Rank-at-a-Time Query Processing

Ahmed Elbagoury, Matt Crane, and Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada
{ahmed.elbagoury,matt.crane,jimmylin}@uwaterloo.ca

ABSTRACT

Query processing strategies for ranked retrieval have been studied for decades. In this paper we propose a new strategy, which we call rank-at-a-time query processing, that evaluates documents in descending order of quantized scores and is able to directly compute the final document ranking via a sequence of boolean intersections. We show that such a strategy is equivalent to a second-order restricted composition of per-term scores. Rank-at-a-time query processing has the advantage that it is anytime score-safe, which means that the retrieval algorithm can self-adapt to produce an *exact* ranking given an arbitrary latency constraint. Due to the combinatorial nature of compositions, however, a naïve implementation is too slow to be of practical use. To address this issue, we introduce a hybrid variant that is able to reduce query latency to a point that is on par with state-of-the-art retrieval engines.

1. INTRODUCTION

The information retrieval literature discusses three classes of query processing strategies for ranked retrieval: document-at-a-time (DaaT), term-at-a-time (TaaT), and score-at-a-time (SaaT). Each of these strategies has advantages and disadvantages, and are inherently tied to the layout of the inverted index. For example, in DaaT processing, postings are usually sorted in order of document id, whereas for TaaT and SaaT processing, postings are usually sorted in decreasing frequency or score order. Regardless of the technique, query processing involves traversing postings, computing document/score combinations that are held in some intermediate data structure, which is then manipulated to extract the final results.

We describe a novel query processing strategy called rank-at-a-time (RaaT) processing. The core insight is that if we pre-compute and quantize per-term scores, ranked retrieval can be decomposed into a sequence of boolean intersections. More precisely, we show that such a query processing strategy is equivalent to a second-order restricted composition of per-term scores. Due to the combinatorial nature of this problem, a naïve implementation is too slow to be of practical use. However, we introduce a hybrid variant that is able to reduce query latency to a point that is on par with state-of-the-art retrieval engines.

ICTIR '16, September 12 - 16, 2016, Newark, DE, USA

ISBN 978-1-4503-4497-5/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2970398.2970434

2. RELATED WORK

The DaaT, TaaT, and SaaT processing strategies have been well studied in the literature (see discussion below). Research largely focuses on techniques to improve efficiency (i.e., reduce query processing time) without negatively impacting retrieval quality.

For DaaT processing, most research focuses on methods for skipping documents in the postings lists that cannot appear in the top k results. The most common strategy revolves around Wand [4], where the skipping decision is based on the upper bounds of query term scores. The introduction of term weighting parameters and a threshold θ determines whether the query is interpreted as an AND or an OR query, or a hybrid. An enhancement to Wand-style processing is to group postings into fixed-size blocks and to process the groups block-wise using the BM-Wand algorithm [7, 8].

The TaaT strategy has attracted research into reducing the overhead of accumulators to store partial document scores. Moffat et al. [12] proposed a method in which the processing of the query switches from OR to AND in order to limit the number of accumulators that need to be created. Moffat and Zobel [11] extended this method to allow for heuristics on when such a switch should occur.

Frequency-ordered indexes were first proposed by Persin et al. [14]. Anh et al. [1] observed that term weights could be stored in such an index instead of the term frequencies. To facilitate compression, they quantized these weights into integer values (called impact scores); cf. Moffat et al. [12]. The SaaT strategy processes blocks of postings in decreasing impact score, potentially hopping from term to term. Lin and Trotman [10] showed that a specific instance of such a strategy called JASS supports anytime ranking, in which the retrieval algorithm can self-adapt to produce a document ranking given an arbitrary latency constraint. However, JASS achieves anytime ranking by introducing approximations (and hence is not score-safe). One variant of JASS was the fastest in a recent reproducibility evaluation comparing several open-source search engines [9], and thus JASS forms a strong baseline for this paper.

