Predicting the Benefits of Information Retrieval Techniques

Andrew Arnt*

30th August 2001

Abstract

Classically, information retrieval systems have been static, meaning they perform the same sequence of actions on every query. In this paper we examine the possibility of using machine learning techniques to create a dynamic information retrieval system. Specifically, it is shown that by using artificial neural networks to predict whether or not a particular information retrieval technique will be helpful on a given query, it is possible to improve the overall quality of documents returned by the system and reduce the CPU time necessary to do the retrieval. To do this we introduce threshold training, where neural networks are trained on a select subset of the available instances in order to improve prediction performance.

1 Introduction

Information retrieval (IR) systems are made of several components. These components perform various tasks, such as query formation, query evaluation, precision improvement, recall improvement, clustering, and results visualization. Currently, the decision to use a new IR technique in a system is made off-line by the IR system designer based on laboratory tests. In these tests, the new technique is run on a test collection consisting of a set of queries and a set of documents. Each document has been manually tagged with relevance judgments for each query in the test collection. These specify if that document satisfies the information need of each query. If the new technique is found, on average, to improve some evaluation measure substantially, then it is added to the system[1]. Note that even though the average performance is improved, it is still possible that performance will be degraded for some queries.

What if we could decide online, for each query coming into our system, whether or not to use a certain IR technique? There are two obvious advantages to this ability:

- Overall retrieval performance can be improved. We could use the technique only on those queries where retrieval performance is improved, and do not use the technique when it would hurt retrieval.
- Computation time can be saved. When we only use the technique on queries where it would provide substantial retrieval improvement, we do not "waste" time running the technique on queries that the technique would not improve.

This paper discusses our attempt at this online prediction on one IR technique, Local Context Analysis (LCA)[9]. Section 2 provides some background on IR and the LCA technique. Section 3 describes how we do the online prediction. Sections 3 and 4 give our results. Section 5 describes another potential application of online query prediction.

*University of Massachusetts

2 Background

The goal of an information retrieval system is as follows: given a query, it should return to the user a set of documents from a large document collection, each of which answer the information need posed by the query. A document that satisfies the information need of a query is called "relevant". Two basic measures of retrieval performance are used in the IR community:

Recall is the percentage of relevant documents in the entire collection that are retrieved by the system.

Precision is the percentage of documents retrieved by the system that are relevant.

It is desirable, however, to have a single number that describes the performance of a system. One of the most often used evaluation measures is Average Precision. Buckley and Voorhees define Average Precision as the mean of the precision scores obtained after eaach relevant document is retrieved, using zero as the precision for relevant documents that are not retrieved. It has been shown to be both stable and a good indicator of overall system performance[1].

2.1 Local Context Analysis

One of the major obstacles IR systems must overcome is the word mismatch problem[8], which refers to the fact that users of IR systems often use different words in their queries to describe the same information need. This can lead to many relevant documents not being retrieved by the system. For example, the user's query may use the word "car", but some documents might only use the word "automobile," and therefore would not be retrieved. One of the best ways to overcome the word mismatch problem is through query expansion, where a user's query is expanded by adding new terms to it before performing the retrieval step.

The best terms to add during query expansion would be those that are closely related to the original query and would cause the retrieval process to bring in more relevant documents. LCA presents a good method for doing this. The process is described in Figure 1:

- First, the user's query is run through InQuery, an IR system developed at the University of Massachusetts[2]. InQuery returns a list of documents ranked in order of estimated relevance to the query.
- The top n documents are selected from the list, and are analysed to find concepts (noun groups). Each concept is given a weight based on co-occurrence of query terms with that concept and the IDF (see section 3.1 for a definition) values of the concept. Essentially we are finding those noun phrases that occur very often near query terms in the top ranked documents, while not occurring with high frequency in the overall document collection. The concepts are ranked in order of decreasing weight.
- The top m concepts are merged into the user's original query, creating a new, larger query.
- The new query is run again through InQuery, and the documents retrieved are returned to the user.

We use $n = 100$ and $m = 26$, as these values have produced good empirical results. An example expansion using LCA is shown in Figure 2[9].

