University of Cambridge

MPhil Advanced Computer Science Project

Applying Language Models to Language Learning

Author:
Duncan Roberts
St Edmund's College
dr369@cam.ac.uk

Supervisors:
Dr Paula Buttery
pjb48@cam.ac.uk

Dr Andy Rice
acr31@cam.ac.uk

16th June 2011

I Duncan Roberts of St Edmund's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

Word count: 14,975. This excludes appendix B, in line with the guidelines.

Abstract

Language learners have access to large-scale resources for learning vocabulary that are amenable to machine processing. Similarly comprehensive and machine-readable resources for learning grammar are less evident, and producing such content requires a considerable time investment from experts. Such resources are desirable as repetition is central to learning. LearnGrammar! *addresses this by generalising from a handful of examples and counter-examples provided by an expert, to produce larger lists of examples that learners can use for studying.*

Acknowledgements

Thanks to my two supervisors, Dr Paula Buttery and Dr Andy Rice for their guidance and suggestions, and Dr Andrew Caines for evaluating the application.

Contents

1 Introduction

Proven strategies for second language acquisition are varied [28], but are invariably resource-intensive, involving classes, textbooks, films, music, conversations with native speakers, and so forth. To an extent, the Internet has democratised learning by providing a wealth of free resources: Wikipedia is the most famous, but sites like about.com offer online lessons for language learners, and busuu.com aims to connect people looking to learn each others' languages. Some of these resources are amenable to machine processing, and have in turn been exploited to provide new learning tools: *Learn!*[1], an Android smartphone application, extracts vocabulary and associated images from Wiktionary to produce vast collections of vocabulary-oriented flash cards for many languages. This represents a considerable labour saving relative to the manual preparation of such datasets.

What comparable resources do we have for teaching grammar? Getting exposure to a high concentration of examples of a particular grammatical point — relative clauses, the present continuous, prepositional phrases — is more difficult. Endless repetition is crucial [29], and so we want a deep and varied pool of examples of grammatical learning points, in a standard format, amenable to processing by machines. But if we can search the Web for 'present continuous examples English', what we'll find will be a jumble of webpages of varying degrees of relevance and quality. Short of widespread adoption of the Semantic Web[2], building applications on top of such heterogeneous data would be challenging: each page in our search results is likely to present examples of the present continuous in different formats. Grammar textbooks such as *English Grammar Today* [8] give us different problems, typically giving us a handful of examples per subject. These examples are chosen by professional linguists: how can we democratise access to a large volume of examples of arbitrary grammatical structures without giving every learner access to such a linguist?

Addressing this concern, I advance the following **research hypothesis**:

> Given 3–5 sentences that illustrate an arbitrary, unseen grammatical point (examples), and a similar number that do not (counter-examples), we can identify the grammatical point using language modelling tools and resources, and then generalise to produce a longer list of example and counter-example sentences.

We say that the grammatical point is *unseen* because the application should be agnostic as to the type of grammatical learning point being demonstrated. Generalising from a small set of examples to a larger set forms the core of this research; I demonstrate usage of these datasets by implementing a smartphone application that presents examples and counter-examples of sentences to students in a flash card format.

2 Requirements

The goal of the project is to produce an application satisfying the following requirements:

[1] `http://www.cl.cam.ac.uk/research/dtg/language/`
[2] `http://semanticweb.org/`

1. *(Essential) Teacher and student modes.* There are to be two modes: one for *teachers*, who define learning tasks; and one for *students*, who practice language points using the content provided by the teachers.
2. *(Essential) Generalisation from minimal input.* In the teacher mode, it should be sufficient for the teacher to provide around ten sentences. The application should generalise, producing many other sentences that illustrate the same point.
3. *(Essential) Presentation of correct and incorrect answers to student.* In the student mode, the student should be presented with a few sentences, and a title indicating the nature of the grammatical structure being taught, e.g. "future tense" or "relative clause". One of the sentences should exemplify the grammatical structure in question (i.e. the correct answer); the other sentences should not (i.e. incorrect answers). The learning process consists of the student trying to pick out the correct sentence, getting feedback as to whether the correct sentence was selected, and then being presented with a fresh set of sentences.
4. *(Essential) Variety of learning tasks.* The teaching of a wide variety of grammatical points should be supported.
5. *(Essential) Definition of new learning tasks without code changes.* Learning tasks not considered during development should be supported without changes to the code.
6. *(Essential) Application should run on a smartphone.*
7. *(Desirable) Volume of output.* The application should correctly identify at least 50 sentences that illustrate the learning task in most cases, and at least 150 that do not.
8. *(Desirable) Superficial resemblance of counter-example suggestions to learning point.* The counter-example sentences provided to the student should have a superficial resemblance to the grammatical structure being taught. The exact nature of these sentences, however, is unspecified: they could be ungrammatical approximations of the correct structure being taught, or just grammatical sentences that do not demonstrate the point in question.
9. *(Desirable) Simplicity of suggested sentences.* Selected sentences (both examples and counter-examples) should be as simple as possible, given the point being demonstrated.
10. *(Desirable) Operating system extension and configuration changes should be unnecessary.* The application should operate without configuration changes or extensions to the operating system.
11. *(Desirable) Responsiveness.* The user should not be made to wait for an excessive period of time while waiting for loads, network traffic or processing.
12. *(Desirable) Network access non-dependence.* Both teachers and students should be able to use the program without network access.

3 Approach

There are, broadly, two ways we investigate to select the sentences presented to students:

a. Modify the source sentences in some way that preserves syntactic and semantic coherence, or
b. Identify what the source sentences have in common, and then pick sentences from a corpus that exemplify these commonalities.

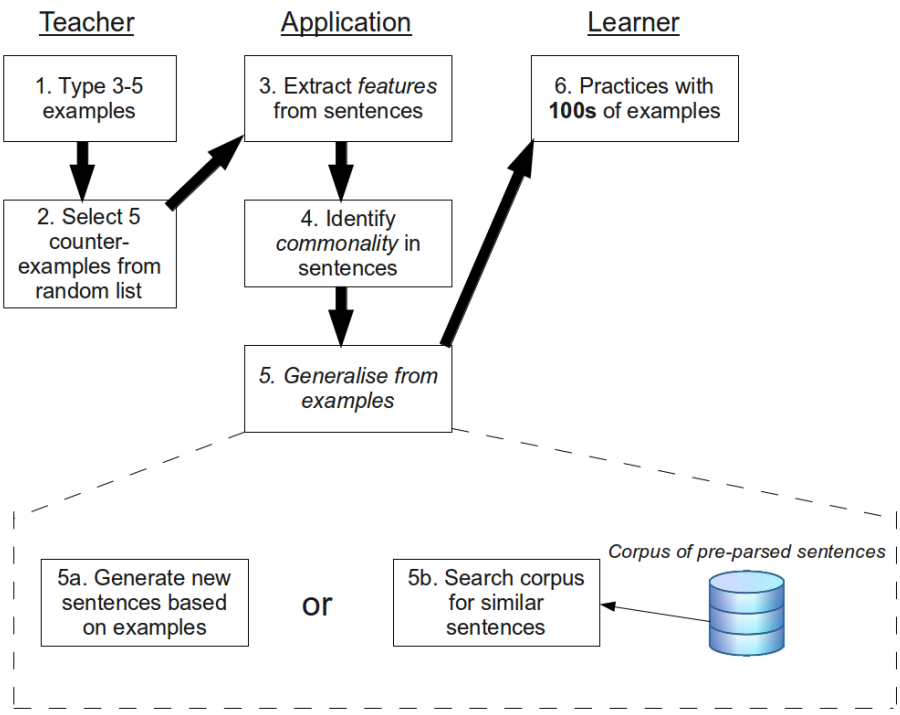

Figure 1: Two approaches to finding sentences that illustrate a learning point provided by a teacher.

See fig. 1 for a visual representation of these two approaches. At face value, we would expect different things from these. It is perhaps conceptually easier to create new sentences using the first approach. When swapping words and constituents around on the basis of part-of-speech (POS) tags or GRs, maintaining fidelity to the original is not too difficult. The first major challenge here is maintaining syntactic and especially semantic coherence. The second is that we need a suite of different approaches to modifying sentences, to layer on top of each other, to get sentences that are not trivial variations of the originals. The second approach gives us the inverse: by pulling real sentences from a corpus, syntactic and semantic coherence should be non-issues, but establishing which sentences embody the features exemplified by the source sentences is perhaps more involved. An additional concern is that we inherit limitations of the corpus. A newspaper text corpus like CCGbank [18] is likely to be heavy on reported speech, the passive voice and the past tense. If our learning task does not reflect these biases, the results may suffer. Newspaper sentences are also lengthy and complex: the PTB WSJ corpus has average sentence length of 16 words, and a Flesch Reading Ease [15] score of 61.2/100[3], which falls into the 60–70% 'standard difficulty' band defined by Flesch [15]. Many such sentences are thus likely to be difficult for non-native speakers. Finally, corpora are big, and hence processing is slow: while we're likely to have a good number of examples of a given syntactic construction in a large corpus, we might have to search through many thousands of examples to find one — and how do we decide when a sentence is 'good enough'? Do we search through every sentence available? Phone memory is also a concern.

Both of these approaches have been explored in this project: generation in §6.2; selection from a corpus in §6.3.

4 Background

4.1 Maximum Entropy Models

In two respects, this project builds upon the emergence of accurate statistical syntactic parsers as pioneered by Charniak [9] and Collins [11]. First, the project represents an application for the wealth of syntactic information provided by such parsers — in our case, Clark and Curran's CCG [37] parser, C&C [10]. Secondly, many aspects of the implementation draw inspiration from established strategies in statistical parsers, and also part-of-speech taggers.

A flexible approach to classifying items such as words is the *maximum entropy model* (henceforth 'maxent'). A variant of maxent for classifying sequences rather than individual items, the maximum entropy Markov model, is widely used in both POS tagging [32] and syntactic parsing [33]. A maxent model defines the probability of some item x belonging to category c as follows:

$$p(c|x) = \frac{1}{Z} \exp \sum_i w_i f_i(c, x) \tag{1}$$

[3]These statistics were derived using Linux command `style`.

Where Z is a normalisation constant, w is a set of weights, and f is a set of *feature functions*, each of which returns a positive value if some simple test suggests that x belongs in c. To take the POS tagging example, each c will be a particular POS tag, e.g. `NN` for common nouns and `JJ` for adjectives in the PTB tagset. Here is a simple feature function for tagging (lifted from Martin and Jurafsky [20]) that returns 1 for *race* tagged as a noun:

$$f_i(c, x) = \begin{cases} 1 & \text{if } x = race \wedge c = NN \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The principle is similar for statistical parsing, though in C&C's case, the features return counts rather than just 0 or 1. A parser's features operate on *rule instantiations* as partial syntax trees are built up. Here is a feature from the C&C parser (adapted from Clark and Curran [10]) that returns an observed occurrence count if we're combining the verb *bought* with argument *company*:

$$f_i(w_1, r, w_2) = \begin{cases} c(w_1, r, w_2) & \text{if } w_1 = company \wedge r = S[dcl] \rightarrow NP\ S[dcl] \backslash NP \wedge w_2 = bought \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where $S[dcl] \rightarrow NP\ S[dcl] \backslash NP$ is a rule whereby the operands on the right hand side (a noun phrase (NP) and a verb phrase ($S[dcl] \backslash NP$)) are combined to produce a declarative sentence ($S[dcl]$), r is the rule actually invoked, w_1 is the head word of the rule's first operand, w_2 is the second operand's head word, and $c(w_1, r, w_2)$ gives the number of times this rule-head words combination was observed while training the model against a gold standard of correctly-parsed sentences.

Maxent involves a training process whereby weights are assigned to each feature, in an attempt to find a configuration that maximises the probability of the training data. As there are a vast number of features (hundreds of thousands in C&C [10]) and training corpora are typically vast (46,532 parses with CCGbank), an exhaustive search is intractable and the absolute maximum may not be found. Maximum entropy is so called because the trained model (=set of feature weights) chosen is the one with the most even weightings of those that maximise the probability of the training data; such a model embodies as few assumptions as possible beyond what is justified by the data.

Our task is to take a small handful of example sentences, and transform it into several hundred sentences, each of which demonstrates the same grammatical point that the examples illustrate. While it must be stressed that the present implementation does not involve a maximum entropy model[4]), there are several parallels (see table 1). While we only have around ten examples, this set acts as our training data for feature selection purposes. While we can extract features from this data as in a maxent tagger or parser, we can't use these sentences to optimise our weights as we have no gold standard available for each grammatical point. Taggers and parsers typically prune features if they occur too infrequently in the training data; we also prune features that aren't *common* to the example sentences (*commonality* is defined in §6.1). Two different approaches are explored for furnishing the set of candidate sentences from which we select ones which exhibit the features we're looking for: creating

[4]In principle, a full maxent implementation appears possible (see §8.1.

Area	POS tagging	Parsing	Generalising
Feature selection	Corpus	Corpus	Example sentences
Feature weighting	Corpus	Corpus	Hardcoded
Feature pruning	Min. occurs	Min. occurs	Commonality-based
Search space	$\#\text{words}^{\#\text{tags}}$	$\#\text{words}^{\#\text{rules}}$	#corpus sentences
Subject of evaluation	Untagged sentence	Unparsed sentence	Corpus sentence
Unit	Word	Rule instantiation	Word
Composition of units	Dynamic programming	CYK	N/A

Table 1: Comparison of two applications of the maximum entropy model with the approach to generalising from a small set of examples presented in this dissertation, with particular reference to selection of sentences from a corpus. The parser described here is C&C; the POS tagger, however, is not C&C's, as C&C's tagger uses an HMM model rather than maxent [10]. The 'POS tagging' column above is representative of other POS taggers such as the Stanford POS Tagger [40] that use maxent. *Feature selection* refers to the process identifying features to evaluate candidates against; POS taggers and parsers are typically trained against tens of thousands of pre-tagged/parsed sentences such as those of the Penn Treebank (PTB) [26] or CCGbank [18], whereas the 'generalisation' task finds features in around ten teacher-provided sentences. These taggers and parsers use maxent to optimise feature weights against a corpus; at present our weights are hard-coded in the generalisation algorithm. Taggers and parsers often prune features not observed with some minimum frequency during training; the generalisation task prunes features that fall outside of some notion of commonality. In tagging, our search space is defined by the number of words in the sentence to be tagged and available POS tags; in parsing, it's the number of words and combinatorial rules; in generalisation, it's much smaller: the set of corpus sentences. Taggers and parsers use dynamic programming and chart parsing respectively to manage the combinatorial possibilities of tag sequences or parses efficiently. We're just selecting sentences rather than constructing them, so this is unnecessary.

new sentences by manipulating the initial handful; and searching through a corpus. Finally, in parsing and tagging, decisions between tags/rules are not modelled independently: we calculate probabilities over complete tag sequences and parse trees. As a result, the parsing search space in a wide-coverage grammar such as CCG is vast for all but the most trivial of sentences [10]. In our case, we're assigning scores to independent corpus sentences, so there is no combinatorial explosion in the search space.

