Juru at TREC 2006: TAAT versus DAAT in the Terabyte Track

David Carmel, Einat Amitay
IBM Haifa Research Lab
Haifa 31905, Israel
Email: {carmel,einat}@il.ibm.com

Abstract

Our experiments focused this year on the ad-hock task of the Terabyte track. We experimented with **WAND**, a document-at-a-time evaluation algorithm we developed recently. Our results demonstrate the superiority of **WAND** over traditional term-a-time strategy while searching over a large collection such as gov2. We demonstrate how Web expansion can be successfully applied to significantly improve search results. In addition, we describe several schemes for creating manual queries, following this year's goal to enrich the pool of results by manual runs.

1 Introduction

The ad-hock task of the Terabyte track evaluates the ability of a search system to retrieve precise search results from a large collection in a reasonable time. Last year, due to scale limitations of our system, we distributed the .gov2 data into 10 sub-collections and applied federated search over them [4]. Following significant improvements in our search system, Juru [2], this is the first time that we were able to index and search the entire .gov2 collection using only one index.

Juru's original query processing is a standard term-at-a-time ($TAAT$) strategy as applied by many IR systems. During query execution, it maintains a heap for holding document partial scores. The algorithm orders the query terms by decreasing weight, according to their document frequency, and then traverses the infrequent terms first until the heap is full. After that, the algorithm moves to a "continue" mode for which it only updates already existing document accumulators in the heap. New non-scored documents in the remaining posting lists are ignored. During the continue mode posting lists are traversed very quickly by accessing only documents already found in the heap.

On the other hand, document-at-a-time ($DAAT$) strategies fully evaluate the document score by considering the contributions of all query terms with respect to the document before moving to the next document. While $TAAT$ strategies are by far most common in traditional IR systems due to the simplicity of their implementation, there are some clear advantages to ($DAAT$) strategies, especially when dealing with very large data sets. First, it is usually not feasible to maintain partial

results for all candidate documents in main memory. Second, I/O operations required for posting retrieval can be easily parallelized when using *DAAT* strategies, since all posting lists are traversed in parallel. Finally, advanced search features such as Boolean operators, proximity operators, and numeric range constraints, can be handled more efficiently by *DAAT* strategies, since all conditions can be evaluated at the same time to decide whether a document "satisfies" the query. It is thus clear that search engines which must handle context-sensitive queries over very large collections should prefer *DAAT* strategies. Indeed most Web search engines employ such strategies.

The main goal of our experiments this year, in the ad-hock task of the Terabyte track, was to implement a *DAAT* algorithm in Juru, and to optimally tune it for Web search. **WAND** [1] is a *DAAT* algorithm specifically designed for high precision search over very large collection, hence, it perfectly fits the Terabyte ad-hock task.

The rest of the paper is organized as follows. Section 2 shortly describes the **WAND** algorithm. A full description can be found in [1]. Section 3 describes our experiments with **WAND**, comparing its performance to our original *TAAT* algorithm. Section 4 describe several schemes for creating manual queries, following this year's goal to enrich the pool by submitting manual runs. Section 5 analyses the results of our official runs. Finally, Section 6 summarizes.

2 The WAND algorithm

The **WAND** algorithm is suited for *DAAT* strategies that evaluates queries using two levels of granularity. The algorithm iterates in parallel over query term postings and identifies candidate documents using a *preliminary evaluation* taking into account only partial information on term occurrences and no query independent factors. Once a candidate document is identified, it is *fully evaluated* and its exact score is computed. Furthermore, as in the standard *DAAT* approach, our algorithm iterates in parallel over query term postings but the nature of the preliminary evaluation is such that it is possible to skip quickly over large portions of the posting lists. If the result of this "fast and rough" evaluation is above a certain threshold, varied dynamically during the execution, then a *full evaluation* is performed and the exact score is computed.

Our approach allows both *safe optimization* and *approximate optimization*. For safe optimization, the two-level strategy makes no false-negative errors and thus it is guaranteed to return the top documents in the correct order and with accurate scores. For an approximate optimization, dynamic pruning techniques are used such that fewer documents pass the preliminary evaluation step, that is, we allow some false-negative errors at the risk of missing some candidate documents whose accurate scores would have placed them in the returned document set. The amount of pruning can be controlled by the user as a function of time allocated for query evaluation.

