Syntactic Pattern Matching with GraphSpider and MPL

Andrew B. Clegg
Research Dept. of Structural
and Molecular Biology
University College London
andrew.clegg@gmail.com

Adrian J. Shepherd
School of Crystallography
Birkbeck, University of London
a.shepherd@mail.cryst.bbk.ac.uk

Abstract

We present MPL (Metapattern Language), a new formalism for defining patterns over dependency-parsed text, and GraphSpider, a matching engine for extracting dependency subgraphs which match against MPL patterns. Using a regexp-like syntax, MPL allows the definition of subgraphs matching user-specified patterns which can be constrained by word or word class, part-of-speech tag, dependency type and direction, and presence of named variables in particular locations. Although MPL and GraphSpider are general-purpose, we developed a set of patterns to capture biomolecular interactions which achieved very high precision results (92.6% at 31.2% recall) on the LLL Challenge corpus. MPL specifications and pattern sets, and the GraphSpider software, are available on SourceForge: http://graphspider.sf.net/

1 Introduction

Various syntactic parsing methods have been used to provide input data for natural-language processing (NLP) tasks in the biomedical domain. The rich grammatical information provided by different kinds of parsers can be useful in relationship extraction (Riedel and Klein, 2005) and classification (Rosario and Hearst, 2004), event extraction (Yakushiji et al., 2001), semantic interpretation (Grover et al., 2005), named-entity recognition (Finkel et al., 2004), term extraction (Aronson, 2001), and information retrieval (Shi et al., 2005). This diversity of usage scenarios for syntax data, coupled with the growing availability of syntactically-annotated biomedical text (Tateisi et al., 2005; Kulick et al., 2004; Pyysalo et al., 2007a), suggests that general-purpose tools for querying and manipulating syntactic structures may be useful to researchers in this field.

In order to facilitate our experiments with dependency parsing of biomedical sentences, we developed a language for defining patterns over dependency graphs, plus a tool for matching patterns to sentences and extracting graph nodes of interest. Although this work was carried out in the context of a research project in gene/protein interaction extraction, there is nothing in either the language specification or software which is specific to this task or indeed to biological applications in general.

Several tools already exist for searching constituent tree representations of parsed sentences. The best-known of these is TGrep2[1] which is itself a successor to the original tgrep[2] that was developed alongside the Penn Treebank (Marcus et al., 1994). Both allow the construction of patterns of arbitrary complexity resembling hierarchical regular expressions, which constrain searches by words, part-of-speech (POS) tags and constituent labels, along with the positions in a subtree these elements must hold relative to each other. Similar features are provided by Tregex (Levy and Andrew, 2006). All of these tools were designed to operate on Penn Treebank-style trees; equivalents for other annotation schemas are provided by JAPE, part of the GATE package (Cunningham et al., 2007), CQP, part of the IMS Corpus Workbench (Christ, 1994), and Mother of Perl (Doran et al., 1996). NetGraph (Mírovský, 2008) is designed specifically for dependency graphs but is rather a complex client-server system. The OntoGene project at the University of Zurich has developed a system very similar to ours (Rinaldi et al., 2006), but it is available only via a web interface, with no source or binaries, making it much less useful for other researchers. None of these tools can perform noun phrase chunking on-the-fly (see below), and none provide native support for the Stanford dependency grammar (de Marneffe et al., 2006), which our own systems use, and which has been proposed as a convenient common schema for syntactic annotation and processing of biomedical text (Pyysalo et al., 2007b).

[1] http://tedlab.mit.edu/~dr/TGrep2/

[2] http://www.ldc.upenn.edu/ldc/online/treebank/

2 Metapattern Language (MPL)

An MPL file is composed of three kinds of rules, which taken together, specify a set of patterns to search for. **Match rules** define variables which hold either plain text strings or regular expressions, designed to match words, POS tags or dependency types in a graph. For example, the following rule declares a variable `@VERB` which matches any two or three-character string starting with the letters *VB*, and is designed to match POS tags like *VBN*, *VBZ* etc.:

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

Pattern rules are composed of variables, literal strings and connectors, and describe the subgraphs which the matching engine must attempt to find in the input sentences. Subgraphs are defined in terms of nodes (words with POS tags) connected by directional, labeled dependencies, although of course wildcard variables can be used in order to leave any of these elements unspecified. The simplest possible pattern simply matches a single node by tag and word, for example `NN~~regulation`. The following pattern matches fragments of the form ⟨*agent*⟩ *inhibits* ⟨*target*⟩; *inhibits* is specified literally, as are the POS tags (*VBZ* and *NN*) and the *dobj* direct object dependency, while the agent and target entities and the subject dependency refer to variables:

```
pattern
VBZ~~inhibits
   ( @NSUBJ NN~~@AGENT )
   ( dobj NN~~@TARGET )
end
```