On average, LCA has been shown to improve retrieval performance. For example Xu and Croft show that LCA improves the average precision from 25.2% to 31.1% on the 49 TREC4 collection queries[9]. However, LCA can also hurt retrieval on a single given query tremendously. For example, if very few of the documents returned in the initial pass of InQuery are relevant to the original query, then the concepts that are added to the query will be mostly noise, causing the retrieval performance to drop significantly. Out of the 49 TREC4 collection queries, LCA hurts performance on 11. Thus, it is advantageous to be able to use LCA on a per-query basis.

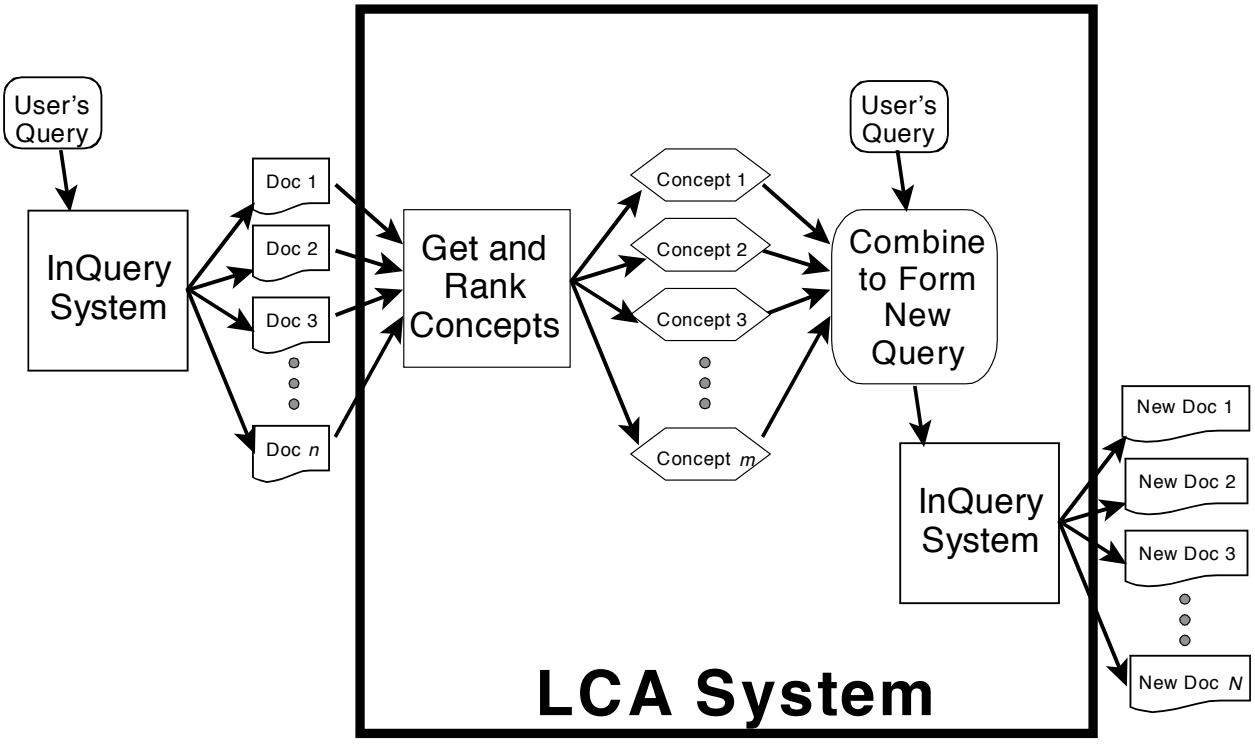

Figure 1: Diagram of the LCA process

3 Methodology

We want to predict, given a query, if using LCA will improve or hurt the retrieval performance. This prediction must be very fast, since it happens online. A sensible thing to do is to first create a set of training queries. The actual average precision obtained by both InQuery and InQuery+LCA on these queries can be computed by using the tagged document sets described above. The training data then can be fed into a machine learning algorithm, training it to output if LCA will improve the average precision for the input query. Artificial neural networks (ANNs) are well suited for this problem, as they are robust with respect to complex, noisy training data and have near-instantaneous evaluation of the learned target function[6].

