Querying large Collections of Mathematical Publications - NTCIR10 Math Task -

Moritz Schubotz
Technische Universität Berlin
schubotz@tu-berlin.de

Marcus Leich
Technische Universität Berlin
marcus.leich@tu-berlin.de

Volker Markl
Technische Universität Berlin
volker.markl@tu-berlin.de

ABSTRACT

In this paper, we present our approach for searching mathematical formulae. We focus on a batch query approach that does not rely on specialized indexes, which are usually domain dependent and restrict the expressiveness of the query language. Instead, we use Stratosphere, a distributed data processing platform for Big Data Analytics that accesses data in a non-indexed format. This system is very effective for answering batches of queries that a researcher may wish to evaluate in bulk on large data sets.

We demonstrate our approach using the NTCIR10 Math task, which provides a set of formula patterns and a test data corpus. We showcase a simple data analysis program for answering the given queries. We interpret the patterns as regular expressions and assume that matches to these expressions are also relevant search results to the end-user. Based on the evaluation of our results by mathematicians from Zentralblatt Math and mathematics students from Jacobs University, we conclude that our assumption holds principally with regard to precision and recall.

Our work is just a first step towards a well-defined query language and processing system for scientific publications that allows researchers to specify their information need in terms of mathematical formulae and their contexts. We envision that our system can be utilized to realize such a vision.

Team Name

FormulaSearchEngine(FSE)

Subtasks

Math Retrieval (English): Formula Search (FS), Full Text Search (FS)

Keywords

Math Search, MathML, Stratosphere, Query Language

1. INTRODUCTION

As research and development happens globally, fast and uncoordinated, researchers must have quick access to relevant related topics in order to advance science. In this context mathematical formulae play an essential role in scientific communication.

A system assisting in literature search should ideally understand the researchers' language, including formulae. Here, the biggest problem is the ambiguity and complex structure of natural language. We propose to use a query language with fixed and well-defined semantics that describes information need in terms of formulae and text. The main advantage of such a query language in contrast to natural language is that the query results are well-defined. This separates the hard research problem of *transforming researchers questions to formal queries* from the technical challenge of query execution.

The *NTCIR10 Math pilot task*[1] provides a set formula/-text patterns without fixed semantics and a reference corpus that consists of 100 000 documents from the Cornell ePrint arXiv[1]. These patterns are split into two sub tasks. For the Formula Search subtask (FS) the patterns are a list formulae with (back-referencing) wild-cards (table 2). The Full Text subtask (FT) provides words in addition to such patterns (table 3). Since there is no meta information on how to process these queries, we interpret the formula queries very strictly, i.e. we only return results that match the pattern without any implicit semantics. Furthermore we treat the given words as a space separated list of keywords.

Recent approaches to formulae search, e.g. [7], focus on matching the tree structures of formulae and patterns. A lot of development effort has been put into the efficient and distributed execution of such substitution tree-based queries [6]. These approaches [7, 8, 6] consist of two phases. In a first step, the content is indexed and in a second step queries are answered based on the information contained in the index. Obviously the advantage of this approach is the short response time, if the query can be answered using the data stored in the index. One drawback is that the corpus must not change after indexing. Even though if update operations are possible, they are connected with computational overhead. Systems based on Lucene [4] for example, allow for temporary updates that require index re-builds in the long term.

In our approach, we focus on the quality of the result, rather than run-time. We choose to improve the quality of our algorithm iteratively, until the result provide a real value add for researchers. Following the concept of rapid prototyping, our approach speeds up the development effort dramatically, and allows to focus on the core functionality i.e. to decide if a formula matches a query. The management of the data volume is done by the Stratosphere platform in the background. In the context of the NTCIR10-task, we showcase that our system, which was developed in a less than two person months, is able to filter and rank formula that match the query.

Currently, we do not address strategies for optimizing the run time and the ease of use. For theses optimizations, one can learn from elaborated concepts for query optimization originating from the database community, which is subject to future work.

2. SYSTEM DESCRIPTION

We use Stratosphere [2] as a platform for executing the queries specified in the NTCIR10-Math task. Stratosphere consists of three layers.

- Meteor [5] the high level scripting language
- The PACT [2] programming model for implementing operators
- The Nephele [2] execution engine for parallel computation

To solve the NTCIR10 tasks, we identified following processing steps.

1. load the data
2. parse queries and data
3. filter data based on queries
4. calculate individual ranking function
5. evaluate ranking functions based on whole-corpus
6. return top ranked results
7. export the results

In this section, we describe how we execute these steps. Thereby we outline, which parts are evaluated by Stratosphere's native code, and which parts are performed by our user defined code. Due to the easy extensibility of the Stratosphere platform that allows to integrate user code as well on the Meteor as on the PACT level, very little overhead is required to extend the core functionality.

2.1 Data preparation

After downloading and extracting the NTCIR10 test collection, we concatenated the included files to a single one that was transferred to our distributed file system (HDFS). A single file is more efficient, since the system can read the data sequentially from disk. The content of this file has the following structure `<ARXIVFILE Filenam=$filename> $file-content</ARXIVFILE>`.