4.2 Statistical Parsing with C&C

Before we can parse a sentence, we must *tokenise* it: punctuation must be separated from adjacent words, and contractions like *don't* must be split into their component words. For instance, *I didn't take Becky's cat, honest.* becomes '*I did n't take Becky 's cat , honest .*'. After detokenisation, we refer to the units — punctuation or words — as *tokens*. Next, we pass sentences to our parser, C&C [10]. C&C infers a wealth of syntactic information about an input sentence:

1. *Part-of-speech (POS) tags*: the syntactic function of an individual token: typically tal-

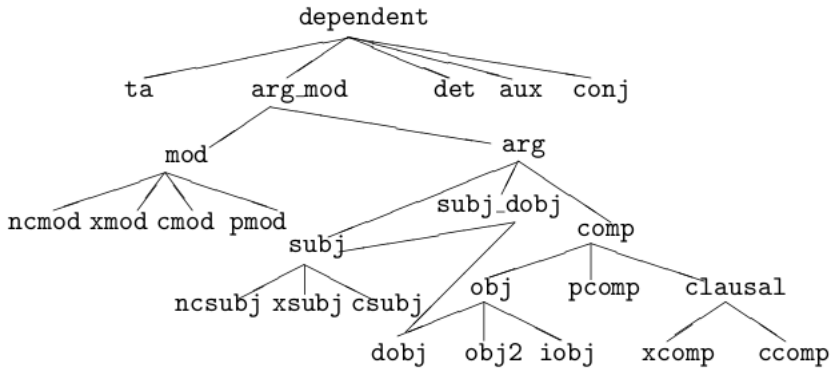

Figure 2: The grammatical relation type hierarchy. Lifted from a RASP technical report [4].

lies closely to common notions of syntax, with categories like past tense verbs, common nouns, adjectives and so forth. C&C uses a minor extension of the Penn-Treebank POS tag specification [26]. I further extend this specification; the resultant tagset of 71 tags is documented in appendix A.

2. *Supertags*: like POS tags, supertags are tags assigned to each token in a sentence. However, supertags have been referred to as *almost parsing* [1], as they contain a great deal of information about relationships between tokens, such as the number and type of arguments a verb is expecting (e.g. an intransitive verb gets a different supertag from a transitive verb). Hence, many decisions have already been made before tokens are combined into a syntax tree. Because of their focus on relationships between tokens, they do not make some distinctions made by POS tags, such as proper vs common nouns, or cardinal numbers vs adjectives. C&C uses 425 supertags [10].
3. *Grammatical relations (GRs)*: a hierarchical set of 26 types of syntactic link between tokens, covering various types of subject (`subj`), object (`obj`), complement (`comp`) and so on. See fig. 2. Note that `dobj` and `subj` have two parent nodes: `obj` and `subj_dobj` in the former case, `arg` and `subj_dobj` in the latter. As such, this hierarchy is only a directed acyclic graph, not a tree.
4. *Lemmas*: the base forms of tokens, removing inflectional suffixes and irregularities. For instance, the lemma of *going* is *go*; for *am*, *are*, *is* and *was*, the lemma is *be*.
5. *Named entities*: largely a fine-grained proper noun classification [13], e.g. names of people (`I-PER`), organisations (`I-ORG`) and places (`I-LOC`).

We can combine GRs with the other, token-level, information into a graph, as shown in fig. 3. As GRs link words together, the intuitive rendering of dependency structures into graphs would have GRs as edges and tokens as nodes. However, GRs can actually link together three tokens, through *head*, *dependent* and *subtype* relationships. For instance, the `xcomp` GR found in *I'm going to eat some apples* has *going* as its head, *eat* as its dependent, and *to* as its subtype. As such, our dependency structure graphs have both tokens and GRs as nodes, and the type of relationship (head/dependent/subtype) as edge labels. Edges always connect one GR and one token.

It's worth noting that these dependency structures are often trees, but not always. They can be cyclic: Briscoe [4] gives the example of *the hidden chest*, where a correct parse will have an

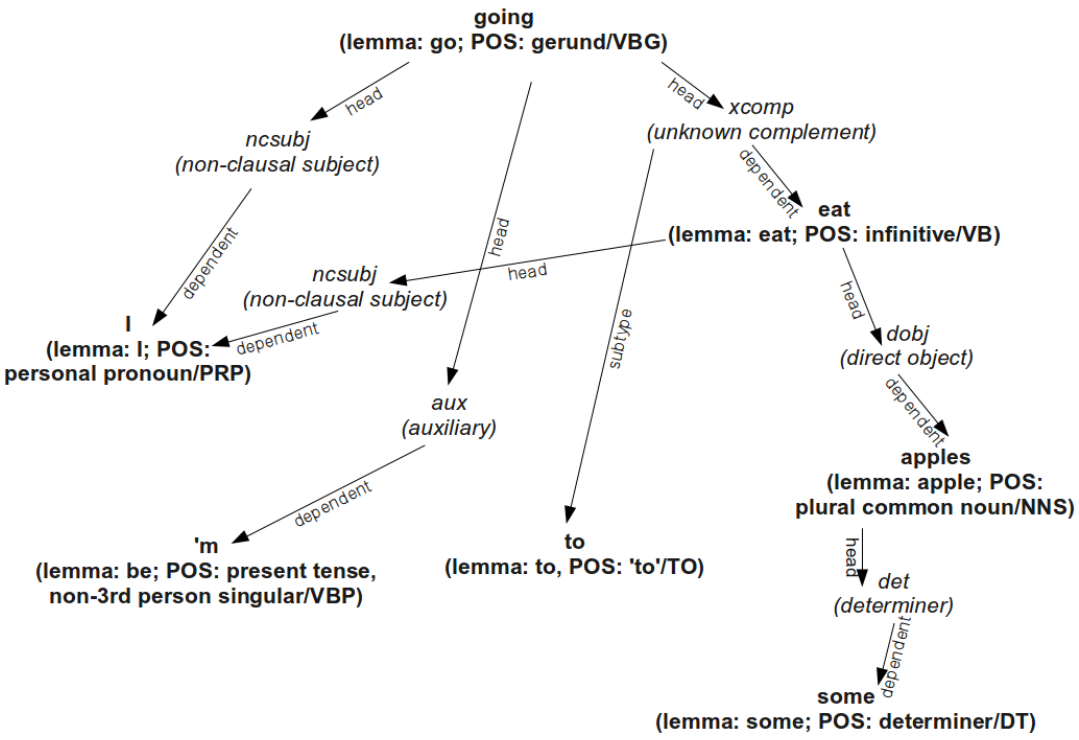

Figure 3: Dependency structure for *I'm going to eat some apples.* Supertags and named entities omitted for readability.

ncmod headed by *chest* and with *hidden* as the dependent, and a ncsubj with *hidden* as the head and *chest* as the dependent. Word nodes can have more than one parent: at least with C&C, in *I'm going to eat some apples* (as seen in fig. 3), *I* is the dependent of two ncsubj GRs: one headed by *going*, the other by *eat*.

4.3 Language Models & Language Learning on Smartphones

The main application of language models on mobiles so far has been for predictive text. Initially, this was a simple dictionary lookup with each word having a 'flat' probability without respect to parts of speech or other context such as previous words. SwiftKey[5] builds on this by modelling sentences with n-grams to predict sequences of words based on input so far. Such applications require large, general-purpose training data sets (or at most, specific to the text messaging domain).

Recent work within the *Computing for the Future of the Planet* research group [19] at the University of Cambridge has looked at mobile phones as a computing platform in Africa [34], where they are more pervasive than televisions, desktop computers and radios combined, and suggests that language-learning tools for mobiles could drive social mobility. Learning tools for smartphones in particular are widely available: a search on the Android Market for keywords like 'learn' gives many results. Applications such as *Learn!* for Android support user-contributed flashcard sets, but, beyond teaching vocabulary, developing large-scale language learning facilities is laborious (see §1). There are obvious parallels with the 'One Laptop Per Child' project, which aims to enrich the education of children in developing countries through distribution of cheap, robust, low-power laptops[6]. The potential of the smartphone as a learning tool for Africa motivates requirements 9 and 12 (see §2): it should run on a smartphone, and without network access, as network access may be prohibitively expensive.

4.4 Set Manipulation for Language Learning

The goal of the application is to teach grammar. But what aspect of grammar? If the teacher enters five sentences that exemplify the passive voice, we need to identify at some level that the passive voice is what they're trying to teach. The obvious (and boring!) option is to have the teacher select from a pre-defined list of learning tasks: future tense, present continuous tense, active voice, passive voice, relative clauses, etc. This, however, utterly undermines the centrality of user-submitted content. Teachers would only be able to provide minor variants of a fixed, centrally-maintained list of options. Such a list also has maintenance implications: each list entry would need to backed by centrally-maintained examples, and users would have to submit requests for new learning tasks. A more satisfactory approach is to automatically infer commonality from the teachers' examples. We can then attempt to find other sentences that share these common features. Such 'learning' of linguistic information by looking for commonality between multiple sentences has precedents. Siskind observes that:

[5] http://www.swiftkey.net
[6] http://laptop.org

> One way that a learner might determine the meaning of a word is to find something in common across all observed uses of that word. Commonality across observed uses can be elucidated by forming a set of possible meanings for each use, from the non-linguistic context, and intersecting those sets. [35]

Siskind was discussing how children map words to concepts, but the principle is applicable to our task. However, as we have counter-examples as well as examples, we see in §6.1 that a few other set operations are worthwhile for deriving commonality. The use of counter-examples in calculating commonality is noteworthy; there is much debate in linguistics as to whether children use such 'negative evidence' when learning languages, but seemingly consensus that such negative evidence, if made available, would aid learning [31].

4.5 Lexical Semantics & Verb Subcategorisation Frames

A variety of resources are available to analyse verbs in sentences. Levin [25] provides a detailed classification of verbs based on their subcategorisation frames (a description of the number and type of a verb's arguments. Henceforth: SCFs); word sense disambiguation algorithms such as Simplified Lesk [21] attempt to identify a word's sense from its context; WordNet [27] provides semantic relationships between verbs as well as a more approximate account of verb SCFs; and syntactic parsers give us a verb's relationships to other words in a sentence [10], along with information such as the tense of the verb. Relationships between some of these resources have been established. A mapping [24] is available between the sense of a verb and its SCF, though this is inherently imperfect [14]. These resources, and the relationships between them, have relevance to this project especially when considering sentence generation. However, even sentence selection, there is value in checking whether the example sentences use verbs with similar features.

One particularly interesting resource is WordNet. WordNet has APIs available for many programming languages. One can traverse a large tree of synonyms, hypernyms and hyponyms of a given *synset* (synonym set); and retrieve a list of known senses of a given lexical item, in order of frequency of occurrence — e.g. for *reason*, the sense of *the reason that war was declared* comes before the *man is endowed with reason* sense. We see in §6.2.1 how this can be used in sentence generation.

The approach taken here is to map from the grammatical relations surrounding a verb to a WordNet SCF. This has been done before [5] with a different set of target SCF (a 160-frame superset of those in the COMLEX Syntax [17] and ANLT [3] dictionaries), but the crucial point for our purposes is the WordNet integration. As such, I have investigated and implemented a mapping from GR graphs to WordNet SCFs. Given WordNet's broad usage, this may find applications beyond this project. Preiss *et al* worked with SCF definitions for adjectives and nouns [30], but as WordNet doesn't define such SCFs, no such mapping is investigated here. Korhonen and Sun investigated using semantic features for verb classification [38] (a task that subsumes that of assigning SCFs).

5 Tooling & Resources Used

Software and resources used by the project:

- The C&C parser [10], a statistical syntactic parser using the combinatory categorial grammar (CCG) formalism.
- CCGbank [18], a 'gold standard' reference set of CCG parses for 46,532 English sentences lifted from the Wall Street Journal section of the PTB [26].
- WordNet [27], a lexical database of English open-class words. Provides word senses, and synonymy/hypernymy/hyponymy relationships.
- JWI[7] (Java WordNet Interface), a WordNet API for Java.
- SimpleNLG [16], a text generation tool. Here used simply to 'de-lemmatise' words (e.g. *have* marked as past tense → *had*).
- Google Android SDK[8], the APIs for developing applications for Android handsets.
- Google Guava[9], a general-purpose Java library. Used here primarily for its Java Collections Framework[10] extensions.
- Apache Commons Lang[11], another general-purpose Java library. Used solely for escaping strings written to disk in XML format.
- JUnit[12], a unit testing library for Java.
- A Google Nexus One mobile phone.

While the project builds on ideas from *Learn!*, this application was not itself (nor any of its code) used by the project.

Many of these software choices were inconsequential, with many solid alternatives providing much the same functionality (JWI, SimpleNLG, Guava, Commons Lang). Going with a Nexus One, and hence Android, was a simple question of availability. A partial alternative to WordNet exists in Levin classes [25], which provide a detailed classification of verbs according to their SCFs, but this corresponds only weakly to synonymy [14]. Our interest in lexical databases relates to word senses and relationships such as synonymy, hypernymy and hyponymy, making WordNet more appropriate. While I do use SCFs (see §6.2.2), I use them mostly as a stepping stone to semantic information, and so the limitations of WordNet's more coarse-grained SCFs are acceptable.

Among others, Briscoe and Carroll's RASP [6] (Robust Accurate Statistical Parsing) represents a solid alternative to C&C. Both produce grammatical relations which largely adhere to a common specification [4]. C&C's authors, Clark and Curran, benchmark the precision and recall of C&C and RASP against DepBank [22] (a set of 700 gold standard parses from the PTB, expressed as sets of grammatical relations). Their key finding [10] was an overall F_1 score of 76.29% for RASP and 81.14% for C&C. These figures represent the accuracy of individually-generated GRs; as many sentences have ten or more GRs, this accuracy difference becomes considerable at the sentence level. Two other factors that favour C&C are

[7] `http://projects.csail.mit.edu/jwi/`
[8] `http://developer.android.com/sdk/`
[9] `http://guava-libraries.googlecode.com/`
[10] `http://download.oracle.com/javase/6/docs/technotes/guides/collections/`
[11] `http://commons.apache.org/lang/`
[12] `http://www.junit.org/`

that it has a webservice interface, making it easy to get sentences parsed in real-time from a mobile device, and that it ties in well with CCGbank. Parsed sentences in CCGbank cannot be easily compared with RASP output because RASP uses the CLAWS2 tagset[13], whereas CCGbank (and C&C) use the PTB tagset[26]. Many other widely-used parsers exist, such as the Stanford parser [23]. Stanford's parser is written in Java, so resource constraints aside, it should in principle run on Android. Stanford uses the PTB for training and testing rather than CCGbank, but as CCGbank is derived from the PTB this difference is relatively superficial and relates to the underlying grammar formalisms of the parsers. It would appear to be another good candidate, though direct comparisons between C&C and the Stanford parser are difficult: not only do they use different grammar formalisms, they also have entirely different grammatical relation formats.

6 Implementation

In the teacher mode, there are two major tasks: identifying what our examples have in common grammatically, and then generalising to a large number of sentences that exhibit the grammatical features found across the teacher-provided examples. We look first to how we identify commonality between our example sentences, and then look at two solutions for generalising.

6.1 Finding Commonality

6.1.1 Basic Principles

For each example sentence S_i, we identify the set of *features* that it demonstrates. The basic types of feature currently supported are:

1. Lemmas
2. POS tags
3. Supertags
4. GR types

We gather these details for every token and GR in each sentence. Say our teacher wants to teach copula verbs, and offers the sentence *it's hard*. The features we observe for this sentence are:

- Lemmas: `[it, be, hard]`
- POS tags[14]: `[PRP, VBZ, JJ]`
- Supertags[15]: `[(S[dcl]\NP)/(S[adj]\NP), S[adj]\NP]`
- GR types: `[ncsubj, xcomp]`

[13] `http://ucrel.lancs.ac.uk/claws2tags.html`

[14] We remind the reader that a POS tag reference is found in appendix A.

[15] For our purposes, understanding supertags beyond the general description offered in §4.2 is unnecessary. There is no need to interpret individual supertags.

