Math Information Retrieval using a Text Search Engine

by

Dallas Fraser

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2018

Abstract

Combining text and mathematics when searching in a corpus with extensive mathematical notation remains an open problem. Recent results for math information retrieval systems on the math and text retrieval task at NTCIR-12, for example, show room for improvement, even though formula retrieval appears to be fairly successful.

This thesis explores how to adapt the state-of-the-art BM25 text ranking method to work well when searching for math and text together. Symbol layout trees are used to represent math formulas, and features are extracted from the trees, which are then used as search terms for BM25. This thesis explores various features of symbol layout trees and explores their effects on retrieval performance. Based on the results, a set of features are recommended that can be used effectively in a conventional text-based retrieval engine. The feature set is validated using various NTCIR math only benchmarks.

Various proximity measures show math and text are closer in documents deemed relevant than documents deemed non-relevant for NTCIR queries. Therefore it would seem that proximity could improve ranking for math information retrieval systems when searching for both math and text. Nevertheless, two attempts to include proximity when scoring matches were unsuccessful in improving retrieval effectiveness.

Finally, the BM25 ranking of both math and text using the feature set designed for formula retrieval is validated by various NTCIR math and text benchmarks.

Acknowledgements

To my supervisor Dr. Frank Tompa for his patience and guidance during the process, my parents Deborah and Dale Fraser for their support, Dr. Angèle M. Hamel and Dr. Chính T. Hoàng for inspiring me to pursue a Masters. I would like to thank Dr. Mark Smucker and Dr. George Labahn for being the readers of my thesis and their feedback. I would also like to thank Dr. Andrew Kane for his insights and time.

I also thank Japans National Institute of Informatics for allowing me to use the NTCIR document data and test collections for my experiments.

Some material forming the core of this thesis was written in collaboration with Frank Tompa and Andrew Kane and has been accepted to appear as:

Dallas Fraser, Andrew Kane, Frank Wm. Tompa,
Choosing Math Features for BM25 Ranking with Tangent-L,
DocEng 2018,
10 pp.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

An important part of any research is understanding the literature. Finding relevant papers to one's own research is a problem of filtering a large amount of information. This process usually involves searching for keywords to help narrow the list of candidate papers. Sometimes when researchers are unable to find relevant results, they spend excessive time on problems that were already answered. This problem is referred to as rediscovery and results in wasted time and resources. Michiel Hazewinkel outlined this problem with respect to mathematics [19] and gives an example in integrable system theory where work was invested only to discover the question was settled in 1928. Ironically, Hazewinkel notes "that for moderately difficult problems it is often less effort to solve them again than to try to find them in the published literature" but then in a footnote admits he cannot find the source for such a quote.

Most search interfaces such as those for Google Scholar or arXiv are limited to text. However, arXiv has a large collection of papers that contain a lot of mathematics. The lack of ability to search with mathematics can lead to researchers being unable to find relevant papers. Researchers could improve their queries if they were able to query using mathematics and potentially reduce occurrences of rediscovery.

1.2 Problem

Language is used to express ideas and allows humans to communicate with each other. Language is denoted by symbols that form words and those words are structured to form sentences. Similarly, mathematics uses symbols that are structured to form mathematical expressions.

Traditional text retrieval systems do not focus on mathematics as searching units and have limited support for mathematics. Mathematical Information Retrieval (MathIR) focuses on searching and retrieving documents that include mathematics. This thesis focuses on MathIR, its current challenges and how to apply traditional information retrieval methods to mathematics.

The first challenge is defining what is a relevant match to a math query. For a presentation-based approach relevancy depends on equations being structurally similar. For example, $a^2 + \pi^2$ is structurally similar to $x^2 + y^2$ and would be deemed a relevant match using a presentation-based approach. For a meaning-based approach, relevancy depends on whether two equations have the same mathematical meaning. For example, π and $\frac{C}{d}$ would be relevant in a meaning-based approach but not for a presentation-based approach.

The second challenge is that mathematics is polysemic: symbols' meanings are usually derived from the context of the document. A symbol such as π can be a mathematical constant or it can be used to denote the adjustment factor $\pi(Q, D)$, as seen in the paper by Tao and Zhai [45].

The third challenge is how one represents mathematics to capture information given by the structure contained in an expression. Formulas containing all the same symbols can be quite different depending on where those symbols appear. For example, $x^2 + y^3 + 4$ is quite different from $x^4 + 2^{y+3}$ even though all the symbols are the same. The order in which symbols appear and the relationship between symbols are important pieces of information contained in a math expression.

MathIR systems have been able to deal with the above challenges with some level of success, using a wide range of approaches, which are discussed in Section 2.4. However, state-of-the-art MathIR systems recently experienced poor results for the NTCIR-12 MathIR arXiv Main task, which used both keywords and math expressions in its search queries. These results suggest that improvements could be made when ranking both text and math. Most MathIR systems adopt separate approaches for mathematic expressions and text, only to combine each approach using some weighting to create a final score. It

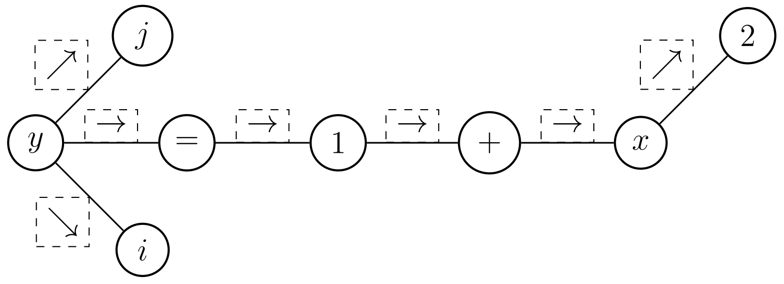

Figure 1.1: Symbol Layout Tree for $y_i^j = 1 + x^2$.

seems most could benefit from a more holistic approach. This thesis focuses on how to rank matches to queries that include both math expressions and text using a single approach.

1.3 Proposed Solution

This thesis examines using pairs of symbols in a math expression, along with their spatial relationships, as tokens that are treated as if they were words. The proposed solution is to take a mathematical expression, convert it into a Symbol Layout Tree (SLT) [41], and capture features from the SLT that are then used for searching. An example of a mathematical expression and its corresponding SLT is shown in Figure 1.1. Features of the SLT are then encoded into tokens that form search units similar to words. We follow an approach similar to the existing Tangent-3 [49] MathIR system that uses pairs of symbols and their relative position, but the proposed solution expands the features of SLT used as search units. Finally, traditional text information retrieval methods are applied to both math tokens and text.

Tangent-3 creates separate indexes for math tokens and text. The proposed solution, called Tangent-L, uses one combined index for both math tokens and text. Tangent-L uses the Lucene framework for indexing and searching in its implementation and scores queries using a traditional text-based IR ranking method called BM25. In this thesis, variations of BM25 that include proximity are explored as well.

1.4 Contributions

This thesis expands on the previous work on Tangent [49]. The result of this work has produced Tangent-L and provides application developers with an example of how to support mathematics retrieval. The development of this system provides the following contributions:

- How to index mathematical tokens
- How to deal with mathematical wild cards
- What features of an SLT to use for mathematical tokens

Tangent-L performance results are comparable to state-of-the-art MathIR systems. The source code of Tangent-L is publicly available.[1]

1.5 Outline

Chapter 2 provides a background on traditional information retrieval, Lucene, current MathIR systems, and the development of Tangent. This chapter highlights the differences between Tangent-L and previous versions of Tangent. Chapter 2 also gives a background on how experiments can be designed to measure the importance of proximity. Additionally, this chapter outlines the various sets of test data used to evaluate Tangent-L.

Chapter 3 provides the details of how Tangent-L works and its implementation in Lucene. It also outlines some of the design decisions and how wild cards can be handled in Lucene. Chapter 4 describes a variety of features derivable from math expressions and the experiments used to test the value of such features for ranking math combined with text. Chapter 5 examines various ways to rank returned documents. An experiment on the value of proximity is discussed, along with various ranking functions that try to leverage proximity. Chapter 6 closes with a discussion about what conclusions can be drawn and what future work should be pursued.

[1]https://github.com/fras2560/Tangent-L

Chapter 2

Background

2.1 Traditional Text Information Retrieval

Traditional text information retrieval (IR) focuses on presenting relevant information from a collection of information based upon a query.

2.1.1 Introduction

Text IR is usually broken into two major functions. The first is the indexing process, which entails building the data structures that enables one to search through the collection. The major components of the indexing process are seen in Figure 2.1. Text acquisition involves acquiring the collection of documents and converting them to a standard format. Text transformation involves parsing the document into a stream of tokens and then applying a set of filters to each token, such as stemming and stopping. Index creation takes the stream of transformed tokens to create the index or data structures that enable searching. Index creation must be efficient with respect to space while still allowing for efficient querying.

The second major function is the querying process, which entails taking a user's query and transforming it to terms that are used to query the index. The major components of the querying process are seen in Figure 2.2. User interaction provides an interface between the person and the IR system. It allows the user to enter a query that then is transformed into query terms. A ranking component compares the query terms to the index terms and generates a ranked list of documents. The ranking is based upon a retrieval model such as the standard boolean model that supports conjunction, disjunction, and negation. The

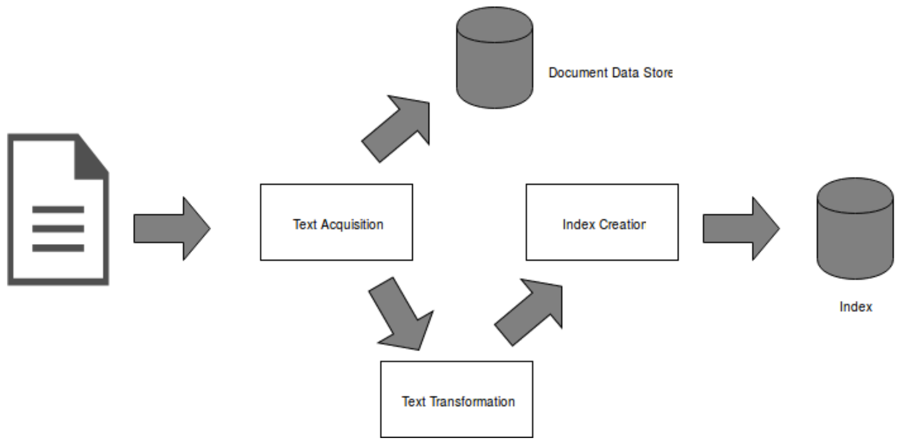

Figure 2.1: The indexing process [11]

evaluation component inspects query logs and updates the ranking component to improve its effectiveness and efficiency.

2.1.2 Performance Measurements

Information retrieval uses heuristics for its ranking and retrieval models. The "right" answer is not completely defined for information retrieval, but rather some results are deemed to be more acceptable than others. Information retrieval performance is usually evaluated by benchmarks that include a collection of documents, a set of queries, and a set of judgments for each query. This allows researchers and developers to compare their systems directly and not require user studies each time one wants to evaluate a system. There are two underlying ways to measure a system: recall and precision.

Recall is the fraction of relevant documents returned:

$$\text{recall} = \frac{\text{number of relevant items retrieved}}{\text{number of relevant items in collection}}$$

Precision is the fraction of documents returned that are relevant:

$$\text{precision} = \frac{\text{number of relevant items retrieved}}{\text{total numbers of items retrieved}}$$

In general recall is a measure of how well a system presents all relevant documents, whereas precision is a measure of how well a system focusses on relevant documents. One

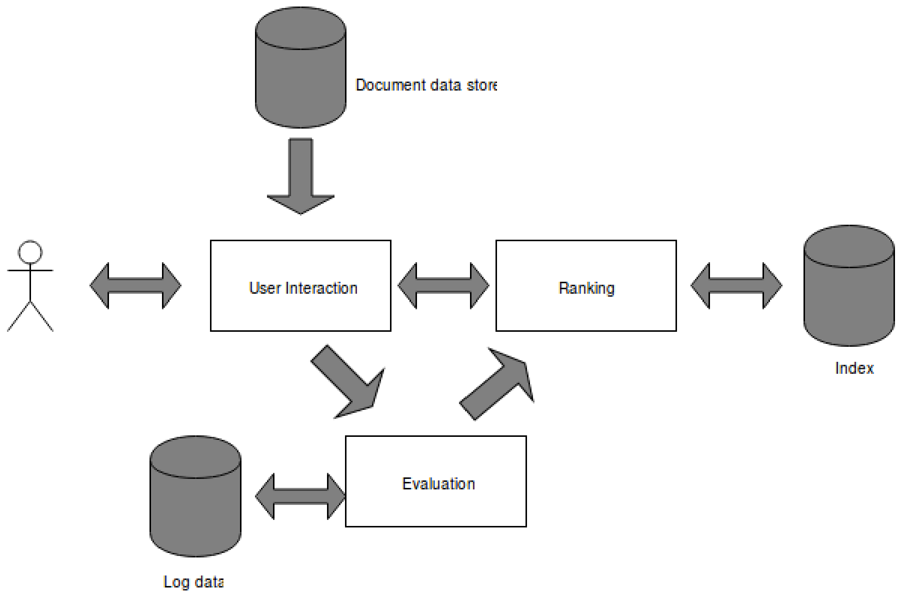

Figure 2.2: The querying process [11]

can increase recall by simpling returning more documents but this would result in a lower precision. In general, as recall increases, precision tends to decrease. Alternative measures (as summarized below) have been created to incorporate both recall and precision.

Precision and recall are both evaluated using the whole list of matched documents returned by the system and do not take the list's order into consideration. One way to incorporate order is to calculate precision based on only the first n documents retrieved, denoted by $P@n$. An alternative way to consider order is to compute average precision (AP) by:

$$AP = \frac{\sum_{k=1}^{n} \Big(P(k) \cdot rel(k)\Big)}{\text{number of relevant documents}}$$

where k is the rank in the sequence of retrieved documents; n is the number of retrieved documents; P(k) is the precision at cut-off k; $rel(k)$ is 1 if a document at rank k is relevant and 0 otherwise. AP is computed on a per query basis and the mean average precision(MAP) is then calculated for all queries by the following:

$$\text{MAP} = \frac{\sum_{q=1}^{Q} AP(q)}{\text{Q}}$$

MAP is a commonly used measurement in information retrieval.

Returned documents may not have a judgment for a query since the set of judgments available is often a subset of the collection. This shortfall becomes more problematic as the size of the collection increases. The MAP score will decrease as the size of not judged documents grows and a measurement called bpref was created to deal with this problem [6]. For a query with R documents judged to be relevant, where r is a document judged to be relevant, n is a document judged to be irrelevant, and N is the number of documents judged to be irrelevant, bpref is calculated as:

$$\text{bpref} = \frac{1}{R}\sum_{r} 1 - \frac{|n \text{ ranked higher than } r|}{\min(R, N)}$$

One additional measurement that is used is *mean reciprocal rank* (MRR), calculated as:

$$\text{MRR} = \frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

where $rank_i$ is the position of the first relevant document.