The resulting file was stored in a distributed file system so that all computers that have been used to process the data could read the required data block wise from local disk.

2.2 High level program design

After having imported the data to our environment, we designed the principal data flow of the system. Therefore we had to specify the inputs and outputs and the type of each operator. Stratosphere extends the MapReduce [3] concept and introduces new operator types like *Cross*, *Match* and *CoGroup* [2]. However, for this task the operator types *Map*, *Reduce*, *Cross* and *Match* are sufficient. These operators can be regarded as second order functions that execute

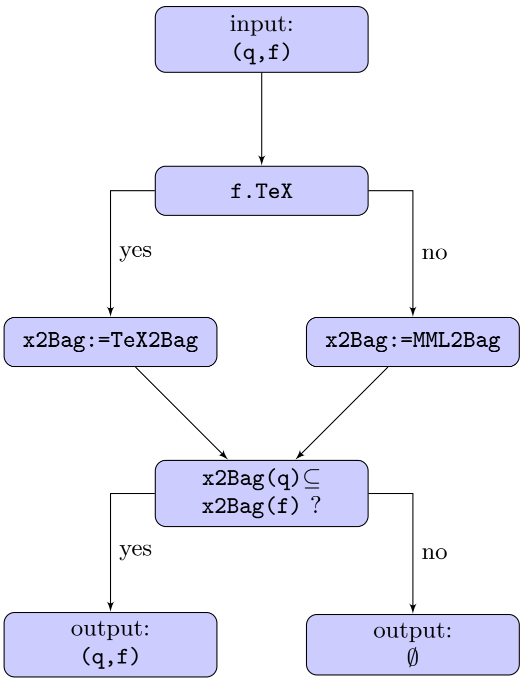

Figure 1: Logic of `formulafilter` operator: Filters formula that contain all tokens specified in the query.

arbitrary user code based on the following number of inputs and outputs record

$$Map : \mathbf{R} \rightarrow \mathbf{R}^* \tag{1}$$
$$Reduce_k : \mathbf{R}^* \rightarrow \mathbf{R}^* \tag{2}$$
$$Cross : \mathbf{R}^2 \rightarrow \mathbf{R}^* \tag{3}$$
$$Match_{k,l} : \mathbf{R}^2 \rightarrow \mathbf{R}^* \tag{4}$$

Here $\mathbf{R}$ denotes the space of all possible records and $\mathbf{R}^*$ the Kleene Closure of $\mathbf{R}$. Each record $r \in \mathbf{R}$ consists of a finite number of fields r_i.

The *Map*operator gets one record at a time as input and can output zero or more records for each input record.

In contrast to the original *Reduce* operator [3], which gets all records with the same key, the Stratosphere *Reduce* operator (Reduce$_k$) has an additional parameter k that defines the index of the record field that is used as key. As a consequence the input record set $\mathbf{I} \subset \mathbf{R}^l$ of length l is portioned into $n = |\operatorname{dom} \mathbf{I}_k| \leq l$ lists of records, each having the same value for field k. Here n is the number of distinct values for $\mathbf{I}$. Thus, exactly one list will contain a record that has a particular key-value for field k. For the special case, $n = l$, *Map* and *Reduce* are equivalent. Furthermore, the *Reduce* contract has the option to sort the input list with regard to a secondary field that we use to get a sorted list of the query result.

Cross, receives the Cartesian product of the input record sets and *Match* is a *Cross* followed by a filter that only emits record pairs with matching key attributes $r_k = r_l$.

2.3 Implementation of the operators

This section describes the user defined operators that we developed extending the core functionality of the Stratosphere system in detail. The principal data flow is the following: First the queries (section **??**) and the data (section 2.3.2) is loaded. In a subsequent step (section 2.3.4), token based query filters identify the equations that are match candidates.

Independent, overall word- and variable frequencies are obtained (section 2.3.3).

In the following scoring phase (section 2.3.5), we compare the equation structure of query and match candidate and calculate a score that takes into account the overall frequencies. Finally, the results are sorted (section 2.3.6) and the top scored hits are returned.

2.3.1 Query compilation

The query compilation is implemented via a *Map* operator (listing 1 [line 5-12]) that gets one record of a single text field with the NTCIR topic XML subtree, which includes formula patten and keywords, as input. From that, we derive exactly one output record with six fields that contains the compiled query for further processing. To illustrate this process, we use the query NTCIR10-FT-14 ($\sum ?p_n ?a_n$, `convergence`) as running example.

The first field is the query number `NTCIR10-FT-14` that is extracted using an XML library and is used as a *Reduce* key for the final ranking.

The second field is the multi set `[n x 2, sum,  _ x 2]`, which is generated by our TEX token filter from the TeX-Query (\sum \qvar{p}_{n} \qvar{a}_{n}) based on the build in Java tokenizer. In the same way the presentation MathML tokenizer is used to generate the third filed that contains `[Σ, n x 2]`.

