RMIT at the TREC 2016 LiveQA Track

Joel Mackenzie, Ruey-Cheng Chen, J. Shane Culpepper
RMIT University
Melbourne, Australia
{*joel.mackenzie, ruey-cheng.chen, shane.culpepper*}*@rmit.edu.au*

***Abstract*—This paper describes the four systems RMIT fielded for the TREC 2016 LiveQA task and the associated experiments. Similar to last year, the results show that simple solutions tend to work best, and that our improved baseline systems achieved an above-average performance. We use a commercial search engine as a first stage retrieval mechanism and compare it with our internal system which uses a carefully curated document collection. Somewhat surprisingly, we found that on average the small curated collection performed better within our current framework, warranting further studies on when and when not to use an external resource, such as a publicly available search engine API. Finally, we show that small improvements to performance can substantially reduce failure rates.**

***Keywords*-TREC LiveQA 2016; RMIT; paragraph retrieval; summarization; learning to rank**

I. Overview

In the TREC LiveQA 2016 challenge, we continued to explore ideas within an established two-stage answer-finding framework used in last year's LiveQA challenge. Our long-term goal is to build a fully functional, modular multi-stage retrieval system that cascades candidate results through a series of increasingly complex filters. In our current system, a first-stage retrieval module is employed to first populate a good set of answer-bearing passages, and then a summarization module is used to generate the final answers via passage re-ranking or query-biased summarization.

We configured four different systems following two distinct strategies. First, we looked at the retrieval part and made comparisons between two common retrieval settings: retrieving passages from a home-made question-answering test collection, and retrieving snippets from a commercial search engine. Second, we made comparisons between two common answer-producing strategies: answer re-ranking using a Learning-to-Rank (LtR) model, and query-biased summarization using a max-coverage optimization model. This has led us to consider the following research questions:

RQ 1:. *Which strategy produces better passages, retrieving from a local test collection or pulling content from a commercial search engine?*

RQ 2:. *Which strategy produces better answers, locating the best passages directly or generating a succinct summary out of the top passages?*

RQ 3:. *Does the efficiency of the first-stage retrieval module contribute to the failure rate on longer questions?*

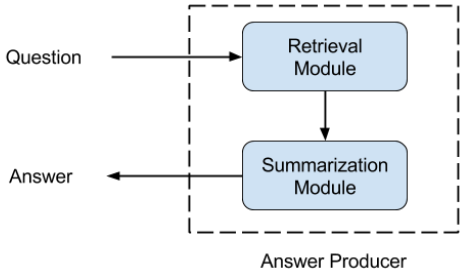

Figure 1. System architecture for the RMIT systems.

We also did a careful per query analysis using the 2015 queries to determine how efficiency impacted the performance with regard to the time budget. We found that returning the top-k paragraphs from our local collection followed by generating the answer with the coverage-based summarizer provided the most effective answers.

II. Methodology

We now describe the collection and retrieval setting used in our system.

A. Server architecture

The servers are built on top of the computing resources we allocated from NecTAR,[1] the Australian National Research cloud computing network. Throughout the challenge, we use only one instance to host all of the services.

All four RMIT systems implemented a two-stage system architecture, as illustrated in Figure 1. Upon receiving a question, the system would first convert it into a bag-of-words query, and run the query through the Retrieval Module (cf. Section III). The Retrieval Module retrieved a set of passages from the underlying corpora and served them to the Summarization Module (cf. Section IV), which in turn produces a piece of text that is coherent and long enough to fill the required answer size.

The various modules were connected using a resource allocator called Answer Producer, and written in the Go Programming Language. It included graceful handling of timeouts, and guaranteed responses within the 60 second window.[2]

[1] https://www.nectar.org.au
[2] https://github.com/TimothyJones/trec-liveqa-server

Table I
SUMMARY OF COLLECTIONS INDEXED TO ANSWER QUESTIONS.