2.1 Document scoring

The final score of a document involves a textual score which is based on the document textual similarity to the query, as well as other query independent factors such as connectivity for web pages, citation count for scientific papers, inventory for e-commerce items, etc. We assume an additive scoring model, that is, the textual score of each document is determined by summing the contribution of all query terms belonging to the document. Thus, the textual score of a document d for query

q is:

$$\text{Score}(d, q) = \sum_{t \in q \cap d} \alpha_t w(t, d) \tag{1}$$

For example, for the $tf \times idf$ scoring model, α_t is a function of the number of occurrences of t in the query, multiplied by the inverse document frequency (idf) of t in the index and $w(t, d)$ is a function of the term frequency (tf) of t in d, divided by the document length $|d|$.

In addition we assume that each term is associated with an upper bound on its maximal contribution to any document score, UB_t such that

$$UB_t \geq \alpha_t \max(w(t, d_1), w(t, d_2), \ldots).$$

Thus, by summing the upper bounds of all query terms appearing in a document, we can determine an upper bound on the document's query-dependent score.

$$UB(d, q) = \sum_{t \in q \cap d} UB_t \geq Score(d, q). \tag{2}$$

At the heart of our approach, there is a Boolean predicate called **WAND** standing for Weak AND, or Weighted AND. **WAND** takes as arguments a list of Boolean variables $X_1, X_2, \ldots, X_k$, a list of associated positive *weights*, $w_1, w_2, \ldots, w_k$, and a threshold θ.

By definition, $\textbf{WAND}(X_1, w_1, \ldots X_k, w_k, \theta)$ is true iff

$$\sum_{1 \leq i \leq k} x_i w_i \geq \theta, \tag{3}$$

where x_i is the indicator variable for X_i, that is

$$x_i = \begin{cases} 1, \text{ if } X_i \text{ is true} \\ 0, \text{ otherwise.} \end{cases}$$

Given this setup, our preliminary scoring consists of evaluating for each document d, $\textbf{WAND}(X_1, UB_1, X_2, UB_2, \ldots, X_k, UB_k, \theta)$, where X_i is an indicator variable for the presence of query term i in document d and the threshold θ is varied during the algorithm as explained below. If **WAND** evaluates to true, then the document d undergoes a full evaluation.

The threshold θ is set dynamically by the algorithm based on the minimum score m among the top n results found so far, where n is the number of requested documents. The larger the threshold, the more documents will be skipped and thus we will need to compute full scores for fewer documents. It is easy to see that if the contribution upper bounds are accurate, then the final score of a document is no greater than its preliminary upper bound, and therefore all documents skipped by WAND with $\theta = m$ would not be placed in the top scoring document set by any other alternative scheme that uses the same additive scoring model.

However, (a) we might have only approximate upper bounds for the contribution of each term, (b) the score might involve query independent factors, and (c) we might want to use a higher threshold in order to execute fewer full evaluations. Thus in practice we will set

$$\theta = F \cdot m \tag{4}$$

where F is a *threshold factor* chosen to balance the positive and negative errors for the collection.

2.2 Setting the WAND Threshold

Assume that we wish to retrieve the top n scoring documents for a given query. The algorithm will maintain a heap of size n to keep track of the top n results. When a new candidate is returned by the WAND iterator, this document is fully evaluated using the system's scoring model resulting in the precise score for this document. If the heap is not full the candidate is inserted into the heap. If the heap is full and the new score is larger than the minimum score in the heap, the new document is inserted into the heap, replacing the one with the minimum score.

The threshold value that is passed to the WAND iterator is set based on the minimum score of all documents currently in the heap. Recall that this threshold determines the lower bound that must be exceeded for a document to be considered as candidate and to be passed to the full evaluation step.

The initial threshold is set based on the query type. For an OR query, or for a free-text query, the initial threshold is set to zero. The approximate score of any document that contains at least one of the query terms would exceed this threshold and would thus be returned as a candidate. Once the heap is full and a more realistic threshold is set, we only fully evaluate documents that have enough terms to yield a high score. For an AND query, the initial threshold can be set to the sum of all term upper bounds. Only documents containing all query terms would have a high enough approximate score to be considered candidates.

The initial threshold can also be used to handle mandatory terms (those preceded by a '+'). The upper bound for such terms can be set to some huge value, H, which is much larger than the sum of all the other terms upper bounds. By setting the initial threshold to H, only documents containing the mandatory term will be returned as candidates. If the query contains k mandatory terms, the initial threshold should be set to $k \cdot H$.

So far we have only described methods in which we are guaranteed to return accurate results for the query (safe evaluation). However, the threshold can additionally be used to expedite the evaluation process by being more opportunistic in terms of selecting candidate documents for full evaluation. In this case, the threshold would be set to a value larger than the minimum score in the heap. By increasing the threshold, the algorithm can dynamically prune documents during the approximation step and thus fully evaluate less overall candidates but with higher potential. The cost of dynamic pruning is the risk of missing some high scoring documents and thus the results are not guaranteed to be accurate. However, in many cases this can be a very effective technique. For example, systems that govern the maximum time spent on a given query can increase the threshold when the time limit is approaching thus enforcing larger skips and fully evaluating only documents that are very likely to make the final result list.

