Retrieval from Captioned Image Databases Using Natural Language Processing

David Elworthy*
Canon Research Centre Europe, Guildford, United Kingdom
dahe@acm.org

ABSTRACT

At first sight, it might appear that natural language processing should improve the accuracy of information retrieval systems, by making available a more detailed analysis of queries and documents. Although past results appear to show that this is not so, if the focus is shifted to short phrases rather than full documents, the situation becomes somewhat different. The ANVIL system uses a natural language technique to obtain high accuracy retrieval of images which have been annotated with a descriptive textual caption. The natural language techniques also allow additional contextual information to be derived from the relation between the query and the caption, which can help users to understand the overall collection of retrieval results. The techniques have been successfully used in a information retrieval system which forms both a testbed for research and the basis of a commercial system.

Categories and Subject Descriptors

H.3.1 [**Information Systems**]: Content Analysis and Indexing; H.3.3 [**Information Systems**]: Information Search and Retrieval

Keywords

Information retrieval, Natural language processing, Image databases

1. INTRODUCTION

Text information retrieval is concerned with finding documents which match against user's query, and assigning a measure according to the closeness of the match. Natural language processing (NLP) can provide rich information about the text, and it might appear reasonable that this would result in better retrieval than conventional "bag of words" approaches. Fagan [2] reports experiments in which simple keywords were augmented with compound terms consisting of pairs of keywords. While the addition of compound terms produced better accuracy, no significant difference was observed between terms selected on the basis of their linguistic relationship and ones selected purely on the basis of their statistical association. Smeaton [12] makes similar observations and on the basis of a number of experiments concludes that NLP has little to offer IR. However, there are exceptions in some niche areas. For example, Flank [3] describes a retrieval system in which the "documents" are short image captions. She uses the techniques of searching on heads and head-modifier combinations introduced by Strzalkowski [14], and obtains high precision and recall. It therefore appears that in specialised applications, NLP may have something to offer.

Here we will look at a technique called *phrase matching*, which attempts to use lightweight, symbolic natural language analysis to improve retrieval accuracy. Like Strzalkowski's work, it relies on looking for combinations of words which stand in certain modification relationships, and like Flank, we have applied it to searching a database of annotated images. It differs from the earlier work in two important ways. Firstly, it does not simply use the analysis of the captions and queries as a source of compound terms, as Strzalkowski does. Instead it recursively explores the structure of the caption and query, checking that terms stand in equivalent modification relations in the two phrases. This also allows the match score to be finely tuned and special cases such as negation to be handled. Secondly, by means of a further algorithm called *context extraction*, information about non-matching parts of the caption, related to the parts which did match, can be obtained. The retrieval results can then be organised and categorised by the contexts they have in common, with the goal of helping users of the retrieval system to understand and organise the results. This is an important step, because it provides information which is unavailable without natural language analysis, and shows that NLP can contribute in adding new functionality as well as improving accuracy.

In section 2 of this paper, we will introduce the phrase matching algorithm, and give some evaluation results. Section 3 then moves on to context extraction. Some conclusions and suggestions for future work are presented in section 4. We first briefly look at the application for which the work was intended.

*Now at: Microsoft Research Limited, St George House, 1 Guildhall Street, Cambridge CB2 3NH, United Kingdom

1.1 The ANVIL system

ANVIL (Accurate Natural Language Visual Information Locator) is a retrieval system for databases of digital photographs, intended for operation over the world wide web. The photographs are annotated with captions, typically between 10 and 30 words in length, which describe the subject matter of the image. The system is intended for casual users, and it is therefore important to make it easy to formulate and refine queries, and to help the users understand the results. This is the main motivation for using phrasal captions and phrase matching: while traditional IR techniques over collections of keywords may give good recall, they are not really suitable for users who will give up if they do not see an acceptable result in the first few presented by the system. ANVIL is further enhanced by an interactive user interface, details of which can be found in Rose et al. [8].