This, in isolation, is not very useful information: we might guess from the above that the learning task relates to personal pronouns or use of the verb *be*. That is, we want to separate the features pertaining to the intended learning task (`xcomp`, `JJ`) out from the noise (*it*, *be*, *hard*, `PRP`, ...). A step towards filtering out the noise is to ask for several sentences. We might add these other examples of the copula: *you look funny*, and *she seemed rather scared*. We now calculate the *intersection* of our sentences' features. We call this *weak commonality*, or $\hat{M}_w$:

$$\hat{M}_w = \bigcap_{i=0}^{N} f(S_i) \tag{4}$$

Where f returns the set of features for a given sentence. Our weak commonality looks like this:

- Lemmas: `[]`
- POS tags: `[PRP, JJ]`
- Supertags: `[(S[dcl]\NP)/(S[adj]\NP), S[adj]\NP]`
- GR types: `[ncsubj, xcomp]`

This is an improvement, but it's only sufficient if the teacher either puts in a relatively large set of sentences, or is aware of all the feature types and thinks hard about how to make the sentences varied enough to eliminate unintended commonality — this places an undesirable burden on the teacher. For instance, the commonality of the current set of example sentences includes `PRP` (personal pronoun), when this is not an essential component of copula verb usage: *the soup smells tasty* has no `PRP` yet uses a copula verb.

A simple technique for relieving this burden on the teacher is to ask for a set of counter-examples sentences, C. We then arrive at our set of common features by taking the intersection of features in S and subtracting the union of features in C. This is *strong commonality*, or $\hat{M}_s$:

$$\hat{M}_s = \bigcap_{i=0}^{N} f(S_i) \setminus \bigcup_{j=0}^{M} f(C_j) \tag{5}$$

In the case of the copula, a single counter-example like *she went to the park* would give us the following strong commonality:

- Lemmas: `[]`
- POS tags: `[JJ]`
- Supertags: `[(S[dcl]\NP)/(S[adj]\NP), S[adj]\NP]`
- GR types: `[ncsubj, xcomp]`

Here, strong commonality is identical here to weak, except for the absence of `PRP`.

Finally, on occasion GR types or POS tags etc are conspicuously absent from our example sentences. For instance, what distinguishes sentences illustrating the second person imperative is that none of them will have a `subj` GR (i.e. a subject). For instance, the GR types for *think about it* are `iobj` and `dobj`; those of the declarative sentence *I think about it* also

include ncsubj. The *absence of* this ncsubj is thus a noteworthy aspect of a second person imperative.

To demonstrate this with the second person imperative, calculating the *union* of features in *think about it*, *go away* and *dream a little dream of me* gives us:

- Lemmas: [think, about, it, go, away, dream, a, little, of, me]
- POS tags: [VB, DT, JJ, PRP, IN, RB]
- Supertags: [(S[b]\NP)/NP, NP, (S\NP)\(S\NP), S[b]\NP, NP[nb]/N, N/N, (NP\NP)/NP, PP/NP]
- GR types: [dobj, iobj]

And now, the *intersection* of the features of some counter-examples: *I think about it* and *the profits are lower*.

- Lemmas: []
- POS tags: [VBP]
- Supertags: []
- GR types: [ncsubj]

Subtracting the union of example features from the intersection of counter-example features gives us *absence-of commonality*, or $\hat{M}_a$. This is implemented by reversing the roles of the examples S and counter-examples C in equation 5:

$$\hat{M}_a = \bigcap_{j=0}^{M} f(C_j) \setminus \bigcup_{i=0}^{N} f(S_i) \tag{6}$$

In our example, this gives us:

- Lemmas: []
- POS tags: []
- Supertags: []
- GR types: [ncsubj]

This is our absence-of commonality: we've identified that an important aspect of our example sentences is that none of them contain ncsubj.

Conceptually, weak, strong and absence-of commonality together form $\hat{M}$: an approximation of M, the *intended commonality* corresponding to the unseen intended learning task. In practice, we keep the three commonality sets separate: as discussed in §6.3.2, we apply a commonality type multiplier to feature scores.

6.1.2 Composite Feature Types

So far, we've treated the various pieces of information about our example sentences — lemmas, GR types, POS, supertags — as isolated units. That is, when searching for example suggestions, if a candidate sentence contains the right lemmas, GR types and so on then we score it highly. There are several problems with this approach. Say we're teaching the 'going-to' future tense. This construction is used to describe events in the future when we wish to

stress current evidence, or to emphasise our decision [8]. Say we use the examples *I'm going to think about it*, *Becky's going to be rich* and *she is going to read her book*. The lemmas these examples share are *be*, *go* and *to*; shared POS tags are `VBG` (gerund or present participle), `TO` (the word *to*) and `VB` (infinitive); GR types include `aux` (auxiliary), `ncsubj` (non-clausal subject) and `xcomp` (a type of complement). It's not difficult to think of sentences that tick all these boxes without being examples of the 'going-to' future tense: *she wants me to admit I'm going to the shops because I'm a shopaholic* uses only present tense constructions. A second, related, problem with our current set of features is that they can be easily neutralised by our counter-example sentences. If one counter-example contains a non-clausal subject, the word *to* and an infinitive, and another contains an `xcomp` and a `VBG`, then we no longer have any idea of what the examples have in common.

What we really need is *composite features*: combinations of GRs and information about tokens (lemmas, POS, supertags) observed in our examples. Syntactic parsers make these links available as dependency structures (see §4.2). In fig. 4, we see two example dependency structures that illustrate the 'going-to' future tense, with common aspects in boldface. These shared parts of the graph are more granular than a set of edges and nodes: for instance, there is a node where the POS is shared (`VB`) but the lemmas are not (*eat* vs *be*). We break each example's dependency structure into a large number of *directed paths*: that is, 'straight', non-cyclic graphs where every node has a degree of 1 or 2. We generate paths with different permutations of the following:

1. *Start token*: the first node of the path.
2. *Height*: the number of nodes (GRs + tokens) covered, following dependent/subtype and head edges towards the root node.
3. *Capture configuration*: which aspects (lemmas, POS, supertags, GR types) of the nodes along the path are recorded by the path.

Collectively, we refer to the above configuration as a *feature type profile*. For *Becky's going to be rich*, we might have a 'GR type + POS' path that captures a GR with `type=xcomp` whose head is a token with `POS=VBG` and whose subtype is a token with `POS=TO`. Henceforth, we refer to these directed paths as *twigs*, because intuitively, they're a small, straight part of a tree[16].

Twigs lend themselves to a convenient notation, whereby captured aspects of tokens are written in square brackets, and GR types are written in round brackets. If a particular twig captures a POS tag, we prefix the POS tag with `|` to distinguish it from a lemma; for supertags we use `#`. If a twig only captures aspects of tokens, we omit the GR entirely; vice versa if only GR types are captured. Tokens and GRs are described from the bottom of the dependency structure upwards towards the root node, with an `s` between a token and GR when the token is the GR's subtype. Some examples of this, all using the same example sentence:

- `<|TO>s(xcomp)<|VBG>`: the example previously given, of the `xcomp` GR that links subtype and head tokens with POS tags `TO` and `VBG` respectively. Height: 3 nodes.
- `<be>(aux)<go>`: lemmas and GR types for *'s going*. Height: 3.
- `<be|VBZ>(aux)<go|VBG>`: lemmas, POS and GR types with the same token/GR

[16] Or rather, a structure that isn't quite a tree (see §4.2).

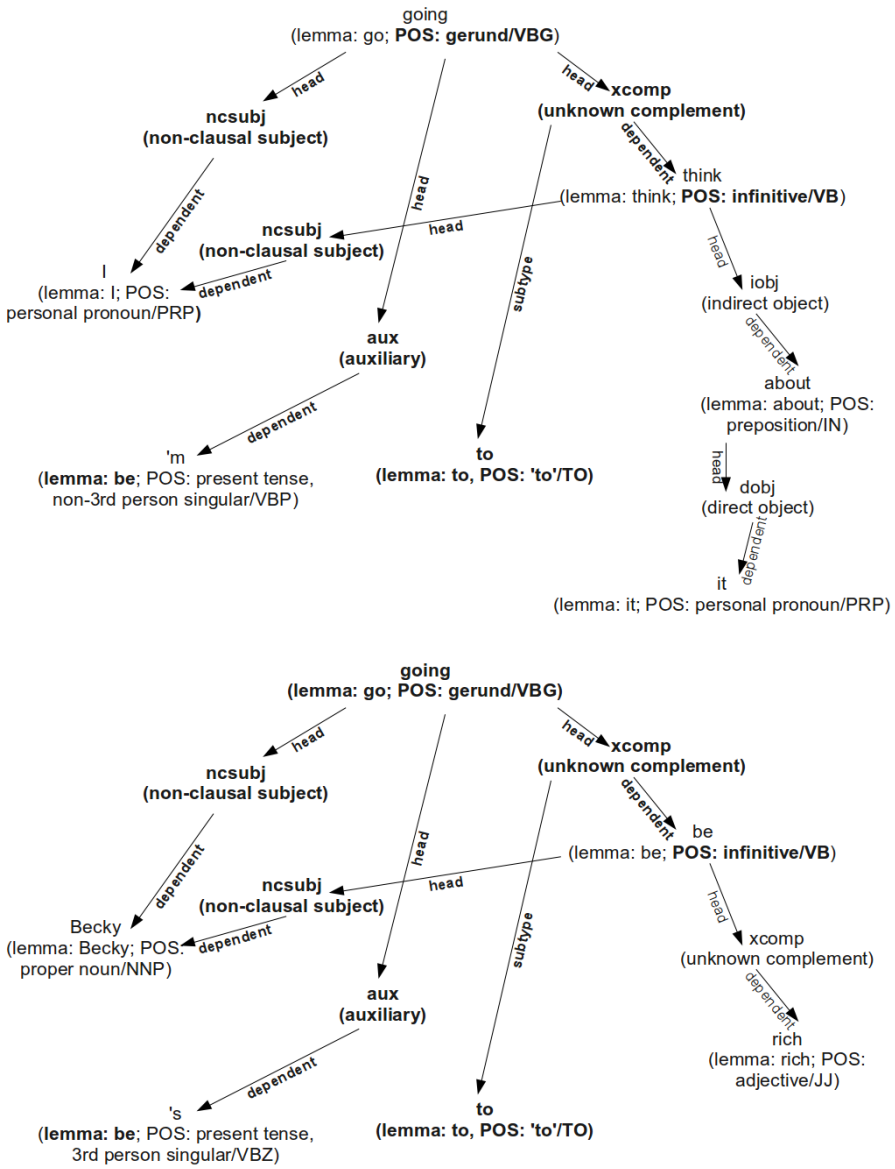

Figure 4: Dependency structures for (a) *I'm going to think about it* and (b) *Becky's going to be rich*, with common aspects in boldface.

coverage. Height: 3.

- <be|VBZ>(aux): as above, but height of 2.
- <rich><be><go>: lemmas only. Height is 5, as the stretch of the graph it covers includes 3 tokens and 2 GRs, even if no aspects of the GRs are actually captured.
- (ncsubj)(xcomp): GR types only, covering the 4 nodes from *Becky* to *be* and then to the uppermost xcomp.

We generate lots of twigs for each example sentence, but then throw away any which don't occur in the whole set of examples.

For some learning tasks, it's useful to be able to distinguish between root and non-root nodes — e.g. to distinguish between embedded clauses and 'top-level' verbs. To model this, we place a null GR and null token above the real root node of every dependency structure. Hence, a height 4 'lemmas and GR types' twig starting from *be* would be <be>(xcomp)<go>(null). This resembles processing with n-grams, where we often have a start-of-sentence token prior to the first token in a sentence [20].

Twigs are the core of the commonality algorithm, but we also determine whether our example sentences share any verb SCFs, using the approach described in §6.2.2. There is some conceptual overlap here, but where SCFs represent broad, static syntactic constructions; twigs are localised, dynamically-created syntactic constructions. SCFs remain useful because they capture a sometimes large number of complex co-occurring aspects of a GR graph into a single feature.

6.1.3 Hierarchical POS & GR Types

It's worth exploiting the hierarchical nature of GR types: a teacher might input one sentence with a non-clausal subject (*I agree*), and another sentence featuring clausal subject (*whether they succeed depends on luck*). Intuitively, our program ought to conclude that what interests the teacher is the presence of subjects, and that whether these subjects are clausal or not is immaterial (this should also mitigate corpus sparsity issues somewhat by broadening our search). In fig. 2, we see that the GR type hierarchy provides for this, as csubj (clausal subject) and ncsubj (non-clausal subject) GRs have a common parent: subj.

A similar argument can be made for POS tags: to recycle the 'going-to' future tense example, we always use a present tense inflection of the verb *be* as an auxiliary. However, the Penn-Treebank gives us two present tense POS tags: VBZ for the third person singular, and VBP for all other forms. Unlike GR types, however, the PTB tagset is fairly flat. In many cases you can strip off the last letter of the POS tag to get a base form: e.g. from VBP (past tense verb) to VB (verb infinitive), or JJR (comparative adjective) to JJ ('vanilla' adjective). But this simple two-level hierarchy is too crude: VBZ and VBP have far more in common with each other (they're different present tense inflections) than they do with other verb forms like past participles (VBN). Indeed, there are tags with multiple logical parents: we might teach the use of superlatives through a mixture of superlative adjectives (JJS) and superlative adverbs (RBS). As such, we want our superlative adjectives to have a 'superlative' parent as well as an adjective parent, and likewise for superlative adverbs. To allow for this, we build an explicit hierarchy of new tags on top of the PTB POS tagset (documented in appendix A).

Lemmas	POS	Supertags	GR Types	Hierarchy Traversal	Maximum Height	Base Score	Depth Bonus
Yes	Yes	No	Yes	No	3	100	50
Yes	No	No	Yes	Yes	5	40	22
No	Yes	No	Yes	Yes	5	14	8
No	No	Yes	Yes	Yes	5	16	10
Yes	Yes	No	No	Yes	3	12	15
Yes	No	Yes	No	Yes	3	14	18
Yes	No	No	No	No	1	5	1
No	Yes	No	No	Yes	1	1	1
No	No	Yes	No	No	1	2	1
No	No	No	Yes	Yes	2	1	1

Table 2: Twig feature configurations. The first four columns describe the aspects of each token and GR that are captured. 'Hierarchy traversal' is whether a configuration involves traversal of the GR type and POS hierarchies, e.g. generating a twig with `obj` when finding `dobj` or `iobj`. 'Maximum height' is the maximum number of nodes (tokens + GRs) in a twig. All salient lower lengths are also considered, e.g. a GR type-only configuration with a maximum height of 4 will create twigs with heights of 2 and 4 only, as nodes denoted by odd indices are tokens. 'Base score' refers to the score assigned to a twig with the lowest salient height for that configuration; the 'depth bonus' is added for each salient height above the base — both of these scoring-related values are discussed in §6.3.2.

This significantly increases the number of features we generate: starting with the twig <|JJR>(ncmod)<|NNPS>, we now generate twigs such as <|ADJECTIVE>(ncmod)<|NNPS>, <|JJR>(mod)<|NNPS>, <|JJR>(ncmod)<|NOUN_PLURAL>, <|JJR>(ncmod)<|NOUN_PROPER>, <|COMPARATIVE>(mod)<|NOUN_PLURAL>, and so forth. Note that we never generate any twigs that include the root nodes of the two hierarchies, i.e. `dependent` for GRs, and `TOKEN` for POS: as catch-alls, they have no discriminative power.