Collection	# Paragraphs	# Words	Description
Wikipedia-EN	47,193K	1,775M	Online knowledge base
Yahoo! Answers CQA v1.0	31,972K	1,462M	Answers items from the Yahoo! Answers website.
TREC 2015 LiveQA Data	22K	1.8M	

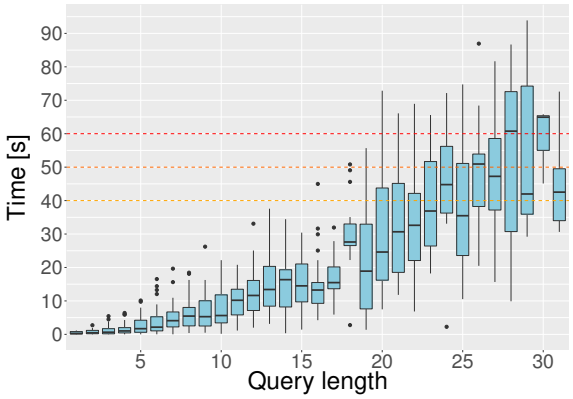

Figure 2. Query times across our test set of queries, timing the Indri passage retrieval function, `#combine[passage100:50](...)`. The horizontal lines at 40, 50 and 60 seconds denote an increasing chance of failure, as the retrieval budget becomes too high. Clearly, any query above the 60s line was a failure, and any above the 50s line was likely a failure, given the time required to generate an answer from the candidates and return the answer to the LiveQA server.

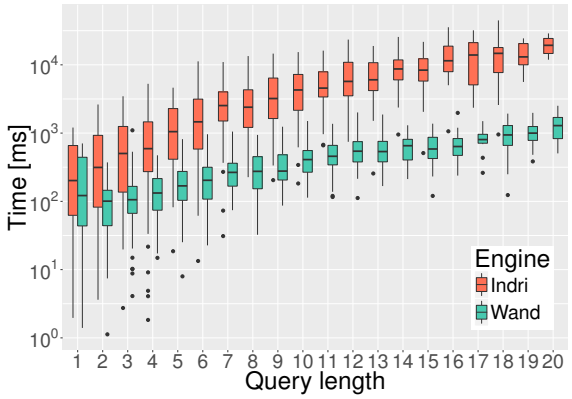

Figure 3. Query times across our test set of queries, comparing Indri to our WAND algorithm. These timings are for top-1000 paragraph retrieval, using the bag-of-words BM25 ranking function. Note that time is in log scale msec, while Figure 2 on the left is in seconds without a log scale.

B. Run descriptions

RMIT-1 (automatic): A WAND bag-of-words passage retrieval using all of the terms in the question title, with answers generated from top-k passages by using a Learning-to-Rank model. See Section IV-A for a full description of our Learning-to-Rank model.

RMIT-2 (automatic): Bing Search API snippets using all of the terms in the question title, with answers generated from top-k passages by using a Learning-to-Rank model.

RMIT-11 (automatic): A WAND bag-of-words passage retrieval using all of the terms in the question title, with answers generated from top-k passages by using a coverage-based summarization algorithm. See Section IV-B for details on the summarization model.

RMIT-12 (automatic): Bing Search API snippets using all of the terms in the question title, with answers generated from top-k passages by using an optimization-based summarization algorithm.

III. RETRIEVAL MODULE

A. Passage Retrieval Using WAND Indexes

Our first retrieval module was built on top of several question-answering test collections, including English Wikipedia, Yahoo! Answers CQA data 1.0, and the TREC 2015 LiveQA dataset. Table I shows the details for each of these test collections. To prepare the test collection for English Wikipedia, open-source tools such as wp-download[3] and wikiextractor[4] were used to fetch a recent dump and extract the XML contents. Paragraphs extracted from the output of wikiextractor were assigned a unique docno and got indexed using Indri.[5] For the Yahoo! Answers CQA data, we stripped all answer items (not just the best answers) from the data and indexed them as documents. No further paragraph separation was done to these answer items as most of them were short. No other subject/content information was incorporated (i.e., question title or description). The TREC 2015 LiveQA data was prepared and indexed in a similar fashion, with answer items treated as standalone paragraphs.

The approach taken here was similar to what we did last year [2], but with a few subtle differences: Instead of using the Indri index to answer queries directly, and utilizing the passage retrieval operator `#combine[passage100:50](...)` to return a set of candidate passages from these documents, we chose to index paragraphs and passages directly, and using our own search system [6] to perform the top-k retrieval. This allows us to use standard bag-of-words retrieval models to get a reasonable candidate set of passages more efficiently, which can then be passed onto the next stage of our system. This was motivated by the scalability of Indri passage retrieval: It does not scale well as query length increases (Figure 2). Given that there is a fixed

[3]https://github.com/babilen/wp-download
[4]https://github.com/attardi/wikiextractor
[5]http://www.lemurproject.org/indri/php
[6]https://github.com/jsc/WANDbl

Table II
COMPARING TITLE-ONLY TO TITLE + DESCRIPTION FIRST-STAGE BAG-OF-WORDS QUERIES. WE REPORT RECALL@k, AND ERR@20. NOTE THAT ALL OF THE TITLE-ONLY RESULTS WERE FOUND TO BE SIGNIFICANTLY BETTER WITH A $p < 0.01$, USING A TWO-TAILED PAIRED t-TEST.

