Natural Language Engineering **1** (1): 000–000. Printed in the United Kingdom

A Natural Language System for Retrieval of Captioned Images

David Elworthy, Tony Rose, Amanda Clare and Aaron Kotcheff
Canon Research Centre Europe, 1 Occam Court, Occam Road, Guildford GU2 5YJ, UK

(*Received 13 August 2001*)

Abstract

NB! This version does not include the final proof corrections.

ANVIL is an information retrieval system using natural language processing techniques, intended for retrieval of captioned images. It extracts dependency structures from the image captions and user queries, and then applies a high accuracy matching algorithm which recursively explores the dependency structures to determine their similarity. A further algorithm allows additional contextual information to be extracted following a successful match, with the intention of helping users understand and organise the retrieval results. ANVIL was developed to high engineering standards, and as well as looking at the research aspects of the system, we also look at some of the design and development issues. English and Japanese versions of the system have been developed.

1 Introduction

Information retrieval (IR) systems present a challenging area for natural language processing. Information retrieval is, in most cases, concerned with the process of matching a user's statement of their information needs with a collection of documents. To many researchers entering the field it seems intuitive that natural language processing should help in this process, as it means we can abstract away from exactly how the queries and documents are expressed to a deeper representation of the information they contain. Unfortunately, experiments have not usually borne this out. A classic example is Fagan (1987), in which a simple retrieval system based on weighted keywords was compared with one in which pairs of related words were used. In one trial, the combinations of words were selected according to syntactic relationships between the words, and in a second, they were found purely by looking at co-occurrence statistics. The results show that both ways of finding compound terms did better than simple keywords, but the trial based on NLP did no better than the statistical one. While some criticisms of Fagan's work can be made – he made no attempt at semantic analysis, for example – there have been similar findings from other authors, for example Smeaton (1999) and the research summarised in Sparck-Jones (1999). If NLP does not provide improvements in terms of accuracy, the extra processing time and engineering effort to apply it in IR seem to be unjustified, and at present there appears to be a widespread scepticism in the IR community about the benefits of using NLP.

Nevertheless, there have been some successes in the field. A notable example is the Intermezzo/PhotoFile system described by Flank (1998). The documents to be retrieved are images which have been annotated with phrasal captions. The captions are much shorter than the documents usually used in IR research. The retrieval technique is based on work by Strzalkowski (1994), in which compound terms are extracted from the query and documents by finding pairs of words standing in certain normalised structural relations, using a fast parser. Flank quotes a precision of around 90%, examining the top-scoring results of 50 queries on a 500,000 image database, where precision is defined as the proportion of the retrieved documents which are relevant, i.e. which the user actually wanted. Another apparently successful system is described by Schwarz (1990), in which a lightweight analysis is performed, and the resulting representations are again converted to a normalised representation before comparing them. Some further related ideas can be found in Grefenstette (1997) where simple dependency structures are used for query expansion.

The ANVIL (Accurate Natural-language Visual Information Locator) system described in this paper is intended to be an accurate and fast natural language information retrieval system, for use with captioned images. As with Flank's work, we had a database of images with phrasal captions, ranging from 1 to around 20 words. Some example captions appear in (1)–(3).

(1) golden columns reaching up towards a deep blue sky.

(2) Big Ben, London.

(3) blue surface covered in stippled yellow paint.

Our approach to NLP was to perform a lightweight analysis and compare the resulting structures. Unlike the work of Strzalkowski and others in the field, we do not aim to identify compound indexing terms, but to assign a match score by means of a recursive exploration of the analysed structures. ANVIL uses a novel form of parsing which extracts a dependency structure using finite-state processing techniques. The matching is carried out by an error tolerant algorithm which allows fine tuning of scores, and also identifies constructs which weaken the match, such as negated modifiers. The matcher in addition extracts contextual information which helps users understand and organise the search results. In doing so, we provide functionality which is simply not possible without NL techniques.

A design goal of the system was that the engineering should be of a high standard, so that it could be transferred from the research team to a product development group as soon as possible. This meant that issues to do with robustness and speed of operation had to be addressed from early on in the project. Furthermore, the language specific portions were carefully isolated, and versions of ANVIL for both English and Japanese have now been developed. There is one further aspect of ANVIL which we will not explore in detail here: a visual user interface, so that users can explore the database using their initial query as a starting point, and

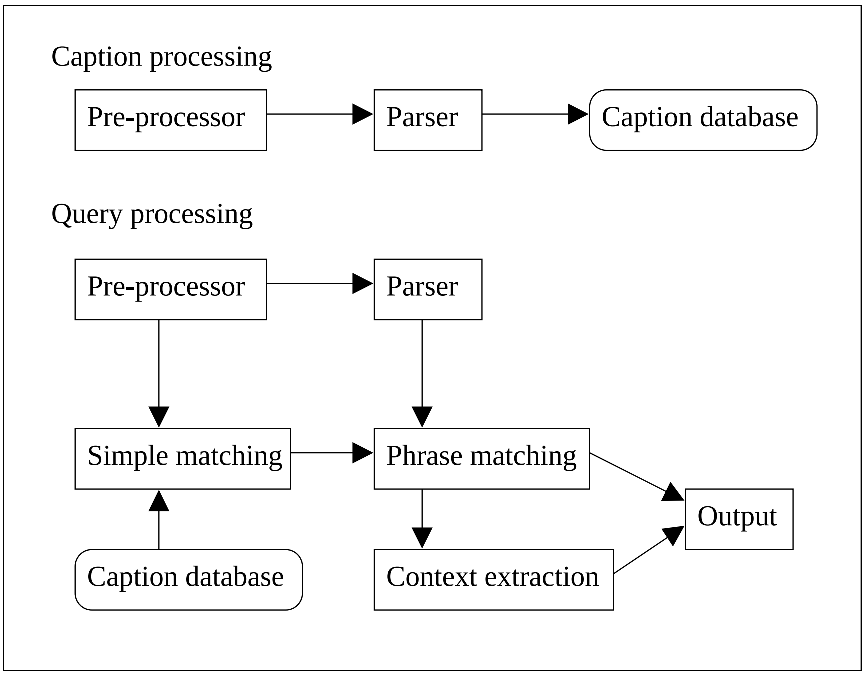

Fig. 1. ANVIL system architecture

then continue by interactive navigation, thus reducing the need to formulate an exact query. The visual user interface is described in Rose *et al.* (2000).

In section 2 of this paper, we describe the overall architecture of the system, and go into some detail about the early stages of processing. The algorithms for parsing and matching are outlined in section 3, together with a technique called context extraction. Context extraction illustrates how natural language analysis can provide better functionality as well as better accuracy than conventional information retrieval approaches. An evaluation of the system appears in section 4, in terms of the conventional IR measures of accuracy, and also of scalability, and language engineering aspects such as adaptation to another language and ease of writing the grammar and rules. The final section presents some suggestions for future work, and attempts to set ANVIL in the context of natural language information retrieval more generally.

2 The structure of ANVIL

The architecture of ANVIL is shown in figure 1. The pre-processor takes a stream of characters and produces one or more analyses consisting of series of tokens (words, roughly) annotated with lexical information. Each analysis is then parsed, resulting in both the phrase bracketing and a set of dependencies between words. When a caption is being added to the database, ANVIL now just stores the analyses and the parse information. In the case of query processing, an initial stage of simple matching is carried out, using standard keyword-based information retrieval weighting

```
Query = 'camera with a lens.'
5 results:
SCORE  CAPTION
1      black SLR camera, with zoom lens, on a white surface.
       * camera: black, SLR, on a white surface
       * lens:   zoom
1      old-style black camera, with protruding lens, on a white surface.
       * camera: black, on a white surface
       * lens:   protruding
0.588  old camera, hip flask, box and album filled with sepia photographs.
       * camera: old
0.5    Canon camera, magnifying lens and fashion magazine on ridged surface.
       * camera: Canon, on a ridged surface
0.1    astronaut floating within space craft, showing the on-board cameras.
       * cameras: on-board
```

Fig. 2. Example ANVIL output

schemes. The simple matching produces a list of candidate captions, which are put through phrase matching. Phrase matching compares the dependency structures of the query and the caption, resulting in a numerical measure of similarity. The scores from simple matching and phrase matching are combined to produce an overall ranking for the retrieval results. Context information about the match is extracted following phrase matching. An example query run appears in figure 2, slightly simplified from the actual ANVIL output. The contexts are shown following each caption.