We moderate this somewhat by only turning on this 'hierarchy navigation' for certain twig configurations, but even so, we end up with a vast number of features. We don't perform an exhaustive exploration of the possible permutations of twig height, capture configuration and the option of traversing the hierarchies: see table 2 for the variations used. Note that we're only generating twigs en masse for our example and counter-example sentences; this in of itself is not too expensive. Were we to evaluate thousands of corpus sentences against thousands of features, we would have millions of comparisons to make. Thus, for resource usage and execution time to be acceptable, we need to perform pruning.

6.1.4 Feature Pruning

The simplest type of pruning we can perform is to remove all features from $\hat{M}_w$ (weak commonality) if they also exist in $\hat{M}_s$ (strong). As features in $\hat{M}_s$ are more heavily weighted (see §6.3.2), this is 'lossless'.

The other type of pruning is concerned entirely with the additional twig features we produce through hierarchy navigation (§6.1.3). In said section, we hypothesised that a teacher might want to demonstrate subjects of verbs, without reference to whether clausal or non-clausal subjects are involved: perhaps the teacher's examples are *I like Nablus* and *how quickly Cairo changed is questionable*. One twig of the *Nablus* sentence would be `(ncsubj)<|VBP>`; a twig with the same configuration for the *Cairo* sentence would be `(csubj)<|VBP>`. These two features will ultimately be discarded as neither occurs in every example sentence, but we'll keep a variant that arises from exploring the GR type hierarchy: `(subj)<|VBP>`. However, our approach to generating hierarchical variations of twigs is exhaustive, so some *other* twigs that are common to our examples include `(subj_dobj)<|VBP>`, `(arg)<|VBP>` and `(arg_mod)<|VBP>` (among many others, when the POS hierarchy is considered). These have some value: when looking for counter-example suggestions, a sentence containing another type of verb argument (e.g. a relative clause with `xcomp`) might be ideal: it would have a superficial resemblance to the subject constructions, while remaining distinct. However, this subtlety comes at a heavy price due to the overwhelming volume of features it requires, so at present, we perform 'lossy' pruning to remove such features. Limited testing suggests that in practice, the impact on the quality of output is acceptable, while reducing execution time and memory usage by a couple of orders of magnitude. Note that as pruning is performed *after* rejecting features that are not part of our commonality sets, this pruning process does not simply negate the work performed traversing the POS and GR type hierarchies.

More formally, pruning is performed on the basis of *twig subsumption*, $\rhd$, which takes two twigs A and B as arguments. $A \rhd B$ iff:

1. The feature type profiles of A and B are identical, and
2. *If the capture configuration includes POS tags*: for each token, either the POS in A is an ancestor of the POS in B, or their POS tags are equal, and
3. *If the capture configuration includes GR types*: for each GR, either the GR type in A is an ancestor of the GR type in B, or their GR types are equal.

As all twigs are unique, this operation determines which of two twigs is 'more specific'. Some examples (using $\not\rhd$ to mean 'does not twig subsume'):

- `<|NNP>(arg_mod)<|VBP>` $\rhd$ `<|NNP>(xmod)<|VBP>`
- `<|NNP>(arg)<|VBP>` $\not\rhd$ `<|NNP>(ncmod)<|VBP>`
- `<|PROPER_NOUN>(dobj)` $\rhd$ `<|NNP>(dobj)`
- `<|NNP>(dobj)` $\not\rhd$ `<|PROPER_NOUN>(dobj)`
- `(obj)(comp)` $\rhd$ `(dobj)(xcomp)`
- `<|ARGUMENT>` $\rhd$ `<|PRP>`

For pruning, we don't call a subsumption function, but the process is logically equivalent. When searching through a corpus (see §6.3.4), we do call a subsumption function.

To prune, we divide our twigs into sets, whereby the twigs in each set have the same commonality type (weak/strong/absence-of) and the same feature type profile. Some of these sets will be huge: some profiles have twigs of height 5, capture both GR types and POS, and recurse the GR type and POS hierarchies. Excluding `dependent`, the maximum number of parents for any GR type is 6 (`dobj`, `obj`, `subj_dobj`, `comp`, `mod`, `arg_mod`). Excluding `TOKEN`, the maximum for POS tags is 5 (e.g. `NN`, `NOUN_SINGULAR`, `NOUN_COMMON`,

NOUN, ARGUMENT). As such, in the worst case, hierarchical variations of a single twig (e.g. <|NN>(dobj)<|NN>(dobj)<|NN>) could result in a set with 4,500 ($5 * 6 * 5 * 6 * 5$) elements. The algorithm uses hashes to avoid comparing every element of the set to every other element of the set.

The pruning process is shown in algorithm 1. First, we populate table (a Guava data structure; akin to a map, but with two key lookups, labelled *row* and *column*) *similarTwigs*. Its row keys are *partial hashes*; its column keys are *exclusion indices*. A partial hash is the hashcode of a twig, excluding one node (the exclusion index). For instance, if our twig t has height 4, calling *partialHash*(t, 3) gives us a hash calculated from the GRs and tokens at nodes 1, 2 and 4. Each value is a list of twigs that are almost certainly *very similar* to the current one. More precisely, within a given list, it is highly probable that every twig is identical except for one node[17]. We check for this later on. This table is populated by looping through every twig and every node index, calculating partial hashes, and inserting the twig into an appropriate values' lists in *similarTwigs*. Each twig should end up in a number of lists equal to its height.

The second step is the search for pruneable twigs. Again, we loop through every twig at every exclusion index. We calculate the partial hash, and using said hash, retrieve similar structures from the table. We go through these similar structures, and skip to next if we're comparing the twig against itself or against a twig that (due to hash collisions) isn't similar. Otherwise, we compare the GR types or POS of the twigs at the current index. Whichever twig's GR type or POS is an ancestor of the other is pruned.

6.2 Sentence Generation

Sentence generation was not brought to the conclusion of a full, testable application capable of meeting at least our mandatory requirements. However, it did lead to the implementation of a mapping from GRs to WordNet's SCFs. This mapping becomes an important part of the commonality algorithm as described in §6.1.2. Furthermore, this mapping itself could be a significant, reusable contribution. As such, we discuss generation as far as the mapping; thereafter, we investigate sentence selection.

6.2.1 Basic Dependency Structure Manipulation

We divide the problem of modifying dependency structures to create new sentences in two. First, we need a suite of sentence manipulation tools to generate new sentence candidates. Second, we need some unsupervised measure of *quality* for generated candidates, giving us some assurance of the syntactic (and even semantic) coherence of the result. Here, we give a brief discussion of the first question. The second is left uninvestigated, but a combination of two approaches appears likely: an n-gram language model (as seen in statistical machine translation [7]); and reparsing our candidates: if C&C can only provide a partial parse, we reject the candidate. With these quality measures in mind, when tackling the first problem (sentence manipulation tools), we should aim to create operations that are stackable, with the output at each stage being (at least plausibly) a valid sentence of English.

[17]Because of hash collisions, twigs which differ at more than one node could end up in the same list.

Algorithm 1 Subsumption-based twig pruning. Input: a set of twigs whereby each twig has the same commonality type (weak/strong/absence-of) and the same feature type profile. *partialHash*(t, e) calculates the hashcode of twig t, excluding the node at index e. *twigsPartiallyEqual*(t_1, t_2, e) determines whether twigs t_1 and t_2 are equal at every index except e. *addToListInTable* takes a table, row and column keys, and a twig. It checks whether the table has a list as a value at the specified row and column, inserting an empty one if not. It then adds the twig argument into this list.

```
for all twig do
  for exclusionIdx= 1 →height(twig) do
    partialHash ← partialHash(twig, exclusionIdx)
    addToListInTable(similarTwigs, partialHash, exclusionIdx, twig)
  end for
end for
for all twig do
  for exclusionIdx= 1 →height(twig) do
    partialHash ← partialHash(twig, exclusionIdx)
    similarTwigs ← similarTwigs.get(partialHash, exclusionIdx)
    for all similarTwig do
      if twig ≠ similarTwig and twigsPartlyEqual(twig, similarTwig, exclusionIdx) then
        if exclusionIdx  mod 2 = 0 then {even indexed nodes are for GRs}
          if profile covers GR types then
            grTypeA ← twig.get(exclusionIdx)
            grTypeB ← similarTwig.get(exclusionIdx)
            if grTypeA.ancestorOf(grTypeB) then
              twigs.remove(twig)
            else
              twigs.remove(similarTwig)
            end if
          end if
        else {odd indexed nodes are for tokens}
          if profile covers POS or supertags or lemmas and lemmas and supertags match
          then
            posA ← twig.get(exclusionIdx)
            posB ← similarTwig.get(exclusionIdx)
            if posA.ancestorOf(posB) then
              twigs.remove(twig)
            else
              twigs.remove(similarTwig)
            end if
          end if
        end if
      end if
    end for
  end for
end for
```

The simplest modification we can perform is to replace a single word. We provide word and phrase substitution operations which requires only that the replacement should share the same POS or top-level GR as the original. For this, as we have only a handful of example sentences, we use CCGbank as a pool of pre-tagged candidate word replacements.

This approach to word replacement is trivial: we load the corpus, building up a mapping from POS tags to lists of tokens. Some approximate capitalisation normalisation is applied (e.g. words not marked as proper nouns or inserted at the beginning of the sentence are lowercased). We specify the word to be replaced, and update the dependency structure by replacing the token (preserving the old word index in the sentence) and GR references. Phrase replacement is similar — based on a mapping from GR types to lists of occurrences — but now we replace the transitive closure of the GR's dependent token. For instance, the transitive closure of the `ncsubj` (subject) headed by *wish* in *once again, both of us wish you a merry Christmas* is *both of us*. Substituting in the transitive closure of random instances of `ncsubj` from CCGbank gives interesting results (and a reminder of the biases of newspaper text):

- *Once again, the IRS wish you a merry Christmas.*
- *Once again, about 54% wish you a merry Christmas.*
- **Once again, exile in Hollywood wish you a merry Christmas.*

See fig. 5 for an illustration of phrase replacement. Aside from the semantic curiosities of the above, the *Hollywood* substitution suggests that we could improve quality by enforcing subject-verb agreement.

Another approach to word replacement uses synonyms from WordNet, which covers only open-class words: verbs, nouns, adjectives and adverbs. First, we must identify the word sense. We take the word's lemma, and search WordNet using JWI. For nouns, adjectives and adverbs, we take the first sense[18], as WordNet orders word senses commonest-first. We then retrieve the corresponding synset, and pick a synonym at random. As we've been working with lemmas at this point, we have to inflect our synonym to fit in context — performed with SimpleNLG [16] by combining the new word and the old word's POS. Finally, we substitute the new token into the dependency structure as before.

For verbs, both the identification of word sense and the selection of a synonym is more sophisticated. Every word sense of every verb in WordNet is accompanied by a set of applicable SCFs, given a particular word sense. SCFs for two senses of *fail*:

Sense 1: *She failed to notice that her child was no longer in his crib* (synonym: neglect)

1. *Somebody* —s to *infinitive*

Sense 3: *His sense of smell failed him this time* (synonym: betray)

1. *Somebody* —s *somebody*
2. *Something* —s *somebody*

If our source sentence is *I've failed her this time*, we don't want the first (*neglect*) sense. That is, we don't want to generate the sentence *I've really neglected her this time*, as the semantics are different from the original sentence. The third sense is a better fit; swapping in a synonym like *betray* would be more likely result in a semantically coherent sentence.

[18] Using Simplified Lesk [21] would be an improvement.

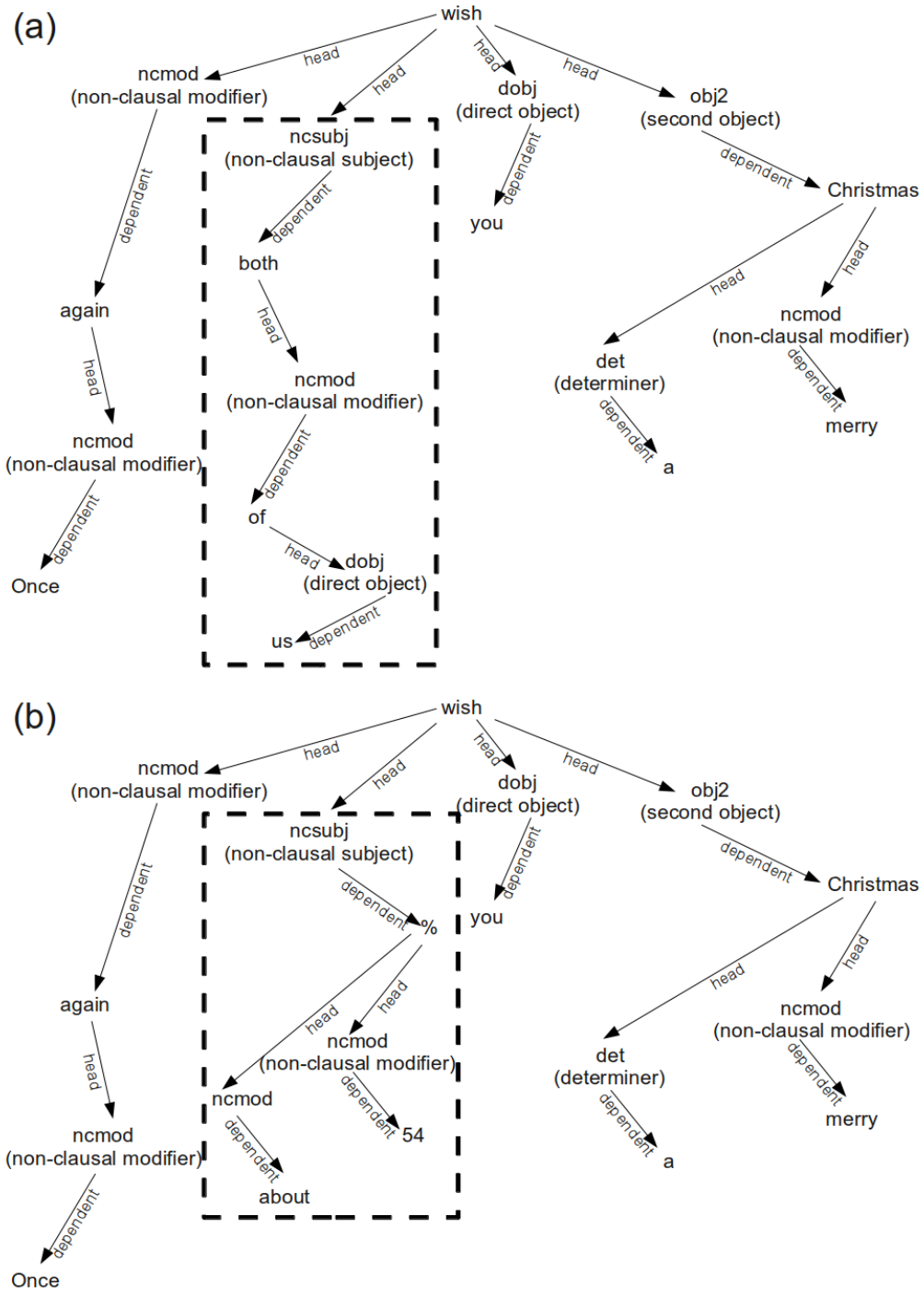

Figure 5: Replacement of the transitive closure of the `ncsubj` GR headed by *wish* in the sentence *once again, both of us wish you a merry Christmas. Both of us* in (a) becomes *about 54%* in (b).

Correctly identifying the appropriate SCFs allows us to filter out inappropriate synonyms using WordNet. To find appropriate verb senses for a verb v, we determine:

- F: the set of SCFs that v matches given its context in the sentence, and
- $z(v_i)$: the set of SCFs applicable to sense i of v, as found in WordNet.

