the sentences that interacting entities are found in—words and short word sequences, lemmas, POS tags, orthographic features etc. These defined similarity scores for a pair of entities in one sentence and a pair in another (or, obviously, another pair in the same sentence) based on the regions of the sentence around and between the two entities. The order of the words was not taken into account, meaning that their solution was more akin to a text classification method than a typical information extraction algorithm. They trained a support vector machine (Cortes and Vapnik, 1995) on the LLL training set, using non-interacting pairs from that set as negative examples. The SVM could then be used to classify entity pairs in the test set into interacting and

not-interacting based on their similarity to the training data. They achieved $P$ = 56.0, $R$ = 61.4 and $F$ = 58.6—impressive given their apparently superficial 'bag of words' model.

The other paper (Fundel *et al.*, 2007) described a system called RelEx which was much more similar to my own approach, using graph-traversal rules over dependency graphs from the Stanford tools—although they also used Stanford's lexicalized parser which Chapter 3 suggests may have unwittingly cost them a few points. However, they combined the graphs with the output of a noun phrase chunker so that each noun phrase occupied a single graph node. They then used three separate routines to detect three different broad classes of construct. The first looked for *A activates B* style phrases by attempting to traverse the graph between each two pairs of noun phrases containing entities. Traversal could go upstream or downstream, but the graph was split above the parent of each `nsubj` or `nsubjpass` arc, to avoid crosstalk between the arguments of different verbs. The second routine looked for *activation of A by B* statements by finding the longest possible paths made up of just noun phrases and prepositions such as *by*, *for* and *with* which also contained gene names. A variant of this routine performed an analogous task within noun phrases. The third routine was similar to the second but extracts phrases like *interaction between A and B*. After extracting candidate interaction pathways with these three rules, Relex used a few pre-processing stages to filter out false positives, verify the agent/target associations and enumerate the members of coordinating conjunctions (*X activates Y and Z* etc.). This relatively simple approach attained scores of $P$ = 68, $R$ = 78 and $F$ = 72. The paper does not state explicitly whether this is on the full sentence set, but they used the easy 55-sentence training set for development purposes, and mention that RelEx does not attempt to resolve anaphora (coreference), which implies that these results are for the easy sentences only. Nonetheless it is very impressive.

Both of these algorithms scored higher than Algorithm I seemed to be capable of. Despite varying the genetic algorithm's parameters and trying several different parameter selection methods, I was unable to generate any parameter sets that scored significantly higher than the ones presented here. One limitation of Algorithm I is that although it can find qualitative trends in the training data—e.g. "for interaction keywords that remain without entities by the fourth pass, the most likely agent is the rightmost entity, and the most likely target is the leftmost"—it has no mechanism for learning systematic exceptions to these rules. Also, its core routine, the graph-traversal in pass 1, is rather rigid despite all the parameters, and does not allow for cases where there is no semantically-important keyword at the root of a subgraph covering two genuinely interacting entities.

However, the strength of dependency-based methods in general was reinforced by RelEx's striking results. It is hard to do a detailed comparison without access to RelEx itself (it is not publically available), and some of the advantage may be due to an ab-

sence of hard sentences in its test data, but there are interesting aspects to its design which seemed like they might be important. Firstly, noun phrase chunking seems like a sensible way to reduce unnecessary variability in the graphs and thus make simple methods more effective. In a phrase like *X inhibited Y expression* the target of the interaction is not attached directly to the verb as an object—*expression* is the object node in the graph, and *Y* is attached to it as a prenominal modifier. However, if a noun phrase chunker is applied so the phrase becomes *X inhibited Y_expression* then a simple pattern or routine designed to capture *X inhibited Y* will work here too. This means that the system must detect entities that are substrings of nodes, but this is already necessary as the Charniak-Lease parser treats strings like *sigma(H)-dependent* as single words so they end up in one graph node.

Another notable difference between their approach and mine is that RelEx does not require an interaction keyword to be at the head of a subgraph covering the two entities. Although this is a powerful way to filter out false positives, as my *high-P* results demonstrate, there are legitimate cases where this simply isn't how the sentence is laid out. For example in *X is under the control of Y* constructions, the Stanford scheme roots the graph at the verb *is*, and the keyword *control* is a prepositional object via the preposition *under*. Cases like these stand a decent chance of being captured by one of the fallback passes of my algorithm, but ideally, low-precision fallback processing should be reserved for problematic sentences where there is an actual error in the parse or the graph preventing interpretation, or a very obscure syntactic pattern that is hard to model in a sensible way. RelEx's approach—extract candidate pathways first and then check to see if they contain keywords at any point—means a much broader range of constructions can be coped with.

## 4.4 Algorithm II

In the light of the results achieved by Giuliano *et al.* (2006) and particularly Fundel *et al.* (2007), it seemed clear that additional development was necessary, so I begun work upon a replacement interaction extraction algorithm. Some of the design decisions were informed by the features of RelEx mentioned above, but I did not wish to simply reimplement RelEx or a variation on it. Instead, I decided to explore a different region of the solution space. Fundel *et al.* mentioned a few specific syntactic patterns that RelEx's small number of broad rules could not interpret, so I chose to design a larger number of more specific rules, along with mechanisms to generate variations on those rules. The other motivation for taking this approach was that the close analysis of the training data necessary to construct the ruleset would yield more insights into the nature of the problem and the characteristics of the chosen solution. This knowledge-rich methodology would also differentiate Algorithm II from Algorithm I's knowledge-poor approach.

### 4.4.1 Methods

One design goal for Algorithm II was to create an extensible platform for relationship extraction tasks that could be easily adapted to other scenarios rather than being too closely tied to the LLL data. To help maintain this flexibility, I designed a language called Metapattern Language (MPL) that permits the definition of subgraph patterns and the specification of expansion rules to allow the generation of allowable variants on those patterns. The patterns are defined over the node texts (i.e. words), node types (i.e. parts of speech) and arc types in a region of a dependency graph, and include placeholders and matching rules for identifying entities. Each pattern thus defines the syntactic relations that hold between a number of entities and a number of keywords, classes of words or POS tags; in other words, one or more syntactic constructions that are characteristic of a semantic relationship between those entities. In order to keep the matching algorithm and the MPL syntax simple, I restricted the patterns to those which can be expressed as trees—that is, they cannot contain arcs which would introduce cycles or multiple parenthood. These phenomena are not widespread in Stanford dependency graphs, however, and in my analysis of the LLL training data I found no examples where this restriction was problematic.

This approach was inspired by programs such as TGrep2[7] and Tregex (Levy and Andrew, 2006), the latter of which is included with the Stanford toolkit. These allow pattern matching over phrase structure trees such as those found in the Penn Treebank, using patterns which are essentially regular expressions describing hierarchical rather than linear data. The expressive capabilities of MPL are somewhat restricted compared to these tools, although MPL has the added complexity of supporting two labels on each node (word and tag) and one on each arc, compared to the singly-labeled nodes and unlabeled arcs of a syntax tree. Similar capabilities are provided by JAPE, part of the GATE package (Cunningham *et al.*, 2007) from Sheffield University,[8] CQP, part of the IMS Corpus Workbench (Christ, 1994) from the University of Stuttgart,[9] and Mother of Perl (Doran *et al.*, 1996), from the University of Pennsylvania. None of these systems provide native support for Stanford dependency graphs, however.

The software I wrote to support this approach consists of an MPL parser, for building in-memory representations of subgraph patterns supplied in a simple text format, and a matching engine, for detecting matches between a pattern and a given Stanford graph. In addition, I manually constructed a set of patterns to cover all of the examples found in the training set (excluding a small number of apparently erroneous interactions), making the rules as general as possible while at the same time aiming to keep the number of false positives to a minimum. After a few notes on preparing the data, I will describe the three elements of the Metapattern Language—match rules, pattern

[7] `http://tedlab.mit.edu/~dr/Tgrep2/`
[8] `http://www.gate.ac.uk/`
[9] `http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/`

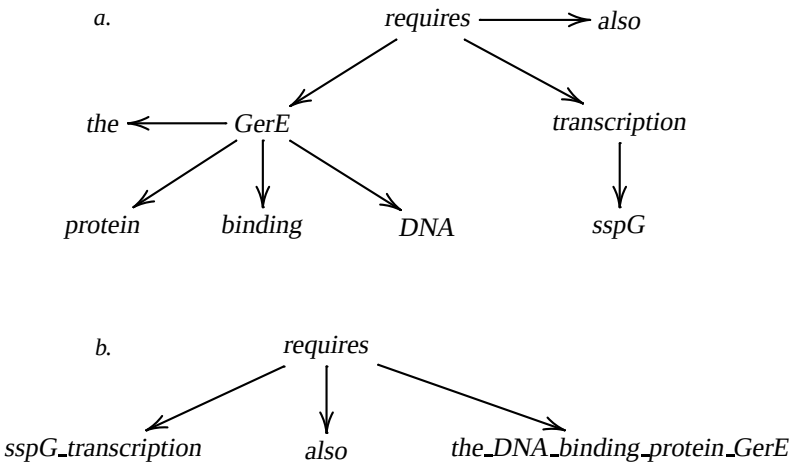


Figure 4.9: These two graphs show the sentence *sspG transcription also requires the DNA binding protein GerE*, both before (*a*) and after (*b*) noun phrase chunking. The number of nodes has been halved and the structural simplicity of the sentence has come to the surface.

rules and replacement rules—and give some examples of patterns created with them. A more formal language specification is provided in Appendix D. I will then describe the matching algorithm itself, and briefly summarize the experimental protocol in graphical form, before presenting my results on the LLL corpus.

**Data preparation**

The data preparation pipeline was largely the same as in Section 4.2.1, using the same expanded training set for algorithm development, with one major exception. In order to reduce the variation in the data, I adopted the RelEx policy of chunking noun phrases together into single nodes. To accomplish this, I developed a simple deterministic algorithm that processed the parse trees output by the Charniak-Lease parser before performing the tree→graph conversion with the Stanford tools. The algorithm looks for all subtrees containing only nouns, adjectives, numbers, foreign words, determiners (articles) and verb gerunds/participles. These are then glued together into a single word (with the spaces replaced by underscores) occupying a single tree node at the root level of the original subtree, with a new POS tag equal to the POS tag of the final word in the sequence. Similar operations are performed to flatten out coordinating conjunctions and appositional structures with parentheses. The resulting graphs generated from such trees are much simpler, as shown by Figure 4.9, and significantly reduce the number of patterns that must be generated to cover the training data (see below).

**Match rules**

The simplest rules in the MPL, these declare variables which can be used in the construction of subgraph patterns to match individual pattern elements—node texts, node types or arc types. A match rule requires a variable name (prefixed by `@`) and a regular expression to define the set of strings which the variable can match; this can just be a string literal in the simplest case, but this would be somewhat pointless as pattern rules (see below) allow for the embedding of literal strings anyway. A few examples of match rules follow:

```
match @AGENT = Entity[a-z]{1,2}
```

This rule defines the regular expression required to extract the agent of each interaction, according to the encoding scheme described in Section 4.2.1. Identical rules were declared for `@TARGET`, and `@ENTITY` (for additional entities that may be required to make sense of a syntactic construction but which are not the agent or target of the interaction being described). This rule will match any string *containing* the regular expression specified, even if there is additional text before or after.

```
match @VERB = ^VB.?$
```

This rule is designed to capture any part of speech representing a verb. The Penn Treebank tagset contains six POS tags for different verb forms—`VB`, `VBD`, `VBG`, `VBN`, `VBP` and `VBZ`. The regular expression in this rules looks for the letters VB followed by a single optional character. It will not match any longer strings containing these letters, because the `^` and `$` characters anchor to the start and end of a search string respectively. However, with the standard PTB tagset, this does not make a difference as no other labels contain `VB`.

```
match @AFFECT = ^(activate|affect|antagonise|antagonize)$
```

This rule groups together a set of verbs that appear in similar syntactic contexts, so that patterns can be defined using the variable `@AFFECT` which will match phrases that use any of the alternative words in the regular expression. The actual list is much longer than this, these are only the first four valid alternatives. Note the explicit spelling variation for 'antagonize'. Similar verb classes for `@AFFECTS`, `@AFFECTED` and `@AFFECTING` hold the inflectional variants (e.g. *activates*, *activated* etc.). These variations could have been automatically generated using morphological analysis software, but given the reasonably restricted scope of the LLL data, I elected to manually enter them rather than choose, test and evaluate another tool. The word lists were compiled from examples in the LLL training data and lists inherited from a previous text mining project (Domedel-Puig and Wernisch, 2005), and then manually expanded.

```
match @PREP_BY = ^(prep_by|prep_through|prep_via)$
```

This rule matches any of the dependency arcs representing prepositional *by*, *through* or *via* relations, as they are semantically similar and tend to occur in similar contexts in the LLL training set. Because of the collapsing described in Section 4.2.1, prepositions do not exist as separate nodes in the graph but simply become specifically-typed dependencies between the modified phrase and the modifying phrase.

Match rules can be negative as well as positive. A match rule preceded by an exclamation mark means the variable will match any string that the regular expression does not match. For example, the following rule means "match any string not containing the substring *Entity*":

```
!match @NOT_ENTITY = Entity
```

**Pattern rules**

Pattern rules define the basic syntactic patterns that can be matched, setting constraints on subgraph topology, the textual content of the nodes in the subgraph, the POS tags of the nodes and the dependency types of the arcs connecting the nodes. (Topology is the only strictly required information as the others may be completely generalized with wildcards if necessary.) For a matched pattern to yield an interaction, an `@AGENT` node and a `@TARGET` node must be defined. The following examples illustrate the syntax and various features of patterns:

```
pattern
VB~~@AFFECT
  ( @NSUBJ @NOUN~~@AGENT )
  ( dobj @NOUN~~@TARGET )
end
```

The first element in a pattern declaration represents a node of the graph—the root node of the subgraph matched by the pattern. The string to the left of the `~~` separator is the POS tag of the node, and the string to the right is the text of the node. These may be either literal strings or variables. In the pattern shown here, the root node will match any node with the exact POS tag `VB` and text matching the `@AFFECT` variable above (e.g. *activate*).

Each opening parenthesis indicates a dependency leading off the currently-open node (remember dependencies are directional), and must be followed by an element defining the dependency type. These are followed by POS-content pairs as for the root node. A closing parenthesis closes the current node. In this case, then, the root node can have two children. Both must contain entity placeholders and have POS tags matching the `@NOUN` variable. One, the agent, must be attached via any dependency matching the `@NSUBJ` variable (`nsubj` itself and some common substitutions), and the other must be attached via the `dobj` dependency (no variation allowed). This

pattern, then, will match phrases of the sort *Entityaa can activate Entitybb*, although the modal verb *can* is not explicitly matched by the pattern. Therefore the same pattern will also match *Entityaa should activate Entitybb*, *Entityaa must activate Entitybb* etc., and indeed *Entityaa will not activate Entitybb*, but negative cases can be easily removed by post-processing.

A node can have any number of children, although more than two is only required for certain very specific constructions, and a matched node in a sentence graph can have other children that are not covered by the subgraph pattern, as long as all the arcs and nodes specific by the pattern are matched. Children can of course be nested, allowing patterns to be specified more than two levels deep:

```
pattern
VBN~~@APPEARED
  ( xcomp VBN~~@AFFECTED ( agent @NOUN~~@AGENT ) )
  ( @NSUBJPASS @NOUN~~@TARGET )
end
```

This pattern matches phrases of the sort *Entityaa appeared to be activated by Entitybb*. The root node here is not the content-bearing keyword matched by `@AFFECTED`, but the rhetorical verb matched by its parent `@APPEARED`. While the agent of the interaction is the syntactic child of `@AFFECTED`, the target is the child of `@APPEARED`. This is one example of a pattern that succeeds where pass 1 of Algorithm I fails, because of Algorithm I's requirement that the interaction keyword is at the root of the subgraph containing the entities. Of course, pass 4 in Algorithm I may have picked up interactions like this by looking outside the interaction keyword's subgraph, but this is a rather unprincipled and unreliable solution.

This pattern is also an example of a passive-voice construction, a more complex version of the simple case *Entityaa is activated by Entitybb*. In these sentences, although *Entityaa* is the syntactic subject, it is the semantic target, and *Entitybb* which is a prepositional object is the semantic agent. The Stanford algorithm makes detecting such reversals easy, as the subject is given the dependency type `nsubjpass` rather than the usual `nsubj`, and the object is given the `agent` dependency type, both of which are unique to passive sentences and unambiguous.

One other important capability of pattern definitions is support for composite nodes. Composite nodes are nodes in the subgraph pattern whose textual content must match a sequence of variables and string literals; this gives the matching algorithm more granular access to the substrings of the node's text. They are useful when the semantics of a pattern rely on other text being present in the same node as one of the entities:

```
pattern
@NOUN~~@ROLE
```

```
  ( prep_of @NOUN~~{#AGENT_regulon} )
  ( @NSUBJ @NOUN~~@TARGET )
end
```

This pattern matches sentences like *Entityaa is a_member of the_Entitybb_regulon* (note the underscores where the noun phrase chunker has grouped noun phrases into single tokens), but the syntax with which the composite node is defined is a little different to that for a variable node or a simple string literal. The composite node is marked out by brace characters ({ }) and contains a variable (#AGENT) and a literal string (*regulon*) separated by an underscore. The underscore is a special wildcard which matches any single non-alphanumeric character, so the composite as a whole will match any node containing an entity placeholder (for the #AGENT variable) followed by a separator character followed by the word *regulon*. Additional material before or after the matched sequence (such as *the*) is allowed, but any additional material within it will cause the match to fail. Notice that the variable name is introduced by a hash character rather than the usual at-sign. This tells the MPL parser not to expand that variable with variable replacement rules (see below), as some of the replacement rules I designed will cause agent or target nodes to be expanded into entire new tree branches. If this happens to a composite sequence element it will break the structure of the composite. Hash-variables can be used anywhere a normal variable cannot be used because of problems caused by replacement rules. (In fact it is also possible to write different replacement rules specifically for hash-variables, should the need arise.)

The other major use of composite nodes is to specify patterns that only consist of a single node, for example:

```
pattern
@NOUN~~{#AGENT_@DEPENDENT_#TARGET}
end
```

This pattern matches phrases like *Entityaa-dependent_Entitybb_expression*, where the entire semantic content of the interaction is present in a single chunked noun phrase.

Patterns that are defined in MPL are known as seed patterns. This differentiates them from patterns created automatically by replacement rules, the final kind of rule in the language.

**Replacement rules**

Already mentioned above, these are simple macros that enable part of a pattern to be replaced with any arbitrary string. Thus, they facilitate the automatic generation of additional patterns from original patterns used as seeds. They are applied using simple string replacement on the ‘source code’ of the patterns after normalization of whitespace in the patterns, but before they are parsed and compiled into Java objects, making

them very flexible; this means however that care must be taken to avoid generating variant patterns which are syntactically invalid. (These concerns led to the introduction of the hash-variables described above.) After reading an MPL source file, the parser has a pool of patterns and an ordered list of replacement rules. The replacement rules are applied one by one to the pool in list order, keeping the original patterns in the pool; any new patterns are added to the pool after the final application of the rule that created them, meaning they are then subject to any replacement rules later in the list. If a pattern contains several instances of a string that is due to be replaced by a particular rule, as many new strings will be created as there are possible ways to apply the rule at least once. For example, if a pattern contains two instances of a string due to be replaced, three new variants will be created, one with each of the replacements made individually, and one with both made.

There are several uses for replacement rules. One is the substitution of entire nodes (content and POS) for semantically equivalent nodes which encode different syntactic forms, e.g. conjugations of verbs:

```
replace VB~~@AFFECT = VBP~~@AFFECT
replace VB~~@AFFECT = VBZ~~@AFFECTS
replace VB~~@AFFECT = VBD~~@AFFECTED
replace VB~~@AFFECT = VBG~~@AFFECTING
```

For each pattern containing a node specified as `VB~~@AFFECT`, the English verb base form, each of these rules will create an equivalent pattern with a different verb form specified—the present tense (non-3rd-person singular), the present tense (3rd-person singular), the past tense, and the present participle or gerund. The variables `@AFFECT`, `@AFFECTS`, `@AFFECTED` and `@AFFECTING` must be set up in advance to contain parallel word lists in the different tenses—this conversion is not performed automatically. Note that in English, the present tense (non-3rd-person singular) of a regular verb (e.g. *activate*), tagged `VBP`, is identical to its base form, tagged `VB`. Therefore, an alternative way to handle this situation would be to define patterns using the base form of the word list `@AFFECT` with a variable POS tag that matched either `VB` or `VBP`, and then have three replacement rules to generate variant patterns for the other three inflections.

Analogous operations can be carried out on words that are specified by string literals rather than variables, because of irregularity or lack of synonyms:

```
replace VBZ~~is = VBP~~are
replace VBZ~~is = VBD~~were
replace VBZ~~is = VBD~~was
replace VBZ~~is = VBG~~being
```

```
replace VB~~belong = VBP~~belong
replace VB~~belong = VBZ~~belongs
replace VB~~belong = VBD~~belonged
replace VB~~belong = VBG~~belonging
```

Once again, these tense variants could conceivably have been automatically generated; such an approach for example is taken by the LVG program (Divita *et al.*, 1998). Alternatively, all the words in the input data could have been replaced with their lemmas (canonical forms) and the patterns written to use these rather than their inflected forms. However, either approach could introduce additional sources of error.

The other main use for replacement rules is in the actual expansion of patterns to cover more nodes, in cases where a basic pattern needs to be extended to cover semantically cognate instances with extra syntactic detail which does not change the essential meaning. For example, the following pattern given at the start of this discussion matches the simple declarative statement *Entityaa can activate Entitybb*:

```
pattern
VB~~@AFFECT
  ( @NSUBJ @NOUN~~@AGENT )
  ( dobj @NOUN~~@TARGET )
end
```

The following replacement rule swaps the target node, which is a leaf node, for an intermediate node governing the new target node:

```
replace @TARGET = @PROCESS ( prep_of @NOUN~~@TARGET )
```

Assuming `@PROCESS` can match any arbitrary string, this creates a new pattern which can interpret sentences like *Entityaa can activate expression of Entitybb*. The overall pattern is the same, but the replacement rule has modeled a common substitution whereby a process in or change of an entity is given as the endpoint of an interaction, rather than the target entity itself. If it is not clear how this works, remember that the replacement rules work on the string representations of patterns before they are parsed; all it does is create a new string representation from the original that can be parsed and compiled independently:

```
pattern
VB~~@AFFECT
  ( @NSUBJ @NOUN~~@AGENT )
  ( dobj @NOUN~~@PROCESS ( prep_of @NOUN~~@TARGET ) )
end
```

There are many similar examples to this one where even the creation of a replacement rule was unnecessary because of the effects of the noun phrase chunker. For example, the sentence *Entityaa can activate Entitybb_expression* is matched by the original, unmodified pattern. Without the chunker, a separate replacement rule would have been required.

