Using Shallow Natural Language Processing in a Just-In-Time Information Retrieval Assistant for Bloggers

Ang Gao* and Derek Bridge

Department of Computer Science,
University College Cork, Ireland
ang.gao87@gmail.com, d.bridge@cs.ucc.ie

Abstract. Just-In-Time Information Retrieval agents proactively retrieve information based on queries that are implicit in, and formulated from, the user's current context, such as the blogpost she is writing. This paper compares five heuristics by which queries can be extracted from a user's blogpost or other document. Four of the heuristics use shallow Natural Language Processing techniques, such as tagging and chunking. An experimental evaluation reveals that most of them perform as well as a heuristic based on term weighting. In particular, extracting noun phrases after chunking is one of the more successful heuristics and can have lower costs than term weighting. In a trial with real users, we find that relevant results have higher rank when we use implicit queries produced by this chunking heuristic than when we use explicit user-formulated queries.

1 Introduction

No longer is the Web a place where we do little more than browse and search; it is a place where we increasingly author content too. In some cases, we are authoring traditional content, in the form of essays and reports, creating them through our browsers and storing them 'in the cloud', e.g. using GoogleDocs.[1] But the Web 2.0 era has brought new forms of user-authored content too, of which wikis and blogs are popular examples. For instance, in June 2009, the English Wikipedia contained nearly 3 million content pages;[2] and as of April 2007, the blogosphere contained over 72 million blogs, originating about 17 posts per second.[3] However, an information-seeking/information-authoring dichotomy is a false one. We frequently engage in both activities in tandem. When creating a blogpost, for example, we may interleave search with writing: we seek information to deepen the content; and we seek multimedia resources that we can embed or link to.

Switching between these two tasks (search and writing) and the different tools for accomplishing them (search engines and editors) both wastes time and imposes

* Supported by Science Foundation Ireland Principal Investigator Grant, 07/IN.1/I977

[1] http://docs.google.com

[2] http://en.wikipedia.org/wiki/Special:Statistics, accessed 04/06/2009

[3] http://www.sifry.com/stateoftheliveweb/, accessed 04/06/2009

L. Coyle and J. Freyne (Eds.): AICS 2009, LNAI 6206, pp. 103–113, 2010.

an extra cognitive load on the user [8]. Tools that offer integrated search and editing functionality can reduce the disruption. In particular, when a user edits her document, an integrated tool can proactively extract queries from the document, send them to search engines, and display the results alongside the document. In fact, this is the idea of Just-In-Time Information Retrieval (JITIR): information is retrieved proactively by the JITIR agent, based on queries that are implicit in, and formulated from, the user's current context, without the user formulating explicit queries [1,8,4]. The user's current context might be documents that the user has opened for reading or editing, the user's Web browsing history, or even aspects of physical context such as location and nearby people [7].

In this paper, we present experiments with the Blogoduct system, a JITIR agent for bloggers and other authors of short documents. The contribution of the paper is its consideration of different heuristics by which the agent can select terms to include in the implicit queries that it creates from the document that the user is editing. We propose and compare several heuristics that, to the best of our knowledge, do not feature in existing JITIR agents. Each of the heuristics that we compare uses some relatively lightweight techniques from the field of Natural Language Processing (NLP).

In Section 2, we review other JITIR agents, focusing on the way they create implicit queries. Section 3 describes our Blogoduct system and the heuristics that we have implemented for creating implicit queries. Section 4 reports an experimental comparison of our heuristics and results from a live user trial.

2 Related Work

The literature contains many descriptions of JITIR systems [1,8,6]. They may differ in several respects: the type of system (e.g. standalone desktop systems, plug-ins, etc.); the files over which they search (e.g. a user's local files, the Web, etc.); how they aggregate results from multiple sources or cluster the results; and how they present results to the user (e.g. separate result panes, transparent pop-up windows, etc.). But for the purposes of this paper, the difference we focus on is how implicit queries are created from the user's document.