In outline, the processing in ANVIL proceeds as follows. When images are registered with the system, their captions are analysed into a meaning representation. The terms from the captions are stored in an index database, pointing to records containing the image identifier and the analysed caption. In retrieval, the terms are extracted from the query and used to find candidate captions using conventional IR techniques such as vector-cosine matching or the similarity techniques of Smeaton and Quigley [13]; this phase is called simple matching. The query is analysed to a meaning representation in the same way as the captions, and the representations of the query and candidate captions are compared using natural language matching techniques. The result of the comparison is a score, which is combined with the score from simple matching. Contexts may also extracted at this stage, and the resulting images with their scores, captions and contexts are presented to the user.

2. PHRASE MATCHING

The basic idea in phrase matching is as follows. We start by analysing the query and the caption into dependency structures, in which the words are connected by labelled links indicating the relationship between them. One word (or occasionally more) will not be a modifier of any other words. It is designated the head, and is the word which says, in most general terms, what the caption is about. The head of the query is compared against words in the caption, starting from its own head and progressing to modifiers if no match is found. If there is a match, the modifiers of the query head are compared against modifiers of the corresponding term in the caption. For each word that matches, the process recurses in a similar way down through the dependency structure. The modification relationships can be simple ones, or they can involve tracing through several dependency links. Each stage of the comparison has a score associated with it, so that strong and weak matches can be assigned different scores. Finally, we allow matching of elements in the dependency structure against fixed expressions, to detect special cases such as negation.

Figure 1 shows the dependency structures for two phrases with similar meanings. Dependencies are shown as pointing from a modifier to the term it modifies. Although dependency structures go some way to abstracting away from the syntactic analysis, we still need a way of assigning a similarity between non-identical structures. In this example, we

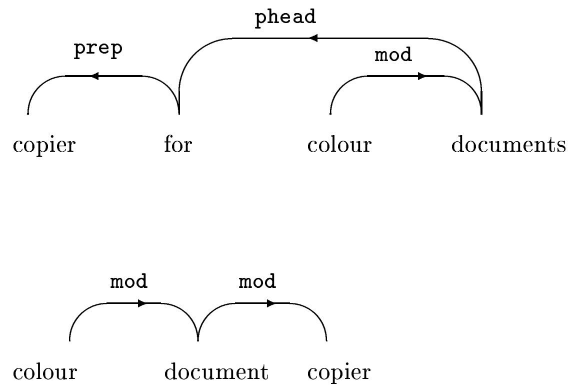

Figure 1: Example dependency structures

want the noun-noun modification between *copier* and *document* in the lower phrase to have a high similarity to the modification via the preposition *for* in the upper one.

For convenience, we represent dependency structures using a notation of indexed variables, in which the name of the variable stands for the name of the dependency, and the variable is indexed on the modified word. An unindexed variable is used for the head. The examples can then be written as

```
colour document copier
    head = copier
    mod[copier]   = document
    mod[document] = colour

copier for colour documents
    head = copier
    prep[copier]   = for
    phead[for]     = documents
    mod[documents] = colour
```

Thus, for example, `mod[copier] = document` indicates that *copier* stands in the `mod` relation to *document*, i.e the `mod`(ifier) of *copier* is *document*.

Dependency structures are especially suitable for this kind of processing. They are closely related to the syntactic form, but abstract away from the linear order of the words and fine details of phrase structure. From a practical point of view, dependency structures can be computed quickly and efficiently; see for example, the dependency parser built by Järvinen and Tapanainen [5] or the Link grammar parser of Sleator et al. [11]. We use a finite-state parser which has been modified to deliver the dependencies as well as the phrase bracketing (Elworthy [1]). It works in time roughly proportional to the square of the number of words in the phrase.

2.1 Matching rules

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

Figure 2: Example matching rules

A system of rules specifies what relationships can be treated as equivalent. A small set of example rules appears in figure 2. The left and right hand sides of a comparison express paths through the dependency structure. The idea is that if we have already found a query word which matches a word from the caption, we then follow the specified paths from these words, and compare the words lying at the end of the paths.