**Rule development procedure**

Using the LLL training set, I iteratively developed a set of MPL rules that jointly maximized precision and recall while still maintaining a degree of flexibility to deal with unseen sentences. This proceeded as follows, after coding a few simple cases. I would take the next interaction from the training set that was not yet predicted by Algorithm II using the current set of MPL rules, and examine its graph structure visually using the `dot` program from the Graphviz suite.[10] If it resembled an existing pattern sufficiently closely, I would extend that pattern (using replacement rules or vocabulary additions, or by relaxing the constraints of the pattern) to match the new interaction. If not, I created a new pattern to match it in a non-specific manner (i.e. allowing as much flexibility in the pattern as was practical). Then I would run the algorithm again, to ensure that the newly-changed or newly-created pattern worked properly, and to check for false positives incurred. If any false positives had been brought about by the changes, I would tighten up the pattern constraints until they disappeared. Then I would move on to the next unpredicted interaction.

In practice, I allowed a small number of false negatives due to apparently erroneous annotations in the training data, and a small number of false positives where the alternative would have been to tailor a pattern so closely to a particular sentence that it lost all generality. Where there were obvious errors in the training data's dependency graphs, I added patterns for both the incorrect version (unless this would cause a large number of false positives) and the corrected version, if I was able to correct it. Also, where an existing replacement rule looked potentially useful and productive in a context not found in the training data, I adapted it to that context if possible. The principle behind these decisions was to anticipate, as far as possible, plausible syntactic variations similar to phenomena in the training set, in order offset the shortage of data.

**The matching engine**

The matching algorithm itself is a fairly straightforward depth-first search. For each node in the graph, the root node of every pattern in the pattern set is examined to see if it matches. If it does, two 'current position' pointers are created, one into the graph at the matching node, and one into the pattern at the root node. It then proceeds to the first child node in the pattern and looks for a matching child node of the current graph node.

[10] `http://www.graphviz.org/`

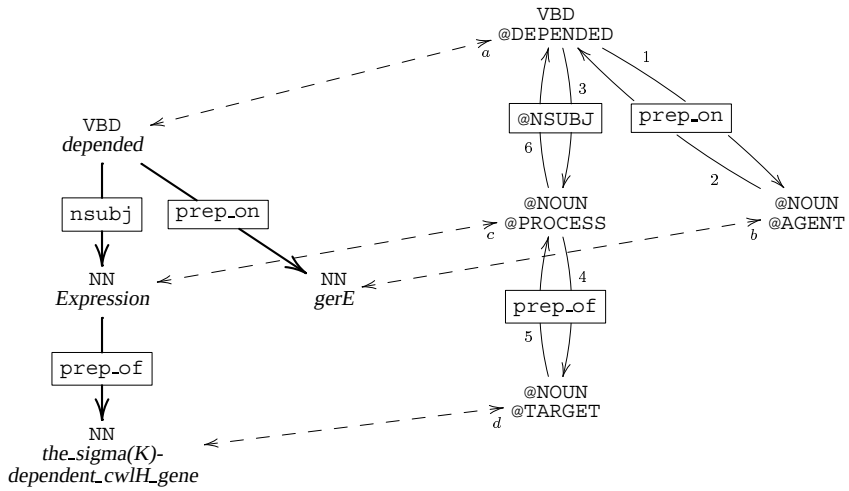


Figure 4.10: This shows a subgraph pattern on the right matched to a graph from the corpus on the left. Each POS tag, word and dependency type in the pattern has been matched to one in the graph, most of them by match rules representing regular expressions (e.g. `@NSUBJ`, `@PROCESS`) but some of them by literal string matching (e.g. `VBD`, `prep_of`). The numbers 1–6 show the order of traversal of the pattern (mirrored in the graph) and the letters a–d show the order in which the nodes were matched.

(Note that this involves matching the dependency types, POS tags and node values.) If one is found, the algorithm advances its pointers so that the two matching children are now the current nodes, and repeats the process (keeping a record of where it has been in both structures to avoid accidentally matching the same node twice). If it runs out of child nodes from the pattern, it has successfully come to the end of a branch in the pattern, and backtracks up both the pattern and the graph until the start of another unmatched pattern branch is found, from where it can continue the process. When it reaches the root of the graph without finding any more unmatched pattern branches, the match is complete. If, on the other hand, it ever reaches a state where there is another child node to match from the current position in the pattern, but the current node in the graph has no matching children available, the match fails.

This process is illustrated in Figure 4.10. The basic method is complicated slightly by the fact that a pattern can match more than once starting from the same node in a graph. These situations become possible when more than one child of the current graph node can match the next child from the pattern. If this is the case, the current state of the solution (the pointers and history) is cloned so that there is one solution for each of the matching children. These are then extended down their respective branches individually.

Once all matches between a pattern and a sentence have been found, the matching

engine outputs a set of $\langle AGENT, TARGET \rangle$ pairs consisting of the entity names corresponding to the pattern's `@AGENT` and `TARGET` placeholders at each distinct match position. Note that since each node can be an entire chunked noun phrase, sometimes containing several entities (e.g. *X_and_Y_genes*), it is possible for multiple distinct matches to be reported for the same set of nodes within a sentence's graph, when one node provides several agents or targets. Each of these $\langle AGENT, TARGET \rangle$ pairs is recorded as an interaction in the LLL file format.

**Overview of experimental protocol**

One aspect which sets Algorithm II apart from Algorithm I is that there is a degree of manual work required in the initial stages. Rather than adjusting the algorithm's parameters automatically in response to sample data, I used the LLL training data as examples from which more general rules could be derived. Nonetheless, the pattern development process described above and illustrated in Figure 4.11 could be considered analogous to the parameter optimization procedure in Algorithm I, in that they both rely on an iterative, empirical cycle of refinement. When viewed as a high-level workflow abstracted from implementation details, the actual testing process for Algorithm II on unseen sentences (Figure 4.12) is almost identical to that of Algorithm I (Figure 4.5).

### 4.4.2 Results and discussion

The rule development procedure resulted in an MPL file containing 82 pattern definitions for seed patterns (those explicitly written rather than automatically generated), 67 match rules and 59 replacement rules. After pattern expansion, there were a total of 505,352 patterns, indicating that on average 6,162 new patterns were created for each original pattern. These patterns covered the training data with a precision of 97.5 and a recall of 96.9, for an F-measure of 97.2.

I processed the LLL test set using these patterns, and submitted the results to the LLL website for scoring based on both sentence types together. Although they achieved an excellent precision of 90.9, the recall was only 24.0, giving an F-measure of 38.0, a large and somewhat surprising reduction compared to the training set. Although I had expected some performance difference between training and test, the degree of recall loss was surprising; given that there were half a million patterns after the pattern expansion process, I had expected better coverage of the problem space. Using F-measure as an overall indicator, these results were slightly better overall than Algorithm I's worst run *high-P* (see Section 4.2.2), but lower than its other two, lower than Group 6's comparable run without manual annotations or sentence segregation (see Section 4.1.2) and considerably lower than those presented by Giuliano *et al.* (2006) and Fundel *et al.* (2007). Looking at the rather limited output of the LLL scoring service, I discovered that my system had only made 22 predictions, with 2 of those turning out to be false

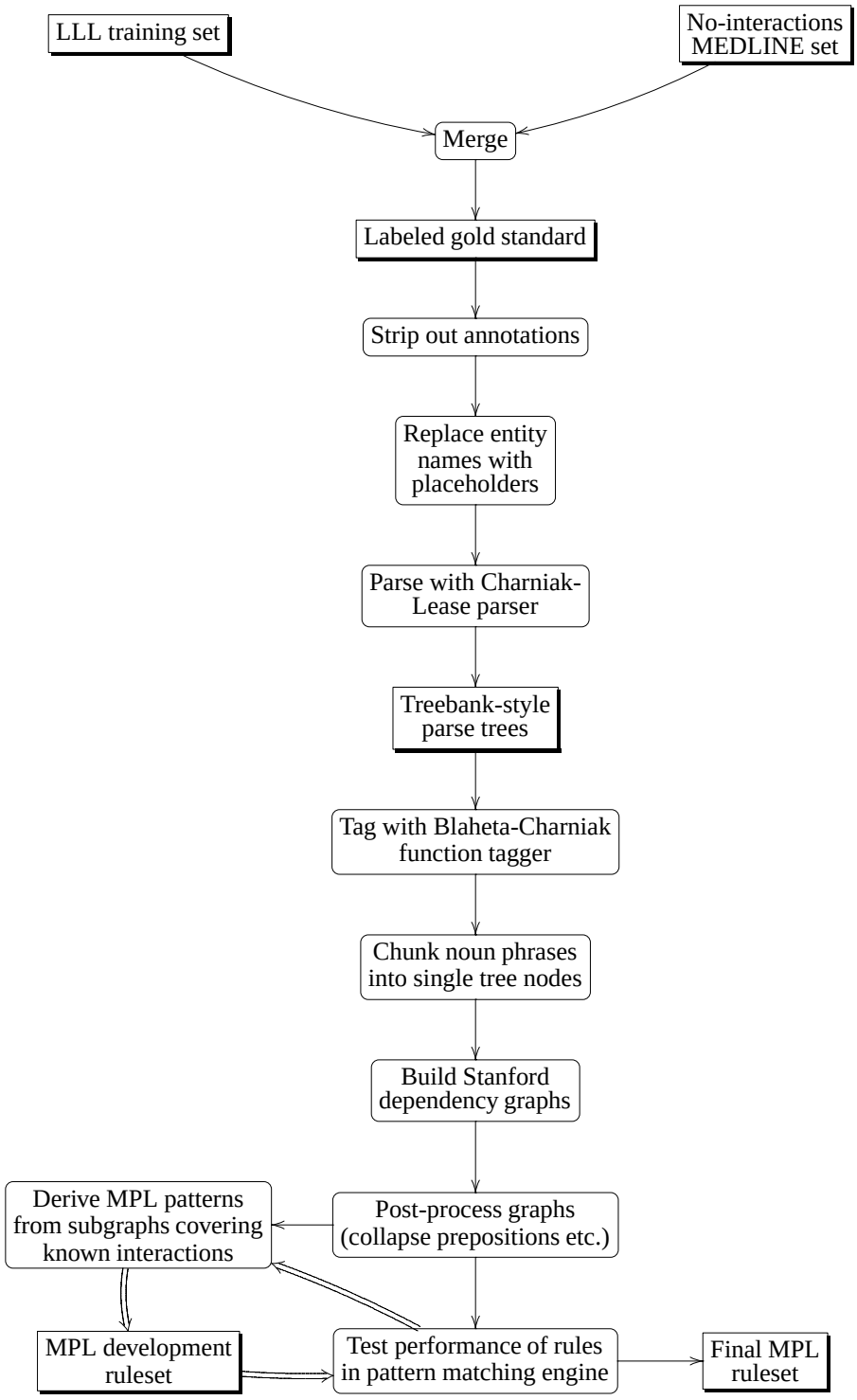


Figure 4.11: Overview of data preparation and MPL pattern development protocol for Algorithm II. Shaded rectangles represent data and rounded boxes are actions. The iterative rule development cycle continues until the pattern matcher achieves as close to perfect accuracy as possible.

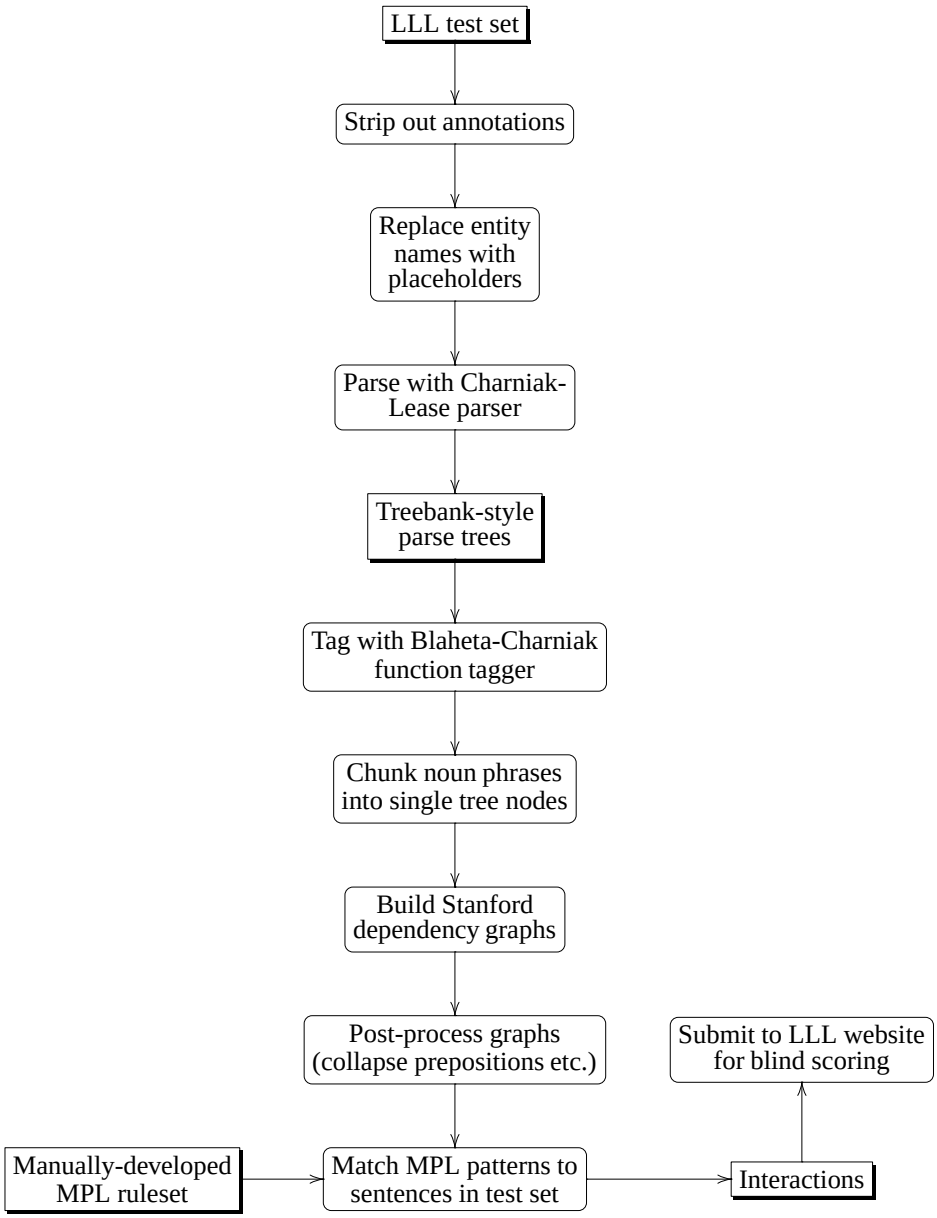


Figure 4.12: Overview of data preparation and evaluation protocol for Algorithm II. Shaded rectangles represent data and rounded boxes are actions.

positives. It had predicted 3 out of 5 regulon membership interactions correctly, but only 1 out of 23 molecular (physical) interactions and 16 out of 55 gene regulatory interactions.

**Probing the problem space**

The LLL organizers had decided to withhold the correct answers from the public in order to avoid tainting their status as a completely blind test set. Partly out of respect for this decision, and partly due to time and skill constraints, I decided not to make a detailed manual inspection of the test set. However, leaving this result without any further analysis would prevent any lessons being learnt from Algorithm II. I decided that an interesting experiment would be to start with a basic set of MPL rules capturing some fairly straightforward syntactic structures, and measure their results on both the training and test sets, thus getting an idea of how many of the 20 correct predictions in the test set were down to simple, broadly applicable rules. Retaining all the match rules, I stripped out all the replacement rules that make structural changes to patterns, and all the patterns that were marked as complex phrases, found in one training sentence only, or designed to cope with a known error. This left a base set of 29 seed patterns and 47 replacement rules, very few of which actually applied to the base set of patterns, as only 72 patterns existed after expansion. This basic set scored $P$ = 100.0, $R$ = 31.0, $F$ = 47.1 on training, and $P$ = 100.0, $R$ = 14.4, $F$ = 25.2 on test. In other words, more than half of the correct results achievable with half a million patterns on the test set had been recovered with just 72 patterns.

Going back to the MPL definitions once more, I added two simple replacement rules to allow common variations on agent and target names:

```
replace @AGENT = @PROCESS ( prep_of @NOUN~~@AGENT )
replace @TARGET = @PROCESS ( prep_of @NOUN~~@TARGET )
```

Since `@PROCESS` is defined to match any word, these replacements allow a pattern representing *A inhibits B* to also match *A inhibits production of B*, *Expression of A inhibits B*, and *Expression of A inhibits production of B*.

These increased the total number of patterns to 228, and achieved $P$ = 100.0, $R$ = 43.4, $F$ = 60.5 on training, but exactly $P$ = 90.9, $R$ = 24.0, $F$ = 38.0 on test. This was the same score profile on the test set as the initial run with the full rule set, and indeed turned out to consist of the same set of predicted interactions. The results for the original parameter set and the two cut down versions are summarized in Table 4.10. I have also summarized the test set score breakdown by interaction category provided by the LLL scoring service, for *min* and *max* only since *med*'s output on test was identical to *max*'s—see Table 4.11.

| Scores on LLL corpus for Algorithm II | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Pattern | Seed | Final | Training set | | | Test set | | |
| set | patterns | patterns | $P$ | $R$ | $F$ | $P$ | $R$ | $F$ |
| *max* | 82 | 505,352 | 97.5 | **96.9** | **97.2** | 90.9 | **24.0** | **38.0** |
| *min* | 29 | 72 | **100.0** | 31.0 | 47.1 | **100.0** | 14.4 | 25.2 |
| *med* | 29 | 228 | **100.0** | 43.4 | 60.5 | 90.9 | **24.0** | **38.0** |

Table 4.10: Algorithm II's relationship extraction scores (precision, recall and F-measure) on LLL corpus, training and test, hardest subtask. *max* is the original pattern set scoring as high as possible on the training set; *min* is the minimal set of commonly-found patterns in the training set, without complex replacement rules, and *med* is the minor extension of *min* which does as well as *max* on the test set.

| Scores by category on LLL test set for Algorithm II | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic | | Physical | | Regulon | | |
| Pattern set | $P$ | $R$ | $P$ | $R$ | $P$ | $R$ | $X$ |
| *max* | 94.1 | 29.1 | 100.0 | 4.3 | 100.0 | 60.0 | 1 |
| *min* | 100.0 | 16.4 | - | 0.0 | 100.0 | 60.0 | 0 |

Table 4.11: Algorithm II's precision and recall on each interaction category in the LLL Challenge test set, and the number of false positives produced for sentences without interactions ($X$). Categories in which no predictions were made cannot have a precision score since this would mean dividing by zero. Bear in mind that the relative sizes of the interaction categories are roughly in the ratio 68:25:7 in the order given here.

| Revised scores on LLL test set for Algorithm II | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Pattern set | Seed patterns | Final patterns | Test set (website) | | | Test set (local) | | |
| | | | $P$ | $R$ | $F$ | $P$ | $R$ | $F$ |
| *max* | 82 | 505,352 | **90.9** | **24.0** | 38.0 | 92.6 | **31.2** | **46.7** |
| *min* | 29 | 72 | **100.0** | 14.4 | 25.2 | **100.0** | 19.2 | 32.3 |
| *med* | 29 | 228 | **90.9** | **24.0** | 38.0 | 92.6 | **31.2** | **46.7** |

Table 4.12: Algorithm II's relationship extraction scores (precision, recall and F-measure) on LLL test set, hardest subtask. Pattern sets are described in Table 4.10. This table shows the differences in scores reported on the same task, between the LLL Challenge scoring service and the local fully-annotated test set supplied later.

**Error analysis on LLL test set**

Early in 2008, several months after this experiment was completed, the LLL Challenge organizers agreed to share the gold standard version of their test set—complete with details of the correct interactions for each sentence—in advance of an enhanced public release. Although time constraints meant that a detailed comparison of the two sets was not practical, the new data did enable me to investigate the serious recall differences suffered by Algorithm II between the training set and the test set.

The first interesting phenomena was that the scores achieved by Algorithm II's three runs as reported by the LLL Challenge's website—shown in Table 4.10—did not match those reported by the LLL scoring program when run locally on the fully-annotated test set. These are shown in Table 4.12. The reasons for this apparent improvement are not clear—a manual comparison of Algorithm II's output and the answers in the test set yielded scores which agreed with the local scoring program. The LLL organizers did remove one mis-annotated sentence from the test set after the original release, but this cannot explain the increase in recall since my algorithm did not make any predictions on this sentence. This difference may be due to bugs in the web scoring service or differences in the interpretation of the scoring rules (see below) between the two implementations.

Looking next at the interactions which Algorithm II missed, using the *med* pattern set, it is possible to make a few interesting observations. Of the 32 sentences which yielded at least one false-negative prediction, half relied on co-ordinating conjunctions (*and*/*or*). This indicates that more work on co-ordinated patterns and a thorough examination of parser behaviour on these sentences would be of benefit, especially since many of these unrecognized structures contained three or more actual interactions and thus affected recall severely. Almost a quarter of the problematic sentences contained an agent or target in an appositive structure like *another gene (X)* or *Y, a protein*, suggesting that this is another area worth investigating. The fact that the *max* pattern set

missed as many of these particular constructions as the *med* set, despite having 200 times the number of patterns, indicates that the expansion rules used in *max* were not effective at generating variant patterns covering these phenomena and will need to be revisited. Some of the false negatives, but surprisingly few, were due to unexpected words such as *occur*, *mapped* or *utilize* governing an interaction. The narrowness of the vocabulary encountered was helpful in this regard.

Subtle differences in the semantics of a 'true positive' between different experiments are common in NLP research, and inspection of the annotated test set provided two distinct illustrations of this. Firstly, three sentences in this set contained doubled interactions between the same agent and target, for example when one protein *activates or represses transcription of different genes* by another. My algorithm collapsed multiple relationships in the same direction between two entities into single interactions, although in hindsight this was a mistake. The LLL criteria required these to be reported separately, and the precision problems which caused me to collapse them in Algorithm I (see Section 4.2.2) would probably not have applied to Algorithm II. The other example comes from a sentence describing three genes that do *not* interact, in a *neither... nor...* co-ordination. Whether these should be included or not (also discussed in Section 4.2.2) was not clear from the instructions; I chose to disregard them as this seemed intuitively correct, but it seems that the LLL organizers expected them to be reported.

Unlike Algorithm I, Algorithm II is not very tolerant of small errors in the graphs. Apart from errors that can be identified or predicted in the pattern development stage, a single mistake in a subgraph will stop the appropriate pattern matching to it. Perhaps that one reason alone will restrict it to a high-precision, low-recall niche, at least in its current form.