If $|F \bigcap z(v_i)| > 0$, then i is a candidate word sense. As before, we pick the first (most common) candidate at present. We retrieve the word sense's synset, and the process is repeated to determine which synonym verbs are eligible like-for-like replacements: unlike Levin classes [25], being in the same WordNet synset doesn't guarantee that all constituent verbs will accept the same SCF.

6.2.2 GR-WordNet Verb Subcategorisation Frame Mapping

GRs form a crucial part of the mapping: for instance, with SCF *Something —s something*, we can represent the second 'something' as a direct object (`dobj`) of the verb (further constraints are discussed later), and the first as some kind of subject (`subj`). The hierarchical structure of the GR specification (see §4.2) is important: we want to be able to express that the SCF *Something —s something* has some kind of subject (`subj`), without specifying whether it's a clausal subject (`csubj`) or non-clausal subject (`ncsubj`).

Each of the 35 WordNet SCFs is represented using the set of acceptable POS tags for the verb itself; and a set of *constraints*. A constraint defines mandatory features of a matching GR graph. A constraint can define one or more of the following requirements:

- GR type,
- GR slot,
- Remote token slot,
- Remote token POS tag,
- Remote token lemma, and
- Remote token nested constraint
- Semantic type (*someone* vs *something*)

The subject *he* and the verb *failed* in *he failed her* are linked by a `ncsubj` GR. In our terminology, *failed* would be a 'remote token' with respect to *he*, as we can reach *failed* by traversing a GR.

'GR slot' refers to the fields in a GR that can reference a token. Valid values are `HEAD`, `DEPENDENT` and `SUBTYPE`. `SUBTYPE` words are grammatical markers such as infinitive *to* and complementiser *that*; `HEAD`s define the syntactic function of the phrase they contain; `DEPENDENT` words typically qualify `HEAD`s.

WordNet SCFs distinguish between personal (*someone*) and impersonal (*something*) subjects/objects. C&C's `I-PER` (person) named entity tag gives us the distinction for proper nouns. For common nouns, we look in WordNet at whether the lemma of the first word sense has *person* or *animal* as a hypernym, and flag as *somebody* if so, e.g. the first sense of *soldier* and *friend* have *person* as a hypernym. Personal pronouns other than *it* are marked *somebody*. See fig. 6.

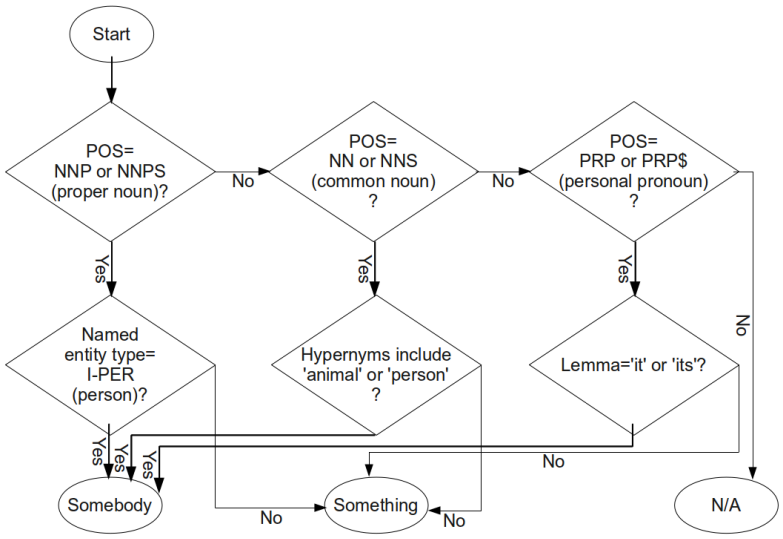

Figure 6: Use of POS, named entity tags and WordNet to flag a noun phrase head as *somebody* or *something*.

In *he failed her*, we might wish to place some constraints on the remote token along the `ncsubj` GR from the verb. For instance, to limit this remote token's POS tag to `NN` or `PRP`, we specify the constraint `[grType:=ncsubj,slot:=HEAD,remoteSlot=DEPENDENT,remotePos∈{NN,PRP}]`. Constraints can be combined using `AndConstraint`, `OrConstraint` and `NotConstraint`.

As a less trivial example of a SCF definition's constraint set, consider *Somebody —s something on somebody* (a ditransitive verb SCF). `slot=HEAD` and `remoteSlot=DEPENDENT` unless stated.

- `verbPos ∈ {VBD,VBN,VBP,VBZ}`
- `[grType=subj,semType=SOMEBODY]`
- `[grType=dobj,semType=SOMETHING]`
- `[grType=iobj,remoteLemma=on,remoteNested=[ grType=dobj,semType=SOMEBODY]]`
- `[not[grType=clausal]]`
- `[not[grType=pcomp]]`
- `[not[grType=obj2]]`

This ensures that infinitives (`VB`) and gerunds (`VBG`) are excluded; requires a 'somebody' subject, a 'something' direct object, and the word *on* as an indirect object, which itself must have a 'somebody' as a direct object; but rejects any GR graph with other types of complement (`clausal`, `pcomp`). The subject requirement (`grType=subj`) can be satisfied with a subtype of `subj`.

It's important to note what parts of the GR type hierarchy the mapping *doesn't* restrict.

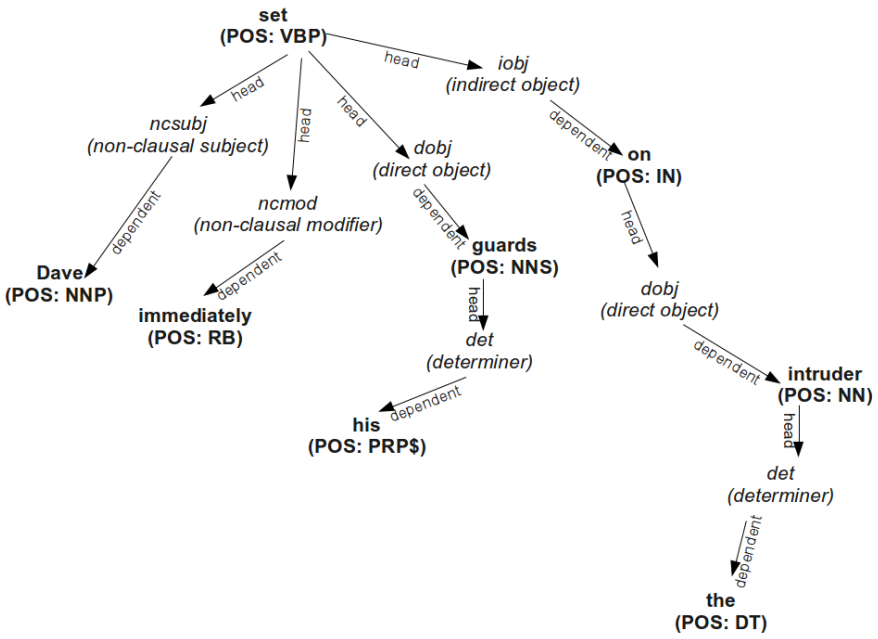

Figure 7: The GRs and tokens surrounding *set* match our constraints for *Somebody —s something on somebody.*

In fig. 7, the above SCF definition will match *set*: *immediately* is linked to *set* through an `ncmod`, a type not restricted by this SCF (or any other).

6.3 Sentence Selection from a Corpus

6.3.1 Linking in CCGbank

CCGbank [18] gives us CCG derivations for the virtually all of the Wall Street Journal section of the PTB [26]: 46,532 sentences in total. Several resource and performance concerns must be considered with a corpus of this size. One can of course take a subset of the corpus, but not all sentences in the corpus are created equal. In particular, we wish to satisfy requirement 8: that sentences should be as simple as possible, given the point being demonstrated. Sentence length correlates well with subjective measures of complexity [15], so a simple way to achieve this is to select short sentences.

We apply a sentence length cutoff — the length-complexity tradeoff is illustrated in fig. 8. After some experimentation, a threshold of 13 tokens was chosen, giving 8,688 sentences and a worst-case Flesch reading ease score of 70.7%, described by Flesch as 'fairly easy' [15]). Any higher, and we drop out of the 'fairy easy' band; this is noteworthy when potentially targeting learners of English. We discuss this decision further in §7.2.

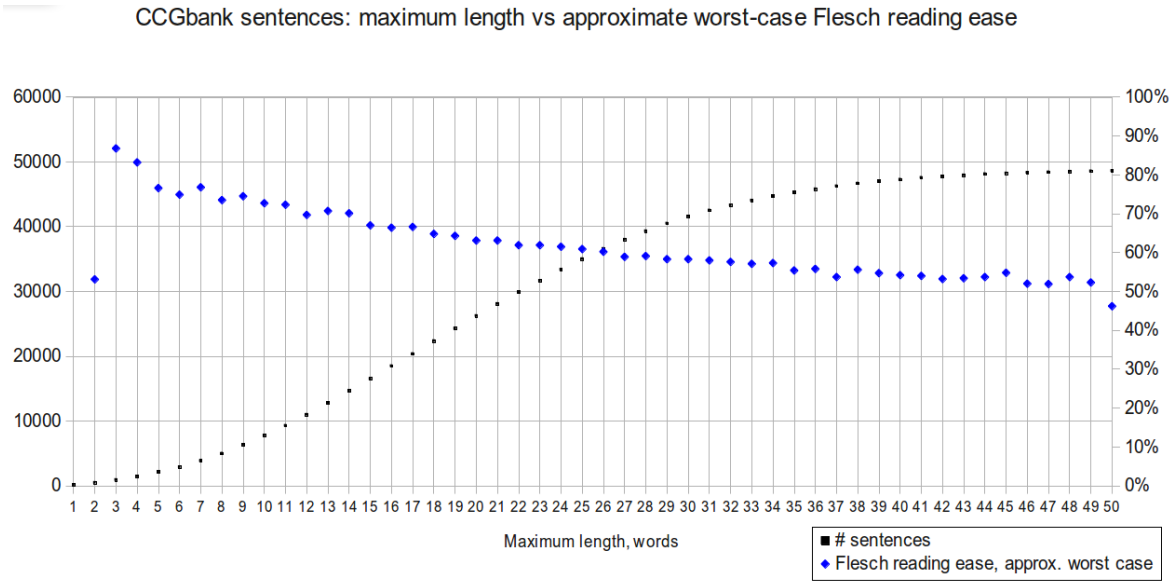

Figure 8: As we augment the maximum length (in words) of sentences lifted from CCGbank, a tradeoff occurs between sentence complexity and availability. The left-hand y axis represents cumulative sentence counts with a maximum length. The right-hand y axis gives Flesch reading ease [15] scores, calculated across sentences with a length exactly equal to the x axis value, to represent worst-case complexity.

6.3.2 Assigning Scores

Given some set of sentences P, can then pick out the top n sentences, as scored by a scoring function r.

$$r(s) = \sum_{g \in (f(P_k) \cap \hat{M})} w(g) \tag{7}$$

Where w gives the weight of feature g; and P_k is the sentence from P that we want to evaluate. The weight of a twig is the product of the following:

1. *Concreteness*: the product the twig's node weights. A node has a weight of 1 if it captures only supertags or lemmas, or if its POS or GR type is a leaf in the appropriate hierarchy (e.g. `ncmod`, `dobj`, `VBD`, `NN`). Non-leaf nodes each have fixed weights (e.g. `obj`=0.8, `argument`=0.6, `VERB_PRESENT`=0.8, `VERB`=0.5. Full list in appendix A).
2. *Commonality type multiplier*: 2 for weak; 5 for strong/absence-of.
3. *Configuration & height score*: 'base' and 'depth bonus' scores apply according to twig length. See table 2.

Strong twig `<|ARGUMENT>(subj_dobj)<|VBD>(cmod)` scores 36:

1. *Concreteness*: 0.24 $(0.3 * 0.8 * 1 * 1)$.
2. *Commonality type multiplier*: 5.

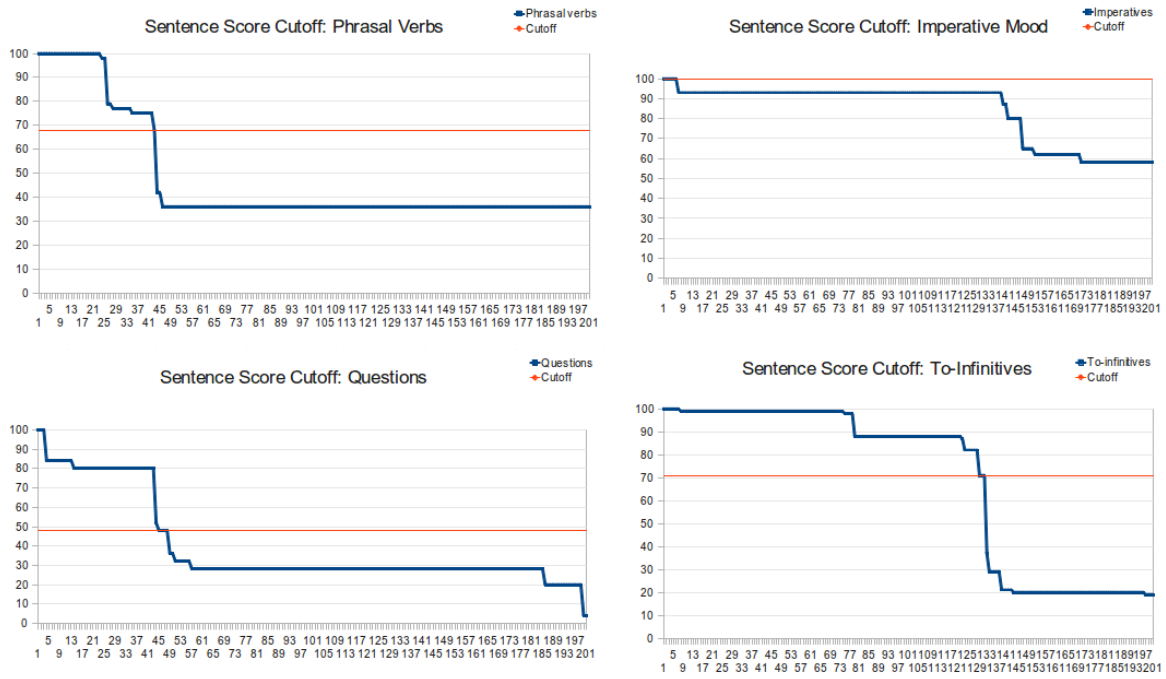

Figure 9: Sentence score cutoffs for a variety of grammatical learning tasks. The manually-validated cutoff varies wildly by learning task.

3. *Configuration & height score*: 30 $(14 + 2 * 8)$. b is 14, applicable to minimum applicable height of 2, as profile scores both GRs and tokens. Two increments of d, which is 8, as twig height is 4.

We also add 40 if there are ≥ 1 SCFs in common. All this gives us a large, hard-coded parameter space, configured through an error-prone process of trial-and-error. We consider optimising this with maxent in §8.1.

6.3.3 Selecting Example & Counter-Example Suggestions

With a corpus and sentence rating function r (see equation 7), to select the best example suggestions, we take the highest-scoring sentences.

However, we also need counter-example suggestions. That is, sentences which *do not* demonstrate the grammatical point in question. We could find such sentences simply by taking the lowest-scoring sentences, as scored by r above. However, we want counter-examples to have a shallow, superficial resemblance to the learning task (see §2). For instance, when teaching the present continuous (*I'm going home soon*), ideal counter-examples would include the past or future continuous (*I'll be going home soon*; *I was going home*). We investigated using sentences scoring ~20% of the top score, but found that when learning tasks consists of low-scoring features (e.g. questions, where the only commonality is a token with POS `.` and lemma *?*), this threshold occasionally gives us false negatives in practice. No unsupervised means of producing superficial commonality without significant risk of false negatives has been identified.