3 Experiments

Our experiments are based on the the .gov2 collection and on the 99 topics of the Terabyte tracks of 2004 and 2005. Short queries are based on topic titles while long queries are based on topic title plus description. We measure average query time by measuring the time required to retrieve the top 20 results (following the track's instructions). Precision is measured by p@5, p@10, and MAP, by retrieving the top 1000 results.

3.1 *TAAT* versus *DAAT*

In the first experiment we compared the original *TAAT* strategy, originally used by Juru, with the new *DAAT* strategy as implemented by **WAND**. For *TAAT* we used 2 different variants of heap size, 100K and 200K, which are reasonable for the expected number of results (1000). For **WAND** we used much smaller heaps of 1K and 2K (for query time evaluation we used a heap of size 20). The threshold factor F was fixed to 1.0 (see Equation 4). Both implementations used the same scoring mechanism of Juru. Table 1 shows the results in terms of query execution time and precision.

	Short Query				**Long Query**			
	time(sec)	P@5	P@10	MAP	time	P@5	P@10	MAP
TAAT 100K	2.9	0.53	0.49	0.25	11.8	0.53	0.523	0.25
TAAT 200K	3.3	0.53	0.49	0.25	12.3	0.53	0.523	0.25
WAND 1K	1.87	0.58	0.56	0.29	3.02	0.58	0.55	0.25
WAND 2K	1.87	0.62	0.58	0.31	3.02	0.59	0.56	0.27

Table 1. The search results of the original Juru's *TAAT* strategy compared to **WAND**, averaged over the 99 Terabyte topics.

The results clearly show the advantage of **WAND** compared to *TAAT* in terms of precision and search time. While handling a much smaller heap **WAND** enables to achieve much better results than the *TAAT* strategy. The main reason is the better pruning mechanism of **WAND** compared to the pruning process concealed in *TAAT*. There is also significant improvement in search time. Retrieving top 20 results by **WAND** only requires a heap of size 20 - a factor that strongly affects search time, while in contrast, *TAAT* must handle a huge heap even when only 20 results are required.

3.2 Phrase Expansion

During query evaluation, query terms are constructed to include the query's original keywords and lexical affinities (LAs) of the query [3]. This is achieved by finding all pairs of query words found close to each other in a window of some predefined small size. For each LA, Juru creates a pseudo posting list by merging the posting lists of the LA terms. It then finds all documents in which these terms appear close to each other, and adds them to the posting list of the LA with all the relevant occurrence information. After creating the posting list, the new LA is treated by the retrieval algorithm as any other term in the query. The user can also control the relative weight between keywords and LAs, thus giving more (or less) significance to LAs in relation to simple keywords in computing the relevance score.

The query can also be expanded by the query phrase. The posting list of the query phrase is created by merging the postings of all terms, considering only documents containing the query terms in adjacent offsets and in the right order. Thus, documents containing the query phrase are biased compared to other documents.

Similarly to LA weight which specifies the relative weight between an LA term and simple keyword term, a phrase weight specifies the relative weight of a phrase term. Table 2 shows the results for phrase expansion with several different phrase weights. The results are averaged for the 99 short queries of the Terabyte topics, using **WAND** with 1K heap size.

Phrase weight	0	0.1	0.2	0.3
P@5	0.618	0.614	0.602	0.594
P@10	0.575	0.577	0.573	0.562
MAP	0.305	0.310	0.307	0.304

Table 2. Phrase expansion with different phrase weights.

It seems that phrase expansion slightly contributes to average search precision. However, it does not help to improve the top results. This is quite surprising due to the common assumption that the existence of the query phrase in a document is a good indication for its relevancy. This phenomenon should be further investigated.

3.3 Web query expansion

Our next experiment dealt with query expansion based on external resource. Following the good results obtained by several groups using Web expansion in previous years, we upgraded our system to benefit Web expansion using *Answers.com* search engine. Answers.com is a free search service, providing instant answers over many topics. As opposed to standard search engines that serve up a list of links to follow, it displays quick, snapshot answers with concise, reliable information. If it fails to answer the query it returns the first result returned by *Google* for that query. Our expansion procedure works by first submitting the topic title to *answer.com*, and then using the result page for query expansion. All query terms are expanded by their lexical affinities as extracted from the expanding Web page [3].