Obviously the TEX token filter contains more information compared to the MathML filter. The latter extracts variables, numbers, and operators only, rather than the full feature set of MathML elements, which would require an unified MathML representation. Unification of MathML is not yet solved, or at least not available as a program library. Without unification, we face a lot of problems due to degrees of freedom in the MathML standard. For example the underscore (_) in the TEX code could be represented via the `msub`, `msubsup` or `mmultiscripts` presentation MathML[2] element. Thus, the we use the MathML filter only, if no TEX-code is available (figure 1).

In order to speed up the execution, we compress the MathML mark-up. The recursive compression, e.g.,

```
<a b="c">
  <qvar name="x">
  <d>
    f                    → a[ (.*?); d[f]; \1 ]        (5)
  </d>
  <qvar name="x">
</a>
```

is inspired by Mathematica's full form style with square brackets. We ignore XML-attributes and replace the `qvar` elements (*?a* and *?p* in the running example) by regular expression. Multiple occurrences of the same name attribute of the qvar element are treated with back-reference as in (5). That said, the fourth and fifth field read *mrow[mo[Σ]; mrow[msub[(.*); mi[n]]; msub[(.*); mi[n]]]]* and *apply[sum; apply[times; apply[csymbol[subscript]; (.*); ci[n]]; apply[csymbol[subscript]; (.*); ci[n]]]]*. These regular expressions take into account the structure of the query, rather than just counting the features as done in the MathML filter.

The last field is a list with keywords (convergence in for the sample query). One attempt is to include the porter stemming algorithm [9]. Even though this algorithm was quite successful in standard NLP tasks, it has some drawbacks for math search. For example the words `derivative` and `derivation` are both transformed to the stemmed term `deriv`. Thus, we just use the original lower-case string for the text search.

2.3.2 Extraction operators

This section describes the extraction of variables and terms from the test-corpus data. Our four extraction operators are implemented as *Map* contracts. The first *Map* (listing 1 [14-17]) gets a publication formatted as HTML-document and shipped in a PACT record with only a single string field as input. It attaches another field that contains the file identifier, for the later use as key.

The consecutive formula filter operator (listing 1 [19-23]) uses a regular expression to find all math tags. It emits one record per formula containing the MathML-element and the document id.

Another regular expression captures TEX-code of the formulae, if available and attaches an additional field to the record.

The following *Map* task (listing 1 [25]) tokenizes the TEX and MathML code respectively, the same way as done with the query TeX/MathML. Even though these tasks are different operators, they share code for the tokenization.

Another *Map* task (listing 1 [26]) tokenizes the words of the article in the same style.

2.3.3 Aggregation operators

As a next step, we calculate the total number of words and variables via bag union (listing 1 [25-26]). The bag union operator is implemented as a combinable *Reduce* contract. This means, it merges the multi sets containing the words or variables of a document or formula in two steps. First, all nodes compute the union of their local all multi sets than these intermediate multi sets of all nodes are united. We use the multi set implementation of the guava libraries[3], which turns out to be quite efficient with regard to memory usage and runtime. As a result there are two records, one for the variables and one for the words. Each contains one field with a multi set of the counts.

2.3.4 Filter operators

The crucial part of the execution is the formula filtering that identifies the match candidates for the queries based on the formula tokenization. Logically this works as shown in figure 1. Technically, the formula filter *Cross* task gets a tuple of records as input that originates from the Cartesian product of query and formula records. If the multi set of the tokens of the formula is a superset of the tokens in the query than the formula is regarded as hit candidate. In that case the fields of the query record and the fields of the formulae record are concatenated and emitted. Otherwise, other nothing is emitted.

We use a similar *Cross* task for the filtering of the documents. Only documents that include the keywords, specified in the query, pass the filter. For those, we emit a record that contains the concatenation of the query and document fields.

```
using math, xml; //load the custom package
$d = read from "hdfs://localhost/NTCIR.xml" split at "ARXIVFILE";
$q = read "hdfs://localhost/queries.xml" split at "topic";
//map-task for query compilation
$q = transform $q into{
  num: xGet($q,"./num"),
  texFilter: TeX2Bag(xGet($q,"./TeXquery")),
  mmlFilter: mml2Bag(xGet($q,"./pquery")),
  pmml: compress xGet($q,"./query/pquery"),
  cmml: compress xGet($q,"./query/cquery"),
  words: split xGet($q,"./query/words"),
};
//map-task for file ID extraction
$d= transform $d into {
  fileID: xGet($data,"/.@Filename","f(\d{6})\.xhtml","$1"),
  HTML: $d
  }
//map-task for formulae extraction
$f = xtransform "//math" as $m in $d.HTML into{
  $fileID: $d.fileID ,
  $mml = $m,
  $tex = xGet($m,"/.@altText")
}
//map-task formulae tokenization
$f = addFields $f { vars = $f.TeX ? TeX2Bag($f.TeX):mml2Bag($f.math) };
$d = addFields $d { words = HTML2Bag($d.HTML)};
//reducer for counting total number variables
$vars= bag_union $f[*].vars;
$words = bag_union $d[*].words;
//cross-task for formulae filtering
$fq = formulafilter $q in $f;
$dq = textfilter $q in $d;
//cross-task that assignes scores with a TFIDF-like method
$fq = score $fq use $vars;
$dq = score $dq use $words;
//join-task: increase formula by document score
$fq = raisescore $fq use $dq where $fq.num=$dq.num and $fq.fileid=$dq.fileid;
//reduce-task concatinates results for each query
$results = formulagroup $fq by $fq.num sort desc by $formulae.score limit 30;
//reduce-task formats results
$result =format results $results parallel 1;
write $result to "hdfs://localhost/results.xml"
```

