1. Introduction

Cranfield collection [1] is a well-known IR test collection containing 1,400 aerodynamics' documents and 225 common queries. It also includes a set of correct results from human experts, which make the collection suitable for an automatic IR system evaluation.

The queries are formulated in plain English. For example, the Q1 is:

> *What similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft?*

Remember the Q1 – we will be using it as an example throughout the document.

Due to the format of queries and a very specific domain, achieving high precision on Cranfield collection is not easy. Measured at 10 retrieved documents, Lucene [2] returns only 37.3% out of the box. In fall 2004, CS276A students attempted to tune Lucene's scoring model (TF-IDF) to increase this value [3]. These efforts were largely unsuccessful, resulting in precisions below 45% [4]. In addition to that, [9] used four different methods to increase the precision. However, their best precision was only 41.2% (using LSI).

This project tackled this problem from a different angle. Instead of tuning Lucene's scoring model, we improved the precision by leveraging the fact that the Cranfield queries are formulated in the natural language. This paved way to several query transformations based on the NLP understanding of the queries.

The project implemented three query transformations: a simple POS-based query cleansing, noun phrase query expansion, and fixed thesaurus-based query expansion (using WordNet [6]). In addition to Lucene and WordNet, Stanford parser [5] was used in these tasks.

The original project proposal also planned to implement an automatically generated similarity thesaurus [11]. However, after discovering that LSI was already applied to the Cranfield collection in [9], I decided to implement WordNet and noun phrase expansion instead (as LSI and automatically generated thesaurus are conceptually very similar).

The rest of the document will provide more details on both the status of research in these areas, as well as the results achieved by this implementation.

2. Related work

Some related work on NLP IR was done in the last decade, mostly in the context of the TREC [7] competition. The results, however, varied.

Noun phrases were successfully used in [12] and [13]. [12] reported precision improvement of 2.5% by using them. The paper advocates a use of "phrase-based indexing". This is something we get for free from Lucene (once the query is constructed properly).

Even though query expansion is usually effective in improving recall, [14] also showed that using similarity-based query expansion can improve precision. Unfortunately, they do not provide results of applying this technique standalone; their precision figures include other methods as well.

On other hand, [8] claims that the attempts to use NLP in IR failed and cites numerous failure examples. The presentation, however, accepts that NLP can still be useful for a small domain with short documents, which is exactly what Cranfield collection is.

Other known attempt at Cranfield collection is [9]. This work employed four vector-based IR methods: vector space model (VSM), latent semantic indexing (LSI), generalized vector space model (GVSM), and approximate dimension equalization (ADE) (essentially a combination of the prior three). Their achieved precisions varied from 21.7% (GVSM) to 41.2% (LSI).

3. Implementation details

We started from the CS276A PE1 starter code, which gives a precision of 37.4%. Pretty soon it was clear that all our transformations require a stemmer in order to work properly. So we reapplied stemming to the baseline code as well, receiving a baseline precision of **38.7%**. This is the baseline precision we will be comparing ourselves against.

In general, the Cranfield queries return from 159 to 1,356 hits on the baseline. Given the fact that the evaluation was on the top 10 documents, it is obvious that our job of finding proper top 10 was not simple.

Note that we did not reuse any CS276A PE1 TF-IDF optimizations, in order to enable a fair comparison with the former.

All precision numbers in this document refer to an average precision of 225 Cranfield queries measured at 10 retrieved documents.

3.1. POS-based query cleansing

The default Lucene stop word list contains only 35 terms. This is not nearly enough to filter out noise words in the Cranfield queries.

On the other hand, applying Cornell SMART system stop word file [15] (containing 571 terms) reduced the precision to 38.4%. It was an unexpected result as using the same stop word file improves precision when applied together with some TF-IDF parameter manipulation.

So the compromise idea was to filter out words based on their part of speech. This, actually, worked well.

The best precision was **39.3%** (+0.6%)**.** It was achieved when stripping off the following parts of speech: -LRB-, -RPB-, WDT, WP, DT, TO, CC, ‘.’, SYM, PRP, PRP$, EX, POS.