Also of note from the same evaluation is the "model B" two-pass retrieval approach of Boldi and Vigna [3], whose MG4J system was the second fastest. Their approach first

Term	(Quantized) Impact-Ordered Postings			
a	$10:[\ldots]$	$9:[\ldots]$	$6:[\ldots]$	$2:[\ldots]$
b	$15:[\ldots]$	$12:[\ldots]$	$3:[\ldots]$	
c	$11:[\ldots]$	$9:[\ldots]$	$6:[\ldots]$	

Table 1: Example terms with (quantized) impact-ordered postings. Each $[\ldots]$ indicates a group of document ids with the same impact score.

employs pure boolean retrieval to generate a candidate set, which is then scored using BM25 and sorted to produce the final document ranking. We show that their technique represents a special case of our grouped rank-at-a-time query processing variant (see Section 5).

3. RANK-AT-A-TIME EVALUATION

We make the key observation that using a quantized index with pre-computed per-term scores (i.e., impact scores) allows us to generate document rankings directly in descending order of rank via decomposition into boolean intersection queries. Consider the example terms and associated impact-ordered postings shown in Table 1. In this example, the highest score that any document can achieve is 36, obtained when a document has a score contribution of 10 from term a, 15 from b, and 11 from c. We denote these blocks of postings by their term and score contribution $\{a:10\}$, $\{b:15\}$, and $\{c:11\}$, respectively.

In rank-at-a-time query processing, for each document score descending from the highest possible, we enumerate all combinations of score contributions, which correspond to intersection queries that are then executed. The results of these intersection queries are directly added to the final output, since they are by design sorted by score. In this manner, we compute documents that appear in rank one, rank two, etc. (hence the name RaaT), but allowing for score ties in cases where the intersection queries produce multiple documents. In this example, there are unique combinations that yield scores 36, 35, and 34, and two that produce a score of 33, namely $\{a:10\} \cap \{b:12\} \cap \{c:11\}$ and $\{a:9\} \cap \{b:15\} \cap \{c:9\}$.

The set of possible combinations that yield a given score is exactly the set of integer compositions for that score. In a so-called A-restricted composition, each term must draw from the same set of values. For example, one A-restricted composition of 36 might be $(12, 12, 12)$, which is not a valid combination given the postings in our running example. To eliminate invalid combinations, we require what Page [13] calls a second-order restricted composition, which allows the set of possible values for each term to be different (drawn from the actual impact scores). Page describes an algorithm for generating second-order restricted compositions, which we refer to simply as compositions for the remainder of the paper. Note that in our current implementation, we generate compositions that require all terms to be present, which corresponds to conjunctive query processing (i.e., an AND query). In future work we discuss extending our approach to so-called *weak* compositions, which do not require the presence of all query terms.

For a given query q and a score s, there are a total of $\binom{s-1}{|q|-1}$ integer compositions. This calculation, however, does not take into account the range of impact scores for each postings list. We empirically compute and plot the number of compositions that are generated for test queries in the experimental section.

Rank-at-a-time query processing is an anytime ranking method (as discussed in Section 2). Since each intersection is independent, we can simply perform as many intersections as time allows. RaaT is simultaneously score-safe, in that it computes exact document scores (unlike JASS [10]). Because results are generated in rank order, processing lower ranks has no impact on higher ranks.

4. INITIAL EXPERIMENTS

To evaluate RaaT query processing, we follow the approach of Lin and Trotman [10] for JASS: we consume indexes that have been produced by the ATIRE system [15] and construct our own internal representation. This allows direct comparison against JASS, while removing all confounding issues related to index construction. We follow the method of Crane et al. [5] for computing a quantization range (1–255 for .GOV2) that does not reduce effectiveness.

The internal representation we chose uses Elias-Fano codes to store the document ids for each impact score. This was selected because it has been reported in information retrieval contexts as being efficient, particularly for intersection queries [16]. We use Facebook's implementation of Elias-Fano encoding in their open-source Folly library.[1] Intersections were performed using the holistic method presented by Culpepper and Moffat [6].