Metric	Title-only	Title + Description
4+ Relevance		
Recall@10	0.1268	0.0995
Recall@100	0.1870	0.1518
Recall@1000	0.2485	0.2036
3+ Relevance		
Recall@10	0.2134	0.1606
Recall@100	0.3108	0.2457
Recall@1000	0.4129	0.3277
Weights 4,3,2,1		
ERR@20	0.1624	0.1351

time budget in the task, and queries can be long, our new approach allows us to more reliably retrieve a set of candidate documents within the time budget.

Our custom indexing system (WANDbl) is a faithful re-implementation of WAND [1] with several engineering performance enhancements that maximize efficiency [3, 5, 6]. The index required for our system is generated from the Indri index, and we used Krovetz stemming and the default InQuery stoplist[7]. This yielded a single 5.6 GB index that contained 79.2 million paragraphs and 15.5 million unique terms. The average length of an indexed paragraph was 47.4 terms. We use BM25 to rank the candidate passages, with the parameter configuration of $k_1 = 0.9$ and $b = 0.4$.[8]

The performance of the WANDbl index and Indri index are shown in Figure 3. Clearly, it is more beneficial to use the WANDbl system, as this allows more time for the second-stage re-ranking and summarization. In the production system, the module was deliberately configured to return only the top-10 passages as increasing this number showed no benefit in overall effectiveness with our current summarizer.

Another design decision was whether to use `Title`-only, or both the `Title` and the `Description` for our first-stage retrieval. Given that we opted to use simple bag-of-words models, we used the LiveQA2015 dataset to test the performance of `Title`-only queries, compared to `Title` + `Description` queries. Table II presents the effectiveness results for this experiment. We present Recall@k including only answers marked as "highly relevant", and also Recall@k including both "highly relevant" and "relevant" answers. We also present ERR@20 to gauge how well the first-stage system does in terms of early precision, using the weights 4, 3, 2 and 1, as found in the LiveQA2015 QRELs. Based on these simple experiments, we found that `Title`-only queries performed better in our current system configuration. Thus, we use only the `Title` during the first-stage candidate retrieval.

[7]http://www.lemurproject.org/indri.php

[8]The values for b and k_1 are different than the defaults reported by Robertson et al. [7]. These parameter choices were reported for Atire and Lucene in the 2015 IR-Reproducibility Challenge, see github.com/lintool/IR-Reproducibility for further details.

Table III
LIST OF THE 5 QUERY-MATCHING FEATURES

Feature	Description
ExactMatch	Query is a substring in the passage
Overlap	Fraction of query terms covered
OverlapSyn	Fraction of query-term synonyms covered
LM	Log-likelihood from the passage language model
Length	Number of terms in the passage

B. Bing Search API snippets

Our second retrieval module was built on top of the Bing search engine, using its search result snippets as answer candidates. The advantage of this approach is that more answer-bearing passages can be directly discovered from the Web, although these passages might contain incomplete sentences or truncated texts. As will be shown in Section V-A, Bing snippets were found to be indicative of relevant webpages, but perhaps not optimized for revealing answer-bearing sentences.

Our implementation used the Bing Search API which was available via the Azure Data Market.[9] For each incoming question, the question title was submitted to Bing, and then the top-50 search result snippets were retrieved and passed on to the next stage. Increasing the number of snippets to 100 showed no benefit in our earlier experiments.

IV. SUMMARIZATION MODULE

A. Learning-to-Rank Model

Our first answer generation module implements a Learning-to-Rank model proposed by Metzler and Kanungo [4]. The model was originally used in query-biased summarization, using six simple yet effective features to predict the relevance of sentences from retrieved documents. A summary can then be put together by repeatedly incorporating top-ranked sentences until the character limit is reached. This method was a common baseline in snippet generation, and in some recent work it was also used to rank answer sentences [9].

In our production system, this model was applied to directly ranking paragraphs/passages. One reason for not using finer-grained text units such as sentences is that the top-ranked sentences do not always produce a coherent text (even when sorted in their original order). Another reason, and perhaps more compelling, is that adapting the model to answer ranking provides an interesting comparison to conventional summarization methods. In our preliminary tests, the answer ranking strategy appeared to deliver comparable results to the sentence ranking approach.