It is convenient to gather rules into named groups, such as `head_rule` and `mod_rule`. One group is designated the start group, and its rules are applied to start the matching process. Within a group, the rules are applied in order, so that later rules in a group can be used to test words which were not caught by the earlier rules. Each rule has a continuation, which specifies what should happen after it has been applied. As an example, consider the rule

```
head = head    1.0 => mod_rule 0.7;
```

This says that after matching head words, continue with the rule group `mod_rule`. The words which have just matched provide the starting point for paths in the continuation; in effect they are substituted where `[]` appears in `mod_rule`. The special continutation `Done` indicates that no further comparison is to be carried out from the words that matched.

The process is started by comparing words without indexing, stored in `head`. Thus, the structures in the examples of figure 1 can be matched by starting with `head = head` and then continuing with `mod[] = phead:prep[]`, indicating that a modifier of the head (`mod[]`) can be compared with the head of a prepositional phrase (`phead`) reached by following from a matched caption word a preposition (`prep[]`).

There are two special sorts of rules: mopping-up rules and token rules. Mopping-up rules specify that certain words are to be considered to have matched, without actually consuming any words from the other phrase. One use is to catch words from the query which did not have a counterpart in the caption. For example,

```
mod[] ?     0.3 => Done 1.0;
```

causes modifiers from the query to be mopped up[1]. Token rules allow matching against specific words. For example, the rule

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

allows an `amod` ("adverbial" modifier) in the caption to be tested against the literal word *not*, with an effect on scoring described below. There are a few further variants of rules which we will not discuss here, for example rules with a negated test, and ones which are sensitive to word order.

2.2 The scoring scheme

The scoring scheme is a critical part of phrase matching, as it will allow us to distinguish exact and near-exact matches from partial and weak ones. The general approach is to assign each word of the query phrase two numeric values, called the *score* and the *weight*. The score of a query word is a measure of how well it matched considered in isolation from the rest of the caption, while the weight indicates the importance of the rule application. In general, words which are compared in the start rule group, such as the head, will be more important than ones compared as a result of a continuation, such as modifiers. Scores are assigned to query words if they actually matched a word in the caption, or if they were caught by a mopping up rule or token rule. The score does not take the caption words into account, other than an allowance for their similarity with query words. A special score, called an *up-score* is also used to handle words at the end of paths for special cases such as negation. Writing the scores as s_i and the weights as w_i, the overall score of the match is $\sum s_i w_i / \sum s_i$, modified by the up-scores as described below.

The rules are annotated with two values, called the t (term) factor and the d (down) factor. In general, the t-factor provides the basic score for words which were matched by the rule, and the d-factor sets the weight for continuations. Thus, in

```
head = mod[]    0.5 => mod_rule 0.7;
```

the t-factor is 0.5 and the d-factor is 0.7.

At the start of matching, the weight is 1.0. As we follow through continuations, it is the product of the d-factors of the rules leading to this point. If the rule above were in the start group, the weight of words matched in `mod_rule` would be 0.7, and if `mod_rule` contained

[1] Since this is in the start rule group, the whole unmatched range of the `mod` variable is used, without indexing.

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

car which is yellow
    head = car
    rel[car]    = which
    cop[which] = is
    vhead[is]   = yellow

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

Figure 3: Dependency structures for the matching example

```
mod[] = mod[]   1.0 => submod_rule 0.6;
```

then the weight in `submod_rule` would be 0.7×0.6. The scores are formed from the product of the t-factor of the rule, and two special factors. Firstly, the similarity between the words can be used. For example, we might allow *car* to match *vehicle*, but with a reduced score. This factor could be calculated using lexical similarity metrics such as those of Resnik [7] or Jiang and Conrath [4]. A further extension would be to recognise that the agentive suffix *X-er* (as in *copier*) allows a match against the whole phrase *machine for X-ing* (as in *machine for copying*), and similar rules based on derivational morphology. We do not take this step in the current version of phrase matching.

The second special factor is the up-score. When a `Done` continuation is reached, its d-factor is multiplied into the score assigned by the rule which invoked it[2]. Usually the factor will be 1.0, 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 2. When a negation is seen, we effectively cancel the score on the word which is negated, by using a d-factor, and hence an up-score, of 0. Note that making this kind of adjustment based on word pairs without the recursion through the overall structure, as in Fagan's and Strzalkowski's work, is very hard to do.