The most common technique is to use weights based on term frequency (tf) within the document that the user is editing and inverse document frequency (idf) over the document collection. In fact, the way tf-idf weights are computed and used will depend on the operation of the search engine. Consider the situation where the search engine operates on a repository of local files. In this case, each document in the repository and the document that the user is editing can be represented by a term vector of tf-idf weights; the most similar documents according, e.g., to cosine similarity can be retrieved. This is the default technique used in Savant, which is the back-end of the Remembrance Agent, although Savant is also capable of parsing documents to discard mark-up and to extract special fields on which domain-specific similarity measures can be defined [8].

There is an alternative technique that uses tf-idf weights to create implicit queries, but does not use them during retrieval. Consider the situation where

retrieval is done by an Internet search engine. The agent does not supply the search engine with a term vector of tf-idf weights; instead it supplies just a set of keywords, like what any one of us would type into a search box. The search engine uses its matching and ranking algorithms to find the set of results. But how can the JITIR agent select terms from the user's document to include in the implicit query? An obvious approach is to use those with highest tf-idf weights. The complication comes in calculating idf: without access to the documents we cannot compute document frequencies. To overcome this, the search engine is queried with each individual term and the engine's estimate of the number of results is used as a proxy for the true document frequency, e.g. [3]. This is not the true document frequency because it fails to count multiple occurrences of the term in the same document, and it fails to eliminate double-counting that results from indexing duplicate copies of a document.

Watson, one of the best-known JITIR agents, is an example of this kind of system: it computes term weights to choose what to include in a query [1]. However, it has a bespoke term weighting algorithm that assigns higher weights to words that appear more frequently in the document, that appear earlier in the document, that are emphasized in the document (e.g. in italics, in a section heading), and so on. A subsequent version of Watson was also built that creates queries in a *task-specific* way; for example, for someone who is writing arguments for and against a point, domain knowledge is used to create two queries, a similarity query and an opposing query [2].

A rather different approach is taken by the system reported in [6]. It attempts to identify a document's topics, where candidate topics are WordNet word senses. It computes topic probability based on occurrences of on- and off-topic words in the document, relative to their occurrence under WordNet senses. It can create queries in one of two ways: either it creates one query per topic, where the query comprises all the words for the topic, or it creates a single query using tf-idf weights while ensuring that at least two topics are covered.

In the next section, we describe our system, Blogoduct, which uses shallow NLP to create implicit queries.

3 The Blogoduct System

Figure 1 shows, in simplified form, the architecture of Blogoduct, the Just-In-Time Information Retrieval system that we have built. While aimed at bloggers, it can support authors of other forms of content too. It proactively provides aggregated search results from Google and Yahoo!, along with image search results from flickr.[4]

Blogoduct is a client-server webapp. A screenshot of the client is shown in Figure 2. The client is an AJAX application, written in Java and compiled to optimized JavaScript using the Google Web Toolkit.[5] In her browser, the user sees a pane that contains a rich-text editor for creating her document, and a

[4] `www.google.com`; `www.yahoo.com`; `www.flickr.com`

[5] `http://code.google.com/webtoolkit/`

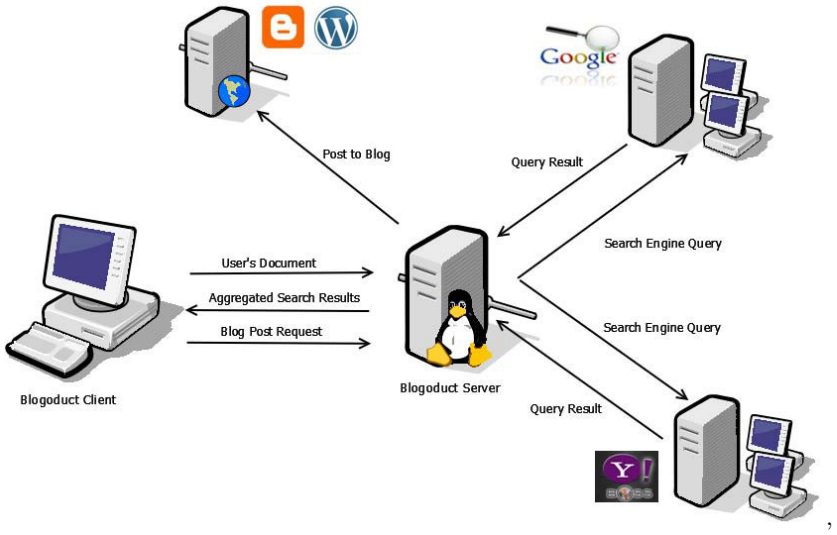

Fig. 1. The Blogoduct architecture (simplified)

