Report on "Score-Consistent Algebraic Optimization of Full-Text Search Queries with GRAFT" by Nathan Bales,Alin Deutsch,Vasilis Vassalos

Swati Verma,Susmita Gupta

1 Introduction

Full-text search is an important aspect of many information systems that deal with large document collections with unknown or ill-defined structure. The common full-text search method is to use simple keyword search queries, which are usually interpreted as a disjunction or conjunction of query keywords. Many new and emerging applications, however, require full-text search capabilities that are more powerful than simple keyword search. For instance there are queries in which the user is interested in finding the keywords such that they occur in a particular order or have a particular proximity between them.

Once matching documents are found for the full text queries, the user wants to rank the results based on various ranking functions. The system that does not support different ranking functions is of limited use to the user as he might need to change the ranking function as per his requirement. Thus, a system which can take the ranking function as a parameter and then adjust its plan according to that is needed.

Once the documents are matched, and ranked, the next thing that comes into mind is to find an optimization that can be applied to save on the execution cost. But, the need is to maintain the overall score of the document; otherwise the top-k matching documents will change before and after the optimization is applied. Hence,**score consistency** is an important aspect to be considered.

Thus, the overall challenge is to build a system which supports **generic scoring schemes** and an optimizer that can work **without introducing score inconsistency**. Also, the performance of such a system should be comparable with the systems which use fixed scoring schemes.

1.1 Motivation

The motivation behind the author′s work is the challenge of solving the score-consistency problem found in existing full-text algebras. For a given full text query, the approach proposed in [1] is to extend each match tuple with a score, and each algebra operator with a function to manipulate the scores. As plan evaluation constructs and combines match tuples, it will simultaneously compute and aggregate match scores using the scoring functions. This framework also supports generic ranking i.e. new ranking algorithm can be plugged in by providing new scoring function implementations. But the problem in such an approach is that the optimization does not preserve the document score. The reason for the inconsistency is the encapsulation of score computation into operators that compute the matches. For example, the operators designed for Relational algebra can preserve the matches obtained, but they cannot guarantee the score to be preserved.

1.2 Previous work

The work in [1] provides a new model for structured full text queries i.e. Full-Text Calculus (FTC). FTC was based on first-order logic and an equivalent Full-Text Algebra (FTA) was based on the relational algebra. [1] shows how scoring can be incorporated into these models but as mentioned above, the problem was that the scores were not preserved in such a scheme.

2 Proposed Solution

The paper under consideration also models scoring using similar framework as in [1] but without encapsulating them in relational algebra operators. Instead, scoring functions are standalone aggregate functions that

interact with the other relational algebra operators in the standard way.

To optimize the plan further, with the condition to maintain score consistency, the paper proposes how to interleave matching and scoring, when the optimizer is provided with a fundamental set of properties about the scoring function, so that the optimizer can decide its optimization strategy based on that.

2.1 Matching Calculus and Matching Algebra

The solution proposed in the paper, uses Matching Calculus (MCalc) which is a full text calculus used to specify a set of matches to a full text query. MCalc has a set of predicates which define the relationships between the positions of the keywords. Some of the basic set of predicates of MCalc are HAS(d,p,k) , DISTANCE(p1,p2,n), PROXIMITY(p1,p2,n), EMPTY(ϕ). These individual predicates are disjuncted or conjucted based on the requirement to generate the final query.

After laying down the MCalc queries, the evaluation plan for the queries are expressed in Matching Algebra(MA). Matching algebra operates on, and outputs Match Tables. The Match tables are lists of matches, which are tuples of form $< d; p_0; ::: ; p_n >$ where d ranges over documents, and p_i ranges over term positions and ϕ. Match tables are lists (rather than sets) of tuples; table rows and columns are both sequenced, and tables may contain duplicate rows. MA consists of operators like natural join, outer bag union, selection, projection, anti-join, group and sort.

2.2 Scoring

The results of the query are awarded scores which are the measure of evidence of the connection between the document and the query. For GRAFT, the author considers a class of scoring algorithms, called match-scoring algorithms, that measure specifically the evidence contained in the list of matches to the query against the document.

A score for document d is computed from the query matches to d using the following three-step process:

Step 1: Initialization In this step,the term position p in each table cell is replaced with an initial score value. This score value is typically calculated from term weighting functions TF-IDF[2],BM25[2], KL Divergence[2] etc.