I.e. our Q1 becomes

similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft

Other things tried:

1. Removing MD significantly decreases the precision (-0.2%). This is one of the reasons we perform better than the larger stop word list (which contains all the modals).

2. The above POS list is optimal; removing any other parts of speech reduced the precision. So the initial thought that you can throw away adverbs and increase precision was plain wrong – apparently all meaningful parts of speech matter!

3. We also tried boosting/deboosting query terms based on the POS. This did not work well.

4. Replacing the default OR operator (connecting query terms) with a more discriminating AND never worked – there are never enough results to return for an AND query. We also tried it in other approaches – it never worked.

There are couple of possible explanations why POS-based cleansing worked better than the Cornell stop list:

1. As mentioned, modal verbs improve the precision here, so keeping them was helpful.
2. POS-based cleansing removes non-words (dashes, colons, parenthesis) from queries and the stop word list does not. This prevents the Lucene query parser from (mis)treating them as special characters.

But, of course, since all removed parts of speech represent closed sets (unlike nouns/verbs/adjectives/adverbs), this query transformation is equivalent to some predefined word stop list.

<u>Transformation summary</u>
Input: 38.7%
Output: 39.3%
Best result achieved on: remove -LRB-, -RPB-, WDT, WP, DT, TO, CC, ‘.’, SYM, PRP, PRP$, EX, POS.

3.2. Noun phrase query expansion

Two things matter in the noun phrase query expansion: what phrases we extract and how we transform the query to incorporate the extracted phrases. We will discuss ways to do both followed by the results of their application.

3.1.1. Noun phrase extraction

In order to identify the noun phrases inside the sentence, we had initially planned to use YamCha NP chunker [16]. It proved to be impossible as the chunker requires annotated training data, which we do not have. Instead, we implemented our own simple chunker logic on top of the Stanford parser.

There were two decision points here:

1. What to do when one NP (outer) contains another NP (inner)?
2. Which regular expressions (over tags) include in the phrases?

We tried all the possible combinations. The best results were achieved on including only outer NPs and using {noun, adjective}+.

For example, our Q1 parsed as

```
(S
  (NP (NN similarity) (NNS laws))
  (VP (MD must)
    (VP (VB be)
      (VP (VBN obeyed)
        (SBAR
          (WHADVP (WRB when))
          (S
            (VP (VBG constructing)
              (NP
                (NP (JJ aeroelastic) (NNS models))
                (PP (IN of)
                  (NP (JJ heated) (JJ high) (NN speed) (NN aircraft))))))))))))
```

will result into following phrases:

“similarity laws”, “aeroelastic models heated high speed aircraft”

3.1.2. Query transformation

The extracted phrases are appended to the query using the OR operator. Appending them as is, however, will result in a significant precision loss, because Lucene will try to match them word-by-word, i.e. *aeroelastic* followed by *models* followed by *heated* etc, which will not find them.

The solution is to employ Lucene’s phrase slop. Phrase slop is the “degree of sloppiness” that the phrase match allows. It is defined as an allowed edit distance where the units correspond to moves of terms that are out of position in the query phrase.

For example, a phrase “aeroelastic model” will be matched by “aeroelastic airplane model” (doc 78) for slop >= 1, as we need to move “model” one word left in the document for the full match to happen. By default, slop is 0, and the document will not be hit.

We devised several formulas for slop calculation. The best was to use a brute force multiplying by 10 the edit-distance between the noun phrase the way it appears in the query and the way it is extracted by the NP chunker (the two can be different since we only preserve nouns and adjectives).

I.e. our Q1 will become something like:

> *similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft "similarity laws" "aeroelastic models heated high speed aircraft"~40*

since the edit-distance between "a*eroelastic models of heated high speed aircraft"* and "a*eroelastic models heated high speed aircraft"* is 4.

Also note that 1) phrases are quoted in Lucene and 2) the slop is marked by '~' after the phrase.

3.1.3. Results

As mentioned, the best result was achieved on extracting only outer NPs of nouns and adjectives using 10 times the noun phrase edit-distance as the slop.

This, however, only achieved a precision of 39.25%, which is 0.1% below what we had before.