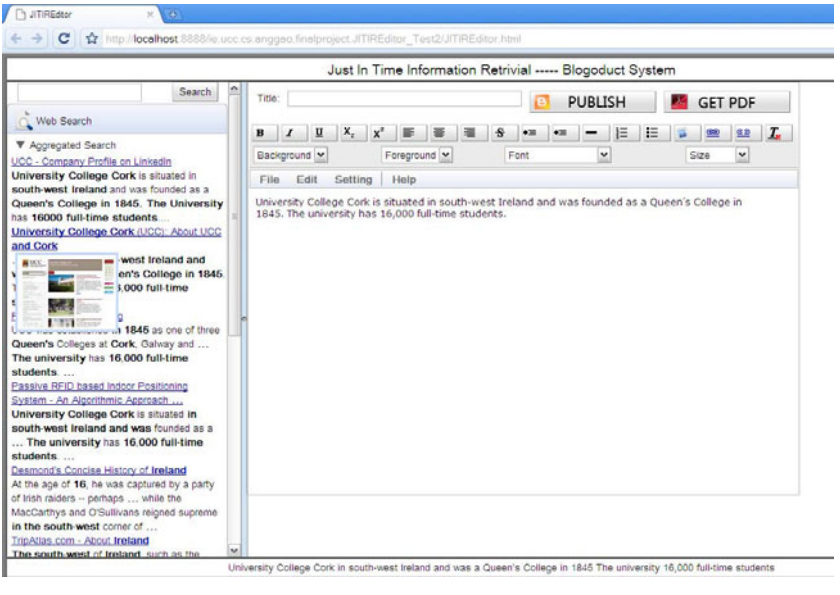

Fig. 2. A screenshot of the Blogoduct client

pane in which search results are listed. Other functionality includes a standard search box, and a button for publishing to the user's blog.

Server-side, Blogoduct is implemented as a Java servlet. The client communicates with the Blogoduct server using GWT Remote Procedure Calls. The server

receives a copy of the user's document, from which it can extract search engine queries. The client handles the server's responses asynchronously.

Ordinarily, the Blogoduct server uses Internet search engine APIs to obtain search results, in which case queries will be sets of keywords. Presently, we have implemented interfaces to the Google, Yahoo! and flickr search engines. The Google and Yahoo! results are aggregated; the flickr results are kept separate. However, for running off-line experiments, we can run the server software as a standard application, allowing access to local files. In this case, we can use the Lucene search engine,[6] and queries will be represented as term vectors of tf-idf weights. We exploit this possibility in the experiments we report in Section 4.1.

We experiment with five ways for Blogoduct to decide which terms from the user's document to include in a query. Each of the first four of these heuristics uses shallow NLP techniques available in the OpenNLP toolset:[7]

N: OpenNLP's part-of-speech (POS) tagger identifies the syntactic category (e.g. determiner, noun, verb) of each word. Our heuristic is then to extract all nouns, and create the query from these. The motivation for this is that nouns often carry a lot of the semantic content of a sentence.

Open Class: Our second heuristic also uses POS tagging, but this time we extract all 'open class' words. In linguistics, an open class is one that accepts the addition of new words, e.g. nouns, verbs, adjectives and adverbs but not prepositions, conjunctions, etc. In its effect, this strategy is not much different from simply removing stop-words.

NP: For this heuristic, we use chunking. Chunking is a form of lightweight parsing. In terms of depth of analysis, chunking sits somewhere between POS tagging and full parsing. It finds phrases within a sentence, but it does not aim to find a complete syntactic analysis. We use it to extract all noun phrases, and we use the union of the words in the noun phrases as the query. The chunker we use is OpenNLP's maximum-entropy-based chunker.

