Information Needs and Automatic Queries

Experiments for the TREC 2004 Genomics AdHoc Retrieval Task

Richard M. Tong
Tarragon Consulting Corporation
1563 Solano Avenue, #350
Berkeley, CA 94707
rtong@tgncorp.com

Abstract

Tarragon Consulting Corporation participated in the adhoc retrieval task of the TREC 2004 Genomics Track. We used a standard deployment of the K2 search engine from Verity, Inc. in which we exploited the free-text query parser to interpret the information need statements provided in the task. The primary goal of our participation was to establish a performance baseline using "of-the-shelf" tools and then to explore how knowledge-based extensions could enhance performance. Time and resource constraints prevented us from performing the knowledge-based experiments, but our official submissions show that reasonable performance can be achieved using just our baseline strategy.

Overall Approach

In our approach, we emphasize the use of "off-the-shelf" tools to provide a baseline capability and then explore ways in which custom algorithms and technologies can be used to enhance performance. For the TREC 2004 Genomics AdHoc Retrieval Task, we used Verity's K2 search engine (see: http//www.verity.com/ for basic information about the K2 family of products) both to index the documents and to provide a baseline interpretation of the test topics.

To build the collection for the adhoc experiments, we created a minimal XML variant of the PubMed records in the test set (see Figure 1), and indexed them using the standard K2 indexer.

To create the test queries, we use the standard K2 free-text query parser to convert each element of the statement of information need into a K2 query fragment, and then combined these into an overall query topic that used the components in different ways. In particular, we experimented with giving different importance weights to the three elements of the topic.

For example, Topic 1 as provided by NIST is:

```
<TOPIC>
<ID>1</ID>
<TITLE>Ferroportin-1 in humans</TITLE>
<NEED>Find articles about Ferroportin-1,
an iron transporter, in humans.</NEED>
<CONTEXT>Ferroportin1 (also known as
SLC40A1; Ferroportin 1; FPN1; HFE4;
IREG1; Iron regulated gene 1; Iron-
regulated transporter 1; MTP1; SLC11A3;
and Solute carrier family 11 (proton-
coupled divalent metal ion transporters),
member 3) may play a role in iron
transport.</CONTEXT>
</TOPIC>
```

To process this into a K2 query, we first separate out the <TITLE/>, <NEED/> and <CONTEXT/> elements. Then for each element, we: (1) remove any leading noise phrases that match entries in a library of pre-built patterns developed from an analysis of previous TREC topics; (2) add a period to the end if there is no terminating punctuation; and, (3) map everything into upper-case.

For Topic 1 this gives:

```
<TITLE>FERROPORTIN-1 IN HUMANS.</TITLE>

<NEED>FERROPORTIN-1, AN IRON TRANSPORTER,
IN HUMANS.</NEED>

<CONTEXT>FERROPORTIN1 (ALSO KNOWN AS
SLC40A1; FERROPORTIN 1; FPN1; HFE4;
IREG1; IRON REGULATED GENE 1; IRON-
REGULATED TRANSPORTER 1; MTP1; SLC11A3;
AND SOLUTE CARRIER FAMILY 11 (PROTON-
COUPLED DIVALENT METAL ION TRANSPORTERS),
MEMBER 3) MAY PLAY A ROLE IN IRON
TRANSPORT.</CONTEXT>
```

The next step is to pass the content of each element to Verity's free-text query parser. This does a minimal amount of phrase detection and noise word removal, and then automatically creates a query fragment that models the various ways in which the words and phrases might be expected to appear.

For example, in the <NEED/> element, the parser recognizes that Ferrorportin-1 might appear as "Ferroportin-1" or "Ferroportin 1", detects "iron transporter" as a phrase, and determines that "humans" is a key term.