**Discussion**

Although the performance difference between *min* and *med* demonstrate the benefits achievable by a small number of replacement rules, these rules are currently rather limited in their abilities. They cannot change two separate parts of a pattern at once, for example, nor can they change one part of a pattern conditionally on the value of another part. Furthermore they cannot insert a new node at any point of a subgraph that isn't a leaf node, because they have no way to reattach the moved material with correct bracketing. Take for example the following two patterns:

```
pattern
VBN~~@AFFECTED
  ( agent @NOUN~~@AGENT )
  ( @NSUBJPASS @NOUN~~@TARGET )
end
```

```
pattern
VBN~~@APPEARED
  ( xcomp VBN~~@AFFECTED ( agent @NOUN~~@AGENT ) )
  ( @NSUBJPASS @NOUN~~@TARGET )
end
```

There is no way to turn the first into the second by applying a replacement rule, even ignoring whitespace. A rule could replace `@AFFECTED` in the first pattern with `@APPEARED ( xcomp VBN~~@AFFECTED` but that would leave a missing right parenthesis after the agent. Of course, one could write a rule that replaced the root and whole agent branch of the first pattern, but that may as well be a new pattern in its own right. In consequence, an insufficiently wide selection of seeds will leave parts of the syntactic space unreachable, no matter how many combinations of replacement rules are applied.

## 4.5 Algorithm III

In order to put the results of Algorithms I and II and the other groups in perspective, it is important to consider what can be achieved on the same task by a very simple approach—an 'intelligent baseline' using a minimum of common-sense rules. Group 2's high-recall entry in the original task provided a baseline of sorts, showing what could be achieved by predicting an interaction in each direction between each pair of entities in the same sentence, achieving a recall of 98.1 (due to entity pairs with multiple interactions in the gold standard) for a precision of 10.6. This is a somewhat limited benchmark as it provides no useful frame of reference for precision. Also, scores were only provided on the easier subset of sentences, which did not include trickier grammatical phenomena like co-reference and ellipsis. To address this requirement, I designed a simple algorithm which does not take into account any grammatical information at all—constituents, dependencies or even POS tags.

### 4.5.1 Methods

The algorithm begins by locating each pair of entities in a sentence which have an interaction verb between them. Proximity is not taken into account, so there can be any number of words (including other entities) between the interaction verb and each entity. The verb list was extracted from the list of verbs used in the MPL files for Algorithm II. The algorithm then predicts an interaction for each of these pairs. The entity before the verb is used as the agent, unless the verb ends in *-ed* and is followed immediately by the word *by*, in which case the entity after is assumed to be the agent. This algorithm

| Scores on LLL corpus for Algorithm III | | | |
|---|---|---|---|
| Sentence set | $P$ | $R$ | $F$ |
| Training (original) | 38.8 | 30.3 | 34.0 |
| Training (expanded) | 35.0 | 30.3 | 32.5 |
| Test | 53.5 | 45.7 | 49.3 |

Table 4.13: Algorithm III's relationship extraction scores (precision, recall and F-measure) on LLL corpus, training and test, hardest subtask. The original training set is that which is supplied by the LLL organizers, without modifications; the expanded set also includes the 77 extra sentences without interactions which were gathered from MEDLINE.

took less than a day to design and implement, after which I ran it on the full training set (both with and without the additional non-interaction sentences) and on the full test set (via LLL's web scoring service for consistency with prior results).

### 4.5.2 Results and discussion

The results are shown in table Table 4.13 and are somewhat surprising. Although the training set scores were unremarkable, the result on the test set were much better, contrary to my expectations. A decrease in performance when moving from training to test is to be expected if knowledge of the contents of the training set has been used to guide the development of a solution to the task, as in Algorithms I and II, due to over-fitting to the idiosyncrasies of the training set. Such an *increase* in performance is harder to interpret clearly, but it does tell us several things.

Firstly, the scores on the test are very impressive given the simplicity of the algorithm. Comparing F-measures alone, Algorithm III is better than any of Algorithm II's runs, all but one of Algorithm I's, the only team in the original challenge which attempted the hardest version of the task, and indeed all but three of the official runs on *any* version of the task. Although it is not certain that a system optimized for F-measure is necessarily the best solution for a real application (see Section 5.2.2), this result nevertheless casts the previous results in a new light. One might use it to question the wisdom of using complex methods for such a task, and especially inductive-logic approaches like the majority of the entries in the challenge—at least with such a small training set. Of course, the much higher F-measure (72) achieved by Fundel *et al.* (2007) is an obvious counter to this question. A more reasonable interpretation would be that acceptable results can be achieved quickly on an IE task with a minimum of effort, and that from an implementation perspective, this means a working system can be constructed quickly and put into production while more sophisticated methods are perfected.

Secondly, it follows from this result that the training and test sets do differ sig-

nificantly in terms of the distributions of syntactic phenomena used to express their interactions. Since the algorithm was designed without any particular reference to the training set and thus had no opportunity to suffer over-fitting, one would have expected roughly equal scores on the training and test sets, if their syntactic composition was similar. A potential source of 'design bias' still exists, given that the list of interaction verbs was partly based on the training set, but of course any small effect here would be in favour of the training set results, rather than against them.

Finally, the precision of the algorithm on sentences without interactions is very impressive, given its simplicity. Out of the 77 new sentences in the expanded set which contained no interactions, the algorithm only produced false positives in four of them, although this must be tempered by the fact that out of the original 77 training sentences with interactions, it failed to predict any interactions in 35 of them. However, taking these numbers together still means that the algorithm achieved $P$ = 94.8 and $R$ = 67.6 at the simpler task of detecting which sentences contained interactions at all (regardless of the actual interactions predicted). A potential application of this algorithm, then, is as a pre-filter before using a noisier second pass to actually extract interactions. A simple step like this would have almost certainly benefited the test set precision of any of the original teams in the LLL Challenge, who trained their systems on data with no non-interaction sentences.

## 4.6 Concluding remarks

Despite its best F-measure on the LLL test set being worse than Algorithm I's and Algorithm III's, I nonetheless believe Algorithm II has much more potential for development. It seems further exploration of Algorithm I's parameter space is unlikely to yield much better results, and Algorithm III is of most use as a comparative baseline, a sentence classifier or a stopgap solution. Furthermore, there are obvious extensions to Algorithm II's method that would not be hard to implement. The shortage of seed pattern coverage could be addressed by using a larger dataset for development, and automatic pattern induction methods could then be explored next to manual construction. Comparison of pattern recognition performance between noisily parsed text and human-annotated text would test the theory that minor graph errors are responsible for a disproportionate number of missed interactions.

The recall issues could be addressed by extending the syntax of MPL to allow matching of lemmas (canonical forms) as well as literal words and regular expressions. Further benefits could be achieved by using similarity measures such as tree or graph kernels to allow probabilistic identification of subgraphs indicating relationships, based on the degree to which they resemble the patterns supplied, but without requiring exact matches. This kind of solution would have some aspects in common with a different paradigm in information extraction, which eschews explicit syntactic patterns

in favour of feature sets describing the syntactic phenomena present in a sentence at various levels of granularity, and classifiers trained to detect combinations of features characteristic of genuine relationships. This approach has made considerable headway in mainstream NLP research where training data is much more widely available (e.g Culotta and Sorensen, 2004; Bunescu and Mooney, 2005; Zhou *et al.*, 2007); similar methods are beginning to appear in biomedical experiments, with varying degrees of success (Xiao *et al.*, 2005; Eom *et al.*, 2006).

Similarly, an approach based on inductive logic like that of Group 6 in the LLL Challenge could also be applied to Stanford-style dependency graphs, and would be more likely to flourish without human-assisted input if it had more training data. Thankfully the recent release of a large (by LLL standards) corpus with integrated semantic and syntactic information in Stanford dependency format (see Pyysalo *et al.*, 2007b, and Section 5.1.2) gives these ideas the opportunity to take shape. One must also remember that the size of the LLL test set is small (87 sentences) and the opportunity to do more thorough testing on more data, including $n$-fold cross-validation, will help to elucidate the robustness and limitations of each approach further.

Taking the broader view, an important factor in Algorithm II's favour is its extensibility. Developers can begin adapting it to new domains by learning the syntax of three simple language constructs, and typing in some vocabulary and a few seed patterns. Nothing in its design ties it to gene regulation or genomics in general, or to bi-directional or two-entity relationships. Construction of patterns matching predicates with multiple agents or targets would be straightforward, as would the extraction of other semantic roles such as 'source', 'manner' or 'instrument'. Defining a new pattern list requires no coding skills, although the matching engine has the capability to accept post-processing modules written in Java should functionality outside the scope of the Metapattern Language be required.

# Chapter 5

# Conclusions and discussion

This project spans four years of work on natural language processing in the biological domain, with its roots in the two years prior to that during which I was working on biomedical document management and information retrieval. Over that time, the field has grown at a remarkable rate (see Figure 1.1) and has changed significantly with the introduction of various corpora and benchmarking standards, several regular workshops and meetings, and a diverse range of freely-available tools and services, so that one need no longer be an NLP specialist to engage in text mining in the biological domain. However, it is still a young field, the problems tackled by biomedical NLP researchers are far from solved, and in many cases, the best avenues of inquiry are yet to be determined.

Each of the preceding chapters has described a self-contained experiment in biological NLP, with an overall focus on parsing and its applications to information extraction. Discussions of the results for each one along with pointers to related work and other relevant remarks have been presented already. In this chapter, I will begin by evaluating the project as a whole with reference to its stated aims and highlighting some important lessons learnt. I will then analyse some important aspects of system design that are rarely if ever discussed, particularly in experimental contexts—the social and epistemological factors that can make all the difference between irrelevance and usefulness, and between theory and application.

## 5.1 Outcomes of this project

As stated in the Section 1.11, the objective of this project was "to improve the current state of the art in biological IE with the application of novel methods grounded in computational-linguistic principles." Already, the appearance of several citations suggests that this has begun to happen. At the beginning of this project, there had been very little recent research on syntax in biomedical text, but the published versions of

my two parser evaluation experiments (Clegg and Shepherd, 2005, 2007a) have been identified as the forerunners of a recent spate of papers (Cohen *et al.*, 2007)—one of which (Pyysalo *et al.*, 2007b) was directly inspired by our advocacy of the Stanford dependency scheme as a syntactic *lingua franca*. The authors also adopted the Charniak-Lease parser on account of our success with it. We have been cited in three papers on the adaptation of various kinds of NLP tools to the biomedical domain (Buyko *et al.*, 2006; Hara *et al.*, 2007; Pyysalo *et al.*, 2006b), one on adapting the Charniak parser to a different corpus of general English (McClosky *et al.*, 2006), and one on the analysis and reconstruction of coordinating conjunctions in GENIA (Shimbo and Hara, 2007). References have also appeared in a bioinformatics textbook (Sætre *et al.*, 2006) and a PhD thesis on medical text summarization (Elhadad, 2006).

Clearly, then, there is a burgeoning wave of interest in tackling biomedical-domain problems with syntactic parsers, and this project was at the forefront of it. It provided the first thorough, competitive and impartial analysis of the effects on parsers of the differences between biomedical and general English. It also presented some alternative syntactic evaluation techniques that, I would hope, are still in use long after the actual evaluations they were designed for are obsolete. The most far-flung reference to any of this work, however, came from the field of English-Chinese machine translation, much to the surprise of everyone involved. The paper was titled "Learning to parse bilingual sentences using bilingual corpus and monolingual CFG" (Huang and Chang, 2006), and was concerned only with the texts of news reports; the authors had used my statistics on production rule frequencies in the PTB to train a parser. It is gratifying to know that one's work can be useful to researchers so far from one's own speciality, especially when they have serendipitously made use of some minor supplementary data that was only published for the sake of completeness.

On that note, the original experiment from which that data came (Clegg and Shepherd, 2005)—the predecessor of Chapter 2 of this thesis—aimed not only to evaluate the performance of several treebank parsers side by side, but also to try several voting and parse combination techniques to build a consensus parse that was more accurate than any one of the parsers could achieve individually. Some of the results were statistically significant but too marginal to be of any practical use, so I decided not to include that part of the experiment here as the work required to bring it up to date would have been out of proportion to its practical value. However, it remains of theoretical interest and could plausibly be applied to other problems. The paper also contains some potentially useful statistics on GENIA and the PTB, and the differences between the two, and is archived for reference at `http://biotext.org.uk/`.

The work on the LLL corpus was not published before the completion of this thesis, and now that the BioInfer corpus (see below) has been made available—and indeed tailored to our specifications—those results may never be published, in their current form at least. The sophistication of BioInfer's semantic annotation makes the untyped binary

relationships in LLL seem slightly outdated. Nonetheless, Chapter 4 accomplished what it set out to do by demonstrating the practicality of an IE framework based on treebank parsing and the Stanford dependency tools. Algorithm I outperformed most of the systems used in the original challenge, without using any hand-edited linguistic data or artificially segregating the sentences into different classes—in terms of achievements relevant to real world applications, it came out on top. This was despite the fact that its simplistic mapping of dependency structure to semantic structure was guided by a process that can best be described as high-throughput trial-and-error. This ease of implementation reflects the intuitively graspable and computationally tractable nature of the dependency graphs. Algorithm II's more subtle and principled approach was hampered by lack of development time and annotated data to base examples on, but some opportunities for expanding its scope are discussed below. Furthermore, its extensible and open-ended design philosophy will, I hope, encourage other researchers to adopt it and adapt it, and find uses for its high-precision capabilities. As with all the code written for this project, it is available from the URL above.

Although Algorithm III's non-syntactic strategy proved surprisingly effective, and will perhaps be of practical benefit (see below), the fact that all of my results were beaten by a system using a treebank parser, Stanford dependency graphs and handwritten rules (Fundel *et al.*, 2007) is somewhat reassuring. It vindicates my hypothesis that this previously untried combination holds potential for biomedical IE. LLL was a small and highly constrained task, however, and it remains to be seen how transferable any one implementation of this strategy is to the next generation of data.

### 5.1.1 Implementation recommendations

Although testing new algorithms on standardized benchmarks is vital, the ultimate goal of bioinformatics research must be to develop tools or methods that can be usefully applied to real data. The previous chapter produced two important practical recommendations. The first is that a naïve, knowledge-poor solution built in under a day can tackle a simple genomic interaction extraction task with results comparable to the most sophisticated of methods. The steps involved (given a list of interaction verbs) are summarized as follows:

1. Begin with a corpus of sentences without any metadata.

2. Process the text with a named-entity recognizer and replace any entity names found with unambiguous single-word placeholders. Named entity recognition is not the focus of this thesis, and indeed this step was trivial with the LLL dataset, but a discussion of this task and a practical walk-through is provided in Appendix A.

3. Mark every pair of entities in a sentence with an interaction verb falling anywhere between them.

4. Predict an interaction involving this pair of entities.

5. If the verb ends in *-ed* and the next word is *by*, the entity after the verb is the agent; else, the entity before the verb is the agent.

However, this approach is limited to very simple tasks and cannot be easily tuned for precision or recall. If a high-precision solution is required, or interactions more complex than simple directional pairs are sought, then a suitable pipeline can be built using only freely-available components and a little engineering. The steps are summarized below:

1. Begin with a corpus of sentences without any metadata, and mark all entities as above.

2. Parse the text with the Charniak-Lease parser and chunk all multi-word noun phrases in the parse trees into single nodes.

3. Translate the trees into dependency graphs using the Stanford NLP toolkit.

4. Design or adapt MPL patterns to capture the relationships of interest in your domain. For tasks of a similar nature to the LLL Challenge, it is likely that the patterns developed for this project will yield reasonable results.

5. Search the dependency graphs for matching subgraphs using the algorithm described in Chapter 4, recording the agent and target of each successful match.

If a high-recall solution is required, this may be achieved by asserting an interaction between each pair of entities in a sentence; the low precision of this method could be somewhat ameliorated by ignoring all sentences which don't contain an interaction keyword anywhere. The interplay between precision, recall, degrees of certainty and notions of truth will be returned to in the following sections.

All the source code written for this thesis is available from `http://biotext.org.uk/`. This includes Java implementations of Algorithms I and II from the previous chapter, and a Perl implementation of Algorithm III, along with additional materials such as MPL definitions, and will be refined and added to as new developments take place. I am of course keen to hear suggestions or criticisms.

### 5.1.2 Opportunities for future development

During the course of this project, I was somewhat frustrated by the lack of cohesiveness of the annotated corpora available in the biological domain. For example, GENIA has

named entities and POS tags, as well as constituent trees for a subset of sentences, but the treebank is in a different set of files from the entity annotation—and my attempts to merge them were thwarted by overlapping boundaries and discrepancies in tokenization and tagging. The same is true of BioIE, which at least is kind enough to flag sentences with inconsistencies between the two annotations, but it has a subtly different treebank format and set of annotation guidelines from GENIA, making interoperability difficult. Neither GENIA nor BioIE have semantic relationships between entities. LLL's enriched datasets have gene and protein entities and the relationships between them, word lemmas, and syntactic structure, which is definitely progress. However, they don't have POS tags, just five 'morphosyntactic categories' which are very broad and not rigorously applied. The syntax annotation is based on the Link Grammar, which is unorthodox and not directly compatible with other tools or formalisms, and has been simplified by the LLL organizers so it no longer even corresponds with Link itself. And LLL is small by the standards of other corpora (77 sentences in the training set). These limitations constrain both the flexibility of the evaluation protocols that can be performed, and the techniques available to perform them.

The 1,100-sentence BioInfer corpus (Pyysalo *et al.*, 2007a), which was released while I was testing Algorithm I, addressed most of these issues and many other serious deficiencies in the previous generation of corpora. It contains unified annotation for syntax, entities and relationships, as well as coreference and ellipsis phenomena. There are no POS tags unfortunately, but tokenization is fairly sophisticated, with semantically distinct subtokens as in *Arp2/3* marked. Entities and relationships are both ontologically typed, and annotated to the best level of specificity available, unlike LLL which only has one type of entity and one type of relationship. Processes and functions are entities, as are some abstract concepts like amounts and expression levels. Relationships can have more than two arguments, or even just one, and each instance of a relationship is treated as an entity and can be used as the argument to a predicate:

`MEDIATE(`*VEGF receptor 2*`,` `ACTIVATE(`*neuropilin 1*`,` *VEGF165 function*`))`

This means that *VEGF receptor 2* mediates the activation of *VEGF165 function* by *neuropilin 1*. The relationship ontology also contains some less concrete relationships like `EQUAL`, `CONTAIN` and `MEMBER`.

Sampo Pyysalo from the BioInfer group contacted me just before the release of the corpus, expressing interest in my work on the Stanford tools, and a few months later an updated version of the corpus was released with all the Link graphs translated into Stanford graphs (Pyysalo *et al.*, 2007b). It is unfortunate that this impressive resource was not available at the start of this project, as its completeness and logical structure

surpass any of the other corpora available (including those I have seen in 'mainstream' NLP), and it models biological reality more closely. In short, it will make a much more suitable proving ground for further work on the algorithms and concepts presented here.

Both the parser evaluation/selection stages and the algorithm development stages of this project could have benefited from access to BioInfer. At the moment one can only assume that the performance of the Charniak-Lease parser on LLL is roughly the same as it is on GENIA, since LLL's syntactic annotation is not directly comparable with GENIA's or the parser's output. The only way to test it would be to attempt to convert LLL's Link-like annotation to the Stanford format using BioInfer's conversion program, which would introduce unquantifiable noise anyway. Another source of noise in the current experimental pipeline is the Stanford algorithm itself, since without painstaking examination of the trees and graphs, it is impossible to see how many errors—or at least unexpected decisions—it is making on either the GENIA gold standard or any of the parsers' outputs. In BioInfer, the Stanford annotations are manually corrected, so the parser plus the Stanford algorithm could at least be tested jointly against a gold standard of truth. However the lack of POS tags would remove one method of error analysis.

Moving on to semantics, it is easy to suggest strategies for exploiting the additional richness and size of BioInfer. The availability of entity tags is thought-provoking, especially if the parser is also combined with a high-quality named entity recognizer. It should be possible to learn from the corpus the probability of finding each dependency type governed by each entity type; then a selection of parses could be ranked according to their level of agreement with the entity recognizer. Parses could also be discarded if a constituent crossed an entity (like the crossing brackets measure in Chapter 2) or if the head word of an entity was given a non-noun POS tag. Alternatively, one could use several entity taggers in parallel, and pick the combination of a parse and a set of predicted entities that had the highest joint probability of appearing together. This kind of operation would probably only work with conventional biomolecular entities, not for example properties of other entities or implicit process entities representing relationships. Some of these suggestions might be workable on GENIA too if the syntax and entity annotations were synchronized, and possibly even on LLL in a more limited sense. Typically, NLP systems are seen as pipelines made up of distinct modules that are designed, built and tested separately, and which make decisions independently and in series, but the availability of corpora with integrated semantics and syntax might challenge that presupposition.

The factors of size and richness would obviously pay dividends when adapting or replacing the IE algorithms from Chapter 4. Algorithm II would be a natural choice to work on further, as the availability of more example interactions at the pattern design stage would enable the recall to be improved, and potentially allow automatic acqui-

sition of patterns tested by cross-validation. One advantage of Algorithm II is that I designed it to avoid making too many assumptions about the task, so it does not require that patterns have exactly two entity roles. It would need to be modified to ignore POS tags, but that would be trivial, and whether or not noun phrase chunking would be beneficial or even straightforward is unknown, given that it currently works on constituents and POS tags. There are other features of the BioInfer scheme that could be exploited instead, such as the fact that relationships are anchored to the word or words that describe them; in LLL they are more like additional arcs that connect the two entity nodes without touching the rest of the graph.

Of course, having a larger, cleaner gold-standard to train on might also make statistical approaches like those used in the actual LLL Challenge feasible, or other techniques not yet imagined. Named entity recognizers will need to be modified or combined to spot nested entities. How to deal with nested relationships acting as conceptual entities though is very much an open question, and one that will be as challenging as it is fascinating. It seems likely that a number of these will correspond neatly to subgraphs, and clauses in the parse tree, as in ***induction of X by Y*** *inhibits Z*. How many won't, and how to process them, will require careful analysis and planning and will certainly entail new science.

### 5.1.3 Practical lessons learnt

While it is true that more complex and expressive corpora will require correspondingly sophisticated algorithms, NLP projects also have a tendency to stall in unexpected places because of unforeseen problems at less glamorous levels. Seemingly trivial implementation details like character sets, file formats, tokenization, punctuation conventions and alternative representations for special characters can all cause a disproportionate amount of extra work. This is particularly true when performing a broad comparison of several different tools. Quite often, two systems will produce (or require) linguistically-compatible datasets that are nonetheless very different from a program's perspective. A few representative examples, which correspond to weeks of ugly and unrewarding scripting and bug-hunting, will illustrate this point.

Although parsers take essentially plain text and turn it into syntax trees, this is not exactly true. Each parser has slightly different requirements for the presentation of that text (leaving aside the question of whether the text must be POS-tagged already or not) and these can cause serious issues. In the initial run of the experiment in Chapter 2 (Clegg and Shepherd, 2005), an earlier (2003) version of the Charniak parser was used. As well as assigning POS tags internally, this version also performed its own tokenization on the input text. While this does not immediately seem like a potential source of error, many atomic terms in the GENIA corpus have internal punctuation, causing spurious token boundaries. These include such examples as *5′*, *SKW6.4*, *1,25(OH)2D3*,

*mol/l* and *IL-1/tumor*. Indeed, there are several similar examples in the PTB, such as *one-shot*, *O'Brian*, and *US$*, and inverse examples like *75 %* which the other parsers (and the gold standard) considered to be two tokens, while the older Charniak parser 'glued' them back together into one. In order to sidestep Charniak's tokenizer and ensure that the tokens in each parser's output aligned with each other and with the test data, these strings were converted into strings of characters (such as *XcommaX*) before passing them to the Charniak parser, and back again once the parse was complete.