3.1 The Data Set

For our queries and tagged document sets, we made use of the TREC queries and collections as described in Table 1[7]. The shortest formulations of the TREC queries (the title fields) were chosen, as they are most typical of the average query given to a web-based search engine. Furthermore, shorter queries have a higher potential for word mismatch, so they allow LCA expansion to make more of a difference in retrieval performance. Because ANNs require numerical inputs, we must use statistics derived from each query in lieu of the actual query. These are the 'query features'. Those features considered are as follows:

Query Length: The length of the query after stopwords[1] have been removed. Since shorter queries have more potential for useful terms to be added via expansion, it seems logical that LCA would be more

[1] A list of very common words adding no meaning to the query: e.g. the, with, but, said, *etc.*

hypnosis	brain-wave	ms.-burns
technique	pulse	reed
brain	ms.-olness	trance
hallucination	process	circuit
van-dyck	behavior	suggestion
case	spiegel	finding
hypnotizables	subject	van-dyke
patient	memory	application
katie	muncie	approach
study	point	contrast

Figure 2: Example LCA concepts for the query "What are the different techniques used to create self induced hypnosis?"

Query Numbers	Doc. Collection
51-100	TREC-2
101-150	TREC-3
151-200	TREC-3
201-250	TREC-4
251-300	TREC-5
301-350	TREC-6
351-400	TREC-7

Table 1: Queries and document collections used in our experiments.

likely to result in increased performance in shorter queries as opposed to longer queries. This feature requires negligible online computation time to determine.

IDF Statistics: The Inverse Document Frequency (IDF) of a given term is defined as $log(\frac{D}{d})$ where D is the total number of documents in the collection, and d is the number of documents in the collection that contain that term. The IDF is high for commonly occurring terms, and low for rare ones. The IDF is determined for each term in the query (with stopwords removed), and the Mean IDF, the Max IDF, and the Min IDF over the query terms are computed. If the IDF values for each term in the vocabulary are cached, then computing these features also takes negligible online computation time.

LCA Weights: These are the 26 weights computed by the "Get and Rank Concepts" phase of LCA. Determining these features takes significant computation time, as an entire InQuery retrieval is performed, noun phrases are identified and tagged, and weights are computed. However, if the system decides not to use LCA, the returned documents from InQuery are already available, and therefore it is not necessary to run InQuery again.

LCA Weight Statistics: Once the 26 LCA weights are obtained, various statistics based on them are computed in an attempt to help the ANN training. We consider the Mean, Median, and Standard Deviation of the following: All weights, top 5 weights, top 10 weights, middle 6 weights, middle 12 weights, bottom 10 weights, bottom 5 weights. Computing all of these requires little computation time beyond computing the weights themselves.

3.2 Training Method

The NevProp4 neural network simulator was used in our experiments[4]. Many networks were created and trained using various combinations of the inputs described above. Each network was fully connected with a single layer of symmetric logistic hidden units (with range -0.5 to 0.5). For each combination of inputs, three networks were created with the number of hidden units set to be equal to either the number of inputs, half the number of inputs, or a quarter of the number of inputs. The output of the network is dichotomous: the output should be 0 if the network predicts that LCA will not improve the average precision, and 1 if it will. Therefore, a single asymmetric logistic output unit with range 0.0 to 1.0 is used.

Half of the 350 queries were selected to use as training data. The quickprop algorithm (a variation of standard back-propagation) was used to update the network weights[3]. To avoid overfitting of the training data, we used cross-validation with a 10% holdout. This means that 17 queries randomly picked from the training set are set aside to use as 'temporary' testing data, and the neural net is trained on the remaining 158 until the classification error on the 'temporary' testing queries begins to increase. This step is repeated 10 times (randomly picking 17 new queries to hold out each time) for each network to reduce the variance usually associated with cross-validation. The mean testing error over the 10 cross-validation runs is calculated. The network is then reset and retrained using all 175 training queries, stopping training when the mean testing error is achieved.

The best networks are then tested against the 175 queries not used in training as an indication of overall network performance.