From this information, the free-text query parser generates the fragment:

```
<Accrue>
* 0.55 <Many><Any>
** <Many><Stem>
    /wordtext = "FERROPORTIN-1"
** <Many><Phrase>
*** <Many><Stem>
    /wordtext = "FERROPORTIN"
*** <Many><Stem>
    /wordtext = "1"
** <Many><Phrase>
*** <Many><Stem>
    /wordtext = "FERROPORTIN"
*** <Many><Stem>
    /wordtext = "-"
*** <Many><Stem>
    /wordtext = "1"
* 0.55 <Many><Stem>
    /wordtext = "HUMANS"
* 0.55 <Sum>
** 0.40 <Many><Phrase>
*** <Many><Stem>
    /wordtext = "IRON"
*** <Many><Stem>
    /wordtext = "TRANSPORTER"
** 0.10 <Near/4>
*** <Many><Stem>
    /wordtext = "IRON"
*** <Many><Stem>
    /wordtext = "TRANSPORTER"
** 0.50 <Accrue>
*** 0.75 <Many><Stem>
    /wordtext = "IRON"
*** 0.75 <Many><Stem>
    /wordtext = "TRANSPORTER"
* 0.55 <Accrue>
** 0.55 <Many><Paragraph>
*** <Many><Any>
**** <Many><Stem>
    /wordtext = "FERROPORTIN-1"
**** <Many><Phrase>
***** <Many><Stem>
    /wordtext = "FERROPORTIN"
***** <Many><Stem>
    /wordtext = "1"
**** <Many><Phrase>
***** <Many><Stem>
    /wordtext = "FERROPORTIN"
***** <Many><Stem>
    /wordtext = "-"
***** <Many><Stem>
    /wordtext = "1"
*** <Many><Stem>
    /wordtext = "HUMANS"
** 0.55 <Many><Paragraph>
*** <Many><Stem>
    /wordtext = "HUMANS"
*** <Phrase>
**** <Many><Stem>
    /wordtext = "IRON"
**** <Many><Stem>
    /wordtext = "TRANSPORTER"
** 0.55 <Many><Paragraph>
*** <Many><Any>
**** <Many><Stem>
    /wordtext = "FERROPORTIN-1"
**** <Many><Phrase>
***** <Many><Stem>
    /wordtext = "FERROPORTIN"
***** <Many><Stem>
    /wordtext = "1"
**** <Many><Phrase>
***** <Many><Stem>
    /wordtext = "FERROPORTIN"
***** <Many><Stem>
    /wordtext = "-"
***** <Many><Stem>
    /wordtext = "1"
*** <Many><Stem>
    /wordtext = "HUMANS"
*** <Phrase>
**** <Many><Stem>
    /wordtext = "IRON"
**** <Many><Stem>
    /wordtext = "TRANSPORTER"
```

which is a complete description, in Verity's OTL format, of how to search for various combinations of "Ferroportin-1", "iron transporter" and "humans".

The details of this structure are not as significant as the fact that it is automatically generated, and makes use of various operators in the Verity query language (e.g., <Accrue>, <Sum>, <Phrase>), as well as weights on the major sub-components of the structure.

The final step in the automatic query generation process is to combine the fragments into an overall K2 query to run against the collection. We experimented with a number of ways of doing this.

For our official submissions, we used two methods. In the first, labeled "tgnSplit", we combined the fragments using a simple summation model, giving differential weights to the three elements. So using this model, the top-level query for Topic 1 is:

```
tgn_Q1 <Sum>
* 0.50 _Q1_title <Accrue>
* 0.35 _Q1_need <Accrue>
* 0.15 _Q1_context <Sum>
```

In the second official submission, labeled "tgnNecaux", we used a more complex combination strategy. In this model, we first assess whether the <TITLE/> or <NEED/> query fragment matches, and then, if it does, we check to see if the <CONTEXT/> fragment matches. The scores from each fragment are then combined in another weighted sum. However, if neither of the <TITLE/> or <NEED/> fragments match, then we ignore any contribution from the <CONTEXT/> fragment.