To evaluate the efficiency of our method we use the .GOV2 collection with TREC queries 701–850. These were selected so that our experiments could be directly compared to results from the RIGOR Reproducibility Challenge [9]. To that end, we ran our experiments on the same EC2 instance type (r3.4xlarge instance) to maximize comparability. To further support reproducibility, our software is open source and available on Github.[2]

Table 2 compares RaaT and JASS in terms of effectiveness (nDCG@10 and AP) and efficiency (mean and median query processing time) for different values of k in top k retrieval. We show two configurations of JASS: exhaustive processing and approximate processing with $\rho = 2.5$M. Although the later condition is not score safe, it was the fastest algorithm at the RIGOR Reproducibility Challenge [9], so this is a strong baseline. In terms of effectiveness, the differences between JASS (exhaustive) and RaaT can be attributed to the difference between disjunctive query processing in JASS and conjunctive query processing in RaaT (since we only generate compositions that include all terms), as well as idiosyncrasies of how scoring ties are broken. JASS ($\rho = 2.5$M) introduces approximations, although the degradation in effectiveness from JASS (exhaustive) is negligible.

As we can see, a direct implementation of RaaT is too slow to be of practical use. Analyses in Fig. 1 show why. On the left we plot the relationship between the maximum possible score in a query and the number of compositions generated at $k = 1000$ for all 150 .GOV2 queries. The scatterplot is broken down by query length, and note that the y-axis is in log scale. Longer queries and queries with higher maximum possible scores clearly generate more compositions. On the right, we plot the relationship between the number of compositions generated and query processing time across all

[1] `https://github.com/facebook/folly`
[2] `https://github.com/lintool/efir`

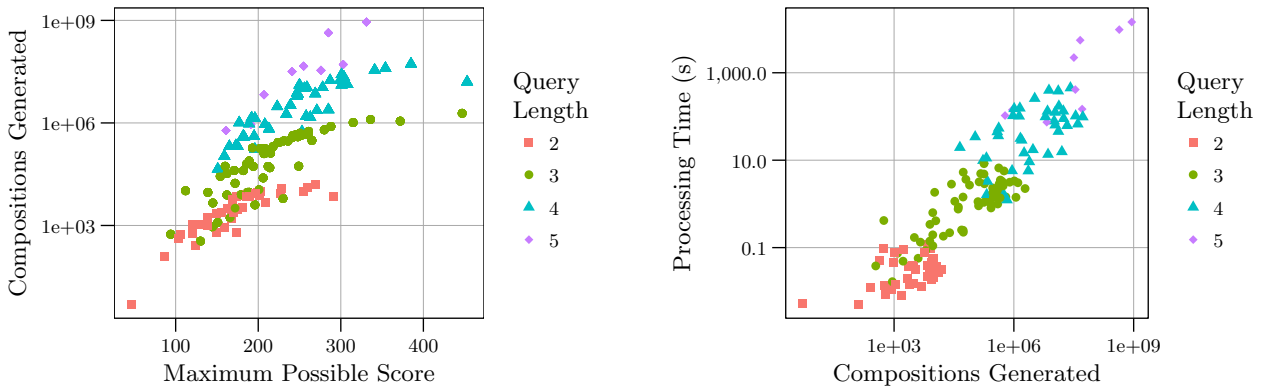

Figure 1: On the left, the number of compositions generated vs. maximum possible score (note y-axis in log scale). On the right, the relationship between the number of compositions generated and query processing time (note axes in log scale). Both plots show queries 701–850 for .GOV2 with $k = 1000$.

				ms	
System	k	nDCG@10	AP	mean	median
JASS (exhaustive)	1000	0.4395	0.2897	47	33
JASS (exhaustive)	100	0.4395	0.1672	47	33
JASS (exhaustive)	50	0.4395	0.1169	47	33
JASS (exhaustive)	10	0.4429	0.0441	47	33
JASS ($\rho = 2.5$M)	1000	0.4372	0.2866	26	29
JASS ($\rho = 2.5$M)	100	0.4372	0.1662	26	29
JASS ($\rho = 2.5$M)	50	0.4372	0.1168	26	29
JASS ($\rho = 2.5$M)	10	0.4402	0.0440	26	29
RaaT	1000	0.4409	0.2646	243,977	1,440
RaaT	100	0.4409	0.1676	63,808	252
RaaT	50	0.4409	0.1191	30,376	118
RaaT	10	0.4412	0.0443	6,447	33
gRaaT	1000	0.4406	0.2570	171	22
gRaaT	100	0.4404	0.1610	58	9
gRaaT	50	0.4334	0.1146	34	6
gRaaT	10	0.4258	0.0439	19	5

Table 2: The effectiveness and efficiency of JASS variants, RaaT, and gRaaT on queries 701–850 for .GOV2 with different k values.