However, it was brought to my attention eventually (M. Lease, Brown U., pers. comm.) that my scores for this parser on the PTB were slightly lower than previously-published results. This seemed to be as a result of these punctuation substitutions leading to inconsistent POS assignments for the words they occurred in, and hence errors in the resulting trees. Although a full analysis of this phenomenon was deemed to be outside the scope of this investigation, upgrading to the 2005 versions of the parser led to scores on the PTB that are consistent with those reported by other authors. These versions include a command-line switch to disable the tokenizer, thus telling the parser that the character stream should be split up only on whitespace. This was used in all the runs in Chapter 2. By then, however, the damage—in terms of lost development and testing time, and discrepancies in published results—had been done. While this is an extreme case, practically all NLP tools have certain characters in the input data that must be escaped or replaced with placeholders, and the list of characters in question (and the correct way to deal with them) is not always documented.

These experiences support the argument for modularizing NLP tools, as does the improved performance on biomedical text that was achievable by decoupling the POS tagger from the parser in the Charniak-Lease parser compared to the original Charniak versions. However, there is a difference between a single tool that performs several functions in a pipeline, which loses nothing by decoupling those functions, and a tool that makes predictions or resolves ambiguities in different levels of annotation in parallel, via iterative processes or joint probability distributions. I suspect that the availability of corpora with integrated syntactic and semantic annotation will facilitate a new strand of research in this area.

Similar character-level issues have been widely observed in a current project in collaboration with two MSc students, who are using Algorithm I from Chapter 4 for reconstruction of regulatory pathways from corpora of full-text papers. The papers have been, for the most part, converted from PDFs using automated tools, and the resulting poor text quality is the source of many errors. Hyphens at line breaks are not always removed by the conversion process, so words that were split across lines in the PDF have hyphens in the middle. Furthermore, sometimes these are not 7-bit ASCII hyphen-minus characters but more exotic glyphs that cause certain NLP tools to fail without error messages. Spaces are sometimes removed from sequences of words, making parsing completely impossible, and typographical ligatures for *fi* and *fl* are

not always converted back to their constituent characters. Orthographic issues aside, the content of full-text articles leads to problems for tools trained or developed on the cleaner world of MEDLINE abstracts. Authors' names can look very much like gene or protein names to named entity recognizers, for example, as can many other strings found more often in whole articles than in abstracts—particularly other proper nouns like strain identifiers, locations, company names, and trade names of reagents or equipment. These problems are not glamorous, but while NLP tools are trained and tested on abstracts alone, their uptake amongst the biomedical community will be limited by such accuracy and robustness failures.

The issue of robustness in the face of 'dirty' data is a broader one. One way of characterizing Algorithm II in Chapter 4 is that its pattern set was derived from the modeling of correct and familiar subgraphs, with the addition of replacement rules allowing it to relax its constraints in a controlled manner to allow correct but unfamiliar subgraphs, and only allowed for actual errors on a case-by-case basis. Contrast this to Algorithm I, which even in its most methodical first pass did not make any assumptions about the linguistic accuracy of the dependency graphs it was operating on, and provided various fallback measures to increase recall of initially missed interactions—without distinguishing between errors in the data and deficiencies in its own approach. Contrast both of these approaches to RelEx (Fundel *et al.*, 2007), which starts with as many candidate interaction paths as its heuristics have found, without any explicit assumption of the underlying accuracy of the parses or graphs, and then narrows them down by the elimination of typical false positives. Both Algorithm I and RelEx suffered drops in F-measure (mainly down to recall) when going from training to test, but RelEx's was much more gentle, and neither of them were comparable to the massive drop encountered by Algorithm II.

Although one must be wary of such simple caricatures of complex systems, the different requirements they had for the linguistic correctness of the input data—from a great deal in the case of Algorithm II, to none at all (explicitly) in the case of RelEx—may well be instructive. As we have seen, the potential sources of error are many and varied, and in a production system can be rooted in the very earliest stages of processing. Although there are potential applications for a high-precision system (see Section 5.2.2), even such tailored solutions will still need to account for a degree of error in the data, although the LLL test set seemed to give most teams less problems for precision than for recall. As an aside, the robustness problem provides even more support to the criticism that optimizing a tool for perfect hand-corrected data (c.f. Group 6 in Section 4.1.2) is rather pointless.

## 5.2 Social aspects of text mining

As social activities, biomedical NLP and text mining have certain requirements and connotations beyond their immediate scientific aspects. The technical aspects of NLP are well studied, but text mining as a human activity raises several interesting points that can guide the design of applications and inform the development of useful experiments and evaluation criteria. In order to demonstrate some of these points, I will introduce two distinct social groups that stand to benefit from biomedical text mining research, and introduce some examples of different uses to which they would put NLP technologies in the course of their work. I will use these hypothetical situations to illustrate some important distinctions between the needs and goals of these two groups, and discuss how better to align basic NLP research with them. Note however that very little of the important points about these two communities is unique to them; these examples are here to help promote the general aim of understanding the relationships between technologies and their users.

### 5.2.1 Typical user communities

Over the duration of my PhD and some time spent working on IR before that, two groups of biomedical researchers have particularly stood out in their public enthusiasm and support for text mining—genomic database annotators and curators, and drug discovery scientists. Several notable biomedical NLP meetings have had invited talks from the database curation (e.g. Blake, 2005) and pharmaceutical (e.g. Vachon, 2004) communities, and in conversation I have found both groups to be very positive on the subject. Interestingly, the same two specialist communities were originally jointly responsible for Gene Ontology, which started out as a collaboration between FlyBase, Mouse Genome Informatics and the *Saccharomyces* Genome Database (Ashburner *et al.*, 2000), and was funded by AstraZeneca before any public grants were awarded (M. Ashburner, pers. comm.). Since then, GO has become an indispensable tool for bioinformatics and systems biology, and the concepts associated with ontologies in general have become familiar and widespread in the field. However, these two groups typically have somewhat different requirements for NLP systems, so I will begin by discussing some typical scenarios.

#### The pharmaceutical sector

An important part of today's rational drug discovery process is the pursuit of a greater understanding of the genomic and biomolecular mechanisms of disease and health. Alongside structural biology and biophysics, the functional insights yielded by analysis of the body's regulatory, signalling and metabolic pathways are crucial to the selection of novel drug targets, and can shed valuable new light on the biological roles of existing

targets which enables them to be drugged more effectively or with fewer side effects. Indeed, similar techniques are applied retrospectively to investigate the hugely expensive failures of once-promising drugs that can severely affect a company's reputation and share price. NLP techniques are ideally suited to this kind of pathway analysis as the required facts might very well be 'known' already, albeit spread across diverse publications spanning multiple topics and decades of research. Also, as we have seen, information extraction algorithms can produce output in the form of a graph or network that maps easily onto the underlying biological pathways.

A typical scenario might involve a set of genes that are observed via a microarray experiment to be expressed differentially in healthy and diseased tissue; some of these genes are modulated by a potential drug. For this information to be of use, a map must be made of the causal mechanisms connecting them together. Some of the connections might be standard textbook knowledge and some might be available from existing databases of protein-protein, gene-gene or transcription factor interactions, for example. However, one would be very lucky to reconstruct the entire network like this, without recourse to a tedious trawl through the literature to fill in the gaps. This is one point of entry for the pharma industry's interest in NLP research. The collection of potentially relevant publications may well be large and diverse, spanning genetics, biochemistry, cell biology, pharmacology, and perhaps even physiology, and any investment in technology which leads to saved time will pay for itself eventually. Of course, this scenario is by no means unique to industry, but the scale of automation available to microarray experiments and other high-throughput assays in the pharma sector makes managing the associated literature a high priority.

Another NLP application in the pharma and biotech sectors, and one which is perhaps less familiar to many researchers in academia, is the mining of patent databases. Before embarking on any major discovery project, or even repurposing an existing therapy for a different disease or class of patient, it is imperative to identify any problematic overlaps with existing patents. Conversely, competitors' patent filings are also used in a business intelligence context, as they can reveal or at least suggest what products might be in the pipeline and at what stages of development.

**-omic database curation**

Keeping a post-genomic database resource such as UniProt or FlyBase up-to-date is a major undertaking, since there is no obligation for bioscience researchers in many fields to deposit their own data or annotations in such a public resource prior to publication, and the rate of new publications keeps increasing. Accordingly, curation assistance tools have been a major focus of biological NLP research for several years now. The KDD Challenge Cup in 2002 was the first organized public benchmarking of NLP techniques for the the curation process, and was notable for bringing together language

engineers and biologists to design a task based on on the needs of the biologists. The goal was to simulate the FlyBase triage process as accurately as possible. This is the painstaking but necessary task of examining every new paper from every potentially relevant journal each month, pick out those that contain experimental results about *Drosophila* genetics and rank them by relevance. Then the curators must look at each individual gene mentioned in the paper and determine whether an observation about its expression was made with sufficiently strong evidence to warrant inclusion in FlyBase.

The KDD Challenge Cup was a success; 18 teams entered and there were some remarkably good scores on some of the performance metrics (the winners scored 84% on the the relevance ranking test). The following year, two more challenges followed in its wake. The TREC Genomics Track was a more traditional IR competition, designed to encourage the development of topic-specific algorithms for judging the relevance of MEDLINE abstracts to query genes. This project was begun with the biomedical community as a whole in mind. BioCreative, on the other hand, looked at two specific tasks vital to the next stage of curation, which is distilling unstructured information about genes and their products in free text into a structured annotation suitable for entry into a database. This is potentially a very involved process which could throw up several unsolved or partially-solved problems in computational linguistics—depending on the complexity and variability of the annotations required, it could entail a degree of natural language understanding which is beyond current systems. Nonetheless, the BioCreative organisers recognized the successful detection and identification of gene and protein names as a fundamental requirement of any such system, and so set such a task using text about fly, mouse and yeast genetics; F-measures above 80% were achieved on some parts of the task.

The second task was somewhat more adventurous and open-ended. It involved the assignment of GO terms to human proteins, and seemed to divide the competing teams into a high-recall camp and a high-precision camp. In their summary of the results, the organizers said:

> "The majority of users tried to submit a result for each case contained in the test set. Those approaches focused on obtaining a high recall rather than a high precision. On the other hand, there were users who submitted results only for a small number of high confidence predictions to achieve a high precision. Although for practical use of text mining applications, high precision is desirable, a reasonable recall is essential, consequently a compromise between both should be favored." (Blaschke *et al.*, 2005)

### 5.2.2 Criteria for success

It should be apparent from these brief sketches that NLP and text mining projects aimed at different user communities can have different criteria for success, and there-

fore should not necessarily be evaluated in the same ways. This project has taught me the importance of well-designed, task-based evaluations. The computational linguistics community is happy to accept papers where, for example, an existing parser has been tweaked to give it half a percentage point lead in constituent F-measure over the previous best algorithm (Charniak and Johnson, 2005), but in an applied field like biomedical NLP such tiny incremental gains are of no interest unless they can provide a practical benefit—if in fact they represent genuine error corrections at all (see Section 3.1).

Even within the realms of evaluations based on real-world tasks, there is still a tendency towards blanket use of balanced F-measure as the key performance indicator, for almost every kind of NLP software benchmark. Balanced F-measure is convenient and intuitive—I cannot deny having relied on it in this thesis—but is taken to be the *sine qua non* of evaluation for so many diverse tasks that it is easy to forget that this venerable measure is simply a special case of the weighted harmonic mean of precision and recall, with the weighting constant $\alpha$ set to 1:

$$F_\alpha = \frac{(1+\alpha)PR}{(\alpha P) + R} \tag{5.1}$$

One rarely sees an 'unbalanced' F-measure outside of old information retrieval textbooks these days (van Rijsbergen, 1979), and while a specialist user of an IR system might once have wanted to tune this behaviour on a search-by-search basis, none of today's consumer IR services like search engines seem to offer this facility.

However, outside of the constrained environments of benchmarking experiments, it is not at all clear *a priori* that giving an equal weight to precision and recall is always necessary or even desirable. Obviously, the world would be a much better place if NLP systems could consistently achieve 98% precision *and* recall. But given that this is currently impossible for most tasks, is it preferable to score (say) 98% on one and 48% on the other, than to hover around the equivalent 64% for both? The answer depends on the application and the needs of the users, and the 'one-size-fits-all' reliance on balanced F-measure obscures these subtleties. The quote above from the BioCreative team recommends a compromise, and strongly recommends against favouring precision in such tasks, but I would like to look at this issue in more detail.

#### Partially automated systems

There were 1,227 real annotations in the BioCreative GO term annotation test set, spread across 99 documents (Blaschke *et al.*, 2005). Despite this, the highest-precision run was so conservative with its predictions that it only returned 26 for the entire test set (Chiang and Yu, 2004), and several others were of a similar order of magnitude. A curation tool that is so precision-driven that it only makes a few percent of the predictions

required will make no friends amongst annotators—even before they have counted the (admittedly small) error rate on those predictions.

The organizers were correct to say that a reasonable recall is essential in this instance. I would go further, however, and argue that because the kinds of tools that might come out of a project like BioCreative would be designed to assist, not replace human annotators—at least in the short and medium term—recall is considerably *more* important than precision. The human expert in the loop makes a difference to the kind of errors that are forgivable, simply because manually removing a false positive (a precision error) is intrinsically less work than manually correcting one that is close but wrong, or manually adding one that is missing (a recall error). The algorithms under test were designed to simulate the needs of a visual curation tool which would be able to display a GO annotation and the text supporting it, thus drawing attention to errors. Once an annotator has noticed that one of the predictions on a document is spurious, removing it is trivial; however, to fix a false negative requires first noticing the omission (which may or may not be conspicuous by its absence) and then to work out what the correct annotation should have been.

Obviously, for equal recall scores, higher precision is better, but this seems to be an example of a situation where high recall and low precision is better than medium recall and precision.

**Recall and full coverage**

The need for good recall in database annotation tools is clear, but there are cases where achieving as close to 100% recall as possible is desirable, despite the corresponding drop in precision. The corporate patent search application described earlier is one such example, for business reasons rather than technical ones. Potentially only a single missed reference to a chemical entity, protein target, delivery mechanism or even a detail of a manufacturing process can lead to costly project cancellations and, potentially, lawsuits. Small details like orthographic variation in chemical formulae can have serious implications, if they result in the mistaken belief that a hit compound is patentable, and of course the well-known variability of gene and protein names (see Section 1.8 and Appendix A) can be a problem here too.

As in the database curation example, this kind of application is by necessity a semi-automated system as any results will need to be inspected by an intellectual property specialist. One way to ensure the best possible recall without swamping the user in false positives is to take a domain-specific IR approach, and rank instead of filter the results. Hits to substances or methods that the system has confidently identified as problematic should be presented near the top for immediate action; if they are genuine prior claims, then any less confident hits further down the list may be rendered irrelevant as the project might have to be abandoned or at least substantially revised. In

evaluating systems like these, single values for precision and recall are less important than precision-recall dynamics: what is the precision at a number of different cut-off points for recall? How many of the top $n$ documents are relevant?

**Recall through repetition**

There are, however, other text mining tasks where it is possible in principle to achieve good coverage even with a high-precision, low-recall algorithm, because of the redundancy inherent in the scientific literature. Take for example the genetic network reconstruction scenario described earlier. Unless the network in question pertains to a very obscure part of biology—unlikely for a pharma company, but not impossible—there will be no shortage of literature to feed into an interaction extraction program. MEDLINE, full-text journal articles, and resources from the NLM such as electronic textbooks and OMIM are obvious starting points, but internal tech reports and white papers, lab notes, and patent records (one's own and one's competitors') could potentially be relevant too. In these circumstances, high-precision/low-recall approaches come into their own. With a comparatively meagre recall of 40%, a relationship only has to be stated in three separate places in the corpus for its chance of being spotted at least once to reach 78%. (A 60% probability of being missed once implies a $.6^3$ = 22% chance of being missed three times on average, assuming uniform variation of syntactic expression.) It is not uncommon for the key claim or claims of a paper to be reported more than three times within the paper itself—title, abstract, introduction and conclusions—let alone citations, replications and parallel discoveries. Indeed, one might expect the claims that only appear once in a large corpus to be of somewhat dubious provenance anyway. Although simple repetition is not necessarily an indicator of veracity (see Section 5.3.3), one might not be too far wide of the mark to suggest claims repeated more times are more trustworthy, at least to a first approximation.

This issue is touched on by Giuliano *et al.* (2006), in the context of their results on LLL and other data. They distinguish between OAOD and OARD scoring, for 'one answer per occurrence in the document' and 'one answer per relation in the document' (when multiple claims repeated in the same document are collapsed into one). In this format, LLL's scoring rules would be described as OAOS or 'one answer per occurrence in the sentence', a stricter criterion. To this I would add OARC for 'one answer per relation in the corpus' for applications or projects where it is only vital that each relevant claim is identified once (or *at least once*) across the whole corpus. In terms of evaluating new methods, I believe the LLL Challenge's stringent requirements are justified, as they give an accurate comparison of algorithms at a granular level. However, for large-scale extraction projects with significant repetition in the corpus, it might be more fruitful to use a higher-precision, lower recall algorithm, minimizing false positives and trusting natural redundancy to assist recall.

### 5.2.3 Human interface factors

No discussion of the social aspects of computer systems of whatever sort would be complete without at least a brief mention of user interfaces, although this is not a topic that comes up with much frequency in biological NLP discussions. There seems to be an unspoken feeling in some circles that user interfaces are simply engineering problems at best, or unnecessary window dressing at worst, and the real science happens at the level of algorithms, data and statistical models. While it is of course possible to write software like this, as this thesis demonstrates, it must not be forgotten that the human brain is one of the most powerful pattern-matching resources we have access to, and perhaps *the* most powerful. The field of data mining has grown up hand in hand with that of data visualization, pushing user interfaces far beyond buttons and lists and into novel ways to render and explore complex multidimensional data. Text mining has just as much potential to be treated the same way.

## 5.3 Epistemological aspects of information extraction

The text of a scientific paper is not simply data, although it may present data and will likely seek to communicate some important piece of data or aspect of a dataset. Rather, it is knowledge encoded in written form, and like all human knowledge, might be incomplete, ambiguous, subject to unconscious bias, inconsistent, obscured by rhetoric or just plain wrong. Molecular interactions and other 'facts' extracted from text are often portrayed as having a simple binary truth value, right or wrong, and while this is a convenient approximation for evaluation purposes, it is rather naïve and may not be satisfactory in large-scale text mining projects. While there is no doubt an underlying reality which ultimately grounds our beliefs, there are at least two distinct levels of abstraction at work. Firstly, a statement in a text is not a fact or a truth but a claim about reality; secondly, any extracted information similarly consists of claims about a text or a collection of texts. In this section I will attempt to pry apart some of the causes of epistemological confusion in this area that are usually ignored, glossed over, or wrongly conflated. It is my belief that as the field matures, considerations such as these will become more and more important in the design of more sophisticated systems and evaluation protocols.

### 5.3.1 Uncertainty in the extraction process

No IE algorithm is, or claims to be, even close to 100% accurate, and most have been evaluated for precision and recall at some point, giving the user some degree of certainty in their aggregated results. (The term 'confidence' sounds more natural but risks confusion with the technical meaning from statistics.) However, very few systems give any indication of certainty on a case by case basis, even though this is often not that

hard. Algorithm I in Chapter 4, for example, records with every interaction the pass of the algorithm that captured it. This is a crude measure of certainty, but the users find it useful, since it means they can choose to filter out the more speculative and potentially error-prone interactions suggested by the later passes. Not all algorithms have analogous methods by which their internal decision-making processes can be meaningfully reported on, but levels of agreement amongst ensembles of algorithms can sometimes be used as a predictor (Clegg and Shepherd, 2005). It is easy to conceive of ways that this information could be made use of in mining and visualizing the results of large literature analyses. Of course, the value of this kind of information is proportional to the precision of the algorithm—or more generally, accuracy, in tasks that do not use the precision/recall model—since there is little benefit in being certain but wrong. Another way of presenting such quality control data is as a prediction about the chance of correctness of each extracted claim, based on prior performance in evaluation experiments. However, a better term than 'correctness' would be 'verisimilitude' since a correctly-extracted claim is not necessarily true.

### 5.3.2 Trustworthiness of the source

Not all claims in papers or abstracts are equal—the experimental evidence used to support a claim has an effect on the reception of that claim by the community. In recent years, for example, debate has raged over whether high-throughput experimental protocols offer the same quality of evidence as traditional assays, with some claiming accuracy as low as 50% for yeast two-hybrid experiments (Sprinzak *et al.*, 2003). GO uses a system of evidence codes[1] to (manually) classify the evidence used to justify assigning a particular GO term to a particular gene or protein; ideally, IE systems would be able to associate similar metadata with extracted claims, although this is probably several years away. Currently, most IE systems do not even determine whether the paper is repeating a claim made elsewhere or presenting a new finding, or even if the claim in question is framed by a figure of speech like *we wish to determine whether...* or *it is currently not known if...*. Between these statements of knowledge or intention, or outright claims of fact, there are a whole spectrum of speculative rhetorical devices such as *these results suggest that ...* which reflect differing levels of author confidence. Some analysis of these phenomena has been performed (Light *et al.*, 2004) but the results have yet to make their way into IE applications.

One can even imagine using meta-scientific information such as the quality of the journal in which a claim is being made as an indicator of its trustworthiness, by comparing impact factors, or the experience or reputation of its authors, based on number of previous publications. On that note, I believe many people would confess to feeling a little suspicious if all the evidence for a controversial claim came from the originating

[1] `http://www.geneontology.org/GO.evidence.shtml`

group. Such considerations are outside the scope of current research in IE, but these are common ways that scientists evaluate the credibility of new claims, and there is no reason why automatic systems cannot be designed to model these methods. The issue of evidence and traceability is important in other ways too, since it is bad scientific practice to propagate unsourced claims, and users will want to see an IE system's sources for any results that they base decisions on. This is irrelevant in experimental evaluations, but end-user applications must at least provide the facility to display a citation for each extracted claim and preferably a link to the specific sentence where the claim was found.