We will look in more detail at the parsing and matching in section 3. The remainder of this section deals with pre-processing and some general resource and implementation considerations.

2.1 Pre-processing

Pre-processing encapsulates the orthographic and lexical processing of ANVIL. Most of the language-dependent parts of the code are contained in the pre-processor. Figure 3 shows the structure of English version.

The first stage of the English pre-processor is the tokeniser, which splits up the incoming character stream into words. Special patterns, such as dates and numbers are identified, as are multi-word lexical items such as *The White House*. The tokeniser does not attempt to perform a morphological analysis or stemming to begin with. Instead, it draws on a lexicon which contains all inflected forms. The reasoning behind this is that if we had just base and irregular forms in the lexicon, we would still have to first look up on the surface form of the word, in case it was an irregular form (such as *took*) before potentially reducing the word to its base form(s) and looking it up again. The lexicon is large enough that it has to be held in an external database, and access to it therefore relatively slow. Consequently, we reduce the number of accesses to the lexicon database by looking up inflected forms directly, and hence save processing time. There are around 170,000 full forms in

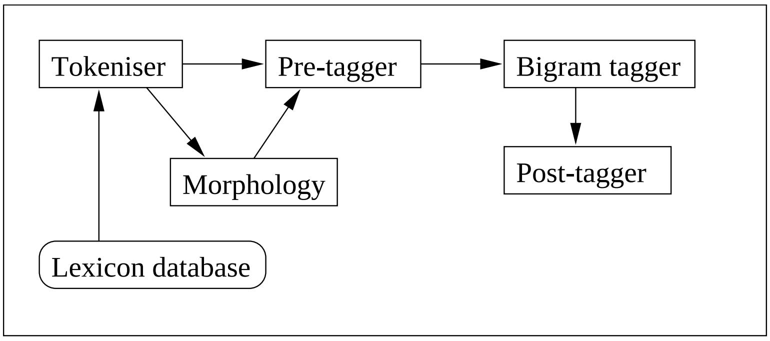

Fig. 3. ANVIL pre-processor component for English

the lexicon. Note that there are alternative strategies for representing the lexicon which do not require separate entries for all the inflected forms. For example, Mohri (1996) describes how a lexicon can be quickly compiled to a finite-state transducer in a compact form. Such techniques would be important for a more highly inflected language than English.

The lexicon stores several fields for each word, of which the two most important are the tag data and the word senses. The tag data consists of part of speech tags for the word with lexical probabilities, for use in a standard bigram tagger. The tags are based on the tagset used for the British National Corpus[1]. The word senses are drawn from the data in WordNet (Fellbaum 1998), and encoded in the form of WordNet "synset identifiers", i.e. integers representing the meaning of the word. Synonyms share the same synset identifier. Each sense has a score which allows us to indicate its relative importance amongst the synsets for the word. For most words, we in fact have no information which tells us this and so the same score is used for all the senses. The exceptions are the very few words which appear in WordNet's SEMCOR semantic concordance; the provision of scores is mainly so the information could be included in future if we had a richer source of information.

The morphological analysis component is invoked only if a word is not found in the lexicon. It examines affixes of the word to try to decide the likely parts of speech in general terms (noun, verb, adjective, adverb or unknown). The lexicon contains special entries to provide default information based on the morphological analyser's decision. A dummy synset identifier is set up for such words.

The result of the tokeniser is a lattice of tokens representing different ways of tokenising the input. For example, if the input is *Kennedy sitting in The White House*, the last three white-space separated words could be a sequence of three tokens or a single token. The pre-tagger component flattens the lattice into separate token streams by taking each possible path through it. In this example, we would get *Kennedy sitting in The White House* (6 tokens) and *Kennedy sitting in The-White-House* (4 tokens). The actual number of cases where there are more than a

[1] A description of the BNC and details of the tagset can be found at `http://info.ox.ac.uk/bnc/`.

few alternative tokenisations is small, and so the number of flattened paths is not excessive. It is also possible to include information in the lexical entry which says that when a multi-word unit has been recognised then the alternative tokenisation as separate words should be discarded, for example for idioms like *in so far as* (though, admittedly, this is unlikely to occur in image captions).

The tagger is a standard bigram HMM tagger (Cutting *et al.* 1992), with an error detection technique of the sort described in Elworthy (1997). The tagger uses the Forward-Backward algorithm to assign a score to every candidate tag, and then normalises the scores across the tags on each word. Tags which have a score greater than a given threshold are retained and the remainder discarded. In most cases, a word will receive a single tag, but in a few cases two or more tags will remain. A common example is certain words ending in *-ing* which can either be a noun or the progressive form of a verb, such as *building*. Following the tagger, we then further flatten out these possibilities in a component called the post-tagger, so that in the resulting analyses, each token has a single tag. Before flattening, the post-tagger also applies a collection of tag modification rules. A tag modification rule lists one more tags and a replacement tag. If a word has all of the given tags, they are replaced by the indicated tag. For example, if *building* had a noun (NN) and gerund (VVG) tag, we would replace them by a composite tag NN*VVG. In many cases, the parser will be able to use its greater information about the syntactic context to implicitly resolve the ambiguity. Alternatively, if the distinction between two tags is not needed in later stages of processing, having been made purely for the purposes of keeping the tagger accurate, it can be dropped later on. An example is the ambiguity between an adverb and a subordinating conjunction seen, often seen on the word *as*. The distinction is made by the tagger because of the tagset it was trained with, but is not needed in later stages of processing. Tag modification rules can also replace a single tag with a different one. Thus, in the English grammar, we do not have separate rules depending on whether a noun is singular or plural, and so the singular and plural noun tags are both replaced by a general noun tag. In cases where the post-tagger does not merge tags, so that a tag ambiguity remains, each tagging is parsed and stored. There is a danger of a combinatorial explosion here if many words were left with multiple tags, but the tag merging rules resolve almost all cases.

2.2 Data sources

The lexicon was derived from two main sources: the British National Corpus, and the list of words used in WordNet. The BNC provided an initial list of words and tags. We excluded low frequency words using a cutoff of 20 occurrences in the 100 million words of the BNC. WordNet contributed the synset data, as well as providing a source of many multi-word units and American English variants. A possible future extension to ANVIL would be to add a gazetteer to these data sources. This has proved to be effective in the system described by Flank (1998).

The tagger was trained using standard techniques using the BNC, providing both lexical probabilities for the tags on each word, and tag-to-tag transition probabilities. To check the accuracy of the training, the tagger was initially trained on about 85% of the BNC. The remainder (section K, roughly 15 million words) was put through the tagger and the output compared against the original tagged text in the BNC, showing an accuracy on the top scoring tag for each word of 96.4% overall, 95.5% on words with more than one tag. Following this validation, the extra 15 million words were incorporated in the training. The BNC text was originally tagged using a semi-automatic process which had an accuracy of around 95%, and so the validation cannot be taken as anything more than suggestive of sufficient training. In general, the results seem adequate for the grammar to construct correct parses.

