Computers and the Humanities **35:** 81–94, 2001.
© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

Finding Syntactic Structure in Unparsed Corpora

The Gsearch Corpus Query System

STEFFAN CORLEY[1], MARTIN CORLEY[2], FRANK KELLER[3], MATTHEW W. CROCKER[4] and SHARI TREWIN[5]
[1]Sharp Laboratories of Europe, Oxford Science Park, Oxford OX4 4GB, UK (E-mail: steffan@sharp.co.uk); [2]Department of Psychology and Human Communication Research Centre, University of Edinburgh, 7 George Square, Edinburgh EH8 9JZ, UK (E-mail: Martin.Corley@ed.ac.uk); [3]Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, UK (E-mail: Frank.Keller@ed.ac.uk); [4]Computational Linguistics, Saarland University, Box 15 11 50, 66041 Saarbrücken, Germany (E-mail: crocker@coli.uni-sb.de); [5]Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK (E-mail: S.Trewin@ed.ac.uk)

Abstract. The Gsearch system allows the selection of sentences by syntactic criteria from text corpora, even when these corpora contain no prior syntactic markup. This is achieved by means of a fast chart parser, which takes as input a grammar and a search expression specified by the user. Gsearch features a modular architecture that can be extended straightforwardly to give access to new corpora. The Gsearch architecture also allows interfacing with external linguistic resources (such as taggers and lexical databases). Gsearch can be used with graphical tools for visualizing the results of a query.

Key words: corpus search, parsing, syntactic annotation, SGML, computational linguistics, psycholinguistics

1. Introduction

Large on-line corpora constitute an increasingly important source of empirical data for language research. While numerous tools exist for lexical analysis of on-line texts, such as concordance software (e.g., CQP/Xkwic, Christ, 1994; WordSmith Tools, Berber Sardinha, 1996), researchers are increasingly interested in syntactic investigations of large corpora. Not only do such investigations directly support purely linguistic research, such as grammar development, they are also of interest to psycholinguists and computational linguists whose models increasingly rely on information such as structural and lexical frequencies.

While large on-line corpora are increasingly available for a variety of languages and genres, most are annotated with shallow information only, such as part of speech information, and possibly lemmas. Parsed corpora – due to the amount of human effort required to achieve reasonable accuracy and consistency – remain rare and relatively small.[1] Furthermore, syntactically annotated corpora are inevitably compromised by particular linguistic assumptions of their designers, which are in turn often influenced by what can be annotated with some degree of consistency. Thus researchers wishing to study particular or subtle linguistic distinctions must often work against the annotation scheme and search tools associated with a particular corpus.

In this paper we present the Gsearch system: a tool designed to facilitate the investigation of lexical and syntactic phenomena in unparsed corpora. Gsearch permits users to search for linguistic structures by processing a query based on a user definable grammar. The grammar notation is powerful and flexible: users define their own context free grammar where the terminals may be regular expressions over elements in the corpus (e.g., words, lemmas, part of speech tags) as well as calls to external databases and resources (e.g., WordNet, Miller et al., 1990).

Gsearch is intended as a flexible tool for scientists wishing to study corpora, and is not intended for accurate unsupervised parsing. The nature of lexical and syntactic ambiguity means that Gsearch will often return a parse which – while strictly correct with respect to the supplied grammar and query – is inappropriate for a particular substring. That is, given a grammar for some constituent (e.g., verb phrases (VPs)), Gsearch will match every substring in the corpus that **can** be parsed as a VP according to the grammar. Human inspection is still required to distinguish false positives from those items of interest. To facilitate this, tools are provided to easily view the parsed output, and also to construct and develop subcorpora.

2. Design

The modular design of Gsearch makes the system easily extendible and allows Gsearch to be integrated with existing corpus tools via a pipeline architecture.

2.1. SYSTEM ARCHITECTURE

The general architecture of the Gsearch system is depicted in Figure 1. The user specifies a query as a sequence of terminals and non-terminals of the grammar (see Section 3.2). The controller reads a sentence at a time from the corpus, and passes the sentence, together with the query, to the chart parser.