Unfortunately, We could not demonstrate significant improvement using Web expansion for the 99 training topics. However, following our belief in the quality of this mechanism, one of our official runs for this year's task is based on Web expansion. Indeed, this is our best run for this year's task (see Table 3).

3.4 Some other experiments

We also experimented with several other parameters to tune our system for the .gov2 collection while using WAND. In agreement with previous results, document static scores, based on number of in-links of the web pages, were found to be very useful. The document's static score is linearly combined with its textual score to provide its final score.

In contrast to static scores, anchor-text which is considered to be very helpful for TREC Web tasks, damaged the results for the 99 training topics we used. We do not have a good explanation for that. One speculation is that anchor text should be only used when it contains all query terms. This will avoid the cases that one query term, frequently appearing within many anchors, negatively biases the results.

4 Manual Run

A challenging task this year was to manually construct queries that will answer the topic while enriching the pool with unique results. In order to promote unique results we first studied the results returned with the plain runs, assuming most automatic runs will have a similar bias toward documents containing topic terms.

Our main assumption was that the way to achieve the best uniqueness score is to try and re-rank the top 50 results of each run to include results that will not appear

in a normal topic title run so that the manual and the automatic runs will agree on as few documents as possible. We came up with taxonomy of query styles that will yield different results from the "conventional" run: Skew Query, Reduce Query, Alternate Query, and Substitute Query. Figure 1 shows the schemes of the different query types.

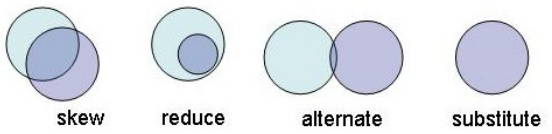

Fig. 1. Schemes of the different query types.

Skew Query

This is a query that keeps the most important terms in the query but changes one of the less important terms to create a partial overlap with the expected automatic run. An example for this is topic number 801 *Kudzu Pueraria lobata*, which we changed to *+Kudzu Eradication, control, herbicide, Tordon*[1]. We identified *Kudzu* to be a unique and unambiguous term which appears 1756 times in the collection. Similarly, topic number 829, *Spanish Civil War support*, was manually changed to *+"Spanish Civil War" aid*. The unambiguous phrase *"Spanish Civil War"* appears in the collection 354 times. In these topics the terms *Kudzu* and *"Spanish Civil War"* were enough to define the scope of the query and by forcing the system to return only answers that contain them we were able to farther filter those relevant documents with terms that slightly change the ranking of the results. In both queries we retrieved all documents in the Qrels set (801: 128/128 and 829: 24/24).

Reduce Query

An example for this can be found in our editing for topic 816 *USAID assistance to Galapagos* which we changed to *+USAID +Galapagos*. In the collection, the term *Galapagos* and the term *USAID* appear as lexical affinity only six times, which means that most occurrences of the terms in the Qrels (there are 23 of those) do not appear next to each other. However, the term *assistance* appears 2798691 times from which 10666 times as a lexical affinity of *USAID*. This means that by reducing the query to just *USAID* and *Galapagos* we were able to promote only the essential terms in the query. We contend that the term *assistance* provide more noise than information in this case and that by reducing the query to two terms we remove most of the noise.

Alternate Query

This type of modification is an "OR" query where both sides of the "OR" are conceived as perfectly good queries. Query 806, *Doctors Without Borders*, is an excellent example. For this query we requested all the documents that contain either *"Doctors Without Borders"* or *"Medecins sans Frontieres"*. Each of the queries describes the topic and is unambiguous.

[1] In the query syntax of our system, a query term marked by '+' is must-appear term.

Substitute Query

This is a query in which all the original terms are replaced with new terms. For example, we replaced the terms in topic 820, *imported fire ants*, with their Latin name *"Solenopsis invicta" "Solenopsis richteri"* since it is likely that only articles that specifically discuss imported fire ants will make the effort to mention their Latin name.

In many of the reformulated queries we reduced the number of possible results and "risked" ourselves in finding fewer relevant results and thus achieve a worse run overall. However, since most of the unique results are found somewhere in the "fringes" of the data this was a calculated risk.

5 Official Submissions

We submitted 4 official runs for this year ad-hock task:

1. **JuruMan**: queries were created manually for the new topics. The main goal is to diversify the results from those returned by automatic runs which are usually biased toward documents containing topic terms.
2. **JuruT**: queries are automatically created based on the topic title.
3. **JuruTD**: queries are automatically created based on the topic title and description.
4. **JuruTWE**: title queries are expanded by LAs extracted from the Web page returned by Answers.com for that query. The LAs are those containing one of the query terms.