During querying, there is one final part of pre-processing which we apply, not shown in the system diagram. The WordNet synsets provide a limited form of thesaurus, in that direct synonyms have a synset in common. To provide additional terms, we also use a process of semantic broadening. For each synset, we navigate through the WordNet hierarchy, moving both to more specific terms (hyponyms) and more general ones (hypernyms). That is, the initial collection of synsets is augmented with ones from hyponym and hypernym nodes in the WordNet hierarchy. A semantic distance between terms is computed using a procedure described by Jiang and Conrath (1997). The distance is computed by assigning a measure of information content to each node in the hierarchy, and taking the distance between two nodes as the sum of the differences in information content from each word to the nearest common parent. The information measures were trained using frequencies from the BNC, weighted using WordNet's SEMCOR semantic concordance for the small number of word types which SEMCOR covers, and dividing the frequency uniformly amongst the senses otherwise. Three thresholds are applied, on the distance up and down the hierarchy and the total semantic distance, and this avoids bringing in a huge number of extra synsets, many of which might only have a weak connection to the original term. All synsets for the original query term are used for broadening. If we had a reliable word-sense disambiguation algorithm, or some way of excluding unlikely synsets, this would further reduce the number of synsets which have to be considered.

All of the data sources are computed off-line, and stored either in memory or in a disc-based database, for use at query time.

2.3 Engineering considerations

A major design criterion in ANVIL was to keep query processing as fast as possible. This lies behind the decision to list all inflected forms in the lexicon. Similarly, the captions are analysed and parsed when they are entered into the database, to avoid the need to do this at query-time.

For the implementation language, we adopted C++ with the Standard Template Library (STL). An early prototype used the functional programming language Haskell, together with components written in C for the user interface and database manager. Haskell has a number of good properties as an implementation language. In particular, it is possible to write complex algorithms in it very quickly, it has a powerful type system which helps keep the code robust. Compiled Haskell runs more slowly than C, although selective coding of time critical components in C could probably have overcome this. However, the main reason for switching from Haskell to C++ was our involvement with a product development group, who were concerned about long-term support of the code by programmers who might not have experience of Haskell. In contrast, a reservation about large C++ programs is that they tend to suffer from memory leaks (since there is no garbage collection), and catastrophic program failures when memory gets overwritten (typically showing up as segmentation faults or general protection faults). Furthermore, software development can be slower as C++ does not provide the rich built-in data types of symbolic programming languages. The STL helps to overcome all of these problems, in providing type-safe templates for data-types such as lists, vectors, sets and maps, together with algorithms over these structures. It is still possible to code badly with the STL and to fall back into the usual C-style problems of memory leaks and memory leaks, but the frequency of such problems is certainly less than with "hand-rolled" data structures. We further used the Purify code-checker (from Rational Software) to check for potential memory problems. There were almost no problems with memory leaks or overwriting during the development of the system.

For the database, we initially worked with GNU gdbm, but we later moved to Berkeley DB[2]. Berkeley DB is faster and has higher functionality, and does not suffer from the restrictive licensing terms of GNU software. Microsoft SQL server was also considered, but preliminary experiments suggested that it was too slow for the kind of simple database we were using. The database implementation is hidden behind an abstract interface which assumes only that keys and data are arbitrary blocks of binary data which can be stored, replaced, deleted and enumerated. We use two main database files for the main ANVIL data. One maps from caption identifiers to analysed captions, and the other from synset identifiers to the identifiers of captions using them. The lexicon and semantic broadening data were represented using the same database package; the former maps word forms to lexical data, and the latter maps synset identifiers to sets of identifiers of similar synsets.

ANVIL was developed simultaneously on UNIX workstations, using egcs (g++) as the compiler, and on Windows 95 and NT using Microsoft Visual C++. Moving code between the two platforms was largely (and surprisingly) straightforward.

2.4 Japanese ANVIL

As well as developing an English version of ANVIL, colleagues in other parts of the company wished to develop a Japanese version. For this reason, we were careful to encapsulate the language-specific elements of the system. Essentially, these were the pre-processor, and the grammar, matcher rules and context rules, although not the code for parsing, matching and extracting contexts. About three-quarters of the code was language independent. Unfortunately, we were unable to obtain

[2] GDBM is available from the GNU project, `http://www.gnu.org/`. Berkeley DB is distributed by Sleepycat software, and free for research purposes; see `http://www.sleepycat.com/`.

figures about how long the development of the Japanese version took. The Japanese lexicon and thesaurus were based on work that had been done for other products; the grammar was written specifically for the application.

3 Parsing and matching

The parser and matcher form the heart of ANVIL. The parser uses a fast finite-state technique to extract the dependency structure from captions and queries, indicating what modifies what. In response to a query, an initial selection of captions is made using a simple, keyword-based matching algorithm. The phrase matcher then compares the captions against the query taking the dependency structure into account. Finally, additional information about the context of the match is extracted to help users to organise the results. We will describe each of these four aspects in the following sections.

3.1 The parser

ANVIL uses a cascaded finite-state parser, in which the grammar is divided into a number of levels, each consisting of a number of grammar rules written as regular expressions. The input to the parser is a sequence of tokens, each annotated with its part of speech. The regular expressions are written in terms of lexical parts of speech, phrasal tags and specific words. Similar cascaded designs have been used before, such as Abney's SCOL parser (Abney 1996). Like SCOL, the ANVIL parser follows a left-to-right, longest match strategy.

A grammar rule specifies a phrasal tag which will be constructed if the body of the rule matches. This gives us the bracketing, but for phrase matching we will also need the modification relationships between words. To accomplish this, we augment the rules with a system of variables. A variable can be unindexed, in which case it holds a list of words, or indexed, in which case it acts as a mapping from words to lists of words. To place words in variables, the elements of phrase rules are annotated with actions, as in (4).

(4) `np -> det? adj:mod{head}* noun:head`

In this example, for recognising a noun phrase, the noun is stored in the unindexed variable *head*, and the adjectives are stored in the *mod* variable, indexed on whatever was put in the head. Thus, the *mod* variable effectively establishes a dependency of the adjectives on the noun. Some examples appear in figure 7. Note that the scope of variables is the rule in which they occur, and in the formalism actually used in the system they are “declared” on the left hand side of the rule, thus:

(5) `np(head,mod) -> det? adj:mod{head}* noun:head`

We omit the declarations here so as the make the rules marginally clearer.

Variables can also be copied to other variables, as in the following example which forms a noun phrase from an existing noun phrase with a prepositional phrase modifier:

(6) `np -> np:(mod=mod,head=head) pp:(prep{head}=p,∧)`

Here, `mod=mod` indicates that all `mod` variables in the input NP are copied to the output NP. The ∧ on the PP is shorthand meaning that all variables of the PP not otherwise mentioned are copied to the output phrase. As a result of this rule, the head of the NP will have a *prep* link to the preposition of the PP (stored in *p* by a previous rule), and any other links within the PP will be copied to the output NP.

The finite-state rules of the grammar are compiled (off-line) into finite-state machines and determinised. There are standard techniques for doing this, for example as described in Aho, Sethi and Ullman (1986). However, the presence of variable setting actions means that we are dealing with something more like a finite-state transducer than a finite-state recogniser, and these algorithms cannot be used without modification. To see this, consider a variant of the noun phrase rule above in which the modifiers are nouns rather than adjectives, and the indexing has been dropped in the interests of simplicity:

(7) `np -> det? noun:mod* noun:head`

When the finite-state machine for this rule is being traversed, the parser cannot decide whether to store nouns in the *mod* variable or the *head* variable, because it does not know whether it had reached the end of the NP or not. The standard determinisation algorithms are unable to resolve this; in effect, a compile-time lookahead is needed. This problem can be solved using an algorithm described by Roche and Schabes (1995), in which output actions are deferred until a decision can be made about which one to execute. In our example, the *mod* and *head* actions are deferred onto the final state, with an indication of the relative position of the transition that caused the action. The determinisation process is illustrated in figure 4, in which *mod(-1)* means set the *mod* variable from the word consumed in the previous transition. Roche and Schabes algorithm always finds a determinisation if one is possible, although it will fail to terminate if no determinisation is possible. We trap this by imposing a limit on how many times the algorithm will iterate before reporting an error.