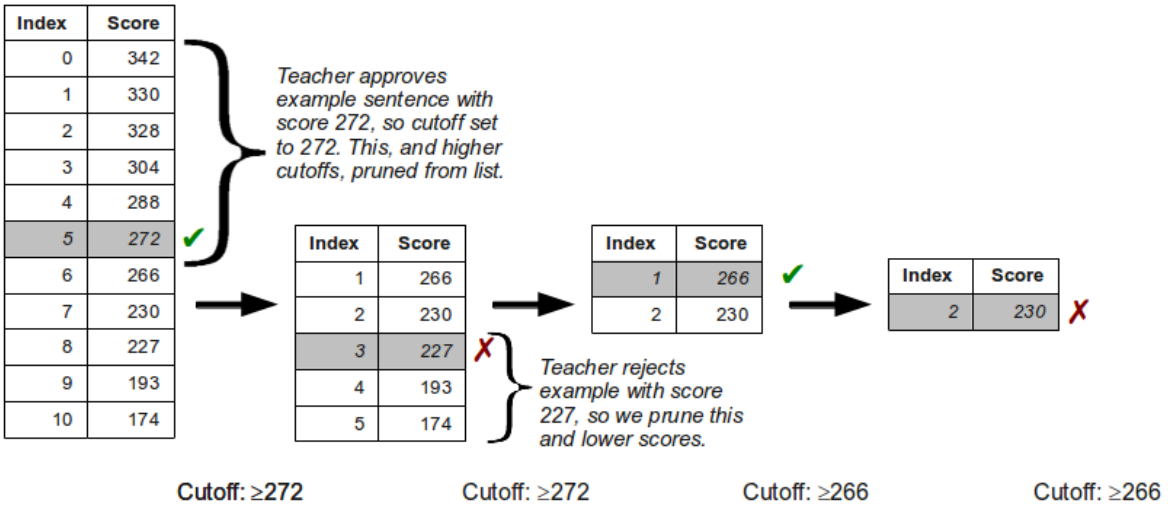

Figure 10: Supervised selection of sentence score cutoffs. Sentences presented to teacher for evaluation highlighted and in italics.

An important issue for example suggestions is that even if we're confident that we've successfully placed sentences in best-first order, it's not obvious how we determine where the high-quality sentences stop and the noise starts. We impose a *unsupervised cutoff* at 68% of the highest-scoring sentence. This value was derived by plotting the distribution of scores for 7 different learning tasks: 68% proved a conservative but safe threshold in 5/7 of these cases. The distributions offered no obvious patterns for automatic differentiation on a case-by-case basis (see fig. 9). There may be merit in trying to set the cutoff dynamically at the point at which scores take a sudden plunge: in this figure, taking everything up to the last sentence before the big drop in score for phrasal verbs, questions and to-infinitives would get us close to the correct value, but not for the imperative. Defining what constitutes a 'big drop' would be tricky: with phrasal verbs, output is acceptable until the second, larger dip.

An alternative cutoff involves teacher supervision. After applying an unsupervised 50% quality cutoff, we group examples by score, and take one example suggestion per unique score, giving us an ordered (best-first) mapping from a score to an example. We present the teacher with the example corresponding to the modal score, asking if it's exemplifies the grammatical point. If so, we delete all mappings from the highest score to the current score (inclusive) and update the threshold to the current mapping's key; otherwise, we delete mappings from the lowest score to the current score (inclusive). We repeat, taking the modal score from our smaller mapping and pruning again, stopping after 5 iterations or once our list has only one element (see fig. 10).

In the evaluation, we report precision using both the unsupervised and supervised cutoffs described above.

Type	# Files	Compression?	Size, MB	Timing, seconds
C&C output	2312	Uncompressed	67	51
C&C output	1	Uncompressed	44	43
C&C output	1	Compressed	10	45
Serialized graph `List`	1	Uncompressed	144	48
Serialized graph `List`	1	Compressed	25	62

Table 3: Corpus load statistics. Each timing is the mean of three consecutive attempts. No caching effect observed between runs.

6.3.4 The Search

At present, the corpus search for example and counter-example suggestions is exhaustive. See §8.2 for how this might be addressed. There is one important optimisation: it never creates extra twigs via the POS and GR type hierarchies as we did for identifying commonality in §6.1.3. Instead, we create twigs for each corpus sentence, and then check whether they're *subsumed* (see §6.1.4) by a common feature. For instance, perhaps one weak commonality feature is `<|VERB_PRESENT>(comp)`. `VERB_PRESENT` and `comp` are not leaf-level nodes, so won't be generated from any corpus sentence. However, we'd assign a corpus sentence points if we generate `<|VBZ>(xcomp)` from it, as `<|VERB_PRESENT>(comp)` ▷ `<|VBZ>(xcomp)`.

6.3.5 Corpus Load Time

CCGbank provides parses in two formats, labelled `AUTO` (machine-readable) and `PARG` (human-readable). Each divides its 46,532 parses across 2,312 files in 25 directories. When using a script to translate the `AUTO` files into GRs, the corpus load takes 52 seconds on the development machine[19] if we replicate the `AUTO/PARG` structure. We experimented (table 3) with loading serialized dependency structure objects instead of C&C's output format, and placing all GR parses in one file: the latter was the more successful approach. Despite the parsing logic being a non-trivial composition of graphs, heavily dependent on regular expressions, the cost of Java serialization appears to outweigh the benefits of bypassing parsing, perhaps because the serialised files were larger, or because of the circular relationships between the GRs and tokens. Deserialisation also required raising the max heap size to 1024MB.

6.4 Google Android

Most development and testing (including the evaluation) of the core application was performed on a laptop, but the application does work on Android, with some limitations. See fig. 11.

Some issues are manageable. At present, we load 1,000 pre-parsed corpus sentences. This takes 42 seconds. This occurs in a background thread, and entering 3–5 example sentences on a touch screen is likely to eclipse this. Counter-examples being typically less sensitive, we

[19] Asus X52F-EX469V laptop: Pentium P6100 2GHz CPU, 4GB DDR3 RAM, SATA-150 5400rpm 320GB disk using the ext4 filesystem (employed by Android v2.3) with 4096-byte blocks, Ubuntu 10.10 64bit.

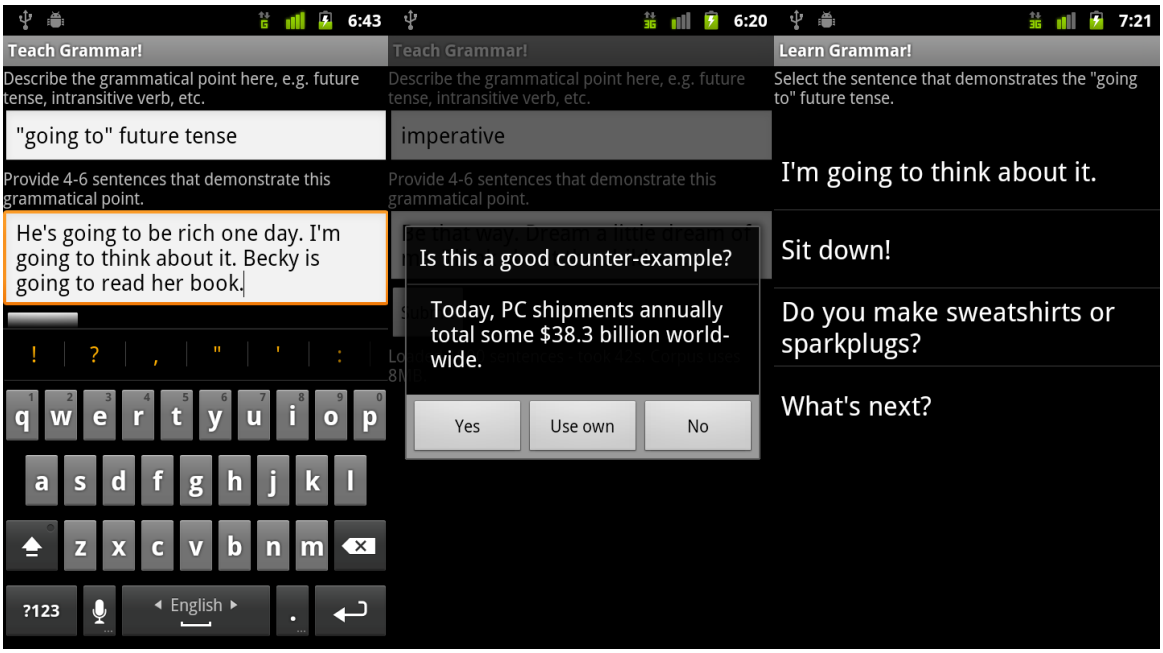

Figure 11: *LearnGrammar!* screenshots. From left: teacher mode, inputting example sentences; teacher mode, selecting counter-examples; student mode, selecting the sentence that demonstrates the 'going to' future tense.

let the user select them through a series of yes/no prompts. At present, parsing of teacher-provided sentences is performed through a webservice call. There are three larger issues:

1. At present, only 1,000 corpus sentences are loaded; this is rarely enough. Even the 8,688 used in evaluation appears to be insufficient in many cases (see §7.2).
2. Searching the corpus takes 2–5 minutes (in the foreground), depending on the task.
3. Even with 1,000 sentences, it was necessary to lift the 24MB per-application memory ceiling on the Nexus One handset.

These three problems are linked. In §8.2, we look at how we might address them.

7 Evaluation

7.1 Evaluation against Requirements

With reference to our requirements (see §2):

1. *Implementation of teacher and student modes*: fully met.
2. *Generalisation from minimal input*: evaluated in §7.2; generalisation algorithm works well for a variety of learning tasks, but accuracy is low or zero for some other tasks. Partially met.
3. *Presentation of correct and incorrect answers to student*: fully met.
4. *Variety of learning tasks*: see §7.2; largely met.

5. *Definition of new learning tasks without code changes*: fully met.
6. *Volume of output*: largely not met, though recall figures (§4) suggest this is largely due to data sparsity, not weaknesses of the approach.
7. *Superficial resemblance of counter-example suggestions to learning point*: not met. It was necessary to sacrifice this to get acceptable precision (see §6.3.3).
8. *Simplicity of suggested sentences*: partially met through capping maximum length of sentences (see §6.3.1).
9. *Application should run on a smartphone*: fully met.
10. *Operating system extension and configuration changes should be unnecessary*: not met. A configuration change is required to remove Android's 24MB cap on memory usage.
11. *Responsivity*: partially met through a background thread (§6.4) for the corpus load. However, calculating commonality and searching for similar sentences takes a substantial period of time and runs in the foreground.
12. *Network access non-dependence*: partially met. Not required in student mode, but teacher-provided sentences are parsed via a webservice.

7.2 Expert Evaluation of Output

A variety of learning points were lifted from sections in *English Grammar Today* [8]. Where a section deals with multiple phenomena, I reduced the scope of the task. For instance, for section 293c (*Punctuation: commas*), I focus on list separators. Each section is illustrated with example sentences; I used these as input (see appendix B for selected sentences). For counter-examples, I took sentences from other sections. Output for 21 of our 29 learning tasks (lists of example and counter-example suggestions) was sent to an EGT contributor. He was instructed to work through the list of example suggestions, marking each as 'correct' (demonstrates the learning task in question) or 'incorrect'. Output for 8 tasks (asterisked in table 4) was not sent because cursory inspection showed that none of the example suggestions were remotely relevant. For these 8 tests only, I assessed the counter-example suggestions myself.

Test scope was limited to syntax and morphology. Some learning points were excluded as they have semantic elements, e.g. adverbs of degree (e.g. moderately, highly, extremely, totally) vs duration (briefly, temporarily, forever). Delexicalised usage of verbs (where actions are described by nouns, not verbs) is also problematic, as few syntactic markers are available, and hence considered out of scope. Compare *I took a shower* (delexicalised) with *I showered* (not delexicalised).

In §6.3.3, I discuss supervised and unsupervised cutoffs for the list of example suggestions. Thus, I report two precision scores in table 4. *Precision-S* approximates the precision expected from a supervised cutoff by stopping after three consecutive incorrect sentences. *Precision-U* uses our unsupervised cutoff. Sometimes the unsupervised cutoff gives us thousands of sentences: I approximate precision-U conservatively by stopping evaluation after five consecutive incorrect sentences, and assuming that the rest are invalid.

Counter-examples have proven far easier to generate in large volumes with near-perfect accuracy; per task, 50 counter-example suggestions (25 from the top of the list, 25 from the middle)

were evaluated. Within these samples, the precision of the counter-example suggestions for all but two cases was 100%.

Using sentences from EGT helps reduce the risk of internal knowledge of the application biasing the results. However, EGT doesn't always have the three examples necessary for optimal results. I used other sentences to 'top up' to three in such cases. On occasion, EGT gives significantly more than three examples, giving us a choice. Ideally, the application's developer shouldn't be making such choices. Originally, I circulated a request among a number of linguists, describing the application and asking for examples and counter-examples, to conduct a blind trial. A small prize was offered, with each test case provided to be entered into a raffle. Unfortunately, insufficient responses were received.

On several occasions, C&C parsed an example from the book incorrectly, resulting in little of the intended commonality being found. For instance, EGT illustrates *About as a preposition* with *We moved house about three years ago.* C&C parses this *about* as an adverb, foiling exactly the distinction we're trying to establish[20]. In these instances, I chose another sentence from EGT where available, or otherwise, made one up; in table 4, I indicate this with the 'Sensitive Data?' column[21]. The reader might argue that C&C is a component of the application and its errors shouldn't be controlled for; if so, precision and recall should be read as 0% in such cases. Where all sentences were misparsed by C&C, I still report results.

In some instances, I've calculated the number of 'correct' sentences available in the corpus; from this, I calculate recall. This was performed by creating a shortlist of sentences that *might* be 'correct' using `grep`[22], and then manually checking each sentence. This approach is dependent on the PTB (and hence CCGbank) being correctly POS tagged; while occasional inaccuracies have been observed, the PTB is considered a 'gold standard' resource. For instance, `grep -i "that|IN" <8688corpusfile>` gives us 144 sentences that should comprise every available *Noun + that-clause*, plus false-positives. This is labour-intensive so was not performed in every case. Appendix B details the commands used.

But which statistics are relevant? I present recall out of convention, but it's acceptable to have low recall if we still retrieve enough correct examples. Precision, meanwhile, must be as close as possible to 100% for a learning task to be useful.

7.3 Discussion

Despite its limitations, the application does perform well in a variety of these tests, finding good examples of reported speech, the passive voice, different types of relative clause and the present continuous. Strikingly, some of the seemingly simpler tests involving word usage according to part of speech have middling performance. Sometimes, this is a tagging problem: with *All as a pronoun*, the PTB and C&C consistently tag *all* as a determiner, not a pronoun. Whether this is a difference of opinion between the PTB and EGT teams or a tagging error in PTB, this gives the application little chance. Similarly, C&C appears to never tag *still* as an

[20] In CCGbank, many seemingly prepositional uses of *about* are marked as adverbs.

[21] 'Sensitive Data?' is negative where I only used additional 'made up' sentences because the book didn't provide enough examples.