3.3 Results

51-400 train	Inquery	Inq+LCA	PredAll	PredFast	Perfect
Num Hurt/Num Used		102/175	50/106	26/52	0/73
Avg. Prec.	0.2407	0.2546	0.2552	0.2464	0.2640
% over Inquery		+5.77	+6.02	+2.38	+9.68
% over Inq+LCA			+0.24	-3.20	+3.70
Avg CPU Time	5.240	16.921	10.448	8.096	8.364
51-400 test	**Inquery**	**Inq+LCA**	**PredAll**	**PredFast**	**Perfect**
Num Hurt/Num Used		114/175	60/93	40/53	0/61
Avg. Prec.	0.2314	0.2465	0.2380	0.2340	0.2537
% over Inquery		+6.51	+2.83	+1.08	+9.61
% over Inq+LCA			-3.46	-5.09	+2.91
Avg CPU Time	4.587	15.668	8.308	6.829	7.387

Table 2: Results for PredAll and PredFast

Two of the best performing networks are presented here. The first (PredAll) uses the concept weight median statistics, query length, and IDF mean as inputs with 10 hidden units. The second (PredFast) uses just the IDF min and max, with 2 hidden units. Notice that the second network does not use any of the query features that involve LCA, so it is not necessary to perform the "Get and Rank Concepts" portion of LCA to use this network to make the prediction. The results are presented in Table 2. "Num Hurt" shows how many times the decision to use LCA hurts retrieval performance on the query. The Inq+LCA column shows the results if LCA is used on every query, while the Perfect column shows the results if the predictions are made perfectly.

PredAll shows a small improvement over Inq+LCA on the training data, but does not improve over Inq+LCA on the testing data. PredFast does not improve over Inq+LCA on either data set. However, both

networks used much less CPU time than Inq+LCA, while providing a modest performance gain over using just InQuery. Not surprisingly, PredAll outperforms PredFast, but also uses almost 2 seconds more CPU time per query.

4 Threshold Training

For many of the queries it is observed that using LCA changes the average precision very little. These queries basically amount to noise when training an ANN. Furthermore, Sparck Jones suggests that small changes in precision (i.e. less than 5%) are unnoticeable to users of an IR system[5] . Perhaps performance can be improved by focusing attention on those queries where LCA changes the average precision greatly. We will train the network only on those queries where the average precision changes more than some percentage (we will try both 5% and 10%). A network trained in such a way should perform very well on test queries that are above the precision change threshold (or would be, if the actual precisions were known). Intuitively, it is anticipated that the network should predict somewhat well those new queries that are just under the threshold, and that the predictive ability degrades smoothly as the precision change decreases.

The same network parameters as described above are used in these experiments. The training set is created by first finding all queries where using LCA produces an above-threshold change in precision relative to using just InQuery. Then half of those above-threshold queries are randomly selected to be in the training set. The networks are trained using the cross-validation with early stopping method described in Section 3.2. The best networks are then tested on all remaining queries.

4.1 Threshold Training Results

51-400 $> \Delta 5\%$ train	Inquery	Inq+LCA	PredAll5	PredFast5	Perfect
Num Hurt/Num Used		8/40	1/33	6/38	0/32
Avg. Prec.	0.3376	0.4094	0.4279	0.4158	0.4313
% over Inquery		+21.26	+26.72	+23.16	+27.76
% over Inq+LCA			+4.50	+1.56	+5.35
Avg CPU Time	5.267	15.401	15.184	14.245	10.994
51-400 test	Inquery	Inq+LCA	PredAll5	PredFast5	Perfect
Num Hurt/Num Used		208/310	172/267	198/299	0/102
Avg. Prec.	0.2229	0.2300	0.2314	0.2300	0.2366
% over Inquery		+3.17	+3.77	+3.18	+6.10
% over Inq+LCA			+0.58	+0.01	+2.84
Avg CPU Time	4.868	16.410	16.517	15.867	7.473
51-400 $> \Delta 5\%$test	Inquery	Inq+LCA	PredAll5	PredFast5	Perfect
Num Hurt/Num Used		13/39	9/34	13/38	0/26
Avg. Prec.	0.3126	0.3522	0.3597	0.3496	0.3810
% over Inquery		+12.67	+15.05	+11.83	+21.89
% over Inq+LCA			+2.18	-0.74	+8.18
Avg CPU Time	4.577	11.957	11.745	11.811	9.567

Table 3: Results for PredAll5 and PredFast5

Results are presented in tables 3 and 4. We present testing results for both the entire test set and just those test queries above the threshold used for training. The best network for the 5% threshold case