Listing 1: Meteor pseudocode: Script for answering the Fulltext search task. For a detailed description of the operators see section ??

2.3.5 *Scoring operators*

After the filtering step the result set is still too large and has to be ordered. Therefore, the scoring task is used. Since we want to use TFIDF like methods to score the tokens, we need to know the total number of occurrences of the tokens.

Foreclosing the evaluation, the following statement can be made: We developed a scoring mechanism that calculates a score based on the absolute and relative number of tokens, presentation and content MathML, as well as the occurrence of the keywords. However, the ranking results are poor, because free parameters that are used to calculate a reasonable norm for this score vector were specified with an ad-hoc-method.

One of future research tasks, is to adjust the free parameters in a way that the ranking results correlate better to the experts relevance ratings. Certainly, during that task over-fitting must be avoided.

For the scoring of the MathML part, the MathML tags were compressed in same way as done for the queries. After that, the regular expressions are applied to the compressed MathML. Thus, even the instances of the place-holders were specified, which is valuable for displaying the results in a future use case.

We demonstrate the scoring function using the sample query FT-14 and the relevant formula $\sum a_n b_n$ 190/f075790 .xhtml#id73874 that is ranked best. The final score of 10155 is calculated in the following way:

Since all tokens are found in the TEX-code a score of 100 is assigned. In addition, the score for the content MathML match (with `$1=ci[a]` and `$1=ci[b]`) was scored with 5000. Additional 5000 points are assigned, because there are no other expressions in the content MathML representation. Presentation MathML did not match by accident, because there is an additional invisible times between a_n and b_n that is not removed, while compressing the presentation MathML. The seven occurrences of the word `convergence` are rated inspired from the geometric series with $2^{-7}(2^{7+1} - 1)$ wordscore(convergence). Here, the word score is obtained from corpus wide frequency of `convergence` in relation to the sum all relevant word frequencies. In the same way token scores for $n, \sum, \backslash, \_$ are calculated.

2.3.6 *Result operators*

There are two result operators. The first one groups formula that have the same query number and the second one summarizes the result of all queries.

The input for the first *Reduce* operator is sorted according to the score specified in descending order. We modified the normal grouping in the way that the operator only collects the first 30 input records and normalizes them according to the NTCIR submission guidelines.

In former steps, the results of all tasks are concatenated to the XML result file that was downloaded from the HDFS and was sent directly to the NTCIR office without manual modifications.

2.4 Limitations

We designed our system in a way that it produces deterministic and traceable results for the given queries. Our conservative approach implies that the result set must be explainable and reproduceable. This allows for easy debugging and testing of the code. However, verbose justification statements lead to a reasonable amount of development work, and influence the performance of the system in a negative way. Especially early selection of the probably best results and random sampling gets impossible with our approach. Furthermore, no query expansion is performed. For example, if the query is x^2 but no equation that contains x^2 exists, the system would answer with an empty result set, rather than displaying results for x or y^2. For that reason some of the results that were considered as partial match by the experts could not be found by our system.

Furthermore, we were not able to discover equations that involved a line break in the MathML-source code, which is a corner case. Additionally only equations that contained an `alttext` attribute were considered. After fixing these bugs the performance of our system has slightly improved.

For the full-text search, we interpreted the text input as a list of keywords. One query contained the word `not (parseval)`, which probably means that the word `parseval` should be excluded. However in this first version of our system, we didn't treat this special case.

Furthermore we ignored the task FT-3 that deals about searching for LaTeX-pseudo-code.

2.5 Optimized execution

Instead of using the Meteor to compile the source from listing 1 automatically to the Pact layer, we hard code the pact plan. In this context, we perform some manual performance optimizations and customizations. We just count the keywords and tokens that occurred in one of the search queries, for example. Furthermore, we summarize concurrent *Map* task, like for example extraction of the TeX code and the tokenization of it. In addition, we specified compiler hints for output cardinality estimation. Especially, for choosing *Cross* and *Match* strategies these estimates make a difference. The query list has about 30 entries and is much smaller than the number of equations with more than 130 million records. Thus a broadcasting the queries and keeping them in memory for the whole time the task runs makes sense.