Finally, **replacement rules** allow variations on explicitly-defined patterns to be generated automatically, to capture known wording alternatives or common variations on simple structures. They take the form of string replacement rules that are applied in turn to each of the patterns in the MPL file. They can operate on any part of a pattern rule (from a single word, POS tag or dependency to an entire subgraph) as long as the resulting pattern is well-formed. The following rule shows how a node matching a single string can be replaced by one matching a simple prepositional phrase:

```
replace @TARGET = expression
   ( prep_of NN~~@TARGET )
```

Applying this replacement rule to the example pattern given above results in the following pattern:

```
VBZ~~inhibits
   ( @NSUBJ NN~~@AGENT )
   ( dobj NN~~expression
     ( prep_of NN~~@TARGET ) )
```

This automatically-generated pattern will match sentence fragments like ⟨*agent*⟩ *inhibits expression of* ⟨*target*⟩. Note that since the pattern is defined over syntactic dependencies rather than linear strings of text, additional words intervening between any of the words covered by the pattern will not stop it matching, provided the grammatical relations between the matched words are correct. In other words, a sentence like ⟨*agent*⟩ *inhibits* ⟨*entity*⟩*-mediated expression of* ⟨*target*⟩ will still be matched.

Although one could attempt to define match rules to recognize named entities using regular expressions or lists of entity names, this is not likely to be successful except in very specific circumstances. Instead, we suggest that the user preprocesses the text with a named entity recognizer, then replaces all the entities found with placeholders (e.g. *Entityaa* ... *Entityzz*) that can be easily and unambiguously found by match rules using regular expressions. If a record of placeholder substitutions is kept, the original entity name behind each placeholder can be recovered trivially. Alternatively, if the variables such as `@AGENT` and `@TARGET` are defined with unrestricted wildcard expressions, then any word—or chunked phrase, see below—playing the appropriate syntactic role in a pattern will be identified. This approach turns the problem of named-entity recognition on its head, by assuming that *any* names found in expressions like *X inhibits expression of Y* represent biologically-interesting entities.

MPL offers several features beyond those described here, but does not yet support cycles or multiple parentage, meaning that its patterns are strictly trees rather than graphs. However, in evaluation (see below) we found no occasions when this was problematic.

3 GraphSpider

GraphSpider is a Java-based tool for performing MPL searches. It requires the Stanford parser distribution[3] to be installed, although it can accept the output of any constituent parser that uses

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

standard bracketed tree notation and Penn Treebank labels, or pre-generated Stanford-style dependency graphs in its own file format. If the text is supplied in trees, the conversion algorithm supplied with the Stanford parser is used to generate the dependency graphs (de Marneffe et al., 2006).

GraphSpider consists of two major components, an MPL parser and a matching engine. The MPL parser is responsible for reading in a pattern file supplied by the user, applying all replacement rules where possible in order to generate variant patterns, and compiling the patterns into in-memory representations. The matching engine then iterates over the sentences supplied, and for each one, finds every location where any pattern matches against the dependency graph of the sentence, including overlapping matches and locations where multiple patterns can match. It can then output the results for the sentence in one of several user-specified formats, ranging in scope from the entire sentence to just the nodes (words) that have matched against variables in the pattern.

Optionally, GraphSpider can apply a noun-phrase chunking algorithm to all constituent trees before converting them into dependency graphs. This simply identifies noun phrases with internal structure and flattens them into single words with the spaces replaced by underscores. The resulting graphs will tend to be much simpler, with single nodes encapsulating entire compound noun phrases (including adjectives, determiners and participles). However, MPL patterns must be written specifically to target chunked graphs, as pattern rules designed to match against traditional word-per-node graphs will not work on them. There is also a mechanism for plugging in Java classes for ad-hoc post-processing of the results, which we used to implement negation filtering.

4 Applications

Given a set of patterns capturing syntactic representations of biological events or interactions, and a corpus of parsed sentences, GraphSpider can be used to extract the entities, the keywords describing their relationships, and optionally any other words of interest. To a certain extent, phrasing variations and parse errors can be accounted for by the use of replacement rules to generate variant patterns automatically.

To test this approach, we developed a pattern set based on the training set from the LLL Challenge gene interaction task (Nédellec, 2005), and ran it against the test set, after replacing all gene/protein names in the sentences with placeholders. Although its coverage of the test set was comparatively low (31.2% recall), the predictions it did make were very accurate indeed (92.6% precision), suggesting that this method would be well-suited to unsupervised applications which require as little noise as possible in the results. We determined that these scores were achievable with as few as 29 hand-crafted patterns and 49 replacement rules, giving rise to 228 patterns in total (Clegg, 2008). Part of the reason for the low recall is that this method is sensitive to small parse errors which are not foreseen during the pattern engineering stage.