Only a single parse is obtained for each analysis. The justification for doing this is to keep the processing efficient, but it is potentially problematic, as it does not allow us to capture genuine structural ambiguities such as prepositional phrase attachment. However, this can often be overcome in the matching rules. We can also represent some ambiguities in the dependency structure even when they are not realised in the phrase structure. As an example of this, suppose we want to obtain alternative attachments of prepositional phrases. Taking an example such as *man in the park on a hill*, where there is a PP attachment ambiguity, we make a *prep* link from *man* to *on*, and also from *park* to *on*. A sample grammar for doing this is show in figure 5. The square bracket notation `pcomp[p]` means store using the `p` variable of the output phrase type, as opposed to something like `pmod{head}` which means use the `head` variable of the input category. The first PP rule builds simple prepositional phrases, which can then either be built into larger prepositional phrases or combined with a noun phrase. The variable *phead* provides access to the

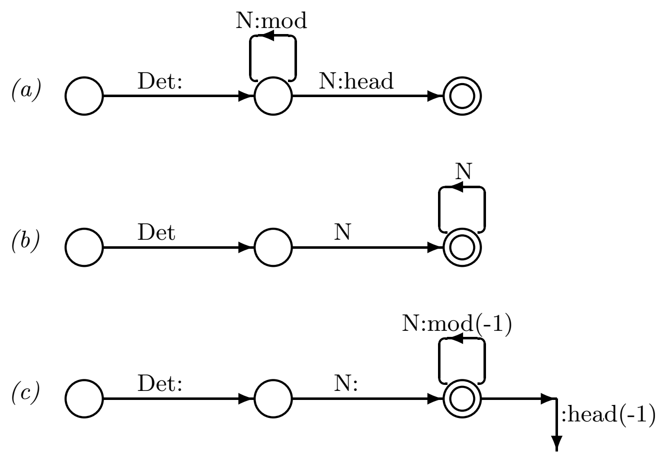

Fig. 4. Determinisation of `Det (N:mod)* N:head`: (a) Compiled FSM; (b) Simple determinisation; (c) Determinisation with deferred transitions

```
np -> det? noun:head
pp -> prep:p np:(pcomp[p]=head, phead=head)
pp -> pp:^ pp:(^,pmod{phead}=p)
np -> np:^ (pp:pmod[head]=p,pcomp=pcomp,pmod=pmod)*
```

Fig. 5. Grammar for packing

head of the NP within the PP, even though the preposition is taken as the overall head of the PP, while *pcomp* provides access via the preposition. In the second PP rule, the existing variables are copied up (`PP:^`), including *phead*, and a *pmod* link from this *phead* is then made to the preposition of the second PP (`pmod{phead}=p`), giving the linkage described above.

To make this clearer, and also to illustrate the rule formalism in general, we will show this grammar in operation on an example, parsing the phrase *man in the park on a hill*.

```
Words:           man  in   the park on   a   hill
Parts of speech: noun prep det noun prep det noun
First NP rule:   [NP man],      head=man
                 [NP the park], head=park
                 [NP a hill],   head=hill
First PP rule:   [PP in [NP the park]],
                  p=in, pcomp[in]=park, phead=park
                 [PP on [NP a hill]],
                  p=on, pcomp[on]=hill, phead=hill
Second PP rule:  [PP [PP in [NP the park]] [PP on [NP a hill]]],
                  p={in,on},
```

```
                   pcomp[in]=park, pcomp[on]=hill,
                   phead={park,hill},
                   pmod[park]=on
Second NP rule:   [NP [NP man] [PP [PP in [NP the park]]
                               [PP on [NP a hill]]]],
                   head=man,
                   pmod[man]={in,on}
                   pcomp[in]=park, pcomp[on]=hill,
                   pmod[park]=on
```

The bracketing is clearly rather odd in this, an artifact of the finite-state grammar. The important thing is that we have made the right dependency links, with two links from *man* and two links to *on*.

The English grammar has 61 rules, although this figure should be treated with caution as many of the rules contain disjunctions over a number of regular expressions. There are certainly deficiencies in the grammar as it stands. One notable omission is a good treatment of coordination, something which is quite difficult in a dependency grammar. Coordination is currently treated in a rather ad-hoc way, with rules for conjoining at the lexical level (nouns and adjectives), as well as conjunction of noun groups, of noun phrases and so on. There is no attempt to resolve the scope of the coordination, which could cause problems in phrases such as *a red and blue ball and bat*, or to make the relation between co-ordination and certain prepositions such as *with*. The latter shows up in the example used in section 2, *camera with a lens* which could be read as either "a camera with a lens attached to it" or "a camera and a lens". Extending the coverage of the grammar for cases like this would certainly be a good thing. One point worth mentioning is that since we also had some control over the authoring of the image captions, we could vet them (if not the queries) for cases like this.

One problem we encountered with the Japanese grammar is that it has many more rules than the English one, and consequently took up a lot of memory. Much of this was due to near duplication in the rules. Most of the duplication consisted of rules which were identical other than some agreement condition between constituents of a phrase (something which we had ignored in the English grammar). To solve this, we introduced a system of features into later versions of the grammar, so that the common aspects could be encoded into the rules with the feature system capturing the differences. Consistency tests on features then enforce agreement. The resulting grammars were much more compact, as well as being more readable.

3.2 Simple matching

Simple matching uses more or less standard techniques from information retrieval to find an initial set of captions. The two aspects of it which are interesting are what it treats as a term, and the ways of deciding the match score between a caption and a query.

In most information retrieval systems, the terms are words, possibly stemmed, and with stop words removed. As we described in the introduction, compound terms may also be used, and extra terms can be added from a thesaurus. In ANVIL, the terms are the synset identifiers of the words in the caption and the query, derived from the lexical entries and modified by semantic broadening. In effect, this means we get the effect of a thesaurus directly, since a synset identifier stands for a collection of words which are synonyms, and we also get terms which are related to those of the original query but are not direct synonyms.

To rate the match between a query and a caption, simple matching can use various scoring methods: vector cosine, two variants of the run 6 scoring used in Smeaton and Quigley (1996) which combines synset-synset similarities, and a simplified version of the probabilistic BM25 formula (Robertson *et al.* 1994). The different measures result in similar quality of results, although Smeaton and Quigley's technique gives slightly better accuracy and runs faster. The idea behind it is to examine the synsets in common between each query word and each caption word, and to select the one with the highest similarity to provide the overall similarity between the words. The similarities are then summed and normalised by the number of words.

One subtle point arises from the fact that there are more terms (synsets) than words from the query. The number of synsets is limited by the thresholds used in semantic broadening, and so avoid the explosion in the number of senses which can occur if WordNet is used blindly. We also have to be careful to assign each synset a "term frequency" less than one, such that the total of these values across all the synsets for a word is equal to one. If we did not do this, then the score in simple matching would be dominated by words with the most synsets. For example, if using vector cosine matching, a word with ten synsets would contribute to the score ten times more than a word with one synset. To correct this, we would ensure that each synset of the first word is treated as if it has a term frequency of 0.1. This information is encoded in the lexicon, so we can adjust the weights on different synsets if we have a suitable source of information, as already described, and modified by the distance measure of the semantic broadening.