Table III provides more details about the features we used. From the work by Metzler and Kanungo, five out of the original six features were adapted to our passage data. Two answer ranking models were developed separately

[9]https://datamarket.azure.com/dataset/bing/searchweb

Table IV
EFFECTIVENESS SUMMARY FOR ALL FOUR RMIT SYSTEMS WHEN COMPARED TO THE AVERAGE ACROSS ALL SYSTEMS PARTICIPATING IN THE 2016 LIVEQA TRACK.

Run ID	Description	Avg. Score (0-3)	Success @2+	Success @3+	Success @4+	Precision @2+	Precision @3+	Precision @4+
RMIT-1	WAND + LtR	0.723	0.384	0.239	0.099	0.388	0.242	0.100
RMIT-2	Bing + LtR	0.422	0.250	0.132	0.039	0.254	0.134	0.040
RMIT-11	WAND + Opt	**0.786**	**0.428**	**0.252**	**0.106**	**0.431**	**0.254**	**0.107**
RMIT-12	Bing + Opt	0.447	0.273	0.137	0.037	0.274	0.137	0.037
All Runs		0.577	0.304	0.190	0.086	0.392	0.243	0.108

using the Yahoo! Answers CQA data 1.0 (WebScope L6) and the TREC 2015 LiveQA data. The model from the Yahoo! Answers data was trained on a sample of 1,000 questions, with the best answer labeled as 1 and the other answer items as 0. The second model was trained on the TREC 2015 LiveQA data using graded relevance: All 1,087 questions with score-4 answers are labeled as 2, score-3 answers as 1, and the others as 0. Both models were trained via 5-fold cross validation using the LambdaMART implementation from RankLib[10], and optimized for ERR@5. All the texts in the training data were stemmed using the Krovetz stemmer, and stopped using the InQuery stoplist. WordNet synsets were used for extracting the `OverlapSyn` feature. The GOV2 test collection was used as the background corpus in the computation of the `LM` feature. The parameter μ in `LM` was set to 10. We ran a small-scale grid search over the number of trees and the learning rate to choose the final parameter settings. Eventually, we settled on the latter model trained on LiveQA data as it delivered stronger results in our preliminary tests. The final model has this configuration: the number of trees was set to 1000, the number of leaves to 10, and the learning rate to 0.1.

B. Optimization Model

For summarization, we used the model proposed by Takamura and Okumura [8] to generate extractive summaries from the top-ranked passages. In this model, summarization is characterized as a two-way optimization problem, in which coverage over important words is maximized, and redundancies are minimized simultaneously. The mathematical formulation is given as follows:

$$\begin{aligned} \text{maximize} \quad & (1-\lambda)\sum_j w_j z_j + \lambda \sum_i \sum_j x_i w_j a_{ij} \\ \text{subject to} \quad & x_i \in \{0,1\} \text{ for all } i; \\ & z_j \in \{0,1\} \text{ for all } j; \\ & \sum_i c_i x_i \leq K; \\ & \sum_i a_{ij} x_i \geq z_j \text{for all } j \end{aligned} \tag{1}$$

To produce an extractive summary, a choice over the set of sentences is made to decide what to include. This choice is modeled in the optimization problem as two sets of variables x_i and z_j, where the former indicating the binary decision on keeping sentence i, and the latter on keeping word j in the summary. In other words, for each sentence i, x_i is set to 1 if sentence i is to be included in the summary, or 0 otherwise. Analogously for each term j, z_j is set to 1 if term j is included.

In this problem, c_i denotes the cost of selecting sentence s_i (i.e. number of characters in s_i), and w_j denotes the weight of word j. We used a TF·IDF weighting scheme in which the term frequency (*TF*) is derived from the question title and body, and the inverse document-frequency (*IDF*) is learned from a background corpus. The term frequency collected from the question body is further penalized with a factor $\alpha < 1$ as the information given in the question body can be less precise than in the title.

$$w_j = [TF_{title}(j) + \alpha\ TF_{body}(j)] * IDF(j) \tag{2}$$

The correspondence between the sentence i and the word j is coded in the indicator variable a_{ij}, whose value is set to 1 if the word j appears in sentence i, and 0 otherwise. With the first constraint, we limit the size of the summary to K characters at most (K is set to 1,000 throughout). With the second constraint, the word coverage is related to the sentence coverage, thus completing the formulation.