This chart parser is the core of Gsearch. The parser matches grammar terminals to corpus data fields using a regular expression matcher. These corpus data fields may either be part of the input corpus (e.g., words, lemmas, part of speech tags) or may be created on the fly using external linguistic resources (e.g., a lexical database). The parser attempts to build syntactic structure that matches the user-specified query. Successful matches are passed back to the controller for output.

The current version of Gsearch only supports lexical markup in the input corpus. Structural markup is ignored, with the exception of sentence markup, as the Gsearch parser computes matches only within a given sentence.

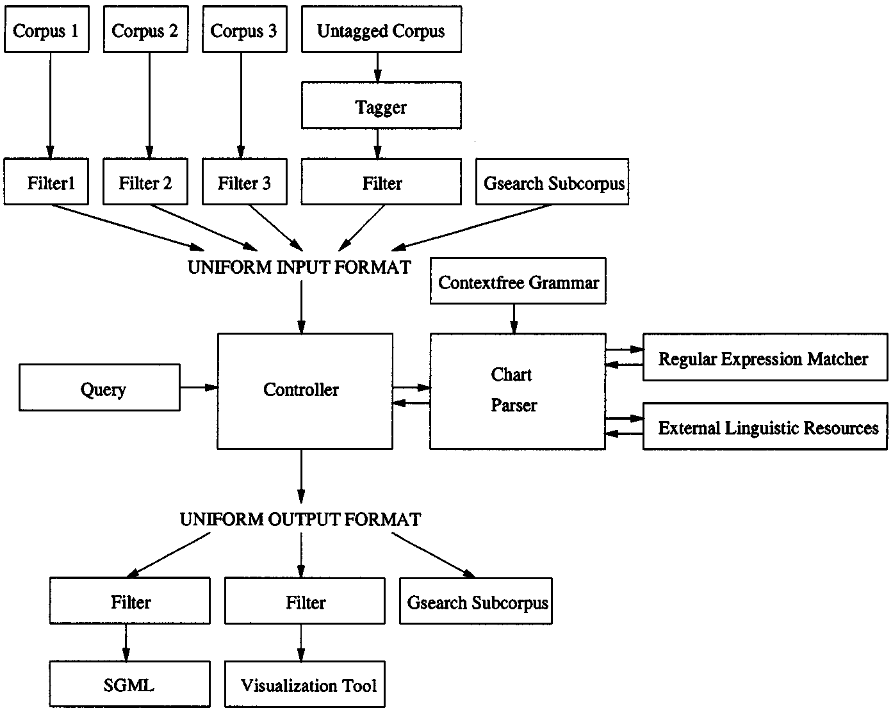


Figure 1. Gsearch architecture.

Gsearch relies on a Uniform Input Format (UIF) for the corpora it accesses. For each corpus supported by Gsearch, a filter is required that translates the specific format the corpus is encoded in into the UIF. The UIF will usually include the word tokens and their parts of speech as marked up in the corpus. However, Gsearch also supports a pipeline architecture where the input is taken from the output of a tagger running on a previously untagged corpus.

The output of a Gsearch query is encoded in a Uniform Output Format (UOF), which is passed through an output filter that transforms it into the specific format required for a given output module (such as a visualization tool to display tree structures).

This modular architecture allows for easy extendibility of the Gsearch system (see Section 5 for details). To add a new corpus, for instance, it is straightforward to write a filter that transforms the corpus format into Gsearch's UIF. Similarly, a new output module can be added by making the relevant UOF filter available.

Gsearch also supports a limited form of subcorpus handling by allowing the saving of the output of a Gsearch query in a subcorpus format, on which further Gsearch queries can be carried out.

2.2. PARSER

The parser is implemented as a bottom up chart parser (Earley, 1970; Aho and Ullman, 1972). A chart parser adds found constituents and partially matched rules to a table or chart, rather than simply using its agenda to maintain such information. The parser avoids adding the same constituent to the table twice; this prevents the parser from repeating the same (possibly unsuccessful) derivation twice based on slightly different analyses of a constituent near the bottom of the parse tree.