Name Detection: Name detection is the identification of the names of entities in a sentence, such as names of people, locations, dates, and so on. These will often be among the most important noun phrases in a sentence. We use OpenNLP's maximum-entropy-based name finder for this.

TFIDF: For comparison purposes, the final heuristic does not use OpenNLP tools. In the fashion described in the previous section, we compute tf-idf weights for each word in the user's document (using Google's estimates of result set size as proxies for document frequencies), and we take the highest scoring 50% of the words to be the query.

In fact, we tried five other heuristics, some of which were combinations of those shown above. Space limitations prevent us from including them in this paper. Their performance in the experiment, in any case, lay somewhere between that of the other heuristics.

The queries produced by the first four of our heuristics for a sample sentence are shown in Table 1.

[6] `http://lucene.apache.org`

[7] `http://opennlp.sourceforge.net/`

Table 1. The queries that the different heuristics produce for the sentence *"The Irish construction industry lurched downwards again in May"*

N	*construction industry May*
Open Class	*Irish construction industry lurched downwards again May*
NP	*The Irish construction industry May*
Name Detection	*May*

4 Experimental Evaluation

Evaluation of JITIR systems up to now has either required human users to judge a single version of the system (e.g. [1,8]) or, where comparisons of system variants has been done, it is done over a small number of variants and on very small document collections (e.g. [6]). We report a system trial with human users and Internet search in Section 4.2. But first we report a precision-recall evaluation, based on two medium-size document collections and using Lucene as our back-end search engine.

4.1 Comparison of Query Extraction Heuristics

The CISI document collection consists of 1460 documents from the Institute of Scientific Information and has 76 statements of information-needs with relevance judgments; the CACM document collection consists of 3204 documents and 52 statements of information-needs with relevance judgments. Note that, for the most part, the statements of information-needs are short paragraphs. We used these as if they were the document that the user was editing.

Our experimental methodology must simulate the idea of a user whose document grows as she authors it, and we have to repeatedly obtain search engine results from Blogoduct as the document grows. We did this at the sentence level. So, for example, if a statement of information-needs contains n sentences, then we invoke Blogoduct on ever larger prefixes of the information-needs: we begin by invoking it on the first sentence; we invoke it again on the first two sentences; then the first three sentences; and so on, until we have invoked it n times. Each time we invoke it, Blogoduct takes the latest prefix of the information-needs, extracts a query using one of the heuristics, and retrieves results from the document collection using Lucene. Of course, different information-needs contain different numbers of sentences (n), and so they do not necessarily invoke Blogoduct the same number of times.

We report 11-point interpolated average precision [5]. For each of 11 recall levels (0.0, 0.1, 0.2,...,1.0), this involves computing the mean over a set of queries of the interpolated precision at that recall level. Interpolated precision at recall level r, $p_{interp}(r)$, is defined as the highest precision found for any recall level:

$$p_{interp(r)} = \max_{r' \geq r} p(r') \tag{1}$$

with the usual definitions of precision and recall [5].

Figures 3 and 4 show plots of interpolated average precision for the CISI and CACM document collections respectively. We see that Name Detection is the least successful heuristic for both CISI and CACM. It is also one of the more computationally expensive, taking as much as 1 second to extract the names from a document. Extracting just nouns (N) is slightly worse than the other heuristics in the case of the CACM document collection. The other heuristics perform similarly to each other: chunking to extract noun phrases (NP) performs best and especially at low recall values, which implies that documents it retrieves to which it assigns high rank are good, but the differences are not significant. Note that TFIDF, while competitive, also has the disadvantage of high costs: recall that document frequencies for each term are obtained from Google.

4.2 User Evaluation

On the basis of the experimental results from the previous section, we decided to adopt the NP heuristic (i.e. chunking to extract noun phrases) for a trial with human users and Internet search engines. We prepared two systems. One is the full Blogoduct; in the other we disabled proactive querying, so a user who wants search results must resort to entering explicit queries through the standard search box. Below, we refer to these respectively as *Implicit* and *Explicit*.