Simple matching starts by making a set of all the synset identifiers used in the query, and then looks these up in an index onto the database, to obtain a list of possible captions. The length of the list can be limited to a fixed value, which is a useful way of controlling the total retrieval time, at the risk of missing some captions. Each caption is then tested using the selected scoring method. A threshold can be used to discard match results with a low score.

3.3 Phrase matching

Phrase matching compares the dependency structures of a query and a caption. It starts with the most important words, such as the head of the query, and progresses down through the dependency structure, either matching words or skipping over them. The process is guided by rules specifying which structures are to be treated as similar. Each rule has a score associated with it, to express different degrees of similarity. Elements of the dependency structure can also be matched against fixed

```
head_rule
{
    head = head  1.0 => mod_rule 0.7;
    head = mod[] 0.5 => mod_rule 0.7;
    mod[]  ?     0.3 => Done 1.0;
}

mod_rule
{
    mod[] = mod[] 1.0 => mod_rule 1.0;

    phead:prep[] = mod[] 1.0 => mod_rule 1.0;
    mod[] = phead:prep[] 1.0 => mod_rule 1.0;
    phead:prep[] = phead:prep[] 1.0 => mod_rule 1.0;

    vhead:cop:rel[] = vhead:cop:rel[] 1.0 => mod_rule 1.0;
    vhead:cop:rel[] = mod[] 1.0 => mod_rule 1.0;
    mod[] = vhead:cop:rel[] 1.0 => mod_rule 1.0;

    amod[] = amod[]  1.0 => Done 1.0;
    'not' = amod[]  0.0 => Done 0.0;
    amod[] = 'not' 0.0 => Done 0.0;
}
```

Fig. 6. Example matching rules

expressions, to detect special cases such as negation, and special rules can be used to penalise missing or superfluous items. To explain how phrase matching works, we start by describing the form that the rules take, and then move on to the scoring scheme.

3.3.1 The rule format

A small set of example rules appears in figure 6. The left and right hand sides of a comparison (i.e. the parts separated by the equals sign) express paths through the dependency structure. Processing starts with the head rule, which specifies that the (unindexed) head variables can be compared, and also that a head can be compared against any modifier word. We will return to the third part of this rule in a moment. When the query and caption have words in common in the indicated variables, the rule to the right of the `=>` sign, called the continuation, is invoked. Within the continuation rule, the symbol `[]` means each word which matched at the previous level is inserted, thus accessing the words linked to it via the indicated variable. (In the top level rules, `[]` just indicates that an indexed variable is to be used, indexing on all words in its domain.) `phead:prep[]` is shorthand for following a *prep* link followed by a *phead* one. In this example, after a match on the head, `phead:prep[]` would find the main noun of a prepositional phrase linked via a preposition from the head. A special continuation *Done* indicates that no further searching is to be carried out down a branch. It can also have a special effect on the score, as we will see below.

Comparisons which contain a `?` symbol are called mopping-up rules. They specify that words from query or caption should be treated as if they had matched, even if they had no counterpart in the other phrase. Mopping-up rules are intended for skipping words which do not have an effect on the match, and (more importantly) for catching words from the query which did not have a counterpart in the caption. The example above,

(8) `mod[] ? 0.3 => Done 1.0;`

allows unmatched modifiers from the query to be mopped up and an appropriate score set up for them. Rules may also contain specific tokens, as in

(9) `'not' = amod[] 0.0 => Done 0.0;`

which allows an `amod` ("adverbial" modifier) to be tested against the literal word *not*.

There are some other forms of rules which are not shown in this example. Firstly, we can specify that if a rule is matching several words simultaneously (for example several modifiers), they must appear in the same linear order in the query and the caption. Secondly, we can require that all of the words on the left hand side must match with words on the right hand side. Finally, it is possible to specify a negated rule which requires no word from the expression on the left may appear in the words from the expression on the right for the comparison to succeed. For example, this allows us to assign a different score when the query *yellow car* is compared with *car* than when it is compared with *green car*. In the first case, a mopping up rule catches *yellow* and assigns one score to it; in the second, a negation rule identifies the mismatch between *yellow* and *green* and assigns a different score[3].

To give an example, consider matching *yellow car* against *car which is yellow*, using the rule set in figure 6. We start with the `head = head` in the *head_rule* group, which causes *car* to match. Next, the *mod_rule* group is invoked as a continuation, with `mod[] = vhead:cop:rel[]` allowing *yellow* from the query, stored in `mod[car]` to match against *yellow* from the caption, using the assignments `rel[car]=which`, `cop[which]=is`, and `vhead[is]=yellow`. Following this there are no words left to match. Note that if we were matching against the caption *car which is not yellow*, there would be a further continuation, back into *mod_rule*, in which the rule `'not' = amod[]` would apply, with an effect described below.

3.3.2 The scoring scheme

The scoring scheme allows us to give a measure of how well a query matched a caption. Each word of the query phrase which participated in a rule is assigned a

[3] The same rule would also block a match between semantically unrelated words such as *yellow* and *old*. This is a weakness of the current scheme which has not been corrected. The authors thank one of the reviewers for pointing this out.

score, representing its degree of match, and a *weight*, representing its importance. Participating in a match means that a query word was found to match against a caption word or a specific token, or that a query word was caught by a mopping up rule, or that it was used in a negated rule. Writing the scores as s_i and the weights as w_i, the overall score of the match is $\sum s_i w_i / \sum s_i$. A further factor called the up-score can modify this, described shortly. Note that the scoring scheme was evolved to capture various cases where there should intuitively be a difference in score, and it should really be viewed as engineering rather than science. A simple set of examples which ought to yield different scores can be seen in matching the query *yellow car* against the possible matching captions *car*, *yellow car*, *green car*, *car which is yellow* and *car which is not yellow*. We will use some of these as examples to illustrate the rules below; see figure 8.

The scores and weights are derived from two factors which appear on the rules, called the t (term) factor and the d (down) factor. The t-factor provides the basic score for words when they are matched by the rule, and the d-factor sets the weight for the continuations. The t-factor appears before the `=>` symbol and the d-factor after the continuation name. For example, in (8), the t-factor is 0.3 and the d-factor is 1.0.

As we progress through continuations, the d-factors are multiplied together, forming the weight for a word. The weight is 1.0 in the initial rule. Thus, in the example, words matched in the `mod_rule` would have a weight of 0.7 (and modifiers of modifiers would have the same weight since the continuations specified within `mod_rule` have a d-factor of 1.0). The word scores are formed from the product of the t-factor of the rule and two further factors. The first is the similarity between the matched words. The similarity scores are derived from the ones used in semantic broadening, so that direct synonyms have a similarity of 1.0, and less directly related terms have a lower score. Thus, for example, matching (one sense of) *car* against *vehicle*, might have a score less than 1.0.

The score is further modified by a factor called the up-score, derived from the special continuation `Done`. When a `Done` continuation is reached, its d-factor is multiplied into the score assigned by the rule which invoked it. Usually the factor will be 1.0, and so the score is unchanged, but in special cases it may be some other value. An example of where this is useful can be found in the rules involving 'not' in figure 6. When a negation is recognised, then we cancel the score on the word which is negated, by using an up-score of 0.

One small detail has been glossed over here. In simple matching, we matched synsets rather than words. In the phrase matching stage, we take a term from the query as matching a term from the caption if it has a synset in common. Some alternatives were tried, such as adjusting the score to reflect the number and importance of the synsets in common between the words. However, it seems to make little difference to the overall results.