3. RESULTS

The *formulasearchengine* team (FSE) submitted 373 results in the category formula search (FS) and 244 results in the category formula search (FT).

In the category FS 290/373 results were judged. 106 of the evaluated results were regarded as (partially) relevant. Thus the precision (relevant/submitted) results evaluates to 28.4% for formula search which is rank 2 of 13 submitted result sets.

For the Full Text search task all 244 submitted results were judged. 54 of them were regarded as (partially) relevant and the precision evaluate to 22.1%. In this category only two teams participated. The other team achieved 31% precision.

Even though we do not perform a detailed performance analysis, we present some indicators of the system runtime. For the actual run our system, consisting of two desktop computers with 8 CPU-cores 1 HDD and 16GB main memory per machine, reported an overall runtime of about 9 minutes (512 207 ms). As an indicator for the shortest possible runtime on our system, we read all the data and write the first 1000 bytes of each file back to disk. This took less than 4 minutes (226918-228194 ms) and meets our expectations, since processing the 42GB dataset with a single disk, and an average effective data transfer rate of 100MB/s leads to 7 minutes reading time in theory.

For a detailed analysis of the results, we performed another run that ignores all results that were not rated by the experts, and provides output even if the formula would have been suppressed by the filter. We published the result of this analysis at `http://www.formulasearchengine.com` in full length and elaborate on some crucial aspects in this section.

Our score s that was used for ranking is partitioned in 3 areas and relates to the experts ratings relevant (++), partially relevant (+) and not relevant (o) via

$$\begin{aligned} s < 50 &\to \text{no match} && \leftrightarrow o\\ 50 \le s < 2000 &\to \text{token only} && \leftrightarrow +\\ s \ge 2000 &\to \text{token + filter} && \leftrightarrow ++. \end{aligned}$$

In table 2 and 3, 9×9 matrices (M) for all results show the relationship between expert rating and the classification of the system score. The diagonal entries denote that our system and the reviewers agree, entries in the upper triangle of the matrix mean that our system rates the result as more relevant than the reviewers did, and entries in the lower triangle denote the opposite. They relate to the classical 2×2 binary classification matrix(B) by summarizing the non corner entries via

$$\begin{aligned} B_{i,j} = M_{2i,2j} &+ \alpha M_{2i,2j+1} + (1-\alpha) M_{2i,2j-1}\\ &+ \beta M_{2i+1,2j} + (1-\beta) M_{2i-1,2j}. \end{aligned} \tag{6}$$

For example summarizing as well relevant(++) as partially relevant(+) to relevant ($\alpha = 1$) and regarding all entries beginning from token only (score > 50) as retrieved entries ($\beta = 1$) leads to $B_{0,1} = M_{0,2} + M_{1,2}$ false positives. The β parameter is not relevant for the further discussion since, the calculated score is more fine grained than shown in the table. Thus, we calculate the average precision $\langle P \rangle$ defined as $\langle P_\alpha \rangle \equiv \int p_\alpha(r)\,dr$. For $r \in R_\alpha$, the set recalls of the ranked result list $p_\alpha(r) \equiv \max\{p'_\alpha(k) : r = r_k\}$ is the maximal precision for $p'_\alpha(k)$ the precision for rank k with recall r_k. Since, this are only discrete values, $\langle P_\alpha \rangle$ is approximated in upper Riemann sum style by using $p_\alpha(r) \equiv p_\alpha(\min\{r' \in R_\alpha : r' > r\})$ for $r \notin R_\alpha \wedge r < \max R_\alpha$ and $p_\alpha(r) \equiv 0$ for $r > \max R_\alpha$.

In the tables, we use the more intuitive notation $\langle P_{++} \rangle = \langle P_0 \rangle$ and $\langle P_+ \rangle = \langle P_1 \rangle$. Furthermore the mean average precision $\langle\langle P \rangle\rangle$ was calculated by averaging over all $\langle P \rangle$ for each subtask. Comparing the result of $\sim 30\%$ for FS and $\sim 15\%$ for FT shows that especially the combined search needs to be improved and indicates that keyword search is not sufficient.

3.1 Qualitative evaluation

In the following subsection, we explain the results based on some examples. Therefore we group our observations into false negatives and false positives.

3.1.1 False negatives

Regarding our running example query (FT-14). The formula 132/ f052473#idp561824: $\sum_{\alpha\in\mathbb{F}_n^+} a_\alpha Z_\alpha$ did not pass the token-filter (figure 1), since the query contains two n, whereas the formula has only one. However, it is one of the seven relevant formulae. This is an argument for lowering the filter barrier further in the future. One option is, to use a set rather than a multi set. On the other hand 76 of the 100 evaluated formulae passed the filter. Thus adjusting the filter-granularity a meta level might not be the option of choice. Looking at more examples indicates that it might be reasonable to improve the filter on a token level. For example if the token `/` could be regarded as a match for the token `frac` as well. According to (14) 5 additional entries were ranked in the intermediate category. Three of them originate from document 15/f005755 that does not involve the specified keyword *convergence* and were down ranked even though they are a perfect structure match. The other two use a specified sum ($\sum_{n=1}^{\infty}$) rather than the unspecified $\sum$ which prevents them from a MathML structure match, since this would require query expansion.