Step2: Aggregation In this step, initial scores are aggregated into a single, but not final, aggregate score value. Three different binary aggregation operators are used the conjunctive combinator,the disjunctive combinator, and the alternate combinator to aggregate the scores.

Step3: Finalization In this final step, a final, floating point, score for a document is computed from the aggregate score produced by Step 2.The aggregate score is a structure, called an internal score, composed of one or more values that are aggregated independently.

In canonical scoring plans, the match tables are passed as input, and the output is a list of scored document.This paper presents an optimized plan called GRAFT(Generic Ranking Algebra for Full Text), which interleaves Scoring and matching algebra operators.This avoids materializing full match tables because; their size could be quite large.

2.3 Optimization

The optimization is the most major step of GRAFT. In this the queries are optimized so as to interleave matching and scoring to avoid materializing the entire match table, while computing the same answers and scores as the canonical score-isolated plan. GRAFT uses various optimization schemes. The optimization schemes that are used by GRAFT are both classical schemes, and novel. The novel schemes are the two new optimization schemes are proposed in this paper. Following are the various optimization schemes in brief:

1. **Sort Elimination** This optimization can be applied when the order of results is not necessary. In such a case,the sort operator could be removed.

2. **Selection Pushing & Join Reordering** These are both textbook relational algebra optimizations.

3. **Eager Aggregation** This is a strategy in which group-by operator is pushed down the plan, which results in eagerly aggregating the matches. Thus, this avoids full materialization of the match table.

4. **Eager Counting** This is a technique that groups n identical tuples into a single tuple with a count value n. When two eagerly counted tuples join, their counts are multiplied. Counted tuples are expanded for aggregation, but otherwise eagerly counted tuples act as regular tuples in the system.

5. **Zig-Zag Joins** This is a special case of sort merge join useful in fully streaming plans/sub plans when each join attribute is indexed. The zig-zag join consumes its inputs by exploiting order to signal the index scan over one join attribute to skip directly to the value of the other join attribute. Zig-zag join signals the index scan operator even if it is several levels down the operator tree, thus bypassing large chunks of intermediate results from earlier joins.

6. **Rank Joins** Top-k optimizations speed up query execution by first exploring the documents that show the highest potential for a high score, and avoiding further exploration of lower scoring documents once the top-K are established.The relational rank-join[4] is a state-of-the-art algorithm for efficiently joining two rank-order tuple streams, producing a rank-order tuple stream.

The two novel optimization schemes proposed by the author are:

1. **Alternate Elimination** This optimization scheme is based on the fact that for constant scoring schemes, alternate aggregation is unnecessary since the score of any match is the document score. Hence, the group-by operators used to aggregate scores may be replaced by an alternate elimination operator.

2. **Pre-Counting** This optimization is useful when the positions are not important in a query. Unless the positions are needed by the scoring scheme, they may be eliminated by the rewrite $A(d;p;k) => \pi(A(d;p;k))$

2.4 Scoring schemes

There are seven scoring schemes from various literatures which the author has chosen as they capture all the scoring algorithms.The seven schemes are prototyped and hence analyzed to see which of the optimizations could be applied on each of them. The seven scoring schemes are as follows:

1. AnySum
2. SumBest
3. Lucene
4. Join-Normalized Weighting
5. Event Model
6. Mean Sum
7. Best Sum+Min Dist

The following table summarizes which of the optimizations could be applied on the above scoring schemes

	Any Sum	Sum Best	Lucene	Join Normal	Mean Sum	Event Model	BestSum +MinDist
τ elim.	✓	✓	✓	✓	✓	✓	✓
⋈ reordering	✓	✓	✓	✓	✓	✓	✓
σ pushing	✓	✓	✓	✓	✓	✓	✓
zig-zag ⋈	✓	✓	✓	✓	✓	✓	✓
forward-scan ⋈	✓						
alt. elim.	✓						
eager agg.	✓	✓	✓	✓	✓		
eager count	✓	✓	✓	✓	✓	✓	✓
pre-count	✓	✓	✓	✓	✓	✓	
rank-join	✓		✓	✓	✓		
rank-union	✓		✓	✓	✓		