As already mentioned, we adopted the NP heuristic in the Blogoduct system. However, Internet search engines treat keywords conjunctively; if a query to a search engine comprises the text of too many NPs, there may be few, if any, search results. This is not generally the case when queries are represented by term vectors and presented to Lucene. Hence, when Blogoduct is used with Internet search engines, it works in the following way: (a) queries comprise only the text of a maximum of the 10 most recent noun phrases, and (b) if there are fewer than 8 results, we repeatedly reduce the number of noun phrases by one, in first-in-first-out fashion.

We recruited 20 University College Cork students from different departments; each had different levels of search engine experience. We prepared a list of topics (e.g. a recent research result in Physics; Banff National Park), and we asked each participant to pick two topics about which they knew little. We asked ten of the students to write a paragraph on the first topic using full Blogoduct (Implicit), and a paragraph on the second topic using the other version of the system (Explicit). The other ten students did the same but using the systems in the opposite order.

In tasks that were completed using Implicit, users clicked on a total of 73 search results, with the average per task being 3.65. When using Explicit, users clicked on a few more results: 77 in total; 3.85 on average. But what is interesting is the position in the search results of the items being clicked on. This is shown in Figure 5. It seems that relevant web pages are ranked higher on average by

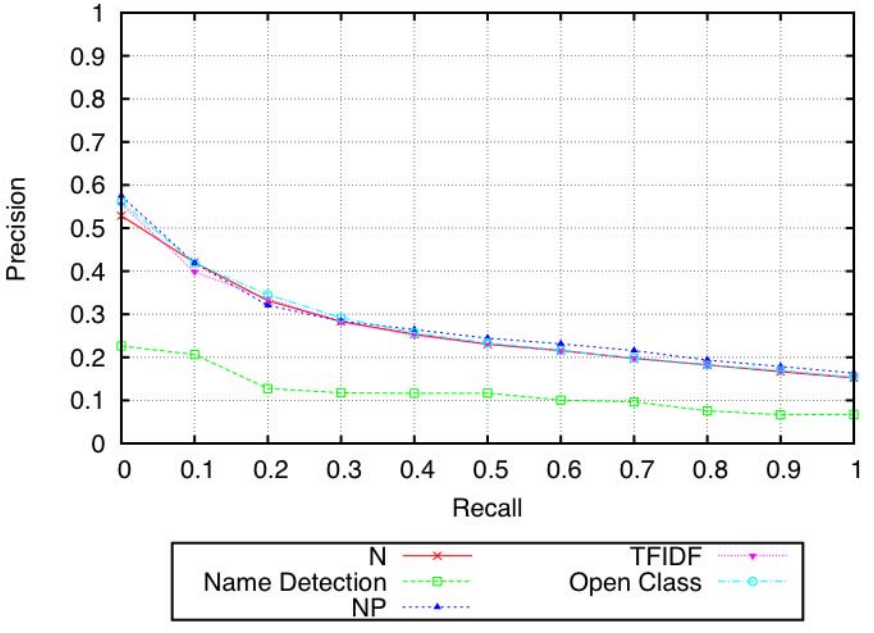

Fig. 3. Interpolated average precision for the CISI document collection

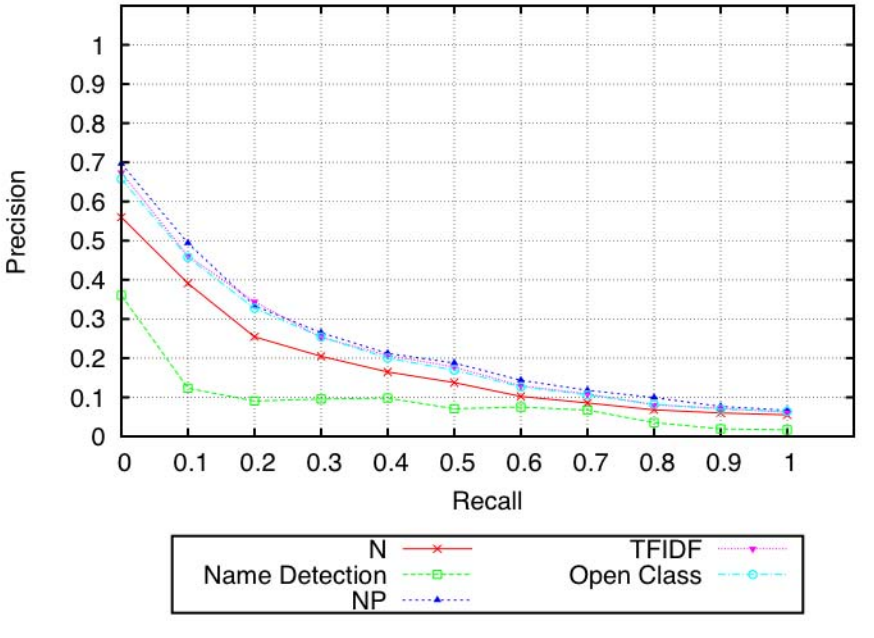

Fig. 4. Interpolated average precision for the CACM document collection

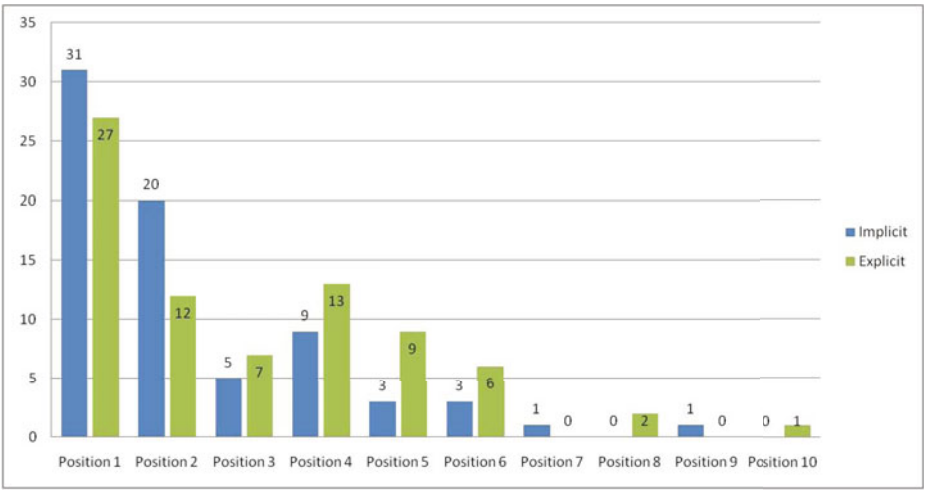

Fig. 5. Click History

Implicit than Explicit (average position of 2.342 compared with 3.0), implying that the queries that Blogoduct automatically creates are better than those that users create explicitly. We also recorded task times (from first click on a search result to final click on a search result), and they were better for Implicit: the average task time for Implicit was 21 minutes, whereas that for Explicit was 24.75 minutes. A follow-up survey seemed to confirm these results. Responses to two of the questions that we asked are given in Figures 6 and 7.