### 5.3.3 Confirmation and contradiction

It has been mentioned already (see Section 5.2.2) that in a large enough body of text, there is likely to be redundancy; however, larger corpora also have an increased chance of containing mutually contradictory statements, such as *X inhibits Y* and *X does not inhibit Y*. The only methodical analysis of the epistemic characteristics of the biological literature in an IE context was by Krauthammer *et al.* (2002), who performed a flawed but interesting time-based study of the repetition and contradiction of claims about biomolecular interactions, in order to allow contradictions to be resolved rationally in their own IE package. They developed a complex statistical model that aimed to be able to predict whether a claim was *really* true or false via maximum likelihood estimation, based on its patterns of agreement and disagreement over time and across the corpus. This daring and vaguely hubristic experiment was tainted with circularity from the start, as the ultimate truth or falsehood of the examples used to train the model came solely from the decisions of domain experts—even though one of their key points was that individual claims made by domain experts can't necessarily be trusted. To my knowledge, no tests of the model's predictive power have been published. They also showed that as the number of different interactions reported for an entity goes up, the repetition rate for each of those claims increases, and the rate of conflicting claims decreases. This was explained in terms of publication bias caused by a reluctance to go against the *status quo*, but could equally well just mean that it is harder to make mistakes when dealing with well-known and well-characterized entities.

One important point made by Krauthammer *et al.* though is that disagreement between two claims does not necessarily imply that one of them is false, since they might both be true under different experimental conditions. An inhibitor might be useless outside a certain temperature range, and a regulatory pathway may not be active in a certain strain of an organism, but current IE techniques generally do not attempt to extract and record such details. Dependency graph analysis might help with such cases, since prepositional phrases are often used to establish the context of sentential claims and Chapter 3 demonstrates the ability of a good parser to attach prepositional depen-

dencies accurately. In other cases, however, simple experimental errors or editing/peer-reviewing oversights allow mistakes to creep in, meaning that the issue of contradiction relates quite closely to that of trustworthiness as discussed above. The relationship between repetition and trustworthiness has already been discussed, and although it would not be too implausible to take the most often repeated of two contradictory claims as the more truthful, Krauthammer *et al.* asked whether 50 repetitions in low-impact journals outrank 10 citations in highly-influential ones, and claimed that their model would be able to answer questions like this on a case-by-case basis. But impact factor is, as we have seen, just one of the variables that decides the credibility of a claim. In a realistic truth model, the overall trustworthiness of each individual repetition of a claim, and indeed the certainty or uncertainty in the extraction of each repetition, would both contribute to the aggregate trustworthiness of a claim across the whole corpus. However, with systems that do not calculate values for either of these variables, the best one can hope for is that they average out over a large enough number of repetitions.

### 5.3.4 Truth in test data

Although the considerations above are of primary interest in large-scale knowledge discovery projects, the concept of truth can be rather slippery in annotated development/test corpora too. It should be required of any published corpus that it reports inter-annotator agreement, as this is an upper limit on the accuracy of any system trained on the corpus, but many do not. Quality control measures in fact vary considerably, which is not surprising since corpus annotation projects also vary in size and scope between those that involve teams of annotators with both linguistics and biology training between them, and those that are initiated by a single investigator for in-house testing purposes. The general notion of truth can also vary between corpora or tasks, implicitly or explicitly; the distinction between OAOS, OAOD, OARD and OARC scoring models (see Section 5.2.2) reflects this, and an algorithm optimized to perform well on one scoring model may not do as well on the others. There is also less intentional variability, such as vagueness in the definition of 'gene' or 'protein' in the annotation of entities; some projects explicitly include or exclude things like protein families or complexes, or other DNA elements like repeats or promoters, but do not even necessarily follow their own guidelines reliably (Clegg and Shepherd, 2007b). Another classic example of ambiguity comes from the LLL Challenge instructions: [2]

> "Notice that when the absence of interaction between two genes is explicitly stated, it is represented as interaction information. For example,
>
> *There likely exists another **comK**-independent mechanism of **hag** transcription.*" [their emphasis]

[2] `http://genome.jouy.inra.fr/texte/LLLchallenge/`

It is not clear what this quote, in the description of the training data, means. Would the annotators mark *comK→hag* as an interaction, as the highlighting suggests? And if so, are we expected to follow this convention? The confusion arises partly because, while this sentence does not describe an interaction between *comK* and *hag*, neither does it claim an "absence of interaction" between these two entities; that would require a sentence like *comK does not interact with hag*. The word *another* can be interpreted to mean two things: either that the authors have already discussed one comK-independent mechanism of hag transcription, and they suspect that there is another one waiting to be found, or that they have just discussed a comK-dependent mechanism but they suspect another one exists which isn't comK-dependent. In neither case though does this sentence itself make any specific claim (positive or negative) about these two entities. In addition, the claim about hag transcription isn't even "explicitly stated" but rather speculated about without reference to any evidential support, making this an even worse choice of example sentence. Unfortunately the organizers gave no other "absence of interaction" examples, and this one doesn't even appear in the corpus, so it is impossible to glean their intention.

One result of these different notions of truth is that it is impossible to fairly compare evaluations of different systems on different tasks, unless one is very sure of the parameters of each task. Even then, varying levels of difficulty between tasks confuse the issue further. This is why high-quality, freely-available data, with transparent conventions and clear standards of assessment, is vital to biological NLP.

## 5.4 Final remarks

I hope that the reader will forgive the length and rather discursive tone of this closing chapter. It was my intention to draw attention to some interesting issues and unanswered questions in text mining that are rarely, if ever, critically analysed. Short papers presenting empirical results and engineering achievements are not always the best medium for open-ended discussions of the dominant paradigms that guide research in a particular field, so any other opportunities to step back and take a broad look over one's chosen field are always welcome.

# Appendix A

# BioNERD: a named entity recognition dictionary

The problem of biomolecular relationship extraction, as addressed in Chapter 4, relies on the accurate identification of the biological entities between which the relationships hold. The LLL Challenge corpus is designed so that this is not problematic; the entities under consideration are named unambiguously and a comprehensive dictionary is provided, meaning simple string matching is feasible. However, this kind of solution is generally not sufficient, for the reasons discussed in Section 1.8. Furthermore, almost all freely-available solutions[a] to this problem only handle named entity *recognition* and not *identification* (sometimes referred to as *grounding* or *normalization*)—although they will indicate the likely boundaries of entity names in text, they will not attempt to map them to identifiers in genomic databases. This is a serious shortcoming that limits the usefulness of these tools in real applications.

As an illustration of this issue, and a pedagogical demonstration of one fairly simple approach using deterministic rules, I developed a prototype system for both recognizing and identifying gene and protein names, for a bioinformatics textbook (Clegg and Shepherd, 2007b). This appendix is adapted from that chapter; it is included here because although named entity identification is a separate issue from relationship extraction, requiring very different solutions, it is absolutely a required element in any practical relationship extraction system.

## A.1 Background

In order to illustrate some of the challenges involved in text mining, and propose a potential solution to the named entity problem which has a bearing on most natural language projects in bioinformatics, I present a simple, high-throughput named entity

recognition and identification protocol. Similar to the approaches of Cohen (2005) Tsuruoka and Tsujii (2003a,b) and Fundel *et al.* (2005), this algorithm is designed for and tested on the human genome, requires no training data, and provides a genomic database accession number for each hit.[b]

The principles behind this approach are twofold. Firstly, although dictionary-based methods have serious coverage limitations for most organisms, the kinds of nomenclature variations that lead to false negative results are, to some extent, predictable and automatically reproducible. Secondly, if broad coverage results in a large number of false positives—acronyms, words, or phrases that resemble genuine entity names—an existing named entity recognizer with high precision (or an ensemble of several) can be employed later as a filter in order to prune the set of candidate hits. The experimental protocol therefore proceeds as follows:

1. Construct a human gene/protein name dictionary from several databases
2. Generate plausible lexical and orthographic variations of each name
3. Compile a 'keyword tree' data structure to hold the dictionary in memory
4. Search a collection of sentences for names occurring in the keyword tree
5. Discard hits which do not agree with a third-party statistical named entity recognizer

An implementation of this protocol—BioNERD, a biomedical named entity recognition dictionary—is available from `http://biotext.org.uk/` and can be easily adapted and extended as discussed in Section A.10.

## A.2 Collating the dictionary

The dictionary collation task can be broken down into the following discrete steps:

1. Collect human gene/protein symbols and full names from UniProt (Apweiler *et al.*, 2004), GeneCards (Rebhan *et al.*, 1997) and Ensembl (Hubbard *et al.*, 2005)[c]
2. Join lists by Ensembl ID and remove redundant names
3. Collapse names to lower case
4. Remove most punctuation[d]
5. Replace all whitespace with logical token boundaries
6. Insert token boundaries between runs of letters and digits, and around the remaining punctuation

7. Check for and remove redundant names again

The initial redundant list of unprocessed records from the three databases yielded 20,988 entities, of which all but 34 had at least one name, and 92,052 name-entity pairs. The normalization process ensured that, for example, the names *NF Kappa-B*, *NF-Kappa B* or *NF-kappa-B* would all lead to the same normalized form *nf kappa b*—where the space now represents a logical token boundary rather than an actual ASCII space character.

## A.3 Generating the name variants

The name variant generation process is deterministic and rule-based, unlike the probabilistic algorithm described by Tsuruoka and Tsujii (2003b), and has two components: an orthographic rewriter, and a lexical rewriter. Both are applied recursively to each normalized name in the dictionary until no new variations are generated. The lexical rewriter runs first, taking the following steps:

1. For each token in the name...
   (a) If token is in stopword list, generate a copy of the name with this token left out[e]
   (b) Add new name to dictionary

For a (normalized) name like *nacht lrr and pyd containing protein 3*, for example, the resulting variations will be *nacht lrr pyd containing protein 3*, *nacht lrr and pyd containing 3*, and *nacht lrr pyd containing 3*. Then the orthographic rewriter runs:

1. For each token in the name...
   (a) If token is numeric and between 1 and 20...
      i. Generate copy of name
      ii. Replace this token in copy with Roman numeral equivalent
   (b) Else, if token is Roman numeral between I and XX...
      i. Generate copy of name
      ii. Replace this token in copy with numeric equivalent
   (c) Else, if token is single letter with Greek equivalent...
      i. Generate first copy of name
      ii. Replace this token in first copy with spelt-out Greek letter[f]
      iii. Generate second copy of name

iv. Replace this token in second copy with SGML-encoded Greek character[g]

(d) Else, if token is spelt-out Greek letter...

i. Generate copy of name

ii. Replace this token in copy with single-letter equivalent

(e) Add new name(s) to dictionary

Thus, an original name such as *ATP1B*, having been normalized to the three-token sequence *atp 1 b*, would be expanded by the orthographic rewriter to the following set of names: *atp 1 b*, *atp 1 beta*, *atp 1 &bgr;*, *atp i b*, *atp i beta*, *atp i &bgr;*.[h]

Once again, redundant names for each entity were pruned after this process, as were names that consisted of a single character, names with more than 10 tokens, and names that exactly matched a list of around 5,000 common English words. This left 354,635 distinct names in the dictionary, of which 17,217 (4.9%) were ambiguous.[i] On average, each entity had 18.1 names, and each name referred to 1.1 entities.

## A.4 Compiling the keyword tree

In order to use the fast, scalable Aho-Corasick dictionary search algorithm,[j] the dictionary must be compiled into a data structure called a keyword tree.[k] This takes several minutes and a considerable amount of memory,[l] but allows the corpus to be searched at a speed proportional only to the corpus size and not to the number of entities in the dictionary.

## A.5 Searching the corpus

Once the keyword tree is built, the corpus can be searched as follows:

1. For each sentence in the corpus...

(a) Collapse to lower case

(b) Remove punctuation as in Section A.2

(c) Retokenize as in Section A.2

(d) For each letter in the sentence...

i. Pass letter to Aho-Corasick algorithm

ii. If algorithm signals a hit ending here...

A. Store start and end points and content of hit

(e) For each hit in this sentence...

i. Remove hit if subsumed by another hit[m]

| Features in use | $P$ | $R$ | $F$ |
|---|---|---|---|
| Just normalization (tokens & punctuation) | 23.1 | 31.0 | 26.4 |
| + Filtering of unsuitable names | 40.4 | 30.9 | 35.0 |
| + Case insensitivity | 35.6 | 41.4 | 38.3 |
| + Numeral substitution (Roman-Arabic) | 35.6 | 42.0 | 38.7 |
| + Letter substitution (Roman-Greek) | 35.6 | 42.4 | 38.7 |
| + Uninformative word removal | 32.4 | 43.5 | 37.1 |
| + ABNER filtering (exact) | 92.6 | 19.6 | 32.3 |
| + ABNER filtering (inexact) | 58.6 | 37.9 | 46.0 |
| ABNER baseline performance | 82.3 | 81.4 | 81.9 |

Table A.1: Precision, recall and F-measure measured after implementing each feature, on a set of 3,306 human-related sentences from GENETAG-05. All values are percentages.

## A.6 Filtering the results

Like all purely dictionary-based methods, this approach makes no attempt to filter false positives out of its results. However, it is trivial to employ a third-party named entity recognizer based on word and sentence features as a filter. I used ABNER (Settles, 2005) as it is fairly easy to use, with a well-documented Java API and good published performance scores, although most of the systems described in Section 1.8 or in (Clegg and Shepherd, 2007b)—or an ensemble of several—could be used instead.

I used the filter in two modes; results for each are given below. In the *exact* mode, all hits generated by the fast dictionary search were required to exactly match hits generated by ABNER. This is equivalent to running ABNER first, and then searching within the dictionary for the candidate entities it discovers, without allowing any fuzziness. In *inexact* mode, hits from the dictionary search are only required to overlap with hits generated by ABNER in order to be retained. This is a looser criterion that is more forgiving of boundary-detection disagreements between the two systems.

## A.7 Evaluation

In order to evaluate the effectiveness of the technique, I derived a test dataset from the non-species-specific GENETAG-05 corpus (Tanabe *et al.*, 2005) by selecting sentences which came from journals related to human biology or medicine,[n] or which contained the strings *human*, *patient* or *child*. This yielded 3,306 sentences. I compared the output of the system against this gold standard after enabling each major feature, calculating the precision and recall score each time,[o] as well as the proportion of true positives where the name refers to more than one entity. The results are presented in Table A.1.

## A.8 Analysing the results

These results illustrate several important points. As with many NLP tasks, there is a natural trade-off between precision, the proportion of predicted entity names which are correct, and recall, the proportion of genuine entity names that are predicted. This trade-off is most strikingly apparent in the difference between requiring exact matches and allowing inexact matches when using ABNER as a filter.

The ABNER baseline performance is included to demonstrate what a non-dictionary-based named entity recognizer is capable of, but it must be noted again that the output of ABNER does not include links back to genomic database identifiers so it cannot provide any information on *which* gene or protein has been tagged each time. The very low recall score when allowing only exact matches with ABNER algorithm makes it clear how few of the entities tagged by ABNER are present (in the exact same form) in even an expanded dictionary. By contrast, the dictionary search algorithm makes many more errors than ABNER, but provides a link to a genomic database (in this case, an Ensembl record) for each entity it finds.

Because the test set was generated in a partly automated manner, I manually examined a random subset of 300 names in the test set which were missing from the dictionary, in order to determine the cause of their omission. Although many were due to genuine deficiencies in the dictionary, a considerable number (33, 11.0%) were occurrences of plural, family, group or otherwise generic names that I did not set out to cover.[p] An even larger subset (44, 14.7%) were genes or proteins from other organisms that happened to be mentioned in a human-relevant sentence, but as many as 20 (6.7%) seemed to be annotator errors.[q] Taken together, these figures suggest that the actual false negative (recall error) rates may be around a third lower than those reported in Table A.1.

This prototype system was originally designed for a planned bibliometric investigation into research coverage of the human genome, and thus was built with a gene's-eye-view of human biology. The results however drew attention to the inadequacy of considering names for genes and for their protein products to be interchangeable, a shortcut also taken by most named entity recognition projects. This practice disregards the fact that biomedically important proteins are often hetero-oligomers or other complexes composed of the products of several genes.[r] It must be noted in this context, however, that even human experts do not find it trivial to distinguish mentions of biomolecular entities into genes and proteins. Hatzivassiloglou *et al.* (2001) set three specialists the task of classifying entity mentions into five categories: gene, protein, RNA, ambiguous or wrongly extracted. They found that the three experts only agreed unanimously 69% of the time, and only achieved a majority (two or three agreeing) 94% of the time.

## A.9 Dealing with ambiguity

Leaving aside this very specific kind of ambiguity, there are two kinds of ambiguity that must be considered when detecting biological entities of one broadly-defined category, as discussed in this appendix. The first is ambiguity between multiple referents in that category, e.g. gene acronyms that stand for more than one gene, and the second is ambiguity between entities of that category and strings of text that might represent other classes of entity or any other words or phrases. The algorithm presented above, as it stands, does not make any attempt to deal explicitly with either kind of ambiguity internally. The latter kind of ambiguity is tackled by the use of ABNER as a filter, as a given string may or may not be tagged as an entity name by ABNER depending on its context in the sentence. The former kind needs a more subtle approach. These problems can be tackled individually or together, and both can be characterized as special cases of the general problem of term-sense disambiguation which is not restricted to biomedical NLP applications. I will briefly discuss some methods that have been or could be applied to them.

The approach used by EBIMed and described by Gaudan *et al.* (2005) is geared specifically towards the disambiguation of abbreviations, but is not restricted to entities of any particular class. It makes a distinction between local and global abbreviations; local abbreviations are those whose long forms are given before they are used, and these are resolved with the help of an automatically extracted dictionary, while global abbreviations, which do not have explicitly-stated long forms, require the use of statistical term-frequency measures. A very different approach is proposed by Widdows *et al.* (2003) for ambiguous biomedical terms from UMLS, and makes use of the fact that UMLS comes with various hand-curated tables of related concepts. For each possible sense (corresponding concept) of an ambiguous term, the algorithm counts the number of related concepts that are referred to by terms in the textual neighbourhood of the ambiguous term, and the sense with the highest count wins. Given the availability of various hand-curated resources in the molecular biology domain that list already-known relationships between genes, proteins and other entities or concepts, it is easy to see how one might adapt this approach.

The National Center for Integrative Bioinformatics has developed a system similar to this but using statistical document similarities rather than term relatedness (D. J. States, U. of Michigan, pers. comm.). The document in which an ambiguous name is found is compared to documents containing unambiguous names for each of the possible referents of the name. The most likely referent is then selected according to which document or documents are most similar to the document with the ambiguous referent. Cohen (2005) use a simpler variation on this overall approach, which does not consult other resources outside the document of interest. Instead they look elsewhere in the same document for other, unambiguous names for the entities referred to by each

ambiguous name. In the rare cases where this does not resolve the ambiguity—i.e. more than one of the referents of the ambiguous name seem to be referred to by other names—the ambiguous name is discarded.

There is one further kind of ambiguity which was not addressed here—ambiguity between species. Many genes and proteins share names between organisms, and while this may reflect sequence homology or functional equivalence, the level of similarity can vary. Furthermore there is nothing preventing biomolecular entities in different species having similar or identical names by accident. Witte *et al.* (2007) describe an ontology-based entity identification system that looks for species names as well as protein names, thus narrowing the number of possible database hits for a protein entity to those with similar names in the species identified (where possible). It also performs similar filtering on mutation information where this is available.

## A.10 Concluding remarks

An important lesson from this worked example is that while naïve dictionary-matching methods perform comparatively poorly, a few simple variant generation methods can improve recall by almost 50%, and these methods in conjunction with inexact filtering from a high-performance named entity recognizer can almost double overall performance (F-measure). The potential is great for improving on these early gains with a more comprehensive dictionary,[s] and more variant generation rules.[t]

Although the precision, recall and F-measure scores for the dictionary algorithm are considerably less than those achieved by ABNER on the same data, the advantage of using an algorithm that returns an actual database identifier cannot be overlooked. Furthermore, it must be noted that some of the existing text mining systems described in Chapter 1 use simple exact-matching techniques that can only be expected to perform about as well as the lowest scores reported above. In a real-world application based on our method, one might want to consider a 'best-of-both-worlds' approach—a set of high-confidence annotations could first be made based on inexact agreement between the dictionary method and ABNER or another named entity recognizer, and then augmented with a set of lower-confidence annotations obtained by finding the closest fuzzy-matched name in the database to each of the remaining tags in the recognizer's output.

## A.11 Acknowledgements

This work was supported by the Biotechnology and Biological Sciences Research Council and AstraZeneca. I would like to thank Mark Halling-Brown for supplying the dictionary and A. G. McDowell for implementing (and advising on) the Aho-Corasick

algorithm.

# Notes

[a]Notable exceptions include NLProt (Mika and Rost, 2004) and LingBlast (available by request from `http://alias-i.com/`), both of which do provide database identifier mappings.

[b]For cases where the same name refers to several distinct entities (homonyms) this method returns all of the associated accession numbers.

[c]I treated genes and their protein products as interchangeable for the purposes of the investigation, a shortcut that is common in many named entity recognition systems, although some of the drawbacks with this approach are discussed in Section A.8.

[d]I retained *&* and *;* as these are necessary for understanding SGML escape sequences, which sometimes appear in MEDLINE to represent special characters—e.g. *&agr;* for $\alpha$. I also kept *+* as it is useful for distinguishing positive ions, which are common in protein names, from similar-looking sequences of characters. I decided to remove all instances of *-* though, as it is much more commonly used as a hyphen or dash which is less selective.

[e]This is a list of uninformative and optional words that occur in gene and protein names, such as *the*, *of* and *a*, as well as generic terms such as *gene*, *protein*, *precursor*, *molecule* etc.

[f]e.g. *alpha*.

[g]e.g. *&agr;*. Note that this is a one-way translation as there are no SGML character codes in our dictionary, although the reverse could be implemented straightforwardly.

[h]The letter *i* is excluded from the Roman to Greek letters conversion, so as to avoid unnecessary and unhelpful generation of the *iota* character every time a digit *1* was processed.

[i]That is, referring to more than one entity.

[j]A thorough description of the algorithm and its associated data structure is given by Gusfield (1997); rather than reimplement it myself, I adapted a Java version available at `http://www.mcdowella.demon.co.uk/programs.html`.

[k]I added a token boundary character at the beginning and end of each entity name so that they would only match when aligned with token boundaries in the corpus; this means that the name *octn2* will be matched in the string *octn2-mediated*, for example, but the name *car* will not be matched in the string *carrier*.

[l]My non-optimized prototype implementation requires 768Mb of memory for the Java virtual machine to complete the task happily.

[m]This step ensures that, for example, the string *OB receptor* does not also register a hit for *OB*.

[n]I looked for journal names containing the strings *Hum* or *Child*, or *Clin* or *Med* but not *Vet*—the journal names are supplied in standard abbreviated format.

[o]Due to the existence of multiple acceptable alternative annotations for many of the names in GENETAG-05, this calculation was slightly more complicated than usual. I had to define a true positive as any predicted name that matches any one of the alternative annotations for a given instance of an entity in GENETAG-05, a false positive as any predicted name that matches none of the annotations in GENETAG-05, and a false negative as a GENETAG-05 entity with no allowed alternative annotations predicted by our algorithm. Also, since GENETAG-05 does not contain any kind of 'instance identifier', I had to preprocess the annotation lists supplied to group annotations together based on overlaps, in order to determine which groups of annotations referred to the same instance of the same entity. Finally, an additional level of complexity was added by the fact that the dictionary algorithm will generate multiple hits for different entities where several entities share the same name. For scoring purposes, each of these ambiguous hits was treated as a single (true or false) positive.