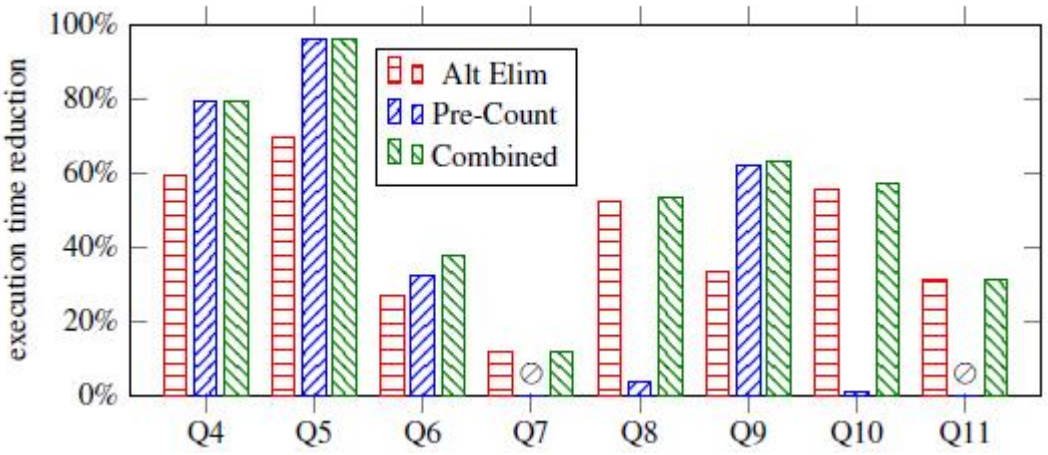
2.5 Experiments performed

The authors conducted experiments to prove the claim that

a) The two new optimization schemes proposed in the paper effectively improve the query performance, beyond classical optimizations.

b) Although there is a known fact that for supporting generic scoring, there would be an overhead involved. Despite of this overhead, the performance of GRAFT frequently exceeds state-of-the-art full-text search systems that do not implement generic scoring.

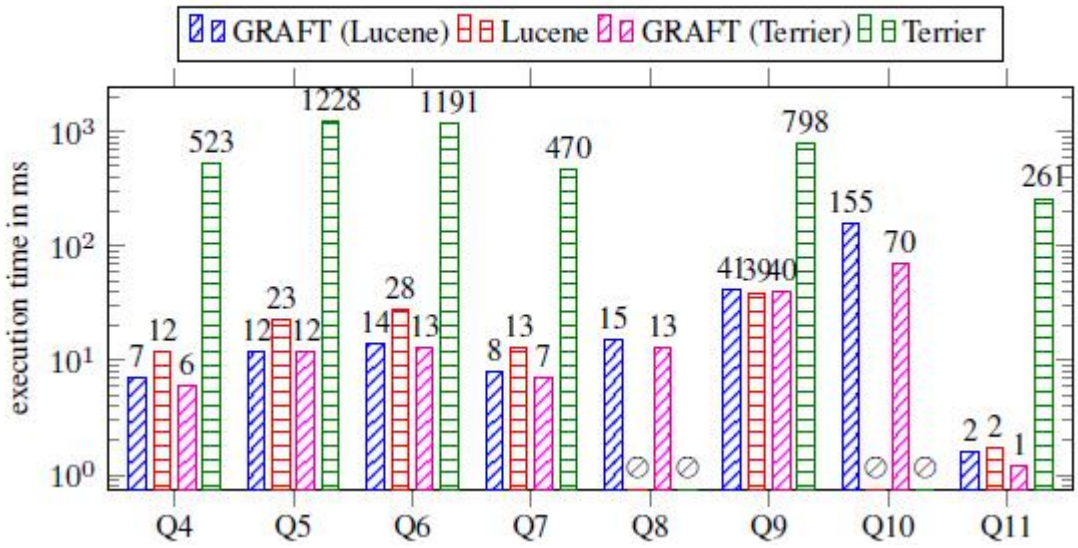
The data on which the experiments were conducted was a snapshot of the English Wikipedia from September 2010. A set of 8 queries was formulated which could cover different predicates of Matching Algebra. These experiments are performed in such a fashion so that the chances of error are reduced to a high extent. The execution time reduction, i.e. the difference between unoptimized and optimized execution time as percentage of the unoptimized time, is reported for all the eight queries with the AnySum scoring scheme. The following figure demonstrates the execution time reduction provided by Alternate Elimination optimization, Pre-Counting optimization, and a combination of both over the classical eager count optimization.