For all runs we used **WAND**, using static scores, and applying phrase expansion for the automatic runs. Anchor text was disregarded by setting the anchor term weight to zero. Table 3 shows the results, compared to the average median and the average best results of all participating systems. Figur 2 shows the difference between the AP of Juru's title-based runs to the median AP of all automatic runs, for each of the new topics.

	MAP	P@10	infAP
JuruT	0.329	0.572	0.269
JuruTD	0.343	0.580	0.280
JuruTWE	0.351	0.638	0.269
auto-Median	0.291	0.540	0.241
auto-Best	0.517	0.870	0.473
JuruMan	0.275	0.570	0.241
man-Median	0.243	0.562	0.198
man-Best	0.512	0.908	0.479

Table 3. Official results of TREC 2006, compared to the average median and the average best results of all runs.

Our best run is JuruTWE. Web expansion significantly improves the results, mostly P@10. However, it is interesting to note that Web expansion works best for "easy" topics – for many difficult topics expansion does not help much.

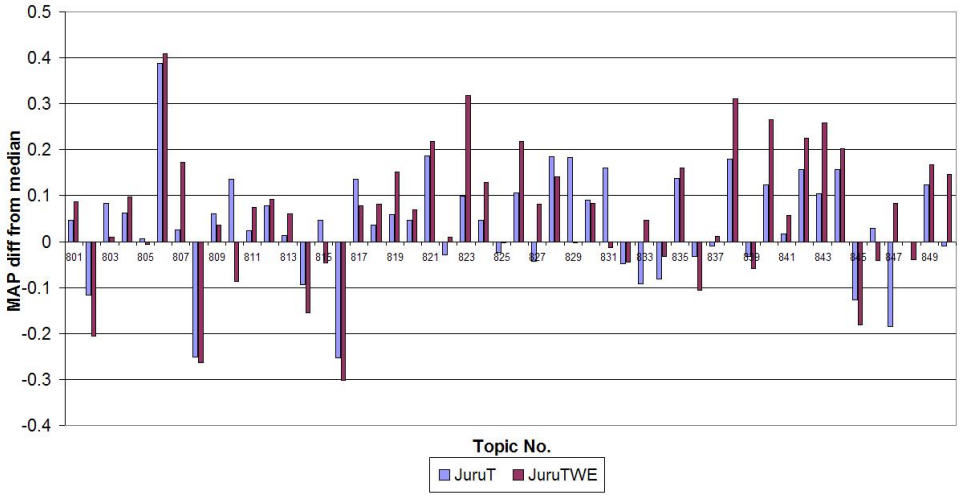

Fig. 2. The difference between the AP of Juru's title-based runs to the median AP of all 61 automatic ad-hock runs.

For most topics, Juru's results seems to be consistently better than the median result of all runs. This is also true for the manual run that its results are better than the median results of all 19 manual runs. However, the results of Juru's manual run are lower than the results of the automatic runs. This is agreement with the other manual runs for which the median MAP is lower than the median MAP of the automatic runs. A possible explanation is that manual runs focus on diversity from the automatic runs. In addition, the manual runs return less results per topic than the 1000 returned by the automatic runs (see Section 4).

6 Summary

The results we obtained from this year's experiments demonstrate the superiority of *DAAT* strategy over *TAAT* for searching over large collection. While handling a much smaller heap in memory, **WAND** enables to achieve better results than the *TAAT* strategy. The main reason is the better pruning mechanism of **WAND** compared to the pruning process concealed in *TAAT*. Furthermore, Web expansion proved to be a powerful tool for improving search results.

Finally, this is the first time that we were able to index and search over (half) terabyte of data in a reasonable time, using only one index, while achieving decent results – an important milestone for our search system.

References

1. Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434, New York, NY, USA, 2003. ACM Press.
2. David Carmel, Einat Amitay, Miki Herscovici, Yoelle S. Maarek, Yael Petruschka, and Aya Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proceeding of Tenth Text REtrieval Conference (TREC-10)*. National Institute of Standards and Technology. NIST, 2001.

3. David Carmel, Eitan Farchi, Yael Petruschka, and Aya Soffer. Automatic query refinement using lexical affinities with maximal information gain. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 283–290. ACM Press, 2002.
4. Elad Yom-Tov, David Carmel, Adam Darlow, Dan Pelleg, Shai Errera-Yaakov, and Shai Fine. Juru at TREC 2005: Query Prediction in the Terabyte and the Robust tracks. In *Proceedings of the 14th Text REtrieval Conference (TREC2005)*. National Institute of Standards and Technology. NIST, 2005.