We have made two modifications to the standard chart parsing algorithm. Firstly, as we are looking for the user query at any sentence position, we add the goal at every sentence position. The second modification is an optimization to the base algorithm. If the user provides a large grammar containing many rules that are irrelevant to the current query, the basic algorithm would build large amounts of structure that could not satisfy the user query. We avoid much of this redundant work using an oracle (Aho and Ullman, 1972). An oracle is a table recording which constituents can be left-corner descendents of other constituents in any parse structure licensed by the grammar. For instance, using the grammar in Figure 2, `adj` may be a left-corner descendent of `np`, but `prep` may not. This oracle can be calculated automatically from the grammar.

Armed with such an oracle, we can discard any analysis that results in a constituent that cannot be a left-descendent of a current goal. We therefore avoid building much extraneous structure; this optimization results in impressive performance even when the user grammar is quite large.

2.3. PERFORMANCE

Table I gives an overview of the performance of Gsearch for different grammars and queries. The following grammars were tested:

- Minimal (19 rules): a minimal grammar (see Figure 2) was included to estimate baseline performance.
- Subcat (106 rules): a grammar for extracting subcategorization information (Lapata, 1999; Lapata and Brew, 1999). Its main focus is VP structure.
- Possessives (140 rules): a grammar for finding possessive NPs and other nominal constructions (Zamparelli, 1998). Coverage for VPs is minimal.

In this evaluation, Gsearch took between 14 and 167 minutes to search a 100 million word corpus, depending on the size of the grammar and the complexity of the query. The average search performance for the baseline grammar was 90,048 words per second. For the more comprehensive Subcat grammar Gsearch achieved a performance of 38,785 words per second, while its performance fell to 13,776 words per second for the specialized Possessive grammar.

Table I. Gsearch performance on the BNC (100 million words). The search times (in CPU seconds) are averaged over five Gsearch runs. The last column specifies the average search performance (in words per second). The data were obtained on a Sparc Ultra 5/10 (256MB memory, 333MHz clock rate, Solaris 2.5.1).

Grammar	Query	NP	VP	NP VP	NP PP PP	Avg. Perf.
Minimal	time	860	864	1,321	1,400	90,048
(19 rules)	hits	5,294,084	4,567,413	1,776,905	674,063	
Subcat	time	1,196	3,262	2,999	2,863	38,785
(106 rules)	hits	5,795,674	4,142,012	3,209,508	874,796	
Possessives	time	4,578	5,425	9,048	10,005	13,776
(140 rules)	hits	4,852,226	4,567,413	1,328,470	464,848	

3. Queries

The general form of a Gsearch query is as follows:

```
gsearch [<options>] <corpus> <grammar> <goal>
```

This section explains the arguments `<corpus>`, `<grammar>`, `<goal>`, and `<options>` in more detail.

3.1. CORPUS

The argument `<corpus>` specifies the name of the corpus to be searched. Each corpus has to be registered in a Gsearch setup file before it can be accessed. The setup file specifies a symbolic name for the corpus and contains information on its location and on the filter to be used for it. This information remains invisible to the user, who only needs to supply the symbolic name to access the corpus (although a user's local setup files can override global defaults).

Gsearch comes with filters for a number of corpora that are widely used. These include the British National Corpus (Burnard, 1995), the Brown Corpus (Francis et al., 1982), the SUSANNE Corpus (Sampson, 1995),[2] the Wall Street Journal Corpus (Marcus et al., 1993), the Frankfurter Rundschau Corpus,[3] and the NEGRA Corpus (Skut et al., 1997). Table II gives an overview of these corpora, detailing their markup, their approximate size, and the symbolic names used to access them in a standard Gsearch setup.

3.2. GRAMMAR

The argument `<grammar>` specifies a grammar file, which has to contain a context-free grammar represented in the Gsearch grammar format, which is similar to the format of definite clause grammars (DCGs).

Table II. Corpora available with Gsearch.

