Multiple Query Processing via Logic Function Factoring

Matteo Catena
ISTI-CNR
Pisa, Italy
matteo.catena@isti.cnr.it

Nicola Tonellotto
ISTI-CNR
Pisa, Italy
nicola.tonellotto@isti.cnr.it

ABSTRACT

Some extensions to search systems require support for multiple query processing. This is the case with query variations, i.e., different query formulations of the same information need. The results of their processing can be fused together to improve effectiveness, but this requires to traverse more than once the query terms' posting lists, thus prolonging the multiple query processing time. In this work, we propose an approach to optimize the processing of query variations to reduce their overall response time. Similarly to the standard Boolean model, we firstly represent a group of query variations as a logic function where Boolean variables represent query terms. We then apply factoring to such function, in order to produce a more compact but logically equivalent representation. The factored form is used to process the query variations in a single pass over the inverted index. We experimentally show that our approach can improve by up to 1.95× the mean processing time of a multiple query with no statistically significant degradation in terms of NDCG@10.

ACM Reference Format:
Matteo Catena and Nicola Tonellotto. 2019. Multiple Query Processing via Logic Function Factoring. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331297

1 INTRODUCTION

Some information retrieval tasks require the capability to process multiple queries (*multi-queries* for short), i.e., queries composed by different sub-queries which must be processed as an ensemble to generate the multi-query's results. For instance, rank fusion allows to produce a single effective list of results from the processing of different *query variations*. In essence, query variations represent a set of different textual formulations of the same information need. In fact, users can submit very different queries to a retrieval system while trying to solve the same search task. For example, consider a scenario where users want to know which are the positive and negative aspects of wind power. Some users will submit the query variation "wind power advantages and disadvantages", while others may issue something like "wind power pros and cons" or "wind power good or bad" [1].

Belkin et al. studied the effect of combining different query variations of the same topic to retrieve relevant documents, and they showed that merging together the top documents of different query variations can improve the effectiveness of the retrieval system for a specific information need [3]. The authors explain such improvement by observing that information representation and retrieval are a complex activity, and that no single query formulation can completely address such complexity. Therefore, they justify the usage of multiple query variations on the basis that their combination will address different aspects of the information need, and retrieve more relevant documents.

Boosting search effectiveness using query variations requires the ability to efficiently process multi-queries composed by several variations as *sub-queries*, and rank fusion techniques can be leveraged for this purpose [2–4]. These techniques compute a separate ranked list of results for each sub-query, and such lists are then aggregated to generate the multi-query's final results. Such multi-query processing strategy can be very effective, but is also a time consuming operation. In fact, a search system may need to traverse its inverted index multiple times to process a multi-query, once for each of its sub-queries.

In this work, we show how to optimize the processing of multi-queries composed by query variations as sub-queries. Firstly, we propose to process such multi-queries as a disjunction of conjunctive sub-queries, i.e., a document is a result for a multi-query only if it contains all the terms for at least one of its sub-queries. We express such condition as a logic function, which we use to test whether a document is a result for the multi-query. For instance, to satisfy the topic presented at the beginning of this section, a document must match a multi-query like "(wind AND power AND pros AND cons) OR (wind AND power AND good AND bad)". Documents are matched against this multi-query in a document-at-a-time fashion and, since the underlying logical formula is in *disjunctive normal form*, we refer to this approach as DNF matching. Our experiments will show that DNF is both efficient and effective.

However, notice how DNF incurs in redundant computations. In our example, DNF will intersect twice the posting lists related to the terms "wind" and "power". Therefore, we propose to further optimize DNF by *factoring* its underlying logic function representation. Factoring represents a logic function in a factored form, i.e., as either a single Boolean variable or as a sum or product of factored forms [5]. While being equivalent, factored forms are more compact than disjunctive normal forms, i.e., they reduce the number of times a Boolean variable must be inspected to determine the value of the whole formula. In terms of multi-query processing, our example multi-query can be optimized as "(wind AND power) AND ((pros AND cons) OR (good AND bad))", which requires to intersect the "wind" and "power"'s posting lists just once. In this work, we propose to automatically optimize multi-queries via logic function factoring, and we refer to this approach as FDNF matching.