The above are so commonly used that the National Institute of Standards and Technology (NIST) has produced an evaluation tool, called trec_eval[1], which is used to calculate all measurements in this thesis. The trec_eval output for all experimental runs reported in this thesis are available.[2]

2.1.3 Ranking

One of the most well known ranking functions for IR is term frequency-inverse document frequency (tf-idf). The first idea of tf-idf is that the more frequent a term appears in a document the more likely it is to be relevant [21]. The second idea is that the rarer a term is in the collection of documents, the more weight that term should have [28]. The simplest formulation of tf-idf for a given collection of documents D, a query q, and a document $d \in D$ is given by

$$\text{TFIDF}(q, d) = \sum_{w \in q} \frac{tf_w}{|d|} \log\left(\frac{|D|}{|D_w|}\right)$$

where tf_w is the number of occurrences of term w in document d; $|d|$ is the total number of terms in document d; $|D_w|$ is the number of documents in D containing term w.

[1] https://github.com/usnistgov/trec_eval
[2] https://github.com/fras2560/Tangent-L_Runs

Okapi BM25 is a state-of-the-art text ranking function that is the variant of tf-idf used by many search engines [34]. Given a collection of documents D containing $|D|$ documents and a query q consisting of a set of query terms, the score for a document $d \in D$ is given by

$$\text{BM25}(q, d) = \sum_{w \in q} \left(\frac{(k+1) tf_w}{K + tf_w} \right) \log \left(\frac{|D| - |D_w| + 0.5}{|D_w| + 0.5} \right)$$

where $K = k\left(1.0 - b + b\frac{|d|}{avdl}\right)$; k and b are constants (following common practice, chosen to be 1.2 and 0.75 respectively); tf_w is the number of occurrences of term w in document d; $|d|$ is the total number of terms in document d; $avdl$ is the average document length; and $|D_w|$ is the number of documents in D containing term w.

A variation of BM25, called BM25+, has been defined to reduce the penalty that BM25 imposes on long documents [29].

$$\text{BM25}^{+}(q, d) = \sum_{w \in q} \left(\frac{(k+1) tf_w}{K + tf_w} + \delta \right) \log \left(\frac{|D| + 1}{|D_w|} \right)$$

where δ is a constant (following common practice, chosen to be 1.0).

2.1.4 Proximity

Researchers have investigated other information that could be used when ranking documents besides term frequency and inverse document frequency. One of the common information measures that can also be considered is proximity.

Proximity is how close the matched terms appear in a document. One measure of proximity, span, is the smallest distance to cover all matched terms [9]. Various other measurements have been developed, and research has been conducted to evaluate the effectiveness with respect to document ranking for the following measures when using text-based search [45]:

- **Span** [18] is defined as the length of the shortest document segment that covers *all* query term occurrences in a document, including repeated occurrences.
- **MinCover** [45], minimum coverage, is defined as the length of the shortest document segment that covers each query term at least once in a document.

- **MinDist** [45], the minimum pair distance, is defined as the smallest distance value of all pairs of unique matched query terms.
- **AveDist** [45], average pair distance, is defined as the average distance value of all pairs of unique matched query terms
- **MaxDist** [45]. maximum pair distance, is defined as the largest value of all pairs of unique matched query terms.

These proximity measures were used against various datasets to quantify how well they distinguish between relevant and non-relevant documents. Span and MinCover had negative results, the non-relevant documents having smaller measurements than the relevant documents. The reason for the negative results is that documents with more matched terms typically have longer spans. This leads to normalizing both Span and MinCover. $Span_{norm}$ divides by the total number of occurrences of query terms in the span segment and $MinCover_{norm}$ divides by the number of unique terms. These normalized measures lead to better results but for some datasets are still negative.

MinDist was the only measure of the five that is always positive: relevant documents have a smaller minimum pair distance than non-relevant documents. In Section 5.2, the same experiment of measuring various proximity definitions for both relevant and non-relevant documents is performed to check whether proximity might be beneficial when ranking both math and text.

These various proximity measures can be used for ranking using a simple adjustment to BM25. For a query q, a document d, and a proximity measurement function δ, the following adjustment factor has been defined [45]:

$$\pi(\mathrm{q}, \mathrm{d}) = \log(\alpha + \exp(-\delta(\mathrm{q}, \mathrm{d})))$$

and this can be added to BM25 to produce the ranking function

$$\text{BM25Dist}(q, d) = \text{BM25}(q, d) + \pi(q, d)$$

Another variation of BM25, BM25TP [33], incorporates proximity with BM25. BM25TP calculates the sum of each term pair weight by multiplying it by an *idf* of the term pair. For each pairs of terms (t_i, t_j), BM25TP calculates a weight of the pair:

$$\text{tpi}(t_i, t_j) = \frac{1.0}{d(t_i, t_j)^2}$$

where $d(t_i, t_j)$ is the distance between the two terms.

However each pair of terms may appear more than once so the weight attached to each term pair is evaluated by summing all the *tpi* by:

$$w_d(t_i,t_j) = (k_1+1)\frac{\sum\limits_{\mathrm{occ}(t_i,t_j)} tpi(t_i,t_j)}{K + \sum\limits_{\mathrm{occ}(t_i,t_j)} tpi(t_i,t_j)}$$

where k_1, k and b are constants (following the original paper, chosen to be 1.2, 2.0 and 0.9 respectively);

BM25TP uses a slightly adjusted idf weight and is calculated by the following:

$$qw_i = \frac{qf_w}{k_3 + qf_w} \cdot log\left(\frac{|D_w| - tf_w}{tf_w}\right)$$

where qf_w is the frequency of term w in the query; and k_3 is a constant (set to 1000).

The term pair weight and the idf weight are then summed as follows:

$$\mathrm{TPRSV}(d,q) = \sum_{(t_i,t_j)\in S} w_d(t_i,t_j) \cdot \min(qw_i, qw_j)$$

Finally, this score is then combined with BM25 to produce a final score:

$$\mathrm{BM25TP}(d,q) = \mathrm{BM25}(d,q) + \mathrm{TPRSV}(d,q)$$

BM25TP and BM25Dist are used in Section 5.2 to explore the effect of proximity on ranking documents for queries that include both math and text. Other proximity-aware ranking functions [30] that are based upon different ranking models have not been explored in this thesis.

2.2 Lucene

Lucene is an open source search library that has Application Programming Interfaces (API) for common IR tasks [5]. Lucene was created in 1997 and donated to the Apache Software Foundation in 2001.

Lucene provides APIs for common tasks in indexing, querying, and searching. Figure 2.3 depicts an overview of Lucene's architecture. Only the relevant parts to Tangent-L are described here, since Lucene is fairly complex.

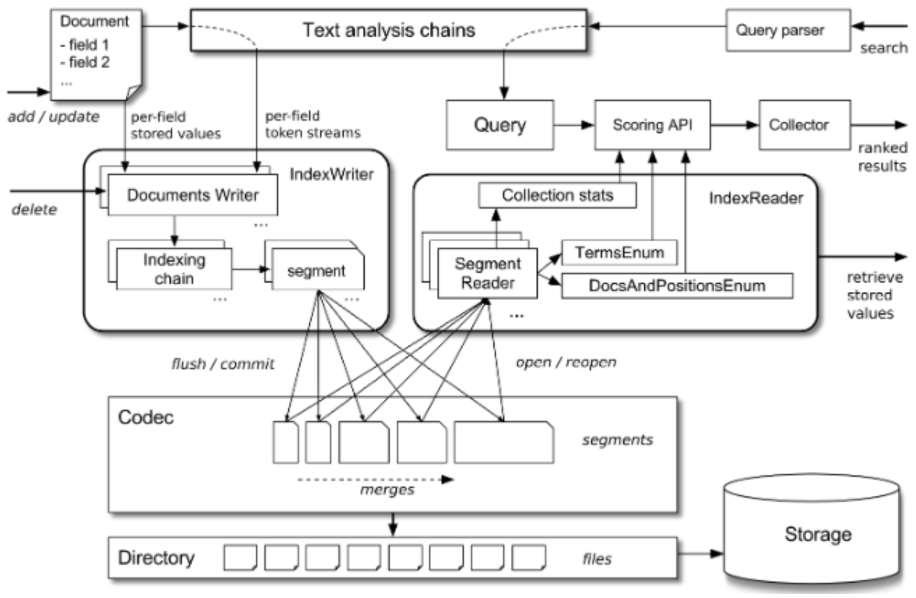

Figure 2.3: Lucene's Architecture [5]

Language analysis (implemented as a text analysis chain) transforms text content into a representation that can be used either for indexing or for querying. Language analysis has three main parts:

- Character Filtering and Normalization
- Tokenization
- Token Filtering

Language analysis in Lucene is applied by tokenizing a stream of characters and then running a series of filters. The tokenizer and chain of filters is referred to as an *Analyzer*.

Lucene includes an *IndexWriter* that is used for creating indexes. After an *Analyzer* processes text from a document, the results are then passed to *IndexWriter* and added to an index.

Lucene also includes an *IndexSearcher* that executes queries against an index. It uses an *Analyzer* to process the query input text and produces a *Query*. A *Query* can then be executed against the index to return a ranked list of matched documents.

Lucene was used to implement Tangent-L. It was selected since it is a common library used by IR researchers and is the basis of many commonly used systems such as Elasticsearch and Apache Solr. Importantly, it allows one to experiment with alternative ranking functions by making small changes to the code.

2.3 NTCIR MathIR

The development of a new system requires one to be able to evaluate it in an objective way. For IR, this requires a set of queries on a collection of documents along with a list of judged documents. This entails creating a collection of documents, queries, and user studies to evaluate the returned documents for each query. This approach has been used by some researchers, such as Zhang and Youssef [51], but has a few downsides. First, it requires a significant amount of time to create a new collection of documents. Second, the creation of the queries may not reflect users' actual queries, and thus the judgments may not reflect the documents' relevance to actual user needs. Third, the results of various systems are not comparable.

For many IR tasks, the annual Text REtrieval Conference (TREC) has created a commonly accepted dataset together with queries and collections of judged documents. TREC then publishes the dataset, queries, and judged documents, packaged together as a task, so new systems can compare their performance against previous results. TREC tasks relieve researchers from spending time creating user studies and allow them to evaluate their systems with less effort.

The lack of a TREC task for MathIR systems prevented researchers from comparing and improving their results. However, at the 10th NII Testbeds and Community for Information access Research (NTCIR), a MathIR focused task was created: NTCIR-MathIR [1]. Since then, two other NTCIR-MathIR tracks have included MathIR-related tasks [2] [48]. NTCIR-MathIR tasks are used to evaluate the results in this thesis.

2.3.1 Formula Representation

When mathematical expressions are included in a document, they are typically linearized and then encoded as text, using a language such as LaTeX. One benefit of this representation is that math formulas can be tokenized and then indexed along with the text in a conventional search engine.

Name	Size(GB)	Tasks
NTCIR-10 Math dataset	63	NTCIR-10 Math Pilot Task
NTCIR-11 arXiv dataset	174	NTCIR-11 Math-2
		NTCIR-12 arXiv Main Task
		NTCIR-12 arXiv Simto Task
NTCIR-11 Wikipedia dataset	2.5	NTCIR-11 Wikipedia Open Subtask
NTCIR-12 Wikipedia dataset	5.2	NTCIR-12 MathWiki Task
		NTCIR-12 MathWikiFormula Task

Table 2.1: Overview of the various NTCIR-MathIR datasets

An alternative text encoding is MathML, a W3C recommendation [7] proposed for exchanging mathematics between software tools. MathML is supported by the Firefox browser, and the MathML Association is working to improve support in browsers. Because most scientific authors write mathematics in LaTeX, MathML is often produced by converting LaTeX into MathML using tools such as MathJax [8] or LaTeXML [17].

Even though MathML is a text representation, it reflects the structure of a mathematical formula as a tree: the structure can be encoded in Presentation MathML, representing how symbols are placed relative to each other on a printed page; alternatively the structure can be encoded in Content MathML, representing how mathematical expressions are built from operators and subexpressions. Some work has been done on whether Content or Presentation MathML is better for search with the results indicating that search could benefit from semantic enrichments [32]. NTCIR-MathIR provides both Content and Presentation MathML for its formula representation.

2.3.2 MathIR Datasets

The documents used for the various NTCIR-MathIR tasks come from two main sources: arXiv and Wikipedia. The documents are in an HTML or XHTML format and the embedded math formulas are all encoded in three different representations: Content MathML, Presentation MathML, and original LaTeX source. Four main datasets have been created as summarized in Table 2.1.

The NTCIR-10 Math dataset is a collection of arXiv documents obtained from the KWARC research group [44]. KWARC used LaTeXML[3] [43] to convert LaTeX sources into MathML and converted over 105,120 scientific articles from mathematics, physics, and

[3]https://github.com/brucemiller/LaTeXML/

computer science. LaTeXML was able to convert 60% of all the documents in the collection successfully but 30% had unsupported macros or conversion errors and the remaining 10% could not be converted at all.

After creating the pilot task, LaTeXML had improved enough that the organizers of NTCIR-11 used it on the same documents as the NTCIR-10 Math dataset. Because the assessors of the NTCIR-10 Math Pilot Task had difficulty in assessing 15-page scientific articles, the articles were segmented into a total of 8,301,578 paragraph-sized search units. In addition, the NTCIR-11 Wikipedia dataset was created for the NTCIR-11 Wikipedia Open Subtask. This second dataset has a total of 30,000 articles with a total size of 2.5 GB.

The NTCIR-11 arXiv dataset was re-used at NTCIR-12 for two tasks. In addition, the NTCIR-12 Wikipedia dataset was created with a total of 319,689 articles. This dataset, unlike the NTCIR-11 arXiv dataset, includes full articles. Only 10% of the dataset contain an explicit <*math*> tag with the other articles sampled from across Wikipedia. The NTCIR-12 Wikipedia dataset used the same process as the datasets before it, but with the additional step to convert mediawiki templates for mathematics in to $\LaTeX$, and thus into MathML as well.

2.3.3 MathIR Queries and Judgements

NTCIR-MathIR tasks are used as benchmarks for this thesis. The NTCIR-MathIR tasks include queries along with relevance judgments for each query. The queries consist of a name, a list of keywords and a list of formulas. The formulas are embedded in the same way as formulas are embedded in a document. The relevance judgments are in a text file with the query name, document name, and a score. For the NTCIR-11 Wikipedia Open Subtask the document name includes the formula id.

Formulas in a query may include a wild card variable. The wild card variable can match any arbitrary symbol or subexpression on a candidate formulae. The wild cards symbols are denoted by ?v for this thesis.

Four of the NTCIR-MathIR tasks, summarized in Table 2.2, are used for training and testing. The NTCIR-11 Wikipedia Open Subtask is used to select various math features, and the NTCIR-12 MathWikiFormula Task is used to test those selected features. The NTCIR-12 MathWiki Task is selected for exploring proximity measures and different ranking functions due to its smaller size. Finally, the NTCIR arXiv Main Task is used as a final test for Tangent-L due to its large size and being able to compare results to other MathIR systems.