150 queries at $k = 1000$. The plot also breaks down queries into different lengths. Note that both axes are in log scale. As expected, we see that query processing time is directly related to the number of intersections that are generated and evaluated, which in turn is dependent on query length. These effects are expected due to the combinatorial nature of the compositions. However, for all queries of length less than three, results are returned in less than 110 milliseconds, indicating that overall efficiency is particularly impeded by the large number of compositions for longer queries.

5. HYBRID RANK-AT-A-TIME

Initial experiments show that a straightforward implementation of RaaT is impractically slow due to the enormous number of intersections queries that are generated, particularly for long queries.

To address this issue, we introduce a hybrid variant of RaaT that dramatically reduces the number of intersection queries that are generated. We call this *grouped* RaaT, or gRaaT, and it begins by dividing each postings list into three equal groups based on their impact scores—for convenience, we refer to these groups as {high, medium, low} scores. We can then apply RaaT at the level of these groups, starting with $\{a : high\} \cap \{b : high\} \cap \{c : high\}$, backing off to $\{a : high\} \cap \{b : high\} \cap \{c : med\}$, $\{a : high\} \cap \{b : high\} \cap \{c : low\}$, etc., repeatedly dropping "score levels" across terms in a depth-first manner until k documents have been retrieved.

Note that these intersections do not produce documents in strictly descending score order, and therefore we need to keep track of document scores in a heap. Also, gRaaT is not score-safe because of its termination condition—the algorithm stops as soon as k documents have been retrieved—and the way the backoffs are sequenced. It is possible that results from subsequent intersection queries yield documents with scores higher than current scores.

This technique can be viewed as a generalization of Boldi and Vigna's model B [3]. In their approach, a sequence of boolean queries is generated with drop-one backoff until k documents have been returned. These candidates are then scored with BM25 and sorted. For example, with three query terms, model B performs $\{a\} \cap \{b\} \cap \{c\}$, backing off to $\{a\} \cap \{b\}$, $\{a\} \cap \{c\}$, $\{b\} \cap \{c\}$, etc. Thus, model B can be viewed as a special case of gRaaT in which each postings list consists of only a single group.

Experimental results with gRaaT are shown in Table 2, organized in the same manner as JASS and RaaT. We see that grouping postings to reduce the number of intersection queries is successful in reducing query processing time. For small values of k, gRaaT is competitive with JASS, and at $k = 10$, gRaaT is actually faster. The tradeoff is a marginal, but not statistically significant, decrease in effectiveness.

A more detailed analysis in Fig. 2 breaks query processing time down by query length in box-and-whiskers plots for $k = 10$. The horizontal lines in the boxes show the medians, and the diamonds show the means (note log scale). We see that for single term queries, RaaT and gRaaT are both substantially faster than both variants of JASS. For queries of length two, RaaT is the fastest: there are relatively few compositions generated, and RaaT does not have the additional scoring overhead introduced by groupings in gRaaT. Both are faster than the JASS variants. The performance of RaaT substantially degrades for longer queries but the postings grouping technique in gRaaT cuts down on the number of intersections and thus keeps query processing time low, remaining competitive with the JASS variants.