[p]These included cases such as *E1 genes*, *anti-viral proteins*, *MHC class II promoters* and *blood hemoglobin*.

[q]Many of these were non-gene/protein entities that had been tagged in error, such as *statin* (a class of drug), *platelet activating factor* (a phospholipid derivative) and *immunoperoxidase* (a lab technique). Also

present were several sequence features that broke the annotators' own guidelines on what should be included, for example *DXS52* (a sequence-tagged site), *Alu* (a class of common repeat sequences) and *HS40* (a regulatory element).

[r]For example, *fibrinogen*, *nuclear factor (NF)-kappaB*, *casein* and the immunoglobulin family.

[s]An obvious way to improve on the dictionary we used would be to include gene/protein families as distinct meta-entities, as these are oftened mentioned collectively by authors, for example *MAPK* or *human E2F*. Another option would be to include names of domains (such as *SH3* and *PH*), and complexes of multiple gene products as discussed in Section A.8.

[t]For example automatic acronym generation, re-ordering of words in long names, and prepending *h* (for *human*) to short symbols as this is a common abbreviation in the literature.

# Appendix B

# Glossary of linguistic terms

This appendix contains brief descriptions of the major linguistic terms used in this thesis. Many of the descriptions are abridged from the Internet Grammar of English, compiled by the Survey of English Usage at University College London:

`http://www.ucl.ac.uk/internet-grammar/`

Additional material was adapted from the corresponding entries in the English Wikipedia:

`http://en.wikipedia.org/`

### Adjective

Adjectives describe a quality or attribute of a noun, for example *logical* or *silent*. They come in **absolute**, **comparative** and **superlative** forms: *dark*, *darker*, *darkest*. This is known as *modifying* the noun.

**Participial adjectives** often end in *-ed* or *-ing* and resemble verbs. They are however a distinct grammatical category, as in *you're very annoying* (participial) vs. *you're annoying me* (verb).

### Adjunct

An adjunct is a phrase which modifies its parent phrase in order to supply additional but optional information. Adjectives and adverbs are usually adjuncts; in the phrase *the old lady*, for example, the adjective *old* is not actually required—*the lady* would have been a complete noun phrase without it—but it added extra information to the meaning (compare complements).

### Adverb

Adverbs, such as *quickly* or *better*, modify verbs, adjectives or other adverbs, in a similar manner to adjectives modifying nouns. Like adjectives, they can be **absolute**, **comparative** or **superlative**, via alternative endings or by modification with *more* or *most*.

In fact, *more* or *most* (in this context) are themselves adverbs, known as **degree adverbs**, along with other examples like *quite* or *extremely*.

### Apposition

In an apposition, two noun phrases are placed alongside each other, in order that one can restrict or otherwise modify the other. Sometimes they are separated **parenthetically** by punctuation such as commas, brackets or dashes. Examples include *human homeobox gene (HOX)*; *another protein, PAK2*; and *the forespore-specific sigma factor sigma(F)*.

### Clause

Clauses are sequences of words which are made up of phrases (including at least one verb phrase) and can be nested within one another. For example, in *I think I'd like coffee*, the **subordinate clause** *I'd like coffee* is nested within the **matrix** (or **superordinate**) clause *I think I'd like coffee*.

There are various different classifications of clauses, depending on the form the verb takes, the word used to introduce the clause, or the semantic function of the clause. An important and diverse subclass is **relative clauses**, which are subordinate clauses used to modify nouns, as in *the man who lives beside us* (the relative clause here is *who lives beside us*).

### Complement

A complement is a phrase which modifies its parent phrase in order to complete its meaning. For example, in *we gave James a present*, the two noun phrases *James* and *a present* are complements; without them, the verb *gave* is under-specified. Complements are often confused with adjuncts, but they have a different function and different rules govern their usage.

### Conjunction

A conjunction joins two words, phrases or clauses together into a single grammatical unit. **Coordinating conjunctions** join their **conjuncts** together at the same grammatical level, e.g. *and* in *cold and wet* or *but* in *I play tennis but I don't play well*. In terms of constituent trees, their conjuncts are siblings.

By contrast, **subordinating conjunctions** are used to to join a subordinate clause to its superordinate. In *I'll be home at nine if I can get a taxi*, the conjunction *if* introduces the subordinate clause *if I can get a taxi* which is syntactically dependent upon its parent clause.

### Constituent

Phrases, clauses and sentences are all types of constituent—hierarchically nestable grammatical units.

### Coreference

A coreference occurs when two or more noun (or pronoun) phrases refer to the same *referent* (external thing). The most common kind of coreference is *anaphora*, where one of the phrases (usually a pronoun) is resolvable by reference to the other phrase (usually a noun). For example, consider the sentence *katX is also a sigmaB-dependent general stress gene, since it is strongly induced by heat*. The pronoun *it* refers anaphorically to the noun *katX*, and both refer to the same real-world entity, the katX gene.

### Dependency

In the sense in which it is used in this thesis, a dependency is a directional syntactic relationship between two words in a sentence. One word, the **dependent**, is said to depend on the other, the **governor**; the role of the dependent in the sentence is constrained or supplied by its relationship to the governor, which is the equivalent (in phrase-structure terms) of the head of a phrase.

For example, when an adjective modifies a noun or an adverb modifies a verb, the modifier is the dependent and the word thus modified is the governor. Another example would be the noun phrases which make up the subject and direct object of a verb. In this case, the verb is the governor; the head nouns of the subject and object noun phrases depend on the verb, but of course other words in the noun phrases depend in turn on those nouns.

### Determiner

Determiners are words which precede nouns and establish scope or quantity, as in *a taxi*, *those apples* or *many people*. Numerals can also be used as determiners.

### Ellipsis

An ellipsis or *elliptical construction* is a grammatical construction where a word or phrase vital to interpretation is left out but can be understood from context. For example, consider the following sentence: *SpoIIID at low concentration repressed cotC*

*transcription, whereas a higher concentration only partially repressed cotX transcription*. The second half, after the comma, is elliptical; the complete reading is *whereas a higher concentration **of SpoIIID** only partially repressed cotX transcription*. The phrase in bold is left unwritten but can be understood by a competent reader.

Such phenomena, although usually trivial for humans, can be very problematic for NLP algorithms.

### Genitive marker

The genitive marker is the *'s* suffix for nouns indicating ownership, or just a bare apostrophe in the case of plural nouns ending with *s*: *the boys' pens*.

### Homonymy

Two words are said to be **homonyms** if they share the same spelling and pronunciation, but have different and unrelated meanings (contrast polysemy).

### Head

The central element of a phrase, the word on which the rest of the phrase depends, is called the head of that phrase—the main verb in a verb phrase, the noun in a noun phrase which the rest of the phrase modifies, etc.

### Holonymy

One word is said to be a **holonym** of another if the referent of the other word is a member or constituent part of its own referent, e.g. *hand* is a holonym of *finger*. This is sometimes known informally as a 'has-a' relationship. Compare meronymy.

### Hypernymy

One word is said to be a **hypernym** of another if its referent subsumes the meaning of the other word, e.g. *limb* is a hypernym of *arm*. Compare hyponymy.

### Hyponymy

One word is said to be a **hyponym** of another if its referent is subsumed by the meaning of the other, e.g. *arm* is a hyponym of *limb*. Compare hypernymy.

### Lemma

A lemma is a canonical, **uninflected** form of a word, such as *phosphorylate* for the verbs *phosphorylate*, *phosphorylates*, *phosphorylated* and *phosphorylating*. Note however that the noun *phosphorylation*, although it comes from the same **root** as the verb

forms, is in a different lexical category (part of speech) and thus has a different lemma (the same as the singular form itself, *phosphorylation*).

The term **stemming** is sometimes used to refer to the process of automatically retrieving the lemma of a word (more properly **lemmatization**), although it is also used in a slightly different sense where a common stem *phosphorylat-* or even *phosphoryl* would be offered for both the verbs and noun given above.

### Meronymy

One word is said to be a **meronym** of another if its referent is a member or constituent part of the other word's referent, e.g. *finger* is a meronym of *hand*. This is sometimes known informally as a 'part-of' relationship. Compare holonymy.

### Morphology

This is the branch of linguistics concerned with the internal structure of words, for example the relationships between the **inflected** forms of words (*running*, *ran*) and their **uninflected lemmas** (*run*).

### Noun

Nouns are commonly thought of as naming words, for concrete entities such as people, places or things, as well as abstract and intangible concepts. Nouns which name *specific* people, places or time periods on a calendar are known as **proper nouns**, e.g. *London* or *Tuesday*; all others are **common nouns**.

**Count nouns** are those which can be preceded by a numeric quantifier, as in *three pens*; all others are known as **mass nouns** or simply non-count nouns. Count and mass nouns also differ in the determiners they can take.

Most nouns can take an *-s* suffix to indicate that they are **plural** (more than one in number, compare **singular**) although there are many irregular plural forms too such as *children*.

### Number

In linguistic terms, this refers to the distinction between **singular** (referring to one entity, as in *myself* or *child*) and **plural** (referring to more than one entity, as in *ourselves* or *children*).

### Object

The object of a verb is the complement of the verb phrase referring to the target or recipient of the effect of the verb, e.g. *the piano* in *David plays the piano*. Ditransitive

verb phrases like *we gave John a present* have a **direct object** and an **indirect object**; the indirect object comes immediately after the verb, and the direct object after that.

A prepositional phrase which is the complement of a verb is sometimes referred to as the **prepositional object**.

### Person

The person of a verb refers to the relationship of its subject to the speaker/writer, and is explained most succinctly with examples: **first person** is the speaker (as in *I/we walk*), **second person** is the person spoken to (as in *you walk*), and **third person** is another party (as in *he/she/they/someone/a person/Jim walks*).

The form of a verb changes to reflect person—the regular third person singular, for example, acquiring a *-s* suffix as above.

### Phrase

Phrases are grammatically complete blocks of a specific type (noun phrase, verb phrase etc.) which can be treated as single units and used in place of a single word of the corresponding type where appropriate. That is, a noun phrase (e.g. *a noun phrase*) can be used anywhere a single noun can be used, even though it might contain several words or nested phrases of other types.

### Polysemy

A word is said to be **polysemous** if it has multiple related meanings (contrast homonymy).

### Preposition

Prepositions are a rather diverse set of words which introduce modifying noun phrases of various sorts. Many of them are spatial (*behind*) or temporal (*after*) in nature, others indicate ownership (*of*) or association (*with*), and some are used in an enormous variety of contexts (*by*, *for*). **Complex prepositions** are two or three word expressions which act as a single preposition, such as *according to*.

### Pronoun

Sometimes regarded as a subset of nouns, pronouns are words which can replace a noun in a sentence and refer to the same entity or concept. They include **personal pronouns** like *she* and *it*, **possessive pronouns** like *mine*, **reflexive pronouns** like *yourself*, and various other categories.

**Semantics**

Semantics refers in general to the meaning of language—or the relationships between language and reality—contrasting with syntax which refers to the form of language. For example, the sentences *ykuD was transcribed by SigK RNA polymerase* and *SigK RNA polymerase transcribed ykuD* are syntactically different but semantically equivalent.

**Sentence**

A single clause is the simplest kind of sentence; a sentence can also consist of two or more clauses joined by conjunction. While many people think of sentences as being delimited by capital letters and full stops, these are of course merely typographical conventions and not grammatical phenomena.

**Subject**

The subject of a verb is the noun phrase referring to the entity whose actions or attributes the verb describes; in English, the noun phrase before the verb, as in *the lion roared* or *David plays the piano*. However, see the entry on **voice** for an additional note on subjects.

**Syntax**

Syntax refers to the structure of language and the grammatical rules by which words (and smaller morphological units) are combined into longer sequences. It is not concerned with the sense or truth-value of a sentence or other linguistic utterance, simply its form; that is, a sentence can be entirely grammatical (syntactically well-formed) but still false or completely nonsensical.

Chomsky (1957) famously used the sentence *colorless green ideas sleep furiously* as an example of a sentence which is syntactically correct but meaningless.

**Verb**

Verbs are traditionally described as "action words" or "doing words", although there are many verbs like *seem* which do not describe an action as such. English verbs have a **base form** or **bare infinitive** (*talk*) as well as **present tense** (*talks*) and **past tense** (*talked*) forms. These are usually indicated by *-s* and *-ed* endings, but there are many irregular verbs which behave differently. Verbs also have two **participle forms** which usually end in *-ing* and *-ed*. The difference between the *-ed* participial and the past tense can be seen more clearly in irregular verbs, e.g. *he has taken* and *he took*.

Verb phrases often contain **auxiliary verbs** which alter the meaning of the main verb. This can be for temporal modification (*has broken*, *will sing*), ability (*can't*

*cook*), intent (*won't cook*), and various other **modes** and **aspects**. The **passive voice** is a special case discussed below.

### Voice

Most clauses in English are in the **active voice**, where the subject of the verb is the agent or actor causally responsible for the effects described by the verb. However, there is also a **passive voice** construction using the auxiliary verb *be* and the *-ed* participle, and often taking a prepositional complement (almost always introduced by *by*).

In these cases, the subject takes the semantic role usually filled by an object in an active-voiced clause, and the *by*-complement (if present) takes the agent role, as in *David was congratulated by Paul*. Although *David* is syntactically the subject, he is the recipient of the congratulations rather than their source.

# Appendix C

# Dictionary of linguistic labels

## C.1 Part of speech tags

The following POS tags are defined by the Penn Treebank, and are used by GENIA and the Stanford tools too. Additional tags corresponding to common punctuation symbols are also in use but are not show here. Descriptions and usage guidelines are here:

`ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz`

`CC` Coordinating conjunction

`CD` Cardinal number

`DT` Determiner (article)

`EX` Existential *there*

`FW` Foreign word

`IN` Preposition or subordinating conjunction

`JJ` Adjective (inc. ordinal numbers)

`JJR` Adjective, comparative

`JJS` Adjective, superlative

`LS` List item marker

`MD` Modal verb

`NN` Common noun, singular or mass

`NNS` Common noun, plural

| | |
|---|---|
| NNP | Proper noun, singular |
| NNPS | Proper noun, plural |
| PDT | Predeterminer |
| POS | Possessive ending (*'s*) |
| PRP | Personal pronoun |
| PRP$ | Possessive pronoun |
| RB | Adverb |
| RBR | Adverb, comparative |
| RBS | Adverb, superlative |
| RP | Particle |
| SYM | Symbol |
| TO | *to* |
| UH | Interjection (exclamation) |
| VB | Verb, base form |
| VBD | Verb, past tense |
| VBG | Verb, gerund or present participle |
| VBN | Verb, past participle |
| VBP | Verb, non-3rd-person singular present |
| VBZ | Verb, 3rd-person singular present |
| WDT | *Wh*-determiner |
| WP | *Wh*-pronoun |
| WP$ | Possessive *wh*-pronoun |
| WRB | *Wh*-adverb |

## C.2 Constituent labels

The following constituent labels are defined by the Penn Treebank, and are also allowed by GENIA, apart from NX and NAC for which GENIA just uses NP. Descriptions and usage guidelines are here:

ftp://ftp.cis.upenn.edu/pub/treebank/doc/manual/root.ps.gz

| | |
|---|---|
| ADJP | Adjective phrase |
| ADVP | Adverb phrase |
| CONJP | Conjunction phrase |
| FRAG | Fragment |
| INTJ | Interjection |
| LST | List element |
| NAC | 'Not a Constituent'—prenominal modifier scope marker |
| NP | Noun phrase |
| NX | Head marker in complex noun phrases |
| PP | Prepositional phrase |
| PRN | Parenthetical phrase |
| PRT | Particle (wrapper for RP words) |
| QP | Quantifier Phrase (numerical modifiers of *NP*) |
| RRC | Reduced relative clause |
| S | Simple declarative clause (most sentences) |
| SBAR | Subordinate clause |
| SBARQ | Direct question introduced by a *wh*-word or -phrase |
| SINV | Inverted declarative sentence (subject after verb) |
| SQ | Inverted yes/no question or main phrase of a *wh*-question |
| UCP | Unlike coordinated phrase (*John liked* ***apples and going swimming***) |
| VP | Verb phrase |
| WHADJP | Wh-adjective phrase |

| | |
|---|---|
| WHAVP | Wh-adverb phrase |
| WHNP | Wh-noun phrase |
| WHPP | Wh-prepositional phrase |
| X | Unknown, uncertain, or unbracketable (e.g. typos) |

## C.3 Dependency labels

The following dependency type labels are used by the Stanford NLP tools. In addition to those listed here, dependencies are created on-the-fly to represent co-ordinating conjunctions (e.g. conj_and) and prepositions (e.g. prep_from) created when collapsing conjunction (e.g. *and*) and preposition (e.g. *from*) words into dependencies. (If this operation is performed, the existing dependencies indication conjunctions and prepositions will be removed.) The dependencies are arranged in a hierarchy of specificity which is not show here (see de Marneffe *et al.*, 2006); less-specific dependencies such as arg, mod and particularly dep (the most generic) are only assigned if a more specific type cannot be determined from the input tree structure. When a description refers to a phrase, the actual attachment is made to the head of the phrase. Dependency types which can never be instantiated have been omitted from this list. Fuller descriptions of the dependency types, and the patterns used to extract them from trees, are in the Javadoc and source code for the class EnglishGrammaticalRelations in the Stanford tools. These are available as part of the Stanford parser distribution, available at:

http://nlp.stanford.edu/software/lex-parser.shtml

| | |
|---|---|
| abbrev | Attaches a parenthetical abbreviation to the parent noun phrase |
| acomp | Attaches the adjectival complement of a verb |
| advcl | Attaches an adverbial clause modifying a verb |
| advmod | Attaches an adverb modifying a verb |
| agent | Attaches the agent (logical subject) of a passive verb |
| amod | Attaches an adjectival modifier of a noun |
| appos | Attaches an appositional noun phrase modifying a parent NP |
| arg | Attaches an argument of a phrase or clause |
| attr | Attaches an attributive modifier of a verb phrase |

| | |
|---|---|
| `aux` | Attaches the auxiliary of main verb |
| `auxpass` | Attaches the passive auxiliary of main verb |
| `cc` | Attaches a coordinating conjunction to the previous conjoined element |
| `ccomp` | Attaches the clausal complement of a verb |
| `comp` | Attaches the complement of a verb |
| `complm` | Attaches the complementizer of a clausal complement |
| `conj` | Attaches a later conjoined element to an earlier one |
| `cop` | Attaches a complement (verbal or adjectival) to a copular verb (e.g. *is*) |
| `csubj` | Attaches the clausal subject of a verb |
| `det` | Attaches the determiner (article) of a noun |
| `dobj` | Attaches the direct object of a verb |
| `expl` | Attaches an existential *there* to the verb |
| `infmod` | Attaches an infinitive verb modifying a noun phrase |
| `iobj` | Attaches the indirect object of a verb |
| `mark` | Attaches the word marking (introducing) an adverbial complement |
| `mod` | Attaches a modifier of a verb |
| `neg` | Attaches a negation word (e.g. *not*) to the negated word |
| `nn` | Attaches a prenominal modifier in a compound noun phrase |
| `nsubj` | Attaches the nominal subject of a verb |
| `nsubjpass` | Attaches the nominal subject of a passive verb |
| `num` | Attaches a numeric modifier of a noun |
| `number` | Attaches part of a complex numeric phrase |
| `obj` | Attaches the object of a verb |
| `partmod` | Attaches a participle verb modifying a noun phrase |
| `pobj` | Attaches the prepositional object of a verb to the preposition |
| `poss` | Attaches the possessor to the possessee in *'s* possession |

| | |
|---|---|
| `possessive` | Attaches the *'s* suffix to the possessor in possession |
| `preconj` | Attaches any preconjunction modifiers of a conjunction |
| `pred` | Attaches the predicate of a verb |
| `predet` | Attaches any predeterminer modifiers of a noun |
| `prep` | Attaches a preposition to a noun or verb phrase |
| `prt` | Attaches the particle to the main verb in a phrasal verb |
| `punct` | Attaches punctuation symbols to the head of the phrase |
| `purpcl` | Attaches a purpose clause modifying a verb (introduced by *in order to*) |
| `rcmod` | Attaches a relative clause modifying a noun |
| `ref` | Attaches the word introducing a relative clause to the modified noun |
| `rel` | Attaches the 'relative' of a relative clause |
| `subj` | Attaches the subject of verb |
| `tmod` | Attaches a temporal modifier or a verb or adjective |
| `xcomp` | Attaches the clausal complement of a verb with an external subject |
| `xsubj` | Attaches the 'controlling' subject of a verb in a subordinate clause |

# Appendix D

# MPL language specification

The following sections describe the syntax of MPL (see Section 4.4) in extended Backus-Naur form (ISO/IEC 14977), along with some notes on semantics.

## D.1 File structure

```
entry           =    comment | rule ;

comment         =    ( "#", ? any text ?, newline ) |
                     newline ;

rule            =    ( match rule, newline ) |
                     ( pattern rule, newline ) |
                     ( replacement rule, newline ) ;

newline         =    ? system-specific newline character(s) ? ;
```

Comments introduced with a # character, and blank lines, are ignored.

## D.2 Match rules

```
match rule       =    { "!" }, "match ", variable, " = ", regexp ;

variable         =    ( "@" | "#" ), letter, { letter} ;

letter           =    "A" | "B" | "C" | "D" | "E" | "F" | "G" |
                      "H" | "I" | "J" | "K" | "L" | "M" | "N" |
                      "O" | "P" | "Q" | "R" | "S" | "T" | "U" |
```

```
                    "V" | "W" | "X" | "Y" | "Z" ;

regexp          =   ? regular expression (Java syntax) ? ;
```

Match rules define regular expressions for matching against words, POS tags or arc labels in patterns (see below). Those beginning with the ! character are inverted; that is, they match any node to which the regular expression does *not* match.

Each regular expression is assigned to a variable, which may be identified either by a # or @ symbol. These variables can be used when writing patterns. The choice of symbol divides the variables into two subsets and allows us to write replacement rules (see below) that target one subset only.

## D.3 Pattern rules

```
pattern rule    =   "pattern", newline, node, newline, "end" ;

node            =   pos tag, "~~", ( word | composite ),
                    { " ", child arc } ;

pos tag         =   variable | literal ;

word            =   variable | literal ;

literal         =   ? any non-whitespace text ? ;

composite       =   "{", word, { "_", word }, "}" ;

child arc       =   "( ", arc label, " ", node, " )" ;

arc label       =   variable | literal ;
```

Pattern rules are textual representations of subgraphs (graph fragments) to be matched against whole-sentence dependency graphs. The first node in the pattern is the root node; each node may have one or more child arcs which themselves end in nodes. This recursive definition means that patterns of arbitrary width and depth may be defined. The order in which child arcs are listed is not important. Excess whitespace within patterns is ignored, allowing the user to format them in a visually appealing manner, using line breaks, indentation etc. However `begin` and `end` must occur on lines by themselves.