Task	# of Queries	# of Queries with a Wild card	Includes Text Keywords
NTCIR-11 Wikipedia Open Subtask	100	43	No
NTCIR-12 MathWiki Task	30	10	Yes
NTCIR-12 arXiv Main Task	29	25	Yes
NTCIR-12 MathWikiFormula Task	40	20	No

Table 2.2: Overview of the various NTCIR-MathIR tasks

The competing systems at NTCIR-12 had their runs evaluated using $P@K$ for $K \in 5, 10, 15, 20$ for the MathWiki, MathWikiFormula and the arXiv Main Tasks [48], and the original runs are available from NTCIR.[4] It should be noted that some of the original submissions had incorrectly assumed that trec_eval evaluated by rank and not score. This resulted in some of the systems have substantially lower precision. Eventually, two different tables were used: one using ranking by score and the other by using the provided ranking. For simplicity, all comparisons of Tangent-L to previous results are evaluated by trec_eval using the provided ranks.

The NTCIR-11 Wikipedia Open Subtask was not evaluated using trec_eval but rather with a different approach [38]. The task has 100 queries, where each query has exactly one target document and in that document one target formula id that is deemed to be relevant. Systems are required to return a list of (*documentId*, *formulaId*) pairs for the top 10,000 hits. Two types of evaluations are performed: *document-centric* and *formula-centric*. *Document-centric* results are computed using a list of document identifiers in order of appearance after projecting on *documentId* and removing duplicates. *Formula-centric* results are computed using the ranked list with both document and formula identifiers. Using those ranked lists, recall is measured by mean Recall@10000, and precision is measured by MRR, with zero used if the target is not found within the 10,000 results returned for a query.

2.4 Math Systems

This section provides a brief overview of the state-of-the-art MathIR systems. Zanibbi et al. [49] describe three main approaches used for MathIR: *text-based*, *tree-based*, and *spectral*. As described in Section 2.3.1, the text-based approach involves taking a math expression

[4]http://research.nii.ac.jp/ntcir/permission/ntcir-12/perm-en-MathIR.html

tree and linearizing it, using the resulting sequence of characters for indexing and searching. The process of linearizing the math expression tree may include normalization with the following main steps:

- canonical orderings - ordering operands of commutative operations (e.g., $x + y$ and $y + x$)
- enumerating variables - substituting variables for unified symbols (e.g., x replaced by var_1)
- symbol substitution - replacing symbols with mathematical type (e.g., 2 replaced by $const$)

These steps aim to allow MathIR systems to match formulas to similar subexpressions. Sometimes multiple text representations of the same formula are used with different weights assigned to each representation.

The tree-based approach usually involves storing a math expression as a tree. Subtrees are indexed as well to allow for partial matching. The trees are then used when searching by exact matching to other trees or using tree-edit distance [22].

The spectral approach uses paths in a tree rather than the whole tree. The paths are then used as retrieval units. The number of paths in a tree can be large so that limiting paths to fall within a certain window size is sometimes appropriate.

A total of five systems participated in the NTCIR-12 Main ArXiv Task [48]. Table 2.3 outlines the various system configurations used.

2.4.1 SMSG5

SMSG5 [46] is a math search engine developed by a group from Samsung R&D Institute India. The system uses Elasticsearch and a text-based approach, and three fields are used to store different levels of information. The first field holds exact MathML expressions with unnecessary information, such as formula id, being removed. The second field holds MathML expressions that are generalized by replacing symbols with "*". The third field stores keywords with stopwords removed. Two rules were developed to map a mathematical equation with its definition that then are indexed into their own field. A query on the normalized fields is then expanded to include the mathematical definition of mathematical equations.

System	*Presentation MathML*	*Content MathML*	*Tree Structure*	*Search Platform*	General Approach
MCAT	Yes	Yes	Yes	Solr	Tree-based
MIaS*	Yes/No	Yes/No	Yes	Lucene	Text-based
SMSG5	Yes	No	No	Elasticsearch	Text-based
Tangent-3	Yes	No	Yes	Lucene & Custom+	Spectral-based
WikiMIR	Yes	No	Yes	Custom	Tree-based

* Had multiple configurations with some using both Presentation MathML and Content MathML and some using just one MathML form.

+ Lucene used for text and custom search engine used for math

Table 2.3: Participant System Configurations for NTCIR-12 arXiv Main Task [48]

Each of the above fields is queried in Elasticsearch and each one produces a ranked list of size c. These ranked lists are combined using Borda Count, an algorithm that is used to combine preferences of many experts [4]. Each ranked list is a voter where its first ranked documents gets c votes, its second ranked documents gets $c-1$ votes and its nth ranked documents gets $c-n$ votes. Borda Count outputs a ranked list by sorting the documents by the number of votes received from all the voters. The Elasticsearch ranked list, along with a LDA model ranked list, a Doc2Vec model ranked list and pattern based rules ranked list are combined using Borda Count to produced a final.

Just using Elasticsearch ranked list produces the best results for partially relevant results. Combining the ranked lists of Elasticsearch, Doc2Vec model and pattern-based rules produces slightly better results for relevant results. Nevertheless, SMSG5 had the worst results for relevant documents on the NTCIR-12 arXiv Main Task.

2.4.2 MIRMU

Math Indexer and Searcher (MIaS) [40] is a system developed by the Masaryk University Math Information Retrieval (MIRMU) team. MIaS is a text-based system that uses Presentation MathML for each formula, encoding it in a compact string representation. MIaS generalizes math expressions structurally or at the operator level. MIaS results for the NTCIR-11 tasks led to a structural generalization strategy to enable substitution of query formula structures for different structures. For example [35], $a^2 + \frac{\sqrt{b}}{c}$ should unify with $a^2 + \frac{x}{y}$. The matching is done by generalizing at different levels and producing multiple strings that could match. The formula $a^2 + \frac{\sqrt{b}}{c}$ would have the following generalization

L	Original	Generalized
1	$(x+y)\times\frac{a}{b}$	$(*+*)\times\frac{*}{*}$
2	$(x+y)$	$(*)$
2	$\frac{a}{b}$	$\frac{*}{*}$
3	$x+y$	$*+*$

Table 2.4: WikiMirs extracted terms for $(x+y)\times\frac{a}{b}$ [27]

levels:

1. $a^2 + \frac{\sqrt{*}}{c}$
2. $*^* + \frac{*}{*}$
3. $* + *$

MIaS generalizes at the operator level by tree ordering the formula, replacing variables with a unified symbol, replacing constants with a unified symbol and finally assigning weights to math expressions and all of their logical subparts. MIaS uses Lucene for indexing and querying. Structural generalization does increase recall, but at the cost of decreasing precision and the NTCIR-12 MathIR results are disappointing when compared to the NTCIR-11 results [35].

2.4.3 ICST

Researchers at the Institute of Computer Science & Technology (ICST) of Peking University [46] developed a system called WikiMirs, which is a hybrid MathIR using a tree-based approach for formulas. WikiMirs converts Presentation MathML into an operator tree (OT) using the semantic enrichment technique [27]. The trees are then traversed recursively at various levels to extract both original and generalized subexpressions, as illustrated in Table 2.4. The subexpressions are linearized into MathML, and the resulting formulas are indexed.

WikiMirs uses a ranking model that is trained using the RankBoost algorithm, which works by having a collection of weak learners that are used to produce a score for a learner-specific feature [15]. Each weak learner has a weight that is adjusted while training the model. A final score is produced by summing the scores from each weak learner multiplied times by their weights. WikiMirs uses 12 different features for its weak learners with three

```
<math>
  <mrow>
    <msubsup>
      <mo>&Sigma ;</mo>
      <mrow>
        <mi>i</mi>
        <mo>=</mo>
        <mn>0</mn>
      </mrow>
      <mi>n</mi>
    </msubsup>
  </mrow>
  <msub>
    <mi>a</mi>
    <mi>i</mi>
  </msub>
</math>
```

Figure 2.4: Presentation MathML encoding for $\sum_{i=0}^{n} a_i$ [25]

categories: formula-based, relevance-based, and document-based. Formula-based features include features such as the number of layers in the OT, formula frequency, the number of variables, and the number of constants.

WikiMir is trained on a data set that has 224 queries each with 100 relevance judgments assigned by 12 evaluators at either the undergraduate or postgraduate level. The ranking model produces a ranked list, but the ranked list is then re-ranked using regular expression matching so that documents with many small sub-terms matching could be considered relevant.

2.4.4 MCAT

MCAT is a MathIR system that uses Presentation MathML to extract path features of a tree [24]. MCAT stores vertical paths from the root of the MathML expression to each element in the formula. MCAT normalizes MathML using SnuggleTex[5] and removes tags such as *mrow* and *mfenced*. Vertical paths are encoded in two different ways: ordered paths,

[5] https://www2.ph.ed.ac.uk/snuggletex/documentation/overview-and-features.html

and unordered paths. Using Figure 2.4 as an example, $2\#1\#mi\#a$ is an ordered path from the root to the element a where the # is used to delimit different levels, the 2 indicates that <msub> is the second child of the root and 1 indicates that the <mi> is the first child of <msub>. The unordered path, for the same example, would be $\#\#mi\#a$. Each subexpression of a math expression has its vertical paths stored to allow subexpressions to match, with an example being $2\#mi\#i$. In addition to vertical paths, MCAT stores a collection of elements, called sisters, that share a common parent element. For example, $mi\#a$ and $mi\#i$ would both be stored. Finally, MCAT also uses hash-based encoding as signatures that capture several tree properties . Three different signatures are combined to produce a single string representation of a math expression.

For each math formula, MCAT associates a set of words to it. Twenty words surrounding a formula are considered the context words. MCAT uses a support vector machine (SVM) model to extract a list of words, called descriptions, to denote the math expression. The SVM is trained using previous NTCIR datasets but has no description words for some formulas. So a math dependency graph is used where a directed edge between *expression-1* to *expression-2* indicates that expression-1 contains expression-2. This dependency graph is used so that math expressions can utilize descriptions from its children's descriptions. Nouns that appear in the same sentence of the math expression are stored as well.

All keywords, math encodings, and math textual information are stored in their own fields. The fields are queried and each field's score is normalized by

$$\text{finalScore} = \frac{\text{rawScore}}{1 + \text{rawScore}}$$

After each score is normalized, it is summed up to produce a document score.

MCAT also has a unification step that doubles the score of math expressions that can be instantiated from the query. Each math expression is transformed into prefix notation and checked by the unification feature of Prolog.

MCAT [48] was the best performing search engine submitted for NTCIR-12 MathIR tasks except for the MathWiki Task. However, MCAT has large index sizes: the index for the NTCIR-12 arXiv Main Task is 426 GB [24]

2.5 Tangent

2.5.1 Formula Structure Model

Tangent-3 is a two-stage formula retrieval system [49]. First Presentation MathML is transformed into a simpler representation of a SLT (as in Figure 1.1). Nodes represent operators, variables, constants, fractions, matrices, function arguments, and parenthesized expressions, and each node label reflects its type and its value:

- numbers (N!n)
- variables names (V!v)
- text fragments (T!t)
- fractions (F!)
- radicals (R!)
- matrices, tabular structures and parenthesized expressions (M!$frxc$)
- wildcard symbols (*w)
- mathematical operators

Each directed edge has a label that reflects the spatial relationship from the source node to the target node:

1. next ($\rightarrow$)
2. within ($\boxdot$)
3. element ($\in$)
4. above ($\nearrow$)
5. below ($\searrow$)
6. pre-above ($\nwarrow$)
7. pre-below ($\swarrow$)

S_1	S_2	R
$V!y$	$V!j$	$\nearrow$
$V!y$	$V!i$	$\searrow$
$V!y$	$=$	$\rightarrow$
$=$	$N!1$	$\rightarrow$
$N!1$	$+$	$\rightarrow$
$=$	$V!x$	$\rightarrow$
$V!x$	$N!2$	$\nearrow$
$V!y$	$N!1$	$\rightarrow\rightarrow$
$=$	$+$	$\rightarrow\rightarrow$
$N!1$	$V!x$	$\rightarrow\rightarrow$
$+$	$N!2$	$\rightarrow\nearrow$
$V!y$	$+$	$\rightarrow\rightarrow\rightarrow$
$=$	$V!x$	$\rightarrow\rightarrow\rightarrow$
$N!1$	$N!2$	$\rightarrow\rightarrow\nearrow$
$V!y$	$V!x$	$\rightarrow\rightarrow\rightarrow\rightarrow$
$=$	$N!2$	$\rightarrow\rightarrow\rightarrow\nearrow$
$V!y$	$N!2$	$\rightarrow\rightarrow\rightarrow\rightarrow\nearrow$

Table 2.5: A list of symbol pairs for $y_i^j = 1 + x^2$

8. over ($\uparrow$)
9. under ($\downarrow$)

Given a SLT representing a math formula, Tangent-3 extracts so-called *symbol pairs*, which are features in the form of triples (s_1, s_2, R), where s_1 and s_2 are two symbols appearing on a path in the SLT and R is the sequence of labels on the directed path between the nodes labelled by those symbols. The symbol pairs corresponding to the SLT in Figure 1.1 are shown in Table 2.5.

Search Strategy

Symbol pairs form the search units used by Tangent-3's core engine, which contains an inverted index mapping triples to formulas within documents. The core engine assigns a score to each formula within each document based on the similarity of a query and formula, as reflected by Dice's coefficient. After the core engine retrieves a large ranked list of formulas, they are re-ranked using Maximum Subtree Similarity (MSS) [49] to select the final top-k to be returned to the user. The MSS metric examines SLTs so as to account for symbol unification and to rank subexpressions based on structural similarity. This evaluation is far more costly than using conventional scoring functions for text retrieval, thus limiting the number of results that can be re-ranked.

Beyond Formula Retrieval

Tangent-3 produced state-of-the-art results for the NTCIR-11 Wikipedia formula retrieval task, which focuses solely on formula retrieval [49]. However, mathematical information retrieval more generally must support queries that include both math expressions and keywords. To this end, Tangent-3 supplements the two-stage engine for retrieving math formula with a Lucene index for the remainder of the text, combining the results from each using a linear weighted combination of their scores [12]. However, the disappointing results for the NTCIR-12 arXiv Main task indicate that separate retrieval models for math and text may be the wrong approach to adopt [48].

Chapter 3

Applying Traditional Text Information Retrieval to Math Features

Tangent-3 produced adequate results for just math formula retrieval, as seen by its performance on the NTCIR-11 Wikipedia benchmark [49]. However, Tangent-3 results for the NTCIR-12 MathIR arXiv Main Task has room for improvement [48]. One difficulty of Tangent-3 was combining the text retrieval system's ranking with the math retrieval system's ranking. This difficulty leads to exploring how to incorporate Tangent's math features into a traditional text retrieval ranking function, with the result being Tangent-L. Tangent-L uses traditional text retrieval approaches for formula retrieval so that it can be integrated with keyword retrieval.