Here are other things we (unsuccessfully) tried:

1. Use both inner and outer NPs – 38.0%.
2. Boosting NPs (x 2) – 38.5%.
3. Deboosting NPs:
   0.5 – 39.0%
   0.1 – 38.8%.
4. Fixed slop:
   5 – 39.1%
   10 – 39.0%.
5. Different edit-distance formula (number of words stripped off the original phrase): 38.8%.
6. Different multipliers for the chosen edit distance:
   1 – 37.8%
   2 – 39.1%
   4 – 39.20%.
7. Replacing the entire query with only the NPs: 37.2%.

<u>Transformation summary</u>
Input: 39.3%
Output: 39.3%
Best result achieved on: do not do the noun phrase query expansion.

3.3. Thesaurus-based query expansion

The last transformation was query expansion using WordNet. Though this was not expected to succeed, due to a very specific domain, it is still interesting to analyze the results we received.

We retrieved synonyms of word w by iterating over every sense s of the word and retrieving every word ws, which corresponds to the sense. In addition, we filtered out all synonym candidates that were stemmed variants of w.

For example, expanding Q1 will result in:

similarity
(laws Torah Pentateuch)
(must should have got ought need have)
(be follow "make up" personify constitute embody exist cost live represent equal comprise)
obeyed
(when once)
(constructing build manufacture retrace reconstruct fabricate make)
aeroelastic
(models manikin modelling example modeling poser "fashion model" mannequin simulation framework manakin "role model" exemplar mannikin "theoretical account" "good example")
of
(heated "heated up" "het up" het)
(high gamy eminent high-pitched gamey mellow "in high spirits")
(speed amphetamine swiftness upper velocity "f number" fastness "pep pill" "stop number" hurrying speeding "focal ratio")
aircraft

Note how the query is constructed: a word and all its synonyms must be grouped together inside parenthesis. This will cause Lucene to rank it as a sub-query, which is exactly the effect needed. We don't want one document to be ranked higher than other just because it has more synonyms of the same word.

WordNet Java API requires you to specify a word and its POS (it supports four parts of speech -- noun, adjective, verb, or adverb) to get its senses. So it was natural to experiment and see which parts of speech correspond to what changes in the precision.

POS expanded	**Average precision (%)**
All	35.7
Noun	36.6
Adjective	38.8
Adverb	**0.394954**
Verb	0.394950
Noun + adjective	35.4
Adverb +	0.394950

verb	

So the best result is reached when expanding adverbs only. This will expand our Q1 as:

similarity laws must be obeyed (when once) constructing aeroelastic models of heated high speed aircraft

Transformation summary
Input: 39.3%
Output: 39.4%
Best result achieved on: expand adverbs only.

4. Future directions

The most promising thing to try in the future is query expansion using NASA thesaurus [10]. The NASA thesaurus is domain-specific and could help us significantly with the terms WordNet does not know about.

For example (unfortunately, it is a secure PDF and I cannot copy-paste), in the Q1 context, it contains entries for 'aeroelasticity' and 'aircraft' (3 pages of entries!), while WordNet knows zero synonyms for both.

5. Conclusions

Just as it happened earlier with all the known experiments, our attempt to increase Cranfield collection IR precision (using NLP) was not successful, resulting in the best precision at about 39.5%. This is only 0.7% above our baseline and is several percents lower than the best results of both TF-IDF and LSI.

Among the things we tried, POS-based query cleansing outperformed a regular stop word list whereas noun phrases did not cause any improvement, and fixed thesaurus query expansion only resulted in miniscule improvement over the baseline.

Clearly, given more time, I believe we would eventually break even with the other approaches. NASA thesaurus looks like the most promising thing to try in the future. It contains all the terms we tried to find there, unlike WordNet whose coverage of the area is naturally not sufficient.