Empirically, we fine-tuned the parameters λ and α based on prior test runs. In the challenge, we set $\lambda = 0.1$ and $\alpha = 0.43$. We used the IBM CPLEX solver to compute the optimal allocation. The background corpus used was the GOV2 test collection.

V. RESULTS

A. Effectiveness

The LiveQA challenge results are shown in Table IV, where our submitted runs and the average result across all runs are shown. Our RMIT-11 run delivered the best performance in our experiment, achieving 0.786 in Avg Score. The base run outperformed the average across all runs and metrics, except for `P@4+`, where it was very much equal to the average score.

A post-hoc analysis was performed to understand the effect of the test collection and that of our answer-producing algorithm, using RMIT-11 as the reference run. The distribution of score differences across query topics is shown in Table V, where two related runs RMIT-1 (shared the test collection) and RMIT-12 (shared the answer-producing algorithm) are compared directly. Query-biased summarization and the answer re-ranking algorithms were

[10]http://www.lemurproject.org/ranklib.php

Table V
DISTRIBUTION OF SCORE DIFFERENCES ACROSS QUERIES BETWEEN RMIT-11 AND TWO RELATED RUNS

System Pair	Score Differences											
	-4	-3	-2	-1	0	1	2	3	4	$\Pr(\text{diff} < 0)$	$\Pr(\text{diff} = 0)$	$\Pr(\text{diff} > 0)$
RMIT-11 vs. RMIT-1	1	8	30	86	700	126	26	21	0	12.5%	70.2%	17.3%
RMIT-11 vs. RMIT-12	0	10	29	72	483	211	99	61	37	11.1%	48.2%	40.7%

Table VI
ERROR CAUSES, BASED ON AN ANALYSIS ON THE OBSERVED SCORE DIFFERENCES BETWEEN RMIT-11 AND RMIT-12

Cause	# Queries
Local collection errors	
Navigational intent	1
Formatting (i.e., answer in HTML table)	1
Query drift caused by question body	2
Irrelevant answer	7
Assessor disagreement	4
Bing snippets errors	
Result filled with query terms but no answer text	13
Answer truncated	12
Assessor disagreement	7

found to have differences on 29.8% of the queries, and query-biased summarization appeared to have a slight advantage. Regarding the influence of test collections, our local collection and the Bing snippets performed similarly for around 50% of the queries. For the remainder of the queries, our local collection was roughly 3.67 times more likely to produce a better result than the Bing snippets.

Inspired by this large difference on test collection, we carried out a further error analysis to investigate any potential issues that the test collections may have. A subset of 408 queries was formed which included the queries with the biggest score differences between RMIT-11 and RMIT-12. For these, 47 queries were randomly sampled, which is approximately 10% of the subset size. One of the co-authors was asked to carefully review all 47 queries and the associated answers, and identify possible causes of the observed score differences. A breakdown of the recorded error sources is provided in Table VI. For the Bing snippet system, missing answers and text truncation were responsible for the majority of differences. The local collection also struggles to retrieve the valid answers for some queries, but less-fragmented answer summaries provided by the system were in general more readable, which increased the odds of receiving a favorable judgment by the assessors.

Generally speaking, it would appear that our own first-stage system provides better candidates to the second stage LtR/Opt systems, as both WANDbl-based systems clearly outperformed the Bing systems. This is likely due to how noisy the questions were. Given that our local collection was optimized for QA, there is a greater chance that the passages retrieved were relevant, whereas the Bing system may have returned other extraneous information (as the information need may be unclear), and the corpus used is orders of magnitude larger and more diverse. This

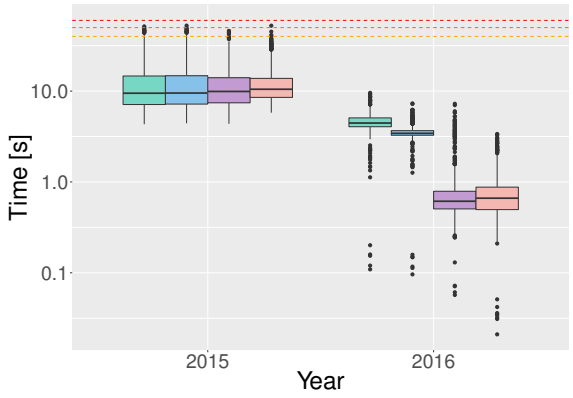

Figure 4. Response times compared to the 4 systems fielded by RMIT in 2015. Clearly, the improved early stage performance provides a much larger budget for late-stage processing. The yellow, orange and red horizontal lines denote the 40, 50 and 60 second intervals respectively, where performance begins to affect the likelihood that results will be returned within the time budget.

highlights the importance of query understanding and query rewriting for QA tasks – something we intend to focus on in future LiveQA initiatives.