To show the rules in operation, suppose the query *yellow car* is tested against *yellow car*, *car which is yellow* and *car which is not yellow*. The dependency structures, written as variables, are shown in figure 3, and a trace through the matching process appears in figure 4. In particular, note how the rule

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

causes the previous score assigment for *yellow* to be replaced by 0 when comparing against *car which is not yellow*. The scores in this rule set are chosen on the basis of examining a variety of examples, some of which might be expected to provide a close match, some a partial match, and some a weak match. No experiments on learning the scores from data have been carried out.

[2]This represents a harmless overloading of the rule notation

2.3 Evaluation

Evaluation of image caption retrieval is limited by the lack of suitable large test collections. We therefore created our own captions for a set of digital photographs. The captions were prepared according to a set of guidelines, so that they emphasised the objects in the image rather than layout or composition. The guidelines were formulated to overcome problems with quality which had been seen both in a pilot study, and the captions used by Smeaton and Quigley [13]. 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. It is relatively easy to construct a grammar which correctly analyses all the phrases.

A query set was constructed by taking pictures from another source, and devising phrases which should elicit a related image. An initial set of results was obtained by pooling several keyword-based retrieval runs, discarding queries which produced no results[3]. The top results from phrase matching with each query were then judged for relevance by two human assessors, acting separately. Neither assessor was responsible for writing the captions; one of them devised the queries. A standard precision-recall measure was then calculated, using the TREC interpolation procedure (from `http://trec.nist.gov/`). An example of the output for a query, showing some sample captions appears in figure 5.

The main comparison point between different tests was chosen to be the precision at 10% recall. This represent the case of naive or casual users, who do not care about completeness in the results and who want high accuracy in the first few (Pollock and Hockey [6]). The precision at 5 documents and the R-precision were also calculated, although they are less useful, partly there is often a very small number of relevant results in such a small test set. Table 1 shows the results for a simple weighted keyword matching strategy, and for phrase matching, using the two sets of relevance judgements.

Phrase matching produces a good improvement over simple matching. 43 of the 47 queries in the best phrase matching run gave a precision of 100% at 10% recall. Inspection of the remaining results shows that the errors could typically only be fixed with a richer semantic representation allowing interaction between the meaning of the words. For example, the query *plastic toys* fails to match *plastic sword* because a sword is not normally a toy. The precision at 5 documents shows less of an improvement as a result of the small numbers of relevant captions.

Note that due to the lack of sources of good quality relevance judgements for this kind of application, the results should be taken as suggestive of the quality of phrase matching rather than as a definitive statement. An evaluation was

[3]With such a small test collection and using a single retrieval systems, it might have been better to construct complete relevance judgements rather than use pooling. However, time pressures obviated doing this.

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

Figure 4: Matching in action

Run (assessor)	Precision at 10% recall	Precision at 5 documents	R-precision
Simple matching (I)	85%	45%	61%
Phrase matching (I)	92%	46%	66%
Simple matching (II)	87%	49%	63%
Phrase matching (II)	95%	53%	72%

Table 1: Evaluation results

also carried out using the data from Smeaton and Quigley [13], but we concluded that the results could not be trusted, because the relevance judgements were made against the images rather than the captions, and both the captions and queries were of relatively low quality. In some cases we found pairs of almost identical captions, one of which was judged relevant and one irrelevant by Smeaton and Quigley's assessors. For comparison, the best precision at 10% recall reported by Smeaton and Quigley is around 62%.

3. CONTEXT EXTRACTION

Context extraction is a means of obtaining additional information about phrases which matched, by using the unmatched parts of the caption which are close in the dependency structure to parts which did match. For example, if the query was *camera lens*, and the captions included *long camera lens* and *camera lens on a table*, then the contexts would be *long* and *on a table*. Context extraction becomes valuable when there are many retrieval results. Captions with similar contexts can be grouped together, for example as shown the bottom half of in figure 7. A user can therefore select or reject several retrieval results in one go by examining just the contexts.