6. References

1. Glasgow IDOM - Cranfield collection
   http://ir.dcs.gla.ac.uk/resources/test_collections/cran
2. Apache Lucene – Overview
   http://lucene.apache.org/java/docs/index.html
3. CS 276A Practical Exercise #1
   http://www.stanford.edu/class/cs276a/handouts/pe1.doc
4. Practical Exercise #1 grading criteria
   http://www.stanford.edu/class/cs276a/handouts/pe1criteria.doc
5. Stanford Parser
   http://nlp.stanford.edu/software/lex-parser.shtml
6. WordNet -- a lexical database for the English language

http://wordnet.princeton.edu
7. Text Retrieval Conference (TREC)
http://trec.nist.gov
8. Artificial intelligence & natural language processing
http://dis.shef.ac.uk/mark/courses/Porto/NLP.ppt
9. Approximate Dimension Equalization in Vector-based Information Retrieval. Fan Jiang, Michael L. Littman.
http://citeseer.csail.mit.edu/jiang00approximate.html
10. NASA thesaurus
http://www.sti.nasa.gov/products.html#pubtools
11. Concept based query expansion. Yonggang Qiu, Hans-Peter Frei.
http://citeseer.ist.psu.edu/qiu93concept.html
12. Noun-Phrase Analysis in Unrestricted Text for Information Retrieval. David A. Evans, Chengxiang Zhai.
http://acl.ldc.upenn.edu/P/P96/P96-1003.pdf
13. Evaluation of Syntactic Phrase Indexing -- CLARIT NLP Track Report
http://trec.nist.gov/pubs/trec5/papers/CLARIT-NLP-corrected.ps.gz
14. Natural Language Information Retrieval TREC-3 Report. Tomek Strzalkowski, Jose Perez Carballo, Mihnea Marinescu.
http://citeseer.ist.psu.edu/51110.html
15. SMART system stop word file
ftp://ftp.cs.cornell.edu/pub/smart/english.stop
16. YamCha: Yet Another Multipurpose CHunk Annotator
http://chasen.org/~taku/software/yamcha

7. Appendix

7.1. Usage

1. Environment setup.

```
cd <submission directory>
source setenv.csh       # for csh
      # or
. setenv.sh             # for sh
```

2. Collection creation.

```
Usage: java indexer <input_corpus_file> <output_index_dir>

E.g. java indexer cran.all c
```

3. Query transformation.

```
Usage: java QueryTransformer [-cleanse] [-np] [-wordnet] -in <input_file>
                                          -out <output_file>
    where:
        -cleanse – do POS-based query cleansing
        -np – add noun phrases
        -wordnet – expand synonyms using WordNet
        input_file – original query file (query.text)
        output_file – transformed query file
```

For example:

```
java QueryTransformer -cleanse -wordnet -in query.text -out query_best.text
```

Note:

a. The best results are achieved on "-cleanse –wordnet". The output file of this is included in the submission (query_best.text).
b. It takes 4-5 minutes to run on elaine.

4. Search.
   Usage: java runqueries <index_dir> <query_file> <qrel_file>

   E.g. java runqueries c query_best.text qrels.text

   The program prints results for every query followed by the average precision for all queries.

7.2. Files

7.2.1. Source files

QueryTransformer.java – query transformation code.
StemmedAnalyzer.java – Lucene analyzer that uses Porter stemmer.
indexer.java – collection indexer (mostly CS276A PE1 starter code).
runqueries.java – the search program (mostly CS276A PE1 code).
synonyms.java – simple WordNet synonym test program.
makefile – builds the source code.

7.2.2. Data files

cran.all – documents to index (link to /usr/class/cs276a/pe1/cran.all).
qrels.text – correct results (link to /usr/class/cs276a/pe1/qrels.text).
query.text – original query file (link to /usr/class/cs276a/pe1/query.text).
query_best.text – transformed query file (after cleaning and adverb synonym expansion).

7.2.3. Configuration files

setenv.csh – sets up csh build/runtime environment.
setenv.sh – sets up sh build/runtime environment.
wordnet_properties.xml – WordNet property file.

7.2.4. Binary files

There are also binary files, which were too big to submit. They were left in /afs/ir.stanford.edu/users/o/l/olegs/public/cs224n/bin. The setenv.* files are configured to locate them there.

englishPCFG.ser.gz – Stanford parser English grammar.
javanlp.jar – Stanford parser library.
jwnl.jar – Java WordNet API (link to /afs/ir/class/cs276/software/jwnl/jwnl.jar).
lucene-1.4-final.jar – Lucene library (link to /afs/ir/class/cs276a/software/lucene-1.4-

final/lucene-1.4-final.jar).

7.2.5. Report

fp.doc – this document.