Corpus	Name	Markup	kWords	Description
BNC	`bnc`	`tag`, `lemma`	100,000	Balanced corpus of British English
Brown	`brown`	`tag`	1,000	Balanced corpus of American English
SUSANNE	`susie`	`tag`, `lemma`	128	Subcorpus of Brown corpus
Wall Street Journal	`wsj`	`tag`	8,000	Newspaper corpus; American English
Frankfurter Rundschau	`fr`	`tag`, `lemma`	40,000	Newspaper corpus; German
NEGRA	`negra`	`tag`, `morph`	268	Subcorpus of FR corpus

```
%% general part
vp --> v1
vp --> v1 np
vp --> v1 pp
v1 --> adv* verb adv*
np --> det n1+ pp*
np --> n1+ pp*
np --> np conj np
n1 --> adj* noun+
n1 --> n1 conj n1
pp --> prep np

%% corpus specific part
verb --> <tag=VV.*>       % main verb
noun --> <tag=NN.*>       % common noun
noun --> <tag=NP.*>       % proper noun
det  --> <tag=AT.*>       % determiner
adj  --> <tag=AJ.*>       % adjective
adj  --> <tag=ORD.*>      % ordinal
adv  --> <tag=AV0.*>      % adverb
prep --> <tag=PR.*>       % preposition
conj --> <tag=CJ.*>       % conjunction
```

Figure 2. Example of a Gsearch grammar.

Figure 2 gives an example of a Gsearch grammar. The grammar is divided into a general part, which typically consists of phrase-structure rules, and a corpus-specific part, which typically contains a mapping from the terminal symbols of the grammar to the part of speech tags defined in the corpus to be queried (here, the BNC).

The left hand side of a grammar rule contains a non-terminal, while the right hand side specifies a terminal symbol or one or more non-terminal symbols. For non-terminals, the characters ‘*’ and ‘+’ may be used to express the Kleene star and the closure operator, respectively.

Terminals take the form «field>=<regex», where <field> is a data field in the corpus, and <regex> specifies a regular expression to be matched against this field. The conjunction of regular expressions over data fields is supported via the ‘&’ operator, and negation can be expressed by using ‘!=’ instead of ‘=’.

The fields available for all corpora are word for the word token, and tag for the part of speech. Additional fields may be available if the corpus contains a richer markup, or if an external linguistic resource is used to supply the value of a given field (see Section 5.4).

3.3. GOAL

A Gsearch search expression (<goal>) is formulated as a sequence of terminal and non-terminal symbols of the grammar. As an example, consider the following query, which searches the BNC using the grammar GrammarBNC (given in Figure 2) for a pattern that consists of an NP followed by two PPs:

```
gsearch bnc GrammarBNC np pp pp
```

A subset of the result of this query is given in Figure 3.

The following query can be used to search the SUSANNE Corpus for instances of the noun *show* followed by a PP:

```
gsearch susie GrammarSusie "<tag=NN.* & word=show>" pp
```

Conjunction and negation of regular expressions over data fields is also supported in Gsearch queries.

3.4. OPTIONS

A number of command-line options can be specified to control how Gsearch processes a given query. There are two types of options:

Output options control which information is included in the output of a Gsearch query. The user can specify whether terminals and preterminals are to be included and which data fields are to be printed. The user can also control how much context is to be supplied with a matched sentence.

Parse options allow the user to influence the behavior of the Gsearch parser. This includes whether or not multiple matches per sentence will be returned, and whether or not punctuation is taken into account when matching a sentence. There is also an option that controls the maximum number of matches that Gsearch will return before terminating the search. This allows the testing of a grammar and a query without having to search the full corpus.

A detailed description of the options is provided in the Gsearch User Manual (Corley et al., 1999).

```
<div>
<head>"file=A/A0/A00 sentence=s40"</head>
<p>
<s40>There is
  <np><w AT0>no
    <n1><w NN1>limit</n1></np>
  <pp><w PRP>to
    <np><w AT0>the
      <n1><w NN1>number</n1></np></pp>
  <pp><w PRF>of
    <np><n1><w NN2>ways</n1></np></pp>
to raise money.</s40>
</p>
</div>

<div>
<head>"file=A/A0/A00 sentence=s81"</head>
<p>
<s81>Many people with AIDS have to spend
  <np><n1><w AJ0>long <w NN2>periods</n1></np>
  <pp><w PRF>of
    <np><n1><w NN1>time</n1></np></pp>
  <pp><w PRP>in
    <np><n1><w NN1>hospital</n1></np></pp>
unless there is someone at home who can
help and look after them.</s81>
</p>
</div>
```

Figure 3. SGML output.

4. Output and Visualization

By default, Gsearch generates SGML output, with phrase boundaries marked up as SGML attributes (a sample output for the query in Section 3.3 is given in Figure 3).