Words, POS tags and arc labels (dependency types) can be specified in terms of variables (see above) or literal strings. A variable's regular expression can match any-

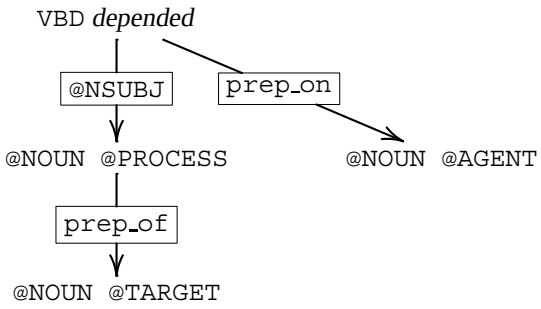


Figure D.1: A graphical representation of an MPL pattern rule, showing the constraints on words, POS tags and dependency types. This pattern will match clauses like *Entityaa depended on expression of Entityab*.

where in the target string to count, but literals must match the entire target string exactly. For example, an arc label given as the literal `VB` would not match against a `VBD` dependency, but the regexp `/VB/` would. Composites consist of variables and literals, separated by underscores, which match any one character. These sequences must match the target string contiguously; excess material at either end is ignored.

Obviously, a pattern must be compiled into an in-memory graph of Java objects before it can be used. Figure D.1 shows a graphical representation of the structure of the pattern defined as follows:

```
pattern
    VBD~~@DEPENDED
        ( prep_on @NOUN~~@AGENT )
        ( @NSUBJ @NOUN~~@PROCESS ( prep_of @NOUN~~@TARGET ) )
end
```

## D.4 Replacement rules

```
replacement rule =    "replace ", old text, " = ", new text ;

original text    =    ? any string ? ;

new text         =    ? any string ? ;
```

Replacement rules allow unconstrained search-and-replace functionality over patterns before they are compiled, allowing variant patterns to be easily generated. *Typically*, the string to be replaced would be a single node definition, or a POS tag or arc label, or a variable, but this is not mandated by the language.

Note that replacement works on the raw text of a pattern, so unexpected whitespace inside a string can prevent a match.

On initially reading an MPL file, the MPL parser builds a pool of pattern rules (those specified explicitly in the file) and a list of replacement rules in the order they occur in the file. One by one, each replacement rule is applied to every pattern in the pool, and any new patterns generated are added to the pool, so that subsequent replacement rules operate on them as well as on the original 'seed' set. Thus the order the replacement rules are declared makes a difference to the ultimate outcome. Note that a replacement rule can match a pattern in multiple places, generating a new pattern for each distinct combination of matches.

## D.5 An example

Consider the following simple MPL file.

```
# Match rules

match @AGENT = Entity[a-z]{1,2}
match @TARGET = Entity[a-z]{1,2}

# Pattern rules

pattern
VB~~activate
        ( nsubj NN~~@AGENT )
        ( dobj NN~~@TARGET )
end

# Replacement rules

replace VB~~activate = VBZ~~activates

replace NN~~@TARGET = expression ( prep_of NN~~@TARGET )
```

The match rules define regular expressions to identify the agent and target in the placeholder form *Entityaa*, where *Entityaa* could be any placeholder composed of the string *Entity* and two lower-case letters. This is the minimum set of variables required to retrieve interactions. In the current implementation, the two variables `@AGENT` and `@TARGET` are treated differently from other variables, in that their bindings in successfully-matched patterns are saved and used to generate interactions.

The pattern rule matches just the simple fragment *Entityaa activate Entitybb*. The first replacement rule generates another pattern which matches *Entityaa activates Entitybb*, while the second replaces the whole target node with a prepositionally-modified noun. This is applied to both the original pattern and the new one generated by the first replacement rule, resulting in two new patterns that match *Entityaa activate expression of Entitybb* and *Entityaa activates expression of Entitybb*.

# Bibliography


Abney, S. (1996). Partial parsing via finite-state cascades. In J. Carroll, editor, *Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information (ESSLI)*, pages 8–15, Prague.

Adar, E. (2004). SaRAD: A simple and robust abbreviation dictionary. *Bioinformatics*, **20**(4), 527–533.

Ahmed, S. T., Chidambaram, D., Davulcu, H., and Baral, C. (2005). IntEx: A syntactic role driven protein-protein interaction extractor for bio-medical text. In *Proceedings of the ACL-ISMB Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, pages 54–61, Detroit. Association for Computational Linguistics.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques and Tools*. Addison Wesley, Boston.

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, **25**(17), 3389–3402.

American Heritage Dictionaries (2000). *American Heritage Dictionary: Fourth Edition*. Houghton Mifflin, New York.

Apweiler, R., Bairoch, A., Wu, C., Barker, W., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M., Natale, D., O'Donovan, C., Redaschi, N., and Yeh, L. (2004). UniProt: the Universal Protein knowledgebase. *Nucleic Acids Research*, **32**, D115–D119.

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J. C., Richardson, J. E., Ringwald, M., Rubin, G. M., and Sherlock, G. (2000). Gene Ontology: tool for the unification of biology. *Nature Genetics*, **25**(1), 25–9.

Averbuch, M., Karson, T. H., Ben-Ami, B., Maimon, O., and Rokach, L. (2004). Context-sensitive medical information retrieval. In M. Fieschi, E. Coiera, and J. Li, editors, *Proceedings of the 11th World Congress on Medical Informatics (MEDINFO 2004)*, pages 282–286, Amsterdam. American Medical Informatics Association, IOS Press.

Baclawski, K., Futrelle, R. P., Noy, N. F., and Pescitelli, M. J. (1993). Database techniques for biological materials and methods. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 21–28. AAAI Press.

Baruch, J. J. (1965). Progress in programming for processing english language medical records. *Annals of the New York Academy of Sciences*, **126**(2), 795–804.

Bechhofer, S., Goble, C., and Horrocks, I. (2001). DAML+OIL is not enough. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University.

Bies, A., Kulick, S., and Mandel, M. (2005). Parallel entity and treebank annotation. In *Proceedings of the Workshop on Frontiers in Corpus Annotations II: Pie in the Sky*, pages 21–28, Ann Arbor, Michigan. Association for Computational Linguistics.

Bikel, D. M. (2002). Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of the Human Language Technology Conference 2002 (HLT2002)*, San Diego.

Bikel, D. M. (2004). A distributional analysis of a lexicalized statistical parsing model. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, Barcelona. ACL.

Black, E., Abney, S., Flickenger, S., Gdaniec, C., Grishman, C., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). Procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of a workshop on speech and natural language*, pages 306–311, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Blaheta, D. and Charniak, E. (2000). Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 234–240.

Blake, J. A. (2005). Bioinformatics and the biomedical literature: Current challenges. In *Invited talk: ACL-ISMB Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, Detroit. Association for Computational Linguistics.

Blaschke, C., Andrade, M. A., Ouzounis, C., and Valencia, A. (1999). Automatic extraction of biological information from scientific text: protein-protein interactions. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Proceedings of the International Conference on Intelligent Systems in Molecular Biology (ISMB99)*, pages 60–67. The AAAI Press.

Blaschke, C., Leon, E. A., Krallinger, M., and Valencia, A. (2005). Evaluation of biocreative assessment of task 2. *BMC Bioinformatics*, **6 (Suppl. 1)**(S16).

Bodenreider, O. and Pakhomov, S. V. (2003). Exploring adjectival modification in biomedical discourse across two genres. In S. Ananiadou and J. Tsujii, editors, *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, Sapporo, Japan. Association for Computational Linguistics.

Bos, J. (2005). Towards wide-coverage semantic interpretation. In *Proceedings of the 6th International Workshop on Computational Semantics (IWCS 6)*, pages 42–53.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Computational Linguistics*, **21**(4), 534–566.

Briscoe, E. J. and Carroll, J. (2002). Robust accurate statistical annotation of general text. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1499–1504, Las Palmas, Canary Islands.

Briscoe, T., Carroll, J., Graham, J., and Copestake, A. (2002). Relational evaluation schemes. In *Proceedings of the Beyond PARSEVAL workshop of the third LREC conference*, Las Palmas, Canary Islands.

Brody, T. (1999). The Interactive Fly: gene networks, development and the Internet. *Trends in Genetics*, **15**, 333–334.

Bunescu, R. and Mooney, R. (2005). A shortest path dependency kernel for relation extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 724–731, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Buyko, E., Wermter, J., Poprat, M., and Hahn, U. (2006). Automatically adapting an nlp core engine to the biology domain. In *BioLink & Bio-Ontologies SIG Meeting*.

Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Comput. Linguist.*, **22**(2), 249–254.

Carroll, J., Briscoe, T., and Sanfilippo, A. (1998). Parser evaluation: a survey and new proposal. In *Proceedings of the first international conference on language resources and evaluation (LREC)*, Granada, Spain.

Carroll, J., Minnen, G., and Briscoe, T. (1999). Corpus annotation for parser evaluation. In *Proceedings of the EACL workshop on linguistically interpreted corpora (LINC)*, Bergen, Norway.

Castaño, J., Zhang, J., and Pustejovsky, J. (2002). Anaphora resolution in biomedical literature. In *Proceedings of the International Symposium on Reference Resolution*, Alicante, Spain.

Chambers, N. and Allen, J. (2004). Stochastic language generation in a dialogue system: Toward a domain independent generator. In M. Strube and C. Sidner, editors, *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, pages 9–18, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Charniak, E. (1997). Statistical techniques for natural language parsing. *AI Magazine*, **18**(4), 33–44.

Charniak, E. (1999). A maximum-entropy-inspired parser. Technical report, Brown University.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 132–139. Morgan Kaufmann Publishers Inc.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan. Association for Computational Linguistics.

Chiang, J.-H. and Yu, H.-C. (2004). Extracting functional annotations of proteins based on hybrid text mining approaches. In *Proceedings of the BioCreAtIvE Challenge Evaluation Workshop*.

Chomsky, N. (1956). Three models for the description of language. *Information Theory, IEEE Transactions on*, **2**(3), 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Christ, O. (1994). A modular and flexible architecture for an integrated corpus query system. In *Proceedings of the Third Conference on Computational Lexicography and Text Research (COMPLEX '94)*, pages 23–32, Budapest.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 103–110, Barcelona, Spain.

Clark, S. and Curran, J. R. (in press—2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, **33**.

Clegg, A. B. and Shepherd, A. J. (2005). Evaluating and integrating treebank parsers on a biomedical corpus. In M. Jansche, editor, *Association for Computational Linguistics Workshop on Software CDROM*, Ann Arbor, MI.

Clegg, A. B. and Shepherd, A. J. (2007a). Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, **8**(24).

Clegg, A. B. and Shepherd, A. J. (in press—2007b). Text mining. In J. Keith, editor, *Bioinformatics*, Methods in Molecular Biology, New Jersey. Humana Press.

Cohen, A. M. (2005). Unsupervised gene/protein named entity normalization using automatically extracted dictionaries. In *Proceedings of the ACL-ISMB Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, pages 17–24, Detroit. Association for Computational Linguistics.

Cohen, K. B. and Hunter, L. (2004). Natural language processing and systems biology. In W. Dubitzky and F. Azuaje, editors, *Artificial intelligence methods and tools for systems biology*, Dordrecht. Kluwer.

Cohen, K. B., Demner-Fushman, D., Friedman, C., Hirschman, L., and Pestian, J. P. (2007). Front matter. In *Proceedings of the Workshop on BioNLP 2007: Biological, translational, and clinical language processing*, pages i–xvi, Prague, Czech Republic. Association for Computational Linguistics.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Phd, University of Pennsylvania.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, **29**(4), 589–637.

Collins-Thompson, K. and Callan, J. P. (2004). A language modeling approach to predicting reading difficulty. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 193–200, Boston, Massachusetts, USA. Association for Computational Linguistics.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, **20**(3), 273–297.

Crouch, R., Kaplan, R., King, T., and Riezler, S. (2002). A comparison of evaluation metrics for a broad coverage parser. In *Proceedings of the LREC-2002 workshop "Beyond PARSEVAL: Towards Improved Evaluation Measures for Parsing Systems"*, Las Palmas, Spain.

Culotta, A. and Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 423–429, Barcelona, Spain.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., Dimitrov, M., Dowman, M., Aswani, N., Roberts, I., Li, Y., and Shafirin, A. (2007). *Developing Language Processing Components with GATE Version 4 (a User Guide)*. The University of Sheffield, `http://www.gate.ac.uk/`.

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 133–140. Association for Computational Linguistics.

de Marcken, C. (1995). On the unsupervised induction of phrase-structure grammars. In *Proceedings of the 3rd Workshop on Very Large Corpora*. Association for Computational Linguistics.

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC2006)*, Genoa, Italy.

Ding, J., Berleant, D., Nettleton, D., and Wurtele, E. (2002). Mining MEDLINE: abstracts, sentences, or phrases? In *Proceedings of the 7th Pacific Symposium on Biocomputing*, pages 326–337, Lihue, Hawaii. World Scientific Publishing.

Divita, G., Browne, A. C., and Rindflesch, T. C. (1998). Evaluating lexical variant generation to improve information retrieval. In *Proceedings of the American Medical Informatics Association Annual Symposium*. Hanley & Belfus.

Domedel-Puig, N. and Wernisch, L. (2005). Applying GIFT, a Gene Interactions Finder in Text, to fly literature. *Bioinformatics*, **21**(17), 3582–3583.

Domingos, P., Kok, S., Poon, H., Richardson, M., and Singla, P. (2006). Unifying logical and statistical ai. In *AAAI*, Boston.

Dong, S. and Searls, D. B. (1994). Gene structure prediction by linguistic methods. *Genomics*, **23**(3), 540–551.

Doran, C., Niv, M., Baldwin, B., Reynar, J., and Srinivas, B. (1996). Mother of Perl: A multi-tier pattern description language. Technical report, Department of Computer Science, University of Pennsylvania.

Elhadad, N. (2006). *User-Sensitive Text Summarization: Application to the Medical Domain*. Phd, Columbia University.

Eom, J.-H., Kim, S., Kim, S.-H., and Zhang, B.-T. (2006). A tree kernel-based method for protein-protein interaction mining from biomedical literature. In E. G. Bremer, J. Hakenberg, E.-H. Han, D. P. Berrar, and W. Dubitzky, editors, *Knowledge Discovery in Life Science Literature, PAKDD 2006 International Workshop, Proceedings*, volume 3886 of *Lecture Notes in Computer Science*, Singapore. Springer.

Finkel, J., Dingare, S., Nguyen, H., Nissim, M., Manning, C., and Sinclair, G. (2004). Exploiting concepts for biomedical entity recognition: From syntax to the web. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 88–91, Geneva, Switzerland.

Fiszman, M., Rindflesch, T. C., and Kilicoglu, H. (2004). Abstraction summarization for managing the biomedical research literature. In D. Moldovan and R. Girju, editors, *HLT-NAACL 2004: Workshop on Computational Lexical Semantics*, pages 76–83, Boston, Massachusetts, USA. Association for Computational Linguistics.

Forbes-Riley, K. and Litman, D. (2004). Predicting emotion in spoken dialogue from multiple knowledge sources. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 201–208, Boston, Massachusetts, USA. Association for Computational Linguistics.

Franzén, K., Eriksson, G., Olsson, F., Lidén, L. A. P., and Cöster, J. (2002). Protein names and how to find them. *International Journal of Medical Informatics*, **67**(1–3), 49–61.

Friedman, C., Kra, P., Yu, H., Krauthammer, M., and Rzhetsky, A. (2001). GENIES: a natural-language processing system for the extraction of molecular pathways from journal articles. *Bioinformatics*, **17**(Suppl. 1), S74–S82.

Friedman, C., Kra, P., and Rzhetsky, A. (2002). Two biomedical sublanguages: a description based on the theories of Zellig Harris. *Journal of Biomedical Informatics*, **35**(4), 222–235.

Fukuda, K., Tsunoda, T., Tamura, A., and Takagi, T. (1998). Toward information extraction: Identifying protein names from biological papers. In *In Proceedings of the Pacific Symposium on Biocomputing (PSB'98)*, pages 707–718.

Fundel, K., Güttler, D., Zimmer, R., and Apostolakis, J. (2005). A simple approach for protein name identification: prospects and limits. *BMC Bioinformatics*, **6 (Suppl. 1)**(S15).

Fundel, K., Küffner, R., and Zimmer, R. (2007). RelEx—relation extraction using dependency parse trees. *Bioinformatics*, **23**(3), 365–371.

Futrelle, R. P. and Smith, L. C. (1982). Expert system using semantic representation of full text for question answering. In *Proceedings of the 45th Annual Meeting of the American Society for Information Science (ASIS-82)*, pages 103–106, Columbus, Ohio.

Futrelle, R. P., Dunn, C. E., Ellis, D. S., and Jr, M. J. P. (1991). Preprocessing and lexicon design for parsing technical text. In *Proceedings of the 2nd International Workshop on Parsing Technologies (IWPT '91)*, pages 31–40, Cancun, Mexico. Association for Computational Linguistics.

Futrelle, R. P., Grimes, A. E., and Shao, M. (2003). Extracting structure from html documents for language visualization and analysis. In *Proceedings of the Second International Workshop on Web Document Analysis (WDA2003)*, Edinburgh.

Gaizauskas, R., Demetriou, G., Artymiuk, P. J., and Willett, P. (2003). Protein structures and information extraction from biological texts: The PASTA system. *Bioinformatics*, **19**(1), 135–143.

Gaudan, S., Kirsch, H., and Rebholz-Schuhmann, D. (2005). Resolving abbreviations to their senses in Medline. *Bioinformatics*, **21**(18), 3658–3664.

Ge, N., Hale, J., and Charniak, E. (1998). A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, Hong Kong.

Gildea, D. and Hockienmaier, J. (2003). Identifying semantic roles using combinatory categorial grammar. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sapporo, Japan.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, **28**(3), 245–288.

Giuliano, C., Lavelli, A., and Romano, L. (2006). Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, pages 401–408, Trento, Italy. ACL.

Goadrich, M., Oliphant, L., and Shavlik, J. (2005). Learning to extract genic interactions using Gleaner. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Goldin, I. M. and Chapman, W. W. (2003). Learning to detect negation with 'not' in medical texts. In *ACM SIGIR'03 Workshop on Text Analysis and Search for Bioinformatics: Participant Notebook*, Toronto, Canada. Association for Computing Machinery.

Gopnik, M. (1972). *Linguistic Structures in Scientific Texts*. Mouton, The Hague.

Greenwood, M. A., Stevenson, M., Guo, Y., Harkema, H., and Roberts, A. (2005). Automatically acquiring a linguistically motivated genic interaction extraction system. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Grover, C., Carroll, J., and Briscoe, E. J. (1993). The alvey natural language tools grammar (4th release). Technical report 284, Cambridge University.

Grover, C., Lapata, M., and Lascarides, A. (2005). A comparison of parsing technologies for the biomedical domain. *Natural Language Engineering*, **11**(1), 27–65.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Hakenberg, J., Plake, C., Leser, U., Kirsch, H., and Rebholz-Schuhmann, D. (2005). LLL'05 challenge: Genic interaction extraction—identification of language patterns based on alignment and finite state automata. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Hara, T., Miyao, Y., and Tsujii, J. (2007). Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an hpsg parser. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 11–22, Prague, Czech Republic. Association for Computational Linguistics.

Harris, Z. S. (2002). The structure of science information. *Journal of Biomedical Informatics*, **35**(4), 215–221.

Hatzivassiloglou, V. and Weng, W. (2002). Learning anchor verbs for biological interaction patterns from published text articles. *International Journal of Medical Informatics*, **67**(1-3), 19–32.

Hatzivassiloglou, V., Duboué, P. A., and Rzhetsky, A. (2001). Disambiguating proteins, genes, and RNA in text: a machine learning approach. *Bioinformatics*, **17**(Supplement 1), 97S–106S.

Hayes, W. S. (2004). Text mining - next steps for drug discovery. In L. Hirschman and J. Pustejovsky, editors, *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, pages 49–49, Boston, Massachusetts, USA. Association for Computational Linguistics.

Hearst, M. (1999). Untangling text data mining. In D. Traum, editor, *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*. University of Maryland.

Heerman, V. (1930). Animal Crackers. Movie.

Hersh, W. (2005). Evaluation of biomedical text-mining systems: Lessons learned from information retrieval. *Briefings in Bioinformatics*, **6**(4), 344–356.

Hersh, W. and Bhupatiraju, R. T. (2003). Of mice and men (and rats and fruit flies): The TREC genomics track. In *ACM SIGIR'03 Workshop on Text Analysis and Search for Bioinformatics: Participant Notebook*, Toronto, Canada. Association for Computing Machinery.

Hersh, W. R. and Greenes, R. E. (1990). SAPHIRE: an information retrieval system featuring concept matching, automatic indexing, probabilistic retrieval, and hierarchical relationships. *Computers and Biomedical Research*, **23**, 410–425.

Hillard, D., Ostendorf, M., Stolcke, A., Liu, Y., and Shriberg, E. (2004). Improving automatic sentence boundary detection with confusion networks. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Short Papers*, pages 69–72, Boston, Massachusetts, USA. Association for Computational Linguistics.

Hirschman, L., Morgan, A. A., and Yeh, A. S. (2002). Rutabaga by any other name: extracting biological names. *Journal of Biomedical Informatics*, **35**(4), 247–259.

Hirschman, L., Yeh, A., Blaschke, C., and Valencia, A. (2005). Overview of BioCreAtIvE: critical assessment of information extraction for biology. *BMC Bioinformatics*, **6 (Suppl. 1)**(S1).

Horrocks, G. (1997). *Generative Grammar*. Longman, London.

Huang, C.-C. and Chang, J. S. (2006). Learning to parse bilingual sentences using bilingual corpus and monolingual cfg. In *ROCLING XVIII: Conference on Computational Linguistics and Speech Processing*, Taiwan.

Huang, Y., Lowe, H. J., Klein, D., and Cucina, R. J. (2005). Improved identification of noun phrases in clinical radiology reports using a high-performance statistical natural language parser augmented with the umls specialist lexicon. *Journal of the American Medical Informatics Association*, **12**(3), 275–285.

Hubbard, T., Andrews, D., Caccamo, M., Cameron, G., Chen, Y., Clamp, M., Clarke, L., Coates, G., Cox, T., Cunningham, F., Curwen, V., Cutts, T., Down, T., Durbin, R., Fernandez-Suarez, X. M., Gilbert, J., Hammond, M., Herrero, J., Hotz, H., Howe, K., Iyer, V., Jekosch, K., Kahari, A., Kasprzyk, A., Keefe, D., Keenan, S., Kokocinsci, F., London, D., Longden, I., McVicker, G., Melsopp, C., Meidl, P., Potter, S., Proctor, G., Rae, M., Rios, D., Schuster, M., Searle, S., Severin, J., Slater, G., Smedley, D., Smith, J., Spooner, W., Stabenau, A., Stalker, J., Storey, R., Trevanion, S., Ureta-Vidal, A., Vogel, J., White, S., Woodwark, C., and Birney, E. (2005). Ensembl 2005. *Nucleic Acids Research*, **33**, D447–D453.

Hudson, R. (1997). Word grammar. In R. Asher, editor, *The Encyclopedia of Language and Linguistics*, New York. Pergamon Press.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society of Information Science*, **47**(1), 70–84.