3.1.2 False positives

The two best ranked false positives ($\sum \lambda_n Q_n, \sum \epsilon_n x_n$) for the running example originate from documents that contain

Table 1: Overview of the top 10 result: The queries that had less than 10 relevant results are not displayed. The symbol ++ denotes relevance, + partially relevant and o not relevant.

	formula search (FS)						fulltext search (FT)			
#	5	8	16	18	20	21	1	8	9	15
1	++	++	++	++	++	++	++	+	+	++
2	++	++	++	++	+	++	++	+	+	++
3	++	++	++	++	++	++	o	+	+	+
4	++	++	++	++	++	++	++	++	+	o
5	++	++	++	++	+	++	++	++	o	++
6	++	+	++	++	+	++	++	+	o	++
7	++	++	++	++	+	++	++	+	o	++
8	++	++	o	++	o	++	++	o	o	o
9	++	++	++	++	o	++	++	o	o	o
10	+	++	++	++	o	++	++	+	o	++

all required keywords. However, the hit were considered as not relevant by the experts. The reason for that can not be determined without considering the context of the equation. Therefore, it will be important to consider the context of the hits in the future. The same argument holds for the second best ranked hit $\sum a_n b_n$ that was classified as partially relevant. Due to the fact that the TEX-filter contains only two symbols (section 2.3.1) the relative large number of 63 *false positives* for the filter is obvious.

3.2 Ranking

Even though we do not focus on ranking, the results are reasonable, especially for cases were more than 10 relevant results were found by the experts. In table 1, we list those queries and print out the experts ratings compared to the ranking position. According to this measure most results are quite relevant. However since only 6 of 22 (for FS) queries lead to ten or more relevant results, it has to be remarked that this good ranking holds for the "easy" queries. In a further step of investigation we investigated the impact of the fine ranking. For the detailed evaluation in tables (2 and 3) we calculated the mean average precision based on a numeric score with two internal decimal places. Rounding to natural numbers has a large effect on the overall result. Our evaluation shows that just improving the inner decimal digit score can lead to an increase in mean average precision for almost 50% ($\langle\langle P_{++}\rangle\rangle$ increased from 31.5% to 46.6% and $\langle\langle P_{++}\rangle\rangle$ increased from 28.5% to 44.4%). This indicates that TFIDF methods have to be checked carefully in the future.

4. CONCLUSION

In this paper we present a batch oriented approach to formula search that is characterized by its short development / test cycles. We achieved rank 2 of 13 in the formula search subtask with regard to precision of partially relevant hits. However, the achieved precision of our system is not satisfying, yet. In general our two phase approach of token filtering and structure matching seems to be viable and will be investigated further.

Our main focus for future work will be query expansion. Given a pattern that includes the identifier i it would be beneficial to consider formulae that include the identifier j, if j is known to be the imaginary unit. The first requirement for this kind of query expansion is the integration of variable definition detection in the surrounding text which requires deep parsing of the text and analysis of the equation. This way our approach can incorporate valuable information that would be lost, if only equations are considered. Due to the absence of precomputed structures, such as indexes, this extension is comparatively easy to implement in the next iteration of our system.

The second requirement for query expansion is a clear definition of the actual semantics of the query patterns which was not available for this competition. Additionally, we believe that query patterns should provides means to tag identifiers with meta information, such as "i is the imaginary unit," so the identifiers in the pattern can be matched better to the identifiers in the corpus.

Acknowledgements. Thanks to Alan Akbik, Holmer Hemsen and Michael Kohlhase for the fruitful discussions and Jochen Adamek for proofreading this paper. This work is funded by the European Commission under FP7 Project No. 296448 – "Data Supply Chains for Pools, Services and Analytics in Economics and Finance."

5. REFERENCES

[1] A. Aizawa, M. Kohlhase, and I. Ounis. Ntcir-10 math pilot task overview.

[2] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: A programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 119–130, New York, NY, USA, 2010. ACM.

[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[4] B. Goetz. The lucene search engine: Powerful, flexible, and free. *JavaWorld. Available http://www. javaworld. com/javaworld/jw-09-2000/jw-0915-lucene. html*, 2000.

[5] A. Heise, A. Rheinländer, M. Leich, U. Leser, and F. Naumann. Meteor/Sopremo: An Extensible Query Language and Operator Model. *stratosphere.eu.*

[6] M. Kohlhase and C. C. Prodescu. Scaling an Open Formula Search Engine. *Challenge*, pages 1–15, 2012.