51-400 > Δ10% train	Inquery	Inq+LCA	PredAll10	PredFast10	Perfect
Num Hurt/Num Used		3/18	0/15	1/16	0/15
Avg. Prec.	0.2661	0.3669	0.3975	0.3860	0.3975
% over Inquery		+37.90	+49.40	+45.06	+49.40
% over Inq+LCA			+8.33	+5.19	+8.33
Avg CPU Time	4.607	11.501	11.355	10.465	10.300
51-400 test	Inquery	Inq+LCA	PredAll10	PredFast10	Perfect
Num Hurt/Num Used		213/332	175/283	192/309	0/119
Avg. Prec.	0.2344	0.2442	0.2454	0.2451	0.2512
% over Inquery		+4.18	+4.69	+4.54	+7.20
% over Inq+LCA			+0.49	+0.35	+2.90
Avg CPU Time	4.930	16.555	16.676	14.796	7.744
51-400 > Δ10% test	Inquery	Inq+LCA	PredAll10	PredFast10	Perfect
Num Hurt/Num Used		4/17	3/16	4/17	0/13
Avg. Prec.	0.2735	0.3622	0.3706	0.3622	0.3915
% over Inquery		+32.45	+35.52	+32.45	+43.16
% over Inq+LCA			+2.32	0.00	+8.09
Avg CPU Time	5.613	17.091	17.156	17.091	12.340

Table 4: Results for PredAll10 and PredFast10

(PredAll5) used all concept weight statistics, the query length, and the IDF mean as inputs, with 7 hidden units. Without using LCA-derived features, the best network (PredFast5) used the IDF min and max, with 2 hidden units. In the 10% threshold case, the best network (PredAll10) used the concept weight mean and standard deviation statistics, the query length, and the IDF min and max, using 10 hidden units. The best network not using LCA-derived features (PredFast10) used the query length and the IDF min and max, with 3 hidden units.

These results are more encouraging, with all four networks showing modest performance improvements over Inq+LCA on both train and test data. However, notice that by restricting the training data, the resulting training sets are very small (40 queries for 5%, 18 for 10%). Furthermore, there are very few training instances where LCA does not improve the precision; LCA improves 32 out of 39 in the 5% case, and 15 out of 18 in the 10% case. This lack of negative training examples makes prediction more difficult. Presumably adding more queries above the threshold to the training set would further improve prediction accuracy.

Also notice that the 'fast' prediction networks did not save much CPU time in this case. The computation time saved in not computing the LCA concept weights was offset by the reduction in prediction accuracy. These networks predict that LCA improves nearly every query, resulting in many unnecessary uses of LCA.

5 Online Query Prediction in Progressive Processing

The progressive processing framework as described by Zilberstein *et al*, has been applied to information retrieval[10]. This framework allows IR components to be selected at run-time in order to allow the system to adapt to its environment. The environment includes such things as system load, number of queries waiting to be processed and how long they have been waiting. This IR system is then able to choose the best subset of information retrieval components so as to maximize the quality of the answers produced with respect to its limited computational resources. For example, if the system is under a heavy load, it can pick a set of