To see how scoring works, suppose the query *yellow car* is tested against the captions *yellow car*, *car which is yellow* and *car which is not yellow*. The variables are shown in figure 7, and a trace through the matching process appears in figure 8. In particular, note how the rule shown in 9 causes the previous score assignment

```
                    yellow car
                        head = car
                        mod[car] = yellow

car which is yellow                car which is not yellow
    head = car                         head = car
    rel[car]   = which                 rel[car]     = which
    cop[which] = is                    cop[which]   = is
    vhead[is]  = yellow                vhead[is]    = yellow
                                       amod[yellow] = not
```

Fig. 7. Variables (dependency structures) for example phrases

yellow car + yellow car

Query word	Rule group	Comparison	Score	Weight
car	head_rule	head = head	1.0	1.0
yellow	mod_rule	mod[] = mod[]	1.0	0.7

Overall match score $= (1.0 \times 1.0 + 1.0 \times 0.7)/(1.0 + 0.7) = 1.0$

yellow car + car which is yellow

Query word	Rule group	Comparison	Score	Weight
car	head_rule	head = head	1.0	1.0
yellow	mod_rule	mod[] = vhead:cop:rel[]	1.0	0.7

Overall match score $= (1.0 \times 1.0 + 1.0 \times 0.7)/(1.0 + 0.7) = 1.0$

yellow car + car which is not yellow

Query word	Rule group	Comparison	Score	Weight
car	head_rule	head = head	1.0	1.0
yellow	mod_rule	mod[] = vhead:cop:rel[]	1.0 (initially)	0.7
(none)	mod_rule	'not' = amod[]	0.0	0.0
yellow	mod_rule	mod = vhead:cop:rel[]	0.0 (on up-score)	0.7

Overall match score $= (1.0 \times 1.0 + 0.0 \times 0.7)/(1.0 + 0.7) = 0.59$

Fig. 8. Matching in action

for *yellow* to be replaced by 0, by propagating its up-score.

3.4 Context extraction

The main goal of phrase matching is to increase accuracy, by taking the relationships between the words into account. However, it also allows us to perform an additional operation called *context extraction.* In context extraction, we identify non-matching parts of the phrase which are close in the dependency structure to the parts which did match. These represent modifiers to what the user asked for, and serve to distinguish one matched caption from another. The contexts can be displayed with the retrieval results, and captions with identical or related contexts can be grouped together. A user is therefore able to select or reject several of the retrieval results in one go on the basis of the contexts. An example of using contexts to group results appears in the bottom half of in figure 9.

The algorithm for extracting the context is quite straightforward. In outline, it is as follows. We start with a set of context rules, each having the form $\langle t, v, p, u, C\rangle$ where t and u are lexical parts of speech, v a variable, p is a path consisting of one or more dependencies (expressed as the names of variables), and C is a lexical or phrasal category. Each of the elements may be a wildcard. The algorithm examines each word in the caption which matched against a word in the query. If there is a rule $\langle t, v, p, u, C\rangle$, such that the matched word has part of speech t and is stored in variable v, we see if there is a path p through the dependency structure from this word to one which was not matched with part of speech u. We then look for the smallest phrase with category C which surrounds the word. The resulting phrase is then added to the list of contexts, together with the matched word and the path that led to the context. The context rules are applied in order, and once a word or phrase has been added to the context, it is removed from consideration by later rules. Any of the items in the rule can be a wildcard `*`. In the case of a C, this is taken to limit the context to a single word.

An example of a path rule is $\langle noun, *, mod, *, *\rangle$, which selects all modifiers of nouns. A rule of this sort might be used for extracting an unmatched adjectival or nominal modifier. To get a whole prepositional phrase as a context, a suitable rule is $\langle noun, *, phead : prep, *, PP\rangle$, i.e. selecting a PP phrase surrounding a word which is connected to a matched noun via a *prep* dependency followed by a *phead* dependency. Some examples are shown in figure 9, for the query *camera with a lens*[4]. The scores of the retrieval results are ignored in this example, and it would probably be necessary to apply some sort of thresholding so that low-scoring captions did not appear in the collated results.

[4] Note that the attachment of the prepositional phrases starting with *with* can be handled by a special rule in the grammar, which attaches them to the immediately preceding NP, rather than using a general PP rule or the packing scheme described earlier. Similar principles are used for phrases starting with *of.* Such specialised rules are not a perfect solution, and represent an engineering compromise.

```
Captions and contexts
---------------------
  Camera with a lens
    {none}

  Large camera with a lens
    <camera [mod], large>

  camera with a lens on a table
    <camera [phead:prep], [on a table]PP>

  large camera with a zoom lens
    <camera [mod], large>
    <lens [mod], zoom>

  camera on a table with a long zoom lens
    <camera [phead:prep], [on a table]PP>
    <lens [mod], zoom>
    <lens [mod], long>

Captions gathered by context (with number of occurrences)
---------------------------------------------------------
  camera with a lens:
    camera modifiers:
      {none}     (1)
      on a table (2)
      large      (1)
    lens modifiers:
      zoom       (2)
      long       (1)
```

Fig. 9. Example contexts

4 Evaluation

There are a number of ways of evaluating the quality of ANVIL. Conventionally, information retrieval systems are evaluated according to their precision and recall, and this is what we will look at first. A detailed examination of the results from this test suggests how the matching process could be improved, and reveals some of the limitations of phrase matching. Since ANVIL was designed for use in products and not purely as a research system, a second important criterion is the engineering quality of the system; for example, how fast does it operate, and how does it scale with the size of the dataset. Thirdly, it is important to assess how easy it is to engineer the system from a NL point of view: how easily can we write the grammars, build the resources and so on. A human factors evaluation would also be interesting, to see how much factors such as high precision retrieval and context extraction contribute to the user's experience of the system. Some results in this area are reported by Rose *et al.* (2000).

4.1 Precision/recall evaluation

A standard way of evaluating information retrieval systems is to issue a series of queries, and compare the results delivered by the system with human judgements of the results that should have been achieved. The quality of the system is then measured using the precision, defined as the proportion of the retrieved documents (captions) that a user judges as relevant, and the recall, defined as the proportion of the the relevant documents which were retrieved. We adopt the procedure developed for the TREC conferences, in which the precision is computed by interpolation at standard values of recall, although in future evaluations it may be better to use the mean reciprocal answer rank used in the TREC question-answering track[5]. The evaluation procedure relies on the availability of human judgements of relevance for the queries. There is extensive data of this sort available for full-text document retrieval, in particular from the TREC conferences. However, for retrieval using short captions, the evaluation data is rather limited, and we therefore constructed our own test set as the principal means of evaluation.

The main dataset used a set of captions for the 1700 images in the Canon Photoessentials collection (`http://www.photoessentials.com/`). The captions were prepared to a set of guidelines, which placed the emphasis on such as things as the objects in the image rather than the composition of the image or value judgements about it. The guidelines were constructed following an in-house pilot study, and also in response to seeing the captions used by Smeaton and Quigley (1996)[6]. A few images were given multiple captions, usually a description (*tall ornate clock tower*) and a proper name (*Big Ben, London*). There were 1932 captions in the set, with lengths ranging from 1 to 22 words (9.0 average). Almost all of the captions were noun phrases.

The queries were constructed by taking pictures from a glossy magazine, and generating phrases which might elicit a related image. Simple matching with various parameters (including very low minimum score thresholds) was used to produce an initial pool of results for judgement. Queries which produced no results, either because no suitable images were available or because keyword matching failed to locate any, were discarded. The top 10 results from phrase matching with each query were judged for relevance separately by two human judges. The interpolated precision and recall were then calculated from the captions and scores returned by ANVIL.

ANVIL is aimed at casual users. Studies (and common sense) suggest that such users are not interested in getting complete retrieval results, and are not willing to look through a large number of results to find a good quality one (Pollock and Hockley 1997). As a consequence, we are mostly concerned with what happens in the first few retrieval results, and the precision at a 10% recall level was used as the main measure of system performance. Table 1 shows the results for the best run

[5] Details of TREC and links to the precision/recall evaluation procedure can be found at `http://trec.nist.gov`.
[6] Kindly supplied to us by Alan Smeaton.