The above results help to infer that Alternate Elimination is most effective when it replaces a group-by operator with moderate to large group sizes. This is because, in this case, a regular grouping operator waits for its inputs to produce every tuple in the group. The alternate elimination operator does not wait, rather it recursively signals operators in its input sub plan to skip further members of the same group, speeding up the sub plan.

For the Pre-counting, the experiments prove that this optimization substantially improves the performance of queries which have their keywords as free.

In the next set of experiments, GRAFT was tested against two state-of-the-art IR systems Lucene[5] and Terrier[6] to show that the additional bookkeeping does not negatively impact GRAFT performance. The following figure demonstrates the execution time of Lucene, GRAFT optimized for Lucenes scoring scheme, Terrier, and GRAFT optimized for Terriers scoring scheme, and hence the shows the improvement that GRAFT does in terms of the execution time.



The first difference between classical Lucene and Terrier, and GRAFT optimized Lucene and Terrier is in the expressive power. Lucene and Terrier support only the PROXIMITY and PHRASE predicates.GRAFT additionally supports DISTANCE, ORDER, WINDOW, and can support as plug-ins virtually any predicate on positions, which can further be extended to support predicates like SAMESENTENCE or SAMEPARAGRAPH.

The experiments show that the results are fast enough for interactive use. Properly optimized GRAFT plans run as fast, if not faster than both Lucene and Terrier. Lucene and Terrier do not support some queries(Q8 and Q10 as in figure) because they do not support the WINDOW predicate.

3 Discussion

The paper shows how to build an optimizer that can take a scoring scheme as a parameter and perform optimization on the plan without introducing score inconsistency. The claim of having a scoring model to support various scoring algorithms in a generic manner and then to optimize such a plan preserving the overall score of the document is supported with the help of detailed experiments.

3.1 Observations

1. The scoring algorithms considered in this paper are only based in match scoring schemes. The work could be further extended to see if the model holds good for other scoring algorithms like probability based schemes [3]. In the probabilistic relational algebra(PRA)[3] tuples are assigned probabilistic weights giving the probability that the tuple belongs to a relation.

2. The author claims that proposed methodology would work for different scoring schemes. But this claim is not backed by experimental evidences. Although the author claims that various scoring algorithms were prototyped, but the results are not shared with and without GRAFT for those prototyped models.

3. The two new optimization schemes proposed by the author are meant for scoring schemes which are having properties of having CONSTANT scoring and NON POSITIONAL scoring. An extension to the work could be to have further optimization schemes for queries which are having position of the keyword in the document as a required factor.

4. The experimental implementation of GRAFT is single threaded. A multi threaded approach could also be used to exploit the features of parallel processing wherever possible. This might further improve the performance.

5. The author claims that GRAFT avoids producing large intermediate results by interleaving matching and scoring. A supporting experiment would have been useful in terms of memory peaks while processing in classical schemes versus GRAFT, through which the claim could be proved well.

6. A further direction to reduce the space cost could be to optimize the INPUT table, where the token offset for each tokens are given. The #DOCS field could be safely removed from that table based on the type of query the user is interested in.

7. The cost model that is involved in optimization has not been discussed in the paper. Generally, optimizers cost query plans using a mathematical model of query execution costs that rely heavily on estimates of the cardinality, or number of tuples, flowing through each edge in a query plan. Cardinality estimation in turn depends on estimates of the selection factor of predicates in the query.

4 References

1. C. Botev, S. Amer-Yahia, and J. Shanmugasundaram. Expressiveness and performance of full-text search languages. In EDBT, 2006.

2. C. D. Manning, P. Raghavan, and H. Schtze. Introduction to Information Retrieval. Cambridge University Press, 2007.

3. N. Fuhr, T. Rolleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. ACM TOIS 15(1), 1997.

4. I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. VLDB J., 13(3), 2004.

5. The Apache Foundation. Lucene. http://lucene.apache.org/.

6. I. Ounis, G. Amati, P. V., B. He, C. Macdonald, and Johnson. Terrier Information Retrieval Platform. In ECIR 2005. Springer.