This chapter outlines the general approach of Tangent-L, its implementation in Lucene, and some experiments to test the effectiveness.

3.1 Approach

This section introduces the general approach of Tangent-L and how to extract features from an SLT to form a stream of math tokens that can be indexed by traditional information retrieval systems. Additionally, this section discusses how to handle wild cards and how to rank math features.

The first step of Tangent-L is to convert MathML expressions into an SLT. SLTs are created by parsing Presentation MathML, and Tangent-L uses the same parser as Tangent-3 [49] and uses the same node labels, node types, and spatial relationships.

The second step of Tangent-L is to extract features of the SLT. This can be done by recursively traversing the tree and adding features it encounters to a list. Tangent-L uses symbol pairs as one of its features, but additional features are explored in Chapter 4.

The final step is to create a stream of tokens, where each token represents a single SLT feature. The encoding of a token must meet a certain set of requirements. How to create a stream of tokens is discussed next.

3.1.1 Tokenization of Math Features

The stream of tokens created by Tangent-L needs to be able to be parsed by an IR system parser. The following properties are required of math tokens for them to be properly parsed:

- distinguishable
- unambiguous
- recognizable

Math tokens need to be distinguishable from text tokens. The encoding of math tokens should not allow for a text token to appear to be a math token. For example, encoding the symbol pair $(>, N!0, \rightarrow\rightarrow\rightarrow\rightarrow)$ as gt-nnnn0 makes it indistinguishable after case-folding from GT-Nnnn0, which is a Samsung Galaxy model number. Like a shift character, Tangent-L uses a start and end symbol for each math token to distinguish it from all text tokens.

Math tokens need to be unambiguous with respect to other math tokens. Two math tokens should be equal only when they both encode the same feature. For example, one must distinguish between $x+$ and $+x$. Tangent-L orders the two symbols in a pair by their distance to the root, where the first symbol is closer to the root.

Tokens need to be recognizable and allow for a tokenizer to recognize the start and end of each token. Tangent-L separates tokens by whitespace and no token contains any whitespace.

3.1.2 Wild cards

Wild card symbols are used in queries to denote something that can match any other subexpression. This requires Tangent-L to be able to handle math expressions such as $?v + y$ where $?v$ represents any possible symbol or subexpression. Such an expression produces the symbol pair $(?v, +, \rightarrow)$ that should match symbol pairs such as $(V!x, +, \rightarrow)$ and $(N!3, +, \rightarrow)$. There are two possible approaches to handling wild card symbols.

The first approach is to expand math features at query time to include all possible matches. This approach works by searching the postings lists that correspond to a set of features that match a wild card. For example $(?v, N!5, \rightarrow)$ would map to both $(+, N!5, \rightarrow)$ and $(-, N!5, \rightarrow)$, and therefore those two postings lists (among others) would be included in processing a query that generates the wild card token.. Tangent-3 uses this approach and restricts expansion to include only one wild card.

Tangent-L's approach is to expand math features at indexing time. This approach involves taking all math features and expanding them to include possible wild card matches. For example, a symbol pair such as $(V!x, +, \rightarrow)$ would expand to include $(?v, +, \rightarrow)$ and $(V!x, ?v, \rightarrow)$. All three possibilities are then indexed. To avoid excessive computation and to save index space, symbols pairs are limited to having at most one wild card symbol. Hence $(?v, ?v, \rightarrow)$ is not indexed. Now a symbol pair with a wild card will have its *idf* computed based on the number of other symbol pairs that also match the wild card symbol. This allows one to combine the scores of symbols pairs with symbols pairs that contain a wild card in a intuitive way and without changing the underlying ranking function. A disadvantage, however, is that this approach cannot distinguish a match for ?v from a match for ?w. Thus ensuring that two occurrences of the wild card ?w in an expression match the same symbol in a document cannot be supported.

3.1.3 Ranking Math Features

Formula features may appear more than once within a query. BM25 is normally defined as accepting a set of query terms but can be applied to a bag of query terms by accumulating repeated terms multiple times. However, is this the best way to adopt BM25 for formula retrieval? Two variations of BM25 are explored: BM25Math and BM25+.

Text documents with many instances of a term t are more likely to be relevant than text documents with fewer instances of a term t. However, the question arises whether this holds for formula features. If searching for $x + 1$, it is not obvious that more occurrences of $+1$ make a document more relevant. It seems more likely that documents with the same number of occurrences of a math feature as are presented in the query are more likely to be relevant. A new variation of BM25 called BM25Math can be defined to reflect this observation. It adjusts BM25 so that no document's rank is improved for having extraneous occurrences of a math feature when compared to the number of occurrences of that math feature in the query.

Given a collection of documents D containing $|D|$ documents and a query q consisting of a set of query terms, the score for a document $d \in D$ is calculated as follows:

$$\text{BM25Math}(q,d) = \sum_{w \in q} \left(\frac{(k+1)\mathit{tf}_m}{K + \mathit{tf}_m} \right) \log \left(\frac{|D| - |D_W| + 0.5}{|D_w| + 0.5} \right)$$

where $K = k\left(1.0 - b + b\frac{|d|}{avdl}\right)$; k, and b are constants (following common practice, chosen to be 1.2, and 0.75); $|d|$ is the total number of terms in document d; $|D_w|$ is the number of documents in D containing term w; tf_w is the number of occurrences of term w in document d; qf_w is the number of occurrences of term w in query q; $avdl$ is the average document length and

$$\mathit{tf}_m = \begin{cases} min(\mathit{tf}_w, \mathit{qf}_w), & \text{if math term} \\ \mathit{tf}_w, & \text{otherwise} \end{cases}$$

Formula features will increase the document length and large formulas will have many more features than shorter formulas. This may negatively impact documents with long formulas. BM25+ addresses this issues by introducing a larger lower bound.

3.2 Implementation

This section provides technical details of the implementation of Tangent-L. Full descriptions of how to index, query and search with formula features using Lucene are provided. Tangent-L was developed using Lucene 6.4.0 APIs.[1]

3.2.1 Tokens Encoding

Symbol pairs are encoded with the following form $\#(s_1, s_2, p)\#$ where p is the relative path between symbol nodes s_1 and s_2. Table 3.1 lists the encoded tokens for Figure 1.1.

[1]https://lucene.apache.org/core/6_4_0/index.html

Tokens
$\#(V!y, V!j, a)\#$
$\#(V!y, V!i, b)\#$
$\#(V!y, =, n)\#$
$\#(=, N!1, n)\#$
$\#(N!1, +, n)\#$
$\#(=, V!x, n)\#$
$\#(V!x, N!2, a)\#$
$\#(V!y, N!1, nn)\#$
$\#(=, +, nn)\#$
$\#(N!1, V!x, nn)\#$
$\#(+, N!2, na)\#$
$\#(V!y, +, nnn\#$
$\#(=, V!x, nnn)\#$
$\#(N!1, N!2, nna)\#$
$\#(V!y, V!x, nnnn)\#$
$\#(=, N!2, nnna)\#$
$\#(V!y, N!2, nnnna)\#$

Table 3.1: A list of encoded symbol pairs for $y_i^j = 1 + x^2$

Tangent-L meets the properties described in Section 3.1.1 for its encoding of math tokens and can be parsed by a traditional text IR parser. These math tokens are then treated in the exact same way as words for the rest of the indexing process with the one exception being symbol pairs that contain a wild card.

3.2.2 Converting MathML into tokens

A Python program called *Convert* takes a document and converts all MathML expressions into math features. *Convert* is built on Tangent-3 code that was written in Python. *Convert* strips all the html tags from a file and converts any MathML expression to a SLT. The SLT is then traversed recursively to extract a list of features. This list of features are then encoded into the document in place of the MathML expression. The result of the conversion can be saved to a file, which can then be used when indexing or searching. Currently, *Convert* only supports MathML, but a tool such as LaTexML can be used for converting $\LaTeX$ to MathML.

3.2.3 Indexing

The indexing process begins by giving the *Indexer* a directory of documents and a configuration file. The *Indexer* loops through each document and passes each document to *Convert*. The result of the conversion is read by Java's *Reader* class.

The *Indexer* then passes a *Reader* object to Lucene's *IndexWriter* with a *MathAnalyzer*, which is a custom *Analyzer*. The *MathAnalyzer* uses a custom *Tokenizer*, called *MathTokenizer*, which splits upon a whitespace, hyphen or apostrophe for all text words. *MathTokenizer* recognizes math tokens since they start and end with a #. Upon recognizing #, *MathTokenizer* starts to process it as a math token and does not break upon hyphens or apostrophes but still breaks upon whitespace. All math tokens are encoded in such a way that they do not have any whitespace.

Tokens are then filtered in a chain process that is outlined in Figure 3.1.

- *ClassicFilter*: strips periods from acronyms and *'s* from possessives
- *EnglishPossessiveFilter*: deals with all situations of possessive nouns such as *s'*
- *LowerCaseFilter*: converts tokens to lower case
- *StopFilter*: removes all tokens that are stop words

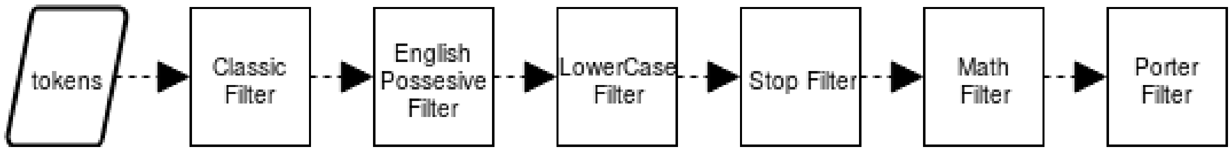

Figure 3.1: MathAnalzyer: A series of filters applied to a list tokens.

- *MathFilter*: expands math features to include wild cards matches and position math features
- *PorterFilter*: filters tokens based upon the Porter stemming algorithm

All the above filters are common to Lucene's *EnglishAnalyzer* except for the addition of the *MathFilter*. *MathFilter* performs two main operations. The first operation positions a math token in the document and is explored in Section 5.2. The second operation deals with wild cards as discussed above in Section 3.1.2.

After indexing a collection of documents, Lucene creates files that are referred to as an index. This Lucene index can then be used for querying the collection of documents.

3.2.4 Searching

To search using Lucene one has to create a *Query* object. Normally, *Query* objects are built using Lucene's *QueryParser*. However, Lucene's *QueryParser* would not handle math tokens correctly. *MathQuery* is used to build a Lucene *Query* object that can be used by Lucene's *IndexSearcher*.

First an XML file is parsed by *ParseQueries* that first converts all MathML expressions appearing in the queries using *Convert* (Figure 3.2). After that each query is used to construct a *MathQuery* object. Additionally, *MathQuery* takes the text from the queries and filters it with the above *MathAnalyzer*. Each query holds information about the query name, keywords and math formulas. A list of *MathQuery* objects is then returned to be used for searching.

Now each *MathQuery* can be used to to construct a Lucene *Query* object. An example of a query appears in Figure 3.3. *MathQuery* constructs a *TermQuery* for each math token or keyword. Then it uses Lucene's *BooleanQuery Builder* to combine each *TermQuery* to form a *BooleanQuery*. This resulting *BooleanQuery* is then passed to *MathScoreQuery*, which extends Lucene's *CustomScoreQuery*. Lucene's *CustomScoreQuery* is used to adjust the

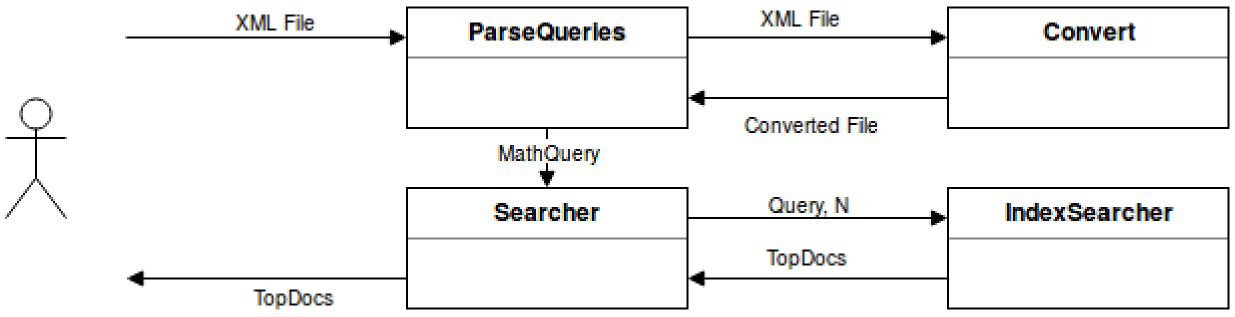

Figure 3.2: Tangent-L: Executing a Search.

score of each document from Lucene. See Chapter 5 for a description of which adjustments were explored. The search is performed by Lucene's *IndexSearcher* on a *MathScoreQuery*. Lucene's *IndexSearcher* allows for each query to be limited to the top N results.

3.2.5 Configuration

Tangent-L has a few options for indexing and searching. These options include which ranking function to use, the positioning of math tokens, the type of math features to use, and whether Lucene should store the positioning information. The effects of each option are discussed in Chapter 4 and 5. A configuration file is created when indexing to record all the options and then are loaded back when searching. A *Configuration* can be changed after loading the index for searching, but invalid changes will raise an *ConvertConfigException*. For example, if proximity is to be considered when ranking, then the index needs to have stored positions when indexing.

3.2.6 Wild cards

Because wild card symbols are handled by expanding math tokens to include their wild card token matches, there is no adjustment needed for Lucene except that all features that contain a wild card are indexed at the same position as their corresponding feature. To accomplish this, *MathFilter* changes the position increment of all math tokens that contain a wild card to 0. Expanding symbol pairs for wild cards mean that symbol pairs that contain a wild card can now be used as a *TermQuery* to match symbol pairs that are equivalent when considering wild cards. Additionally, *idf* will automatically be based upon the total number of possible math features that match when considering the wild card symbol.

```
<topic>
 <num>NTCIR12−MathIR−15</num>
 <query>
  <formula id="f.0">
   <m:math>
    <m:semantics>
     <m:apply>
     <m:plus/>
      <m:apply>
       <m:times/>
       <m:cn type="integer">3</m:cn>
       <mws:qvar xmlns:mws="http://search.mathweb.org/ns" name="n"/>
      </m:apply>
      <m:cn type="integer">1</m:cn>
     </m:apply>
     <m:annotation−xml encoding="MathML−Presentation">
      <m:mrow xml:id="m23.1.5.pmml" xref="m23.1.5">
       <m:mrow>
        <m:mn>3</m:mn>
        <m:mo>   </m:mo>
        <mws:qvar xmlns:mws="http://search.mathweb.org/ns" name="n"/>
       </m:mrow>
       <m:mo>+</m:mo>
       <m:mn>1</m:mn>
      </m:mrow>
     </m:annotation−xml>
     <m:annotation encoding="application/x−tex">
      3\qvar@construct{n}+1
     </m:annotation>
    </m:semantics>
   </m:math>
  </formula>
  <keyword id="w.0">collatz</keyword>
  <keyword id="w.1">conjecture</keyword>
 </query>
</topic>
```

Figure 3.3: A query example for the NTCIR-12 arXiv Main Task

3.2.7 Ranking Implementation

Lucene supports various ranking functions such as BM25 and tf-idf. Additionally, Lucene supports new ranking functions by allowing one to create them through the *Similarity* interface. All ranking functions are scored using the following approach:

$$\mathrm{score(q, d) = coord(q, d) \cdot queryNorm(q)} \cdot \sum_{tinq} \left(\mathrm{tf(tind) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t, d)} \right)$$

where q is the query; t is the term; d is the document; *coord* is a score factor based on how many query terms are found in a document; *queryNorm* is a normalizing factor used to make different scores comparable across queries; *tf* is a score factor based on term frequency; *idf* is a score factor based upon inverse document frequency; *t.getBoost* is a search time boost of term t; and norm encapsulates length factors.