Table 1. *Evaluation results*

Run	Precision at 10% recall
Simple matching (judge 1)	86%
Simple matching (judge 2)	85%
Phrase matching (judge 1)	95%
Phrase matching (judge 2)	92%

with simple matching alone, and for phrase matching following simple matching, using the two sets of relevance judgements. The results show a good improvement using phrase matching, improving precision by 7% to 9% overall simple matching. A query-by-query examination of the results shows that 43 of the 47 queries have a precision of 100% at 10% recall, and even at 50% recall, 34 of the queries achieve 100% precision. These are excellent results and we believe they help to show that NLP can provide significant value in information retrieval applications of this sort.

An evaluation was also carried out using the captions, queries and relevance judgements from Smeaton and Quigley (1996), but we considered the results to be unreliable, mainly because the relevance judgements were made against the images rather than the captions. Although this is the ultimate arbiter of system accuracy, it becomes difficult to tease apart the contribution from the retrieval engine from the effect of quality control on the captions. For example, the query *a race car driving on a race track* was judged relevant against an image with caption *racing cars on the race track* but not against one with caption *yellow race car on the race track*.

Unfortunately, we do not have figures for how much the use of WordNet contributed to the overall accuracy, although the results in Smeaton and Quigley (1996) suggest it could be as much as 20% when used in conjunction with a suitable similarity measure such as the Jiang and Conrath (1997) one.

4.1.1 Improving the matching process

The experiments on matching reveal a few cases which phrase matching cannot handle.

A first problematic case arises because the dependency structure, while expressing modification, still has a syntactic flavour about it. In the phrase *mountains reflected in a lake*, the word *mountains* will be designated as the head, with the remainder of the phrase linked to it. Consequently, there will not be a strong match against *lake reflecting mountains*, where *lake* is the head. This could probably be corrected through the grammar, by recognising that *reflected* is a passive form, and hence allowing the head of the PP complement to become the effective head of the phrase.

A more difficult problem arises because the matcher takes the first match it finds in the caption, rather than searching for all possible ones. A (contrived) example is the case of matching the query *red flower* against the caption *one flower which is not red and one flower which is red*. Depending on the internal representation of the phrases and the ordering of the rules, the matcher could choose to match *flower*

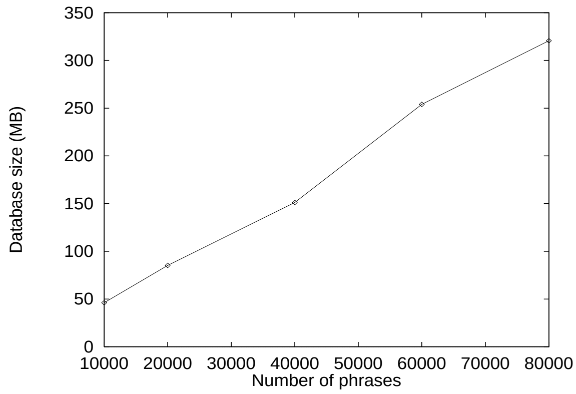

Fig. 10. Database size

and *red* against the first conjunct, and then give a poor score to the match because of the presence of *not*. It should be possible to solve this by allowing the matcher to search for the best match, but at the cost of a more complex and time-consuming algorithm.

A third, and harder, case arises in captions such as *man hitting a golf ball*, which does not match against *golfer playing a shot*. To match *man* with *golfer* we would need to use the remainder of the phrase to provide some sort of semantic coercion, and this is beyond the information we can get from the dependency structure. While this is a problem for phrase matching, it is also a problem for simple keyword matching. A more powerful form of NLP would be needed to solve it.

4.2 Performance and Scalability

The test set used in the previous section was only 1700 images and 1932 captions, and this is clearly not a reasonable size for a real image database. Flank (1998), for example, refers to an image database of 500,000 images, and even relatively small stock photography collections have several tens of thousands of images. We need to be concerned with how ANVIL will scale up to larger databases, and therefore measured the database size and creation time and the time to execute queries as the number of captions was increased.

To provide data for a large database, noun phrases were extracted from the parsed Wall Street Journal corpus, with a minimum length of 3 words, up to a maximum of 60 words. Databases were created with various numbers of phrases, from 10000 up to 80000. The database size (disc space) is shown in figure 10, for various numbers of phrases. The size shown is the total over the various files in which the database is stored. The time taken to create the database is shown in figure 11. The timings were taken on a Sun Ultrasparc 5 running Solaris, and are the elapsed time on an otherwise unloaded machine. The graphs show that the growth in both database size and creation time is roughly linear with the number of captions, which is what we would hope for. Although the time to index the 80000 caption database is around 3 hours, we do not consider this excessive, as the indexing only needs to be done

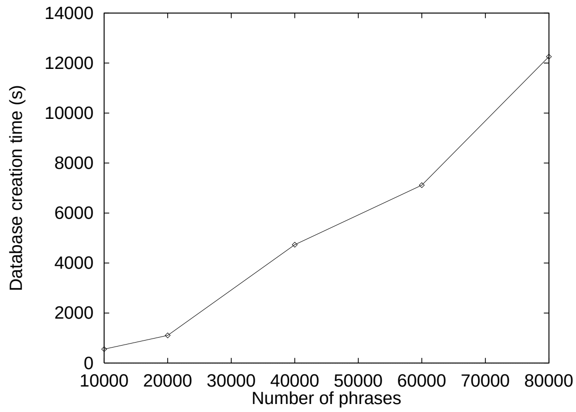

Fig. 11. Database creation time

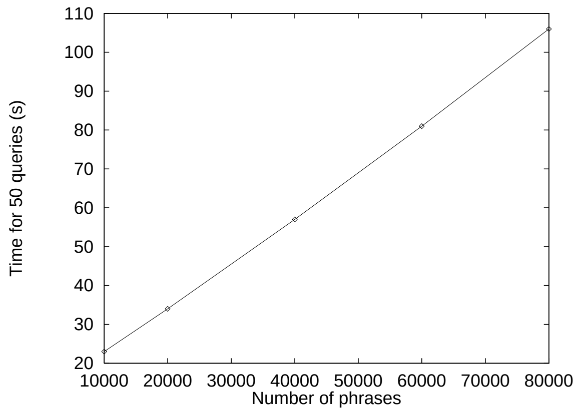

Fig. 12. Query time

once. Very large image collections might require investigation of distributing the database over multiple machines, which can be done easily as the captions can be analysed and stored in parallel.

The time to execute 50 queries is shown in figure 12. The results show the time for Smeaton and Quigley's algorithm for simple matching, followed by phrase matching. If vector cosine or BM25 is used for simple matching, ANVIL runs significantly more slowly. The time taken is closely linear with the number of captions. It is possible to limit the time by setting various system parameters, which control how many captions are selected as possible candidates for simple matching, how many are passed from simple matching to phrase matching, and how many are output. Score thresholds allow further control of the results. In these tests, the size of the initial set of candidates, which tends to dominate the overall time, was left effectively unlimited. Again, a distributed retrieval system could be constructed for high throughput applications, since both simple matching and phrase matching should be amenable to parallelisation.

More detailed profiling of the code shows that the database access is by far the dominant part of the timings. The best way of speeding the system up is likely to be using a more compact external representation, perhaps along the lines of Mohri (1996), and possibly increasing the degree of database caching or using an operating system which has been tuned for disc access. The NL part of the system is not the bottleneck, and so we could probably afford to move to more sophisticated techniques in the parsing and matching, or to allow a greater degree of ambiguity.