B. Efficiency

Each RMIT system returned $1{,}006 \pm 6$ answers, which was considerably more than the average of 771 answers. This is likely due to how efficient the RMIT systems were – Figure 4 compares the 4 systems fielded by RMIT in 2015 against the 4 RMIT systems used in this years challenge. The 2016 systems were able to generate answers under 10 seconds for every received query, a significant improvement over last year.

Additionally, Figure 5 breaks down the timings for each system by the query length. Interestingly, the query length does not have an effect on the median query time. This is because the timings are dominated by the second-stage summarization module, which is not directly effected by the length of the query. Both System 1 and 2 shared a similar LtR last stage configuration, which is more computationally expensive than our summarizer, as the figure shows.

Clearly, there is much more time in our budget that can be utilized to improve our answer quality; future work includes adding additional stages to the retrieval system, such as a query-rewriting stage, which should help improve effectiveness, while utilizing the remaining time budget.

VI. CONCLUSION

We have explored four different system configurations for the TREC LIVEQA Track in 2016. While we remain

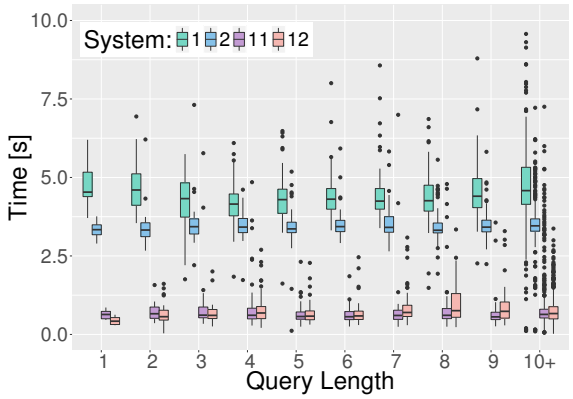

Figure 5. Response times broken down by query length for all 4 systems used in the challenge. Query length does not adversely effect the timings, as these are likely dominated by the second-stage summarization module.

pleasantly surprised with the performance of our simple system configurations, we are hopeful that further improvements can still be realized through better filtering steps, better query parsing and prediction, and increasing the coverage in our current documents collections.

In summary, we found that retrieving the top-k paragraphs from a local test collection combined with generating a succinct summary of these paragraphs provided the most effective solution. Additionally, we show that the first-stage retrieval efficiency cost is dominated by the second-stage re-ranking/summarizing stage. Finally, we show large efficiency improvements compared to our systems from the 2015 LiveQA challenge, which allows us to include more expensive stages in future work.

Acknowledgment.

This work was supported in part by the Australian Research Council's *Discovery Projects* Scheme (DP140102655). Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275). The statements made herein are solely the responsibility of the authors.

References

[1] A. Z. Broder, D. Carmel, H. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th ACM international conference on Information & knowledge anagement*, pages 426–434. ACM, 2003.

[2] Ruey-Cheng Chen, J. Shane Culpepper, Tadele Tadela Damessie, Timothy Jones, Ahmed Mourad, Kevin Ong, Falk Scholer, and Evi Yulianti. RMIT at the TREC 2015 liveqa track. In *Proceedings of TREC 2015*, 2015.

[3] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. Efficient location-aware web search. In *ADCS*, pages 4.1–4.8, 2015.

[4] Donald Metzler and Tapas Kanungo. Machine learned sentence selection strategies for query-biased summarization. In *SIGIR Learning to Rank Workshop*, pages 40–47, 2008.

[5] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *ADCS*, pages 58–65, 2013.

[6] M. Petri, A. Moffat, and J. S. Culpepper. Score-safe term dependency processing with hybrid indexes. In *SIGIR*, pages 899–902, 2014.

[7] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proc. TREC-3*, 1994.

[8] Hiroya Takamura and Manabu Okumura. Text summarization model based on maximum coverage problem and its variant. In *Proc. of EACL*, pages 781–789, 2009.

[9] Liu Yang, Qingyao Ai, Damiano Spina, Ruey-Cheng Chen, Liang Pang, W. Bruce Croft, Jiafeng Guo, and Falk Scholer. Beyond Factoid QA: Effective Methods for Non-factoid Answer Sentence Retrieval. In *Proc. of ECIR*, pages 115–128. 2016.