Experiments conducted on the ClueWeb12 (Cat. B) and UQV100 datasets shows that DNF is faster than the state-of-the-art Single Pass CombSUM (SP-CS) [4] in processing multiple-queries composed by up to four query variations, while providing statistically indistinguishable effectiveness in terms of NDCG@10, and FDNF exhibits even lower query processing times than DNF/SP-CS in processing multiple-queries composed by up to seven variations.

2 DNF AND FDNF MATCHING

In this section we describe how multi-queries are processed by Disjunctive Normal Form (DNF) query processing strategy, and how this strategy can be optimized by leveraging logic function factoring (FDNF).

A multi-query Q is a set $\{q_1, \ldots, q_n\}$ of sub-queries q_i. Each sub-query in Q contains one or more terms, i.e., $q_i = \{t_1, \ldots, t_m\}$. The sub-queries in a multi-query are unique, but they can have some terms in common. Abusing our notation, a term belongs to Q if at least a sub-query of Q contains it.

To process a multi-query, all its sub-queries must be processed and their top results must be fused together, to produce a final list of top documents to be returned to the user. Depending on the search system setting, sub-queries can be processed in *disjunctive* mode or *conjunctive* mode. According to the standard Boolean model, a document d matches a disjunctive sub-query q, i.e., d is a result for q, if d contains *at least* one of the sub-query terms. Conversely, a document matches a conjunctive sub-query q if it contains *all* the sub-query terms [7]. We use a Boolean variable v_t to indicate if term t appears in a given document. We denote the logic 'OR' operator with the sum symbol '+', and the logic 'AND' operator with the multiplication symbol '·'. Therefore, a document matches a disjunctive sub-query q if the logic function $\sum_{t \in q} v_t$ evaluates to true, while it matches a conjunctive sub-query q if the logic function $\prod_{t \in q} v_t$ evaluates to true. In practice, the documents matching a disjunctive (resp. conjunctive) query are retrieved from an inverted index by performing the union (resp. intersection) of the posting lists related to the query terms. Conjunctive query processing is, in general, more efficient than disjunctive processing, even if the latter is considered more effective than the former [8].

In this work, we assume that all the sub-queries of a given multi-query Q are either processed in disjunctive mode or conjunctive mode. When the sub-queries are disjunctive, a document matches Q if it contains at least one term belonging to one of the sub-queries composing the multi-query. As a consequence, processing Q when its sub-queries are disjunctive is equivalent to process a single query containing all the terms of Q. When sub-queries are conjunctive, a document is a matching result for Q only if it contains all terms of at least one sub-query. This condition can be expressed as a logic function in a *disjunctive normal form*, i.e., a given document matches the multi-query Q if the following logic function is true

$$\sum_{q \in Q} \prod_{t \in q} v_t \qquad (1)$$

We propose to leverage the disjunctive normal form to process multi-queries, assuming all its sub-queries are processed in conjunctive mode. The corresponding multi-query processing strategy (referred to as DNF) is implemented as follows. For each sub-query $q \in Q$, DNF creates an *ephemeral* posting list [6] to iterate on-the-fly over the lists of documents containing all the terms in q. This is equivalent to intersect the posting lists of all such terms, which is typically an efficient operation to perform [8]. Once the ephemeral lists are created, the results for Q are retrieved by traversing these lists in parallel and by scoring the results using some retrieval model in a document-at-a-time disjunctive fashion.

While DNF is efficient, it may need to traverse the same posting lists multiple times (see Sec. 1 for an example). This limits the efficiency of DNF, since the processing time increases with the number of posting lists traversals [8]. In particular, DNF needs to traverse the posting list for a query term t every time the corresponding Boolean variable v_t appears in Eq. (1). Ideally, we would like to keep such logic function as compact as possible by reducing the number of occurrences of its Boolean variables and, consequently, to minimize the number of posting lists traversals.

Factoring permits to represent a logic function in a factored form, i.e., as either a single Boolean variable or as a sum/product of factored forms. For instance, $(v_1 \cdot v_2) + (v_2 \cdot v_3)$ can be factored as $v_2 \cdot (v_1 + v_3)$. A logic function and its factored forms are equivalent, but the factored forms are shorter, i.e., they contain less literals. Therefore, logic function factoring can be used to reduce the number of posting list traversals in DNF.