The algorithm for extracting the context is quite straightforward. It is outlined in figure 6.

The algorithm uses pre-defined context rules of the form $\langle r_t, r_v, r_p, r_u, r_C \rangle$. In essence, it looks for words which successfully matched, have a given part of speech r_t and are stored in a variable r_v. It then follows a path r_p through the dependency structure, arriving at an unmatched word u with part of speech r_u, and then extracts the syntactic context around it using the phrase type r_C (for example, *PP*, prepositional phrase). The restriction to the smallest phrase is simply for cases where a phrase of a given type embedded within another phrase of the same type. The elements of the rule can be wildcards, which match anything. The algorithm delivers a set of pairs, each of a matched word and its context. Simpler versions of the rules which do not have all of these elements might also be possible.

An example path rule is $\langle noun, *, mod, *, * \rangle$, which selects all modifiers of nouns. A rule of this sort might be used for extracting *long* in the examples above. To get the context *on the table*, a suitable rule might be $\langle noun, *, phead : prep, *, PP \rangle$, i.e. from a matched word, follow a *prep* link followed by a *phead* link, and select the *PP* surrounding the resulting word. Some example contexts resulting from these rules are shown in figure 7, for the query *camera with a lens*. The bottom half of the figure shows the results gathered together by context. Presenting the results to the user in this way would allow selection or rejection of several results with a single decision, thus making it easier to manage large result sets.

Perhaps the most important point about context extraction is not the algorithm or exactly what the results look like, but the use of NLP to provide extract information. Although there is some work in IR in extracting relevant parts of the text, for example using named entity extraction, in

```
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 grey ridged surface.
       * camera: Canon, on a grey ridged surface
0.1    an astronaut floating within a space craft, showing the on-board cameras.
       * cameras: on-board
```

Figure 5: Example ANVIL query result

```
let P be the set of path rules (input)
let T be the set of current words, initialised to all matched words (input)
let U be the set of available words, initialised to all unmatched words (input)
let S be the set of contexts, intially empty (output)

while T is not empty
{
    select a word t from T

    for each word u in U
    {
       if there is a context rule <rt,rv,rp,ru,rC> in P such that
           has_pos(t,rt)
           AND in_var(t,rv)
           AND has_pos(u,ru)
           AND on_path(t,u,rp)
       then
           find the smallest phrase C such that valid_phrase(C,rC,u)
           if there is such a C then
               add the context <t,C> to S
               remove u from U
    }

    remove t from T
}

where
  has_pos(t,rt)   if t has part of speech rt
  in_var(t,rv)    if t is stored in variable rv
  on_path(t,u,rp) if the path rp connects t and u
```

Figure 6: The context extraction algorithm

```
Query = camera with a lens

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
----------------------------
  camera with a lens:
    camera modifiers:
      {none}     (1)
      on a table (2)
    lens modifiers:
      large      (1)
      zoom       (2)
      long       (1)
```

Figure 7: Example contexts

general IR systems just output a ranked list of matching documents. Context extraction demonstrates that using NLP, which works with more detailed information structures than traditional IR, we can produce a richer form of output.

4. DISCUSSION

The approach most closely related to phrase matching is that of Sheridan and Smeaton [10]. They start by constructing a dependency tree (of a different form to ours), in which interior nodes can be labelled, for example to mark the head or record the preposition which links words on the nodes under it. The matching process looks for pairs of words which are syntactically related in the query tree, and which both appear in the tree for the key (caption). The nearest parent nodes for the pairs of words are then checked for compatibility. Any parts of the dependency structure which hang off the paths to the parent node, called the residual structure, are examined to see if they could disrupt the matching. For example, if words were both nouns, a verb in the residual would block the match, since its presence indicates the nouns cannot stand in a head/modifier relationship. The whole process is launched by looking at the rightmost node in the query structure. A score is assigned based on the proportion of words which match, possibly modified by certain residual nodes.