[22] `http://pubs.opengroup.org/onlinepubs/009695399/utilities/grep.html`

Title	**Sens. Data?**	**Precision-S**	**Precision-U**	**Recall -S**	**Correct -S**	**Available**	**Ctr-Ex Precis.**
Adjective phrases with verbs	No	96.9%	89.4%		511		100.0%
Passive	No	98.6%	98.6%		143		100.0%
Present (continuous)	Yes	97.6%	95.3%		121		100.0%
Verb + that-clause	No	91.7%	77.6%	68.1%	77	113	100.0%
Punctuation: commas (as list separators)	No	90.0%	90.0%		54		100.0%
About as a preposition	Yes	90.0%	36.0%		54		100.0%
Reported speech (direct)	No	85.1%	32.1%		40		78.0%
Only as an adverb	No	96.9%	0.7%	47.7%	31	65	100.0%
Up as a particle	No	100.0%	0.5%	40.9%	27	66	100.0%
Noun + that-clause	Yes	76.5%	0.7%	54.2%	13	24	100.0%
Though meaning 'however'	No	100.0%	100.0%	100.0%	17	17	100.0%
As ... as	No	100.0%	41.7%	26.3%	5	19	100.0%
Adjuncts	Yes	42.1%	1.9%		8		92.0%
Wh-cleft	No	87.5%	43.8%		7		100.0%
Plenty of as a quantifier	No	100.0%	0.6%	83.3%	5	6	100.0%
Adjective + that-clause	Yes	75.0%	2.7%	50.0%	4	6	100.0%
Either ... or	Yes	100.0%	100.0%	25.0%	2	8	100.0%
Future (perfect)	No	100%	5.6%		2		100.0%
Do as an auxiliary verb (emphatic forms)	Yes	66.7%	0.0%		2		100.0%
Do as an auxiliary verb (question tags)	No	100%	0.15%	100%	1	1	100.0%
Future (continuous)	No	100%	2.17%		1		100.0%
Still as an adjective*	No	0.0%	0.0%		0		100.0%
All as a pronoun*	No	0.0%	0.0%		0		100.0%
Possessives with *of**	No	0.0%	0.0%				100.0%
As: simultaneous changes*	No	0.0%	0.0%		0		100.0%
Now as a discourse marker*	No	0.0%	0.0%	0%	0	4	100.0%
It-cleft*	No	0.0%	0.0%		0		100.0%
Apposition*	No	0.0%	0.0%		0		100.0%
*No matter**	No	N/A	N/A	N/A	N/A	0	100.0%
Overall	*22/29*	*67.7%*	*29.3%*	*54.1%*			*98.93%*

Table 4: Evaluation against learning points from EGT. Headings marked '-S' use a supervised threshold; those marked '-U', an unsupervised threshold. See §7.2 for a discussion of this and 'Sens[itive] Data?'. 'Correct' is the manual count of correct examples; 'Avail[able]' is a manual count of the number of matching corpus sentences available; 'Ctr-Ex Precis[ion]' gives sampled precision of the counter-example suggestions. I evaluated tests marked * myself; see §7.2.

adjective; nor any nominal pronoun as PRP[23], foiling *Possessives with of*. Also, isolated POS tags are easily drowned out by the slighted unintended commonality, as large twigs are heavily weighted. This looks to have been the problem with *Up as a particle*. We see the value of including verb SCFs in *Adjective phrases with verbs*: all the highest-scoring examples include the SCF feature *somebody —s adjective*. The *Adjuncts* test introduced another challenge: the validity of the complement-adjunct distinction has been questioned [36]; evaluating this particular test may have been excessively subjective.

The recall figures are encouraging, but the shortest 8,688 corpus sentences just aren't enough for most test cases. Our *No matter* test case is impossible: the construction never appears in the corpus. On closer inspection, while the 8,688 represent 18% of the available sentences in CCGbank, as they're the shortest sentences, they cover only 7.2% of the available *words* (72,187/1,000,923), and 0.08% of available (i.e. non-unique) bigram occurrences[24]. If we assume that the frequency of most grammatical structures correlates well with bigram occurrences, then perhaps cases like *Plenty of as a quantifier* could go from 6 available instances in the corpus to 1,153 — though this would revive the readability issue (see §3).

As: simultaneous changes is a clear example of where further extensions to twig feature types could improve accuracy. On manual inspection, the crucial common factor here in sentences like *As you get older, moving house gets harder* is that *as* is always the head of a ccomp with a verb dependent, and always the dependent of a cmod headed by another verb. What's missing is a 'forked' twig, capturing both of these GRs atomically. A further development of this idea would be required for *Do as an auxiliary verb (question tags)*. The current problem is that both in illustrative sentence *She does look so tired!* and counter-example *I don't want to wait for a bus*, *does* is an auxiliary of the main verb. So this do-auxiliary is only weak commonality; other, inconsequential commonality dominates scoring. In the counter-example, we also have *n't* modifying the verb. A solution would be a forked twig (here, joining at the verb at the top) expressing co-occurrence on a single token of the absence-of commonality for the *n't* and the presence of the *do* auxiliary.

Perhaps the biggest question raised by this evaluation is whether state-of-the-art statistical syntactic parsers like C&C are accurate enough for a task so sensitive to minor parsing errors. In §8.4, I look at a potential mitigation. It's noteworthy that there's no single test that clearly indicates application performance suffering from hard-coded weight parameters.

Finally, a few [laptop] execution timings are given in table 5. I don't investigate these further here, though the proposals in §8.2 should reduce these considerably. The number of common features appears to affect corpus search time weakly ($\rho = 0.615$); more comprehensive testing would be required to establish causation.

[23] Feminists might note that the entire PTB doesn't contain the word *hers*.

[24] Mean corpus sentence length: 16.2; thus 15.2 mean bigrams/sentence; 707,286 total. Mean length in 8,688 sentence subset: 7.5, thus 6.5 bigrams/sentence; 56,472 total (0.08%).

Task	#Features	Execution Time (Seconds)	
		Commonality	Search
As ... as	30	1	26
Passive	27	2	27
Though meaning 'however'	20	<1	22
Present (continuous)	17	<1	19
Wh-cleft	16	1	22
Adjective phrases with verbs	11	2	24

Table 5: Test execution time samples.

8 Future Work

8.1 Maximum Entropy Model Training

Assigning fixed weights derived by trial-and-error with a few test cases is crude. Weightings could surely be improved using maxent models, as applied to many NLP problems such as POS tagging [32] and parsing [33]; it might also give us a 'cleaner' threshold (fewer false-negatives/positives on either side) between correct sentences and noise within our example list — crucial for supervised thresholding.

Maxent requires lots of pre-classified training data: enough to define weights with confidence; varied enough to model the target domain. Our training data would amount to a 2D array of booleans: corpus sentences along one axis, learning points along the other, and whether the sentence in question exemplifies the learning point in question in the cells (see fig. 6). During evaluation, I calculated recall across 8,668 sentences for several learning tasks. These figures represent 'low-lying fruit': those amenable to reliable manual identification through careful use of `grep` and manual verification. A reasonable result would require flagging sentences for a broad range of learning points, inevitably including many not amenable to this approach. The manual effort is considerable: the evaluation demonstrates that a far larger slice of the 46,532-sentence corpus is needed. Even with some filtering, each would require yes/no decisions against dozens or hundreds of tasks.

The open questions are these: to what extent would diverse learning tasks 'agree' on a set of one-size-fits-all weights? do we risk over-fitting our training data? Of course, we already have a set of common weights, currently hard-coded; surely a more rigorous method couldn't fail to improve on this. Another issue: at present, all leaf-level POS and GR types are equally weighted. It's not obvious whether parameterising these weights increases the risks of over-fitting unacceptably.

8.2 Resource Usage

For the application to be practical for Android smartphones, we need to search more than 10,000 sentences, keep memory usage below 24MB and drastically reduce corpus search times. An index might help us achieve all this, at the expense of extra disk usage. We create an index which maps from the hashcode of a feature to a list of corpus sentence references where

Sentence	**Verb+that-clause**	**Passive voice**	**Present continuous**	**List sep. comma**	**...**
Prices were lower in Frankfurt, Zurich, Paris and Amsterdam.				X	
Now it's happening again.			X		
They're buying an operation that's running well.	X		X		
The date was misstated in Friday's edition.		X			
...					

Table 6: Mockup of maxent training data.

the feature has been observed. The sentence references would be offsets into the corpus. If the mappings in this index are ordered by feature hash, then we can avoid loading the index into memory through a simple hash-to-index-file-block function $b(h)$:

$$b(h) = \frac{b_{max}}{2^{31} - 1}(h + 2^{31}) \tag{8}$$

Where h is a feature hash, and b_{max} is the last block in the file. Java hashes are signed 32-bit integers, and hence range from -2^{31} to $2^{31} - 1$. Looking up the sentences containing some feature x would involve identifying our index block through $b(hash(x))$, finding $hash(x)$ in the block, retrieving the corresponding sentence pointer list, and seeking to those pointers in the corpus file (see table 7). But we're not interested in finding sentences with one feature: we want to efficiently find sentences that have *all* (or most) of our common features. We can achieve this by building a $\langle$*sentence-pointer*,*score*$\rangle$ list. We don't want to calculate scores for all 46,532 sentences, but this is avoidable. Sorting common features in descending weight order, we can disregard sentences when they're no longer likely to score highly.

This strategy is incompatible with subsumption (see §6.1.4). We would have to index all twig variants produced through GR type and POS hierarchy navigation. This would considerably increase the size of the index; modelling features found in sentences with Bloom filters [39] might make this manageable.

Given the low probability of hash collisions with 2^{32} possible values, to save memory, we could just store sentences on the phone, not their parses (except for ~100, for counter-example selection), and never validate features against dependency structures.

8.3 Disjoint Set Commonality

The imperative is tricky to capture in several respects. It was the motivation for 'absence of' coverage (see §6.1). But we've brushed over an important point: while the second person form of the imperative is distinguished by a subjectless verb, there's also a first person plural form, formed with *let's*: *let's go to the park*. These two constructions are syntactically distinct,

(a) Hashing (Function)	
Twig	**Hash**
...	...
<|JJ>(ncmod)<|NN>	*-123345*
<|JJ>(ncmod)	-122282
<be|VBP>	-122049
...	...

(b) Index (Disk)	
Hash	**File Offset**
...	...
-123345	*0xf32ab083*
-122282	0x32e8f988, 0xf32ab083
-122049	0xb8430e01
...	...

(c) Scores (Memory)	
Sentence	**Score**
...	...
0xf32ab083	*3453*
0xb8430e01	2833
0xf32ab103	2798
...	...

(d) Corpus file (Disk)	
File Offset	**Sentence**
...	...
0xf32ab083	*We were out of the box.*
0xf32ab09b	We're in the dark, he said.
0xf32ab103	That's for the future.
...	...

Table 7: Proposed mappings for less resource-intensive corpus searches: (a) shows a twig hashing function; (b) is an on-disk index mapping from feature hashes to offsets into the corpus file for sentences with that feature; (d) is the corpus itself, illustrated with sentences at various byte offsets; (c) is an in-memory data structure, tracking sentences' scores. Follow the italics to see the mappings chained together.

but together express a single idea. The current implementation would be able to detect and extrapolate sentences from either form individually, but wouldn't find commonality given a mixed set of sentences, e.g. both *go to the park* and *let's go to the park*. What we want is to split our example sentences into two groups automatically: one group filled with the second person imperative sentences (i.e. those without subjects), the other with first person plural imperatives (those with *let's*). Then, we'd search the corpus for sentences that score highly against *either* feature group. First person plural imperatives would score highly against the group containing features like <us>(dobj)<let>; second person imperatives against the group containing absence-of features like <PRP>(subj).

This isn't an isolated case: for instance, there are several forms of future tense in English, with diverse syntactic constructions, e.g. *I'm going to apply* vs *I'll apply*.

This is a clustering problem, and an established means of addressing it is latent Dirichlet allocation (LDA) [2]. In LDA terminology, our set of examples represents our *corpus*; individual examples are *documents*; features are *words*; our topics are our two types of imperative sentence.

8.4 Further Feature Types

During evaluation, it became evident that the current types of feature are insufficiently expressive to reliably capture some grammatical phenomena. Here I cover a few extensions that might improve accuracy at the expense of execution time and memory requirements.

Token n-gram Sequences

At present, the only relationships between words we consider are GRs, as provided by C&C. There are several reasons to think that building n-gram features across tokens might complement GRs. First, word collocations occur frequently in English [8], and in fixed phrases like *at the very least*, the exact grammatical relationships become less important. Second, the word sequence is given to us, whereas we must eke out GRs probabilistically. Clark and Curran report a headline F_1 score for GRs of 81.1% [10]; even if we add tags to our tokens, accuracy is considerably higher: the reported F_1 of C&C's tagger is 96.6% [12]. As such, sequences are both reliable and relevant to some learning tasks. Using POS tags would allow for 'wildcards': for instance, the 'as ... as' task used in evaluation might use a 'as `JJ` as' sequence feature, with an arbitrary adjective in the middle slot. Using such features might mitigate the current issue whereby a single incorrectly-parsed example sentence can completely throw off commonality analysis.

Other Extensions

Briefly, an assortment of other feature extension ideas:

1. *Occurrence counts*: a ditransitive learning task, without reference to whether the objects are indirect or direct is currently impossible. Given one sentence featuring a verb with `dobj` and `obj2` GRs, and another with `iobj` and `dobj`, our commonality could be that both sentences have two instances of `obj`.
2. *Per-level configurations*: cases like 'verb + that-clause' might benefit from a `<that>s(ccomp)<|VBP>`: lemma at level 1, GR type at level 2, POS at level 3. At present, if a twig captures the POS and supertag for one token, it must capture the POS and supertag of the other tokens.
3. *Null leaves*: we have null tokens and GRs as the root of our dependency structures (see §6.1.2); why not do so at the leaves too, to distinguish unmodified tokens? E.g. `(null)<NN>` is noun with no modifiers or determiners.

8.5 Modifying Corpus Sentences

We see in the evaluation section that for many learning tasks, the shortest 8,688 sentences prove insufficient. Increasing the maximum sentence length is problematic, given the link between sentence length and perceived complexity (see §6.3.1). As an alternative, we could address our data sparsity issue by combining the two approaches discussed in §3: selecting sentences from a corpus, and modifying sentences. During this project, only the former approach was explored to any real satisfaction, but a combination of the two might bear fruit. We could take our top-scoring, but unsatisfactory, corpus selections and mutate them to tick off more of our commonality criteria. Divergent high-scoring sentences could be combined, with the embedded clause of one being inserted into the other, with the result being evaluated for language fluency using a language model — echoing the approach used in statistical machine translation [7].

9 Conclusions

So: has our approach to applying language modelling tools and resources to language learning borne fruit? The evaluation certainly shows promise, though evidently more work and, crucially, more data is needed to pass the point where the application saves enough time through reliably identifying commonality in the vast majority of cases, and finds enough high-quality examples, that it becomes significantly less bother than writing long lists of examples. Reduced smartphone resource usage and processing time is also imperative. There are two more fundamental concerns: the strong dependency on the parser providing reliable parses; and finding a subset of CCGbank's sentences copious enough to ensure plenty of examples for most grammar points, without degrading peak sentence complexity unacceptably. This last point is important: if presented with four thirty-word sentences and asked which exhibits the past continuous, the user is likely to find many verbs in each of these sentences. Despite these reservations, this work represents an interesting new application for syntactic parsing.