Brayton illustrates several techniques to factor logic functions [5]. The simplest one, *literal factoring*, recursively factors a logic function by considering it as an algebraic expression, i.e., by ignoring its Boolean nature and by considering logic OR/AND operators as arithmetic sums/products. At each step, literal factoring divides the logic function by its most frequent literal using elementary algebra. Then, the algorithm recursively factors the resulting quotient and rest of the division. The factoring stops when the input logic function of the recursive step is a literal.

In this work, we use literal factoring to optimize multi-query processing as follows. Firstly, we factor the logic function associated to the multi-query to process. Then, for every logic AND operator in the factored form, we generate an ephemeral posting list to iterate over the intersection of the posting lists of its operands. Similarly, for every logic OR operator we generate an ephemeral posting list to iterate over their union. Finally, we use the ephemeral list associated to the root operator of the factored form to traverse the inverted index and to retrieve the matching document for the multi-query. We refer to this approach as *factored DNF*, or FDNF for short. Since Eq. (1) and its factored form are equivalent, DNF and FDNF generate the same query results.

3 EXPERIMENTAL SETUP

The experiments in Section 4 are conducted to address the following research questions:

RQ1. Which multi-query processing strategy gives the best results in terms of effectiveness?

RQ2. Which multi-query processing strategy gives the best results in terms of efficiency?

In our experiments, we compare the effectiveness and efficiency of the different processing strategies for multi-queries: SP-CS, DNF, and FDNF. SP-CS is the state-of-the-art strategy for processing multi-queries when considering their sub-queries as disjunctive [4]. Thanks to its disjunctive nature, SP-CS needs to traverse just once the posting list of the query terms to process a multi-query (see Sec. 2). SP-CS associates a score $s(d, Q)$ to each document d matching multi-query Q as follows:

$$s(d,Q) = \sum_{t \in Q} n_t \cdot s(d,t), \tag{2}$$

where n_t is the number of sub-queries of Q containing term t, and $s(d, t)$ is the document-term score according to some ranked retrieval model (e.g., BM25). The efficiency of SP-CS can be further improved by exploiting *dynamic pruning* techniques [8], and their authors find that the MaxScore algorithm is the most suitable while dealing with multiple query variations. Therefore, we adopt the same optimization in our experiments.

We compare SP-CS's performance with our DNF multi-query processing strategy, which considers sub-queries as conjunctive, and with FDNF, which optimizes multi-queries by performing the logic factoring of the sub-queries. Note that, when $n = 1$, a multi-query coincides with its unique sub-query, and the SP-CS (resp. DNF/FDNF) query processing produces the same results as the processing of a single query in disjunctive (resp. conjunctive) mode. In particular, SP-CS coincides with the traditional MaxScore algorithm, while DNF/FDNF coincides with the ranked AND algorithm [8].

In the following experiments we measure the mean NDCG and recall at cutoffs 10 and 1,000 to evaluate the effectiveness of our multi-query processing strategies. We also measure the mean processing times (over three runs), and the mean number of scored postings for these strategies to evaluate their efficiency.

To build our multi-queries, we use the UQV100 dataset [1]. UQV100 contains 100 topics, associated to human-generated query variations. On average, each topic is associated to about 57 unique query variations. For a given topic, we use its query variations to generate sub-queries. Hence, in our experiments, we generate a multi-query for every topic in UQV100, for a total of 100 multi-queries. We vary the number n of unique variations per multi-query, ranging from 1 to all the available unique variations. In this way, we aim at measuring how effectiveness and efficiency change as multi-queries become more and more complex. Following [2], query variations are selected in decreasing order of popularity, i.e, when $n = 1$, we build the multi-queries using the most popular variation for each topic, when $n = 2$ we use the two most popular variations, and so on. For each multi-query, we retrieve the top 1,000 matching documents and we use BM25 as the underlying retrieval model.