New ranking functions in Lucene are created by changing the above functions to fit a specific need. The norm is calculated at index time by the *Similarity.* The *Similarity* can be changed at index time or at query time. However, two different *Similarity* implementations may not calculate length factors in the same way, so it is generally advised to use the same *Similarity* as the one used while indexing. Lucene's default ranking function is BM25. Lucene's *BM25Similarity* uses the norm to encode length normalization information and is computed as $\frac{\mathrm{boost}}{\sqrt{\mathrm{fieldLength}}}$.

BM25Math was implemented by changing the frequency parameter passed to the *Similarity* class. Instead of just using term frequency, the minimum of term frequency and query frequency is used. BM25+ has two main changes from Lucene's BM25 implementation. The first is to alter the *idf* function to

$$\mathrm{log((docCount + 1.0)/(docFreq))}$$

The second main change is to alter the *score* function to

$$\mathrm{weightValue} * ((\mathrm{freq} * (k1 + 1))/(\mathrm{freq} + \mathrm{norm}) + \delta)$$

The same length factor as BM25 is used for BM25+ and BM25Dist, allowing for the ranking function to be changed at query time.

	Relevant		Partially Relevant	
Ranking Function	MAP	bpref	MAP	bpref
BM25	0.1038	0.3419	0.1466	0.4904
BM25$^+$	**0.1309**	**0.3648**	**0.1763**	**0.5189**
BM25Math	0.0582	0.2885	0.0893	0.4055

Table 3.2: A comparison of BM25 Variations on NTCIR-12 MathWiki Task. All results were evaluated by trec_eval with each query returning 1000 results

3.3 Experiment

The NTCIR-12 MathWiki Task was run against the Wikipedia Collection for each ranking function. The Wikipedia collection was indexed using only symbol pairs as features. The various ranking functions were then run against the queries, and the top 1000 results for each query were recorded. The results were evaluated using trec_eval for both partially relevant hits and relevant hits. The results in Table 3.2 show there is no gain from limiting BM25 to just the number of query occurrences for each math feature. Additionally, it shows that BM25 can be improved by using a better lower bound for *tf* normalization. Therefore, moving forward, BM25+ is used for all experiments.

Chapter 4

Math Features

4.1 Types of Features

Tangent-3 uses symbol pairs as the primary feature of a SLT and uses MSS to increase the rank formulas that are structurally similar to the query formula. This chapter examines what other features of a SLT might have useful information when using BM25+ for ranking. All new features should expand the information captured to increase the rank of a math formula that is structurally similar to the query formula. This section outlines the math features examined, and these are evaluated in Section 4.2. Figure 4.1 serves as the example to illustrate what information is captured by each new feature.

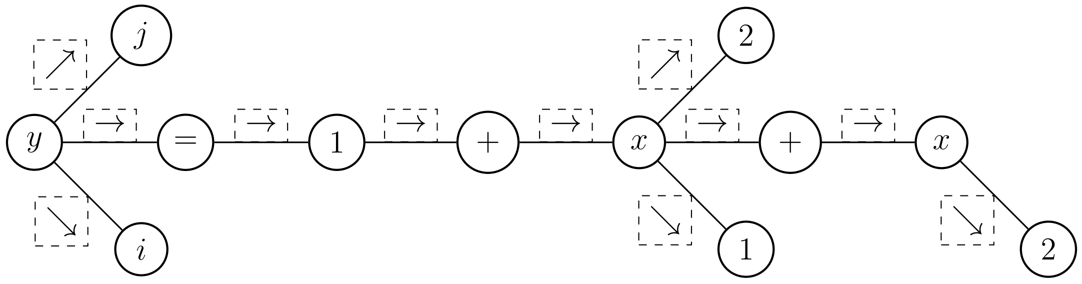

Figure 4.1: Symbol Layout Tree for $y_i^j = 1 + x_1^2 + x_2$.

S	$!0$	$\rightarrow$
$V!j$	$!0$	$\rightarrow$
$V!i$	$!0$	$\rightarrow$
$N!2$	$!0$	$\rightarrow$
$N!1$	$!0$	$\rightarrow$
$V!x$	$!0$	$\rightarrow$
$N!2$	$!0$	$\rightarrow$

(a) A list of end of line symbols for $y_i^j = 1 + x_1^2 + x_2$

S	$!0$
$V!j$	$!0$
$V!i$	$!0$
$N!2$	$!0$
$N!1$	$!0$
$N!2$	$!0$

(b) A list of terminal symbols for $y_i^j = 1 + x_1^2 + x_2$

Table 4.1: Comparing end of line symbols and terminal symbols

4.1.1 Terminal Symbols

Tangent-3 includes End-of-Line (EOL) symbols with the form $(s, !0, \rightarrow)$, where s is a symbol on a node that has no out-edge labeled $\rightarrow$, and $!0$ denotes an end-of-line. Table 4.2a depicts the EOL symbols for the SLT in Figure 4.1. The wild card expansion for EOL symbols is large and this increases retrieval times with no improvement to effectiveness [49]. Tangent-3 limits the EOL symbols to formulas with a SLT height less than 3 to improve retrieval time and effectiveness.

The new math feature proposed is called *terminal symbols*, with the form $(s, !0)$, where s is the label on a leaf in the SLT (i.e. a node labelled containing s and having no outgoing edges). Terminal symbols is applied to all math formulas regardless of size, but they are expected to be particularly helpful for small math formulas. Table 4.2b depicts a list of terminal symbols for the SLT in Figure 4.1. Terminal symbols are not expanded to include a wild card terminal symbol such as $(?v, !0)$ since such a terminal symbol would match all other terminal symbols.

As can be seen in Table 4.1 there are two main differences between terminal and EOL symbols. The first difference is terminal symbols do not include $V!x$ since it contains an outgoing edge to $N!2$, but it is included in Table 4.1 since EOL symbols include any node with no outgoing $\rightarrow$ edge. The second difference is the encoding for terminal symbols is using pairs rather than the triples used for EOL symbols.

S	$[E]$
$V!y$	$[\nearrow, \rightarrow, \searrow]$
$V!x$	$[\nearrow, \rightarrow, \searrow]$

Table 4.3: A list of compound symbols for $y_i^j = 1 + x_1^2 + x_2$

4.1.2 Compound Symbols

Symbol pairs only capture path information and for some instances may lack structural information about branching. For example, suppose one is searching for x_1^2; then just using symbols pairs would not distinguish between $y = x + x_1^2$ and $y = x_1 + x^2$. To deal with such a situation, a feature called *compound symbols* is introduced. Compound symbols have the form $(s, [e_1, e_2...e_k])$ where s is a symbol of a node with more than one outgoing edge and edges $e_1, e_2, ..., e_k$. This distinguishes the two formulae by including the feature $(V!x, [\nearrow, \rightarrow, \searrow])$ for the former and the features $(V!x, [\rightarrow, \searrow])$ and $(V!x, [\rightarrow, \searrow])$ for the latter. Table 4.3 depicts the compound symbols for the SLT in Figure 4.1. Compound symbols are expanded to include $(?v, [e_1, e_2...e_k])$.

4.1.3 Symbol pairs with Location

In traditional text information retrieval systems a longer document is ranked lower than a shorter one if they both match the same keywords at the same frequency. Now applying a similar reason to math formulas, one should rank whole matched expressions higher than partially matched subexpressions assuming both match the same keywords at the same frequency. When querying for $x_1 + x^2$, one should rank $x_1 + x^2$ higher than the formula in Figure 4.1. However, BM25 would see no difference between these two formulas since it does not penalize features that occur in the indexed formula but do not occur in the query (other than a small penalty of an increased document length).

Symbol pairs only include the path between the two symbols, but the path from the root of the SLT to the first symbol, its *location*, provides additional information that could be used to distinguish subexpressions from a whole expression. For example, the symbol pair $(V!x, N!1, \searrow)$ would have a location of $\rightarrow\rightarrow$ and would not match the query $x_1 + x^2$ when considering location. However, any expression that had $(V!x, N!1, \searrow)$ in a similar location would result in a higher rank.

One problem with including location for all math feature is that no query formula could match a subexpressions unless it was located at the start of the expression. Tangent-L

S_1	S_2	R	L
$V!y$	$V!j$	$\nearrow$	$-$
$V!y$	$V!i$	$\searrow$	$-$
$V!y$	$=$	$\rightarrow$	$-$
$=$	$N!1$	$\rightarrow$	$\rightarrow$
$N!1$	$+$	$\rightarrow$	$\rightarrow\rightarrow$
$+$	$V!x$	$\rightarrow$	$\rightarrow\rightarrow\rightarrow$
$V!x$	$N!2$	$\nearrow$	$\rightarrow\rightarrow\rightarrow\rightarrow$
$V!x$	$N!1$	$\searrow$	$\rightarrow\rightarrow\rightarrow\rightarrow$
$V!x$	$+$	$\rightarrow$	$\rightarrow\rightarrow\rightarrow\rightarrow$
$+$	$V!x$	$\rightarrow$	$\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow$
$V!x$	$N!2$	$\rightarrow$	$\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow$

Table 4.4: Expanded Symbol Pairs with Location for $y_i^j = 1 + x_1^2 + x_2$ and a window size of 1

therefore uses location as supplementary information to consider when ranking. Tangent-L expands all math features to both include location and exclude location. Symbol pairs with location are encoded with the form (s_1, s_2, r, l) where l is the path from the root to symbol node s_1 and r is the relative path between symbol nodes s_1 and s_2. Table 4.4 depicts the expanded symbol pairs for the SLT in Figure 4.1. This expands the size of the index but avoids the trade-off between capturing location information and not matching subexpressions.

4.1.4 Edge Pairs

Symbol pairs capture information about a pair of symbols and the relative path between them. An alternative approach is to capture information about a pair of edges along each path. *Edge pairs* is a math feature with the form (e_1, e_2, s) where e_1 and e_2 are edge labels on a path that share a common symbol s. Table 4.5 shows a list of edge pairs for the SLT in Figure 4.1. Edge pairs are expanded to include $(e_1, e_2, ?v)$ at indexing time.

4.1.5 Long Paths

Experiments with Tangent-3 explored the effectiveness of various window sizes and concluded that it had little effect on performance [49]. An unbounded window size, a window

E_1	E_2	s
$\rightarrow$	$\rightarrow$	$=$
$\rightarrow$	$\rightarrow$	$N!1$
$\rightarrow$	$\rightarrow$	$+$
$\rightarrow$	$\nearrow$	$V!x$
$\rightarrow$	$\searrow$	$V!x$
$\rightarrow$	$\rightarrow$	$V!x$
$\rightarrow$	$\rightarrow$	$+$
$\rightarrow$	$\nearrow$	$V!x$

Table 4.5: A list of edge pairs for $y^i = 1 + x_1^2 + x_2$

size that includes all possible symbol pairs, slightly decreased Tangent-3's recall in the presence of wild cards. The increased window size provides more information about a formula, but it may be matching frivolously with poorly matching candidate formulas. Tangent-3 uses Dice's coefficient for the first stage of matching and treats all symbol pairs as equally important, whereas the ranking function for Tangent-L uses *idf* to give less weight to commonly used terms. BM25 may benefit from including symbol pairs separated by a longer path.

A long path contains a lot of information but may be too restrictive when matching subexpressions. Additionally, any time a wild card symbol appears, matching exactly to a long path is problematic. For example, a query with $y_i^j = ?v + x_2$ should match an expression such as the one in Figure 4.1 regardless of how many terms appear between y_i^j and x_2. However, this query would include the symbol pair $(V!y, V!x, \rightarrow\rightarrow\rightarrow\rightarrow)$ but the corresponding symbol pair in Figure 4.1 is $(V!y, V!x, \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow)$ causing a mismatch. This mismatch could be avoided by storing less information about a path.

The above situation is accommodated by setting a window size within which paths must be matched exactly but beyond which only part of the path has to match. In our experiments a window size of one is used and symbol pairs outside the window size are included as "long path symbols pairs", but their paths are adjusted. Two different approaches are explored for how much of the long path to preserve. The first approach, *long paths empty*, does not include any path information and only stores the pairs of symbols. The second approach, *long paths abbreviated*, only preserves the one-edge prefix and one-edge suffix of the long path between symbol pairs. Table 4.6 provides a list of abbreviated symbol pairs for the SLT in Figure 4.1. Long paths that are empty are not expanded to include any wild card symbols. Long paths that are abbreviated are expanded to include one wild card.

S_1	S_2	R
$V!y$	$N!1$	$\rightarrow\rightarrow$
$V!y$	$+$	$\rightarrow\rightarrow$
$V!y$	$V!x$	$\rightarrow\rightarrow$
$V!y$	$N!2$	$\rightarrow\nearrow$
$V!y$	$N!1$	$\rightarrow\searrow$
$V!y$	$+$	$\rightarrow\rightarrow$
$V!y$	$V!x$	$\rightarrow\rightarrow$
$V!y$	$N!2$	$\rightarrow\searrow$
$=$	$+$	$\rightarrow\rightarrow$
$=$	$V!x$	$\rightarrow\rightarrow$
$=$	$N!2$	$\rightarrow\nearrow$
$=$	$N!1$	$\rightarrow\searrow$
$=$	$+$	$\rightarrow\rightarrow$
$=$	$V!x$	$\rightarrow\rightarrow$
$=$	$N!2$	$\rightarrow\searrow$
$N!1$	$V!x$	$\rightarrow\rightarrow$
$N!1$	$N!2$	$\rightarrow\nearrow$
$N!1$	$N!1$	$\rightarrow\searrow$
$N!1$	$+$	$\rightarrow\rightarrow$
$N!1$	$V!x$	$\rightarrow\rightarrow$
$N!1$	$N!2$	$\rightarrow\searrow$
$+$	$N!2$	$\rightarrow\nearrow$
$+$	$N!1$	$\rightarrow\searrow$
$+$	$+$	$\rightarrow\rightarrow$
$+$	$V!x$	$\rightarrow\rightarrow$
$+$	$N!2$	$\rightarrow\searrow$
$V!x$	$V!x$	$\rightarrow\rightarrow$
$V!x$	$N!2$	$\rightarrow\searrow$
$+$	$N!2$	$\rightarrow\searrow$

Table 4.6: Additional Symbol Pairs with Abbreviated Paths for $y_i^j = 1 + x_1^2 + x_2$ and a window size of 1

4.2 Experiments

All of the above features were tested using the NTCIR-11 Wikipedia benchmark. The benchmark includes 100 queries, where each query has a specified target formula that appears within the specified document. The target formula and document are the only ones considered “relevant” when judging performance.