Jain, P., Mital, M. R., Kumar, S., Mukerjee, A., and Raina, A. M. (2004). Anaphora resolution in multi-person dialogues. In M. Strube and C. Sidner, editors, *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, pages 47–50, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Jenssen, T.-K., Lægreid, A., Komorowski, J., and Hovig, E. (2001). A literature network of human genes for high-throughput analysis of gene expression. *Nature Genetics*, **28**(1), 21–28.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing*. Prentice Hall, New Jersey.

Kabiljo, R., Stoycheva, D., and Shepherd, A. J. (2007). ProSpecTome: a new tagged corpus for protein named entity recognition. In *Proceedings of the ISMB BioLINK'07 SIG workshop*, Vienna.

Kaplan, R., Riezler, S., King, T. H., Maxwell III, J. T., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 97–104, Boston, Massachusetts, USA. Association for Computational Linguistics.

Kaplan, R. M. and Maxwell, J. T. (1996). LFG grammar writer's workbench. Technical report, Xerox Palo Alto Research Center.

Kato, Y., Seki, H., and Kasami, T. (2003). A comparative study on formal grammars for pseudoknots. *Genome Informatics*, **14**, 470–471.

Katrenko, S., Marshall, M. S., Roos, M., and Adriaans, P. (2005). Learning biological interactions from Medline abstracts. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Kawasaki, Y., Kazama, J., and Tsujii, J. (2003). Extracting biomedical ontology from textbooks and article abstracts. In *ACM SIGIR'03 Workshop on Text Analysis and Search for Bioinformatics: Participant Notebook*, Toronto, Canada. Association for Computing Machinery.

Kim, J.-D., Ohta, T., Tateisi, Y., and Tsujii, J. (2003). GENIA corpus – a semantically annotated corpus for bio-textmining. *Bioinformatics*, **19**(Suppl. 1), i180–i182.

Kinyon, A. (2001). A language-independent shallow-parser compiler. In *Proceedings of the 10th European Association for Computational Linguistics Meeting*, pages 322–329, Toulouse, France.

Klein, D. and Manning, C. D. (2002). Fast exact inference with a factored model for natural language parsing. *Advances in Neural Information Processing Systems*, pages 3–10.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics (ACL'03), Main Volume*, Sapporo, Japan. ACL.

Knight, K. and Marcu, D. (2000). Statistics-based summarization—step one: Sentence compression. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, Austin, Texas.

Krauthammer, M., Rzhetsky, A., Morozov, P., and Friedman, C. (2000). Using BLAST for identifying gene and protein names in journal articles. *Gene*, **259**, 242–252.

Krauthammer, M., Kra, P., Iossifov, I., Gomez, S. M., Hripcsak, G., Hatzivassiloglou, V., Friedman, C., and Rzhetsky, A. (2002). Of truth and pathways: Chasing bits of information through myriads of articles. *Bioinformatics*, **18**(Supplement 1), S249–S257.

Kübler, S. and Telljohann, H. (2002). Towards a dependency-oriented evaluation for partial parsing. In *Proceedings of the Beyond PARSEVAL workshop of the third LREC conference*, Las Palmas, Canary Islands.

Kulick, S., Bies, A., Liberman, M., Mandel, M., McDonald, R., Palmer, M., Schein, A., Ungar, L., Winters, S., and White, P. (2004). Integrated annotation for biomedical information extraction. In L. Hirschman and J. Pustejovsky, editors, *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, pages 61–68, Boston, Massachusetts, USA. Association for Computational Linguistics.

Lease, M. and Charniak, E. (2005). Parsing biomedical literature. In R. Dale, K.-F. Wong, J. Su, and O. Y. Kwong, editors, *Proceedings of the Second International Joint Conference on Natural Language Processing (IJCNLP'05)*, Jeju Island, Korea.

Lee, C., Hou, W.-J., and Chen, H.-H. (2004). Annotating multiple types of biomedical entities: A single word classification approach. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 80–83, Geneva, Switzerland.

Leech, G. N. (1974). *Semantics*. Penguin, Harmondsworth.

Leroy, G., Chen, H., and Martinez, J. D. (2003). A shallow parser based on closed-class words to capture relations in biomedical text. *Journal of Biomedical Informatics*, **36**, 145–158.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, **10**(8), 707–710.

Levow, G.-A. (2004). Assessing prosodic and text features for segmentation of Mandarin broadcast news. In B. Ramabhadran and D. Oard, editors, *HLT-NAACL 2004 Workshop: Interdisciplinary Approaches to Speech Indexing and Retrieval*, pages 28–32, Boston, Massachusetts, USA. Association for Computational Linguistics.

Levy, R. and Andrew, G. (2006). Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC2006)*, Genoa, Italy.

Li, X. and Roth, D. (2001). Exploring evidence for shallow parsing. In W. Daelemans and R. Zajac, editors, *Proceedings of CoNLL-2001*, pages 38–44. Toulouse, France.

Light, M., Qiu, X. Y., and Srinivasan, P. (2004). The language of bioscience: Facts, speculations, and statements in between. In L. Hirschman and J. Pustejovsky, editors, *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, pages 17–24, Boston, Massachusetts, USA. Association for Computational Linguistics.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJACI-95)*, Montreal, Quebec.

Lin, D. (2003). Dependency-based evaluation of minipar. In A. Abeillé, editor, *Building and using Parsed Corpora*, Dordrecht. Kluwer.

Lin, Y.-H. and Liang, T. (2004). Pronominal and sortal anaphora resolution for biomedical literature. In *Proceedings of the ROCLING XVI Conference on Computational Linguistics and Speech Processing*, Taipei, Taiwan.

Litrán, J. C. C., Satou, K., and Torisawa, K. (2004). Improving the identification of non-anaphoric it using support vector machines. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 58–61, Geneva, Switzerland.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1994). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, **19**(2), 313–330.

McClosky, D., Charniak, E., and Johnson, M. (2006). Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 337–344, Sydney, Australia. Association for Computational Linguistics.

McCray, A. T., Bodenreider, O., Malley, J. D., and Browne, A. C. (2001). Evaluating UMLS strings for natural language processing. In *Proceedings of the American Medical Informatics Association Symposium*, pages 448–452. Hanley and Belfus, Inc.

Merlo, P. and Stevenson, S. (2001). Automatic verb classification based on statistical distributions of argument structure. *Computational Linguistics*, **27**(3), 373–408.

Mika, S. and Rost, B. (2004). Protein names precisely peeled off free text. *Bioinformatics*, **20**(S1), i241–i247.

Miller, S., Fox, H., Ramshaw, L., and Weischedel, R. (2000). A novel use of statistical parsing to extract information from text. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Munroe, R. (2006). Computational linguistics. `http://xkcd.com/114/`.

Mutalik, P. G., Deshpande, A., and Nadkarni, P. M. (2001). Use of general-purpose negation detection to augment concept indexing of medical documents: A quantitative study using the umls. *Journal of the American Medical Informatics Association*, **8**(6), 598–609.

Nédellec, C. (2005). Learning Language in Logic—Genic Interaction Extraction Challenge. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Niu, Y., Hirst, G., McArthur, G., and Rodriguez-Gianolli, P. (2003). Answering clinical questions with role identification. In S. Ananiadou and J. Tsujii, editors, *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, Sapporo, Japan. Association for Computational Linguistics.

Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., Kumar, S., Shen, L., Smith, D., Eng, K., Jain, V., Jin, Z., and Radev, D. (2004). A smorgasbord of features for statistical machine translation. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 161–168, Boston, Massachusetts, USA. Association for Computational Linguistics.

Park, K.-M., Kim, S.-H., Lee, K.-J., Lee, D.-G., and Rim, H.-C. (2004). Incorporating lexical knowledge into biomedical NE recognition. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 76–79, Geneva, Switzerland.

Polanyi, L., Culy, C., van den Berg, M., Thione, G. L., and Ahn, D. (2004). A rule based approach to discourse parsing. In M. Strube and C. Sidner, editors, *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, pages 108–117, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Popelínský, L. and Blaťák, J. (2005). Learning genic interactions without expert domain knowledge: Comparison of different ILP algorithms. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Pradhan, S. S., Ward, W. H., Hacioglu, K., Martin, J. H., and Jurafsky, D. (2004). Shallow semantic parsing using support vector machines. In D. M. Susan Dumais and S. Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 233–240, Boston, Massachusetts, USA. Association for Computational Linguistics.

Pruitt, K. D., Katz, K. S., Sicotte, H., and Maglott, D. R. (2000). Introducing RefSeq and LocusLink: curated human genome resources at the NCBI. *Trends in Genetics*, **16**(1), 44–47.

Pustejovsky, J., Castano, J., Zhang, J., Kotecki, M., and Cochran, B. (2002). Robust relational parsing over biomedical literature: extracting inhibit relations. In *Proceedings of the 7th Pacific Symposium on Biocomputing*, pages 326–337, Lihue, Hawaii. World Scientific Publishing.

Pyysalo, S., Ginter, F., Pahikkala, T., Boberg, J., Järvinen, J., and Salakoski, T. (2006a). Evaluation of two dependency parsers on biomedical corpus targeted at protein-protein interactions. *International Journal of Medical Informatics*, **75**(6), 430–442.

Pyysalo, S., Salakoski, T., Aubin, S., and Nazarenko, A. (2006b). Lexical adaptation of link grammar to the biomedical sublanguage: A comparative evaluation of three approaches. *BMC Bioinformatics*, **7(Suppl 3)**(S2).

Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., and Salakoski, T. (2007a). BioInfer: a corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, **8**(50).

Pyysalo, S., Ginter, F., Laippala, V., Haverinen, K., Heimonen, J., and Salakoski, T. (2007b). On the unification of syntactic annotations under the Stanford dependency scheme: A case study on BioInfer and GENIA. In *Proceedings of the Workshop*

*on BioNLP 2007: Biological, translational, and clinical language processing*, pages 25–32, Prague, Czech Republic. Association for Computational Linguistics.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania.

Rebhan, M., Chalifa-Caspi, V., Prilusky, J., and Lancet, D. (1997). GeneCards: encyclopedia for genes, proteins and diseases. `http://bioinformatics.weizmann.ac.il/cards`.

Reynar, J. C. and Ratnaparkhi, A. (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, D.C.

Riedel, S. and Klein, E. (2005). Genic interaction extraction with semantic and syntactic chains. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany.

Rindflesch, T. C., Rajan, J., and Hunter, L. (2000). Extracting molecular binding relationships from biomedical text. In *Proceedings of the 6th Applied Natural Language Processing Conference*, pages 188–195. Association for Computational Linguistics.

Ringger, E., Moore, R. C., Charniak, E., Vanderwende, L., and Suzuki, H. (2004). Using the penn treebank to evaluate non-treebank parsers. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC2004)*, volume IV.

Roark, B. (2002). Evaluating parser accuracy using edit distance. In *Proceedings of the LREC-2002 workshop "Beyond PARSEVAL: Towards Improved Evaluation Measures for Parsing Systems"*, Las Palmas, Spain.

Rosario, B. and Hearst, M. (2004). Classifying semantic relations in bioscience texts. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 430–437, Barcelona, Spain.

Rössler, M. (2004). Adapting an NER-system for German to the biomedical domain. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 92–95, Geneva, Switzerland.

Rzhetsky, A., Iossifov, I., Koike, T., Krauthammer, M., Kra, P., Morris, M., Yu, H., Duboué, P. A., Weng, W., Wilbur, W. J., Hatzivassiloglou, V., and Friedman, C. (2004). GeneWays: a system for extracting, analyzing, visualizing, and integrating molecular pathway data. *Journal of Biomedical Informatics*, **37**, 43–53.

Sætre, R., Søvik, H., Amble, T., and Tsuruoka, Y. (2006). GeneTUC, GENIA and Google: Natural language understanding in molecular biology literature. In C. Priami, X. Hu, Y. Pan, and T. Y. Lin, editors, *Transactions on Computational Systems Biology V*, volume 4070 of *Lecture Notes in Computer Science*, pages 68–82. Springer, Heidelberg.

Sampson, G. (August 2000). A proposal for improving the measurement of parse accuracy. *International Journal of Corpus Linguistics*, **5**, 53–68(16).

Sampson, G. and Babarczy, A. (2003). A test of the leaf-ancestor metric for parse accuracy. *Journal of Natural Language Engineering*, **9**(4).

Sanchez, O. and Poesio, M. (2005). Acquisition of causal knowledge from text: Applications to bioinformatics. In *First International Symposium on Semantic Mining in Biomedicine (SMBM) poster session*, Hinxton, UK.

Santorini, B. (1990). Part-of-speech tagging guidelines for the Penn Treebank project (3rd revision, 2nd printing). `ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz`.

Saric, J., Jensen, L. J., Bork, P., Ouzounova, R., and Rojas, I. (2004). Extracting regulatory gene expression networks from pubmed. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 191–198, Barcelona, Spain.

Schneider, G. (1998). *A Linguistic Comparison of Constituency, Dependency and Link Grammar*. Msc thesis, University of Zurich.

Schneider, G. (2003). A low-complexity, broad-coverage probabilistic dependency parser for English. In *Proceedings of HLT-NAACL 2003 Student Research Workshop*, pages 31–36, Edmonton, Canada.

Schneider, G., Dowdall, J., and Rinaldi, F. (2004a). A robust and deep-linguistic theory applied to large-scale parsing. In *Coling 2004 Workshop on Robust Methods in the Analysis of Natural Language Data (ROMAND 2004)*, Geneva.

Schneider, G., Rinaldi, F., Kaljurand, K., and Hess, M. (2004b). Steps towards a genia dependency treebank. In *Proceedings of the Third Workshop on Treebanks and Linguistic Theories (TLT 2004)*, pages 137–148, Tübingen, Germany.

Schwartz, A. and Hearst, M. (2003). A simple algorithm for identifying abbreviation definitions in biomedical text. In *Proceedings of the 2003 Pacific Symposium on Biocomputing*, pages 451–462. World Scientific Publishing Co. Pte. Ltd.

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

Shatkay, H., Edwards, S., Wilbur, W. J., and Boguski, M. (2000). Genes, themes, and microarrays: Using information retrieval for large-scale gene analysis. In P. Bourne, M. Gribskov, R. Altman, N. Jensen, D. Hope, T. Lengauer, J. Mitchell, E. Scheeff, C. Smith, S. Strande, and H. Weissig, editors, *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 317–328. AAAI Press.

Shi, Z., Gu, B., Popowich, F., and Sarkar, A. (2005). Synonym-based query expansion and boosting-based re-ranking: A two-phase approach for genomic information retrieval. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, Gaithersburg, Maryland.

Shimbo, M. and Hara, K. (2007). A discriminative learning model for coordinate conjunctions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 610–619.

Shimbo, M., Tamamori, S., and Matsumoto, Y. (2004). Finding cue expressions for knowledge extraction from scientific text: early results. In B. H. Kang, editor, *Proceedings of the Pacific Knowledge Acquisition Workshop*, pages 1–13, Auckland, NZ.

Sleator, D. and Temperley, D. (1993). Parsing English with a Link Grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies (IWPT 1'93)*, Tilburg, Netherlands.

Smith, B. (2003). The logic of biological classification and the foundations of biomedical ontology. In D. Westerstahl, editor, *Invited Papers from the 10th International Conference in Logic Methodology and Philosophy of Science*, Oviedo, Spain.

Smith, L., Rindflesch, T., and Wilbur, W. J. (2004). MedPost: a part-of-speech tagger for biomedical text. *Bioinformatics*, **20**(14), 2320–2321.

Smith, T. C. and Cleary, J. G. (2003). All of MEDLINE indexed to the Gene Ontology. In *Sixth Annual Bio-Ontologies Meeting booklet*, pages 7–12. Manchester University.

Song, Y., Kim, E., Lee, G. G., and kee Yi, B. (2004). POSTBIOTM-NER in the shared task of BioNLP/JNLPBA2004. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 100–103, Geneva, Switzerland.

Song, Y., Kim, E., Lee, G. G., and kee Yi, B. (2005). POSBIOTM-NER: a trainable biomedical named-entity recognition system. *Bioinformatics*, **21**(11), 2794–2796.

Sprinzak, E., Sattath, S., and Margalit, H. (2003). How reliable are experimental protein-protein interaction data? *Journal of Molecular Biology*, **327**(5), 919–923.

Srinivas, B., Doran, C., Hockey, B. A., and Joshi, A. (1996). An approach to robust partial parsing and evaluation metrics. In *Proceedings of the Eight European Summer School In Logic, Language and Information*, Prague, Czech Republic.

Sullivan, D. (2001). *Document Warehousing and Text Mining*. John Wiley and Sons, Inc., New York.

Svolovits, P. (2003). Adding a medical lexicon to an English parser. In *Proceedings of the AMIA 2003 Annual Symposium*, pages 252–259. American Medical Informatics Association.

Swanson, D. R. (1986). Fish oil, raynaud's syndrome, and undiscovered public knowledge. *Perspectives in Biology and Medicine*, **30**(1), 7–18.

Tanabe, L., Xie, N., Thom, L. H., Matten, W., and Wilbur, W. J. (2005). GENETAG: a tagged corpus for gene/protein named entity recognition. *BMC Bioinformatics*, **6 (Suppl. 1)**(S3).

Tateisi, Y. and Tsujii, J. (2004). Part-of-speech annotation of biology research abstracts. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC2004)*, volume IV, pages 1267–1270.

Tateisi, Y., Ohta, T., and Tsujii, J. (2004). Annotation of predicate-argument structure of molecular biology text. In *Proceedings of the IJCNLP-04 workshop on Beyond Shallow Analyses*, Sanya City, Hainan Island, China. JST CREST (Japan).

Tateisi, Y., Yakushiji, A., Ohta, T., and Tsujii, J. (2005). Syntax annotation for the GENIA corpus. In *Proceedings of the International Joint Conference on Natural Language Processing 2005, Companion volume*, pages 222–227, Jeju Island, Korea.

Temkin, J. M. and Gilder, M. R. (2003). Extraction of protein interaction information from unstructured text using a context-free grammar. *Bioinformatics*, **19**(16), 2046–2053.

Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., and Salakoski, T. (2005). Regularized least-squares for parse ranking. In *Proceedings of the 6th International Symposium on Intelligent Data Analysis (IDA 2005)*, pages 464–474, Madrid, Spain.

Tsuruoka, Y. and Tsujii, J. (2003a). Boosting precision and recall of dictionary-based protein name recognition. In S. Ananiadou and J. Tsujii, editors, *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, Sapporo, Japan. Association for Computational Linguistics.

Tsuruoka, Y. and Tsujii, J. (2003b). Probabilistic term variant generator for biomedical terms. In *Proceedings of the 26th Annual International ACM SIGIR Conference*, pages 167–173, Toronto, Canada. Association for Computing Machinery.

Tsuruoka, Y., Miyao, Y., and Tsujii, J. (2004). Towards efficient probabilistic hpsg parsing: integrating semantic and syntactic preference to guide the parsing. In *Proceedings of IJCNLP-04 Workshop: Beyond shallow analyses*. Asia Federation of Natural Language Processing.

Vachon, T. (2004). Applications of text mining in the pharmaceutical industry. Presentation (given to JNLPBA04), Novartis Research, Basel, Switzerland.

van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Butterworths, London.

Verspoor, K., Cohen, K. B., Goertzel, B., and Mani, I. (2006). Introduction to bionlp '06. In *Proceedings of the HLT-NAACL BioNLP Workshop on Linking Natural Language and Biology*, pages iii–iv, New York, New York. Association for Computational Linguistics.

Voutilainen, A. (1995). Morphological disambiguation. In F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila, editors, *Constraint Grammar: a Language-Independent System for Parsing Unrestricted Text*, pages 165–284, Berlin. Mouton de Gruyter.

Walker, D. J., Clements, D. E., Darwin, M., and Amtrup, J. W. (2001). Sentence boundary detection: A comparison of paradigms for improving MT quality. In *Proceedings of the MT Summit VIII conference*, Santiago, Spain. European Association for Machine Translation.

Wattarujeekrit, T., Shah, P., and Collier, N. (2004). PASBio: predicate-argument structures for event extraction in molecular biology. *BMC Bioinformatics*, **5**(155).

Weeber, M., Vos, R., Klein, H., de Jong-van den Berg, L. T. W., Aronson, A. R., and Molema, G. (2003). Generating hypotheses by discovering implicit associations in the literature: A case report of a search for new potential therapeutic uses for thalidomide. *Journal of the American Medical Informatics Association*, **10**, 252–259.

White, J. A., Maltais, L. J., and Nebert, D. W. (2001). An increasingly urgent need for standardized gene nomenclature. `http://www.nature.com/ng/web_specials/nomen/nomen_article.html`.

Widdows, D., Peters, S., Cederberg, S., Chan, C.-K., Steffen, D., and Buitelaar, P. (2003). Unsupervised monolingual and bilingual word-sense disambiguation of

medical documents using UMLS. In S. Ananiadou and J. Tsujii, editors, *Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine*, Sapporo, Japan. Association for Computational Linguistics.

Witte, R., Kappler, T., and Baker, C. J. O. (2007). Ontology design for biomedical text mining. In C. Baker and K.-H. Cheung, editors, *The Semantic Web—Revolutionizing Knowledge Discovery in the Life Sciences*, pages 281–313. Springer-Verlag, Heidelberg.

Xiao, J., Su, J., Zhou, G., and Tan, C. (2005). Protein-protein interaction: A supervised learning approach. In *Proceedings of the First International Symposium on Semantic Mining in Biomedicine*, Hinxton, UK.

Yakushiji, A., Tateisi, Y., Miyao, Y., and Tsujii, J. (2001). Event extraction from biomedical papers using a full parser. In *Proceedings of the Sixth Pacific Symposium on Biocomputing*, pages 384–395. World Scientific Publishing.

Yakushiji, A., Tateisi, Y., Miyao, Y., and Tsujii, J. (2004). Finding anchor verbs for biomedical ie using predicate-argument structures. In *The Companion Volume to the Proceedings of 42nd Annual Meeting of the Association for Computational Linguistics*, pages 158–161, Barcelona, Spain. Association for Computational Linguistics.

Yu, Z., Tsuruoka, Y., and Tsujii, J. (2003). Automatic resolution of ambiguous abbreviations in biomedical texts using support vector machines and one sense per discourse hypothesis. In *ACM SIGIR'03 Workshop on Text Analysis and Search for Bioinformatics: Participant Notebook*, Toronto, Canada. Association for Computing Machinery.

Zhao, S. (2004). Named entity recognition in biomedical texts using an HMM model. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 84–87, Geneva, Switzerland.

Zhou, G. (2004). Recognizing names in biomedical texts using Hidden Markov Model and SVM plus Sigmoid. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 1–7, Geneva, Switzerland.

Zhou, G. and Su, J. (2004). Exploring deep knowledge resources in biomedical name recognition. In N. Collier, P. Ruch, and A. Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 96–99, Geneva, Switzerland.

Zhou, G., Zhang, M., Ji, D., and Zhu, Q. (2007). Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proceedings of the*

*2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 728–736.