Gsearch output can be visualized using Viewtree, Adwait Ratnaparkhi's tool for displaying syntactic trees (distributed with Gsearch).[4] Viewtree allows browsing through the trees generated by a Gsearch query, and the access of a tree directly via its number (a unique identification is created for each sentence). Viewtree has the facility to generate Postscript files for individual trees. A Viewtree screenshot is provided in Figure 4.

As an alternative to Viewtree, the diagram engine Thistle (Calder, 1998a, b)[5] can be used to visualize the result of a Gsearch query. In contrast to Viewtree,

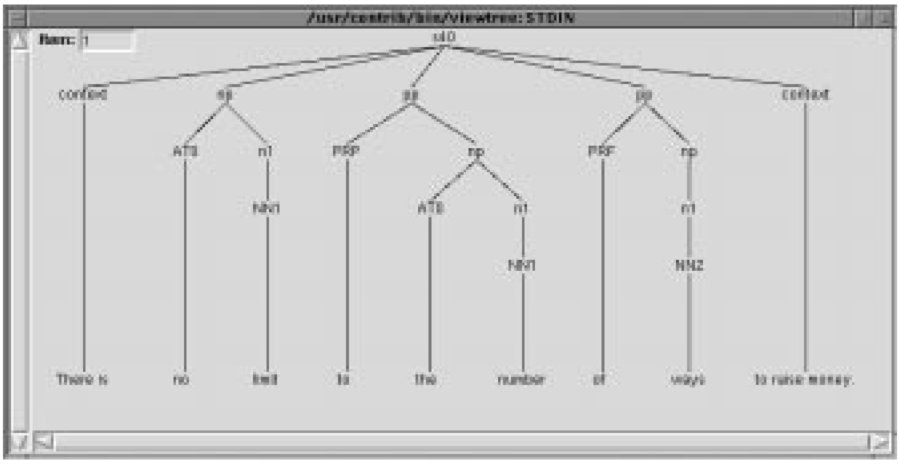
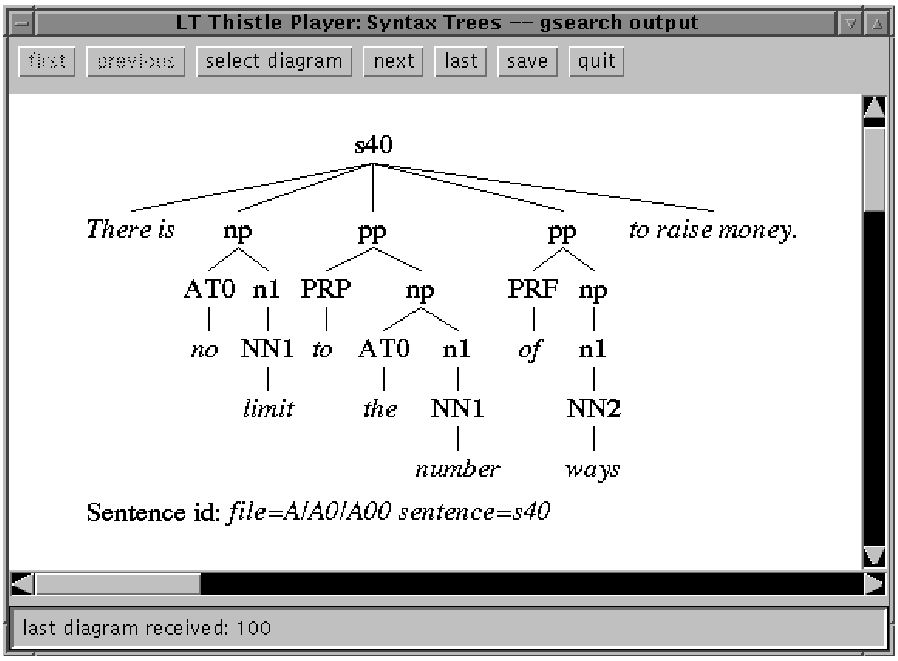


Figure 4. Viewtree output.



Sentence id: *file=A/A0/A00 sentence=s40*

last diagram received: 100

Figure 5. Thistle output.