The benchmark requires a system to rank each individual formula and return a ranked list of (documentId, formulaId) pairs. The ranked list is then evaluated at either the document level or the formula level. When evaluating at the document level, all duplicates of the documentId are removed from the ranked list, leaving the first match for that documentID only, and all results just have to match the documentId of the target formula. When evaluating at the formula level, no changes is made to the ranked list, but all results have to match both the documentID and formulaID of the target formula. Recall is measured by mean Recall@10000 and precision is measured by MRR. A MRR of 0 is assumed for any target formula that is not returned.

All the formulas were indexed in Tangent-L as their own document with information about the source document stored separately. Two indexes were created: *Index-Emp* includes all the features in Section 4.1, but longs paths are empty, and *Index-Abb* has all the features in Section 4.1, but long paths are abbreviated.

4.2.1 Feature Effectiveness

An experiment was conducted to test the average effectiveness of all the features in Section 4.1 with respect to both recall and precision. The baseline was to only use symbols pairs with a window size of 1 when querying. All possible query configurations of the additional features for querying were tested. Thus, query configurations cover the inclusion or omission of terminal symbols, the inclusion or omission of compound symbols, the inclusion or omission of symbol pairs with location, the inclusion or omission of edge pairs, and the inclusion or omission of long paths. Queries against the two indexes provide two possibilities for paths outside the window size: querying with empty paths and querying with abbreviated path information.

The NTCIR-11 Wikipedia benchmark was run for all possible configurations against both indexes. For each index, there are 32 configurations since each index contains a total of five features. For each feature 16 configurations include that feature and 16 configurations exclude that feature. The average difference in performance between a configuration

Feature	avg(Δ Recall@10000)		avg(Δ MRR)	
	Documents	Formula	Documents	Formula
Terminal Symbols	2.00	2.00	1.23	0.92
Compound Symbols	0.00	0.50	0.25	0.28
Symbol Pairs with Location	1.00	0.50	0.84	0.48
Edge Pairs	0.00	0.00	-0.04	-0.14
Long Paths Empty	0.00	0.00	0.01	-0.76

(a) Tangent-L with Index-Emp (all features indexed; long paths empty)

Feature	avg(Δ Recall@10000)		avg(Δ MRR)	
	Documents	Formula	Documents	Formula
Terminal Symbols	2.00	2.00	1.12	0.98
Compound Symbols	0.00	0.50	0.32	0.33
Symbol Pairs with Location	1.00	0.50	0.89	1.21
Edge Pairs	0.00	0.00	0.00	0.00
Long Paths Abbreviated	0.00	0.00	-0.33	-0.32

(b) Tangent-L with Index-Abb (all features indexed; long paths abbreviated)

Table 4.7: Comparison of Tangent-L feature configurations measured against NTCIR-11 Wikipedia Retrieval benchmark. avg(Δ Recall@10000) is the average percentage change for recall; avg(Δ MRR) is the average percentage change for MRR.

including a feature and a configuration excluding a feature is shown in Table 4.7 for each index.

Tangent-L with just symbols pairs has a recall of 97%: there are three formulas that are not matched. Two of the missing targeted formulas are so small that no symbol pair feature is present in the SLT, therefore they cannot be found. Querying with terminal symbols introduces a feature for those two small formulas that enables them to be recalled. The third formula is recalled only when both compound symbols and location are used for querying. Using a combination of terminal symbols, compound symbols, and location results in a recall of 100%.

Tangent-L with just symbol pairs has an average MRR of 75% at the document level and 72% at the formula level. As seen in Table 4.7, terminal symbols, compound symbols, and symbol pairs with location all on average increased the MRR when used for querying versus omitting them. Edge pairs has a small negative effect on MRR when used for querying. The match to two successive edges might not capture enough additional information to

System[*]	Recall@10000 (%) Documents	Recall@10000 (%) Formula	Mean Reciprocal Rank (%) Documents	Mean Reciprocal Rank (%) Formula
TUW Vienna	97	93	80	82
NII Japan	97	94	74	72
Tangent-3 Core (w=1, EOL)	98	98	81	77
Tangent-3 Re-rank (w=1, EOL)	98	98	82	79
Tangent-L (symbol pairs)	97	97	75	72
Tangent-L (recommended features)	100	100	80	77

[*] Previous results from Zanibbi et al. [49].

Table 4.8: Comparison of MathIR systems measured against NTCIR-11 Wikipedia Retrieval benchmark.

help when ranking. Finally, long paths, whether abbreviated or empty, do not seem to improve MRR. Long paths empty has a slight increase in MRR at the document level but the decrease in MRR at the formula level outweighs the slight increase.

4.2.2 Testing Features

The NTCIR-12 MathWikiFormula Task is used as a final test for the new math features and the effectiveness of Tangent-L for formula retrieval. An index was created with a window size of 1, compound symbols, terminal symbols and expanded symbol pairs to include location. BM25$^+$ was the ranking function used. The 40 queries were run against the index and the top 1000 results were recorded. Bpref and MAP were both calculated for the run using trec_eval. The results are shown in Table 4.9 along with the results of the other systems from NTCIR-12.

Table 4.10 gives the number of relevant, partially relevant, not relevant and not judged documents at various levels of precision. Tangent-L has a large number of documents that have no judgments on their relevancy, and, due to this, bpref is a better precision measure.

Tangent-L has a 11.5% increase in bref over Tangent-3's best run for partially relevant formulas. Tangent-L has a similar bpref for relevant documents when compared to Tangent-3's best run, with the difference being around 1%. Tangent-L has the highest bpref for partially relevant formulas. Overall, Tangent-L has comparable precision to Tangent-3, which demonstrates that the effectiveness of the math features is as effective as MSS.

	Relevant		**Partially Relevant**	
Ranking Function	MAP	bpref	MAP	bpref
Tangent-L	0.3869	0.4737	0.2742	**0.6040**
Tangent-$3_{\mathrm{f-1}}$	0.4360	0.4294	0.4204	0.5163
Tangent-$3_{\mathrm{f-2}}$	0.4485	0.4568	0.4456	0.5499
Tangent-$3_{\mathrm{f-3}}$	0.4650	0.4574	**0.4534**	0.5416
Tangent-$3_{\mathrm{f-4}}$	0.4726	0.4793	0.4533	0.5370
$\mathrm{MCAT}_{\mathrm{f-af-lr}}$	0.4189	0.4306	0.3654	0.4301
$\mathrm{MCAT}_{\mathrm{f-af-lr-u}}$	0.4557	0.4706	0.3774	0.4397
$\mathrm{MCAT}_{f-af-nw-u}$	**0.5131**	**0.5202**	0.4463	0.5356

Table 4.9: A comparing Tangent-L to various systems performance on the NTCIR-12 MathWikiFormula Task. All results were evaluated by trec_eval with each query returning 1000 results

Precision Level	Relevant	Partially Relevant	Not Relevant	Not Judged	Total
P@5	65	48	2	85	200
P@10	93	92	15	200	400
P@15	129	138	24	309	600
P@20	156	175	34	435	800
P@1000	459	920	399	37367	39145[a]
$P@\infty$	557	1282	848		

Table 4.10: Breakdown of judgements made on documents returned at various levels of precision for Tangent-L on NTCIR-12 MathWikiFormula Task.

[a]This does not add up to 40000 since query 22 only has 145 results

4.3 Conclusion

Additional features of a SLT can be helpful for formula retrieval. As shown in Table 4.8, terminal symbols, compound symbols, and location can increase MRR by 7% and recall by 3%. These three features are recommended to be used when querying for formulas and are used in the rest of the thesis when indexing and searching. Long paths seem to have a negative effect on precision regardless of how the path information is used. This re-confirms that window size has little effect on performance, similar to the conclusions drawn for Tangent-3 [49]. A window size of 1 is used for the rest of this thesis, and all symbol pairs outside the window are not considered.

The above results for the NTCIR-11 Wikipedia Retrieval benchmark show that BM25 can have comparable results to the Tangent-3 core engine and MSS re-ranking by using additional features. Tangent-L has better recall of 2% than Tangent-3 and worse MRR of 2% than Tangent-3's optimal configuration. The loss of precision in formula retrieval may be suitable if offset by an increase in precision for both math and text combined. The thesis now focuses on the performance of Tangent-L when querying with both math and text.

Chapter 5

Text and Math

This chapter explores the performance of Tangent-L when queries include both text and math formulas. It looks at applying various adjustments to ranking and its effect on performance. The first is to examine how weighting the math and text differently affects precision. The second looks at how to include proximity when ranking both math and text.

5.1 Weighting Math and Text

Chapter 4 introduced several math features to use, which results in math features contributing heavily to scoring. However, the increase in the weight assigned to math in general when ranking may cause Tangent-L to place too much importance on math and not enough on text. This section examines the effect of changing the relative weights of math and text. For this purpose, we adapt BM25 to assign weights to query terms as follows:

$$\mathrm{BM25}^{+}_{\mathrm{w}}(q_t \cup q_m, d) = \mathrm{BM25}^{+}(q_t, d) + \alpha \cdot BM25^{+}(q_m, d)$$

where q_t is the set of keywords in a query, q_m is the set of math features in that query, and α is a parameter to adjust the relative weight applied to math features.

The NTCIR-12 MathWiki Task benchmark is used. The Wikipedia collections is indexed with the optimal configuration (terminal symbols, compound symbols, symbols pairs with location) determined by the experiments reported in Section 4.2.1. The task is then run for $\alpha \in [0.01, 2.00]$ at intervals of 0.01. The MAP and bpref are calculated for all 200 runs using trec_eval.

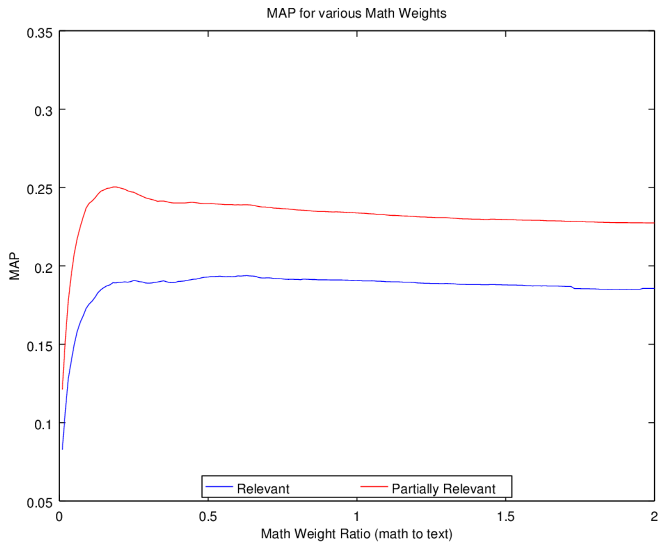

Figure 5.1: The effect of α on MAP for the NTCIR-12 MathWiki Task benchmark.

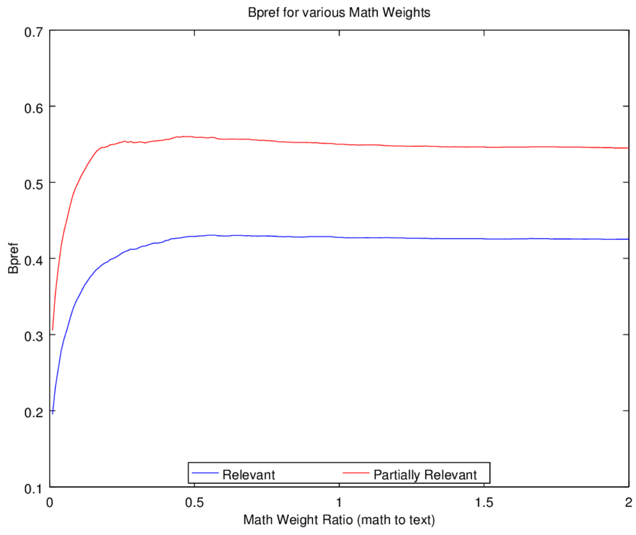

Figure 5.2: The effect of α on bpref for the NTCIR-12 MathWiki Task benchmark.

Measure	Relevancy	Measure(α = Optimal)	Measure($\alpha = 1.0$)	Gain
MAP	Relevant	0.1938	0.1907	0.0031
MAP	Partially Relevant	0.2504	0.2338	0.0166
Bpref	Relevant	0.4306	0.4276	0.0030
Bpref	Partially Relevant	0.5603	0.5501	0.0102

Table 5.1: The optimal weight gain versus using same weight

Figure 5.1 shows the effect of α on MAP for both relevant and partially relevant results on the NTCIR-12 MathWiki Task benchmark. There is a gain in MAP as α increases in the range $0.05 \leq \alpha \leq 0.60$ for relevant assessments. The maximum for relevant assessments occurs at $\alpha = 0.63$. For larger values of α there is a gradual decline. When considering partially relevant documents as well , there is an increase in MAP for $\alpha \leq 0.20$ and the maximum occurs at $\alpha = 0.19$. However, both have modest gains in precision by increasing α when $\alpha < 0.20$.

Figure 5.2 shows the effect of α on bpref for both relevant and partially relevant results against the NTCIR-12 MathWiki Task benchmark. As α increases in the range $0.05 \leq \alpha \leq 0.50$, there is a gain in bpref for relevant and partially relevant assessments. The maximum for relevant assessments occurs at $\alpha = 0.57$ and the maximum for partially relevant assessment occurs at $\alpha = 0.46$. For larger values of α, there is a slight gradual decline.

Thus, both MAP and bpref can be increased by choosing an appropriate value for α. However, as shown in Table 5.1 the increase can be quite small when compared to using $\alpha = 1.0$ (default math weight). The largest gain is for partially relevant results, which may be due to evaluators assessing documents with approximately matching keywords more often as partially relevant.

Perhaps the nature of the queries can provide insights into how to weight math versus text. Figure 5.3 illustrates the results of running $\alpha \in [0.01, 2.00]$ at intervals of 0.01, but precision is separated for each query. Any query with no keywords is in black and is a flat line since changing α does not affect these queries. The other queries are colored based upon $\frac{\#of\ math\ features}{\#\ of\ keywords}$. Some of the queries include words that are considered stop words and were not involved in the ratio calculation.