The rationale for this general strategy is an assumption that the <TITLE/> and <NEED/> elements are the most indicative of the overall user information need, and that the <CONTEXT/> element primarily provides background and/or supporting information.

Results Analysis

The summary results for our official submissions are shown in Table 1, together with a run, labeled "tgnMerge", that treats the <TITLE/>, <NEED/> and <CONTEXT/> elements as a contiguous text, and that we scored ourselves.

The first official run, tgnSplit, performs somewhat above the average level, with a mean average precision (MAP) of 0.2319 compared to the mean MAP for automatic runs from the published results of 0.2227. This run produced individual topic results in which 24 queries got an average precision score greater than or equal to the published median, and 3 queries got the maximum average precision score.

The "precision at N" scores for tgnSplit show a similar behavior. The mean P@10 score is 0.4860 compared to the published mean for automatic runs of 0.4335; and the mean P@100 score is 0.2926 compared to 0.2691.

The second official run, tgnNecaux, does substantially worse, and the unofficial run, tgnMerge, is the worst of the three.

Our failure analysis did not uncover any systematic errors, so it seems clear from these results that all three elements of the query are potentially useful for adhoc retrieval, and also that a differential treatment of them is better than a merged view.

Figure and Tables

```
<PubmedArticle>
<PMID>10636771</PMID>
<DCOM>20000110</DCOM>
<TI>Bacterial bioluminescence: isolation and expression of the luciferase genes from Vibrio
harveyi.</TI>
<AB>Genes for the luciferase enzyme of Vibrio harveyi were isolated in Escherichia coli by a
general method in which nonluminous, transposon insertion mutants were used. Conditions
necessary for light production in E. coli were examined. Stimulation of transcription of the
genes for luciferase (lux A and lux B) was required for efficient synethesis of luciferase.
To enhance transcription bacteriophage promoter elements were coupled to the cloned lux gene
fragments.</AB>
<RN>EC 1.13.12.- (Luciferase)</RN>
<MH>Escherichia coli/enzymology/genetics/physiology</MH>
<MH>Genes, Bacterial</MH>
<MH>Luciferase/*biosynthesis/genetics</MH>
<MH>Luminescence, Bacterial</MH>
<MH>Support, U.S. Gov't, Non-P.H.S.</MH>
<MH>Vibrio/enzymology/*genetics/physiology</MH>
</PubmedArticle>
```

Figure 1: Example <PubmedArticle/> XML Format

Table 1: Summary Results for Official Submissions and Selected Alternate Experiment

	tgnSplit	***tgnNecaux***	***tgnMerge***
# Docs Retrieved	50000	49290	50000
# Docs Relevant	8268	8268	8268
# Docs Rel_ret	3256	2884	2736
Interpolated R-P:			
at 0.00	0.7019	0.6367	0.6145
at 0.10	0.5084	0.4178	0.3325
at 0.20	0.3872	0.3065	0.2443
at 0.30	0.3055	0.2530	0.1431
at 0.40	0.2255	0.1902	0.1012
at 0.50	0.1785	0.1528	0.0625
at 0.60	0.1367	0.1302	0.0435
at 0.70	0.1210	0.1205	0.0288
at 0.80	0.1030	0.0911	0.0201
at 0.90	0.0831	0.0637	0.0155
at 1.00	0.0556	0.0331	0.0100
Average Precision	**0.2319**	**0.1951**	**0.1201**
Precision:			
at 5 docs	0.5120	0.4280	0.4160
at 10 docs	0.4860	0.4080	0.3760
at 15 docs	0.4600	0.3800	0.3627
at 20 docs	0.4400	0.3690	0.3360
at 30 docs	0.4113	0.3480	0.3067
at 100 docs	0.2926	0.2358	0.1846
at 200 docs	0.2091	0.1722	0.1332
at 500 docs	0.1104	0.0958	0.0835
at 1000 docs	0.0651	0.0577	0.0547
R-Precision	**0.2801**	**0.2446**	**0.1599**