Thistle allows the incremental processing of Gsearch output, i.e., the result of a query can be viewed tree by tree, even before the query is complete. Thistle comes with a full diagram editor that can save, edit, and print Gsearch output, as well as generate Postscript from it. A Thistle screenshot is provided in Figure 5.

Gextract (grammar extract) is a post-processor for Gsearch that allows random sampling from the result of a Gsearch query. This functionality is useful if a representative sample of a search result has to be obtained (e.g., because the full result is too large for manual inspection).

5. Extendibility

An important feature of Gsearch is its modular architecture (see Section 2.1), which allows for easy extendibility by adding new corpora, taggers, visualization tools, and external linguistic resources.

5.1. CORPORA

To make a new corpus searchable with Gsearch, one simply has to provide a filter that translates the specific format the corpus is encoded in into Gsearch's Uniform Input Format.

Writing such a filter is straightforward, based on the Programmer's Interface provided by Gsearch and the filter prototype that is part of the Gsearch distribution. To customize this prototype for a specific corpus, the user simply has to write a C function which scans a corpus file and returns the contents of the data fields marked up in the corpus. As an alternative, Gsearch filters can be implemented from scratch in other programming languages such as Perl (which might be less efficient, however).

5.2. TAGGERS

As illustrated in Figure 1, the support for taggers also relies on Gsearch's filter model. For each tagger that is to interface with Gsearch, a filter must be provided that translates the tagger's output format into Gsearch's Uniform Input Format.

The tagger and the associated filter have to be registered in the Gsearch setup file. A user can then query an untagged corpus by specifying the name of the tagger on the Gsearch command line (instead of the name of a corpus) and supplying the name of the corpus file to be queried. Gsearch will then run the tagger on this file, with the output piped into Gsearch's parser via the associated filter.

The present Gsearch distribution includes filters for Thorsten Brants' TnT tagger (Brants, 1998) and for Andrei Mikheev's LT POS tagger (Mikheev, 1997).[6]

5.3. VISUALIZATION TOOLS

The modular architecture exploits output filters to transform Gsearch's Uniform Output Format (UOF) into the format required for a given output module. This allows new output modules to be added straightforwardly to the existing ones (SGML, Viewtree, Thistle) by writing a new output filter.

5.4. EXTERNAL LINGUISTIC RESOURCES

Certain applications require access to linguistic information that is not directly encoded in a given corpus, but can be provided by external linguistic resources. An example is sense information for a given word, based on its entry in a lexical database such as WordNet (Miller et al., 1990).

Gsearch provides access to such external linguistic resources by treating the information they provide as data fields in a corpus. These data fields can then be referenced in Gsearch queries and grammars just like any other field that is directly marked up in the corpus.

Such external data fields have to be declared in the grammar file by specifying how to access the external linguistic resource. Gsearch will then generate a query to the resource based on an existing data field (such as `word` or `lemma`). The external linguistic queries will be optimized where possible so as to occur last in the terminal matching process.

Consider the following example:

```
gsearch susie GrammarSusie "<tag = NN.* & hypo=organism>" pp
```

This command searches the SUSANNE corpus for nouns followed by a PP, where the hyponyms of the noun have to include *organism*. Gsearch obtains the information for the `hypo` field by querying WordNet using the `lemma` field as the argument.

6. Research Conducted Using Gsearch

The Gsearch system has been used for a number of corpus-based studies in psycholinguistics (Corley and Cuthbert, 1997; Corley and Haywood, 1999; Lapata and Keller, 1998; Sturt et al., 1999a, b), computational linguistics (Lapata, 1999; Lapata and Brew, 1999; Lapata et al., 1999), and theoretical linguistics (Zamparelli, 1998). Gsearch has also been used as a teaching tool in courses on Data-Intensive Linguistics and Theoretical Linguistics at the School of Cognitive Science, University of Edinburgh.

Within psycholinguistics, many current experiments rely on the subcategorization frequency biases of particular verbs. While these can be easily extracted from current parsed corpora, such as the Penn Treebank (Marcus et al., 1993), there are often insufficient occurrences of particular verbs to obtain reliable statistics. Further, there is the problem that such corpora are relatively unbalanced – the Penn Treebank consisting in large part of text from the Wall Street Journal – and may therefore give a very skewed perspective on normal usage frequencies. To counter these issues, Sturt et al. (1999a, b) exploited Gsearch to extract subcategorization frequencies from the BNC, a 100M word corpus with much more balanced content. In particular, Sturt et al. were concerned with the local ambiguity which occurs with verbs that can be followed by either an NP or bare S complement, such as in:

(1) a. The athlete realized $[_{\mathrm{NP}}$ his goals].
b. The athlete realized $[_{\mathrm{S}}$ $[_{\mathrm{NP}}$ his goals] were out of reach].