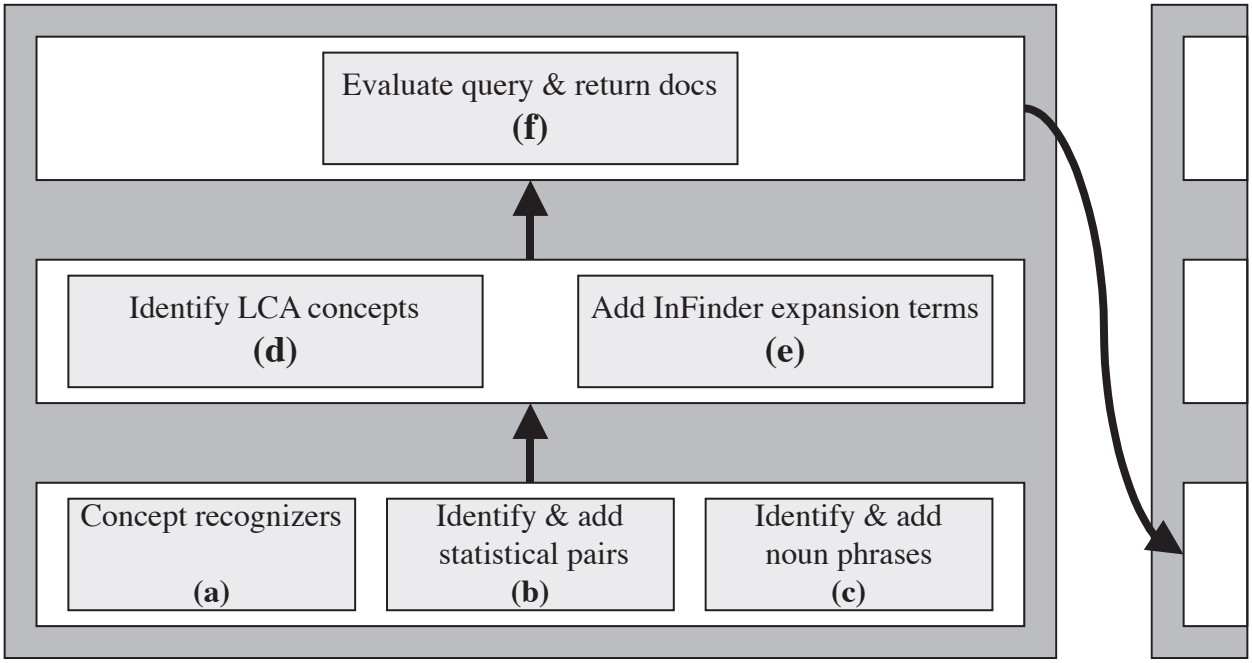

Figure 3: Illustration of a progressive processing task for an IR search engine

components which may not produce a great result, but do so in a reasonable amount of time.

This framework relies on an observable 'quality' of the incoming query and any intermediate results. Of course, quality is difficult to observe, as both the incoming query and intermediate results are represented as just a bag-of-words. Online query prediction as described above could be used to find an estimate of the quality at any point in the progressive processing task. It is shown in [10] that the representation of quality can be very coarse, so that being able to classify a query as simply 'bad','ok', or 'good' is sufficient to make effective use of the progressive processing framework.

6 Conclusions

We have shown that using online query prediction using threshold training can give provide a small improvement in retrieval performance. Future work should address two issues:

- More training queries are needed, especially more negative instances. Creating training queries is hardly a trivial task, as hundreds of manual relevance judgments have to be made. At the very least, future TREC conferences should provide more queries.
- The query features used were fairly arbitrary. We simply used features that were easy to collect and compute, that were not 'tuned' to the task of online query prediction. Perhaps with more carefully engineered features, prediction performance could be increased.

Although we investigate only LCA here, online query prediction could be a useful tool for many different IR techniques. Online query prediction is an important step in developing a truly dynamic information retrieval system. While the results presented here are modest at best, we feel that dynamic information retrieval systems present a previously unexplored method for improving both retrieval performance and

speed. Furthermore, under the progressive processing framework, we can take into account such things as system load and query priority when deciding to use some IR technique.

Acknowledgements

I thank Shlomo Zilberstein and James Allan for their helpful advice, ideas, and feedback, and Vikas Khandelwal for his assistance with the Inquery system.

This work was supported in part by the National Science Foundation under grants IIS-9907331 and INT-9612092, and by the Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not reflect the views of the NSF.

References

[1] Chris Buckley and Ellen M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 33–40, 2000.

[2] James P. Callan, W. Bruce Croft, and John Broglio. TREC and Tipster experiments with Inquery. *Information Processing and Management*, 31(3):327–343, 1995.

[3] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, Carnegie Mellon University, 1988.

[4] Philip H. Goodman. *NevProp software, version 4.* University of Nevada, Reno http://www.scs.unr.edu/nevprop, 1998.

[5] Karen Sparck Jones. Automatic indexing. *Journal of Documentation*, pages 393–432, 1974.

[6] Tom M. Mitchell. *Machine Learning.* McGraw Hill, New York, US, 1996.

[7] Ellen M. Voorhees. Overview of the eighth Text REtrieval Conference. In *Proceedings of TREC-8*, 1999.

[8] Jinxi Xu. *Solving the Word Mismatch Problem through Automatic Text Analysis.* PhD thesis, University of Massachusetts at Amherst, 1997.

[9] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, 1996.

[10] Shlomo Zilberstein, Abdel-Illah Mouaddib, and Andrew Arnt. Dynamic scheduling of progressive processing plans. In *ECAI 2000 - New Results in Planning, Scheduling and Design (PuK 2000)*, 2000.