For this figure, the ratio ranges from 1 to 46 and is broken into five different colors groups: 1-9 is colored blue, 10-18 is colored cyan, 19-27 is colored green, 28-36 is color magenta, and 37-46 is color red. The smallest queries (in blue) generally have precision increase for $\alpha < 0.60$. However, the small to mid-size queries (in cyan) do not exhibit a

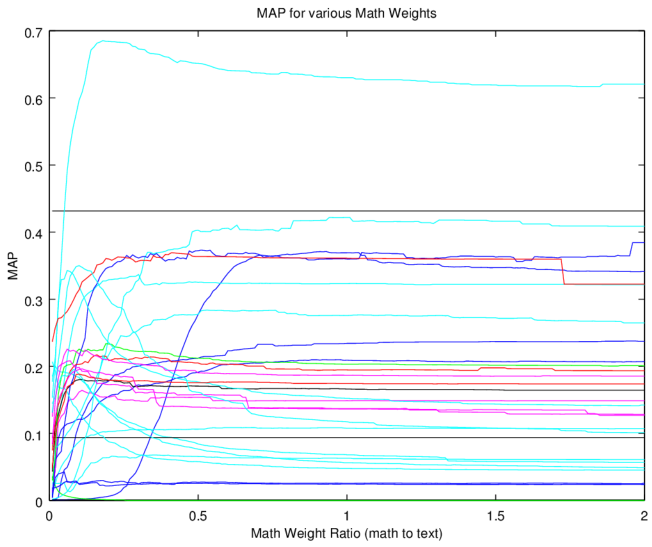

Figure 5.3: The effect of α on MAP for each query of the NTCIR-12 MathWiki Task benchmark.

noticeable pattern. There are some that have precision decrease for $\alpha > 0.20$, whereas others have precision increase for $\alpha > 0.20$. Most of the other queries have a flat curve and changing α has little effect on precision. Hence there is not a clear relationship between math to keywords ratio and α.

Nevertheless, there are some queries that peak with a small α and decline as α increases. The difference with respect to bpref for $\alpha = 1$ and $\alpha = \mathit{optimal}$ is shown in Table 5.2. Three main queries (numbers 8, 20, and 24) have the largest gain for an $\alpha \leq 0.50$. An examination of these queries demonstrates that they all have keywords that are related to their math formula. Query 8 has the keyword light and an equation with light constant c. Query 20 has the keyword polynomial and a math formula that is a polynomial. Query 24 has the keyword gamma and the math formula is the gamma function with a wild card symbol. Overall, math weight does not seem to be dependent on the number of math features, but rather some other properties.

Overall, weighting the math and the text differently can improve precision by a small factor. Using an $\alpha = 0.46$ results in a 0.70% increase for relevant results and a 1.55% increase with respect to bpref. Bpref would increase by only 3.8% if the optimal α were used for each query, which is the upper bound on possible improvements achieved from varying α. Since changing α does not result in a large increase of precision, it is recommended to

Query #	# of Math Features	# of Keywords	$\alpha_{Optimal}$	bpref	$\Delta bpref(\alpha_{Optimal}, \alpha_1)$
1	2	2	1.94	0.6962	0.0147
2	8	1	0.63	0.6039	0.0139
3	6	1	1.53	0.4027	0.0041
4	10	1	1.24	0.4406	0.0026
5	32	2	1.77	0.5381	0.0069
6	56	2	0.62	0.4036	0.0000
7	8	0	–	0.00	0.0000
8	14	1	0.07	0.6564	0.2590
9	34	2	1.16	0.4740	0.0069
10	30	0	–	0.2268	0.000
11	8	0	–	0.8408	0.0000
12	12	1	0.44	0.7485	0.0059
13	38	0	–	0.4554	0.0000
14	22	2	0.52	0.3894	0.01070
15	6	2	0.36	0.5017	0.06230
16	46	1	0.40	0.3429	0.0068
17	50	3	0.05	0.4898	0.1962
18	46	2	0.15	0.4187	0.0165
19	42	2	0.01	0.0000	0.0000
20	48	3	0.12	0.4952	0.1315
21	8	2	0.59	0.2804	0.0278
22	24	2	1.30	0.7669	0.0073
23	8	1	0.69	0.2002	0.0049
24	26	2	0.06	0.6005	0.1549
25	42	1	0.23	0.5185	0.0247
26	24	1	0.25	0.5306	0.0127
27	24	1	0.02	0.3359	0.0664
28	138	3	0.48	0.5407	0.0005
29	42	4	0.12	0.4913	0.0677
30	66	2	0.07	0.5894	0.0457

Table 5.2: The optimal weight gain versus using same weight for each query

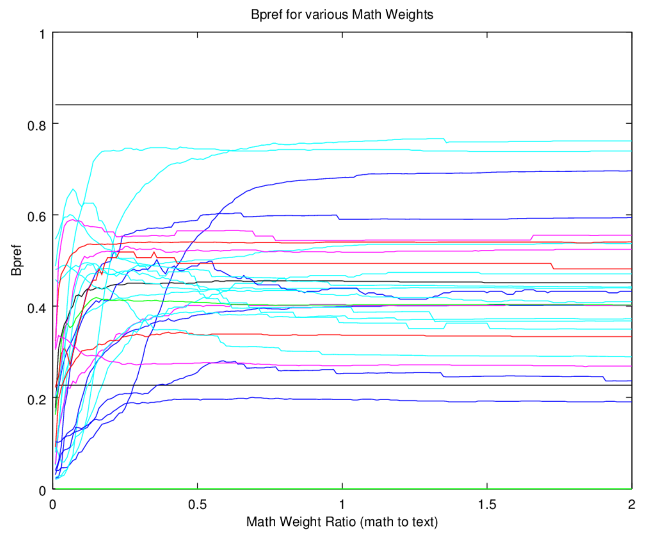

Figure 5.4: The effect of α on bpref for each query of the NTCIR-12 MathWiki Task benchmark.

just use $\alpha = 1.0$.

5.2 Proximity

Queries contain keywords and math expressions that are related to each other. As discussed above, query 8 has both light and c with it being likely that the keyword and the math expression are close to each other in relevant documents. A measure of proximity may be helpful when ranking.

One question that arises when indexing math formulas is how to assign math tokens to positions in a document. The question will impact scoring when considering proximity, and two possibilities are considered for positioning math tokens. The first approach is for each math token to be assigned to its own position. The second approach is for all math tokens extracted from a single formula to share a position. Lucene allows for tokens to share a position for words, and this feature is commonly used when indexing multiple terms that are synonyms of each other.

By default, Lucene and the *MathAnalyzer* will position each math token in its own position. To be able to support the second approach, one needs to change the *MathFilter*

	Features different position			**Features same position**		
Measurement	%Δ(R,PR)	%Δ(PR,N)	%Δ(R,N)	%Δ(R,PR)	%Δ(PR,N)	%Δ(R,N)
$\text{Span}_{\text{norm}}$	-0.3354	-0.05152	-0.3869	-0.3361	0.0917	-0.2136
$\text{MinCover}_{\text{norm}}$	0.5231	-0.2718	0.2514	0.2041	0.0410	0.2368
MinDist	0.5839	-0.7144	-0.1306	0.2379	-0.3172	-0.0039
AveDist	0.4238	-0.4148	0.0090	0.1087	-0.2278	-0.0943
MaxDist	0.4486	-0.4695	-0.0209	0.1016	-0.2991	-0.1671

Table 5.3: The percent difference between Relevant (R), Partially Relevant (PR), and Non-Relevant (NR) documents for the various measures of proximity.

to assign all math tokens arising for a formula to the same position. To accomplish this, a start and end tag are added to all formulas when converting MathML expressions by *Convert*. Now when the *MathFilter* encounters a start tag it increments the position length for the first math token, but all subsequent math tokens consume a position length of zero until a end tag is encountered. Finally, start and ends tags are removed from the token stream.

5.2.1 Importance of Proximity

As explained in Section 2.1.4, five measures of proximity can be used to quantify the proximity of each query on the judged documents. Following the experiment conducted by Tao and Zhao [45], these measures were used in experiments on the NTCIR-12 MathWiki task, choosing the variants $\text{Span}_{\text{norm}}$ and $\text{MinCover}_{\text{norm}}$, rather than Span and MinCover. All thirty queries were used, and the average of each measurement was taken for relevant, partially relevant, and non-relevant documents. Table 5.3 shows the percent difference between each type of relevancy judgment.

Relevant documents have query terms that are closer together than partially relevant documents, but relevant documents have query terms farther apart than non-relevant documents. Also, partially relevant documents have query terms that are farther apart than non-relevant documents. This implies that proximity may not be a benefit to use when ranking. The one exception is when proximity is measured by $\text{MinCover}_{\text{norm}}$, which is the only measurement where relevant and partially relevant have closer query terms than non-relevant documents.

The average values and standard deviations for each measurement are shown in Table 5.4. No measurement can be less than zero and there are large standard deviations

	Features different position			Features same position		
Measurement	μ	*median*	σ	μ	*median*	σ
$\text{Span}_{\text{norm}}$	53	19	126	22	7	59
$\text{MinCover}_{\text{norm}}$	98	29	206	31	8	83
MinDist	22	0	203	8	0	80
AveDist	1024	525	1460	292	155	409
MaxDist	2974	1530	4206	915	489	1247

Table 5.4: The percent difference between Relevant (R), Partially Relevant (PR), and Non-Relevant (NR) documents for the various measures of proximity.

	Formula features different position			Formula features same position		
Measurement	%Δ(R,PR)	%Δ(PR,N)	%Δ(R,N)	%Δ(R,PR)	%Δ(PR,N)	%Δ(R,N)
$\text{Span}_{\text{norm}}$	-0.37	0.46	0.26	-0.43	0.57	0.39
$\text{MinCover}_{\text{norm}}$	0.15	0.30	0.40	-0.11	0.53	0.48
MinDist	-0.34	0.54	0.38	-0.36	0.72	0.61
AveDist	0.11	0.06	0.16	-0.09	0.29	0.23
MaxDist	0.11	-0.03	0.08	-0.11	0.18	0.09

Table 5.5: The percent difference between Relevant (R), Partially Relevant (PR), and Non-Relevant (NR) documents for the various measures of proximity normalized by document length.

for each of the measurements. Additionally, the median is significantly smaller than the mean, which may indicate that the mean is skewed by larger values on the right tail. Re-running the same experiment with each measurement normalized by document length gives all measurements between 0 and 1. The percent difference between each type of judgment for various measurements are shown in Table 5.5.

The proximity of relevant and partially relevant documents becomes smaller than not-relevant documents, as seen in Table 5.5, after normalizing measurements by document length. Additionally, having math tokens of the same formula in the same position increases the difference in proximity between relevant and not relevant documents. The results indicate that proximity may help improve ranking after all, if each math token of the same formula is stored at the same location. Both $\text{MinCover}_{\text{norm}}$ and MinDist appear to be the best proximity measurements to use.

		Relevant		Partially Relevant	
Ranking Function		MAP	bpref	MAP	bpref
BM25		0.1907	0.4276	0.2338	0.5501
BM25Dist	MinDist	0.1908	0.4276	0.2340	0.5501
BM25Dist	MinCover$_{\text{norm}}$	**0.1910**	0.4276	**0.2342**	0.5501
BM25TP		0.1909	**0.4278**	0.2339	0.5501

Table 5.6: A comparison of BM25 Variations on NTCIR-12 MathWiki Task. All results were evaluated by trec_eval with each query returning 1000 results

5.2.2 Ranking Function with Proximity

The NTCIR-12 MathWiki Task is run to test the value of using proximity on precision using three different ranking functions: BM25, BM25Dist and BM25TP. Table 5.6 shows small differences between the various ranking functions. However, using proximity always results in a small increase and never results in decreasing precision. The overhead to store the positional information that is needed to rank with proximity outweighs the small gain in precision. Although proximity may be useful when ranking, BM25Dist and BM25TP were unable to improve precision significantly for this task.

5.3 NTCIR-12 arXiv

The NTCIR-12 arXiv Main Task is the task for evaluating the final design decisions for Tangent-L on an index with a window size of 1, compound symbols, terminal symbols, and expanded symbol pairs to include location. BM25$^+$ is the ranking function, and proximity is not considered. The 29 queries is run against the index, and the top 1000 results are recorded. Bpref and MAP are both calculated with trec_eval. The results are shown in Table 5.7, along with the results of the other systems from NTCIR-12.

Table 5.8 gives the number of relevant, partially relevant, not relevant, and not judged documents at various levels of precision. Tangent-L has a large number of documents that have no judgments on their relevancy, making bpref a more reliable precision measure for this experiment.

Tangent-L has a 55% increase in bpref over Tangent-3's best run for relevant documents. Tangent-L has similar bpref for partially relevant documents when compared to Tangent-

	Relevant		Partially Relevant	
Ranking Function	MAP	bpref	MAP	bpref
MCAT$_{af-lr}$	0.2072	0.2232	0.2629	0.3669
MCAT$_{af-lr-u}$	0.2080	0.2281	0.2633	0.3699
MCAT$_{af-nw-u}$	**0.2659**	**0.2906**	**0.2866**	**0.4025**
MCAT$_{nd-lr-u}$	0.2001	0.2157	0.2550	0.3629
MIaS$_{cm-r-10}$	0.0976	0.1150	0.1563	0.2622
MIaS$_{pm-r-10}$	0.0816	0.0927	0.1281	0.2347
MIaS$_{pcm-l-10}$	0.0868	0.0986	0.1435	0.2276
MIaS$_{pm-l-10}$	0.0581	0.0640	0.1147	0.2070
SMSG5$_{tfidf}$	0.0550	0.0695	0.1357	0.2434
Tangent-3$_1$	0.1837	0.1768	0.2088	0.2967
Tangent-3$_2$	0.1844	0.1811	0.2091	0.2980
Tangent-3$_3$	0.1376	0.1433	0.2489	0.3500
Tangent-3$_4$	0.1399	0.1447	0.2509	0.3522
WikiMir	0.1626	0.1717	0.1856	0.2474
Tangent-L	0.1926	0.2813	0.1469	0.3438

Table 5.7: A comparison of various systems on the NTCIR-12 arXiv Main Task. All results evaluated by trec_eval, with each query returning 1000 results

3's best run, with the loss being just over 2%. Overall, Tangent-L has better bpref than Tangent-3, and demonstrates the effectiveness of the approach. In fact, Tangent-L's bpref for relevant documents puts it within 2% of MCAT's best result.

The increase in precision does not come with an excessive increase in memory used. Table 5.9 shows various index sizes for both Tangent-3 and Tangent-L for the Wikipedia benchmarks. The lower bound and upper bound of both systems are quite comparable. However, the optimal configuration index, (EOL,w=1) for Tangent-3, is 40% smaller than Tangent-L's recommended features index. The Tangent-L index created for the NTCIR-12 arXiv Main Task is 6.2 GB, which is 25% smaller than Tangent-3's reported math index size of 8.3 GB [12] for a window size of 2, to which the size of its text index must be added.