[7] M. Kohlhase and I. Sucan. A Search Engine for Mathematical Formulae. In T. Ida, J. Calmet, and D. Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[8] M. Líška, P. Sojka, M. Růžička, and P. Mravec. Web Interface and Collection for Mathematical Retrieval : WebMIaS and MREC. In P. Sojka and T. Bouche, editors, *DML 2011: Towards a Digital Mathematics Library*, pages 77–84, Brno, 2011. Masaryk University.

[9] P. Willett. The porter stemming algorithm: then and now. *Program: electronic library and information systems*, 40(3):219–223, 2006.

Links

[1]http://www.arxiv.org
[2]http://www.w3.org/TR/MathML3/chapter3.html
[3]http://code.google.com/p/guava-libraries/

Table 2: Statistics for NTCIR10-FS

(1) $\int_0^\infty \mathrm{d}x \int_x^\infty F(x,y)\mathrm{d}y = \int_0^\infty \mathrm{d}y \int_0^y F(x,y)\mathrm{d}x$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	1	1	2
<5000	1	30	68	99
$\sum$	1	31	69	101
$\langle P\rangle$ in %	0.0	3.1		

(2) $X(i\omega)$

	++	+	o	$\sum$
>2k	0	0	16	16
5000-2k	0	0	33	33
<5000	0	2	53	55
$\sum$	0	2	102	104
$\langle P\rangle$ in %	-	0.0		

(3) $x^n + y^n = z^n$

	++	+	o	$\sum$
>2k	7	2	0	9
5000-2k	2	8	24	34
<5000	4	12	42	58
$\sum$	13	22	66	101
$\langle P\rangle$ in %	59.6	41.2		

(4) $\int_{-\infty}^{\infty} e^{-x^2}\, dx$

	++	+	o	$\sum$
>2k	3	0	0	3
5000-2k	6	21	21	48
<5000	5	15	31	51
$\sum$	14	36	52	102
$\langle P\rangle$ in %	27.7	35.8		

(5) $\frac{f(x+h)-f(x)}{h}$

	++	+	o	$\sum$
>2k	13	4	0	17
5000-2k	2	3	24	29
<5000	23	18	14	55
$\sum$	38	25	38	101
$\langle P\rangle$ in %	28.0	30.7		

(6) $\sqrt{2} = 1 + \frac{1}{3} + x{-}y$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	8	36	44
<5000	0	17	41	58
$\sum$	0	25	77	102
$\langle P\rangle$ in %	-	5.0		

(7) $sin(x)/x$

	++	+	o	$\sum$
>2k	5	11	0	16
5000-2k	0	6	9	15
<5000	5	10	59	74
$\sum$	10	27	68	105
$\langle P\rangle$ in %	23.5	55.0		

(8) $ax^2 + bx + c$

	++	+	o	$\sum$
>2k	19	3	3	25
5000-2k	1	0	18	19
<5000	25	3	29	57
$\sum$	45	6	50	101
$\langle P\rangle$ in %	36.8	42.5		

(9) $\frac{e^x+y}{z}$

	++	+	o	$\sum$
>2k	0	28	13	41
5000-2k	0	2	5	7
<5000	0	10	45	55
$\sum$	0	40	63	103
$\langle P\rangle$ in %	-	46.3		

(10) $f^n(z)f^{(k)}(az) \neq c$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	1	44	45
<5000	0	12	43	55
$\sum$	0	13	87	100
$\langle P\rangle$ in %	-	0.3		

(11) $\int_{g\neq 0} |\nabla f|^q dx \leq c \int_{g\neq 0} |\nabla(f+g)|^q dx$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	9	4	13
<5000	0	33	54	87
$\sum$	0	42	58	100
$\langle P\rangle$ in %	-	16.2		

(12) $q_n|a_n - a| \sim_{n\to+\infty} q_n|\frac{p_n}{q_n} - a|$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	8	0	8
<5000	0	18	74	92
$\sum$	0	26	74	100
$\langle P\rangle$ in %	-	30.8		

(13) $N_{k)}(r, \frac{1}{f-a})$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	0	46	46
<5000	2	0	52	54
$\sum$	2	0	98	100
$\langle P\rangle$ in %	0.0	0.0		

(14) $\ddot{u}(x,t) = u''(x,t)$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	1	29	0	30
<5000	0	5	65	70
$\sum$	1	34	65	100
$\langle P\rangle$ in %	4.3	85.7		

(15) $\wp(z;\Lambda)$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	2	0	24	26
<5000	1	0	74	75
$\sum$	3	0	98	101
$\langle P\rangle$ in %	55.6	55.6		

(16) $\wp(z;\omega_1,\omega_2)$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	11	1	15	27
<5000	8	1	66	75
$\sum$	19	2	81	102
$\langle P\rangle$ in %	48.1	45.8		

(18) $O(n \log n)$

	++	+	o	$\sum$
>2k	25	1	2	28
5000-2k	0	13	7	20
<5000	19	18	19	56
$\sum$	44	32	28	104
$\langle P\rangle$ in %	54.6	49.2		