4.3 Language engineering

There are two main parts of ANVIL which are interesting from a language engineering perspective: the grammar, and the matching rules. The context extraction rules are very straightforward to write, and we will not discuss them further.

Development of the English grammar was relatively straightforward. We started with a test set consisting of around 5000 captions and queries. They were grouped together by part of speech sequence. The grammar was developed by inspecting the most frequent sequences and writing rules for them, and then progressing to less frequent ones. It is hard to modularise deterministic finite-state grammars, and a change to one rule can have a knock-on effects on others, so a change to the grammar was always followed by a comparison to see if earlier examples had been damaged. The resulting grammar gives a parse for all the examples in the test set (no tests have been carried out for its broader coverage). Since the dependency structures encoded in variables are the most important information, sometimes a phrase bracketing with poor linguistic motivation would be tolerated if the right dependency representation resulted, although some care has to be taken as the bracketing can be useful for context extraction. The main deficiency of the grammar is the lack of handling for ambiguity, which is potentially a drawback for image captions. For example, there are often several prepositional phrases, which could be attached in various combinations. The techniques described above for packing make this problem less serious.

Creating a set of matching rules was also fairly easy. A core set can be constructed by selecting a few phrases which ought to match, and then setting up rules for them. Examples might be comparing *yellow car* with *car which is yellow*, *car painted yellow*, *car which is not yellow* and so on. After seeding the rules from these examples, a collection of queries and relevance judgements against captions was used to find cases which were missed. Since the number of different relationships between words is fairly small, it is not too hard to check for a reasonable degree of completeness. What is more of a problem is the choice of weights for the rules, which affect the overall similarity between a query and a caption. It is hard to decide how different degrees of similarity should be represented numerically: for example, how similar is *tree by a lake* to *tree reflected in a lake*, and is it more or less similar than *tree with a lake behind*? Should they be derived from a training corpus, or finely tuned to correspond to user's judgements of the correct score? The real answer is that for most purposes it probably does not matter, and so the approach taken was simply to give most comparisons a term factor of 1.0 with a down factor of 0.7 for any continuations. This has the effect of making mismatches or missing matches become less and less important as the matcher goes deeper into the dependency structure, and that we get final scores which separate the strong matches from the weak ones, which is more likely to be important to users than an exact numerical rating for each image. The main cases where other scores were used was for special cases such as skipping over the head of the caption and matching the head of the query directly against a modifier (e.g. for *red sunset* compared to *lake with red sunset behind*).

5 Discussion

In this paper, we have presented ANVIL, a natural-language information retrieval system. ANVIL works on short captions and queries, and this poses a different problem to the lengthier documents and information requests which have characterised most IR work. Short text permits a more detailed analysis to be carried out without having to worry about either excessive computational cost, or the difficulties of discourse-level/document-level processing. On the other hand, longer text gives the retrieval system more to work with, so that a clearer picture of what the text is about can be deduced from clues such as the term frequency.

Perhaps the key point in the selection of NL technology in ANVIL was to pick a level of processing which gave advantages in terms of accuracy but without excessive overheads at run-time or in the development phase. In section 4.1.1, we mentioned some problematic cases which would have been helped by a more detailed analysis, producing a full semantic representation. However, we would have needed a much richer grammar and lexicon, and as well as the development cost, there would have been much greater risks of fragility. The finite-state dependency parser, and the matcher based on dependency structures are an attempt to strike the right balance. The entire system was built from scratch to a product-ready standard in a total of roughly 3.5 person-years of effort (for the English version only), including all of the software design and implementation, the grammar and rules development, the creation of the lexical resources, and the development of a graphical front-end operating through a web interface. Of this, the bulk was in the software development, with a total of around 6–9 person-months on lexical resource development, and 4–6 person-months on grammar and rule development.

Finally, context extraction represents a step forward in what NL contributes to IR. Much of the research in the NL/IR field concentrates on NL as either a way purely of improving accuracy, or as a provider of resources such as thesauruses (Smeaton 1999). Context extraction shows how a simple NL algorithm can provide extra functionality which is unavailable without using NL techniques. What we are doing is to really make information available to users in response to what they asked for, and not just to assign scores to documents.

Acknowledgements

The authors would like to thank Evelyn van de Veen, Tom Wachtel, Robert Fraser, Katsumi Masaki and Kazuyo Ikeda, who were all involved in the process of bringing ANVIL into existence. We also thank the five anonymous reviewers for their comments.

References

Abney, Stephen. 1996. Partial Parsing via Finite State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop.* Also available from `http://www.sfs.nphil.uni-tuebingen.de/~abney/`.

Aho, Alfred V., Sethi, Ravi and Ullman, Jeffrey D. 1986. *Compilers: Principles, Techniques and Tools.* Addison-Wesley.

Cutting, Doug, Kupiec, Julian, Pedersen, Jan and Sibun, Penelope. 1992. A Practical Part-of-speech Tagger. *Proceedings of the 3rd Conference on Applied Natural Language Processing*, pp. 133–140.

Elworthy, David. 1997. Automatic Error Detection in Part-of-speech Tagging. In Jones, Daniel and Somers, Harold (eds.) *New Methods in Language Processing*, pp. 256–268. UCL Press.

Fagan, Joel L. 1987. Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Non-syntactic Methods. Ph.D. Thesis 87–868, Department of Computer Science, Cornell University.

Fellbaum, Christiane (ed.). 1998. *WordNet: An Electronic Lexical Database.* MIT Press.

Flank, Sharon. 1998. A Layered Approach to NLP-based Information Retrieval. In *Proceedings of 36th ACL and 17th COLING*, pp. 397–403.

Grefenstette, Gregory. 1997. SQLET: Short Query Linguistic Expansion Techniques: Palliating One or Two-word Queries by Providing Intermediate Structure to WWW Pages. In *Proceedings of RIAO 97, The 5th Conference On Intelligent Multimedia Information Retrieval Systems and Management.*

Jiang, Jay Y. and Conrath, David W. 1997. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In *Proceedings of ROCLING X.* cmp-lg/9709008.

Mohri, Mehryar. 1996. On Some Applications of Finite-state Automata Theory to Natural Language Processing. *Natural Language Engineering*, **2**: 1–20.

Pollock, Annabel and Hockley, Andrew. 1997. What's Wrong with Internet Searching. *D-Lib Magazine*, March 1997.

Robertson, S. E., Walker, S., Jones, S. and Hancock-Beaulieu, M. M. 1994. Okapi at TREC-3. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pp. 109–126.

Roche, Emmanuel and Schabes, Yves. 1995. Deterministic Part-of-Speech Tagging with Finite-State Transducers. *Computational Linguistics*, **21**: 227–253.

Rose, T. G., Elworthy, D., Kotcheff, A., Clare, A. and Tsonis, P. 2000. ANVIL: A System for the Retrieval of Captioned Images using NLP Techniques. In *Proceedings of Challenge of Image Retrieval*, Brighton, UK.

Schwarz, Christoph. 1990. Automatic Syntactic Analysis of Free Text. *Journal of the American Society for Information Science*, **41**: 408–417.

Smeaton, Alan F. 1999. Using NLP or NLP Resources for Information Retrieval Tasks. In Strzalkowski, Tomek (ed.) *Natural Language Information Retrieval*, pp. 99–111. Kluwer.

Smeaton, Alan F. and Quigley, Ian. 1996. Experiments on Using Semantic Distances Between Words in Image Caption Retrieval. In *Proceedings of SIGIR*, pp. 174–180.

Sparck-Jones, Karen. 1999. What is the Role of NLP in Text Retrieval? In Strzalkowski, Tomek (ed.) *Natural Language Information Retrieval*, pp. 1–24. Kluwer.

Strzalkowski, Tomek. 1994. Robust Text Processing in Automated Information Retrieval. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pp. 168–173.