A noun phrase grammar was written which allowed Sturt et al. to find occurrences of the relevant verb followed by an NP (which could also be the subject of an embedded sentence). A random sample of 100 instances were then classified by hand as NP, bare S, or other. These biases were then used to construct experimental materials in which verb frequency biases were a crucially controlled factor.

An example of a more computational application of Gsearch is the study by Lapata (1999) who examined verb subcategorization alternations such as the dative and the benefactive alternation (Levin, 1993). As examples consider (2) and (3):

(2) a. John offers shares to his employees.
b. John offers his employees shares.
(3) a. Leave a note for her.
b. Leave her a note.

Relevant subcat frames were extracted from the BNC using Gsearch and postprocessed using linguistic heuristics, statistical filtering techniques, and taxonomic knowledge. Lapata (1999) showed that this process reliably acquires verb alternations, and that the subcat frequencies obtained can be used to estimate the productivity of an alternation and the typicality of its members.

7. Availability

Gsearch is implemented in C and runs under Solaris. A Linux port is currently in preparation. Gsearch is freely available for non-commercial use. Download information can be found at `http://www.hcrc.ed.ac.uk/gsearch/`. There is also a mailing list for Gsearch users, which carries announcements relating to Gsearch development. To subscribe, please send an e-mail containing the word ‘subscribe’ in the message body to `gsearch-request@cogsci.ed.ac.uk`.

Acknowledgements

Work on Gsearch was funded in part by the UK Economic and Social Research Council, and is currently funded in a collaborative ARC project between the University of Edinburgh and Saarland University by the British Council and the Deutscher Akademischer Austauschdienst. We are grateful to the following people for suggestions and advice regarding Gsearch: Jo Calder, Claire Grover, Ewan Klein, Maria Lapata, Patrick Sturt, Roberto Zamparelli.

Notes

[1] For example, SUSANNE (Sampson, 1995); Penn Treebank (Marcus et al., 1993); NEGRA (Skut et al., 1997); Prague Dependency Treebank (Hajič, 1998); International Corpus of English (Greenbaum, 1996).

[2] The SUSANNE corpus includes detailed syntactic markup (Sampson, 1995), although this is not made available to Gsearch via the standard (supplied) filter.

[3] This corpus is part of the European Corpus Initiative Multilingual Corpus I (ECI/MCI) distributed by the Association for Computational Linguistics.

[4] Viewtree can also be obtained from `http://www.cis.upenn.edu/˜adwait/`.

[5] Thistle is not included in the Gsearch distribution. It can be obtained from the Language Technology Group at the University of Edinburgh. For details, please refer to `http://www.ltg.ed.ac.uk/software/thistle/`.

[6] The taggers are not included in the Gsearch distribution. TnT can be obtained from the Department of Computational Linguistics at Saarland University, `http://www.coli.uni-sb.de/˜thorsten/tnt/`. LT POS is available from the Language Technology Group at the University of Edinburgh, `http://www.ltg.ed.ac.uk/software/pos/`.

References

Aho, A. V. and J. D. Ullman. *The Theory of Parsing, Tanslation and Compiling*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Berber Sardinha, A. P. “Review: WordSmith Tools”. *Computers & Texts* 12 (1996), 19–21.

Brants, T. “TnT: A Statistical Part-of-Speech Tagger. Installation and User’s Guide”. Department of Computational Linguistics, Saarland University, 1998.

Burnard, L. “Users Guide for the British National Corpus”. British National Corpus Consortium, Oxford University Computing Service, 1995.

Calder, J. “How to Build a (Quite General) Linguistic Diagram Editor”. In: P. Olivier (ed.): *Proceedings of the 2nd Workshop on Thinking with Diagrams*. Aberystwyth, 1998a, pp. 71–78.