Appendices

A Extended Penn-Treebank Tagset

Tag	Description	Supertypes
JJ	adjective	ADJECTIVE
JJR	comparative adjective	ADJECTIVE, COMPARATIVE
JJS	superlative adjective	ADJECTIVE, SUPERLATIVE
WRB	wh-adverb	ADVERB
RB	adverb	ADVERB
RBR	comparative adverb	ADVERB, COMPARATIVE
RBS	superlative adverb	ADVERB, SUPERLATIVE
CC	coordinating conjunction	CONJUNCTION
IN	preposition, or subordinating conjunction	CONJUNCTION, PREPOSITION
DT	determiner	DETERMINER
PDT	predeterminer	DETERMINER
WDT	wh-determiner	DETERMINER
NN	singular or mass noun	NOUN_SINGULAR, NOUN_COMMON
NNP	singular proper noun	NOUN_SINGULAR, NOUN_PROPER
NNPS	plural proper noun	NOUN_PLURAL, NOUN_PROPER
NNS	plural common noun	NOUN_PLURAL, NOUN_COMMON
CD	cardinal number	NUMERIC
POS	possessive ending	POSSESSIVE
PRP	personal pronoun	PRONOUN
PRP$	possessive pronoun	PRONOUN, POSSESSIVE
WP	wh-pronoun	PRONOUN
WP$	possessive wh-pronoun	PRONOUN, POSSESSIVE
EX	existential *there*	PRONOUN
LS	list item marker	PUNCTUATION, NUMERIC

Extended Penn-Treebank Tagset, Continued

Tag	Description	Supertypes
FW	foreign word	TOKEN
RP	particle	TOKEN
TO	*to*	TOKEN
UH	interjection	TOKEN
SYM	symbol	SYMBOL
MD	modal verb	VERB
VB	base form verb	VERB
VBD	past tense verb	VERB_PAST
VBG	gerund verb	VERB
VBN	past participle verb	VERB_PAST
VBZ	3rd person singular present verb	VERB_PRESENT
VBP	non-3rd person singular present verb	VERB_PRESENT
AS*	*as* (underspecified re preposition vs adverb)	IN, ADVERB
LS*	list item marker	PUNCTUATION
LQU*	left quote	PUNCTUATION
RQU*	right quote	PUNCTUATION
LCB*	left curly bracket	PUNCTUATION
LRB*	left round bracket	PUNCTUATION
RRB*	right round bracket	PUNCTUATION
, *	comma	PUNCTUATION
; *	semi-colon	PUNCTUATION
: *	assorted non-sentence-terminal punctuation	PUNCTUATION
. *	assorted sentence-terminal punctuation	PUNCTUATION
$*	currency	SYMBOL
#*	hash symbol	SYMBOL

Note: tags marked * are undocumented additions to the Penn-Treebank tagset, observed in CCGbank and C&C output. Descriptions thereof are based on observed occurrences.

Extended Penn-Treebank Tagset, Continued

Tag	Description	Supertypes	Weight
TOKEN	token		0
ARGUMENT	argument	TOKEN	0.3
NOUN	noun	ARGUMENT	0.45
NOUN_COMMON	common noun	NOUN	0.55
NOUN_PROPER	proper noun	NOUN	0.55
NOUN_SINGULAR	singular noun	NOUN	0.55
NOUN_PLURAL	plural noun	NOUN	0.55
COMPARATIVE	comparative	TOKEN	0.5
CONJUNCTION	conjunction	TOKEN	0.5
POSSESSIVE	possessive	TOKEN	0.5
SUPERLATIVE	superlative	TOKEN	0.5
ADJECTIVE	adjective	TOKEN	0.7
ADVERB	adverb	TOKEN	0.7
DETERMINER	determiner	TOKEN	0.7
NUMERIC	numeric	TOKEN	0.7
PREPOSITION	preposition	TOKEN	0.7
PRONOUN	pronoun	ARGUMENT	0.7
PUNCTUATION	punctuation	TOKEN	0.7
SYMBOL	symbol	TOKEN	0.7
VERB	verb	TOKEN	0.5
VERB_PRESENT	present tense verb	VERB	0.8
VERB_PAST	past tense verb	VERB	0.6

B Test Data

Input sentences used as examples for each test case during evaluation. Sentences lifted from EGT except where asterisked. Each corresponds directly to a stated section in EGT; or where a precision is given in brackets, to a subpoint.

In addition, for tests where recall was calculated, the `grep` commands used to produce shortlists of potential matching sentences in the corpus are given below. `detok.txt` is detokenised and doesn't contain tags; `tok.txt` is tokenised and does contain tags. The output of these lists are emphatically *shortlists*, which were manually reviewed to weed out false positives. `grep`'s `-i` flag disables case-sensitivity.

Adjective phrases with verbs

EGT reference: 12b.

1. I felt sad.
2. This soup smells really wonderful.
3. She thought the room was very strange.

Passive

EGT reference: 257.

1. This book was published by Cambridge University Press.
2. These houses were designed in the 1880s by Edward Barnes.
3. The show is watched by five million people every week.

Present (continuous)

EGT reference: 277.

1. She is sleeping.
2. I am working.
3. I'm cooking now so it'll be ready in about half an hour.

Verb + that-clause

EGT reference: 345a.

Recall shortlist: `grep -i "|VB\[^ \]\( [^ ]\+\)\{0,5\} that|IN" tok.txt`

1. They said that four million workers stayed at home to protest against the tax.
2. The survey indicated that 28 per cent would prefer to buy a house through a building society than through a bank.
3. He knew that something bad had happened.

Punctuation: commas (as list separators)

EGT reference: 293c.

1. We took bread, cheese, and fruit with us.
2. They travelled through Bulgaria, Slovakia, the Czech Republic and Poland.
3. Cats, dogs and humans are all mammals.*

About as a preposition

EGT reference: 2a.

1. Do you know anything about cricket?
2. I'm very worried about my brother.
3. He wrote a book about the Spanish Civil War.

Reported speech (direct)

EGT reference: 313.

1. Barbara said, 'I didn't realise it was midnight.'
2. 'I'm sorry,' said Mark.
3. 'I will love you forever,' he wrote, and then posted the note through Alice's door.

Only as an adverb

EGT reference: 248b.

Recall shortlist: `grep -i "only|RB" tok.txt`

1. Only a few hundred houses survived the hurricane without any damage.
2. This phone is only available in Japan.
3. Only Jason knows where the key is kept.

Up as a particle

EGT reference: 356e.

Recall shortlist: `grep -i "up|RP" tok.txt`

1. He was brought up by his grandmother.
2. Don't give up.
3. What time did you wake up this morning?

Noun + that-clause

EGT reference: 345c.

Recall shortlist: `grep -i "|NN|\[^ \]\( [^ ]\+\)\{0,5\} that|IN" tok.txt`

1. Dutch police are investigating the possibility that a bomb was planted on the jet.
2. He holds on to the belief that he'll walk again.*
3. Everyone knows about the argument that started it all.*

Though meaning 'however'

EGT reference: 32c.

Recall shortlist: `grep -i ", though" detok.txt`

1. I don't mind, though.
2. It's nice, though.
3. Susan was lucky, though.*

As ... as

EGT reference: 40.

Recall shortlist: `grep -i " as .* as " detok.txt`

1. The world's biggest bull is as big as a large elephant.
2. The weather this year is as bad as last year.
3. You have to unwrap it as carefully as you can.

Adjuncts

EGT reference: 15.

1. They waited outside for ages.
2. I kept a copy of the letter in my desk.
3. She quickly realised her mistake.

Wh-cleft

EGT reference: 72b.

1. What they like is smoked salmon.
2. What we need to do is get new batteries for it.
3. What I'd suggest is a good night's sleep.*

Plenty of as a quantifier

EGT reference: Extra149b.

Recall shortlist: `grep "plenty of " detok.txt`

1. Don't worry there are plenty of options.
2. Well, we've got plenty of rice.
3. Plenty of people have dropped out of school early and have still been very successful in their careers.

Adjective + that-clause

EGT reference: 345b.

Recall shortlist: `grep -i "|JJ|\[^ \]\( [^ ]\+\)\{0,5\} that|IN" tok.txt`

1. It's important that we look at the problem in more detail.
2. I'm sure that you'll know a lot of people there.
3. They were afraid that we were going to be late.

Either ... or

EGT reference: 115.

Recall shortlist: `grep -i "either .* or" detok.txt`

1. Either I drive to the airport or I get a taxi.
2. Either the African Union finds a solution or the conflict will drag out indefinitely.*
3. Either the cows are outside or the sound is a tractor.*

Future (perfect)

EGT reference: 134.

1. Do you think the film will have started?
2. A few goats will have escaped by now.*
3. She'll have gone by the time you arrive.*

Do as an auxiliary verb (emphatic forms)

EGT reference: 105d.

1. She does look so tired!
2. I did recognise your mum.
3. We do talk about it a lot.*

Do as an auxiliary verb (question tags)

EGT reference: 105d.

1. You work with Peter, don't you?
2. She plays the piano, doesn't she?
3. They arrived late, did they?

Future (continuous)

EGT reference: 134.

1. I'll be running ten kilometres a day for the next two weeks to get ready for the marathon.
2. He'll be thinking about you while you're gone.*
3. Becky will be dreaming about this.*

Still as an adjective

EGT reference: 331b.

1. Keep your head still.
2. It was a still, calm evening.
3. The dog lay still in the corner.*

All as a pronoun

EGT reference: 27f.

1. All were happy with the outcome.
2. All will be revealed to the public in 25 years' time.
3. All that we had been told turned out to be untrue.

Possessives with *of*

EGT reference: 272b.

1. A friend of mine told me that all of the tickets have already sold out.
2. He's gone to pick up a cousin of his at the station.
3. Is Linda McGrath a close friend of yours?

As: simultaneous changes

EGT reference: 39d.

1. As the show increases in popularity, more and more tickets are sold daily.

2. As you get older, moving house gets harder.
3. As populations migrate, cultural practices are sometimes lost.*

Now as a discourse marker

EGT reference: Extra133b.

Recall shortlist: `grep -i " \?now, " detok.txt`

1. Now, before we start the actual meeting proper, I've invited Carol to come along and tell you about our recycling project.
2. Now, I don't want anyone to call out the answers.
3. Now, before doing that it's important to think through the consequences.*

It-cleft

EGT reference: 72a.

1. It was Nina's car that got broken into!
2. No, it was your sister that I met!
3. It was my husband that you spoke to on the phone.

Apposition

EGT reference: 37.

1. Timothy, their youngest child, is very musical.
2. The living room, the biggest room in the house, looks out on to a beautiful garden.
3. Edinburgh, Scotland's capital city, has a population of around 450,000.

No matter

EGT reference: Extra112c.

Recall shortlist: `grep -i "no matter" detok.txt`

1. No matter how many you send him, he never answers emails.
2. No matter where she looked, she could not find the missing paper.
3. No matter what I wear, I always feel dull and old-fashioned.

References

[1] Srinivas Bangalore and Aravind K. Joshi. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265, 1999.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *The Journal of Machine Learning*, 3:993–1022, March 2003.

[3] Branimir Boguraev and Ted Briscoe. Large lexicons for natural language processing: utilising the grammar coding system of Longman Dictionary of Contemporary English. *Computational Linguistics*, 13(3-4):203–218, 1987.

[4] Ted Briscoe. An introduction to tag sequence grammars and the RASP system parser. Technical Report 662, University of Cambridge, March 2006.

[5] Ted Briscoe and John Carroll. Automatic extraction of subcategorization from corpora. In *Proceedings of the fifth conference on Applied natural language processing*, pages 356–363. Association for Computational Linguistics, 1997.

[6] Ted Briscoe, John Carroll, and Rebecca Watson. The second release of the RASP system. In *In Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 77–80, 2006.

[7] Peter F. Brown, John Cocke, Stephen A.D. Pietra, Vincent J.D. Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[8] Ronald Carter, Michael McCarthy, Geraldine Mark, and Anne O'Keeffe. *English Grammar Today*. Cambridge University Press, 2011.

[9] Eugene Charniak. *Statistical language learning*. MIT Press, 1993.

[10] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4), 2007.

[11] Martin J. Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191. Association for Computational Linguistics, 1996.

[12] James R. Curran and Stephen Clark. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 91–98. Association for Computational Linguistics, 2003.

[13] James R. Curran and Stephen Clark. Language independent NER using a maximum entropy tagger. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 164–167. Association for Computational Linguistics, 2003.

[14] Hoa Trang Dang, Karin Kipper, Martha Palmer, and Joseph Rosenzweig. Investigating regular sense extensions based on intersective Levin classes. In *Proceedings of the 17th international conference on Computational Linguistics, Volume 1*, pages 293–299. Association for Computational Linguistics, 1998.

[15] Rudolph Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221–233, 1948.

[16] Albert Gatt and Ehud Reiter. SimpleNLG: a realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, pages 90–93, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[17] Ralph Grishman, Catherine Macleod, and Adam Meyers. COMLEX syntax: Building a computational lexicon. In *Proceedings of the 15th conference on computational linguistics, volume 1*, pages 268–272. Association for Computational Linguistics, 1994.

[18] Julia Hockenmaier and Mark Steedman. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

[19] Andy Hopper and Andrew Rice. Computing for the future of the planet. *Philosophical Transactions of the Royal Society A*, 366:3685–3697, October 2008.

[20] Daniel Jurafsky and James H. Martin. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2nd edition*. MIT Press, 2008.

[21] Adam Kilgarriff and Joseph Rosenzweig. Framework and results for English SENSEVAL. *Computers and the Humanities*, 34:15–48, 2000.

[22] Tracy H. King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. The PARC 700 dependency bank. In *Proceedings of the EACL03: 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 1–8, 2003.

[23] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[24] Anna Korhonen. Assigning verbs to semantic classes via WordNet. In *COLING-02 on SEMANET: building and using semantic networks-Volume 11*, pages 1–7. Association for Computational Linguistics, 2002.

[25] Beth Levin. *English Verb Classes and Alternations*. University of Chicago Press, 1993.

[26] Mitchell P. Marcus, Mary A. Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the Penn treebank. *Comput. Linguist.*, 19:313–330, June 1993.

[27] George A. Miller. WordNet: a lexical database for English. *Commun. ACM*, 38:39–41, November 1995.

[28] Neil Naiman. *The good language learner*, volume 4. Multilingual Matters Ltd, 1978.

[29] Paul Nation. *Learning vocabulary in another language*. Cambridge Univ Pr, 2001.

[30] Judita Preiss, Ted Briscoe, and Anna Korhonen. A system for large-scale acquisition of verbal, nominal and adjectival subcategorization frames from corpora. In *Annual Meeting of the Association For Computational Linguistics*, volume 45, page 912, 2007.

[31] Geoffrey K. Pullum and Barbara C. Scholz. Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, 18(1-2):9–50, 2002.

[32] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proc. Empirical Methods on Natural Language Processing, New Brunswick, New Jersey*. Association for Computational Linguistics, 1996.

[33] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175, 1999.

[34] Andrew Rice, Paula Buttery, Idris A. Rai, and Alastair Beresford. Language learning on a next-generation service platform for Africa. *Africa Perspective on the Role of Mobile Technologies in Fostering Social and Economic Development*, April 2009.

[35] Jeffery M. Siskind. A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, 61(1-2):39–91, 1996.

[36] Harry L. Somers. On the validity of the complement-adjunct distinction in valency grammar. *Linguistics*, 22(4):507–530, 1984.

[37] Mark Steedman and Jason Baldridge. Combinatory categorial grammar. *Nontransformational Syntax: A Guide to Current Models. Blackwell, Oxford*, 2009.

[38] Lin Sun and Anna Korhonen. Improving verb clustering with automatically acquired selectional preferences. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 638–647. Association for Computational Linguistics, 2009.

[39] David Talbot and Miles Osborne. Smoothed Bloom filter language models: Tera-scale language models on the cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 468–476, 2007.

[40] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.