Additionally, Tangent-L is more efficient in retrieval time. The NTCIR-12 Main arXiv Task are run 10 times for its 29 queries on a Ubuntu Linux 14.04.5 LTS machine with AMD FX-8370E Eight-Core Processor (3.3GHz) and 16 GB RAM. The average time for each query requires 0.69 seconds, the fastest query requiring 0.02 seconds on average and

Precision Level	Relevant	Partially Relevant	Not Relevant	Not Judged	Total
P@5	38	13	21	73	145
P@10	60	33	36	161	290
P@15	71	53	46	265	435
P@20	80	67	65	368	580
P@1000	258	267	323	28152	29000
P@∞	413	876	2962		

Table 5.8: Breakdown of judgements made on documents returned at various levels of precision for Tangent-L on NTCIR-12 arXiv Main Task across all 29 queries.

Configuration*	Index Size (MB)
Tangent-3 (No-EOL,w=1)	64.2
Tangent-3 (EOL,w=1)	73.7
Tangent-3 (EOL,w=unbounded)	503.1
Tangent-L (symbol pairs)	32.3
Tangent-L (recommended features)	124.2
Tangent-L (all features; long paths empty)	261.3
Tangent-L (all features; long paths abbreviated)	575.5

* Tangent-3 sizes reported by Zanibbi et al. [49].

Table 5.9: Comparing Tangent-3 and Tangent-L index sizes for NTCIR-11 Wikipedia collection.

the slowest query requiring 2.56 seconds. This outperforms the reported times for Tangent-3 (mean=27.54, minimum=2.77, maximum=178.51) [12]. The two systems' times are measured with incomparable machine architectures, but the difference in magnitude shows it is unlikely that Tangent-L is slower than Tangent-3.

Chapter 6

Conclusion

This work presents a new math information retrieval system called Tangent-L, which encodes math features of a symbol layout tree as tokens that are indexed in the same manner as traditional words. Tangent-L uses BM25 as its ranking function, which is used to rank both math and text.

Tangent-L increases bpref on the NTCIR-12 MathWikiFormula task by 9.8% for partially relevant formulas compared to Tangent-3. The results indicate that Tangent-L performs approximately as well as Tangent-3 for formula retrieval. This implies that standard information retrieval techniques can have similar performance to customized techniques on formula retrieval if appropriate math features are used.

Tangent-L increases bpref on the NTCIR-12 arXiv Main task by 55% for relevant documents. The results indicated that using a single ranking function for math formulas and text can outperform using separate ranking functions and combining results.

6.1 Contributions

Chapter 3 describes how to encode math features of a symbol layout tree and index them using a text information retrieval system. Additionally, it demonstrates a way to index simple wild cards so that they can be retrieved without having to change the underlying search execution or scoring system.

Chapter 4 introduces new math features of a symbol layout tree. The new math features increase the mean reciprocal rank by 7%. Ranking math features with BM25 has comparable results to Maximum Subtree Similarity.

The results from Chapter 5 demonstrate that using MinCover$_{\text{norm}}$ as a measure of proximity results in relevant documents being closer than non-relevant documents. In spite of this potential for improving retrieval performance, the ranking functions explored did not improve precision by any significant amount.

6.2 Future Work

The thesis does make improvements to the previous system of Tangent-3, but there still exist opportunities for improvement. The first area to examine is alternative text ranking functions and their performance on formula retrieval. Lucene supports a several other ranking functions, including:

- DFISimilarity [23]
- DFRSimilarity [3]
- IBSimilarity [10]
- LMJelinekMercerSimilarity [50]
- LMDirichletSimilarity [50]
- Set of Axiomatic Similarities [13]

A second area to examine is how to better incorporate proximity into the ranking function. Markov random fields have been used to model term dependency and show improvements especially for larger collections [30]. BM25P uses Survival Analysis over an exponential distribution and has results where the term proximity enhances retrieval effectiveness [20]. These alternatives should be examined to determine whether proximity can improve ranking performance.

Section 5.1 examines weighting the math and text differently when ranking. There was no relationship between the number of math features and the weight applied to math. However, the queries with largest gain all had keywords that were closely related to the math formulas. Math expressions likely appear near common words. For example, c most likely appears near the keyword "light". A math information retrieval system could improve precision by including keywords that commonly appear near a math formula. The MCAT system extracted a set of terms that denote a math expression with a support vector machine model [24], and a similar approach was recently suggested by Krstovski and Blei [26]. Adopting such a strategy may help Tangent-L in weighting math and keywords.

The long term work for math information retrieval requires creating more queries and evaluating relevant documents. There is a general trend in information retrieval to incorporating machine learning, such as convolutional neural networks [39]. The advantage of such an approach is that it eliminates the need to create features but instead relies on have the model learn the features to use. This thesis introduces math features to use, but there may exist better features. However, convolutional neural networks usually use Word2Vec [31] and require a large amount of training data. The creation of more benchmarks that include extensive training data will further the field of math information retrieval.

References

[1] Akiko Aizawa, Michael Kohlhase, and Iadh Ounis. NTCIR-10 math pilot task overview. In *NTCIR-10*, pages 654–661, 2013.

[2] Akiko Aizawa, Michael Kohlhase, Iadh Ounis, and Moritz Schubotz. NTCIR-11 math-2 task overview. In *NTCIR-11*, pages 88–98, 2014.

[3] Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *TOIS*, 20(4):357–389, 2002.

[4] Javed A. Aslam and Mark Montague. Models for metasearch. In *SIGIR 2001*, pages 276–284. ACM, 2001.

[5] Andrzej Białecki, Robert Muir, and Grant Ingersoll. Apache Lucene 4. In *SIGIR 2012*, pages 17–24, 2012.

[6] Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. In *SIGIR 2004*, pages 25–32. ACM, 2004.

[7] David Carlisle, Patrick Ion, and Robert Miner. Mathematical markup language (MathML) version 3.0 2nd edition. W3C recommendation, W3C, April 2014. http://www.w3.org/TR/2014/REC-MathML3-20140410/.

[8] Davide Cervone. MathJax: a platform for mathematics on the web. *Notices of the AMS*, 59(2):312–316, 2012.

[9] Charles L. A. Clarke and Gordon V. Cormack. Shortest-substring retrieval and ranking. *TOIS*, 18(1):44–78, January 2000.

[10] Stéphane Clinchant and Eric Gaussier. Information-based models for ad hoc IR. In *SIGIR 2010*, pages 234–241. ACM, 2010.

[11] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 283. Addison-Wesley Reading, 2010.

[12] Kenny Davila, Richard Zanibbi, Andrew Kane, and Frank Wm Tompa. Tangent-3 at the NTCIR-12 MathIR task. In *NTCIR-12*, pages 338–345, 2016.

[13] Hui Fang and ChengXiang Zhai. Semantic term matching in axiomatic approaches to information retrieval. In *SIGIR 2006*, pages 115–122. ACM, 2006.

[14] David Formánek, Martin Líška, Michal Ružička, and Petr Sojka. Normalization of digital mathematics library content. In *CICM 2012 Work in Progress*, pages 91–103, 2012.

[15] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003.

[16] Liangcai Gao, Ke Yuan, Yuehan Wang, Zhuoren Jiang, and Zhi Tang. The math retrieval system of ICST for NTCIR-12 MathIR task. In *NTCIR-12*, pages 318–322, 2016.

[17] Deyan Ginev and Bruce R. Miller. LaTeXML 2012 — a year of LaTeXML. In *CICM 2013*, pages 335–338. Springer Berlin Heidelberg, 2013.

[18] David Hawking and Paul Thistlewaite. Proximity operators-so near and yet so far. In *TREC-4*, pages 131–143, 1995.

[19] Michiel Hazewinkel. Mathematical knowledge management is needed. *arXiv preprint cs/0410055*, Nov 2003. Keynote Lecture at the MKM meeting.

[20] Ben He, Jimmy Xiangji Huang, and Xiaofeng Zhou. Modeling term proximity for probabilistic information retrieval models. *Information Sciences*, 181(14):3017 – 3031, 2011.

[21] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[22] Shahab Kamali and Frank Wm. Tompa. Structural similarity search for mathematics retrieval. In *CICM 2013*, pages 246–262. Springer Berlin Heidelberg, 2013.

[23] İlker Kocabaş, Bekir Taner Dinçer, and Bahar Karaoğlan. A nonparametric term weighting method for information retrieval based on measuring the divergence from independence. *Information Retrieval*, 17(2):153–176, Apr 2014.

[24] Giovanni Yoko Kristianto, Goran Topic, and Akiko Aizawa. MCAT math retrieval system for NTCIR-12 MathIR task. In *NTCIR-12*, pages 323–330, 2016.

[25] Giovanni Yoko Kristianto, Goran Topic, Florence Ho, and Akiko Aizawa. The MCAT math retrieval system for NTCIR-11 math track. In *NTCIR-11*, pages 120–126, 2014.

[26] Kriste Krstovski and David M Blei. Equation embeddings. *arXiv preprint arXiv:1803.09123*, 2018.

[27] Xiaoyan Lin, Liangcai Gao, Xuan Hu, Zhi Tang, Yingnan Xiao, and Xiaozhong Liu. A mathematics retrieval system for formulae in layout presentations. In *SIGIR 2014*, pages 697–706. ACM, 2014.

[28] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317, 1957.

[29] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *CIKM 2011*, pages 7–16. ACM, 2011.

[30] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *SIGIR 2005*, pages 472–479. ACM, 2005.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[32] Minh-Quoc Nghiem, Giovanni Yoko Kristianto, Goran Topić, and Akiko Aizawa. Which one is better: Presentation-based or content-based math search? In *CICM 2014*, pages 200–212. Springer International Publishing, 2014.

[33] Yves Rasolofo and Jacques Savoy. Term proximity scoring for keyword-based retrieval systems. In *ECIR 2003*, pages 207–218. Springer Berlin Heidelberg, 2003.

[34] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[35] Michal Ruzicka, Petr Sojka, and Martin Líska. Math indexer and searcher under the hood: Fine-tuning query expansion and unification strategies. In *NTCIR-12*, pages 331–337, 2016.

[36] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[37] Moritz Schubotz, Norman Meuschke, Marcus Leich, and Bela Gipp. Exploring the one-brain barrier: A manual contribution to the NTCIR-12 MathIR task. In *NTCIR-12*, pages 309–317, 2016.

[38] Moritz Schubotz, Abdou Youssef, Volker Markl, and Howard S. Cohl. Challenges of mathematical information retrieval in the NTCIR-11 math wikipedia task. In *SIGIR 2015*, pages 951–954. ACM, 2015.

[39] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR 2015*, pages 373–382. ACM, 2015.

[40] Petr Sojka and Martin Líška. The art of mathematics retrieval. In *DocEng 2011*, pages 57–60. ACM, 2011.

[41] David Stalnaker and Richard Zanibbi. Math expression retrieval using an inverted index over symbol pairs. In *DRR*, pages 940207–1:12, 2015.

[42] Heinrich Stamerjohanns, Deyan Ginev, Catalin David, Dimitar Misev, Vladimir Zamdzhiev, and Michael Kohlhase. MathML-aware article conversion from LaTeX. *Towards a Digital Mathematics Library*, pages 109–120, 2009.

[43] Heinrich Stamerjohanns and Michael Kohlhase. Transforming the arxiv to XML. In *CICM 2008*, pages 574–582. Springer Berlin Heidelberg, 2008.

[44] Heinrich Stamerjohanns, Michael Kohlhase, Deyan Ginev, Catalin David, and Bruce Miller. Transforming large collections of scientific publications to XML. *Mathematics in Computer Science*, 3(3):299–307, May 2010.

[45] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *SIGIR 2007*, pages 295–302. ACM, 2007.

[46] Abhinav Thanda, Ankit Agarwal, Kushal Singla, Aditya Prakash, and Abhishek Gupta. A document retrieval system for math queries. In *NTCIR-12*, pages 346–353, 2016.

[47] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to BM25 and language models examined. In *ADCS 2014*, pages 58:58–58:65. ACM, 2014.

[48] Richard Zanibbi, Akiko Aizawa, Michael Kohlhase, Iadh Ounis, Goran Topic, and Kenny Davila. NTCIR-12 MathIR task overview. In *NTCIR-12*, pages 299–308, 2016.

[49] Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Wm. Tompa. Multi-stage math formula search: Using appearance-based similarity metrics at scale. In *SIGIR 2016*, pages 145–154. ACM, 2016.

[50] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *TOIS*, 22(2):179–214, April 2004.

[51] Qun Zhang and Abdou Youssef. An approach to math-similarity search. In *CICM 2014*, pages 404–418. Springer International Publishing, 2014.

APPENDICES

Appendix A

Experiment Results

A.1 Precision on the NTCIR-12 arXiv Main Task for each Query

	Relevant			**Partially Relevant**		
Query #	# of Relevant Documents	MAP	bpref	# of Partially Relevant Documents	MAP	bpref
1	18	0.0262	0.0988	40	0.0637	0.1731
2	2	0.0000	0.0000	24	0.0000	0.0000
3	24	0.4857	0.4931	69	0.2267	0.3461
4	6	0.3659	0.6111	52	0.1325	0.2445
5	17	0.0269	0.2249	76	0.0369	0.3265
6	2	0.0000	0.0000	57	0.0540	0.3509
7	0	0.0000	0.0000	30	0.0531	0.2644
8	10	0.5876	0.6600	27	0.2606	0.2949
9	3	0.0185	0.3333	90	0.0060	0.1438
10	10	0.0000	0.0000	34	0.0000	0.0000
11	5	0.7600	0.7600	8	0.4750	0.4844
12	0	0.0000	0.0000	31	0.0276	0.6119
13	0	0.0000	0.0000	10	0.0000	0.0000
14	9	0.1205	0.1481	63	0.4088	0.6732
15	10	0.7426	0.6400	31	0.6305	0.6753
16	3	0.1643	0.4444	29	0.0853	0.6516
17	0	0.0000	0.0000	12	0.0031	0.0000
18	4	0.0000	0.0000	16	0.0003	0.1016
19	0	0.0000	0.0000	0	0.0000	0.0000
20	52	0.0017	0.0318	115	0.0008	0.0207
21	34	0.0689	0.4092	47	0.0569	0.3291
22	4	0.0000	0.0000	9	0.0038	0.0000
23	29	0.3938	0.8205	51	0.2605	0.6767
24	2	0.0029	0.0000	21	0.2052	0.5873
25	4	0.5000	0.5000	87	0.0262	0.0678
26	23	0.1847	0.3346	80	0.1514	0.5469
27	6	0.8417	0.8333	22	0.5210	0.7645
28	134	0.2180	0.8138	141	0.2294	0.8865
29	2	0.0747	0.0000	17	0.3399	0.7474