Calder, J. “Thistle: Diagram Display Engines and Editors”. Technical Report HCRC/TR-97, Human Communication Research Centre, University of Edinburgh, 1998b.

Christ, O. “A Modular and Flexible Architecture for an Integrated Corpus Query System” In: *Proceedings of the 3rd Conference on Computational Lexicography and Text Research*. Budapest, 1994, pp. 23–32.

Corley, M., S. Corley, M. Crocker, F. Keller and S. Trewin. “Gsearch User Manual”. Human Communication Research Centre, University of Edinburgh, 1999.

Corley, M. and M. Cuthbert. “Indiviual Differences in Modifier Attachments: Experience-Based Factors”. Paper presented at the 3rd Conference on Architectures and Mechanisms for Language Processing, Edinburgh, 1997.

Corley, M. and S. Haywood. “Parsing Modifiers: The Case of Bare-NP Adverbs”. In: *Proceedings of the 21st Annual Conference of the Cognitive Science Society*. Vancouver, 1999, pp. 126–131.

Earley, J. “An Efficient Context-Free Parsing Algorithm”. *Communications of the ACM* 13(2) (1970), 94–102.

Francis, W. N., H. Kučera and A. W. Mackie. *Frequency Analysis of English Usage: Lexicon and Grammar*. Boston: Houghton Mifflin, 1982.

Greenbaum, S. (ed.) *Comparing English Worldwide: The International Corpus of English*. Oxford: Clarendon Press, 1996.

Hajič, J. “Building a Syntactically Annotated Corpus: The Prague Dependency Treebank”. In: E. Hajičová (ed.): *Issues of Valency and Meaning: Studies in Honor of Jarmila Panevová*. Prague: Karolinum, 1998, pp. 106–132.

Lapata, M. “Acquiring Lexical Generalizations from Corpora: A Case Study for Diathesis Alternations”. In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, MA, 1999, pp. 397–404.

Lapata, M. and C. Brew. “Using Subcategorization to Resolve Verb Class Ambiguity”. In: *Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. MA: College Park, 1999, pp. 266–274.

Lapata, M. and F. Keller. “Corpus Frequency as a Predictor of Verb Bias”. Poster presented at the 4th Conference on Architectures and Mechanisms for Language Processing, Freiburg, 1998.

Lapata, M., S. McDonald and F. Keller. “Determinants of Adjective-Noun Plausibility”. In: *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*. Bergen, 1999, pp. 30–36.

Levin, B. *English Verb Classes and Alternations: A Preliminary Investigation.* Chicago: University of Chicago Press, 1993.

Marcus, M. P., B. Santorini and M. A. Marcinkiewicz. “Building a Large Annotated Corpus of English: The Penn Treebank”. *Computational Linguistics* 19(2) (1993), 313–330.

Mikheev, A. “Automatic Rule Induction for Unknown Word Guessing”. *Computational Linguistics* 23(3) (1997), 405–423.

Miller, G. A., R. Beckwith, C. Fellbaum, D. Gross and K. J. Miller “Introduction to WordNet: An On-line Lexical Database”. *International Journal of Lexicography*, 3(4) (1990), 235–244.

Sampson, G. *English for the Computer: The SUSANNE Corpus and Analytic Scheme*. Oxford: Clarendon Press. 1995.

Skut, W., B. Krenn, T. Brants and H. Uszkoreit “An Annotation Scheme for Free Word Order Languages”. In: *Proceedings of the 5th Conference on Applied Natural Language Processing*. Washington, DC, 1997, pp. 88–95.

Sturt, P., M. J. Pickering and M. W. Crocker. “Exploring the Reanalysis as a Last Resort Strategy”. Paper presented at the 12th CUNY Conference on Human Sentence Processing, New York, 1999a.

Sturt, P., M. J. Pickering and M. W. Crocker. “Structural Change and Reanalysis Difficulty in Language Comprehension”. *Journal of Memory and Language* 40(1) (1999b), 136–150.

Zamparelli, R. “A Theory of Kinds, Partitives and *of/z* Possessives”. In: A. Alexiadou and C. Wilder (eds.): *Possessors, Predicates and Movement in the Determiner Phrase*. Amsterdam: John Benjamins, 1998, pp. 259–301.