(19) $Rf(L) = \int_L f(\mathbf{x})\,|d\mathbf{x}|$

	++	+	o	$\sum$
>2k	0	4	14	18
5000-2k	0	0	0	0
<5000	0	20	62	82
$\sum$	0	24	76	100
$\langle P\rangle$ in %	-	11.0		

(20) $|G:H| = \frac{|G|}{|H|}$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	4	5	9	18
<5000	28	22	32	82
$\sum$	32	27	41	100
$\langle P\rangle$ in %	4.8	11.4		

(21) $H^n(X) = Z^n(X)/B^n(X)$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	6	0	0	6
<5000	21	12	61	94
$\sum$	27	12	61	100
$\langle P\rangle$ in %	22.2	15.4		

(22) $A_n = \frac{1}{\pi}\int_{-\pi}^{\pi} F(x)\cos(nx)dx$

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	13	0	13
<5000	0	59	29	88
$\sum$	0	72	29	101
$\langle P\rangle$ in %	-	18.1		

$\langle\langle P_{++}\rangle\rangle = 31.5\%$, $\mathrm{MAP}_{+} = 28.5\%$

Table 3: Statistics for NTCIR10-FT

(1) $\wp$ Points derivative vanishes

	++	+	o	$\sum$
>2k	28	2	26	56
5000-2k	4	0	0	4
<5000	18	2	20	40
$\sum$	50	4	46	100
$\langle P\rangle$ in %	39.9	39.4		

(2) $\int_b^a f^2(x)dx$ NOT(Parseval)

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	1	12	45	58
<5000	1	14	27	42
$\sum$	2	26	72	100
$\langle P\rangle$ in %	0.9	16.5		

(4) $\prod_{N=1}^{\infty}(1+Z/N)$ diverges diverge

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	13	12	25
<5000	0	27	48	75
$\sum$	0	40	60	100
$\langle P\rangle$ in %	-	22.7		

(5) $\sum \frac{n!x^n}{n^n}$ radius of convergence

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	18	30	48
<5000	0	38	14	52
$\sum$	0	56	44	100
$\langle P\rangle$ in %	-	18.8		

(6) $\sum_{n=1}^{\infty} \frac{\sin(n)}{n}$ infinite series conditionally convergent

	++	+	o	$\sum$
>2k	0	2	2	4
5000-2k	0	16	26	42
<5000	0	21	33	54
$\sum$	0	39	61	100
$\langle P\rangle$ in %	-	14.8		

(7) $8x^3+4x^2-4x-1$ root

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	12	36	48
<5000	5	18	29	52
$\sum$	5	30	65	100
$\langle P\rangle$ in %	0.0	6.8		

(8) $y^2 = x^3 + ax + b$ mod modulo

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	5	10	4	19
<5000	17	21	43	81
$\sum$	22	31	47	100
$\langle P\rangle$ in %	6.0	24.4		

(9) p -adic diophantine equation

	++	+	o	$\sum$
>2k	5	7	31	43
5000-2k	0	0	0	0
<5000	16	30	11	57
$\sum$	21	37	42	100
$\langle P\rangle$ in %	2.2	11.0		

(10) $r_k(C_4)$ estimated multicolor Ramsey number

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	2	0	32	34
<5000	8	1	57	66
$\sum$	10	1	89	100
$\langle P\rangle$ in %	20.0	18.2		

(11) $x'(t)+\sum_{j=1}^{N} B_j(t)x(t-\tau_j(t)) = F(t)$ conditions boundedness

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	0	2	2
<5000	0	7	91	98
$\sum$	0	7	93	100
$\langle P\rangle$ in %	-	0.0		

(12) $\frac{\partial u}{\partial t} - \triangle u + \frac{\langle D^2uDu,Du\rangle}{1+|Du|^2} = 0$ uniqueness of solutions

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	3	7	10
<5000	0	17	73	90
$\sum$	0	20	80	100
$\langle P\rangle$ in %	-	6.7		

(13) $x_{k+1} = \frac{A_1}{x_k^{p_1}} + \frac{A_2}{x_{k-1}^{p_2}} + \cdots + \frac{A_n}{x_{k-n+1}^{p_n}}$ stability

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	0	2	20	22
<5000	0	9	69	78
$\sum$	0	11	89	100
$\langle P\rangle$ in %	-	1.6		

(14) $\sum p_n a_n$ convergence

	++	+	o	$\sum$
>2k	1	1	2	4
5000-2k	5	4	63	72
<5000	1	5	18	24
$\sum$	7	10	83	100
$\langle P\rangle$ in %	19.6	19.7		

(15) $dX_t = b(t,X_t)dt + \sigma(t,X_t)dW_t$ solution

	++	+	o	$\sum$
>2k	0	0	0	0
5000-2k	12	4	12	28
<5000	21	22	29	72
$\sum$	33	26	41	100
$\langle P\rangle$ in %	16.2	16.6		

$\langle\langle P_{++}\rangle\rangle = 16.7\%$, $\text{MAP}_+ = 15.5\%$