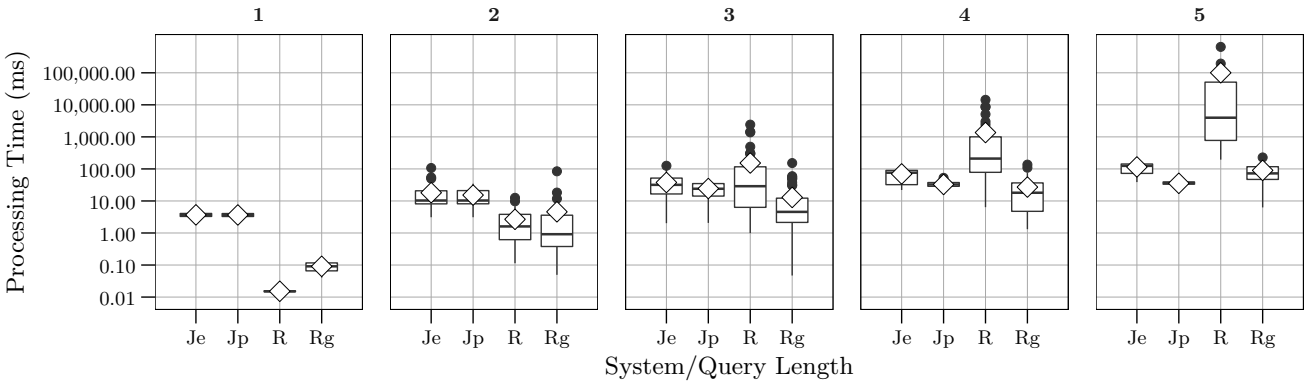

Figure 2: Comparison of query processing time in box-and-whiskers plots (means as diamonds) for JASS exhaustive (Je), JASS $\rho = 2.5$M (Jp), RaaT (R), and gRaaT (Rg), broken down by query length for $k = 10$.

6. FUTURE WORK AND CONCLUSIONS

This paper presents a novel rank-at-a-time query processing strategy that takes advantage of the equivalence between compositions and boolean intersection queries in quantized indexes to directly compute document scores in rank order. Although a naïve implementation is impractically slow, a hybrid variant achieves query processing time on par with state-of-the-art techniques. This promising result can be improved upon in a number of ways.

Our current RaaT implementation generates compositions in which all query terms must be present. It is straightforward to extend this algorithm to generate *weak* compositions in which terms can be absent. Translated into query processing, a non-weak composition corresponds to conjunctive query processing (i.e., AND), whereas a weak composition corresponds to standard "bag of words" ranking.

Experiments show that controlling the number of compositions, particularly for longer queries, is the key to achieving good performance. Our approach of grouping postings into {high, medium, low} scores appears to be effective in accomplishing this, but is rather crude. Obvious extensions include parameterizing the number of groupings, allowing us to tune the tradeoff between the number of compositions generated and the additional work involved in storing disparate document scores. We currently create a fixed number of groups within each postings list, but alternatively, groupings can be based on absolute scores.

The number of compositions can be reduced in other ways. The generation algorithm can be modified to discard compositions that cannot possibly yield results: for instance, if the first two terms of a composition yield no results, then all other compositions that contain the same two terms can be discarded. Alternatively, caching sub-intersections across score compositions can help eliminate unnecessary work.

Finally, there are other ways of extending RaaT. We do not presently support static document scores, which is important for many applications. There are additionally many opportunities for exploring approximate (i.e., not rank-safe) query processing, trading off effectiveness for efficiency, particularly in the context of multi-stage ranking [2].

Information retrieval researchers have extensively studied three classes of query processing strategies for ranked retrieval to date: DaaT, TaaT, SaaT. To this list we contribute RaaT. We have only begun to explore the properties of this approach, and anticipate further advances building on future work outlined above.

Acknowledgments. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the sponsors.

7. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. *SIGIR*, 2001.
[2] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. *SIGIR*, 2013.
[3] P. Boldi and S. Vigna. MG4J at TREC 2006. *TREC*, 2006.
[4] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. *CIKM*, 2003.
[5] M. Crane, A. Trotman, and R. O'Keefe. Maintaining discriminatory power in quantized indexes. *CIKM*, 2013.
[6] J. S. Culpepper and A. Moffat. Efficient set intersection for inverted indexing. *TOIS*, 29(1), 2010.
[7] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. *WSDM*, 2013.
[8] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. *SIGIR*, 2011.
[9] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. Toward reproducible baselines: The Open-Source IR Reproducibility Challenge. *ECIR*, 2016.
[10] J. Lin and A. Trotman. Anytime ranking for impact-ordered indexes. *ICTIR*, 2015.
[11] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *TOIS*, 14(4):349–379, 1996.
[12] A. Moffat, J. Zobel, and R. Sacks-Davis. Memory efficient ranking. *IP&M*, 30(6):733–744, 1994.
[13] D. R. Page. Generalized algorithm for restricted weak composition generation. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 12(4):345–372, 2013.
[14] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *JASIS*, 47(10):749–764, 1996.
[15] A. Trotman, X. Jia, and M. Crane. Towards an efficient and effective search engine. *OSIR Workshop*, 2012.
[16] S. Vigna. Quasi-succinct indices. *WSDM*, 2013.