The main way in which this differs from our algorithm is that the selection of nodes to try is *ad hoc*, rather than being guided directly by the modification structure. The use of rules with a reduced score (such as `head = mod[]` above) and mopping up rules is also more explicit and modular than the use of residuals. Furthermore, the scoring process in our phrase matching takes the depth through the the structure (and hence the significance of the terms) into account better, and is arguably more perspicuous. Some further related work can be found in Schwarz [9], in in which syntactic structures are first converted to a normal form and then compared.

The work was conducted before the rise of interest in question-answering (Voorhees and Tice [15]) which also uses short, precise queries to locate specific information. Most of the TREC-8 question-answering systems used IR followed by entity extraction, and one important limitation of this technique when applied to the application described here is worth noting. The entity extracted as the answer can appear anywhere in the retrieved text and consequently could part of some modifying phrase rather than the main point of the caption, and so result in retrieving images which do not correspond well to the request. By contrast, the phrase matching rules can penalise such matches, provided the captions model the content of the images well.

Two challenges follow. The first is to adapt techniques of this sort to full text documents, in which there is a much richer linguistic structure, and where different parts of the text may have different information content (a title compared to a sentence in parentheses, for example). Secondly, there is a need to use evaluation measures which place more emphasis on interactive retrieval and user reaction. The assumption in much IR is that the results are simply judged by their relevance to the user's information needs, essentially as a binary decision. With an extension such as context extraction, where the retrieval results contain extra information over the original data, we need an evaluation technique which is able to take into account the benefit obtained from the results by the information user.

Acknowledgements

The algorithms were implemented by the author and Aaron Kotcheff. The ideas also benefitted from discussions with Tony Rose and Amanda Clare, and (at an earlier stage) Tom Wachtel and Evelyn van de Veen.

5. REFERENCES

[1] D. Elworthy. A finite state parser with dependency structure output. In *6th International Workshop on Parsing Technology, Trento, Italy*, 2000.

[2] J. L. Fagan. *Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Non-syntactic Methods*. PhD thesis, Department of Computer Science, Cornell University, 1987.

[3] S. Flank. A layered approach to NLP-based information retrieval. In *Proceedings of 36th ACL and 17th COLING, Montreal*, pages 397–403, 1998.

[4] J. Y. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of ROCLING X, Taiwan*, 1997.

[5] T. Järvinen and P. Tapanainen. A dependency parser for English. Technical report TR-1, Department of General Linguistics, University of Helsinki, 1997.

[6] A. Pollock and A. Hockley. What's wrong with internet searching. *D-Lib Magazine*, March 1997.

[7] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal*, 1995.

[8] T. Rose, D. Elworthy, A. Kotcheff, A. Clare, and P. Tsonis. ANVIL: a system for the retrieval of captioned images using NLP techniques. In *CIR2000: Third UK Conference on Image Retrieval, Brighton*, 2000.

[9] C. Schwarz. Automatic syntactic analysis of free text. *Journal of the Americal Society for Information Science*, 41(6):408–417, 1990.

[10] P. Sheridan and A. F. Smeaton. The application of morpho-syntactic language processing to effective phrase matching. *Information Processing and Management*, 28(3):349–369, 1992.

[11] D. D. K. Sleator and D. Temperley. Parsing English with a link grammar. Technical report CMU-CS-91-196, School of Computer Science, Carnegie Mellon University, 1991.

[12] A. F. Smeaton. Information retrieval: Still butting heads with natural language processing? In M. T. Pazienza, editor, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer-Verlag, 1997.

[13] A. F. Smeaton and I. Quigley. Experiments on using semantic distances between words in image caption retrieval. In *Proceedings of 19th SIGIR, Zurich*, pages 174–180, 1996.

[14] T. Strzalkowski. Robust text processing in automated information retrieval. In *Proceedings of the 4th Conference on Applied Natural Language Processing, Stuttgart*, pages 168–173, 1994.

[15] E. M. Voorhees and D. M. Tice. The TREC-8 question answering track evaluation. In *Proceedings of the 8th Text Retrieval Conference*, 1999.