Experiments are conducted using the Terrier search engine[1]. The platform is hosted on a dedicated Ubuntu 16.04.5 server, Linux kernel version is 4.4.0-142-generic. The server has an Intel i7-4770K processor and 32 GB RAM. The inverted index used for the experiments is obtained by indexing ClueWeb12 (Cat. B) corpus, without applying any stemming technique nor stopword removal. The index is kept in main memory, compressed with Elias-Fano.

[1]http://www.terrier.org

4 RESULTS

Table 1 reports the results, in terms of NDCG and recall, for SP-CS and DNF/FDNF, when we vary the number n of sub-queries in the multi-queries from 1 to all the available ones. DNF and FDNF always return the same matching documents as explained in Sec. 2.

As we can see, there is no statistically significant difference in terms of NDCG@10 between SP-CS and DNF. However, the NDCG@1000 of SP-CS is much higher than the one obtained by DNF when processing the multi-queries. This is explainable with the better recall generally obtainable by disjunctive w.r.t. conjunctive processing, since the latter tend to favor precision over recall [8]. When processing a multi-query composed by a single variation, we observe that SP-CS's NDCG@1000 is ~31% higher than DNF. However, DNF's NDCG@1000 increases as more query variations are added to the multi-queries, reducing the gap with SP-CS. When $n = 10$, DNF's NDCG@1000 is only ~3% lower than SP-CS, and less than 0.2% lower when all the variations are used. In fact, multiple query variations help DNF to mitigate the low recall incurred by conjunctive processing, while retaining its high precision.

To conclude on RQ1, we find that the best results in terms of effectiveness are obtained by processing the multi-queries according SP-CS. DNF/FDNF show a similar NDCG@10 (no statistically significant difference) and an inferior NDCG@1000, due to the low recall of conjunctive query processing. However, DNF's NDCG@1000 increases with the number of sub-queries in the multi-queries, being comparable with SP-CS when all the query variations are used.

To address RQ2, Table 2 reports the mean processing time (in milliseconds) and mean number of processed postings for the SP-CS, DNF, and FDNF processing strategies, when using $n = 1, 2, \ldots$ unique queries variations as sub-queries in the tested multi-queries to retrieve 1000 documents. The mean processing times of DNF increase as the number of sub-queries n increases. DNF is faster than SP-CS with few query variation, although it always processed less postings than SP-CS. The improvement in mean processing time oscillates between approximately 1.67× to 1.15×, depending on the value of n.

FDNF is even faster than DNF, thanks to the factoring of multi-queries. As discussed in Section 2, DNF may need to open a posting list multiple times, to process sub-queries in conjunctive mode. FDNF mitigates this issue by factoring the multi-queries, and this explains why FDNF always processes less postings than DNF, with smaller response times.. Moreover, for up to 7 query variations, FDNF improves the mean response times by approx. 1.04× up to 1.95× with respect to SP-CS. On average, FDNF can process four sub-queries in a time close to that required by SP-CS to process just one, with no statistically significant degradation of NDCG@10, as seen for RQ1.

To conclude on RQ2, we find that our proposed FDNF multi-query processing strategy obtains smaller mean processing times w.r.t. SP-CS and our proposed DNF for up to seven sub-queries, allowing to process multiple sub-queries within acceptable time thresholds.

Table 1: NDCG, with cutoff at 10 and 1000, and recall, for SP-CS, and DNF/FDNF when using n unique query variations. The best results for each value of n are in bold. SP-CS's results with △ are statistically significant w.r.t. DNF/FDNF's according to paired t-test ($p < 0.01$).

	SP-CS			DNF/FDNF		
n	NDCG@10	NDCG@1000	Recall	NDCG@10	NDCG@1000	Recall
1	**0.3059**	**0.4335**△	**0.5735**△	0.3000	0.3317	0.3751
2	**0.3391**	**0.4637**△	**0.6079**△	0.3330	0.4023	0.4813
3	**0.3331**	**0.4691**△	**0.6177**△	0.3254	0.4225	0.5206
4	**0.3401**	**0.4774**△	**0.6224**△	0.3329	0.4357	0.5364
5	**0.3480**	**0.4832**△	**0.6282**△	0.3434	0.4546	0.5624
6	**0.3442**	**0.4839**△	**0.6305**△	0.3403	0.4573	0.5712
7	**0.3390**	**0.4855**△	**0.6374**△	0.3372	0.4639	0.5864
8	**0.3408**	**0.4871**△	**0.6401**△	0.3394	0.4693	0.5971
9	**0.3465**	**0.4937**△	**0.6444**△	0.3462	0.4779	0.6046
10	**0.3561**	**0.4990**△	**0.6519**△	0.3551	0.4853	0.6152
all	0.3538	**0.5064**	**0.6634**	**0.3539**	0.5055	0.6593