Another usage scenario is in exploratory corpus analysis and interactive text mining. By designing appropriate patterns, one can use GraphSpider to answer questions like "what entities bind to protein A?", "what temporal/locative modifiers are applied to expression of gene B?" (i.e. when/where does expression take place?), and "what verbs take a gene/protein phrase as their subject or object?". We have used this last technique to automatically extract keyword lists for the creation of further patterns.

The input/output and processing options supported by GraphSpider enable it to be used in a variety of alternative modes as well. For example, it can act simply as a noun phrase chunker, by bypassing the pattern matching engine completely in order to turn traditional constituent trees into chunked graphs. Similarly, it can strip the tree or graph annotation from a sentence and return just the plain text. And its ability to save and load dependency graphs in a human- and machine-readable format provides valuable functionality missing from the Stanford parsing toolkit.

We present MPL and GraphSpider in the hope that the NLP community finds them useful, and not just in the biological context where they were developed. All feedback, code or pattern contributions, and suggestions for future developments, are of course welcomed.

References

Alan R Aronson. 2001. Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program. In *Proceedings of the American Medical*

Informatics Association Symposium, pages 17–21. Hanley and Belfus, Inc.

Oliver Christ. 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.

Andrew B. Clegg. 2008. *Computational-Linguistic Approaches to Biological Text Mining*. PhD thesis, Birkbeck, London.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Cristian Ursu, Marin Dimitrov, Mike Dowman, Niraj Aswani, Ian Roberts, Yaoyong Li, and Andrey Shafirin. 2007. *Developing Language Processing Components with GATE Version 4 (a User Guide)*. The University of Sheffield, `http://www.gate.ac.uk/`.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC2006)*, Genoa, Italy, May.

Christine Doran, Michael Niv, Breck Baldwin, Jeffrey Reynar, and B. Srinivas. 1996. Mother of Perl: A multi-tier pattern description language. Technical report, Department of Computer Science, University of Pennsylvania.

Jenny Finkel, Shipra Dingare, Hoy Nguyen, Malvina Nissim, Christopher Manning, and Gail Sinclair. 2004. Exploiting context for biomedical entity recognition: From syntax to the web. In Nigel Collier, Patrick Ruch, and Adeline Nazarenko, editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA)*, pages 88–91, Geneva, Switzerland, August 28–29.

Claire Grover, Mirella Lapata, and Alex Lascarides. 2005. A comparison of parsing technologies for the biomedical domain. *Natural Language Engineering*, 11(1):27–65.

Seth Kulick, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, Lyle Ungar, Scott Winters, and Pete White. 2004. Integrated annotation for biomedical information extraction. In Lynette Hirschman and James Pustejovsky, editors, *HLT-NAACL 2004 Workshop: BioLINK 2004, Linking Biological Literature, Ontologies and Databases*, pages 61–68, Boston, Massachusetts, USA, May 6. Association for Computational Linguistics.

Roger Levy and Galen Andrew. 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, May.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Jiří Mírovský. 2008. Netgraph—making searching in treebanks easy. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008)*, pages 945–950, Hyderabad, India, January.

Claire Nédellec. 2005. Learning Language in Logic—Genic Interaction Extraction Challenge. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany, August.

Sampo Pyysalo, Filip Ginter, Juho Heimonen, Jari Björne, Jorma Boberg, Jouni Järvinen, and Tapio Salakoski. 2007a. BioInfer: a corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, 8(50), February.

Sampo Pyysalo, Filip Ginter, Veronika Laippala, Katri Haverinen, Juho Heimonen, and Tapio Salakoski. 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, June. Association for Computational Linguistics.

Sebastian Riedel and Ewan Klein. 2005. Genic interaction extraction with semantic and syntactic chains. In *Proceedings of Learning Language in Logic (LLL05)*, Bonn, Germany, August.

Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, and Martin Romacker. 2006. An environment for relation mining over richly annotated corpora: the case of GENIA. *BMC Bioinformatics*, 7 (Suppl 3)(S3), November.

Barbara Rosario and Marti Hearst. 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, July.

Zhongmin Shi, Baohua Gu, Fred Popowich, and Anoop Sarkar. 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, November.

Yuka Tateisi, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 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, October.

Akane Yakushiji, Yuka Tateisi, Yusuke Miyao, and Jun'ichi Tsujii. 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.