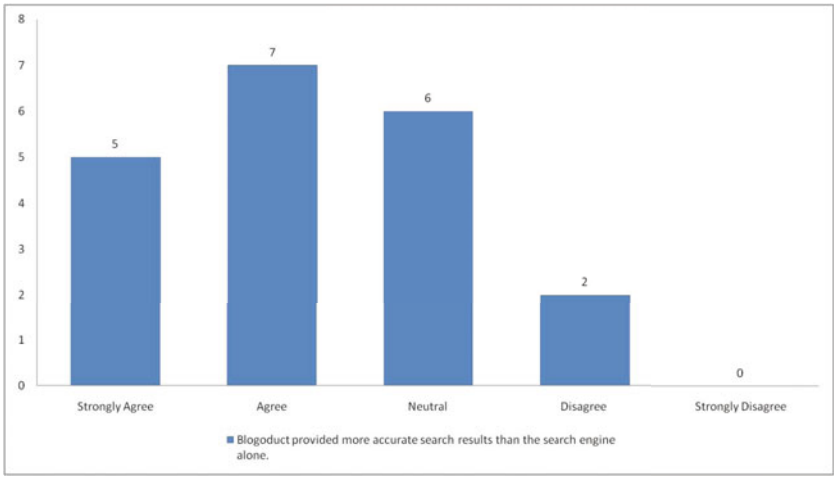

Fig. 6. Survey Responses: Accuracy

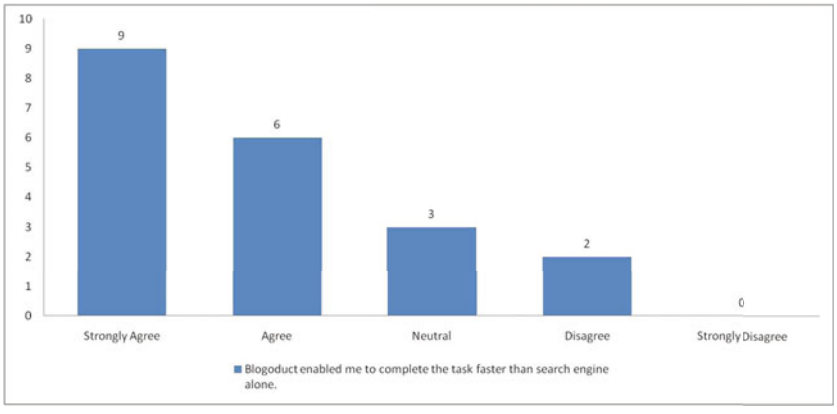

Fig. 7. Survey Responses: Time

5 Conclusions and Future Work

Blogoduct is a JITIR system that is unusual in that it uses shallow NLP techniques to form its implicit queries. In particular, we adopted a strategy of chunking and extracting recent noun phrases, which was competitive in an experimental evaluation on two document collections. In a trial with real users, Blogoduct's automatic, implicit queries produced result lists where relevant items tended to be more highly-ranked than they were when users formulated queries explicitly.

There are many lines of possible future inquiry. First, we would like to take the kind of comparisons that we performed using Lucene and repeat them in the case where the back-end uses Internet search. This, however, raises problems about getting relevance judgments. Second, we would like to compare the NLP techniques with more of the techniques found in systems such as Watson. Third, we believe that JITIR is an ideal scenario to incorporate personalization: users should not see the same results for the same queries. And fourth, we would like to see whether shallow NLP can be used to improve proactive recommendation of blogpost tags.

JITIR systems are seeing real use. For example, Zemanta[8] is a Firefox extension that retrieves ideas for blogpost enrichment from 16 common web services. We hope that in the future ideas from our research can be incorporated into systems such as Zemanta.

References

1. Budzik, J., Hammond, K.: Watson: Anticipating and contextualizing information needs. In: Procs. of the 62nd Annual Meeting of the American Society for Information Science, pp. 727–740 (1999)

[8] `www.zemanta.com`

2. Budzik, J., Hammond, K.J., Birnbaum, L., Krema, M.: Beyond similarity. In: Working notes of the AAAI Workshop on AI for Web Search (2000)
3. Henzinger, M., Chang, B.-W., Milch, B., Brin, S.: Query-free news search. In: Procs. of the 12th Intl. World-Wide Web Conference, pp. 1–10 (2003)
4. Lieberman, H., Fry, C., Weitzman, L.: Exploring the web with reconnaissance agents. CACM 44(8), 69–75 (2001)
5. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
6. Meade, J., Costello, F., Kushmerick, N.: Inferring topic probability for just-in-time information retrieval. In: Bell, D.A., et al. (eds.) Procs. of the 17th Irish Conference on Artificial Intelligence & Cognitive Science, pp. 103–112 (2006)
7. Rhodes, B.: Using physical context for just-in-time information retrieval. IEEE Trans. on Computers 52(8), 1011–1014 (2003)
8. Rhodes, B.J.: Just-In-Time Information Retrieval. PhD thesis, Massachusetts Institute of Technology (2000)