Table 2: Mean processing time (Time, in ms) and mean number of processed postings (Post., in millions of postings) for SP-CS, DNF, and FDNF when using n unique queries variations. The best results for each value of n are in bold.

	SP-CS		DNF		FDNF	
n	Time	Post.	Time	Post.	Time	Post.
1	95	0.84	49	**0.09**	**47**	**0.09**
2	117	1.06	70	0.21	**60**	**0.19**
3	135	1.24	101	0.33	**84**	**0.28**
4	150	1.42	130	0.46	**104**	**0.39**
5	157	1.46	163	0.61	**122**	**0.50**
6	167	1.56	197	0.75	**149**	**0.62**
7	177	1.65	232	0.89	**170**	**0.72**
8	**201**	1.91	266	1.03	**201**	**0.84**
9	**217**	2.11	295	1.11	**217**	**0.90**
10	**235**	2.22	332	1.25	244	**1.00**
all	**621**	**5.46**	2289	10.46	1699	9.03

5 CONCLUSIONS

In this paper, we addressed the problem of efficiently processing multiple queries, i.e., queries composed by different sub-queries that represents variations of a same information need. Following the standard Boolean model, firstly we proposed to represent a group of sub-queries as a logic function where Boolean variables represent query terms. Secondly, we proposed to process a multiple query in disjunctive normal form (DNF), i.e., processing in parallel every sub-query as an intersection of posting lists and the resulting intersected lists as a union of posting lists. Thirdly, we proposed to apply factoring to the logic function of a multiple query, to produce a more compact but logically equivalent representation, and we presented a new processing strategy for multiple queries based on the factored form of their logic function (FDNF).

We experimented our proposed processing strategies using the TREC ClueWeb12 collection, and the UQV100 set of query variations. As baseline, we selected the state-of-the-art SP-CS processing strategy for query variations. Our experiments showed that both DNF and FDNF do not significantly degrade the effectiveness in terms of NDCG@10 with respect to SP-CS. Moreover, our FDNF strategy can improve by up to 1.95× the mean processing time viz. SP-CS.

As future works, we plan to investigate different literal factoring algorithms, taking into account the properties of Boolean algebra as well as new factoring optimizations, exploiting different sub-queries characteristics, such as the posting list lengths.

ACKNOWLEDGMENTS

We would like to thanks Joel Mackenzie for his help in the assessment of the experimental results. This paper is partially supported by the BIGDATAGRAPES project (grant agreement N°780751) that received funding from the EU H2020 research and innovation programme under the Information and Communication Technologies work programme.

REFERENCES

[1] Peter Bailey, Alistair Moffat, Falk Scholer, and Paul Thomas. 2016. UQV100: A Test Collection with Query Variability. In *Proc. SIGIR*. 725–728.
[2] Peter Bailey, Alistair Moffat, Falk Scholer, and Paul Thomas. 2017. Retrieval Consistency in the Presence of Query Variations. In *Proc. SIGIR*. 395–404.
[3] Nicholas J. Belkin, Colleen Cool, W. Bruce Croft, and James P. Callan. 1993. The effect multiple query representations on information retrieval system performance. In *Proc. SIGIR*. 339–346.
[4] Rodger Benham, Joel Mackenzie, Alistair Moffat, and J. Shane Culpepper. 2018. Boosting Search Performance Using Query Variations. *CoRR* abs/1811.06147 (2018).
[5] Robert K. Brayton. 1987. Factoring Logic Functions. *IBM Journal of Research and Development* 31, 2 (1987).
[6] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2017. Efficient & Effective Selective Query Rewriting with Efficiency Predictions. In *Proc. SIGIR*. 495–504.
[7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[8] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *FnT in IR* 12, 4-5 (2018), 319–500.